OPSWAT.



MetaDefender™ Core

MetaDefender Core
v5.13.1

Table of Contents

Installation 1

Recommended System Configuration 1
Differences Between MetaDefender Core v4 and v5 10
Installation 16
  Standalone mode 17
  Shared Database Mode 23
  Deployment automation support 31
  CIS Level 1 Guidelines 41
Wizard Setup 43
License Activation 49
  Scripted license management 54
File Processing 58
Product Upgrading 59
Using external Load Balancer 63
SSL connection for PostgreSQL communication 68
Performance and Load Estimation 73
  Standalone DB - Windows 74
  Standalone DB - Linux 90
  Shared DB - Windows 107
  Shared DB - Linux 131
MetaDefender Core Documentation 155

Configuration 156

Recommended System Configuration 156
Differences Between MetaDefender Core v4 and v5 165
Installation 171
  Standalone mode 172
  Shared Database Mode 178
  Deployment automation support 186

CIS Level 1 Guidelines 196
Wizard Setup 198
License Activation 204
Scripted license management 209
File Processing 213
Product Upgrading 214
Using external Load Balancer 218
SSL connection for PostgreSQL communication 223
Performance and Load Estimation 228
Standalone DB - Windows 229
Standalone DB - Linux 245
Shared DB - Windows 262
Shared DB - Linux 286
MetaDefender Core Documentation 310

Operating 311

Recommended System Configuration 311
Differences Between MetaDefender Core v4 and v5 320
Installation 326
Standalone mode 327
Shared Database Mode 333
Deployment automation support 341
CIS Level 1 Guidelines 351
Wizard Setup 353
License Activation 359
Scripted license management 364
File Processing 368
Product Upgrading 369
Using external Load Balancer 373
SSL connection for PostgreSQL communication 378
Performance and Load Estimation 383
Standalone DB - Windows 384

Standalone DB - Linux 400
Shared DB - Windows 417
Shared DB - Linux 441
MetaDefender Core Documentation 465

Integration 466

Recommended System Configuration 466
Differences Between MetaDefender Core v4 and v5 475
Installation 481
Standalone mode 482
Shared Database Mode 488
Deployment automation support 496
CIS Level 1 Guidelines 506
Wizard Setup 508
License Activation 514
Scripted license management 519
File Processing 523
Product Upgrading 524
Using external Load Balancer 528
SSL connection for PostgreSQL communication 533
Performance and Load Estimation 538
Standalone DB - Windows 539
Standalone DB - Linux 555
Shared DB - Windows 572
Shared DB - Linux 596
MetaDefender Core Documentation 620

Central Hub Deployment 621

Recommended System Configuration 621
Differences Between MetaDefender Core v4 and v5 630
Installation 636
Standalone mode 637

Shared Database Mode 643
Deployment automation support 651
CIS Level 1 Guidelines 661
Wizard Setup 663
License Activation 669
Scripted license management 674
File Processing 678
Product Upgrading 679
Using external Load Balancer 683
SSL connection for PostgreSQL communication 688
Performance and Load Estimation 693
Standalone DB - Windows 694
Standalone DB - Linux 710
Shared DB - Windows 727
Shared DB - Linux 751
MetaDefender Core Documentation 775

Cloud Deployment 776

Recommended System Configuration 776
Differences Between MetaDefender Core v4 and v5 785
Installation 791
Standalone mode 792
Shared Database Mode 798
Deployment automation support 806
CIS Level 1 Guidelines 816
Wizard Setup 818
License Activation 824
Scripted license management 829
File Processing 833
Product Upgrading 834
Using external Load Balancer 838
SSL connection for PostgreSQL communication 843

Performance and Load Estimation 848
Standalone DB - Windows 849
Standalone DB - Linux 865
Shared DB - Windows 882
Shared DB - Linux 906
MetaDefender Core Documentation 930

Container Deployment 931

Recommended System Configuration 931
Differences Between MetaDefender Core v4 and v5 940
Installation 946
Standalone mode 947
Shared Database Mode 953
Deployment automation support 961
CIS Level 1 Guidelines 971
Wizard Setup 973
License Activation 979
Scripted license management 984
File Processing 988
Product Upgrading 989
Using external Load Balancer 993
SSL connection for PostgreSQL communication 998
Performance and Load Estimation 1003
Standalone DB - Windows 1004
Standalone DB - Linux 1020
Shared DB - Windows 1037
Shared DB - Linux 1061
MetaDefender Core Documentation 1085

Metascan Engines 1086

Recommended System Configuration 1086
Differences Between MetaDefender Core v4 and v5 1095

Installation 1101
Standalone mode 1102
Shared Database Mode 1108
Deployment automation support 1116
CIS Level 1 Guidelines 1126
Wizard Setup 1128
License Activation 1134
Scripted license management 1139
File Processing 1143
Product Upgrading 1144
Using external Load Balancer 1148
SSL connection for PostgreSQL communication 1153
Performance and Load Estimation 1158
Standalone DB - Windows 1159
Standalone DB - Linux 1175
Shared DB - Windows 1192
Shared DB - Linux 1216
MetaDefender Core Documentation 1240

Deep CDR 1241

Recommended System Configuration 1241
Differences Between MetaDefender Core v4 and v5 1250
Installation 1256
Standalone mode 1257
Shared Database Mode 1263
Deployment automation support 1271
CIS Level 1 Guidelines 1281
Wizard Setup 1283
License Activation 1289
Scripted license management 1294
File Processing 1298
Product Upgrading 1299

Using external Load Balancer 1303
SSL connection for PostgreSQL communication 1308
Performance and Load Estimation 1313
Standalone DB - Windows 1314
Standalone DB - Linux 1330
Shared DB - Windows 1347
Shared DB - Linux 1371
MetaDefender Core Documentation 1395

Proactive DLP 1396

Recommended System Configuration 1396
Differences Between MetaDefender Core v4 and v5 1405
Installation 1411
Standalone mode 1412
Shared Database Mode 1418
Deployment automation support 1426
CIS Level 1 Guidelines 1436
Wizard Setup 1438
License Activation 1444
Scripted license management 1449
File Processing 1453
Product Upgrading 1454
Using external Load Balancer 1458
SSL connection for PostgreSQL communication 1463
Performance and Load Estimation 1468
Standalone DB - Windows 1469
Standalone DB - Linux 1485
Shared DB - Windows 1502
Shared DB - Linux 1526
MetaDefender Core Documentation 1550

Sandbox 1551

Recommended System Configuration 1551
Differences Between MetaDefender Core v4 and v5 1560
Installation 1566
Standalone mode 1567
Shared Database Mode 1573
Deployment automation support 1581
CIS Level 1 Guidelines 1591
Wizard Setup 1593
License Activation 1599
Scripted license management 1604
File Processing 1608
Product Upgrading 1609
Using external Load Balancer 1613
SSL connection for PostgreSQL communication 1618
Performance and Load Estimation 1623
Standalone DB - Windows 1624
Standalone DB - Linux 1640
Shared DB - Windows 1657
Shared DB - Linux 1681
MetaDefender Core Documentation 1705

Reputation Engine 1706

Recommended System Configuration 1706
Differences Between MetaDefender Core v4 and v5 1715
Installation 1721
Standalone mode 1722
Shared Database Mode 1728
Deployment automation support 1736
CIS Level 1 Guidelines 1746
Wizard Setup 1748
License Activation 1754
Scripted license management 1759

File Processing 1763
Product Upgrading 1764
Using external Load Balancer 1768
SSL connection for PostgreSQL communication 1773
Performance and Load Estimation 1778
Standalone DB - Windows 1779
Standalone DB - Linux 1795
Shared DB - Windows 1812
Shared DB - Linux 1836
MetaDefender Core Documentation 1860

Software Bill of Materials 1861

Recommended System Configuration 1861
Differences Between MetaDefender Core v4 and v5 1870
Installation 1876
Standalone mode 1877
Shared Database Mode 1883
Deployment automation support 1891
CIS Level 1 Guidelines 1901
Wizard Setup 1903
License Activation 1909
Scripted license management 1914
File Processing 1918
Product Upgrading 1919
Using external Load Balancer 1923
SSL connection for PostgreSQL communication 1928
Performance and Load Estimation 1933
Standalone DB - Windows 1934
Standalone DB - Linux 1950
Shared DB - Windows 1967
Shared DB - Linux 1991
MetaDefender Core Documentation 2015

Country of Origin 2016

Recommended System Configuration 2016
Differences Between MetaDefender Core v4 and v5 2025
Installation 2031
  Standalone mode 2032
  Shared Database Mode 2038
  Deployment automation support 2046
  CIS Level 1 Guidelines 2056
Wizard Setup 2058
License Activation 2064
  Scripted license management 2069
File Processing 2073
Product Upgrading 2074
Using external Load Balancer 2078
SSL connection for PostgreSQL communication 2083
Performance and Load Estimation 2088
  Standalone DB - Windows 2089
  Standalone DB - Linux 2105
  Shared DB - Windows 2122
  Shared DB - Linux 2146
MetaDefender Core Documentation 2170

InSights Threat Intelligence 2171

Recommended System Configuration 2171
Differences Between MetaDefender Core v4 and v5 2180
Installation 2186
  Standalone mode 2187
  Shared Database Mode 2193
  Deployment automation support 2201
  CIS Level 1 Guidelines 2211
Wizard Setup 2213

License Activation 2219
Scripted license management 2224
File Processing 2228
Product Upgrading 2229
Using external Load Balancer 2233
SSL connection for PostgreSQL communication 2238
Performance and Load Estimation 2243
Standalone DB - Windows 2244
Standalone DB - Linux 2260
Shared DB - Windows 2277
Shared DB - Linux 2301
MetaDefender Core Documentation 2325

Knowledge Base 2326

Recommended System Configuration 2326
Differences Between MetaDefender Core v4 and v5 2335
Installation 2341
Standalone mode 2342
Shared Database Mode 2348
Deployment automation support 2356
CIS Level 1 Guidelines 2366
Wizard Setup 2368
License Activation 2374
Scripted license management 2379
File Processing 2383
Product Upgrading 2384
Using external Load Balancer 2388
SSL connection for PostgreSQL communication 2393
Performance and Load Estimation 2398
Standalone DB - Windows 2399
Standalone DB - Linux 2415
Shared DB - Windows 2432

Shared DB - Linux 2456
MetaDefender Core Documentation 2480

Utilities engines 2481

Recommended System Configuration 2481
Differences Between MetaDefender Core v4 and v5 2490
Installation 2496
Standalone mode 2497
Shared Database Mode 2503
Deployment automation support 2511
CIS Level 1 Guidelines 2521
Wizard Setup 2523
License Activation 2529
Scripted license management 2534
File Processing 2538
Product Upgrading 2539
Using external Load Balancer 2543
SSL connection for PostgreSQL communication 2548
Performance and Load Estimation 2553
Standalone DB - Windows 2554
Standalone DB - Linux 2570
Shared DB - Windows 2587
Shared DB - Linux 2611
MetaDefender Core Documentation 2635

Release Notes 2636

Recommended System Configuration 2636
Differences Between MetaDefender Core v4 and v5 2645
Installation 2651
Standalone mode 2652
Shared Database Mode 2658
Deployment automation support 2666

CIS Level 1 Guidelines 2676
Wizard Setup 2678
License Activation 2684
Scripted license management 2689
File Processing 2693
Product Upgrading 2694
Using external Load Balancer 2698
SSL connection for PostgreSQL communication 2703
Performance and Load Estimation 2708
Standalone DB - Windows 2709
Standalone DB - Linux 2725
Shared DB - Windows 2742
Shared DB - Linux 2766
MetaDefender Core Documentation 2790

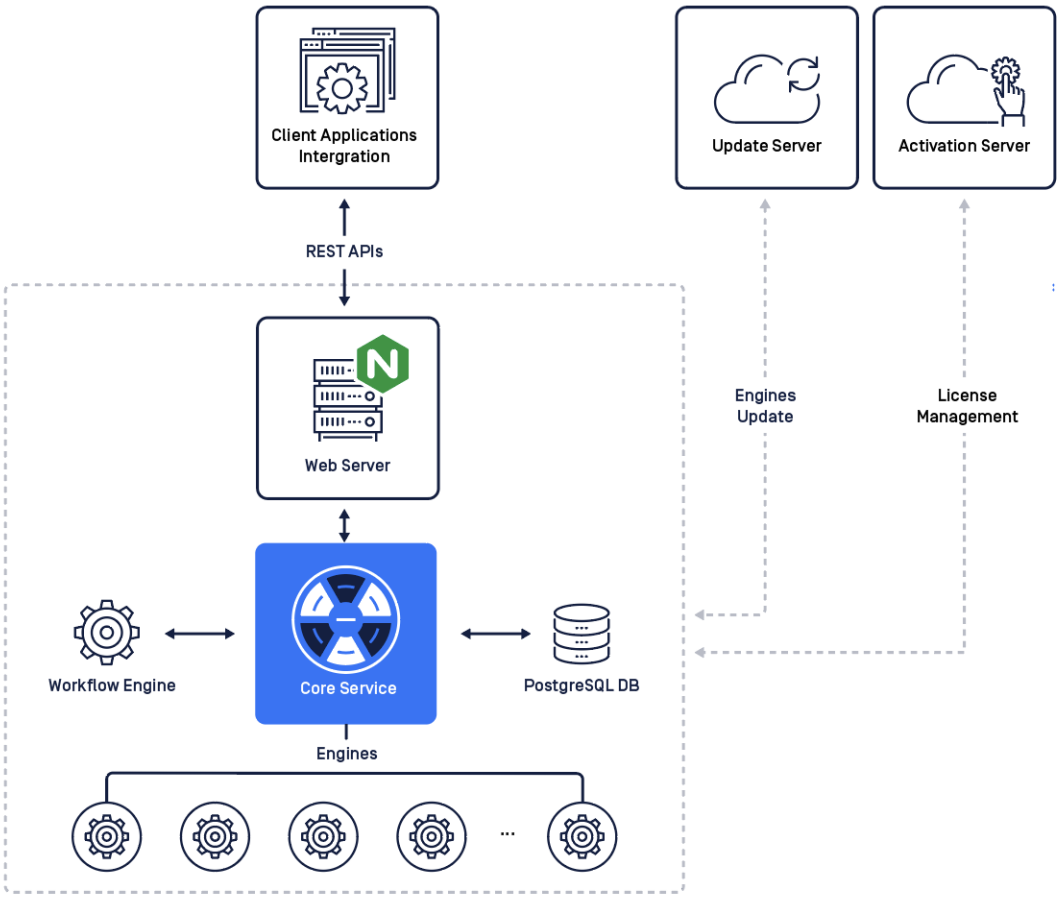
Troubleshooting 2791

Recommended System Configuration 2791
Differences Between MetaDefender Core v4 and v5 2800
Installation 2806
Standalone mode 2807
Shared Database Mode 2813
Deployment automation support 2821
CIS Level 1 Guidelines 2831
Wizard Setup 2833
License Activation 2839
Scripted license management 2844
File Processing 2848
Product Upgrading 2849
Using external Load Balancer 2853
SSL connection for PostgreSQL communication 2858
Performance and Load Estimation 2863
Standalone DB - Windows 2864

Standalone DB - Linux 2880
Shared DB - Windows 2897
Shared DB - Linux 2921
MetaDefender Core Documentation 2945

MetaDefender Core 2946

Recommended System Configuration



MetaDefender Core supports a wide range of deployment options, including air-gapped environments, providing the flexibility to choose the environment that best suits customers' requirements and infrastructure.

- Physical server.
- Virtualization platforms like VMware, Hyper-V and XenServer.
- Cloud Deployment.
- Container Deployment.

> **Note**
> Before installing MetaDefender Core, please refer to the **recommended minimum system configuration** listed below. The requirements are based on a standardized customer deployment and may require more or less resources to achieve the level of performance desired in your environment. Please note that the server specifications are built to allow a high volume daily processing. Please refer to Performance and Load Estimation for more details.

The recommendations below are for MetaDefender Core, API usage only. For any other use cases, please consult the user guide of the licensed products for accurate recommendations.

For certain use cases these might be adjusted and customized on specific needs and SLAs. We highly recommend to engage our ProServ team to assist in fine tuning MetaDefender and get the maximum performance out of your systems.

Microsoft Windows Deployments

[Windows] Supported Operating Systems

Only support 64-bit platforms.

- Windows 10, Windows 11 23H2, 11 IoT (22H2) only for End-User systems (e.g. Kiosk deployments)
  - Based on Microsoft License Terms, you're not allowed to use Windows Desktop as a server, therefore, MetaDefender Core can't be used on these systems for server use cases.
- Windows Server 2016, 2019, 2022

> **Disclaimer**
> End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

> **Plan for legacy OS support**
> OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:
>
> - Ubuntu 20.04 after **April 2025**.
> - Amazon Linux 2 after **June 2025**.
> - Windows 10 after **October 2025**.
>
> If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

[Windows] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 8	8	8 GB	16 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 12	16	16 GB	24 GB + 50 GB + 100 GB
MetaDefender Core 16	16	16 GB	32 GB + 50 GB + 100 GB
MetaDefender Core 20	32	16 GB	40 GB + 50 GB + 100 GB
MetaDefender Core MAX	32	32 GB	120 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core is installed.
- For a better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.
- SBOM processing requires more disk space, please allow at least 15GB free disk for the engine to work properly.

[Windows] Third Party Dependencies

- .NET framework 4.5 or above
- Microsoft Visual C++ Redistributable for Visual Studio 2015-2022 (version 14.38 or higher); x64 version.
- PostgreSQL: Only applicable when using a pre-installed PostgreSQL server running remotely.
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Some engines also have dependencies as described below:

Engine Name	Dependency
Deep CDR	Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 5.8.0 or above)
ESET	MetaDefender Core v4 temporary directory should have more than 200MB free disk space
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
Proactive DLP	• Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 2.0 or above) • .NET 6 dependencies • CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
Scrutiny	Microsoft Visual C++ 2015 - 2019 Redistributable x86 and x64
Systweak	.NET runtime 6.0, 7.0 or 8.0
Bkav Pro	Microsoft Visual C++ 2015 - 2022 Redistributable x64

[Windows] Installation Details

MetaDefender Core on Windows uses "C:\Program Files\OPSWAT" folder for storing resources or the installation directory.

MetaDefender will use its resources folder to store temp files as part of the analysis. **It's recommended to exclude this folder from real-time protection monitoring**. Knowledge base:

- How to set up exclusions on anti-malware software to prevent disruption
- Mitigating scan failure triggered by real-time protection service
- Can local AVs interrupt ongoing scans?

Linux Deployments

[Linux] Supported Operating Systems

Only support 64-bit platforms.

- Red Hat Enterprise Linux 8, 9
- Rocky Linux 9
- Debian 11, 12
- Ubuntu 20.04, 22.04
- Amazon Linux 2 is supported via Amazon Machine Image (AMI)

Plan for legacy OS support

OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:

- Ubuntu 20.04 after **April 2025**.
- Amazon Linux 2 after **June 2025**.
- Windows 10 after **October 2025**.

If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

For Hardended Linux OS

Some AV engines could be failed to initialize and run on MetaDefender Core (e.g. ESET, Kaspersky) due to execution permission required by them on `/var/tmp/ometascan` folder.

Please follow steps at Why have the ESET and Kaspersky scan engines failed to initialize on the hardened Linux OS? for remediation details.

Disclaimer

End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

[Linux] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 5	4	4 GB	10 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 10	8	8 GB	20 GB + 50 GB + 100 GB
MetaDefender Core MAX	16	16 GB	40 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core resides on.
- For better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.

[Linux] Third Party Dependencies

Dependencies list:

- openssl
- grep
- lib32stdc++6 (>= 4.5)
- libc6-i386 (>= 2.10)
- procps
- zlib1g
- libcurl3 (>= 7.19.7)
- libcurl4
- postgresql:
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Not all the above dependencies will need to be installed, it is dependent on different Unix distro and version

Some engines also have dependencies as described below:

Engine name	Dependency
Archive engine	• To extract password protected MS Office files, you need to install .NET 6 dependencies, libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • Rocky Linux 9.x/Red Hat 9.x: libnsl
Deep CDR	• Rocky Linux 9.x: libgomp, ncurses-compat-libs, libnsl • CentOS: libgomp, ncurses-compat-libs • Red Hat 8.x: libgomp, ncurses-compat-libs, libnsl • Red Hat 9.x: libgomp, ncurses-compat-libs, libnsl, compat-openssl11 • Ubuntu/Debian: libgomp1, libncurses5 • libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • .NET 6 dependencies
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
FileType engine	Rocky Linux 9.x/Red Hat 9.x: libnsl
Proactive DLP	libgdiplus, mscorefonts .NET 6 dependencies CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
RocketCyber	libgomp

Note: Some dependencies may need to be installed from external repositories, not from the OS default repositories.

[Linux] Installation Details

MetaDefender Core default installation path is using /var folder for storing resources:

- `/var/lib/ometascan` : installation folder with all its resources (database, DLP processed / CDR sanitized / quarantined file storage, engine package and definition updates).
- `/var/log/ometascan` : application logs.
- `/var/run/ometascan` : runtime folder for Core related processes.
- `/var/lib/ometascan` : all successful deployed engines data files.
- `/usr/lib/ometascan` : all shared libraries required by MetaDefender Core, bundled PostgreSQL server related binaries and libraries.
- `/var/tmp/ometascan/resources` : all temporary processing files for MetaDefender Core to process.

Storage Usage Information

Based on the configuration, MetaDefender Core will need additional disk space to store analysis/result data, quarantined files and temporary files:

	Data Storage
Analysis Reports (Processing History)	Approximately 4.5GB is required for every 1M analysis reports stored. • 100 GB is recommended as a baseline. Database storage requirements are affected by Data Retention settings, engine settings and scanning load.
Quarantined Files	Depends on the customers' dataset and configuration
Sanitized Files	Depends on the customers' dataset and configuration
Temporary processing files	Depends on the customers' dataset and configuration • A 1GB non-archive file requires 1GB free disk space at least. • A 1GB archive file requires (1 * 2)GB free disk space at least. • 50GB is a baseline and should be adjusted based on observation of the solution within the environment, with consideration to peak times and overall file makeup (e.g. complexity, size, archives).
Note	Options are available to control how large and/or complex files are handled to limit the amount of system resources required. See Workflow Configuration Template for more information.

Custom Engines

The recommendations above are specific for MetaDefender pre-packaged bundles.

However for additional Custom Engines, please review the Knowledge base to review additional requirements (if any) for the selected engine.

Browser Requirements for the MetaDefender Core Management Console

One of the following browsers is suggested to view the MetaDefender Core Management Console:

- Microsoft Edge
- Chrome
- Firefox
- Safari

Chrome, Firefox, Safari and Microsoft Edge browsers are tested with the latest available version at the time of release.

Network Bandwidth

In online environment where MetaDefender Core is configured to download engine update packages directly from OPSWAT update server source, then using gigabit network speed (125 MB/sec) is highly recommended to avoid potential network related issue while downloading files, or at least 100 Mbps (12.5 MB/sec) network speed.

Allowlisting URLs

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts':

- https://activation.dl.opswat.com - for license online activation
- https://update.dl.opswat.com - for engines and databases download

Note: *IP address-based allowlisting on your firewall might fail after some time since OPSWAT uses CDN (AWS Cloudfront) to faster delivery updates over the world, and IP address of edge servers might change over time.*

Depending on your location, connections with OPSWAT cloud-based servers be forwarded to specific AWS Cloudfront's edge locations, each is assigned with its own unique IP address ranges. You might also need to allow correct AWS Cloudfront's corresponding IP address ranges by referring to https://ip-ranges.amazonaws.com/ip-ranges.json

Warning

Even the OPSWAT update servers host updates for all of the available engines we support, sometimes custom engines might try to connect to their own cloud for updates, but this can be disabled in firewall and they will be updated just from OPSWAT cloud servers instead.

Differences Between MetaDefender Core v4 and v5

We are very excited to inform you about the release of version 5.0.0 of our MetaDefender Core!

The new version is designed to enhance the product performance, user experience, as well as to provide more flexibility, security, and scalability for our customers.

At the same time, we are also committed to keeping backward compatibility for whom running MetaDefender Core v4, and avoid the need for updating your client integration to move forward with MetaDefender Core v5.

v4 REST API Integration Supported
All public REST API endpoints available on MetaDefender Core v4 will be continously supported on v5.

v4 Product Major Features Supported
All product major features available on MetaDefender Core 4.21.2 (the latest one on v4) will be continously supported on v5.

Seamless Product Upgrade
Upgrading from MetaDefender Core v4 to v5 will be supported and covered with data migration without additional manual steps from users.

Licensing information will be also migrated.

Below are some highlights of the first v5 release, MetaDefender Core 5.0.0:

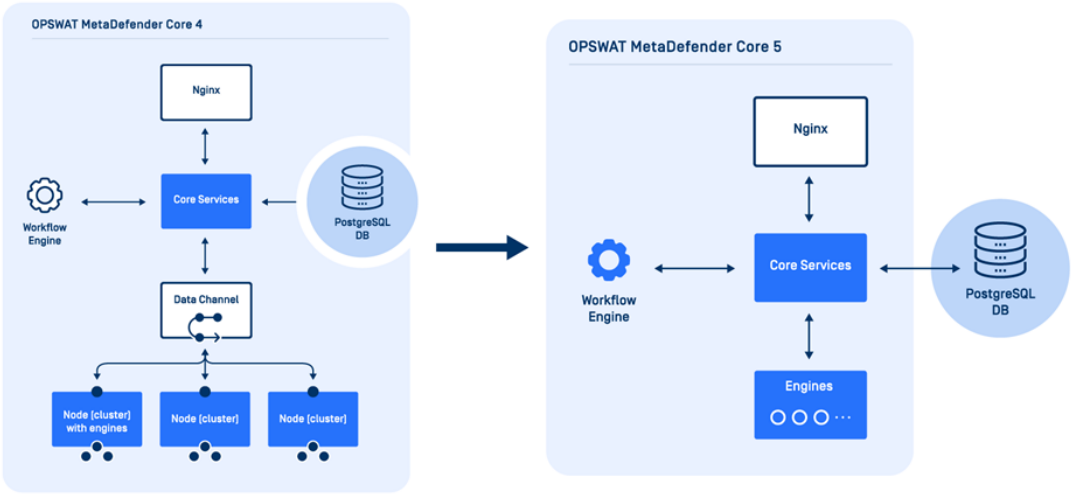
New product architecture

This version introduces a simpler product architecture that speeds up integration and automation, file processing flow, and immensely enhance the product performance and efficiency.

Earlier versions utilized MetaDefender Node concept, designed to support scaling of the scanning infrastructure by distributing scan requests among several different processing Nodes. Product licensing was based on number of Node instances. The benefit of having such a distributed infrastructure is that based on Node loads, Metadefender Core server can always choose the most appropriate Node to assign a new scan task to. However, having nodes in the v4 architecture caused more complexity and possible latency in the processing flow and entailed maintenance cost.

In this release, we retired multi-Node deployment and restructured MetaDefender Core's backend. This alteration significantly improves file processing speed and product efficiency, as well as eliminates extra cost for maintaining node services. Users now can simply install additional MetaDefender Core instances for scalability requirement,

as well as configure their own load balancer to coordinate tasks among multiple MetaDefender Core instances. Accordingly, product licensing is now based on number of the product instances.



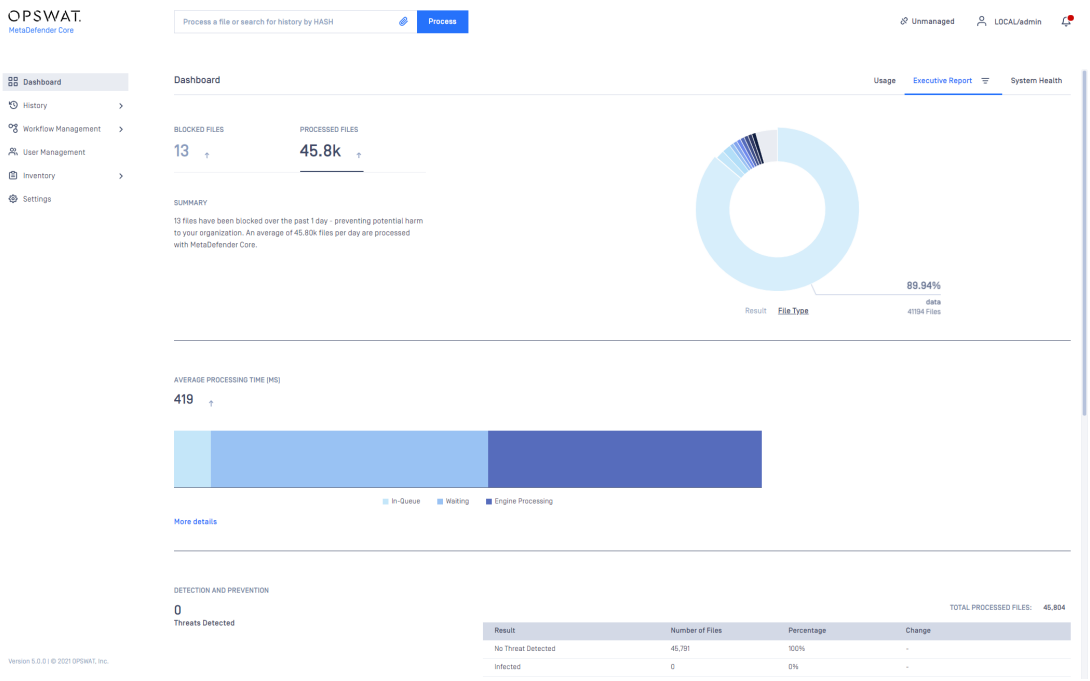
Please be informed that

- All existing administrative APIs related to "Node" on v4 will be kept as it is on v5 to ensure backward compatibility with existing client integrations.
- Applicable settings defined in Windows registry (Windows) or ometascan-node.conf config file (Linux) for Node will be migrated to Core. Details: MetaDefender Configuration

Brand-new product user interface

The second notable update in this version is a brand-new UI for the management console, which was designed with the focus on the quality and thoughtfulness of the user experience. The new administrative console UI provides an advanced set of Core features and observable processing status.

Comprehensive executive dashboard reporting offers dynamic, real-time and actionable threat insights. It also provides visibility of your security posture allowing you to paint a picture of security that aligns with the overall business. The report includes easy-to-read charts, graphs, and statistic explanation that help your security professionals reduce clutter, gather information quicker, spot threat activity early and act swiftly.



We're continuing to improve our product UI to provide users an excellent experience with the application and overall usability. Please expect more powerful UI features coming in next releases.

New compression archive engine for archive sanitization

In previous version, MetaDefender Core's archive engine implemented four functions including decryption, extraction, encryption and compression that with a high volume of files it takes time to process them all. For the purpose of enhancing archive handling performance, this release comes with a separated compression archive engine, which dedicatedly processes archive sanitization (compression and encryption). It is also created to facilitate maintenance and further feature development.

Archive sanitization feature now requires Deep CDR license

As to respect sanitization practice and ensure unknown threats prevention as designed, now starting on version 5, it's mandatory to have Deep CDR licensed for archive sanitization feature enablement on MetaDefender Core workflow configurations.

Archive handing related engines changed

The legacy "Archive" engine on MetaDefender Core v4 will be changed its name to "Archive Extraction" engine, even when you are still on MetaDefender Core v4.

A brand new "compression archive engine" is to dedicatedly handle archive sanitization in v5 (archive compression), that helps relevant archive handling performance improvement, easier for maintainance and further feature development.

Notes when you are still on MetaDefender Core v4 (not upgraded yet)

The legacy "Archive" engine (now name changed to "Archive Extraction" engine) on MetaDefender Core v4.x will still cover both extraction and compression archive tasks as usual.

Even though your MetaDefender Core v4.x might now have a new "Archive Compression" engine auto downloaded, but that engine cannot be deployed (marked as "permanently failed" in relevant engine status check API). This behavior is expected, and not causing any impact to MetaDefender Core processing flow, or its performance though.

- Just in case your client integration has a check for engine status before triggering scan request, then you might want to disable "Archive Compression" engine status on MetaDefender Core v4 using REST API. Detailed instruction: Archive Compression Engine Permanently Failed on MetaDefender Core v4
- When you later upgrade to MetaDefender Core v5 from v4, then you might need to manually re-enable "Archive Compression" engine to make it active and running on MetaDefender Core v5.

Notes for licensing

For offline deployment (MetaDefender Core is not connected to the Internet directly for online update), you will need to re-activate again on OPSWAT portal to retrieve latest engine list in .yml file including "Archive Compression" engine if you are currently licensed with Deep CDR.

Meanwhile, for online deployment, this manual procedure is not required since MetaDefender Core will cover that itself.

Import & export configurations

Not only workflow settings like in previous versions, but also all other general settings (update, security, data retention, server, hashes to skip) and most importantly, user settings, will be able to export and import on MetaDefender Core version 5.

Learn more at Import/Export configuration

Backward compatibility supported

The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

Windows application event logging disabled by default

Applicable only to Windows OS based MetaDefender Core.

Help avoid duplicated logging with separate file option, and improve overall processing performance. Of course, when being still valid to you, you are still supported to enable it back.

CSRF token header renamed

CSRF token originally created for security strengthening and being used by MetaDefender Core management console UI since version 4.21.0 via header `oms_csrf_token`

To get it compliant with some load balancers which do not accept underscore in header name by default, and save you from additional load balancer configuration step, we have now changed the header name to `oms-csrf-token`

This change does not impact to any REST API based client integration.

More comprehensive and accurate error messages on REST APIs

Impacting on following REST API endpoints:

- `PUT /admin/config/session`
- `PUT /admin/config/update`
- `PUT /admin/config/scan`
- `PUT /admin/config/skip`
- `PUT /admin/config/auditlog`
- `PUT /admin/config/webhook`
- `PUT /admin/config/rule`

Upload time and queue time logic updated in JSON scan response

Impacting on following REST API endpoints:

- `GET /file/{data_id}`
- `GET /hash/{md5/sha1/sha256}`
- `GET /archive/{data_id}`
- `POST /file/{async}`

Added new field:

- `receive_data_timestamp`: The timestamp when upload progress started (first byte received) (in milliseconds)
- `upload_time`: Total time elapsed for upload process (in milliseconds)

Modified existing field (new logic):

> **Backward compatibility supported**
> The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

- `upload_timestamp`: The timestamp when upload progress finished (all bytes received) (in milliseconds)
- `queue_time`: Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue time = start_time - upload_ timestamp) (in milliseconds)

Release version 5.0.0 also comes with variety of outstanding features and improvements.

For detailed information about our latest release please view our Release notes

Installation

Starting with MetaDefender Core 4.20.0, the centralized/shared database deployment model is supported (all MetaDefender Core instances share the same database), along with traditional standalone database mode (every MetaDefender Core instance has its own database to manage).

- Standalone mode
- Shared Database Mode

Installation

1. Download the package of your choice from My OPSWAT
2. Install the package on your computer via the Command Line or via the Wizard
3. Open a web browser and point to http://<server name or IP>:<port>
    - The default port is **8008**
    - In case of problem check Inaccessible Management Console page
4. Complete the required steps of the basic configuration wizard
5. You must activate this deployment to use its features

Installation notes

- If the MetaDefender Core package **dependencies are not installed** on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.
- During installation the databases might need to be upgraded. This could take noticeable time depending on database size (eg.: length of scan history).

Standalone mode

Wizard Installation

The Install Wizard is only for the Windows installer (.msi file).

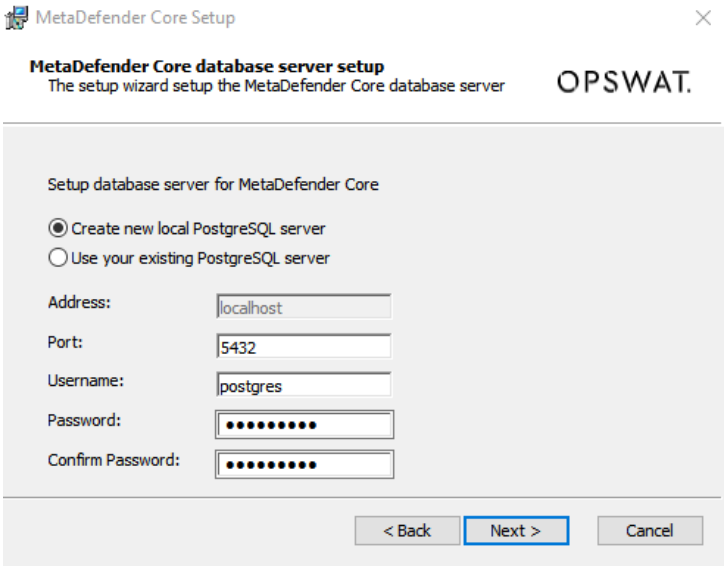
Since MetaDefender Core 4.19.0, we have switched to using PostgreSQL DBMS which needs a dedicated database server. MetaDefender Core supports two modes to setup PostgreSQL server:

Local (Bundle) PostgreSQL server

MetaDefender Core will install a new PostgreSQL server locally in the same box with the product.

> **PostgreSQL user credentials**
> Unicode characters are NOT supported for "Username" and "Password"
>
> "Password" does not accept # (hash) as starting character (e.g. #ABC)

> **The ignition file**
> If the ignition file exist in the designated location, then MetaDefender Core will use the details defined in the ignition file, rather than what is passed via the installation wizard.
>
> More info about the ignition file based installation: Command Line Installation

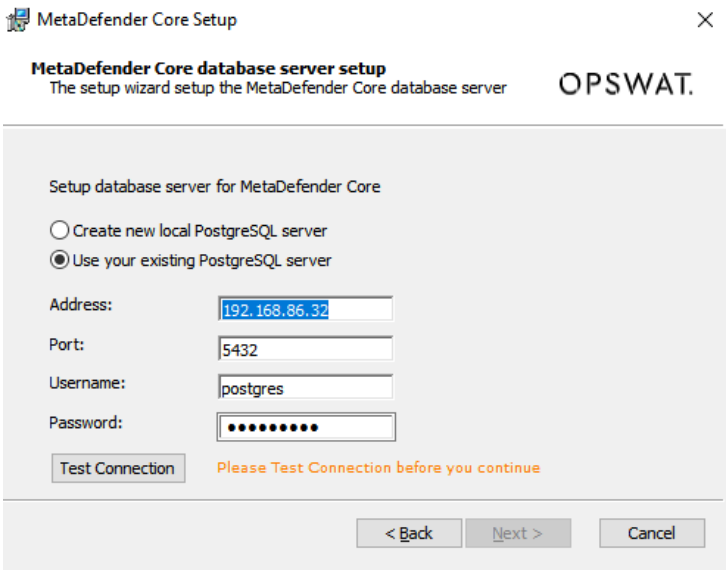


Remote (Existing) PostgreSQL server

MetaDefender Core will connect to remote pre-installed PostgreSQL running remotely.

> **PostgreSQL connection configuration**
> "Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

> **PostgreSQL user privilege requirements**
> MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.
>
> Details: PostgreSQL User Privilege Requirements



Command Line Installation

Ignition File

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Make sure to create the ignition file, if it doesn't exists before you go ahead and install MetaDefender Core via command line.

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

MetaDefender Core supports two modes to setup PostgreSQL server:

1) Local PostgreSQL server

- MetaDefender Core will install a new PostgreSQL server locally, next to the product.
  - A sample iginition file for PostgreSQL server information (***Only non-Unicode characters supported for "user" and "password"__***):

ometascan.conf bash

```
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
```

2) Remote PostgreSQL server

> **Info**
>
> **PostgreSQL version 12 (at least 12.3) or version 14 is required.**
>
> According to PostgreSQL notice, version 12 will stop receiving fixes on November 14, 2024, so we suggest that you use version 14.

MetaDefender Core will connect to a pre-installed PostgreSQL server running remotely and will setup its database.

Make sure to follow steps at Open Connection On PostgreSQL Server first, so that your remote existing PostgreSQL server to accept upcoming connection from MetaDefender Core.

A sample ignition file for PostgreSQL server information:

ometascan.conf bash

```
[dbserver]
type=remote
host=192.168.86.32
port=5432
user=existingserver_admin_user
password=existing_server_pass
```

After installing MetaDefender Core successfully, you may want to remove all credentials info created for [dbserver] section in ignition file for security reason.

> **Warning**
> If the MetaDefender Core package dependencies are not installed on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.

Debian package (.deb)

bash

```
sudo dpkg -i <filename> || sudo apt-get install -f
```

On Red Hat Enterprise 7.x Linux / CentOS package (.rpm)

bash

```
sudo yum install <filename>
```

For systems which enabled GPD check flag:

bash

```
sudo yum install --nogpgcheck <filename>
```

Red Hat Enterprise Linux 8.x package (.rpm)

Install the latest PostgreSQL repository RPM:

> **Note**
> Due to the shorter support cycle on Red Hat Enterprise Linux, all supported versions of PostgreSQL are not available on this platform. The PostgreSQL Yum Repository will integrate with the normal systems and patch management, and provide automatic updates for all supported versions of PostgreSQL throughout the supported lifetime of PostgreSQL. Reference: https://www.postgresql.org/download/linux/redhat/

bash

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

In case installing a remote PostgreSQL on either the same MetaDefender Core server or another server, you need to install the following additional package. PostgreSQL version 12.3 or newer is recommended.

bash

```
sudo dnf install -y postgresql12-contrib
```

Install MetaDefender Core:

bash

```
sudo dnf install -y <filename>
```

Windows package (.msi)

On Windows systems it is possible to install the product by running the corresponding .msi file.

From command line interface it is also possible to install the product by executing

bash

```
msiexec /qn /i <msi file name> <option key>=<option value>
```

where the possible keys and their default values are the following:

Key	Default Value	Description
INSTALLFOLDER	\Program Files\OPSWAT\MetaDefender Core	Customize installation folder for product Example: `INSTALLFOLDER="D:\Products"`
RESTADDRESS	*	REST interface binding IPv4 or IPv6 address ('*' means that service listens on all IPv4 and IPv6 interfaces)
RESTPORT	8008	REST interface binding port
REPORT_ENGINE_ISSUE	true	Enable reporting of engine issue count. (possible values: "true" or "false").

Note

Reporting of engine issue count

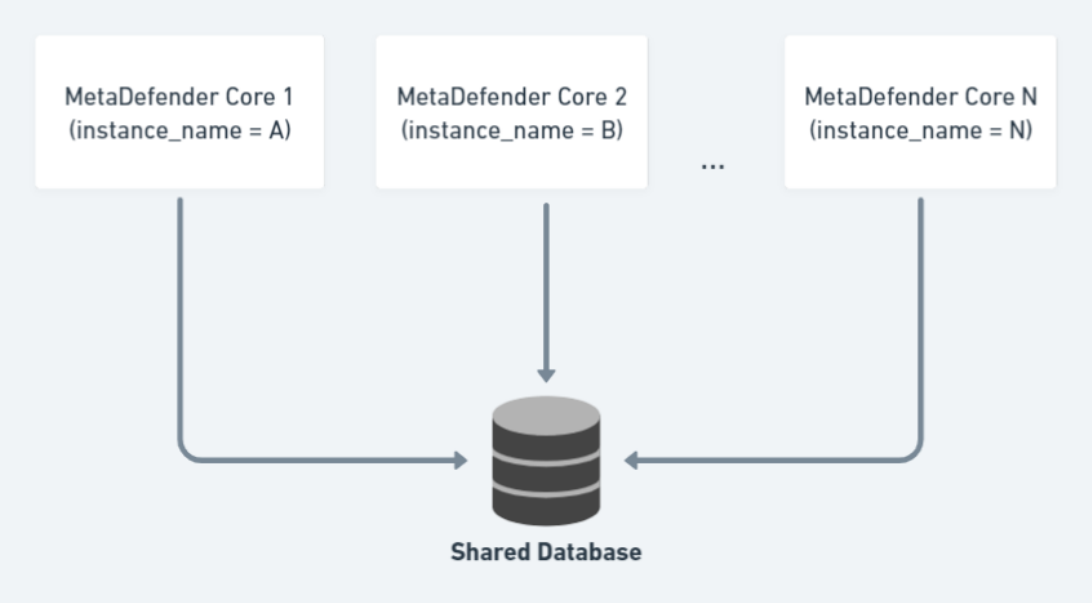
If reporting of engine issue count is enabled, MetaDefender Core server will send only the **number** of initialization errors and **number** of unexpected stops for the specific db/engine version. This information is sent over a HTTPS channel when the product downloads the latest package descriptors. This information is used for early detection of any specific 3rd party engine quality issues.

For details on using msiexec please consult Windows installer documentation.

Shared Database Mode

Starting with MetaDefender 4.20.0, you can use a centralized database deployment model, where all MetaDefender Core instances can share the same external database.

That helps auto-sync all product configurations (**except Single-Sign-On** that requires unique settings for each Core), processing history, live/on-going scan result query, audit, statistics data.



This article demonstrates how to install and setup MetaDefender Core 4.20.0 or newer with shared database mode.

Important notes & current limitations:

Currently not supported

- Does not support Central Management use-case with shared DB mode.
- Does not support to switch from shared database mode back to standalone database mode.
- Does not support to change instance_name of MetaDefender Core in shared database deployment model after installation.
- Does not support upgrading older MetaDefender Core from non-persistent mode to using the shared database mode.
- Does not support upgrading older MetaDefender Core from standalone mode to using the shared database mode.
- Does not support to upgrade multiple MetaDefender Core instances at once in shared database mode
    - Stop all Core instances' scanning service before performing the upgrade.
    - Upgrade each Core server one by one instead.

- For each Core, when the upgrade finished, then you can start using that Core for scanning without waiting for the other Core instances' upgrade to be done.

> **Disclaimer**
> MetaDefender Core does not support mode switching.

Requirements for shared database setup:

- The central remote PostgreSQL server (for shared database) could be utilized with a built-in (native) PostgreSQL database server installed with one of the MetaDefender Core instances in the network. Or using a dedicated existing PostgreSQL database server (see detailed requirements for PostgreSQL version at Recommended System Configuration).
- Before setting up, make sure to follow steps at Open Connection On PostgreSQL Server so that all MetaDefender Core instances could connect to the central remote PostgreSQL server.
- MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core, details: PostgreSQL User Privilege Requirements
- All MetaDefender Core instances must be installed on the same Operating System (Windows / Linux).
- The first MetaDefender Core instance connects to the shared database must be ready first (license activated, users/groups/roles created) before setting up other MetaDefender Core instances to connect and use that shared database.

Engine update:

- If engine updates is configured to be via Folder source, all MetaDefender Core instances must be able to access to the configured update folder, and the option DELETE FILES AFTER IMPORT will be unchecked and prevented for modification.

Authentication:

- Single-Sign-On (SSO) authentication settings (if using) cannot be shared among MetaDefender Core instances because SSO requires an unique login URL for every different Core.
- If an user is locked for login on a certain MetaDefender Core instance(e.g. exceeded the number of failed login times), that user is not prevented for logging in on any other instances.
- If an user is locked for sign-in on a certain MetaDefender Core instance (e.g. exceeded the number of failed login times), if the admin releases lockout on that user on that MetaDefender Core instance, that user is still not able to login on other MetaDefender Core instances.

Sanitized / DLP / Quarantined location changed after MetaDefender Core install:

Windows:

Shared storage path will be stored in 3 strings under key global in registry: quarantinepath, dlppath, sanitizepath as below

(Default)	REG_SZ	(value not set)
dbmode	REG_SZ	4
dlppath	REG_SZ	\\10.40.50.100\test
instance_name	REG_SZ	481rvj
quarantinepath	REG_SZ	\\10.40.50.100\test
restaddress	REG_SZ	*
restport	REG_SZ	8008
sanitizepath	REG_SZ	\\10.40.50.100\test

You can set either 1, 2 or 3 different paths for the 3 settings above (means that quarantined, DLP and sanitized files will be stored in same or different storages corresponding). In case shared storage is not set in installation, values of 3 settings above will be empty.

Restart MetaDefender Core service after changed.

Linux:

Open file `/etc/ometascan/ometascan.conf` and set shared storage path for 3 keys quarantinepath, dlppath, sanitizepath under section [global]

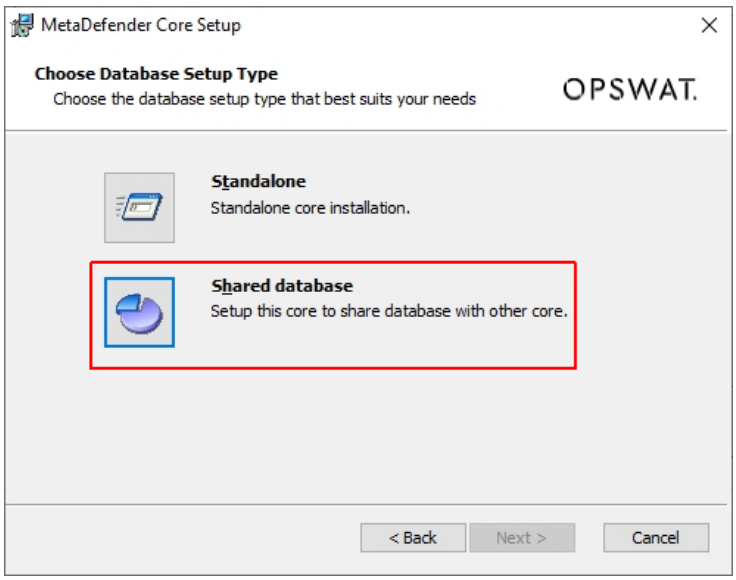
Restart ometascan service after changed.

Wizard Installation (Windows Only)

> **Info**
> Only applicable to MetaDefender Core 4.20.0 or newer using wizard on Windows OS.

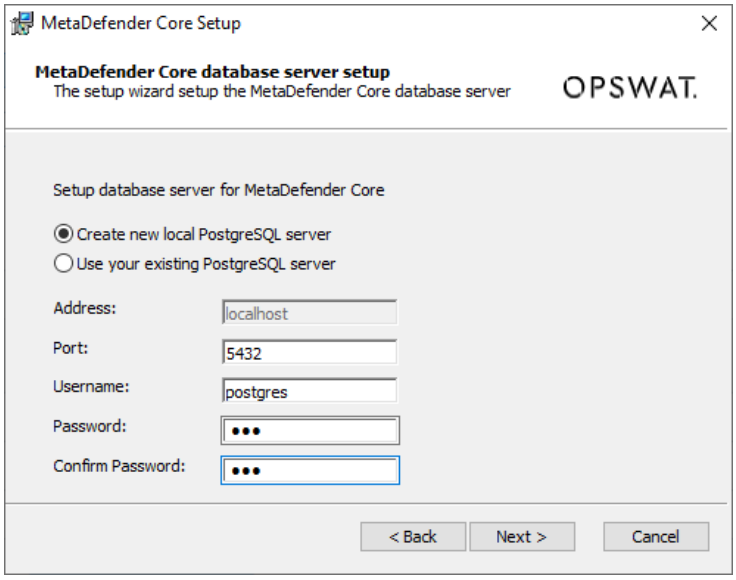
Clean install scenario

1. **Choose Shared database option:**

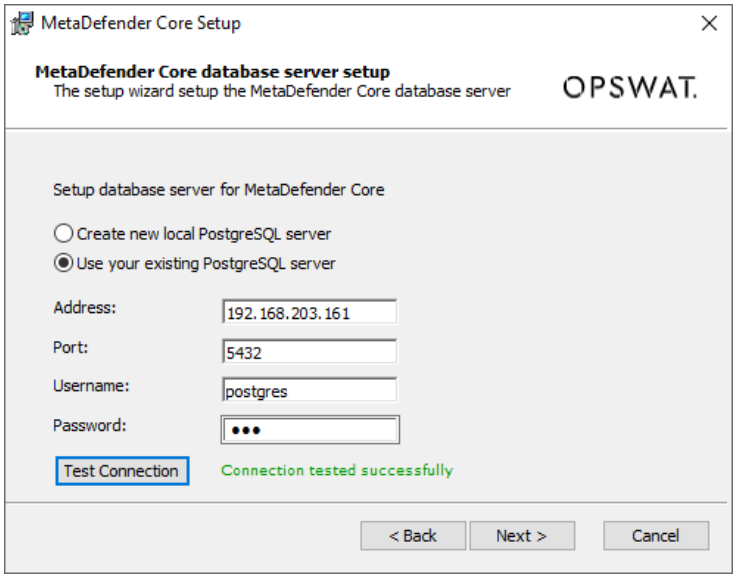


2. **Setup database server for MetaDefender Core:**

You can choose to create a native / built-in PostgreSQL server and setup the shared database on (then other MetaDefender Core instances will want to connect to that PostgreSQL server afterwards)



Or choose to connect to a pre-setup existing PostgreSQL server:



PostgreSQL connection configuration

"Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

PostgreSQL user privilege requirements

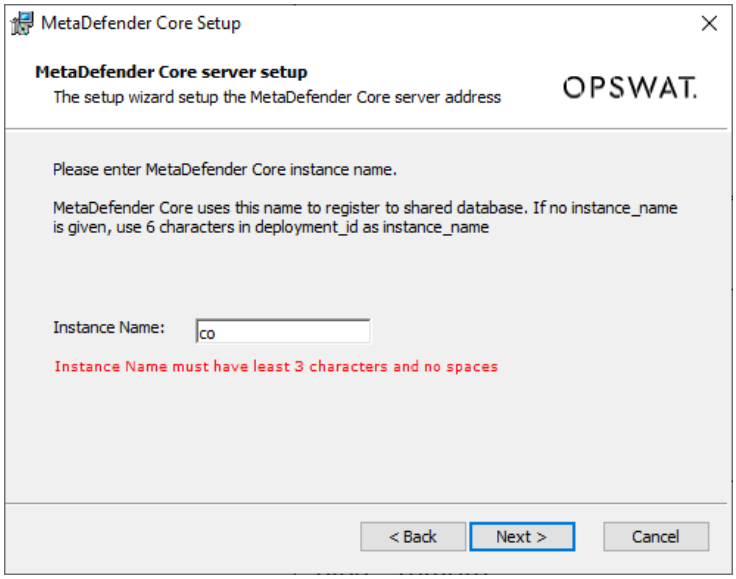
MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.

Details: PostgreSQL User Privilege Requirements

3. **Create an unique instance name for that MetaDefender Core (different from other MetaDefender Core instances):**

Instance name must contain at least 3 characters and no white space.

If you choose to leave it blank, which means it will use the default instance name with 6 characters extracted from that MetaDefender Core deployment ID.

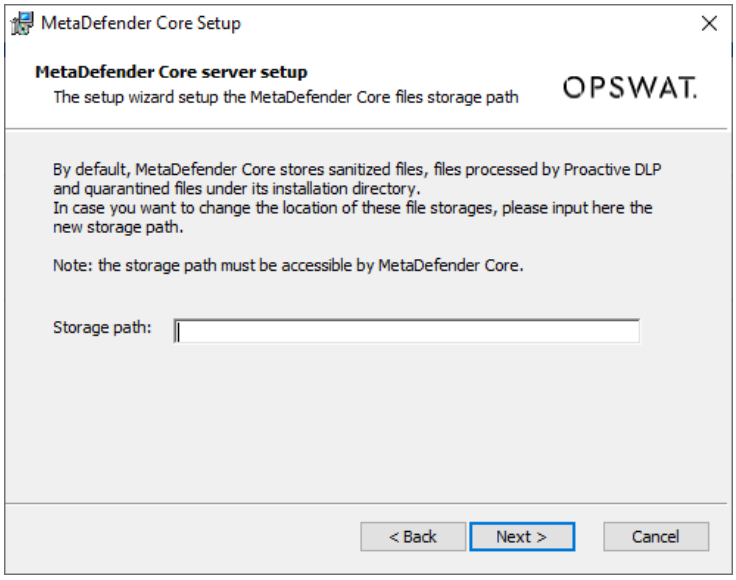


4. **Input the file storage path (sanitized, DLP, quarantined files)**

In the shared database model, all MetaDefender Core instances could use the same folder path to store all their own sanitized, DLP, quarantined files after processed.

The default location is: ***<MetaDefender Core installation folder>\data\ {sanitized|dlp|quarantine}***

If you wish to use that default location, just leave the input box blank and press Next.



Otherwise, you can specify a different location of sanitized/dlp/quarantined files. It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core.

For example:

- C:\new_folder
- \192.168.1.1\new_folder

Command Line Installation

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Clean install scenario

1. Create ignition file if not existed

- Windows: **C:\OPSWAT\ometascan.conf**
- Linux: ***/etc/opswat/ometascan.conf***

An example of ignition file:

ometascan.conf bash

```
eula=true

[global]
dbmode=4
instance_name=MetaDefenderCoreA
storage_path=D:\path\

[user]
name=admin
password=admin
email=admin@local

[dbserver]
type=remote
host=192.168.1.1
port=5432
user=postgres
password=your_own_password
```

Field	Required	Mapping to wizard UI	Possible values
global/dbmode	optional default is 1 (standalone)		1 - Standalone 2 - Central Hub deployment 3 - Non-persistent **4 - Shared**
global/instance_name	optional default is 6 characters of deployment ID		Unicode and number At least 3 characters No whitespace
global/storage_path	optional default is empty		Path to storages (DLP, quarantined, sanitized files) It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core. For example: • C:\folder • \192.168.1.1\folder

2. Run installers using command line to install

Please refer to Command Line Installation for installation command.

Deployment automation support

Note

An Ignition file could be utilized for either / both of following use-cases:

1. Pre-define PostgreSQL server information for MetaDefender Core to connect and setup database: only required on Linux installation, or Windows with command line installation. Learn more details at:
    a. Standalone Mode > Command Line Installation (applicable on version 4.19.0 or newer).
    b. Shared Database Mode > Command Line Installation (only applicable on version 4.20.0 or newer).
2. Automating to accept EULA, create a default local admin user and import MetaDefender Core configurations upon installed: Those steps could be automated using ignition file described in this page.

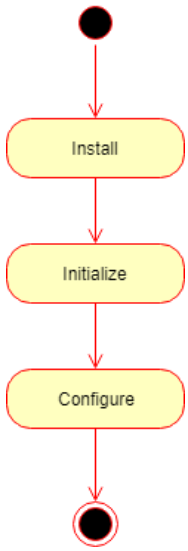
Ignition file location to create:

- **Windows**: `C:\OPSWAT\ometascan.conf`
- **Linux**: `/etc/opswat/ometascan.conf`

The product supports fully automated deployment. It means that it can be installed and configured with no human interaction.

The automated deployment can be split to three steps on a high level:

1. Installation
2. Initialization
3. Configuration



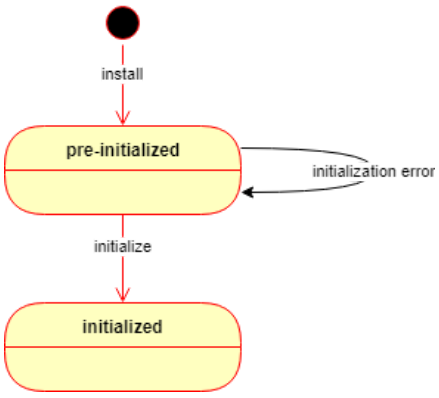
Installation

To automate the installation, install the product from the command line and provide the installation-time options as parameters to the installer. For further details see Command Line Installation

After the installation is complete, the product starts up and waits in a pre-initialized status. The product may be initialized in two ways:

1. Manually using the Wizard Setup or
2. Automatically using an *ignition file* (see below).

> **Warning**
> If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried after fixing the problem (e.g. placing the ignition file in its lookup location) and restarting the OPSWAT MetaDefender Core service.
>
> Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

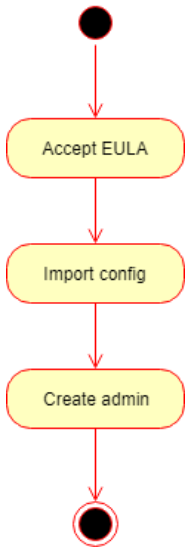


Initialization

Initialization is the process of bringing the product to an operable status.

Basically the initialization consists of the following steps:

1. Accept the End User License Agreement (EULA),
2. Import product configuration and
3. Create the first administrator user account.



Ignition file

The initialization process can be configured in a file called the *ignition file*.

> **Warning**
> In Linux, the ignition file and the folder /etc/opswat should have the permission 755

Ignition file location

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

The ignition file must be in conf format.

ometascan.conf bash

```
eula=true
[user]
name=admin
password=admin
email=admin@local
apikey=44c96cbb4e4e6c0a1d16bcb3c3da24def317
[config]
import=config_package.zip
import_password=123
[security]
session_cookie_samesite=Lax
[internal]
skip_startup_pages=true
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
private_username=my_internal_db_user
private_password=mypassword
[ocm]
ocm_url=https://my.opswat.com
reg_code=XXXXXXXX
```

Ignition file fields

The ignition file accepts the following fields:

Section	Key	Required	Accepted Values	Description
	eula	Optional (mandatory when having [user] section)	true false (default)	Whether to accept the End User License Agreement. This key must be set at the top of ignition file. This key must be set to "true" to accept the EULA. Any other value will cause the initialization to fail. When having [user] section defined, then this key must be set.
[user]		Optional		Initial administrator user account properties. The Administrator role is granted to the account.
	name	Optional	string	User name for the initial administrator user account.
	password	Optional	string	Password for the initial administrator user account. WARNING! Clear text password The password in this configuration file must be stored in its clear-text format and as so it may be visible for unauthorized parties.
	email	Optional	string	E-mail address for the initial administrator user account.
	apikey	Optional	string	API key for the initial administrator user account.
[config]		Optional		
	import	Optional	string	Path to a file in json/zip format that contains a previously exported configuration to be imported. Zip-format file contains a full configuration including user management settings, so it will overwrite [user] section.
	import_password	Optional	string	Password to decrypt .zip configuration package.
[security]		Optional		

Section	Key	Required	Accepted Values	Description
	session_cookie_samesite	Optional	Strict Lax (default) None	Declare if session cookie should be restricted to a first-party or same-site context.
[internal]		Optional		
	skip_startup_pages	Optional	true false (default)	Skip all welcome and other startup pages (e.g. upgrade result) when installing / upgrading MetaDefender, and going straight to the product index default page.
[dbserver]		Optional		
	type	Mandatory when having [dbserver] section	local remote	Where the PostgreSQL database server should physically locate. "remote" means PostgreSQL server and MetaDefender Core are not in the same machine.
	host	Mandatory when having [dbserver] section	string	IP address / domain name of the server where PostgreSQL server locates. "localhost" can be used when applicable.
	port	Mandatory when having [dbserver] section	number	Port of PostgreSQL server is listening for connections from clients (i.e. MetaDefender Core).
	user	Mandatory when having [dbserver] section	string	PostgreSQL server's user. SUPERUSER privilege is required for MetaDefender Core to setup its database and extensions for the first time. **Only non-Unicode characters supported.**

Section	Key	Required	Accepted Values	Description
	password	Mandatory when having [dbserver] section	string	PostgreSQL server's user credentials. **Only non-Unicode characters supported.**
	private_username	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
	private_password	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
[ocm]		Optional		
	ocm_url	Optional	string	URL to My OPSWAT server.
	reg_code	Optional	string	A registration code to enroll to My OPSWAT server.

Detailed initialization process

1) After the product has been started, it looks for the ignition file in the designated location.

Windows: C:\OPSWAT\ometascan.conf ** Linux: /etc/opswat/ometascan.conf

** 2) If an ignition file is found, then

2.1) It gets validated, and if it is valid, then based on the information found in the ignition file:

- The EULA is accepted,
- The configuration is imported,
- The administrator account is created.
- If any of the above steps fails, then the error is logged, and the initialization gets terminated.

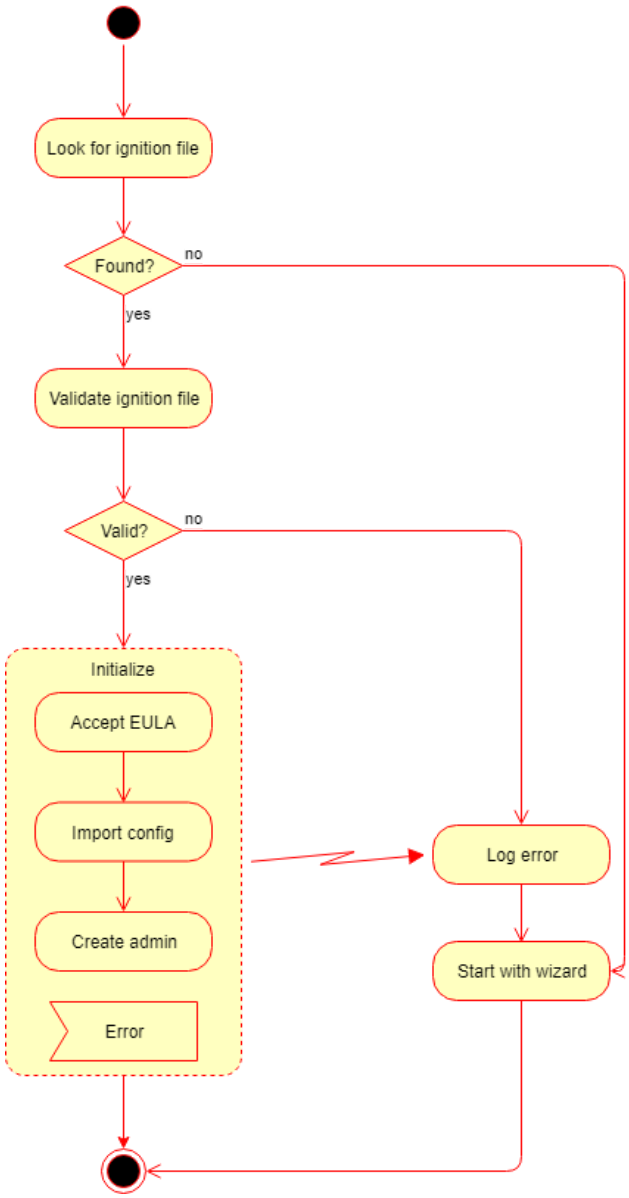
> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

2.2) If it is not valid, then the error is logged, and the initialization gets terminated.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

3) If there is no ignition file, then no initialization is performed.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.



Warning

If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried fixing the problem (e.g. placing the ignition file to its lookup location) and restarting the OPSWAT MetaDefender Core service.

Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

Configuration

After the initialization is complete, the product is ready with the default and the imported configuration.

This configuration can be later changed calling the configuration API functions. For further details about the API see Configuration related APIs API

CIS Level 1 Guidelines

Support Red Hat Enterprise Linux 9
For more details about Center for Internet Security (CIS) please refer to this document

Instruction steps

I. Install OpenSCAP

bash bash

```
yum install openscap-scanner scap-security-guide
```

II. Generate a result file and a HTML report using OpenSCAP scanner tool

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

III. Remediation of CIS Level 1 issues

Generate a remediation script based on the `ssg-rhel9-ds.xml` file:

bash

```
oscap xccdf generate fix --profile
xccdf_org.ssgproject.content_profile_cis_server_l1 --fix-type bash --output
remediations.sh /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

And execute remediation script.

bash

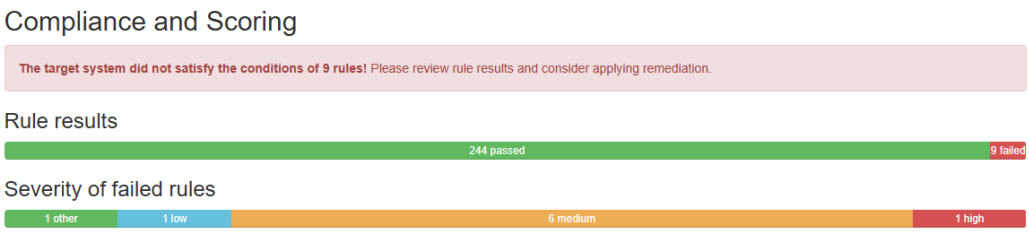
```
./remediations.sh
```

IV. Review the results after remediation

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

Example:



Notes

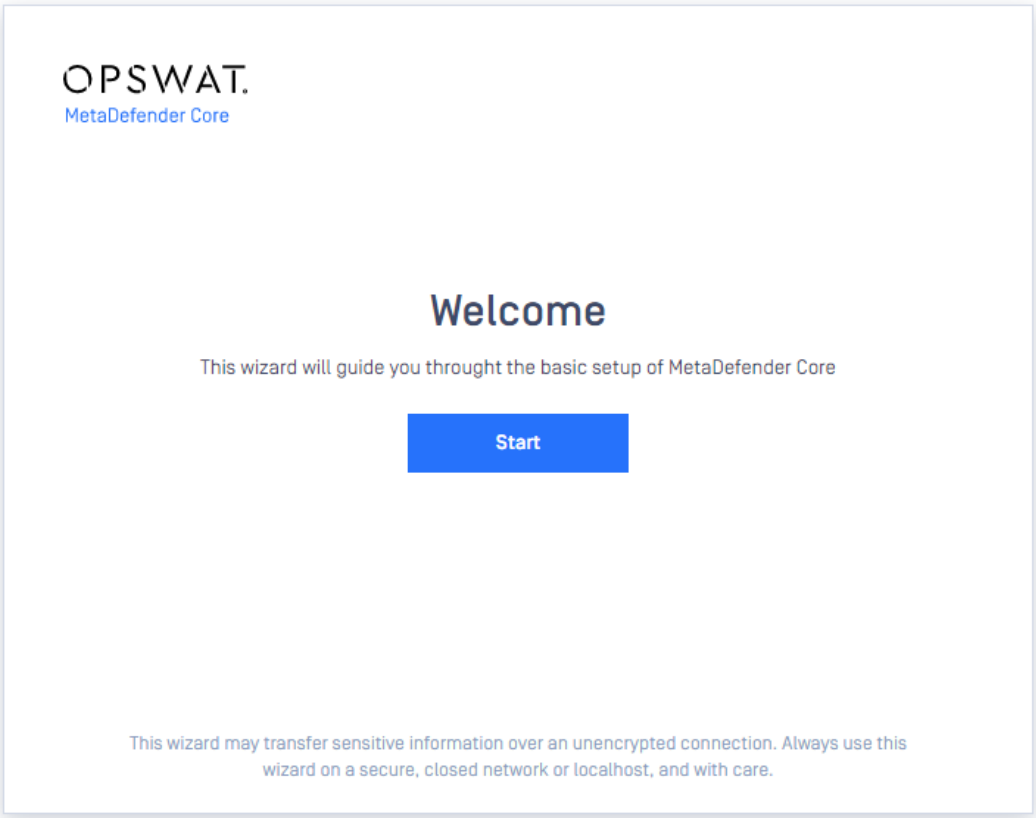
- CIS Level 1 requires `/tmp` folder to be mounted in a separate partition. Please ensure that that new partition have enough disk space for MetaDefender Core to run.
- CIS Level 1 requires that "Ensure No World-Writable Files Exist".
    - For now, when freshly installing MetaDefender Core, all its binary files meet the requirement.
    - When installing/updating engines, some engines might create additional files for its operation, and it might violate this requirement. In this case, you need to again execute the remediation script in the step III.

Wizard Setup

Introduction

Access the MetaDefender Core management console for the first time (i.e. http://localhost:8008), you are to complete a quick walkthrough for wizard setup in order to be able to use the product.

The administrative management console will be available only upon you have successfully finished this wizard stage.

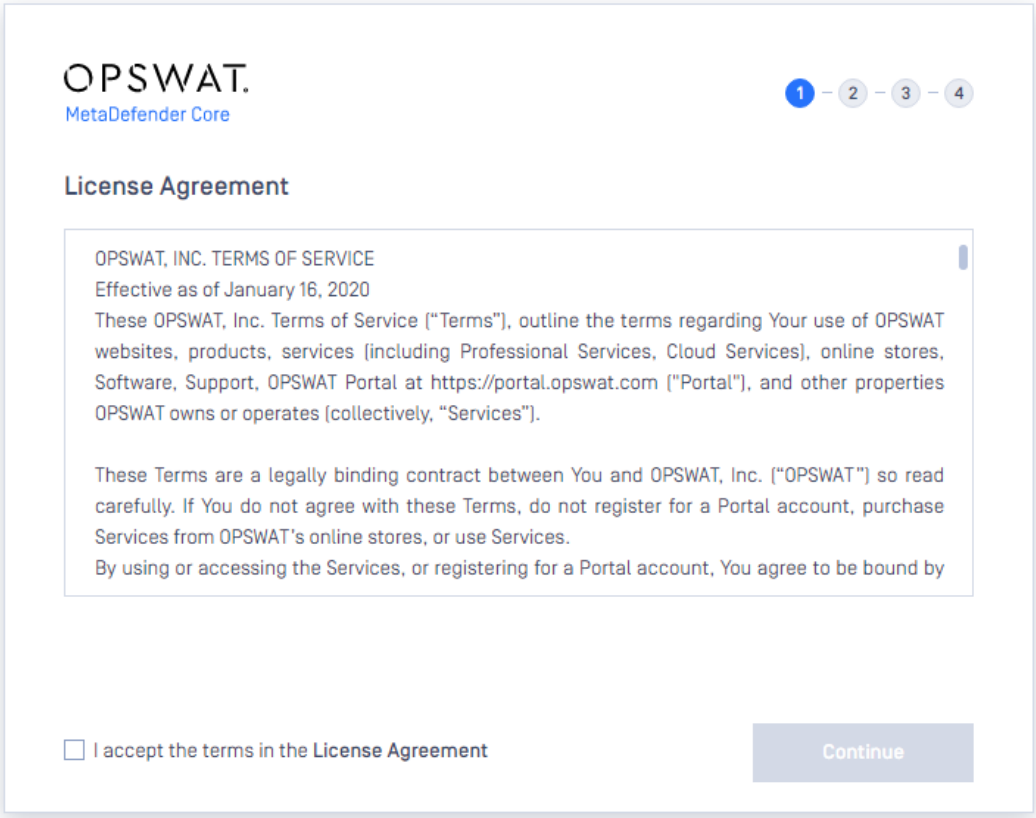


Warning

This wizard may transfer sensitive information over an unencrypted connection. Always use this wizard on a secure, closed network or localhost, and with care.

Basic configuration steps

End-User License Agreement



In the first page you can find the End-User License Agreement. You have to accept the terms before moving on.

Please read through the EULA (https://docs.opswat.com/policies/privacy-policy/portal-privacy-policy) carefully and if you agree with it, check `I accept the terms in the **License Agreement**` and click `Continue` to proceed.

Local Admin User Setup

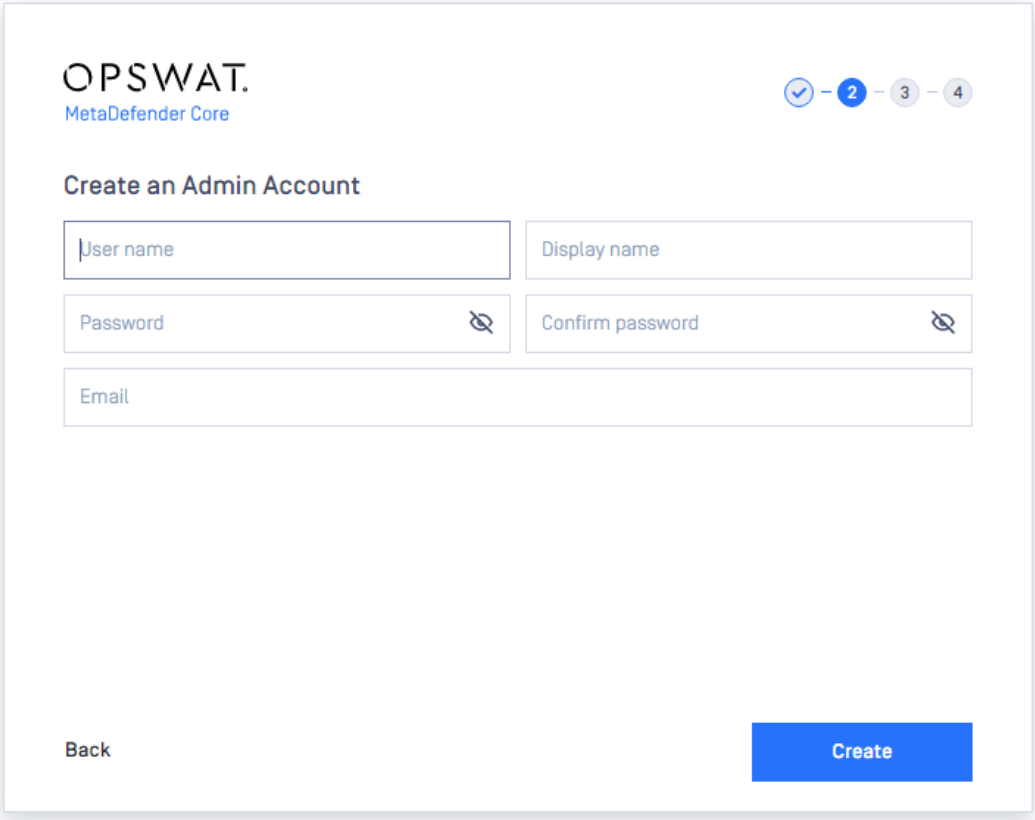
The next step is to set up a default local administrator account.

This account will be the first one being able to access the administrative management console, and to create accounts for other users. You have to fill out all fields in this page to be able to move forward. When you are done, click NEXT to continue.

> **Info - User Directory**
> The administrator account, that is created via the basic configuration wizard, is always added to the LOCAL user directory as a member.
>
> Learn more about user directory: User directories

The following information is required for the administrator account:

ACCOUNT NAME	The unique name of the account that is used at the time of login and in log messages for accountability.
ACCOUNT DISPLAY NAME	Name of the person bound to this account. This name (appended to the name of the account's user directory) is displayed in the top right corner of the administrative management console.
PASSWORD	Password of the user bound to this account that is used at the time of login. Passwords sent clear-text As long as TLS is not configured for the basic configuration wizard, passwords are sent clear-text over the network and may be disclosed to unauthorized parties. As a mitigation action: 1. Either use the wizard on *localhost* or on a direct network connection, or 2. Enabling HTTPS as soon as possible and change the password immediately if it has already been set.
EMAIL	Email address of the person bound to this account.



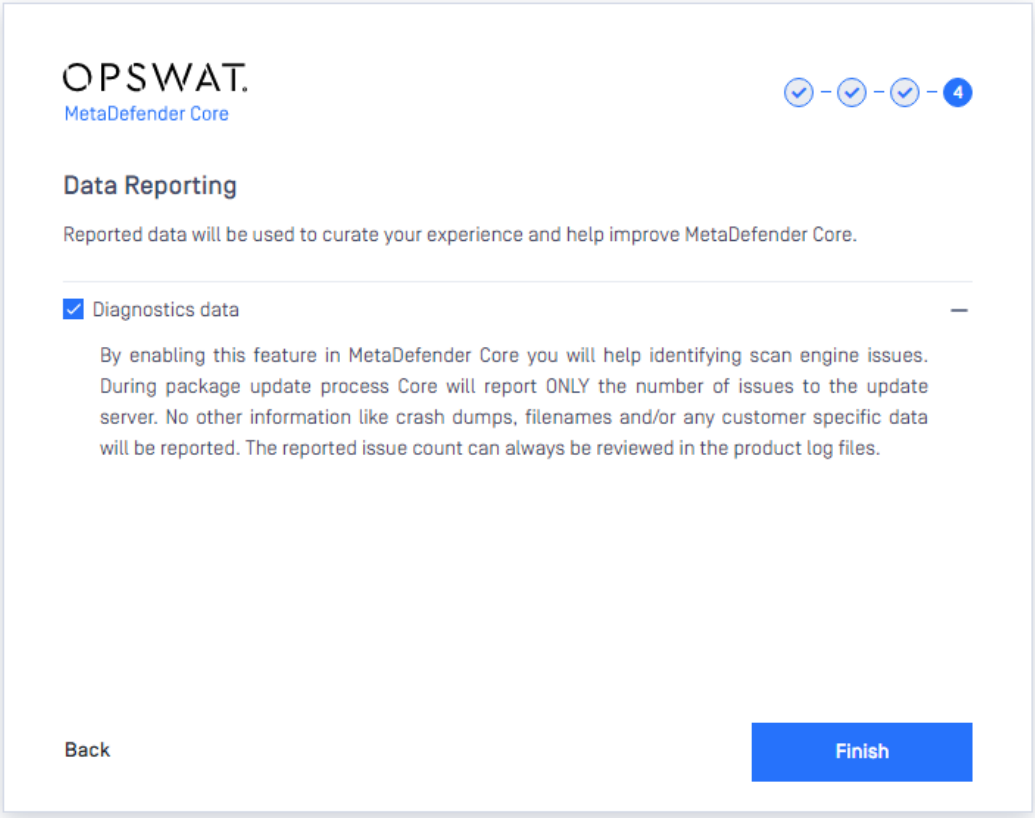
License activation

For license activation details see License Activation

Data Reporting

Enabling this option to allow MetaDefender Core collect its own engine issues and report back to OPSWAT for further analysis.

MetaDefender will not collect any other information.

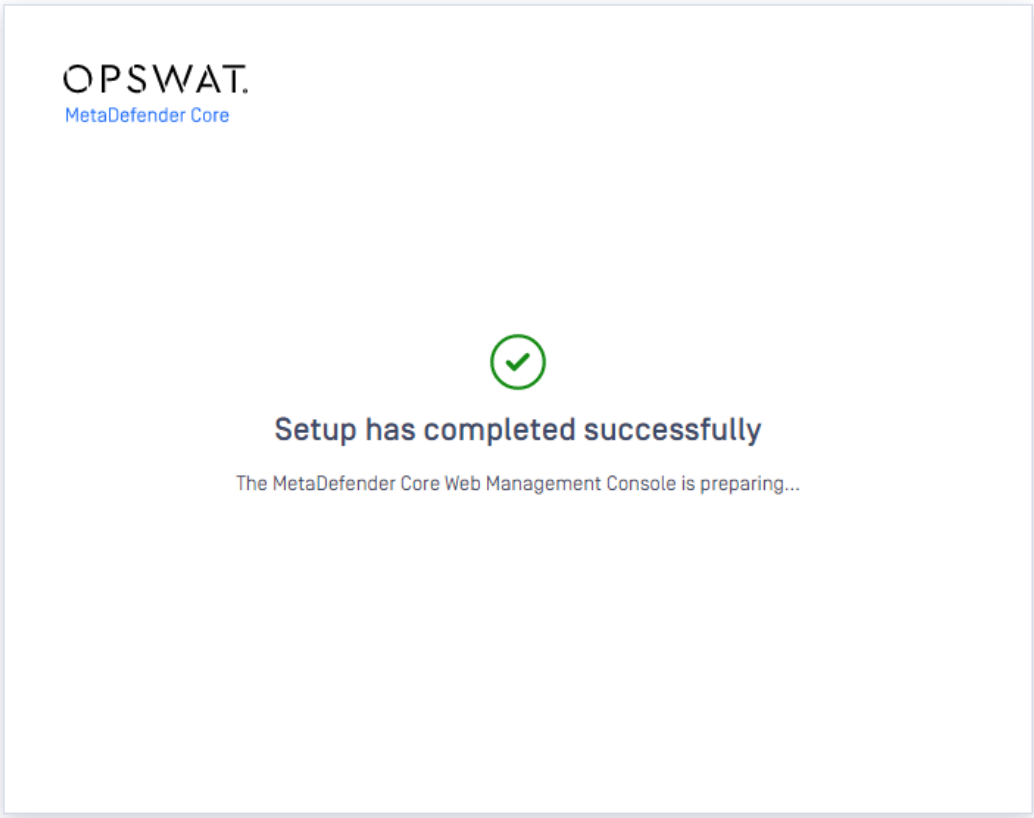


Wizard completion

After you have completed every steps you are ready to finish the wizard and start using the product. Click the `Fishish` button to complete the wizard.

> **Warning**
> The product's service will be restarted and the browser will be redirected to the administrative management console. This could take several seconds.

You can login to the Web Management Console with the administrator user that have just been created in the previous steps.



Setup HTTPS

Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network. Websites, like the Web Management Console, are able to use TLS to secure all communications between their servers and web browsers.

The TLS protocol aims primarily to provide confidentiality (privacy) and data integrity between two communicating computer applications.

Warning

No TLS for the wizard

By default, TLS is not enabled for the basic configuration wizard. As a consequence sessions between the wizard's backend and the browser may be insecure.

Info

Performing the same steps as for the Web Management Console, it is possible to set up TLS for the basic configuration wizard. Remember completing the TLS setup before launching the wizard.

For instructions to set up TLS see Enabling HTTPS

License Activation

##

To activate your installation go to the **Settings > License** menu in the Web Management Console. If you have no valid license, you will only see your installation's **deployment ID**. You will also see a warning in the Web Management Console header.

After successful activation, the product will start downloading the latest available scan engines and malware databases. You can follow the status of the scan engine installation on the **Inventory > Modules** page.

When all the licensed engines are installed, you can start using the MetaDefender Core to process files - File Processing.

Deployment ID

Each MetaDefender Core instance represented by an unique system fingerprint called `Deployment ID`

MetaDefender Core licensing is purely based on deployment ID activated. Therefore, everytime your MetaDefender Core is activated with a different deployment ID, then it will be captured as a different activation record accumulated into your total activation times allowed.

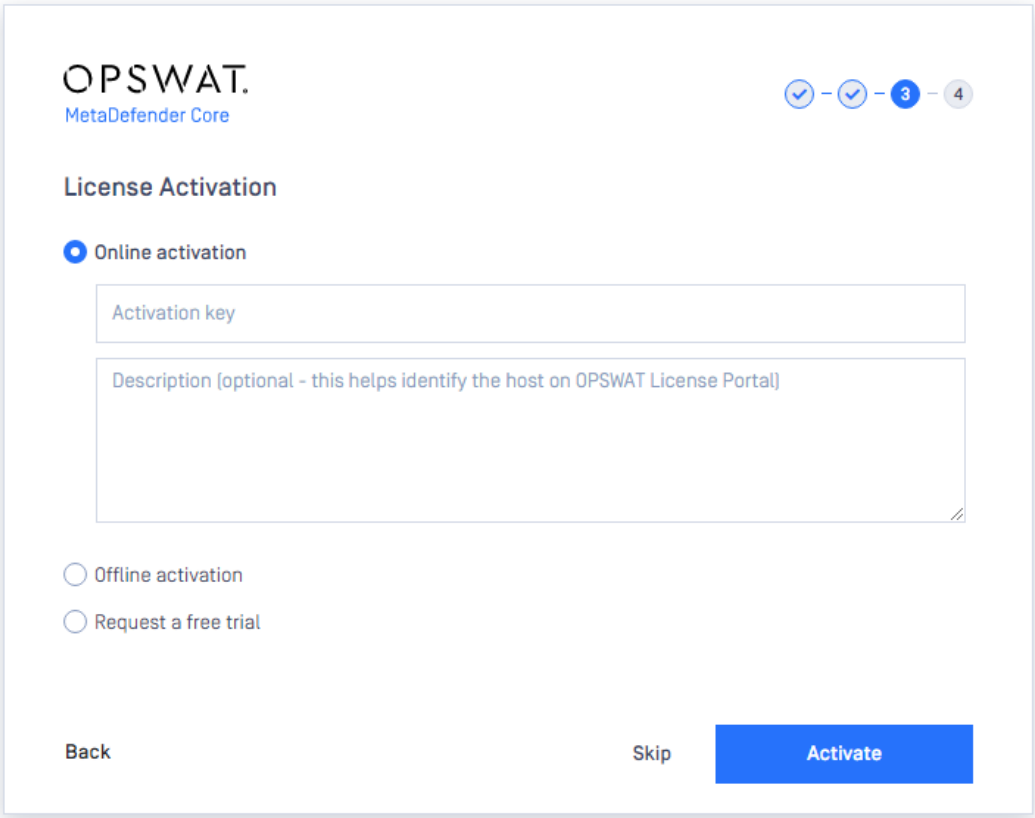
> **Notice**
> Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you alter the system where MetaDefender Core relies on (e.g. network card, OS hostname).
>
> Deployment ID is an UUID version 4 random based string. Learn more at: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you update the system where MetaDefender Core relies on.

Activation Modes

There are three different options for the product activation:

- Online activation: MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.
- Offline activation: Requiring you to upload a license file (.yml). That license file could be retrieved by activating your license via OPSWAT Portal. Follow the displayed instructions for details.
- Request trial key online: For evaluation purpose, you can receive an trial Activation key via email. Follow the displayed instructions for details.



Online activation

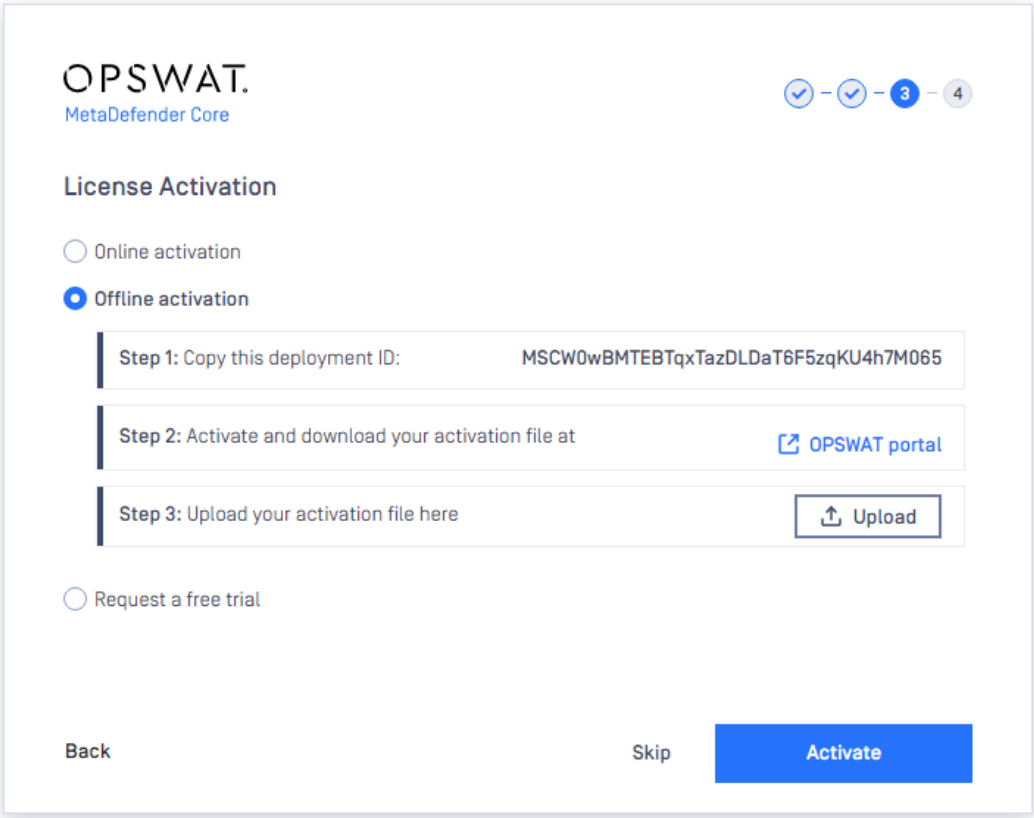
MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.

Required to input your Activation License key, and optional for description (but we recommend this for troubleshooting and auditing purpose if needed).

> **Warning**
> If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts' for license activation:
>
> - https://activation.dl.opswat.com Note: *IP address-based whitelisting on your firewall might fail after some time since OPSWAT uses CDN (Content Delivery Network) to faster delivery updates over the world, and IP address of edge servers might change over time.*

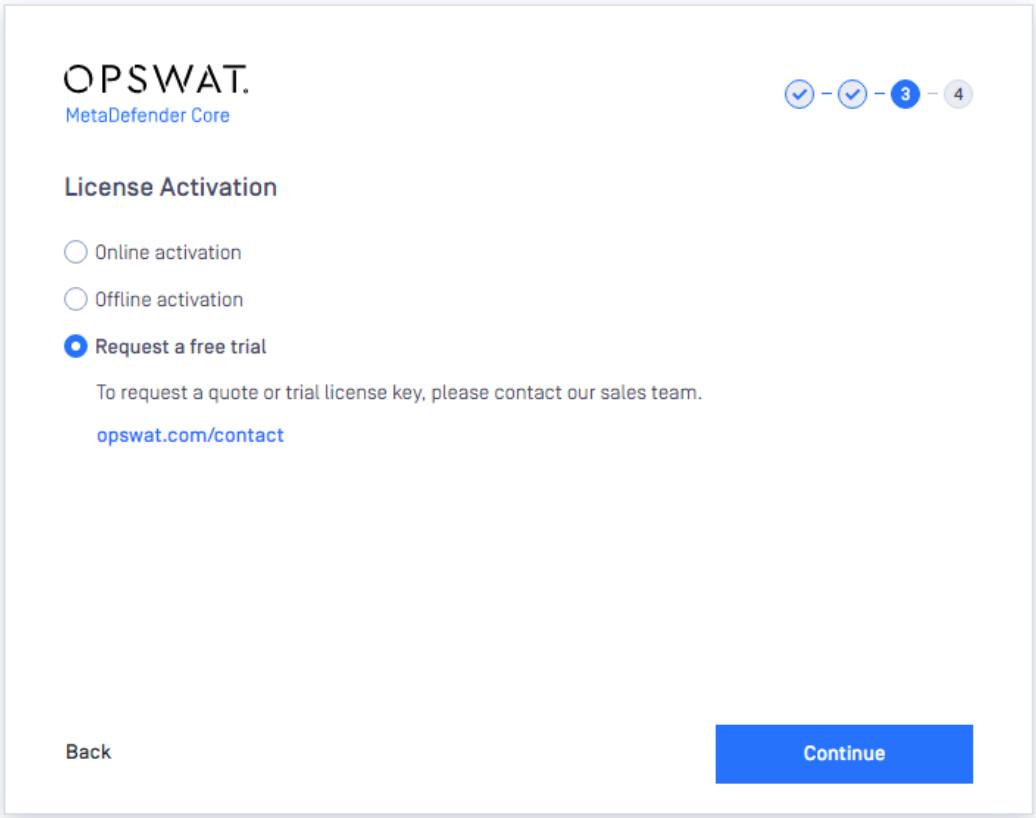
Offline activation

With no internet connection on the server the Metadefender v4 instance may be activated indirectly from a different machine, that has internet connection. The Deployment ID of the Metadefender v4 instance and the the Activation key received at the time of purchasing the product will be required. Follow the steps on the screen to activate the product offline.



Request trial key online

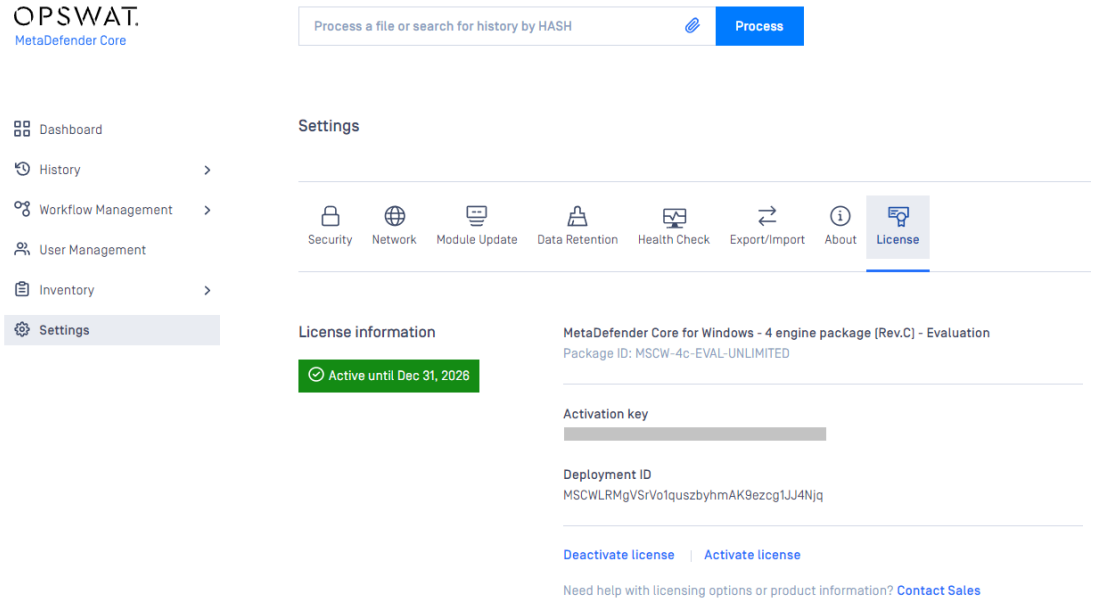
An evaluation license could be acquired by contacting our sales team.



Check Your License Details

For more license details and activating your MetaDefender Core, go to Settings > License menu tab on the Web Management Console:

- Product name: product name as on your order (e.g. MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation)
- Package ID: product identification as on your order (e.g. MSCW-4c-EVAL-UNLIMITED)
- Expiration: last day of license validity
- Activation key: key used for activating product
- Deployment ID: identification of this installation


OPSWAT.
MetaDefender Core
Process a file or search for history by HASH
Process
Dashboard
History
Workflow Management
User Management
Inventory
Settings
Settings
Security
Network
Module Update
Data Retention
Health Check
Export/Import
About
License
License information
Active until Dec 31, 2026
MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation
Package ID: MSCW-4c-EVAL-UNLIMITED
Activation key
Deployment ID
MSCWLRMgVSrVo1quszbyhmAK9ezcg1JJ4Njq
Deactivate license
Activate license
Need help with licensing options or product information? Contact Sales

Scripted license management

Using REST API calls there is an option for scripted activation and deactivation for MetaDefender Core servers with **no Internet connection or if your infrastructure is using automation to create/destroy MetaDefender Core instances**. In this chapter the steps of these two scenarios are described.

Requirements

- an installed MetaDefender Core instance without Internet connection
- another computer that has Internet connection and can run your activation scripts
- a manual or automated way to transfer data between the two computers

Activation steps

1. In order to activate MetaDefender Core, deployment ID and activation key are needed to generate the license file.
    a. Activation key should be purchased from OPSWAT.
    b. MetaDefender Core deployment ID can be queried by using REST API (for details see GET Get Current License Information API page)

> **Info**
> Save this deployment ID in your system, you might need this ID when the instance is unavailable at the time of deactivation.

2. Activate license and get the license file using the following URL, where <quantity> is 1.

activation call bash

```
curl -X GET https://activation.dl.opswat.com/activation?key=<activation-key>&deployment=<deployment unique-ID>&quantity=<quantity>
```

If the activation is successfully, a license file is downloaded. Save this file.

HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	license file	Activation was successfully.
200 Activation failed	error: '<user conform error message>'	Failed activation
200 Invalid parameter	error: 'Could not activate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not activate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
200 Invalid parameter	error: 'Could not activate your product because the quantity you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key, quantity or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Info
In case of any activation issue, contact OPSWAT support for help

3. Upload license file to MetaDefender Core: The license file should be uploaded to the MetaDefender Core v4 to activate the product. For details see page: POST Upload license key API

Deactivation steps

1. For deactivation, following infomation is required:
    a. Activation key
    b. MetaDefender Core deployment ID: can be queried by using REST API (for details see GET Get Current License Information API page)
2. Deactivate license using the following URL:

deactivation call bash

```
curl -X GET https://activation.dl.opswat.com/deactivation?key=<activation-
key>&deployment=<deployment unique-ID>
```

Response HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	result: ok	Successful deactivation
200 No active license found	error: 'Could not found any active license with the given parameters'	The license has not been activated yet or it has been deactivated already.
200 Invalid parameter	error: 'Could not deactivate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not deactivate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Important notes

Warning

Product activation is tied to several hardware, operating system and software parameters. In case of one or more major hardware or software parameter change the product might turn into deactivated status. Operating system updates, other software updates should not affecting the activation status.

If the product is online activated and has live Internet connection then in case of deployment ID change the product reactivates itself. If the product doesn't have Internet connection then the administrator is responsible to reactivate the product.

Because of the above administrator should consider licensing restrictions of the offline deployments if using any virtualization and/or containerization technologies.

File Processing

MetaDefender Core provides REST API interface for file processing business.

There are several ways to process files with MetaDefender Core:

- Analyze Files via Web Interface
- Process Files via REST API

Product Upgrading

> **Info**
> Simply run installer file, details could be found at Installation

> **Info**
> For command line mode, upgrading (e.g. on **Linux**), prior to running the installer file, please make sure to **create the ignition file** with PostgreSQL server information included. Details:
>
> - Standalone DB mode: Command Line Installation (Windows and Linux)
> - Shared DB mode: Command Line Installation

Recommendation for MetaDefender Core upgrade

Our goal is to help you execute the product upgrade continuously, efficiently, and correctly.

PostgreSQL service upgrade

- **Bundled PostgreSQL Service**: If you are using the bundled PostgreSQL server included in the installer, the PostgreSQL service will be automatically upgraded to the newer version (if available). No additional action is required for this process.
- **Remote PostgreSQL Service**: For remote PostgreSQL servers, customers are responsible for managing and upgrading the service independently. Upgrading MetaDefender Core does not update the remote PostgreSQL service.

Before the upgrade

Prepare a proper disaster recovery plan (DRP) including some common practices:

- Backup, or take a snapshot of your current working MetaDefender Core server.
  - This is to help quickly rollback for any upgrade failure might cause later for any reason.
- Optionally backup your PostgreSQL database, especially when using remote shared database mode.
  - Reference: https://www.postgresql.org/docs/current/backup.html
  - Please do it after stopping all linked MetaDefender Core instances to avoid potential data collision.
- Lastly, if backing up entire server is not feasible, then at least export MetaDefender Core configuration package Import/Export configuration just in case you'll need to start off a new server with current MetaDefender Core configuration to keep the scanning service running without any impact.

On the PostgreSQL server, double check server resource to ensure sufficient RAM memory and disk space to be preserved for the upgrade process.

- 8GB RAM available at least during the upgrade.

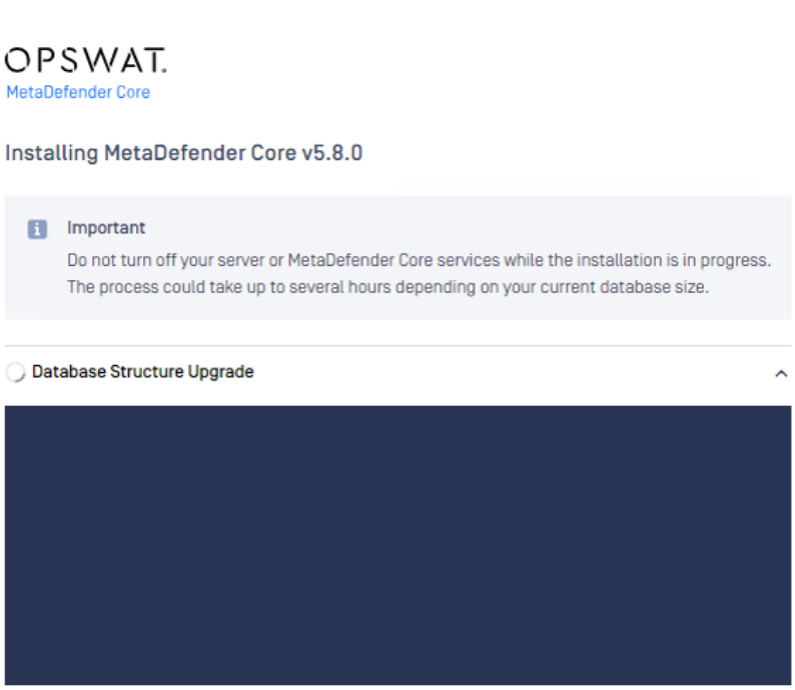
- Disk availability is greater than MetaDefender Core database size.

While doing the upgrade

Applicable for shared database deployment:

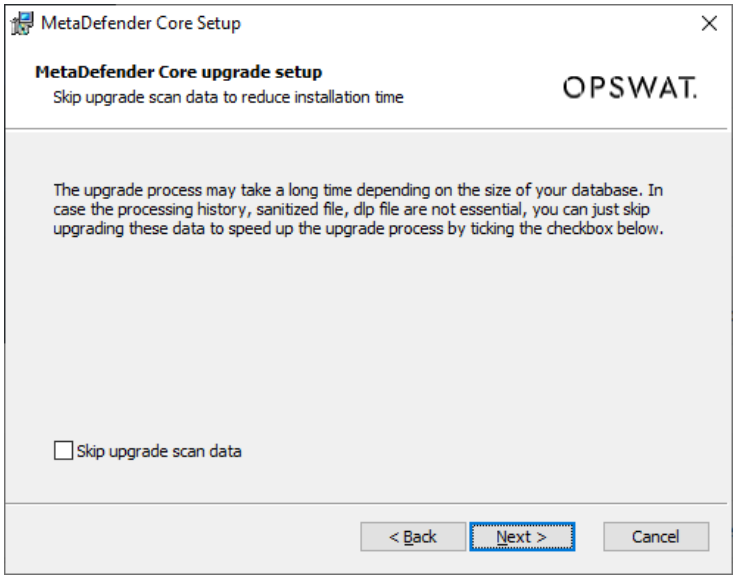
- Stop all MetaDefender Core instances' scanning service before performing the upgrade.
- Upgrade one (1) MetaDefender Core instance at a time.
- Proceed to upgrade MetaDefender Core instance where the bundle local PostgreSQL server is used for shared database setup.
- Otherwise is using remote PostgreSQL then the upgrade order does not matter.

For large size database, mostly for scan history, expecting long time to be spent on the upgrade process.



The product supports an option to skip data migration for scan history to expedite the process drastically and mitigate any failure risk. You can manually backup all data from your PostgreSQL server for offline usage if any.

Check "Skip upgrade scan data" to enable this feature while doing the upgrade.



or add following key in the ignition file:

none

```
[internal]
skip_upgrade_scan_data = 0
```

Troubleshooting

If the upgrade process ever runs into an error:

- Double check your network and services, applicable to remote PostgreSQL server.
- Optionally configure MetaDefender Core to skip migrating scan history.
  - Windows
    - Open Windows registry, and navigate to HKLM\Software\OPSWAT\Metascan
    - Add a new key `internal/skip_upgrade_scan_data` with value = 1
  - Linux
    - Open MetaDefender Core configuration file `/etc/ometascan/ometascan.conf`
    - Add new key into the file
      - [internal]
      - skip_upgrade_scan_data=1
  - Note: This key will be auto removed upon the upgrade finishes.
  - Then restart MetaDefender Core service to retry the upgrade.
    - Might retry this a few times until the upgrade is done.

Just in case the upgrade process still fails, please contact OPSWAT support with the support package - How to Create Support Package? This will help us analyze the root cause and figure out an optimal solution for your system failure.

Using external Load Balancer

Application Load Balancing (Layer 7)

This page provides information about recommended ways to use sessions with a Layer 7 load-balancer.

Most load-balancers have the ability to provide cookies for pinning subsequent traffic from a client to the appropriate server. This method is called sticky session, session persistence or session affinity. Using cookies efficiently requires the client to know when a cookie should be sent, should not be sent or should be deleted.

Sticky session load balancing

Each Metadefender Core instance has its own databases and application sessions that cannot be seen by other instances. Therefore, in order to get the related data/response to our queries we should ask the appropriate Metadefender Core v5 server. On the other hand, to keep the advantages of the used load balancing method, cookies should not be sent if it is not necessary.

Single file scanning

Step	Stage	Task	Cookie usage
1.	Sending file	Initiate processing a file on the client side. Send file through the load-balancer. (See POST File Analysis API)	Cookie should not be sent
2.	Sending file	Save the cookie and the data_id you got from the load-balancer	Save cookie
3.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Send cookie saved in step 2
4.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Send cookie saved in step 2

Batch scanning

Step	Stage	Task	Cookie usage
1.	Open batch	Initiate processing file(s) in batch. Request a batch ID through the load-balancer. (See POST Initiate Batch API)	Cookie should not be sent
2.	Open batch	Save the cookie and the batch_id you got from a Core server through the load-balancer.	Save cookie
3.	Sending files	Send file through the load-balancer. (See POST File Analysis (using batch header) API)	Send cookie saved in step 2
4.	Sending files	Save the data_id you got from the load-balancer.	-
	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Send cookie saved in step 2
5.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
6.	Getting batch status	See GET Batch Status API	Send cookie saved in step 2
7.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Send cookie saved in step 2
8.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API	Send cookie saved in step 2

If it does not matter which upstream server responds, the queries should be sent without cookie.

> **Info**
> It is recommended not to send cookies when it's not necessary to allow load-balancer to use its own method to share the load between Metadefender Core v5 servers.

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration through load-balancer

OPSWAT products that support HTTP load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	4.3.4	-
OPSWAT Client	Windows: 7.6.247.0 Mac: 10.4.243.0	-
MetaDefender Email Security	4.3.0	-
MetaDefender ICAP Server	4.3.0	-
MetaDefender Vault	1.3.0	-
MetaDefender for Secure Storage	2.0.7	-

DNS Load Balancing

Using this method is logically similar to Layer 7 load-balancing.

Briefly how it works

Client uses a domain name to send a query to a server. Client's DNS server has more "A" records for that name with different IPs. When a client resolves the server's domain name DNS server randomly chooses an IP for that name to send back. When a session is used on the application layer, client should know the IP address of the Core server that handles that specific session. In every other case, client should resolve the domain name with DNS query to let requests to be balanced between Metadefender Core servers.

Single file scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Sending file	Initiate processing a file on the client side. (See POST File Analysis API)	Use IP saved in step 2
4.	Sending file	Save the data_id got from the Core server	-
5.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Use IP saved in step 2
6.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Use IP saved in step 2

Batch scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Open batch	Initiate processing file(s) in batch. Request a batch ID. (See POST Initiate Batch API)	Use IP saved in step 2
4.	Sending files	Send file to the specific Core server with the basch ID saved in step 3. (See POST File Analysis (using batch header) API)	Use IP saved in step 2
5.	Sending files	Save the data_id you got from the Core server.	-
6.	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Use IP saved in step 2
7.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
8.	Getting batch status	See GET Batch Status API	Use IP saved in step 2
9.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Use IP saved in step 2
10.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API).	Use IP saved in step 2

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration via DNS load-balancing

OPSWAT products that support DNS load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	does not support yet	-
MetaDefender Client	does not support yet	-
MetaDefender Email Security	does not support yet	-
MetaDefender ICAP Server	does not support yet	-
MetaDefender Vault	does not support yet	-

SSL connection for PostgreSQL communication

> **Note**
> The SSL connection configuration is possible thanks to the support made by PostgreSQL. For reference: https://www.postgresql.org/docs/14/ssl-tcp.html
>
> MetaDefender Core is tested against this mode on particular version 4.21.1

Step 1: Generate SSL certificate for PostgreSQL server

> **Note**
> Following instructions are for self-sign certificate. For production environment, making sure to use the certificate signed by your trusted CA.

1) Generate a private key

bash

```
openssl genrsa -des3 -out server.key 1024
```

Sample output:

bash

```
Generating RSA private key, 1024 bit long modulus ........++++++
...............++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

Type your desired password / pass phrase to encrypt the private key

2) Remove the pass phrase to automatically start up the PostgreSQL server

bash

```
openssl rsa -in server.key -out server.key
```

> **Note**
> On Linux, make sure to set permission on the server.key file

bash

```
chmod og-rwx server.key
```

3) Create a self-signed certificate

bash

```
openssl req -new -key server.key -days 3650 -out server.crt -x509
```

You will be prompted to enter detailed information which is incorporated into your self-signed certificate request.

none

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:CA
Locality Name (eg, city) [Default City]:San Francisco
Organization Name (eg, company) [Default Company Ltd]:OPSWAT
Unit Name (eg, section) []:OPSWAT
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:dummy_email@opswat.com
```

For self-signed certificate, use the server ceriticate as the trusted root certificate:

bash

```
cp server.crt root.crt
```

Copy **server.key**, **server.crt**, **root.crt** to PostgreSQL data folder. For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data**
- Linux: **/var/lib/ometascan/pg_data/**

Step 2: Configure PostgreSQL server for SSL authentication connection

1) Create a custom config file (e.g. ssl.conf) for PostgreSQL.

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\postgres\ssl.conf**
- Linux: **/var/lib/ometascan/postgres/ssl.conf**

2) Content of ssl.conf:

none

```
ssl = on
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
```

3) Modify pg_hba.conf

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data\pg_hba.conf**
- Linux: **/var/lib/ometascan/pg_data/pg_hba.conf**

Add following (modify **::1/128** if needed to match with your environment requirement)

none

```
hostssl all all      ::1/128      scram-sha-256
```

After the modify, the content of pg_hba.conf should be like this (example):

> **Note**
> In a Linux environment, a local connection is required to upgrade from version 5.5.1 or earlier to version 5.6.0 or later.

none

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local         DATABASE  USER  METHOD  [OPTIONS]
# host          DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl     DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostgssenc    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnogssenc  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
local    all all                    scram-sha-256
hostssl all all      ::1/128      scram-sha-256

#More content.......
```

4) Reload the services

- Standalone DB mode: Restart **ometascan** service (MetaDefender Core service)
- Shared DB mode: Restart **ometascan-pg** / **ometascan-postgresql** service (MetaDefender Core PostgreSQL service) and then **ometascan** service (MetaDefender Core service)

Test your SSL database connection:

bash

```
psql.exe "sslmode=require dbname=postgres user=postgres host=localhost port=5432"
Password for user postgres:
psql (12.7)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
Type "help" for help.
```

This message indicate that you are now using SSL connection:

bash

```
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
```

Performance and Load Estimation

> **Disclaimer**
> These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Some factors that affect performance

- MetaDefender Core version
- MetaDefender Core engine package and configuration
  - set of engines (which and how many)
  - product configuration (e.g., thread pool size)
- System environment
  - server profile (CPU, RAM, hard disk)
  - client application location - remote or local
  - system caching and engine level caching
- Dataset
  - encrypted or decrypted
  - file types
    - different file types (e.g., document, image, executable)
    - archive file or compound document format files
  - file size
  - bad or unknown (assume to be clean)
- Performance tool

How test results are calculated

Performance (mainly scanning speed) is measured by throughput rather than unit speed. For example, if it takes 10 seconds to process 1 file, and it also takes 10 seconds to process 10 files, then performance is quantified as 1 second per file, rather than 10 seconds.

- total time / total number of files processed: 10 seconds / 10 files = 1 second / file.

Standalone DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.2.416.0

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

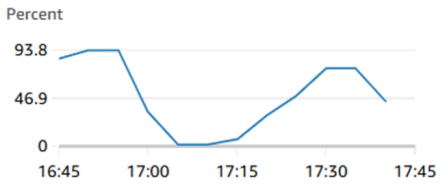
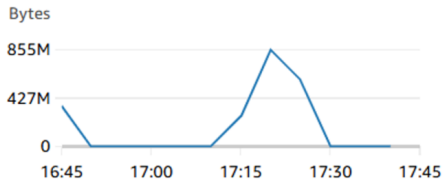
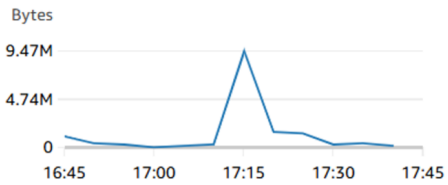
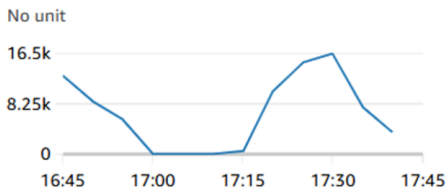
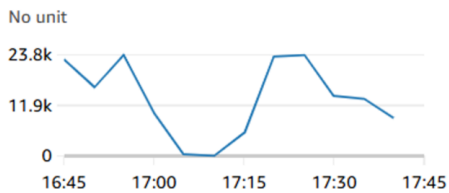
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV)	24.2	47.2 / 80	37.3 / 41.1		315,310	0.0114
Deep CDR	19.9	48.9 / 81.8	32.6 / 37		383,442	0.0094
Proactive DLP	17.1	36.8 / 58.2	30.4 / 33.6		446,228	0.0081
Vulnerability	10.1	59.2 / 85	29.3 / 33.3		755,495	0.0048

MetaDefender Core with common engine packages:

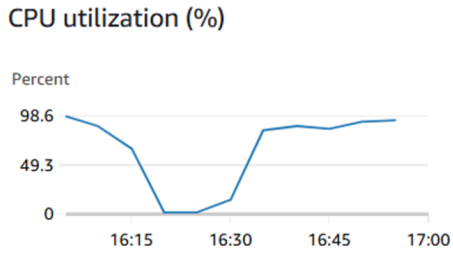
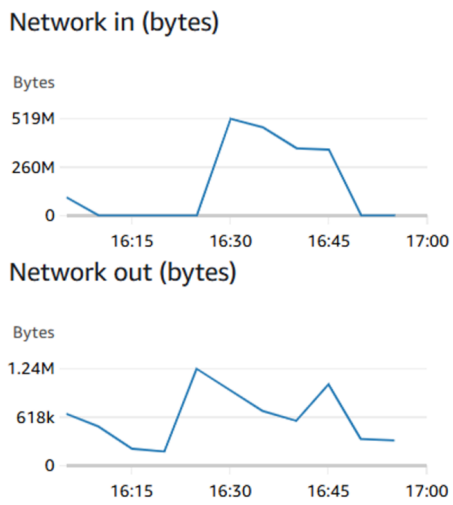
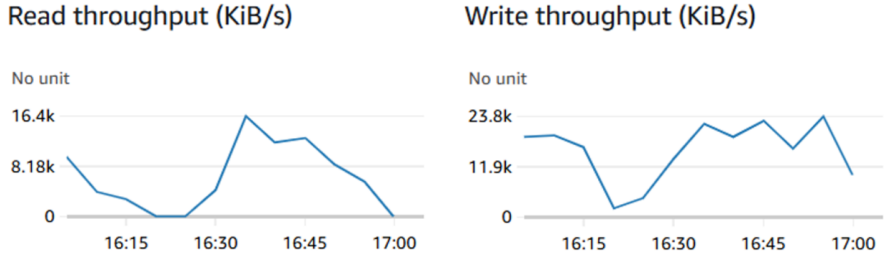
Use case	Scan duration	Avg./Max CPU usage	Avg./Max RAM usage	Avg./Max Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV) + Deep CDR	30.8	68.1 / 88	40.1 / 46.8		247,744	0.0145
Metascan (8 AV) + Deep CDR + Proactive DLP	31.2	74.2 / 98.6	41.5 / 51.1		244,567.3	0.0147
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	31.8	73.5 / 98	40.6 / 51.5		239,953	0.015

System resource utilization charts

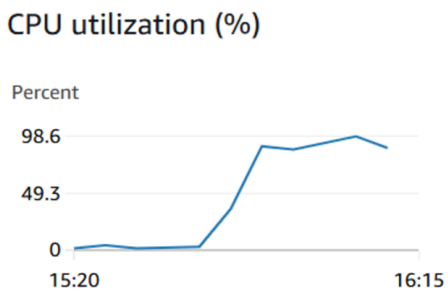
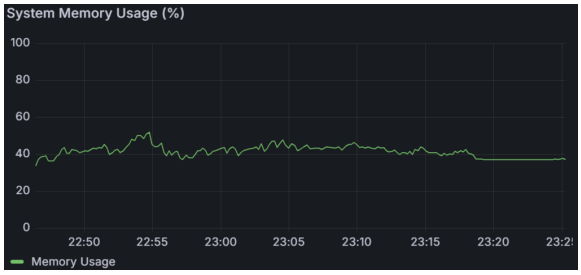
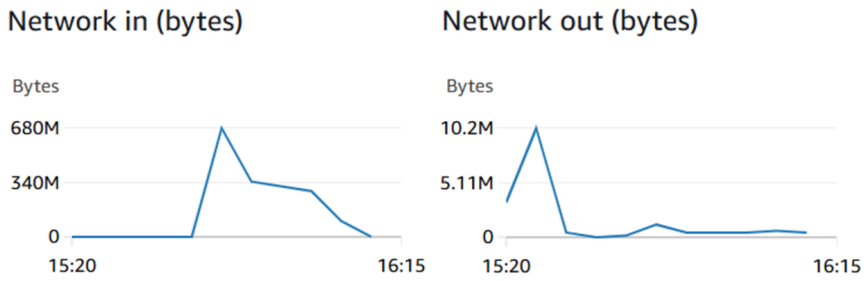
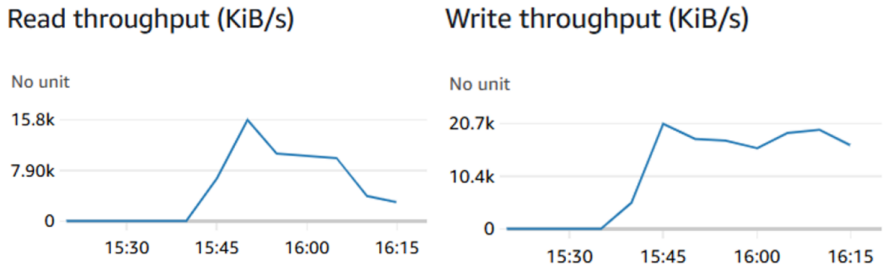
MetaDefender Core with Metascan (8 AV engines):

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

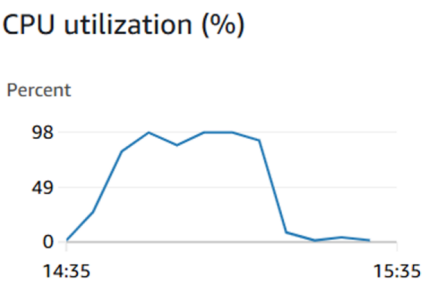
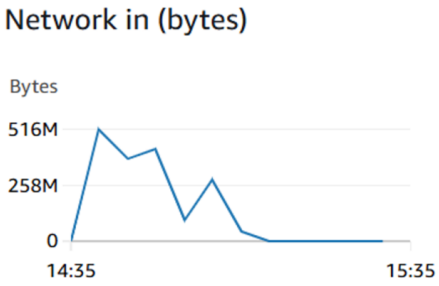
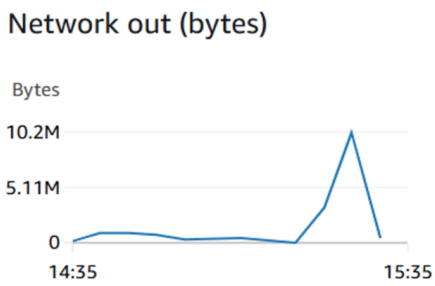
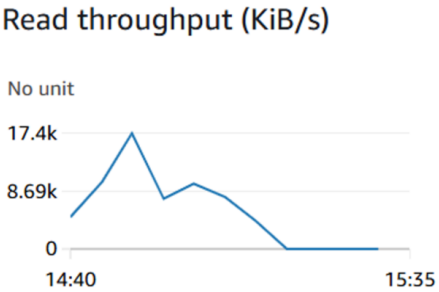
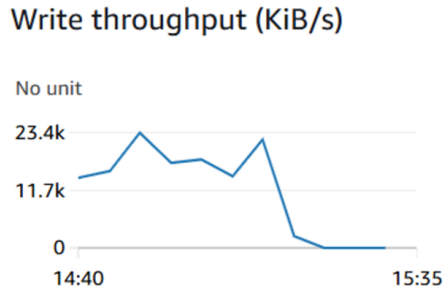
MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

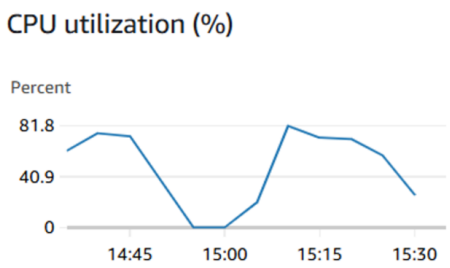
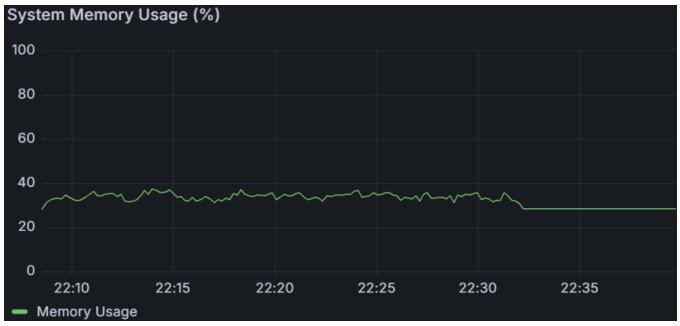
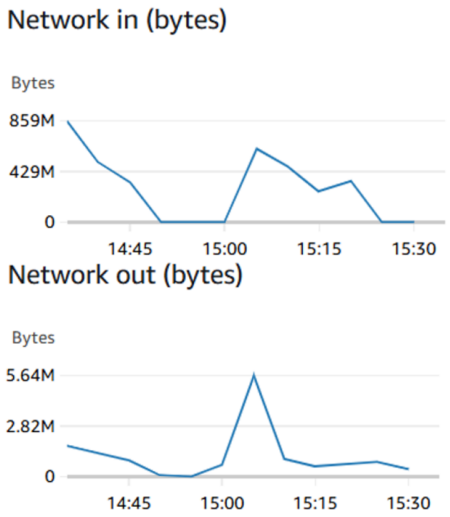
MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

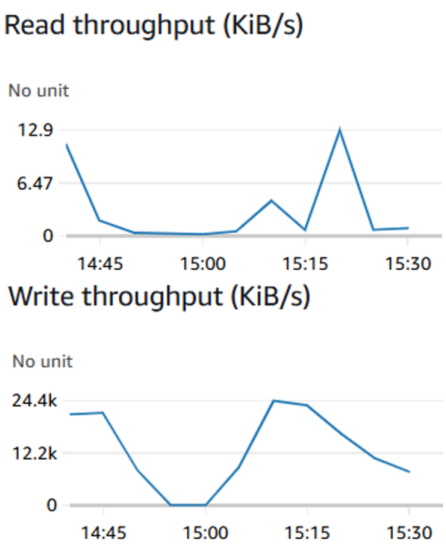
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

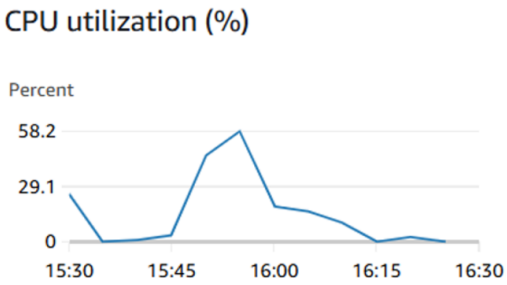
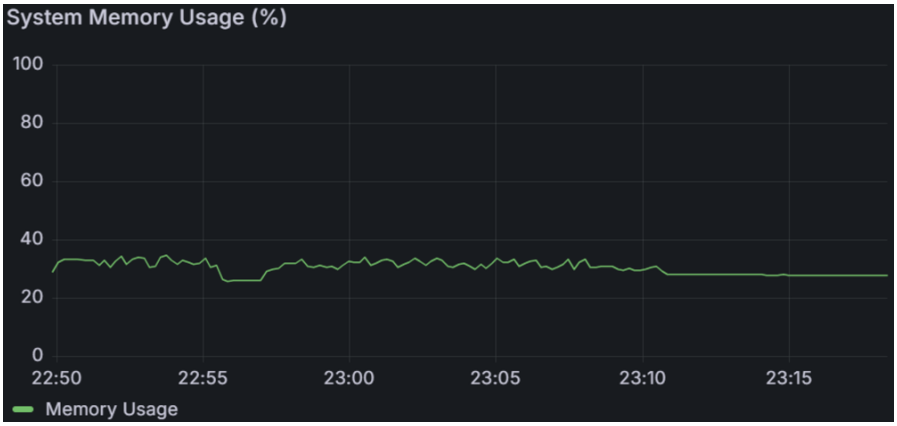
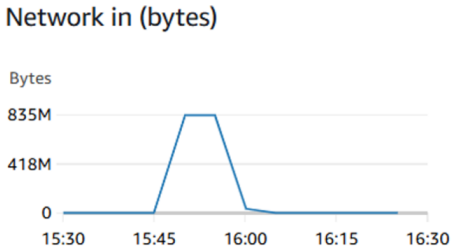
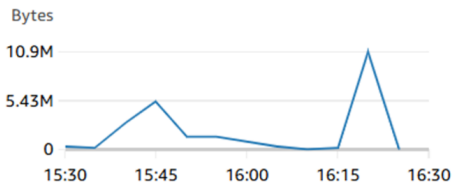
	Utilization charts
CPU usage	
Memory usage	
Network usage	 
Disk usage	 

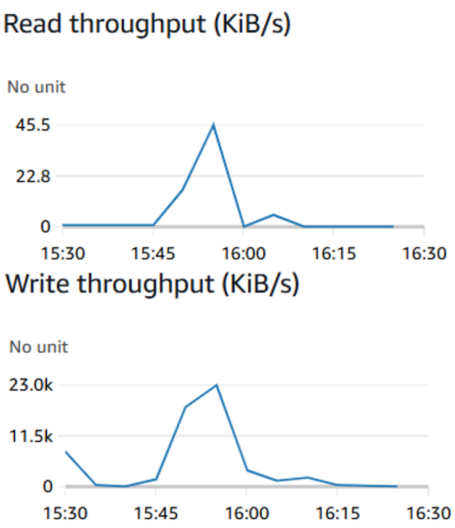
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

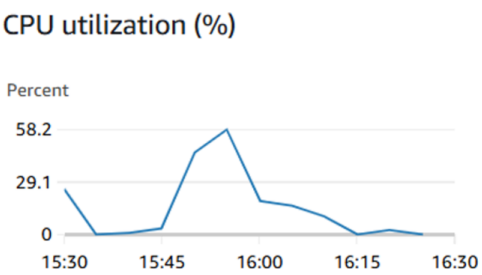
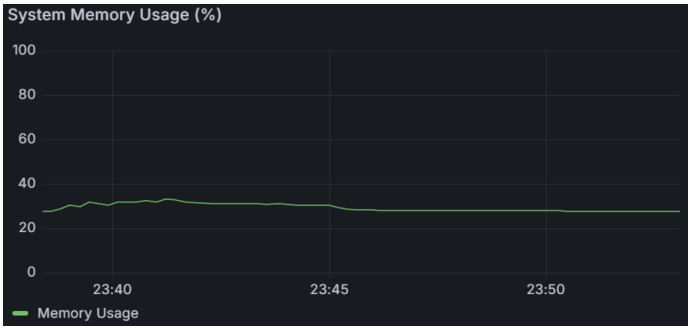
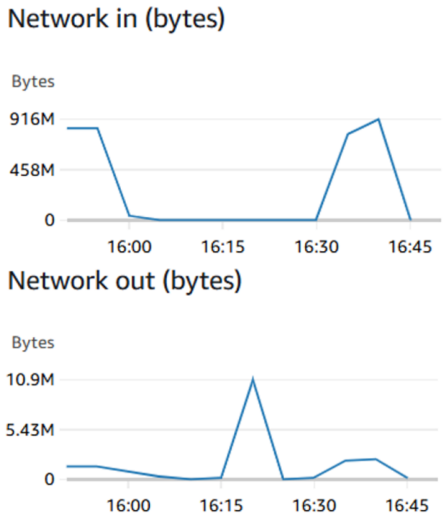
	Utilization charts
Disk usage	

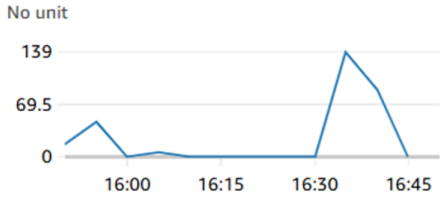
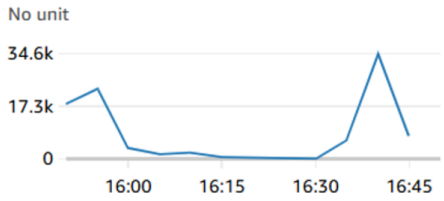
MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	 

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability:

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

Standalone DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Ubuntu 22.04	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.57-236

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

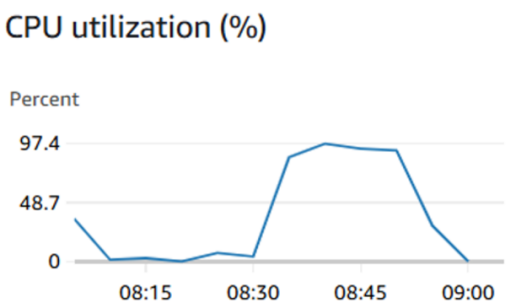
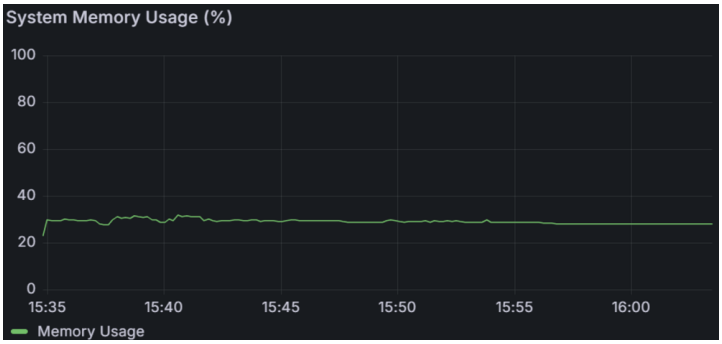
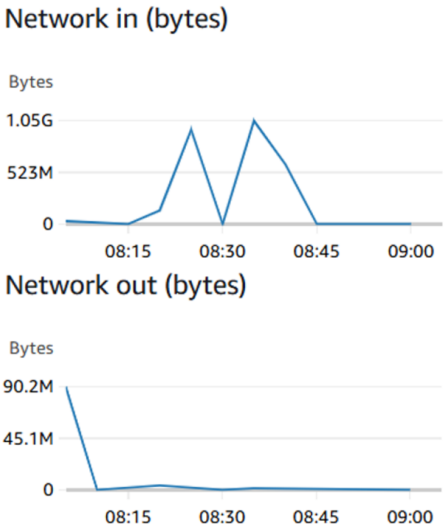
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV)	30.2	69 / 98	29 / 32		252,666	0.0142
Deep CDR	19.4	48 / 62	36 / 49		393,325	0.009
Proactive DLP	16.5	34 / 39	32 / 48		462,455	0.008
Vulnerability	12	50 / 65	46 / 46		635,875	0.006

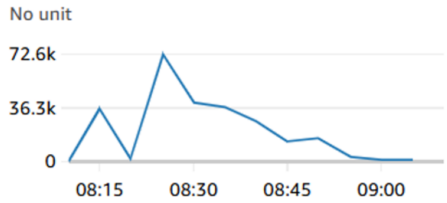
MetaDefender Core with common engine packages:

Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV) + Deep CDR	39.8	78 / 99	62 / 73		191,721	0.0188
Metascan (10 AV) + Deep CDR + Proactive DLP	43.5	61 / 84	47 / 67		175,414	0.0205
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	44.8	62 / 88	69 / 83		170,324	0.0211

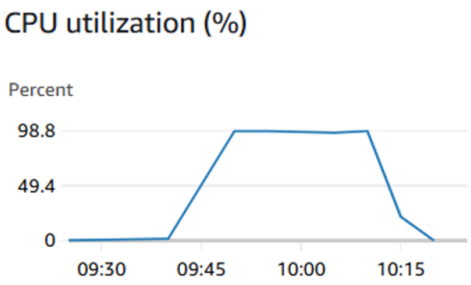
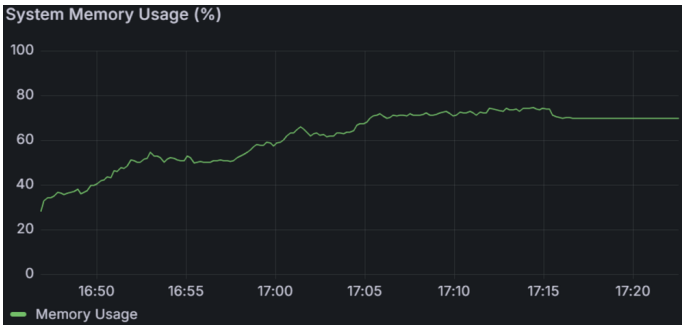
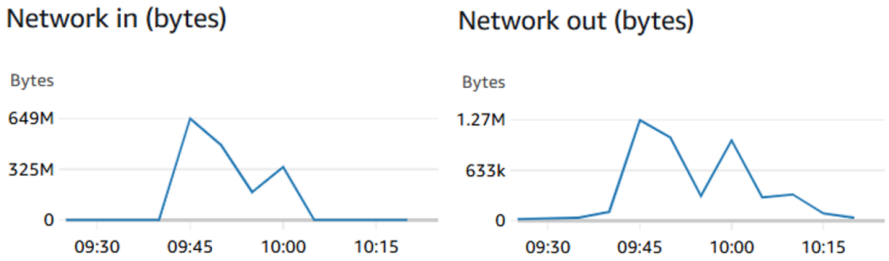
System resource utilization charts

MetaDefender Core with Metascan (10 AV engines) :

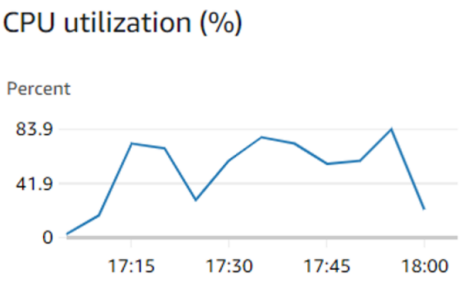
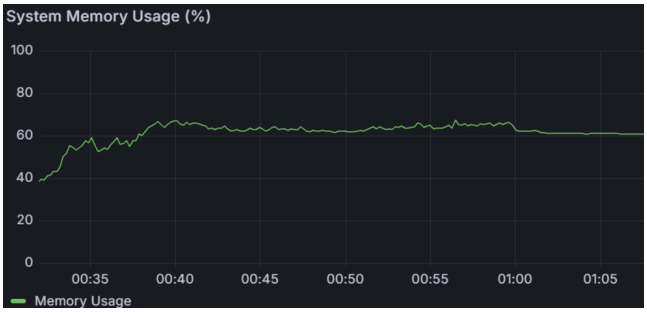
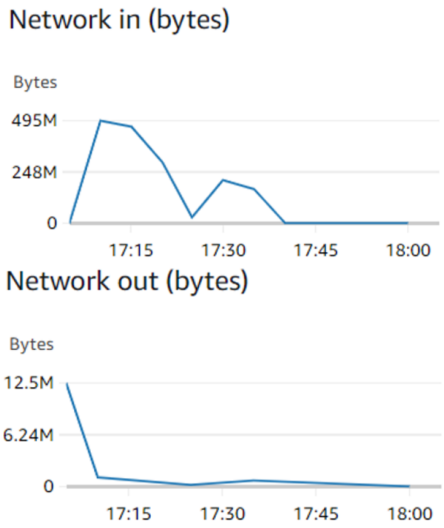
	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

MetaDefender Core with Metascan (10 AV engines) + Deep CDR :

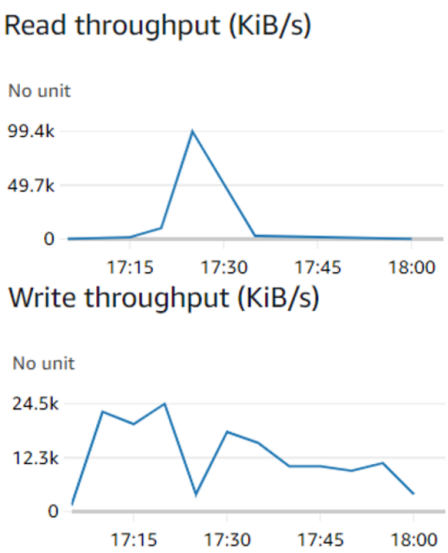
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP :

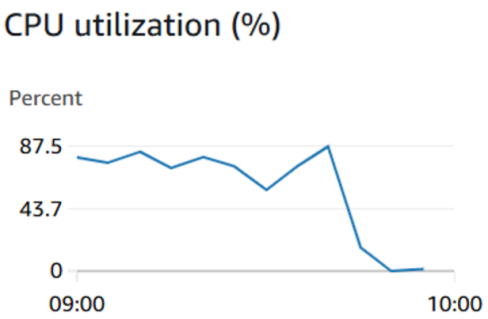
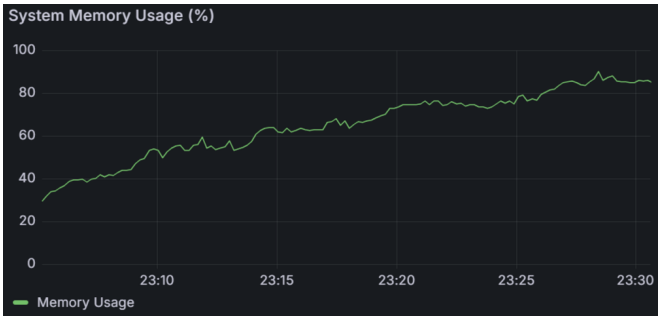
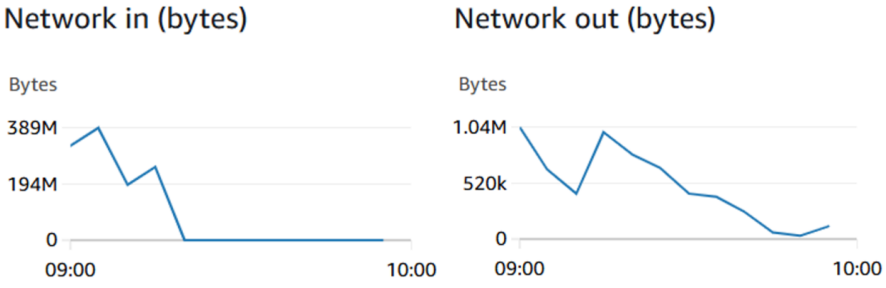
	Utilization charts
CPU usage	
Memory usage	
Network usage	

Utilization charts

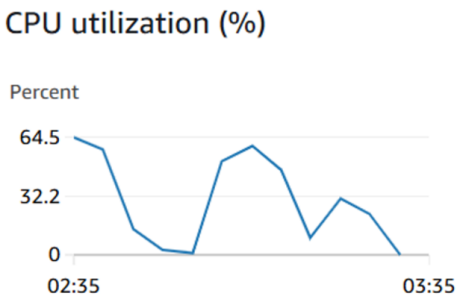
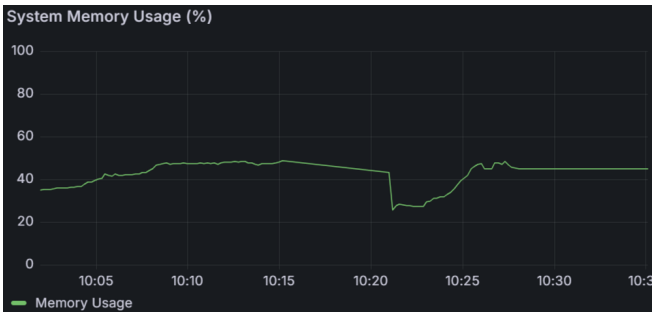
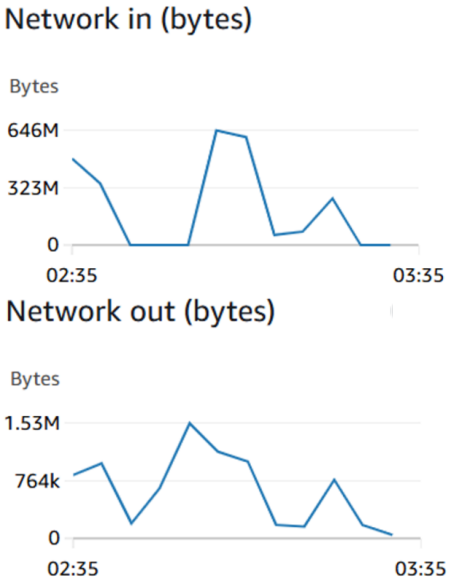
Disk usage

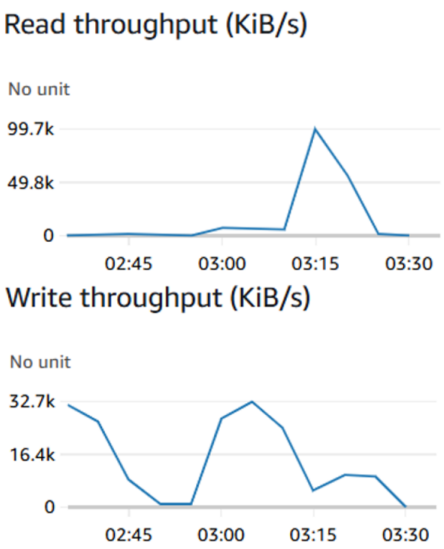


MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP + Vulnerability :

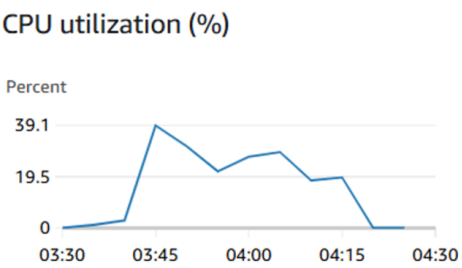
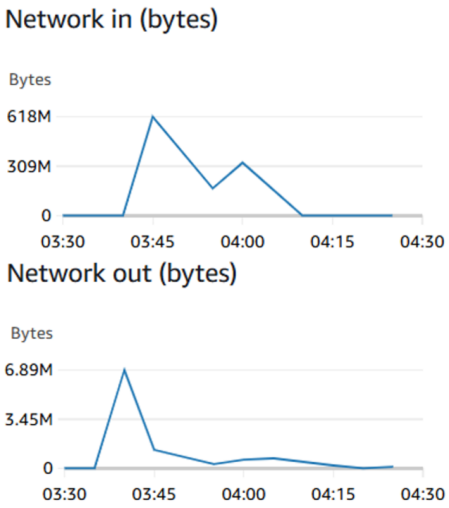
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

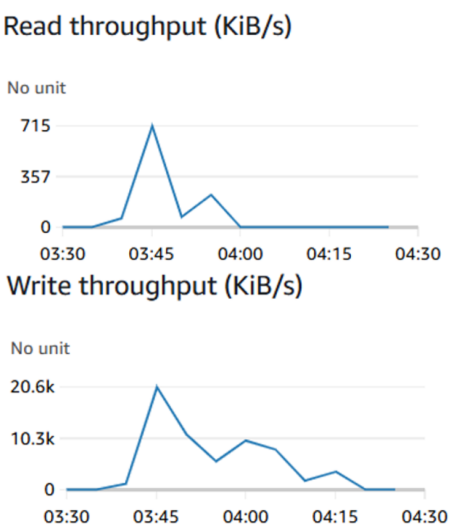
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

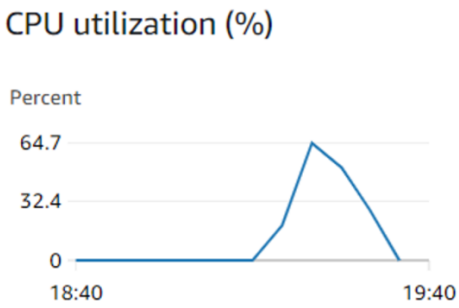
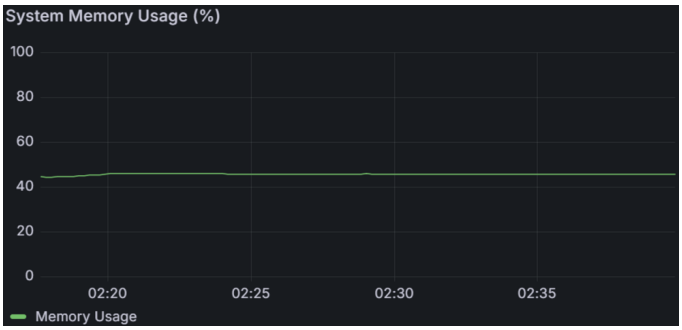
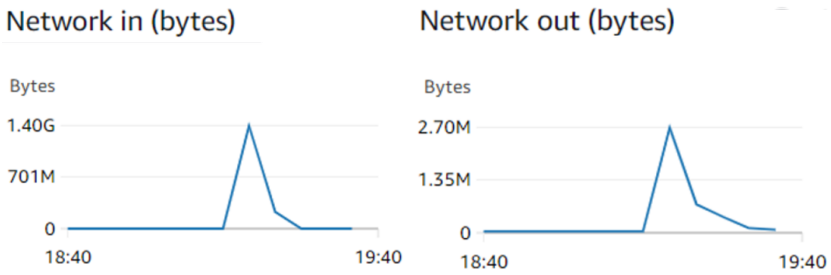
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability :

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

Shared DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

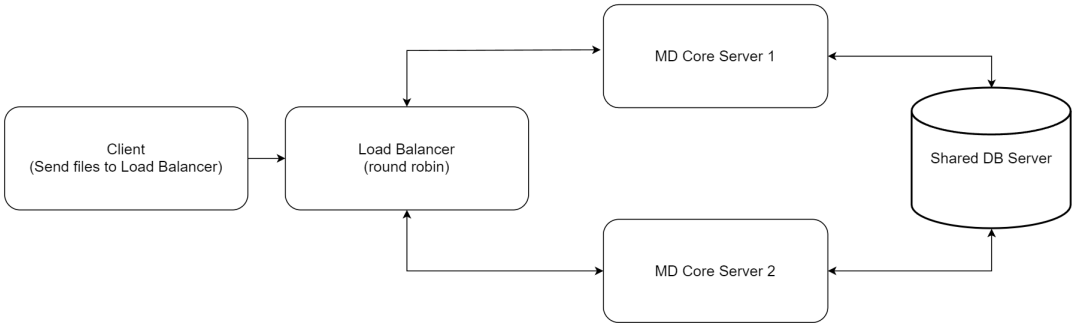
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV)	11.9	37.0	41.9	30.1	30.4	Receive: 6441.1 Send: 769.6	Receive: 6200.2 Send: 693.7	640,821.2	0.0
Deep CDR	10.5	60.1	66.7	30.3	32.2	Receive: 6379.8 Send: 594.8	Receive: 7146.1 Send: 609.6	729,368.5	0.0
Proactive DLP	7.8	38.7	37.5	29.7	31.0	Receive: 8842.6 Send: 791.8	Receive: 8635.6 Send: 729.1	965,455.7	0.0
Vulnerability	4.4	52.2	53.0	26.0	26.9	Receive: 16,215.5 Send: 1006.8	Receive: 15,785.7 Send: 997.2	1714,449.3	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV) + Deep CDR	15.4	72.4	75	34.3	35.4	Receive: 5436.6 Send: 611.3	Receive: 5589.3 Send: 607.2	495,697.4	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP	16.9	77.2	80.2	36.4	37.9	Receive: 5254 Send: 585.4	Receive: 5041.3 Send: 576.1	450,657.6	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	18.0	72.1	80.4	38.6	38.0	Receive: 4504.3 Send: 574.1	Receive: 4649.5 Send: 571.1	423,010	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

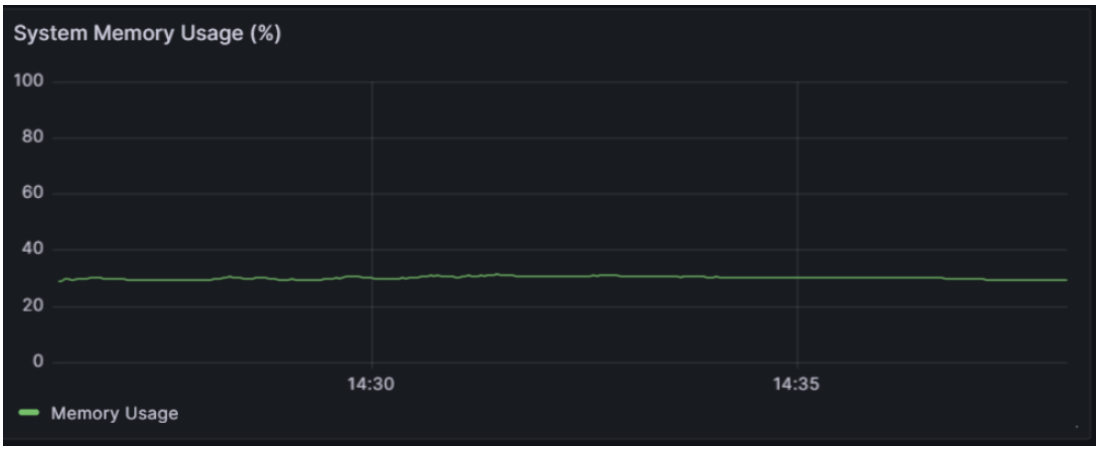
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2

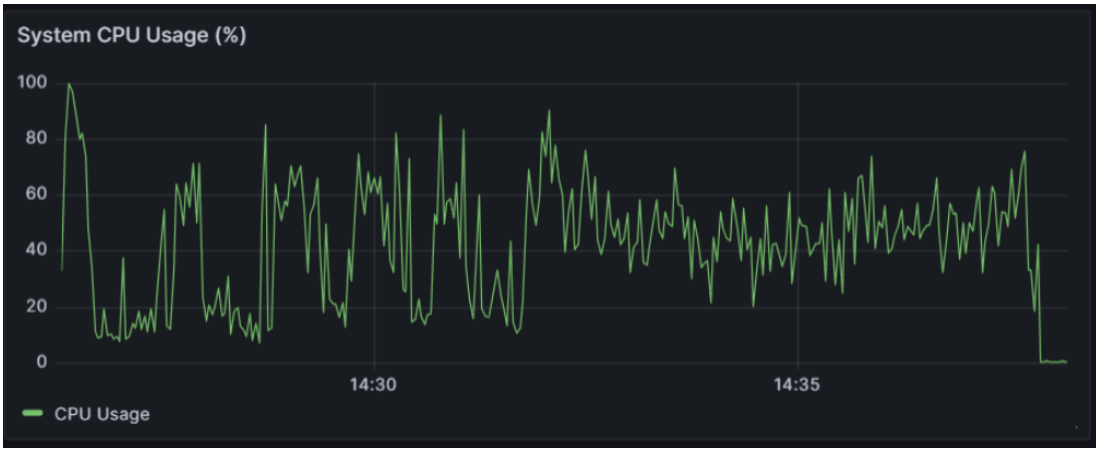


CPU usage

MD Core #1

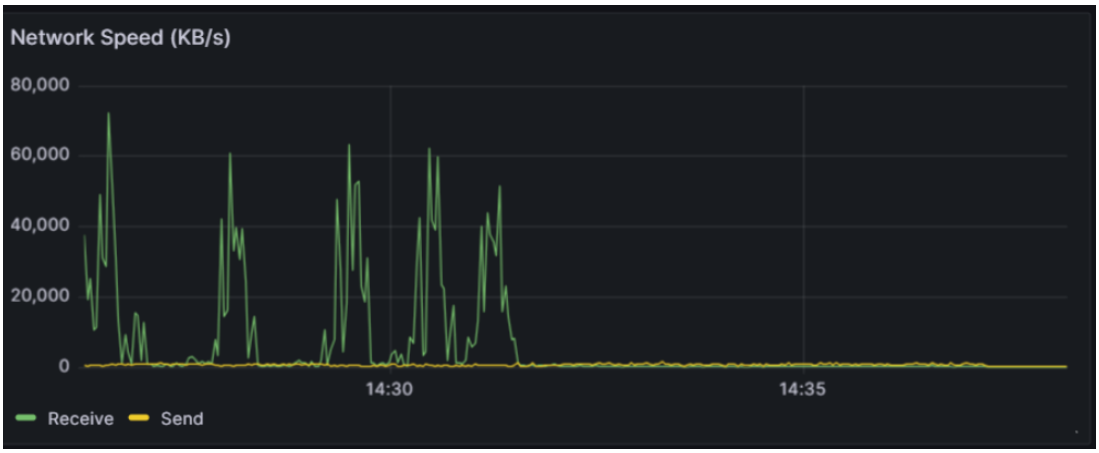


MD Core #2

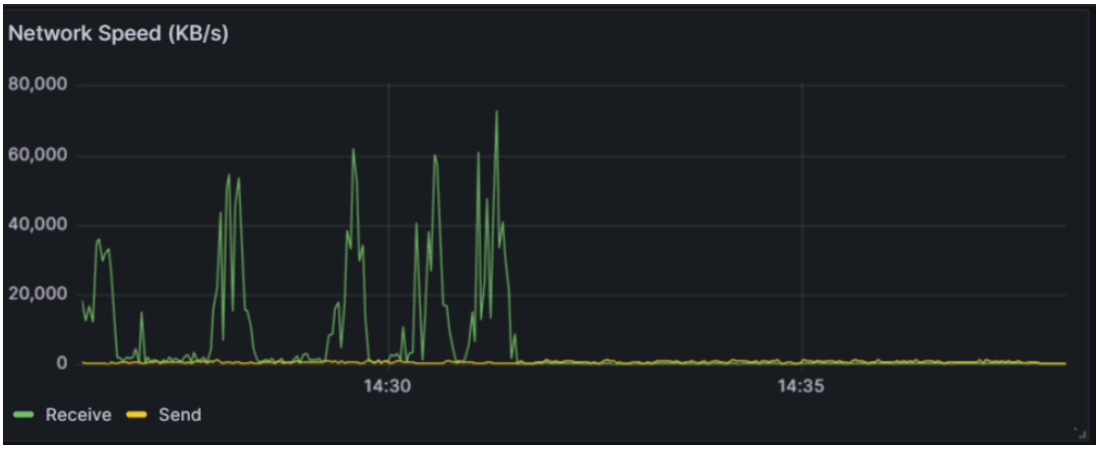


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2

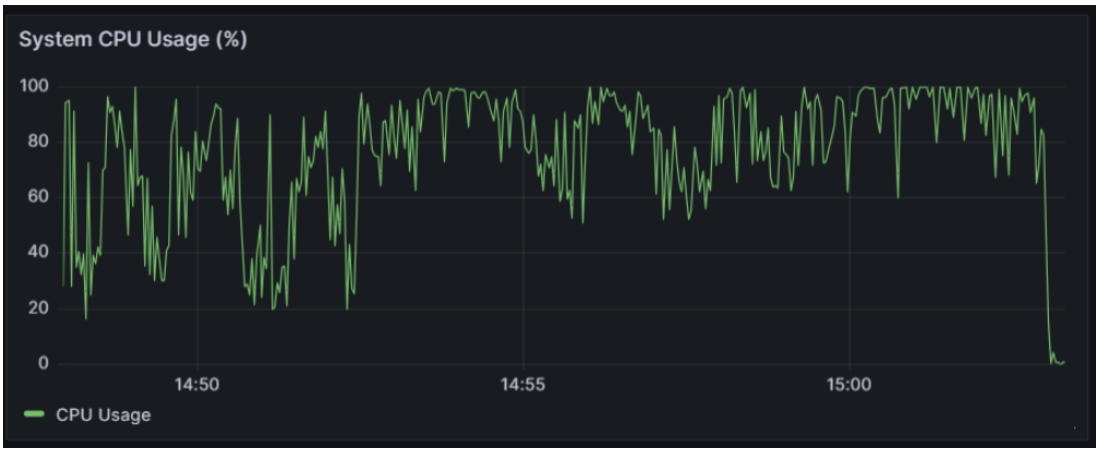


CPU usage

MD Core #1

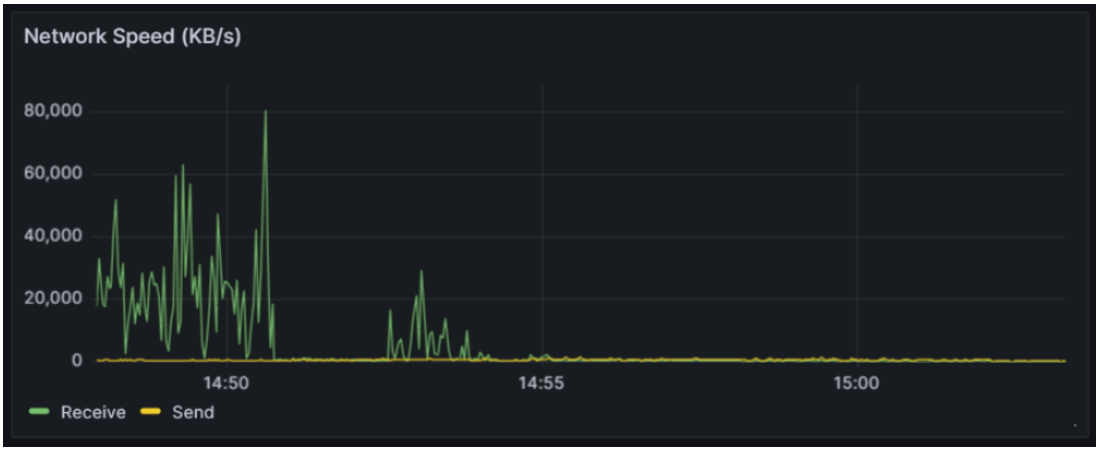


MD Core #2



Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

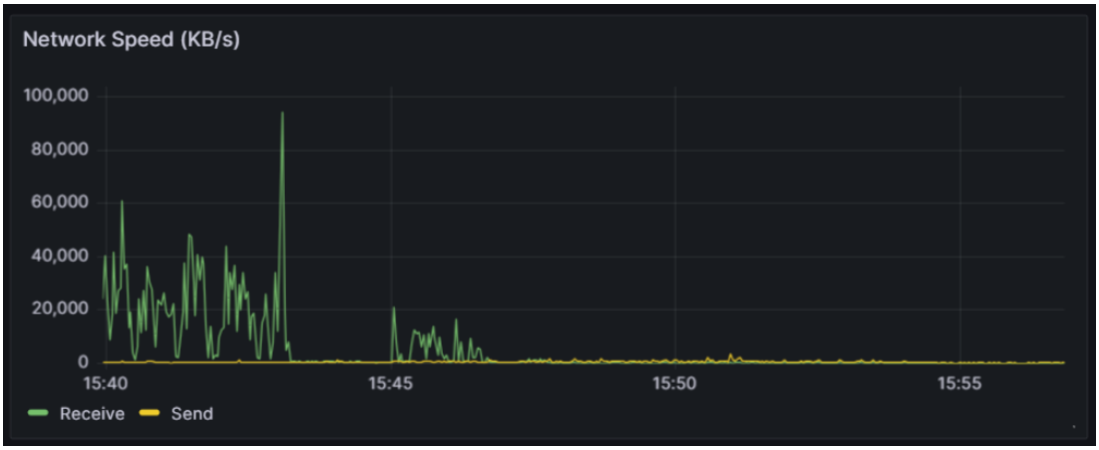


MD Core #2

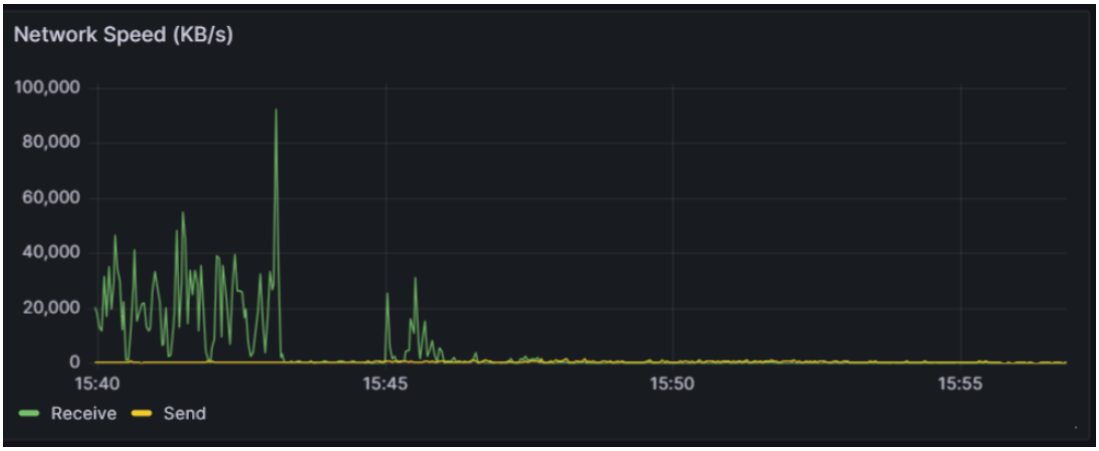


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1

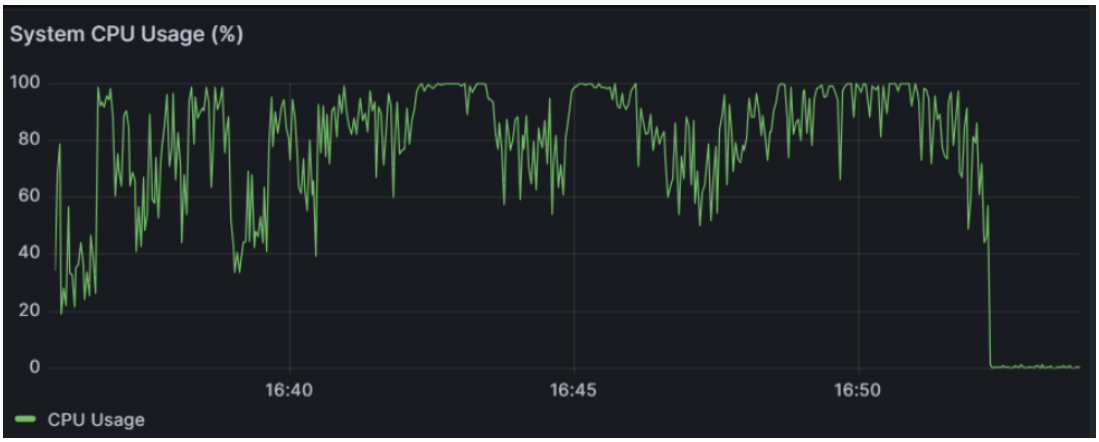


MD Core #2



CPU usage

MD Core #1

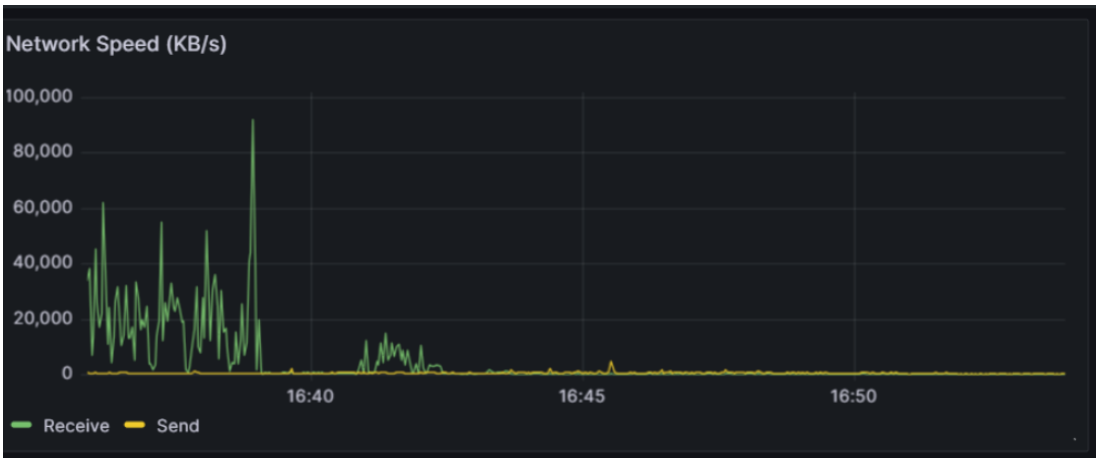


MD Core #2

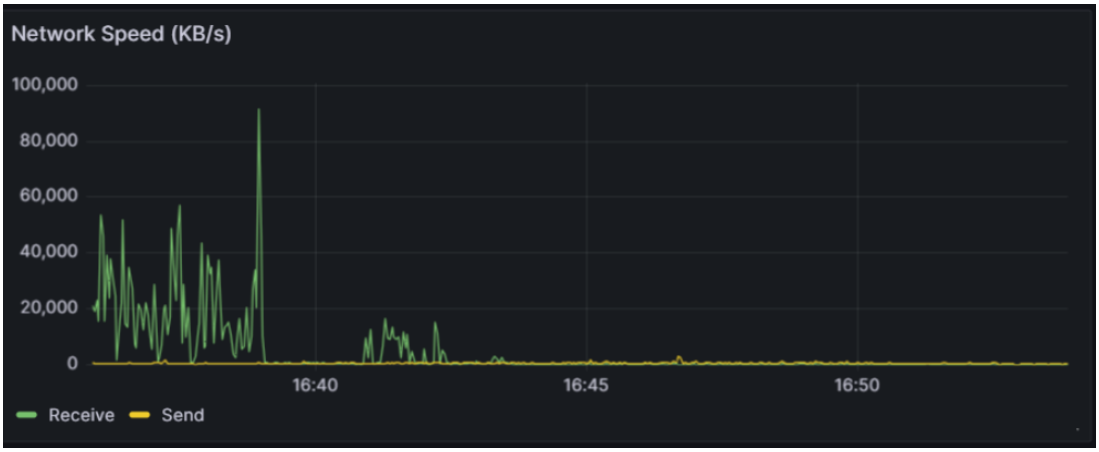


Network usage

MD Core #1



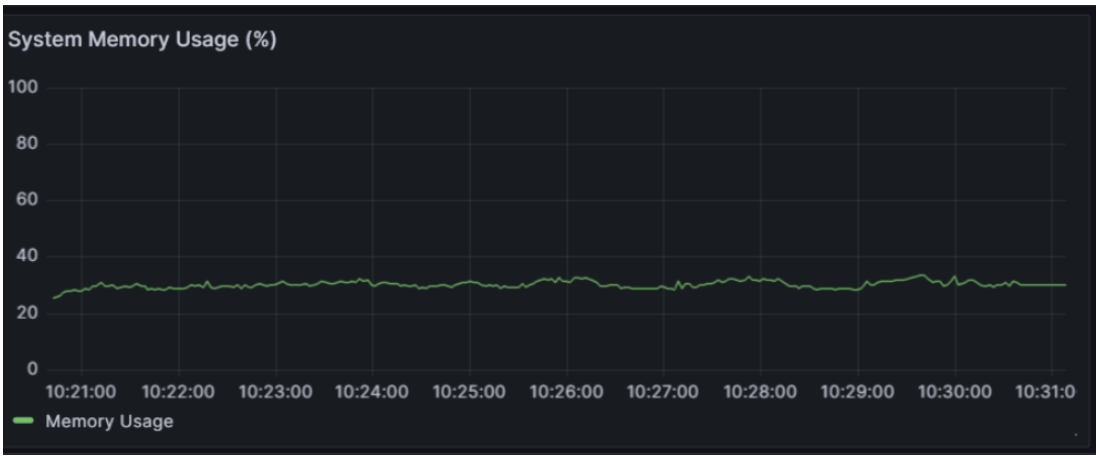
MD Core #2



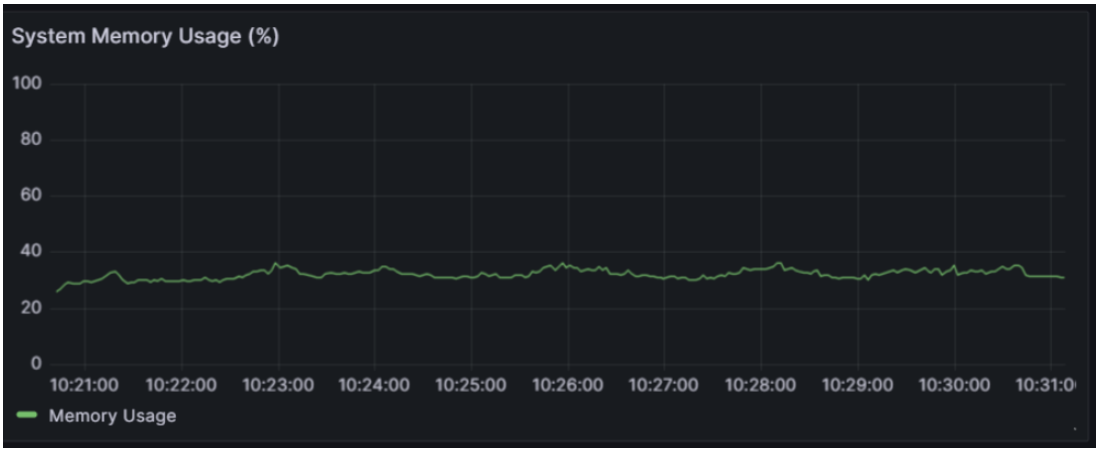
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

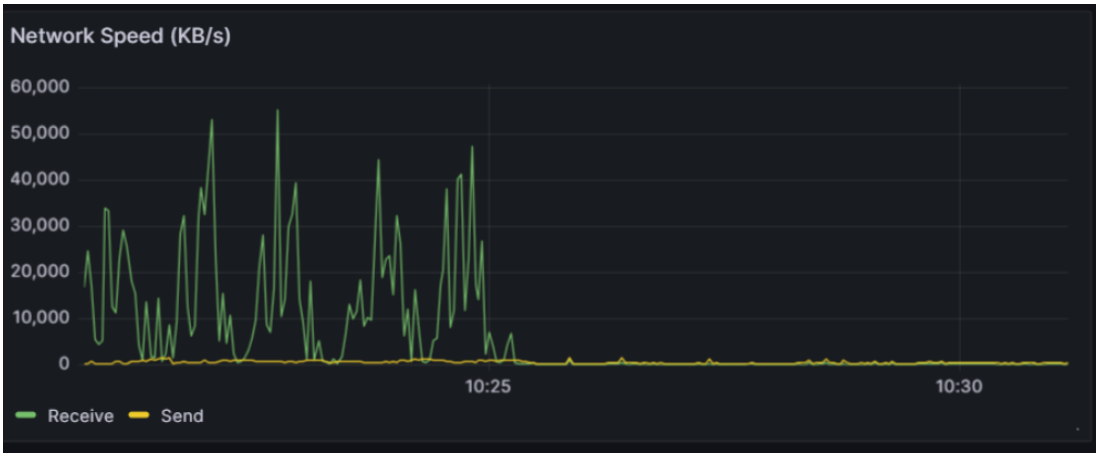


MD Core #2

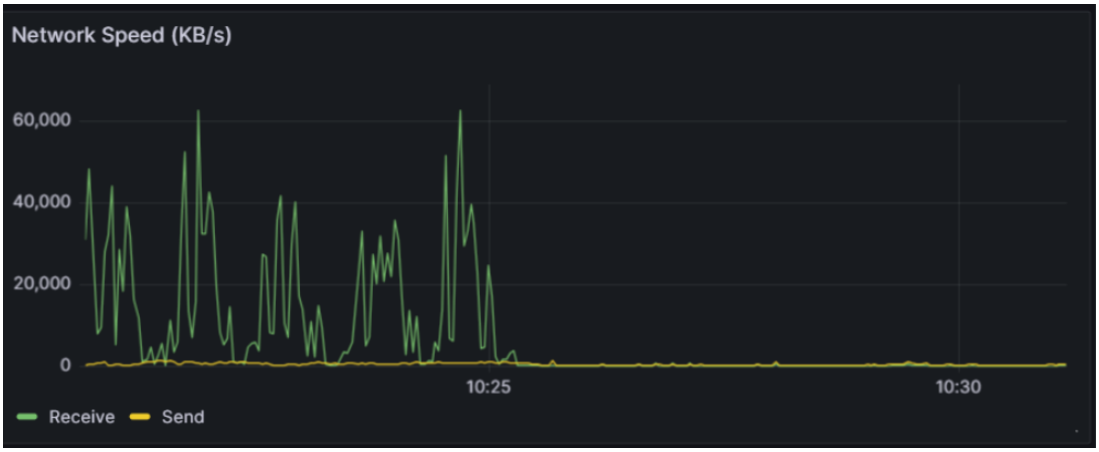


Network usage

MD Core #1



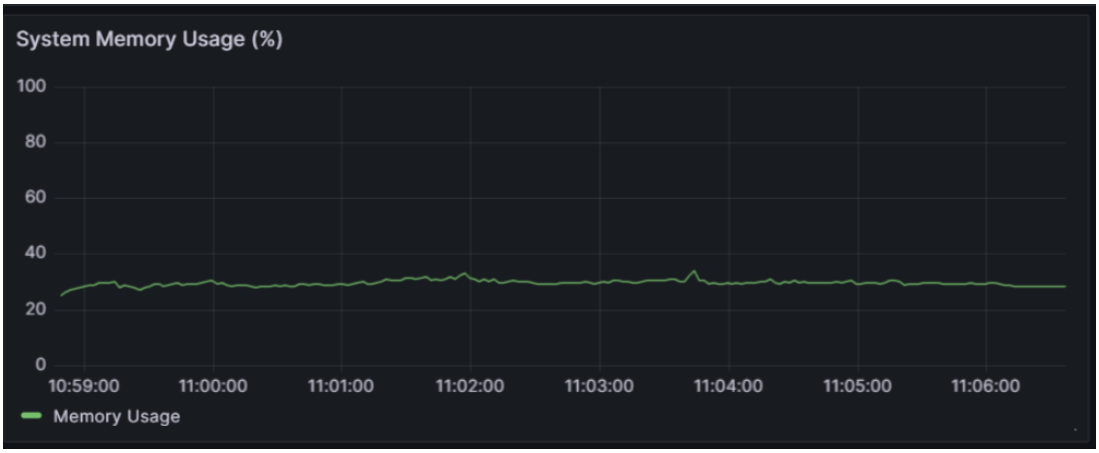
MD Core #2



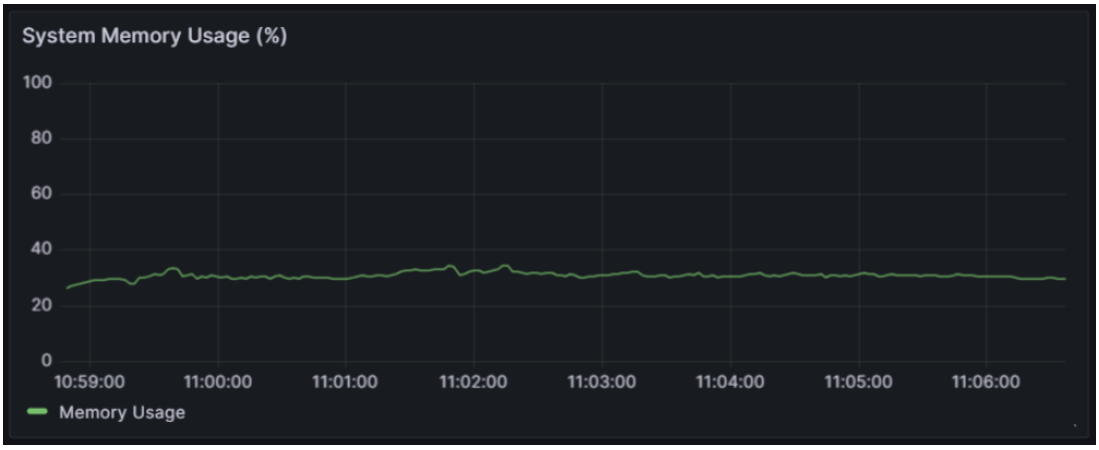
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

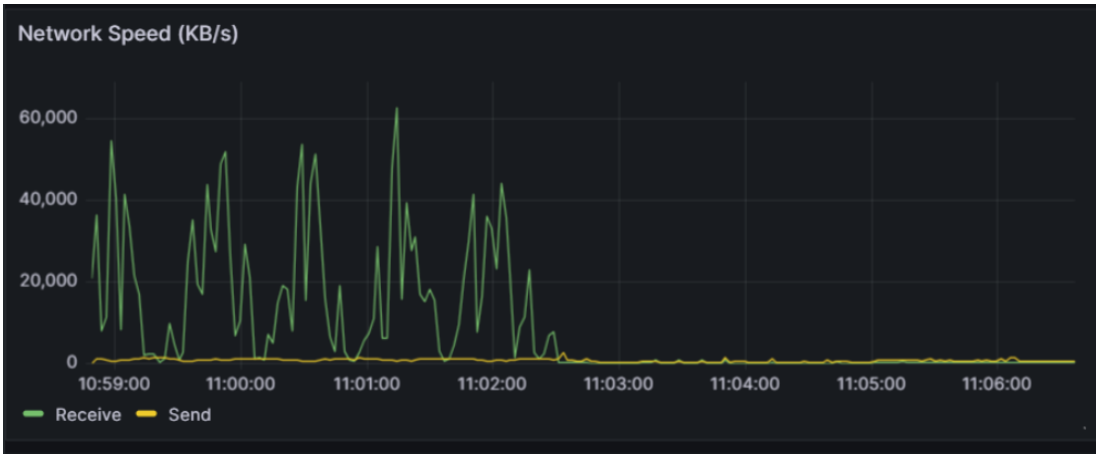


MD Core #2

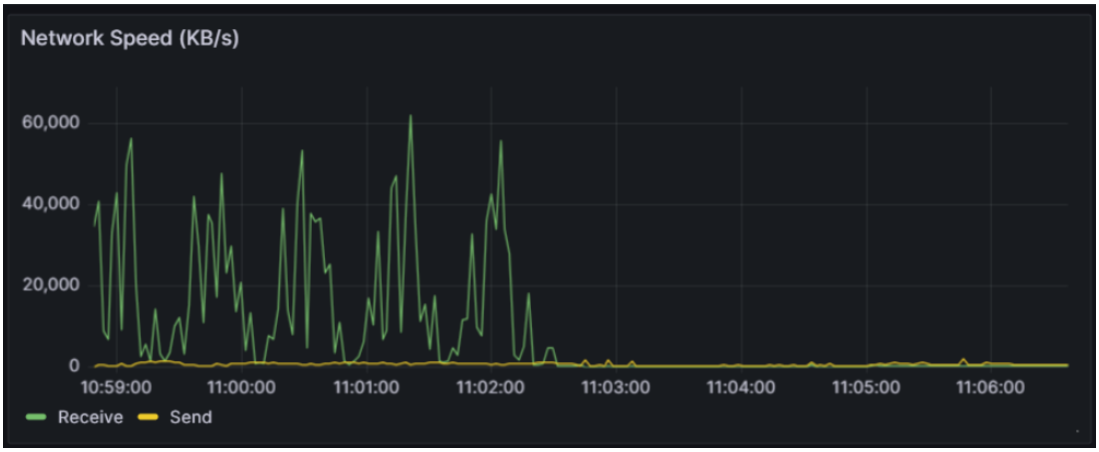


Network usage

MD Core #1



MD Core #2



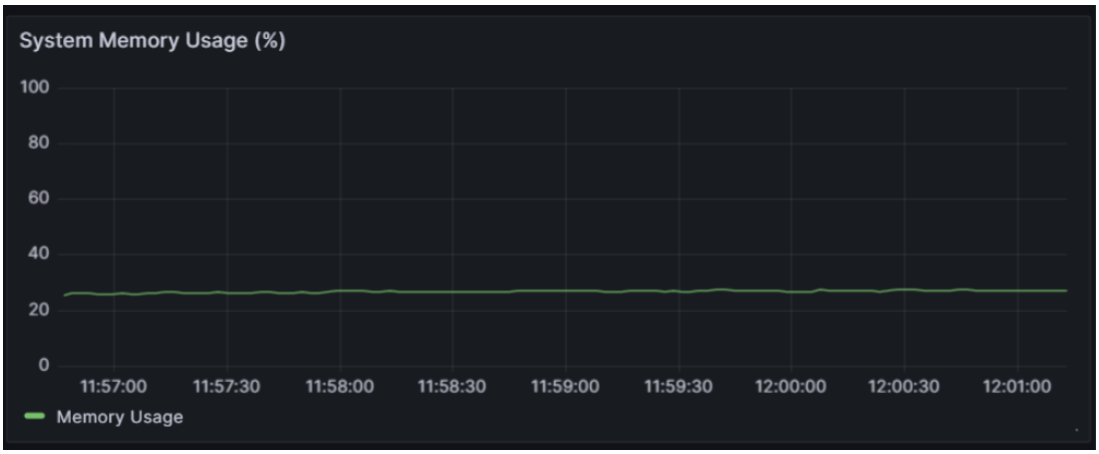
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

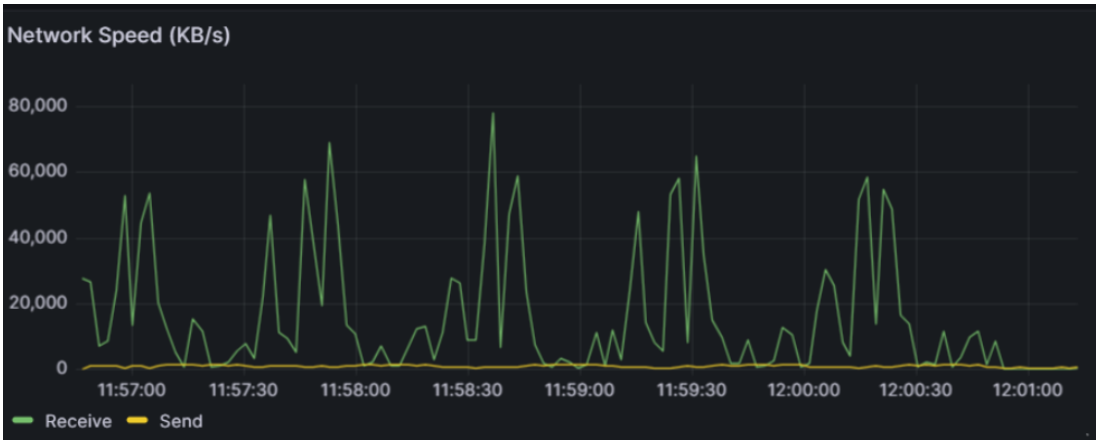


MD Core #2

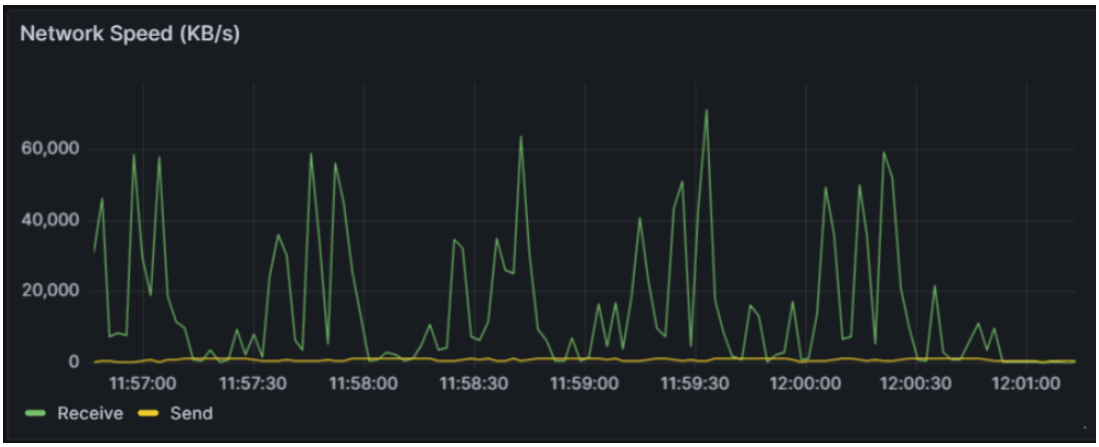


Network usage

MD Core #1



MD Core #2



Shared DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

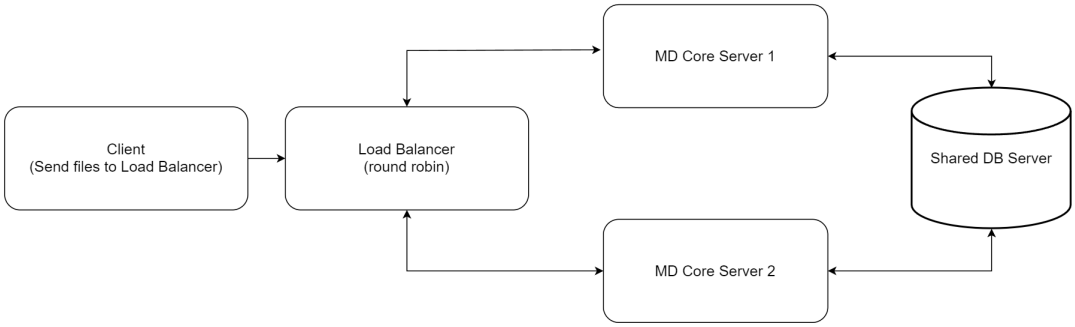
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV)	15.5	63.1	73.3	25.2	25.8	Receive: 4555.9 Send: 694.4	Receive: 4516.2 Send: 692.4	492,620.5	0.0
Deep CDR	9.70	50.7	51.2	31.9	32.5	Receive: 7062.9 Send: 623.2	Receive: 6653 Send: 654.1	786,189.6	0.0
Proactive DLP	7.2	35.2	35.0	27.9	27.5	Receive: 9047.4 Send: 802.4	Receive: 9859.8 Send: 787.5	1,055,110.8	0.0
Vulnerability	5.0	45.7	49.3	24.4	24.5	Receive: 13,585.7 Send: 966.7	Receive: 13,505.8 Send: 987.4	1,519,562.1	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV) + Deep CDR	19.9	79.1	77.6	34.0	34.2	Receive: 3785.9 Send: 573.8	Receive: 3663.1 Send: 577.7	382,716	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP	21.6	80.3	81.5	37.7	44.4	Receive: 3305.6 Send: 543.3	Receive: 3414.3 Send: 556.2	353,694.9	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	22.5	84.2	79.8	37.9	44.7	Receive: 3267.9 Send: 554.9	Receive: 3398.2 Send: 553.8	338,507.6	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

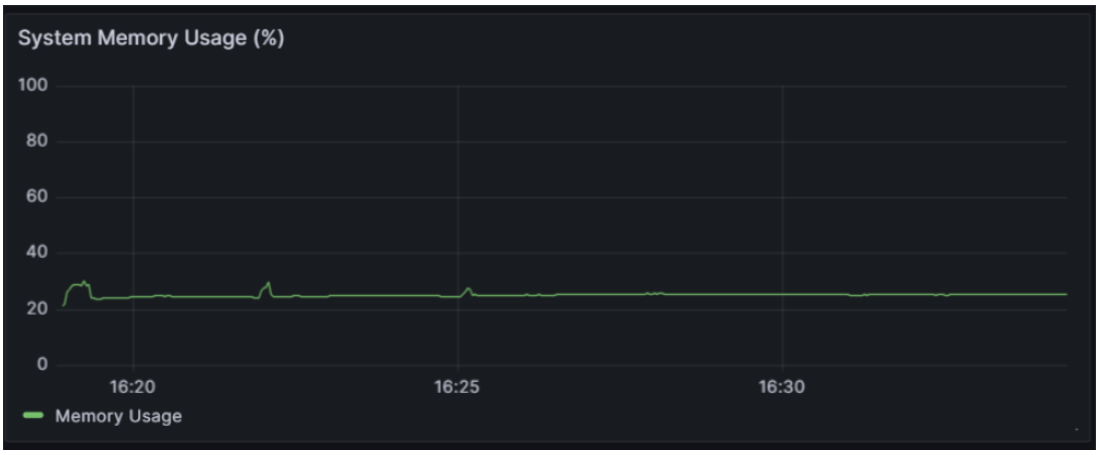
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

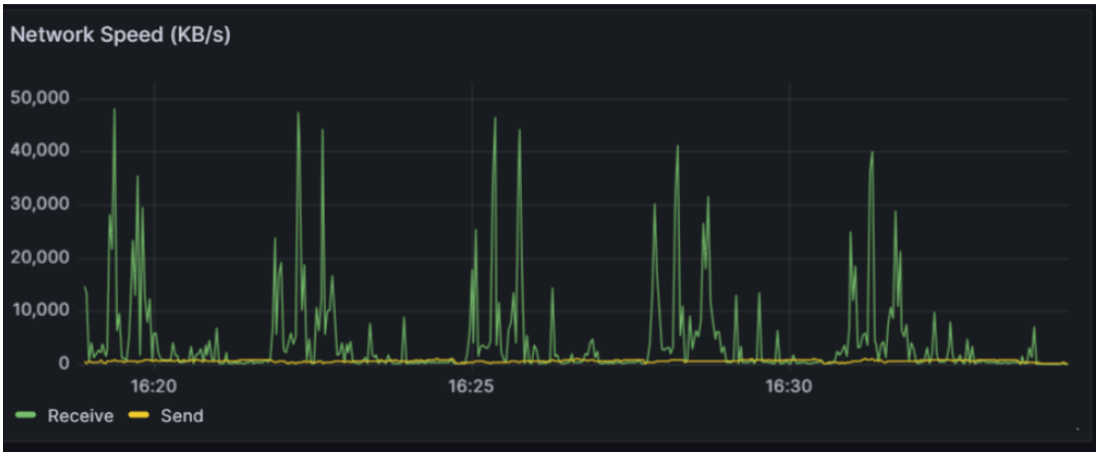


MD Core #2

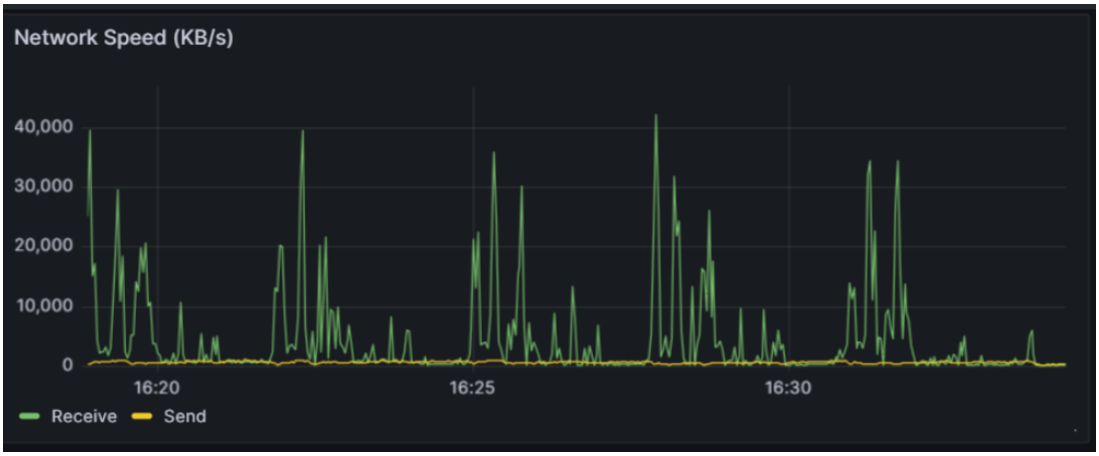


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

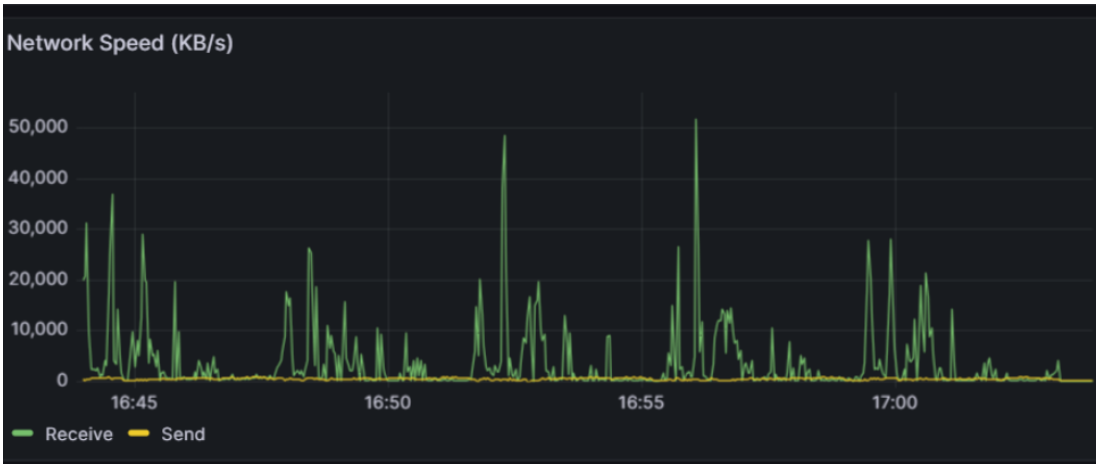


MD Core #2

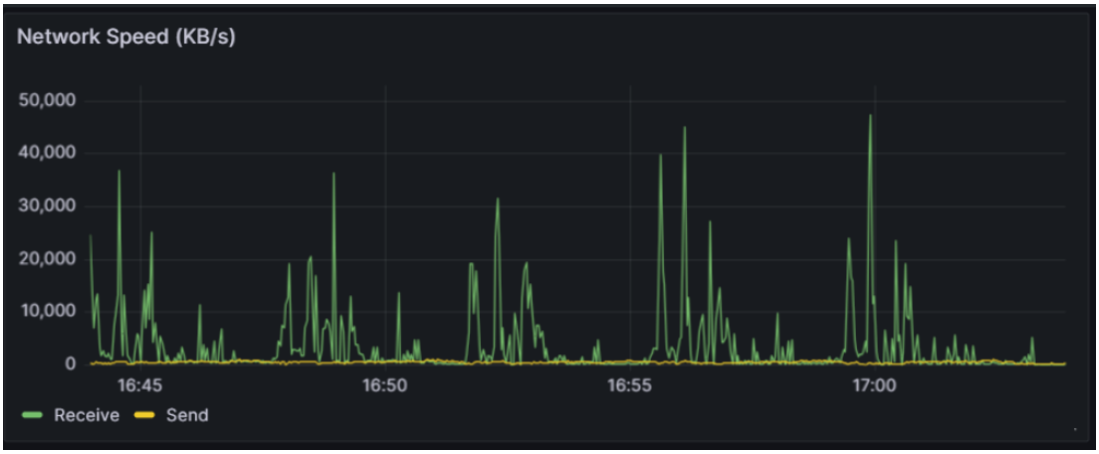


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1

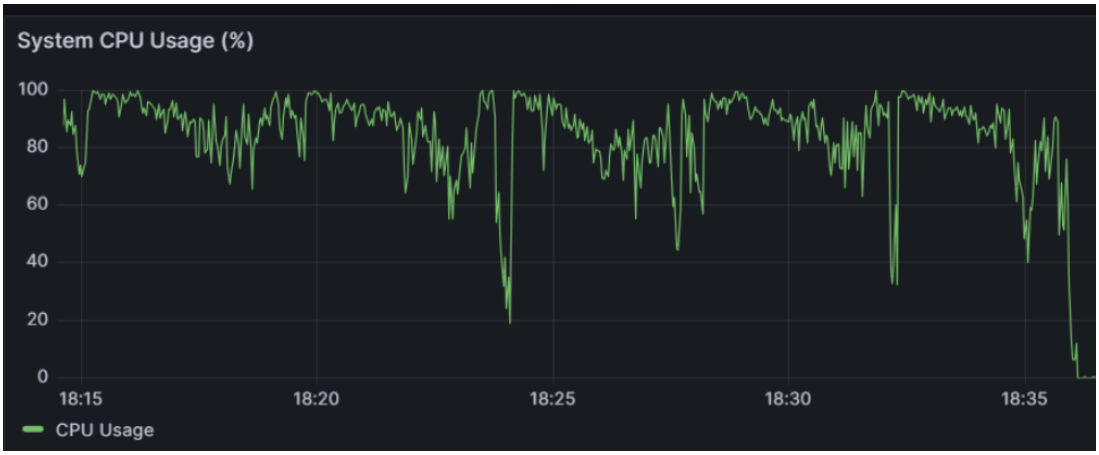


MD Core #2



CPU usage

MD Core #1

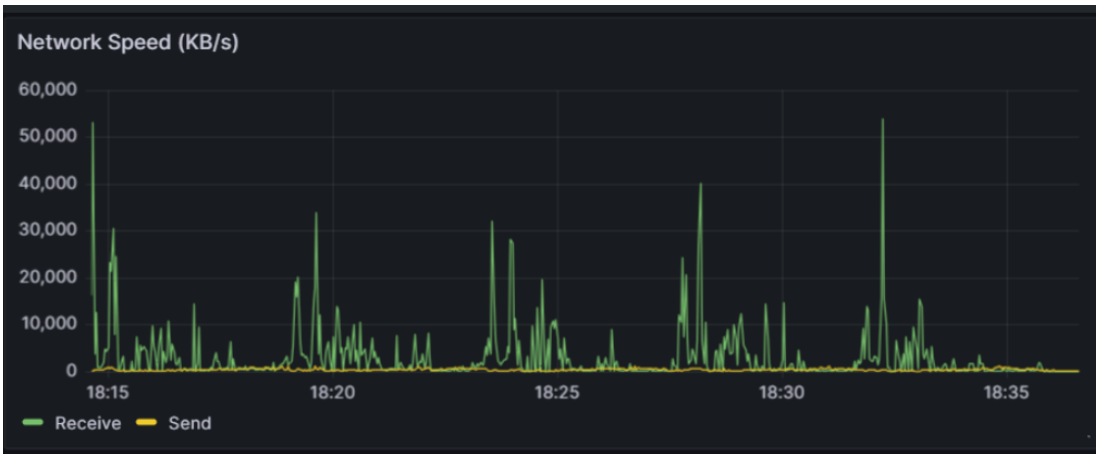


MD Core #2

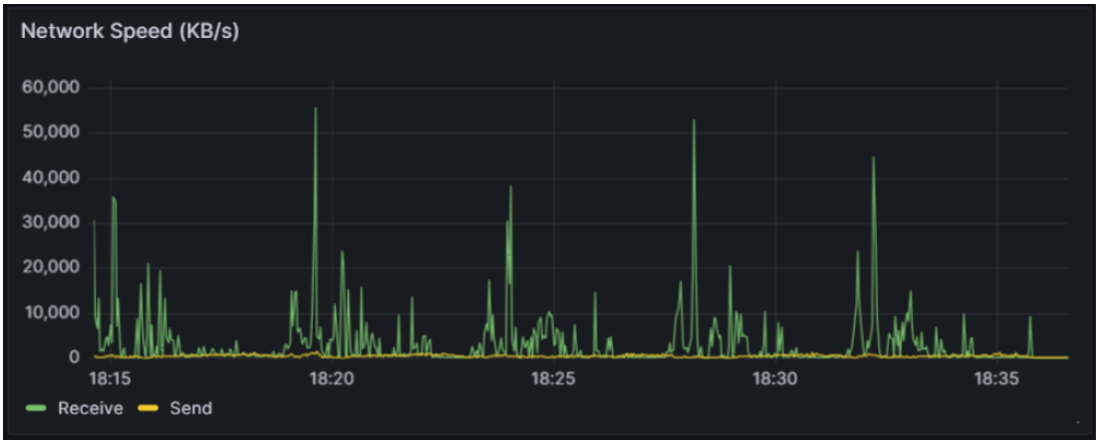


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

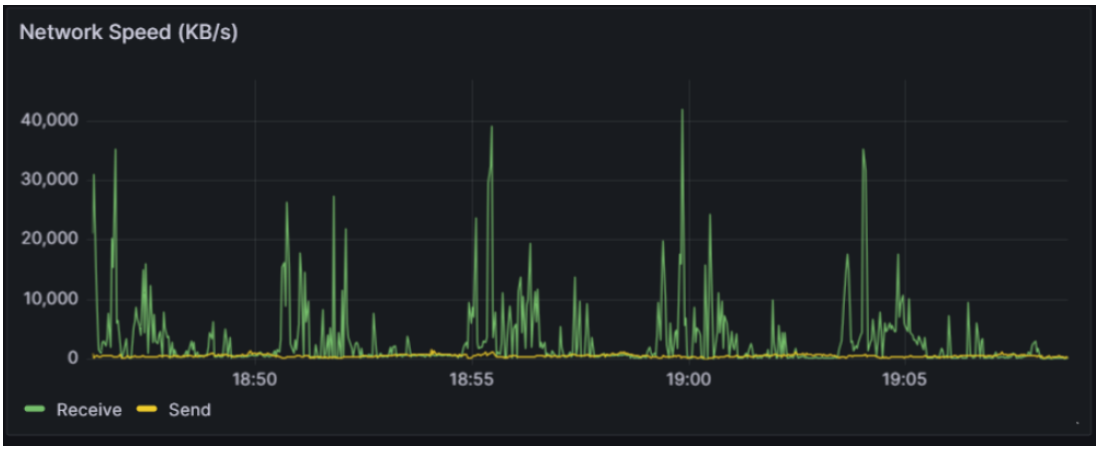


MD Core #2

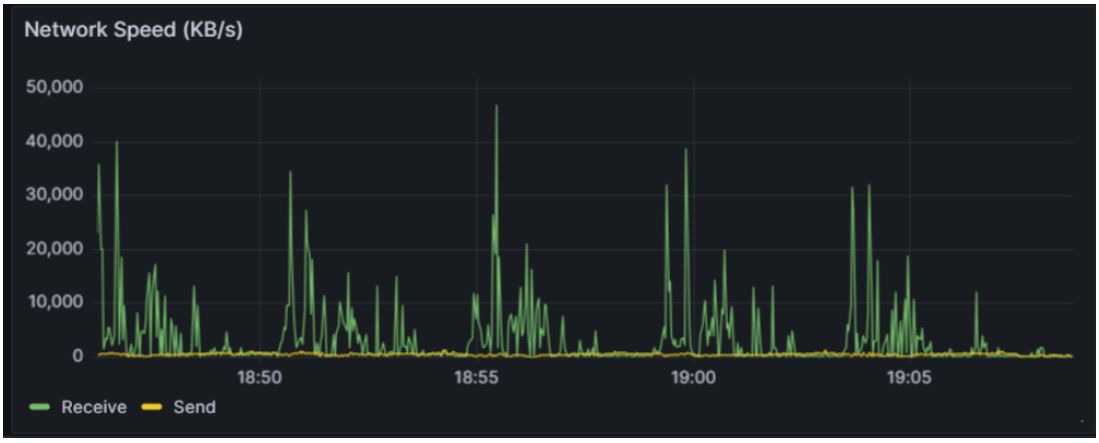


Network usage

MD Core #1



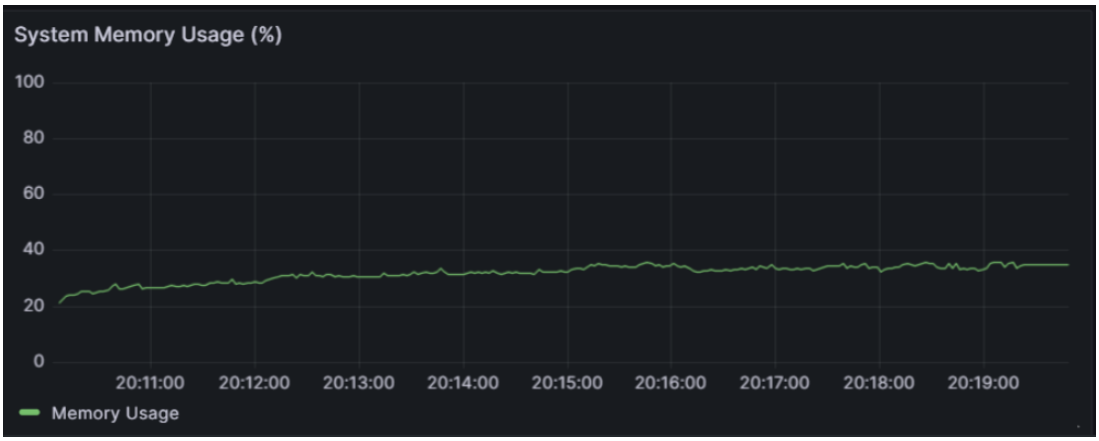
MD Core #2



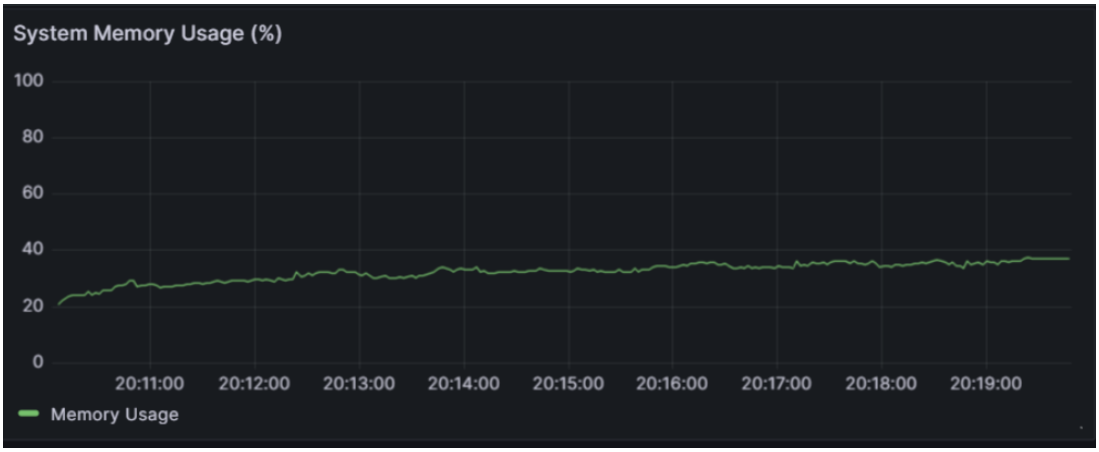
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1

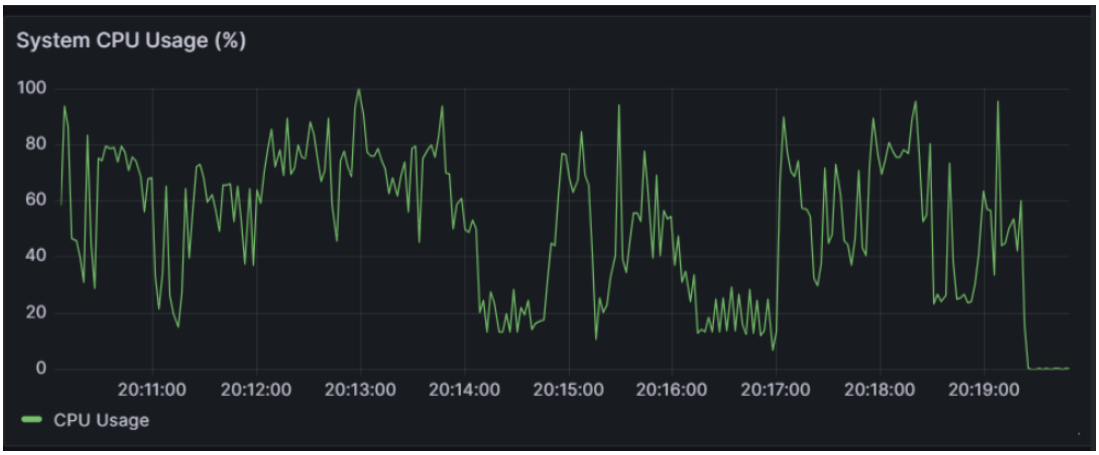


MD Core #2



CPU usage

MD Core #1

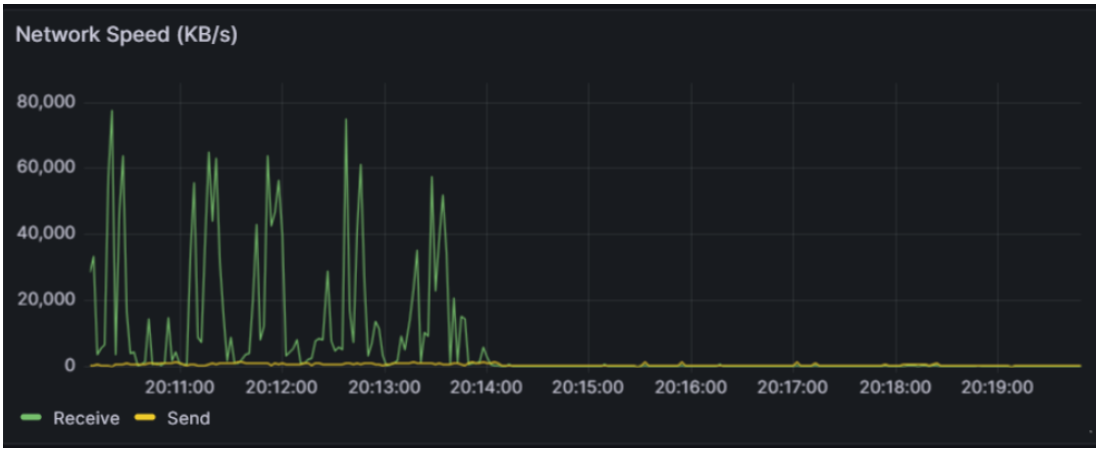


MD Core #2

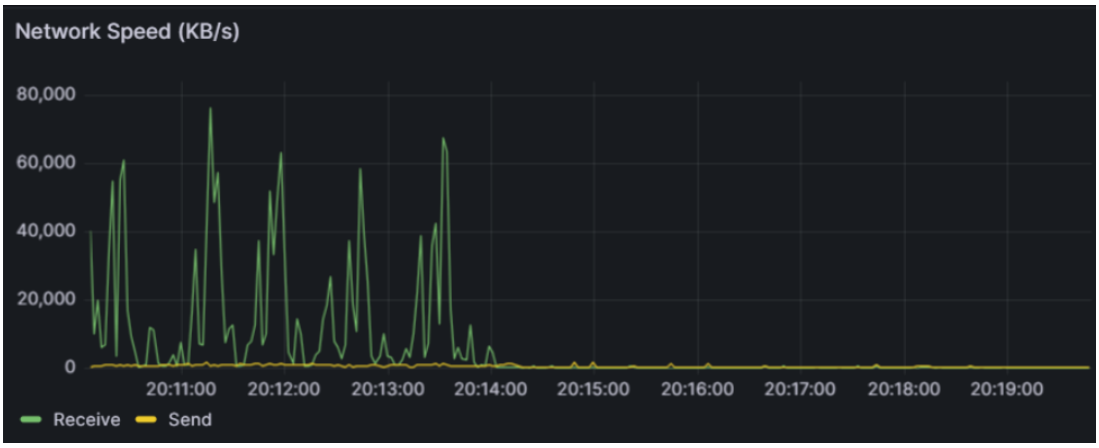


Network usage

MD Core #1



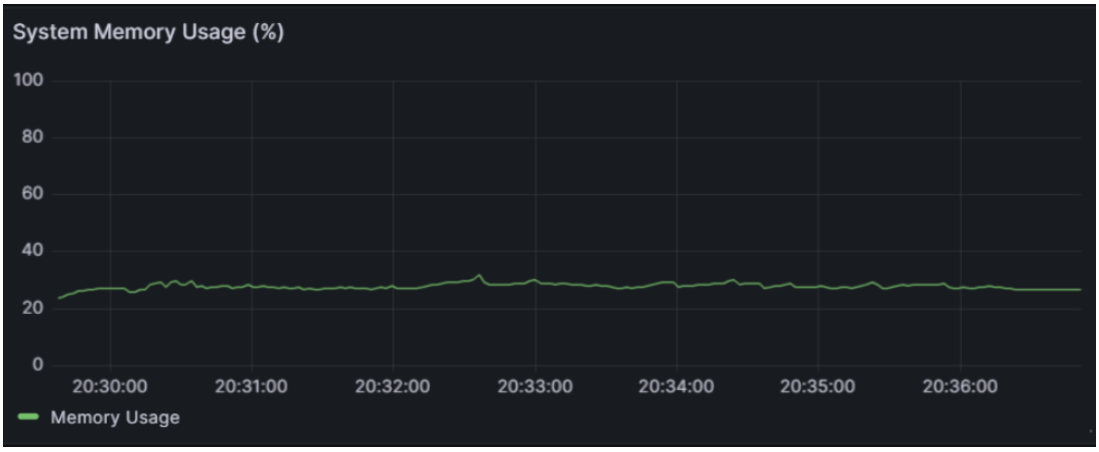
MD Core #2



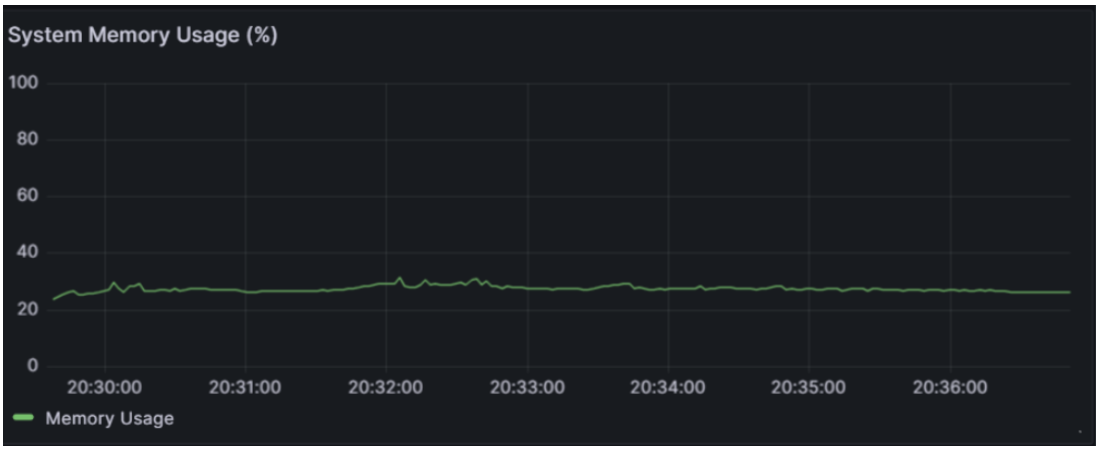
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1

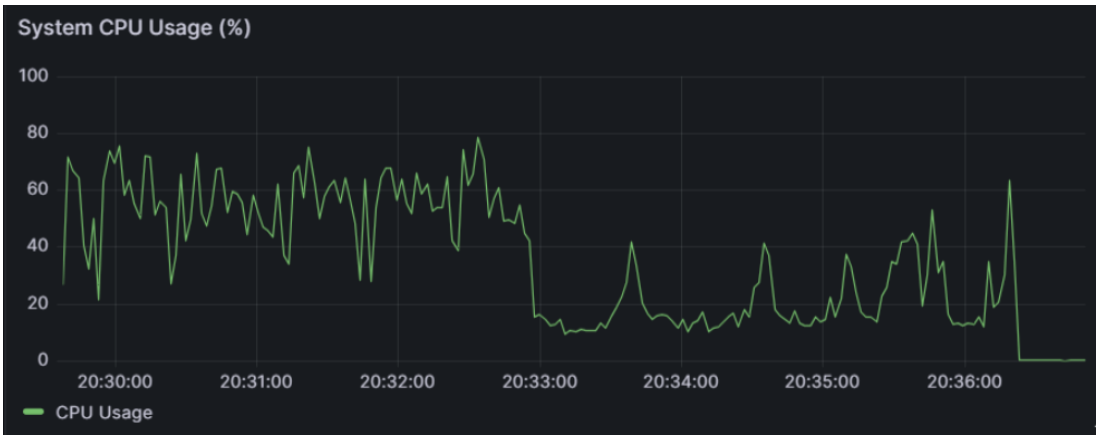


MD Core #2

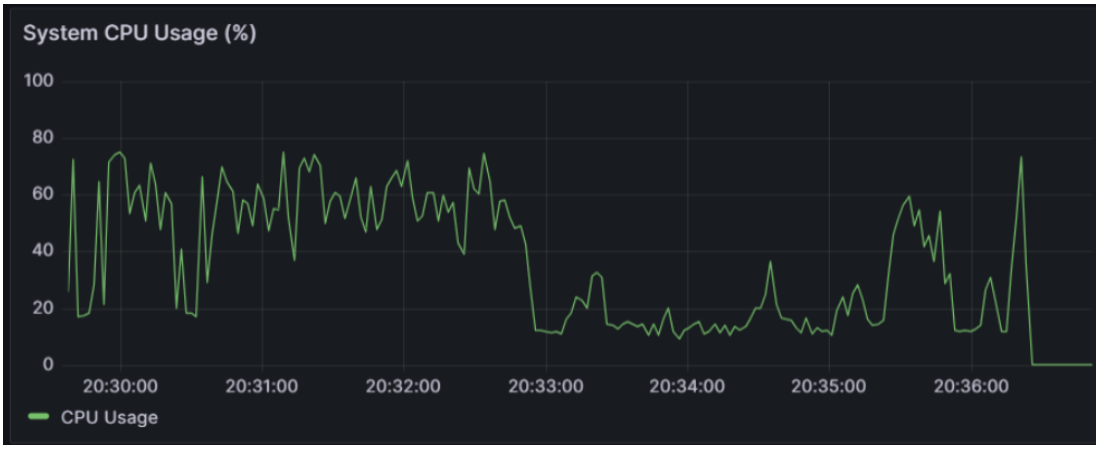


CPU usage

MD Core #1

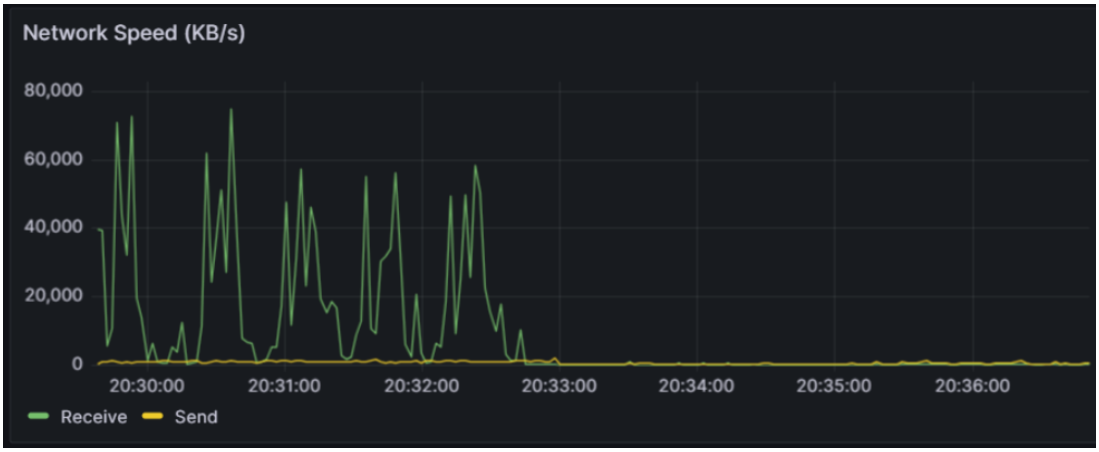


MD Core #2

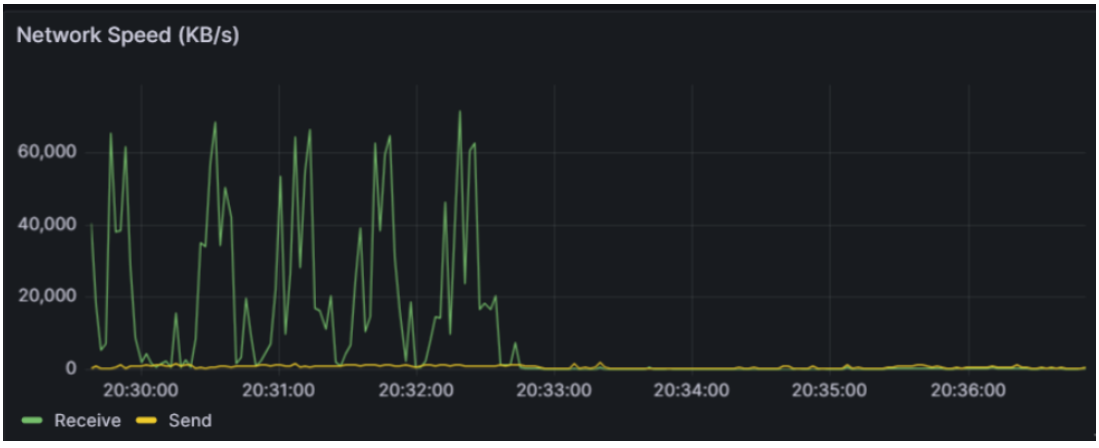


Network usage

MD Core #1



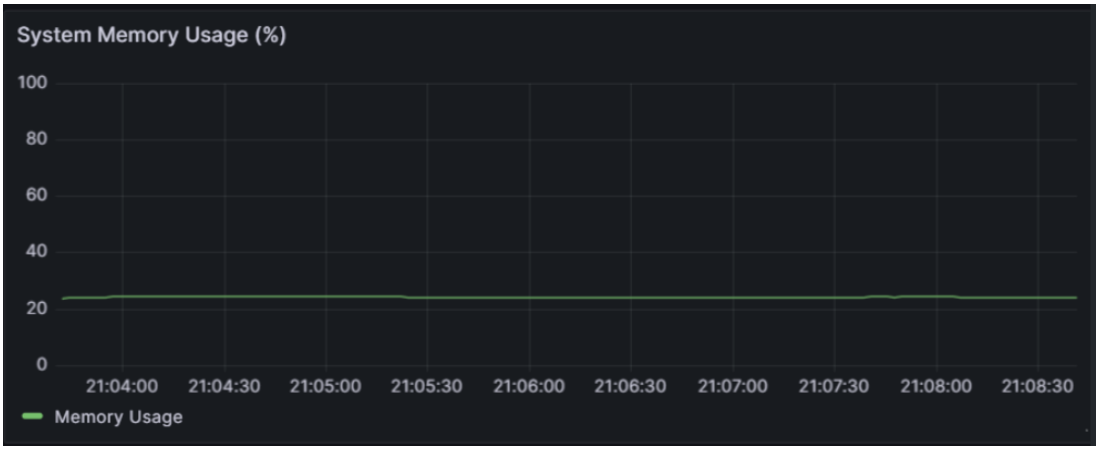
MD Core #2



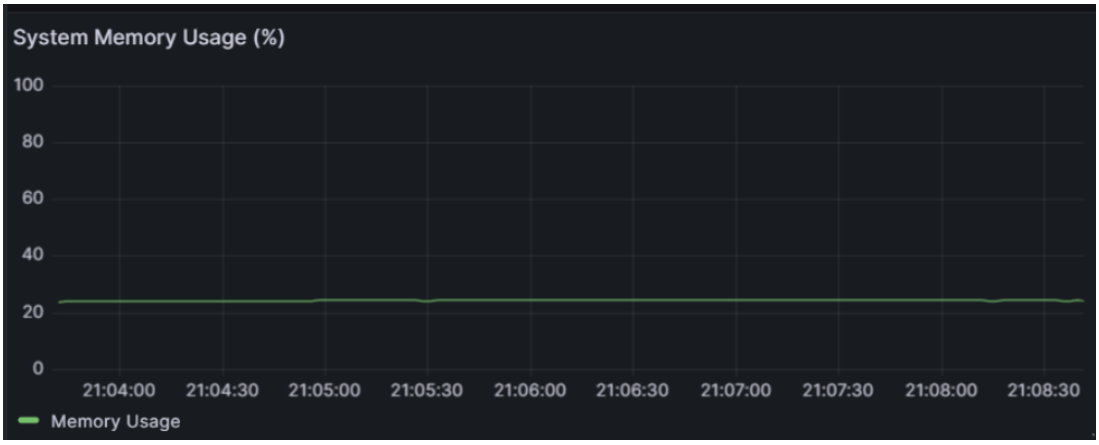
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

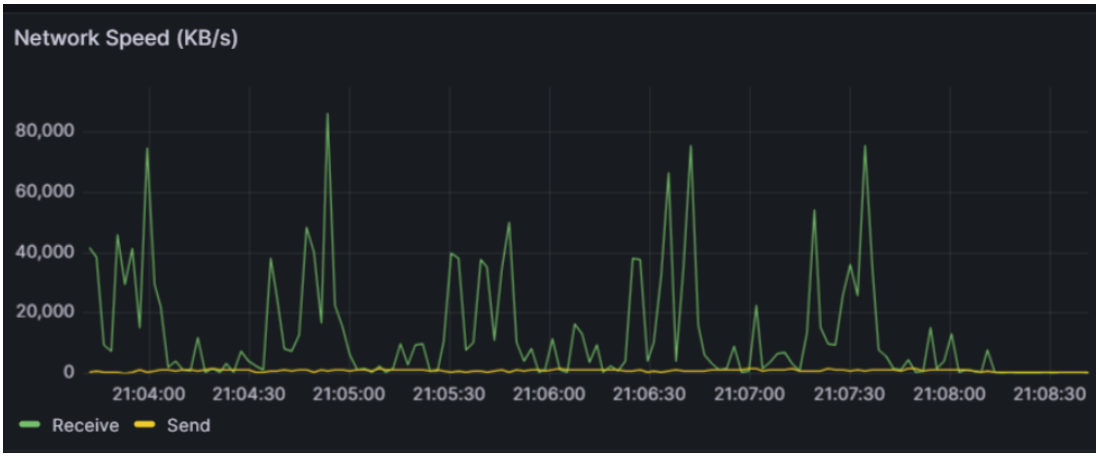


MD Core #2

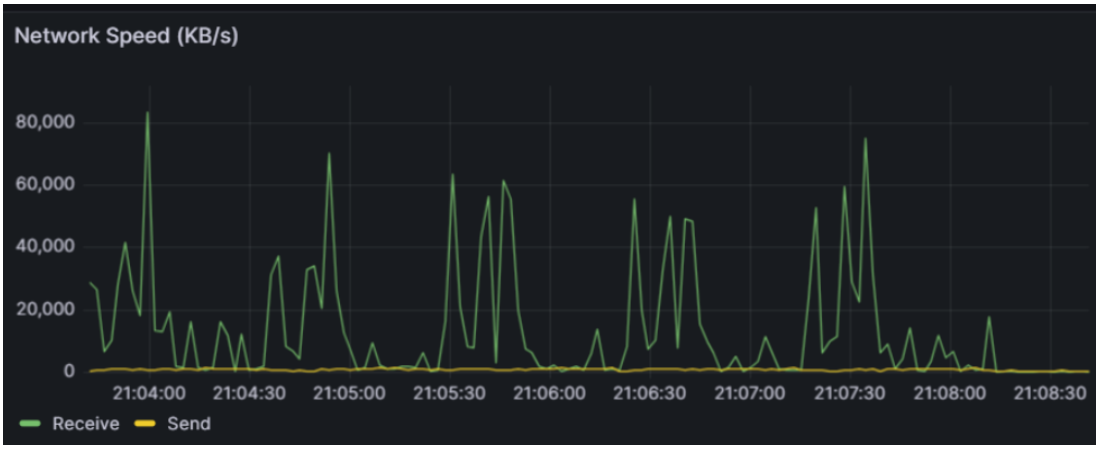


Network usage

MD Core #1



MD Core #2

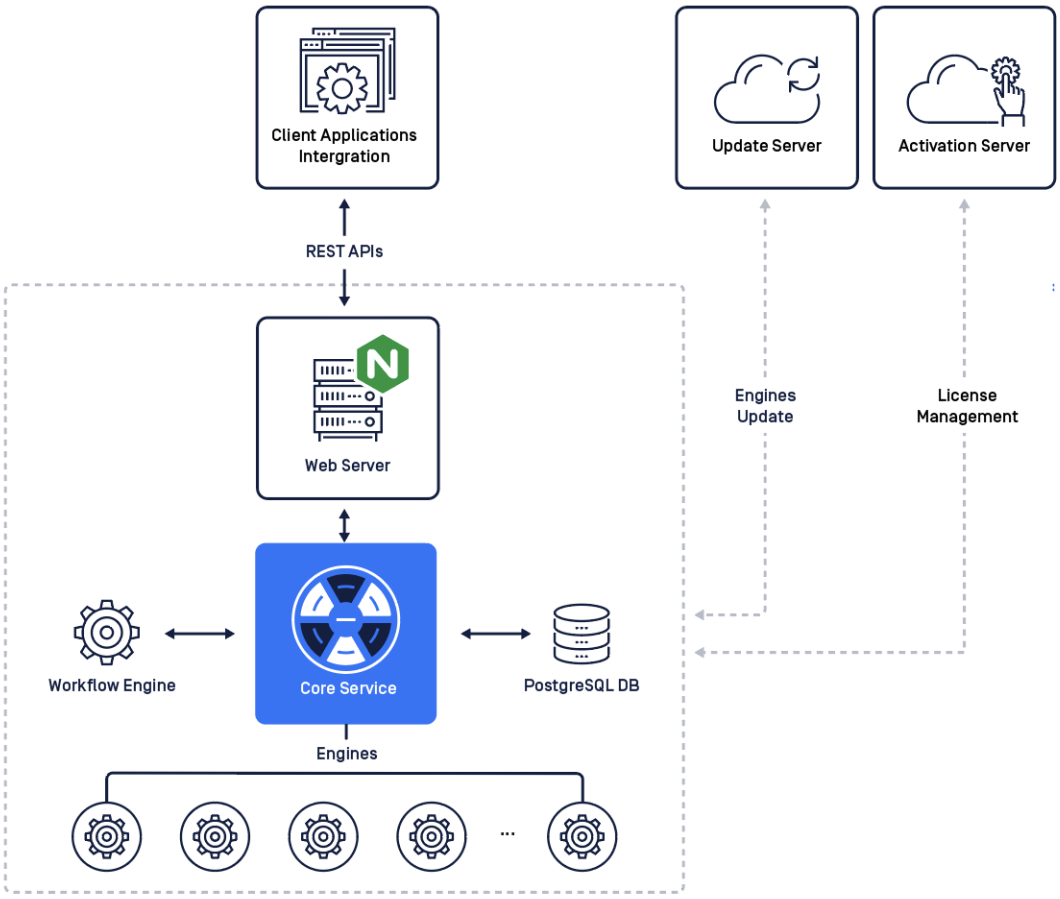

Network Speed (KB/s)
80,000
60,000
40,000
20,000
0
21:04:00
21:04:30
21:05:00
21:05:30
21:06:00
21:06:30
21:07:00
21:07:30
21:08:00
21:08:30
Receive
Send

MetaDefender Core Documentation

The users can consult this web page or, alternatively, they can download the manual in pdf format from the link below:

MetaDefender Core manual (SHA256:).

Recommended System Configuration



MetaDefender Core supports a wide range of deployment options, including air-gapped environments, providing the flexibility to choose the environment that best suits customers' requirements and infrastructure.

- Physical server.
- Virtualization platforms like VMware, Hyper-V and XenServer.
- Cloud Deployment.
- Container Deployment.

> **Note**
> Before installing MetaDefender Core, please refer to the **recommended minimum system configuration** listed below. The requirements are based on a standardized customer deployment and may require more or less resources to achieve the level of performance desired in your environment. Please note that the server specifications are built to allow a high volume daily processing. Please refer to Performance and Load Estimation for more details.

The recommendations below are for MetaDefender Core, API usage only. For any other use cases, please consult the user guide of the licensed products for accurate recommendations.

For certain use cases these might be adjusted and customized on specific needs and SLAs. We highly recommend to engage our ProServ team to assist in fine tuning MetaDefender and get the maximum performance out of your systems.

Microsoft Windows Deployments

[Windows] Supported Operating Systems

Only support 64-bit platforms.

- Windows 10, Windows 11 23H2, 11 IoT (22H2) only for End-User systems (e.g. Kiosk deployments)
  - Based on Microsoft License Terms, you're not allowed to use Windows Desktop as a server, therefore, MetaDefender Core can't be used on these systems for server use cases.
- Windows Server 2016, 2019, 2022

> **Disclaimer**
> End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

> **Plan for legacy OS support**
> OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:
>
> - Ubuntu 20.04 after **April 2025**.
> - Amazon Linux 2 after **June 2025**.
> - Windows 10 after **October 2025**.
>
> If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

[Windows] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 8	8	8 GB	16 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 12	16	16 GB	24 GB + 50 GB + 100 GB
MetaDefender Core 16	16	16 GB	32 GB + 50 GB + 100 GB
MetaDefender Core 20	32	16 GB	40 GB + 50 GB + 100 GB
MetaDefender Core MAX	32	32 GB	120 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core is installed.
- For a better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.
- SBOM processing requires more disk space, please allow at least 15GB free disk for the engine to work properly.

[Windows] Third Party Dependencies

- .NET framework 4.5 or above
- Microsoft Visual C++ Redistributable for Visual Studio 2015-2022 (version 14.38 or higher); x64 version.
- PostgreSQL: Only applicable when using a pre-installed PostgreSQL server running remotely.
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Some engines also have dependencies as described below:

Engine Name	Dependency
Deep CDR	Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 5.8.0 or above)
ESET	MetaDefender Core v4 temporary directory should have more than 200MB free disk space
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
Proactive DLP	• Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 2.0 or above) • .NET 6 dependencies • CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
Scrutiny	Microsoft Visual C++ 2015 - 2019 Redistributable x86 and x64
Systweak	.NET runtime 6.0, 7.0 or 8.0
Bkav Pro	Microsoft Visual C++ 2015 - 2022 Redistributable x64

[Windows] Installation Details

MetaDefender Core on Windows uses "C:\Program Files\OPSWAT" folder for storing resources or the installation directory.

MetaDefender will use its resources folder to store temp files as part of the analysis. **It's recommended to exclude this folder from real-time protection monitoring**. Knowledge base:

- How to set up exclusions on anti-malware software to prevent disruption
- Mitigating scan failure triggered by real-time protection service
- Can local AVs interrupt ongoing scans?

Linux Deployments

[Linux] Supported Operating Systems

Only support 64-bit platforms.

- Red Hat Enterprise Linux 8, 9
- Rocky Linux 9
- Debian 11, 12
- Ubuntu 20.04, 22.04
- Amazon Linux 2 is supported via Amazon Machine Image (AMI)

Plan for legacy OS support

OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:

- Ubuntu 20.04 after **April 2025**.
- Amazon Linux 2 after **June 2025**.
- Windows 10 after **October 2025**.

If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

For Hardended Linux OS

Some AV engines could be failed to initialize and run on MetaDefender Core (e.g. ESET, Kaspersky) due to execution permission required by them on `/var/tmp/ometascan` folder.

Please follow steps at Why have the ESET and Kaspersky scan engines failed to initialize on the hardened Linux OS? for remediation details.

Disclaimer

End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

[Linux] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 5	4	4 GB	10 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 10	8	8 GB	20 GB + 50 GB + 100 GB
MetaDefender Core MAX	16	16 GB	40 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core resides on.
- For better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.

[Linux] Third Party Dependencies

Dependencies list:

- openssl
- grep
- lib32stdc++6 (>= 4.5)
- libc6-i386 (>= 2.10)
- procps
- zlib1g
- libcurl3 (>= 7.19.7)
- libcurl4
- postgresql:
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Not all the above dependencies will need to be installed, it is dependent on different Unix distro and version

Some engines also have dependencies as described below:

Engine name	Dependency
Archive engine	• To extract password protected MS Office files, you need to install .NET 6 dependencies, libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • Rocky Linux 9.x/Red Hat 9.x: libnsl
Deep CDR	• Rocky Linux 9.x: libgomp, ncurses-compat-libs, libnsl • CentOS: libgomp, ncurses-compat-libs • Red Hat 8.x: libgomp, ncurses-compat-libs, libnsl • Red Hat 9.x: libgomp, ncurses-compat-libs, libnsl, compat-openssl11 • Ubuntu/Debian: libgomp1, libncurses5 • libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • .NET 6 dependencies
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
FileType engine	Rocky Linux 9.x/Red Hat 9.x: libnsl
Proactive DLP	libgdiplus, mscorefonts .NET 6 dependencies CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
RocketCyber	libgomp

Note: Some dependencies may need to be installed from external repositories, not from the OS default repositories.

[Linux] Installation Details

MetaDefender Core default installation path is using /var folder for storing resources:

- `/var/lib/ometascan` : installation folder with all its resources (database, DLP processed / CDR sanitized / quarantined file storage, engine package and definition updates).
- `/var/log/ometascan` : application logs.
- `/var/run/ometascan` : runtime folder for Core related processes.
- `/var/lib/ometascan` : all successful deployed engines data files.
- `/usr/lib/ometascan` : all shared libraries required by MetaDefender Core, bundled PostgreSQL server related binaries and libraries.
- `/var/tmp/ometascan/resources` : all temporary processing files for MetaDefender Core to process.

Storage Usage Information

Based on the configuration, MetaDefender Core will need additional disk space to store analysis/result data, quarantined files and temporary files:

	Data Storage
Analysis Reports (Processing History)	Approximately 4.5GB is required for every 1M analysis reports stored. • 100 GB is recommended as a baseline. Database storage requirements are affected by Data Retention settings, engine settings and scanning load.
Quarantined Files	Depends on the customers' dataset and configuration
Sanitized Files	Depends on the customers' dataset and configuration
Temporary processing files	Depends on the customers' dataset and configuration • A 1GB non-archive file requires 1GB free disk space at least. • A 1GB archive file requires (1 * 2)GB free disk space at least. • 50GB is a baseline and should be adjusted based on observation of the solution within the environment, with consideration to peak times and overall file makeup (e.g. complexity, size, archives).
Note	Options are available to control how large and/or complex files are handled to limit the amount of system resources required. See Workflow Configuration Template for more information.

Custom Engines

The recommendations above are specific for MetaDefender pre-packaged bundles.

However for additional Custom Engines, please review the Knowledge base to review additional requirements (if any) for the selected engine.

Browser Requirements for the MetaDefender Core Management Console

One of the following browsers is suggested to view the MetaDefender Core Management Console:

- Microsoft Edge
- Chrome
- Firefox
- Safari

Chrome, Firefox, Safari and Microsoft Edge browsers are tested with the latest available version at the time of release.

Network Bandwidth

In online environment where MetaDefender Core is configured to download engine update packages directly from OPSWAT update server source, then using gigabit network speed (125 MB/sec) is highly recommended to avoid potential network related issue while downloading files, or at least 100 Mbps (12.5 MB/sec) network speed.

Allowlisting URLs

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts':

- https://activation.dl.opswat.com - for license online activation
- https://update.dl.opswat.com - for engines and databases download

Note: *IP address-based allowlisting on your firewall might fail after some time since OPSWAT uses CDN (AWS Cloudfront) to faster delivery updates over the world, and IP address of edge servers might change over time.*

Depending on your location, connections with OPSWAT cloud-based servers be forwarded to specific AWS Cloudfront's edge locations, each is assigned with its own unique IP address ranges. You might also need to allow correct AWS Cloudfront's corresponding IP address ranges by referring to https://ip-ranges.amazonaws.com/ip-ranges.json

Warning

Even the OPSWAT update servers host updates for all of the available engines we support, sometimes custom engines might try to connect to their own cloud for updates, but this can be disabled in firewall and they will be updated just from OPSWAT cloud servers instead.

Differences Between MetaDefender Core v4 and v5

We are very excited to inform you about the release of version 5.0.0 of our MetaDefender Core!

The new version is designed to enhance the product performance, user experience, as well as to provide more flexibility, security, and scalability for our customers.

At the same time, we are also committed to keeping backward compatibility for whom running MetaDefender Core v4, and avoid the need for updating your client integration to move forward with MetaDefender Core v5.

v4 REST API Integration Supported
All public REST API endpoints available on MetaDefender Core v4 will be continously supported on v5.

v4 Product Major Features Supported
All product major features available on MetaDefender Core 4.21.2 (the latest one on v4) will be continously supported on v5.

Seamless Product Upgrade
Upgrading from MetaDefender Core v4 to v5 will be supported and covered with data migration without additional manual steps from users.

Licensing information will be also migrated.

Below are some highlights of the first v5 release, MetaDefender Core 5.0.0:

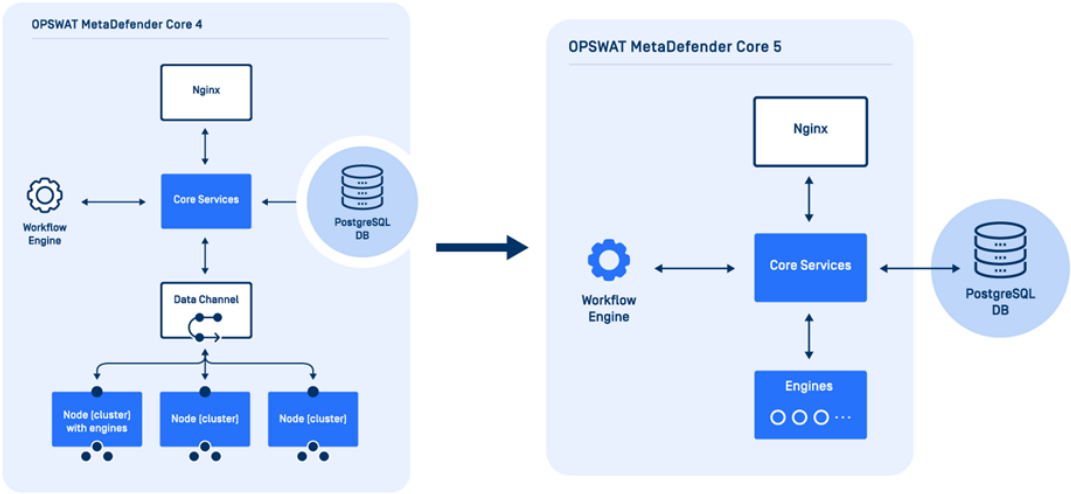
New product architecture

This version introduces a simpler product architecture that speeds up integration and automation, file processing flow, and immensely enhance the product performance and efficiency.

Earlier versions utilized MetaDefender Node concept, designed to support scaling of the scanning infrastructure by distributing scan requests among several different processing Nodes. Product licensing was based on number of Node instances. The benefit of having such a distributed infrastructure is that based on Node loads, Metadefender Core server can always choose the most appropriate Node to assign a new scan task to. However, having nodes in the v4 architecture caused more complexity and possible latency in the processing flow and entailed maintenance cost.

In this release, we retired multi-Node deployment and restructured MetaDefender Core's backend. This alteration significantly improves file processing speed and product efficiency, as well as eliminates extra cost for maintaining node services. Users now can simply install additional MetaDefender Core instances for scalability requirement,

as well as configure their own load balancer to coordinate tasks among multiple MetaDefender Core instances. Accordingly, product licensing is now based on number of the product instances.



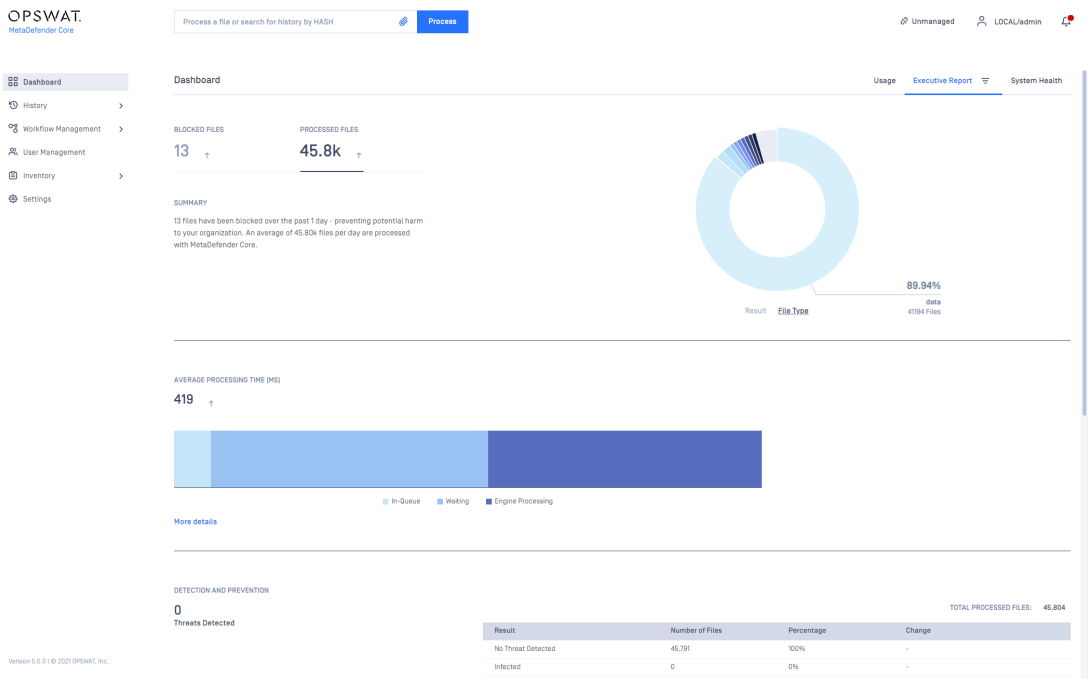
Please be informed that

- All existing administrative APIs related to "Node" on v4 will be kept as it is on v5 to ensure backward compatibility with existing client integrations.
- Applicable settings defined in Windows registry (Windows) or ometascan-node.conf config file (Linux) for Node will be migrated to Core. Details: MetaDefender Configuration

Brand-new product user interface

The second notable update in this version is a brand-new UI for the management console, which was designed with the focus on the quality and thoughtfulness of the user experience. The new administrative console UI provides an advanced set of Core features and observable processing status.

Comprehensive executive dashboard reporting offers dynamic, real-time and actionable threat insights. It also provides visibility of your security posture allowing you to paint a picture of security that aligns with the overall business. The report includes easy-to-read charts, graphs, and statistic explanation that help your security professionals reduce clutter, gather information quicker, spot threat activity early and act swiftly.



We're continuing to improve our product UI to provide users an excellent experience with the application and overall usability. Please expect more powerful UI features coming in next releases.

New compression archive engine for archive sanitization

In previous version, MetaDefender Core's archive engine implemented four functions including decryption, extraction, encryption and compression that with a high volume of files it takes time to process them all. For the purpose of enhancing archive handling performance, this release comes with a separated compression archive engine, which dedicatedly processes archive sanitization (compression and encryption). It is also created to facilitate maintenance and further feature development.

Archive sanitization feature now requires Deep CDR license

As to respect sanitization practice and ensure unknown threats prevention as designed, now starting on version 5, it's mandatory to have Deep CDR licensed for archive sanitization feature enablement on MetaDefender Core workflow configurations.

Archive handing related engines changed

The legacy "Archive" engine on MetaDefender Core v4 will be changed its name to "Archive Extraction" engine, even when you are still on MetaDefender Core v4.

A brand new "compression archive engine" is to dedicatedly handle archive sanitization in v5 (archive compression), that helps relevant archive handling performance improvement, easier for maintainance and further feature development.

Notes when you are still on MetaDefender Core v4 (not upgraded yet)

The legacy "Archive" engine (now name changed to "Archive Extraction" engine) on MetaDefender Core v4.x will still cover both extraction and compression archive tasks as usual.

Even though your MetaDefender Core v4.x might now have a new "Archive Compression" engine auto downloaded, but that engine cannot be deployed (marked as "permanently failed" in relevant engine status check API). This behavior is expected, and not causing any impact to MetaDefender Core processing flow, or its performance though.

- Just in case your client integration has a check for engine status before triggering scan request, then you might want to disable "Archive Compression" engine status on MetaDefender Core v4 using REST API. Detailed instruction: Archive Compression Engine Permanently Failed on MetaDefender Core v4
- When you later upgrade to MetaDefender Core v5 from v4, then you might need to manually re-enable "Archive Compression" engine to make it active and running on MetaDefender Core v5.

Notes for licensing

For offline deployment (MetaDefender Core is not connected to the Internet directly for online update), you will need to re-activate again on OPSWAT portal to retrieve latest engine list in .yml file including "Archive Compression" engine if you are currently licensed with Deep CDR.

Meanwhile, for online deployment, this manual procedure is not required since MetaDefender Core will cover that itself.

Import & export configurations

Not only workflow settings like in previous versions, but also all other general settings (update, security, data retention, server, hashes to skip) and most importantly, user settings, will be able to export and import on MetaDefender Core version 5.

Learn more at Import/Export configuration

Backward compatibility supported

The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

Windows application event logging disabled by default

Applicable only to Windows OS based MetaDefender Core.

Help avoid duplicated logging with separate file option, and improve overall processing performance. Of course, when being still valid to you, you are still supported to enable it back.

CSRF token header renamed

CSRF token originally created for security strengthening and being used by MetaDefender Core management console UI since version 4.21.0 via header `oms_csrf_token`

To get it compliant with some load balancers which do not accept underscore in header name by default, and save you from additional load balancer configuration step, we have now changed the header name to `oms-csrf-token`

This change does not impact to any REST API based client integration.

More comprehensive and accurate error messages on REST APIs

Impacting on following REST API endpoints:

- `PUT /admin/config/session`
- `PUT /admin/config/update`
- `PUT /admin/config/scan`
- `PUT /admin/config/skip`
- `PUT /admin/config/auditlog`
- `PUT /admin/config/webhook`
- `PUT /admin/config/rule`

Upload time and queue time logic updated in JSON scan response

Impacting on following REST API endpoints:

- `GET /file/{data_id}`
- `GET /hash/{md5/sha1/sha256}`
- `GET /archive/{data_id}`
- `POST /file/{async}`

Added new field:

- `receive_data_timestamp`: The timestamp when upload progress started (first byte received) (in milliseconds)
- `upload_time`: Total time elapsed for upload process (in milliseconds)

Modified existing field (new logic):

> **Backward compatibility supported**
> The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

- `upload_timestamp`: The timestamp when upload progress finished (all bytes received) (in milliseconds)
- `queue_time`: Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue time = start_time - upload_ timestamp) (in milliseconds)

Release version 5.0.0 also comes with variety of outstanding features and improvements.

For detailed information about our latest release please view our Release notes

Installation

Starting with MetaDefender Core 4.20.0, the centralized/shared database deployment model is supported (all MetaDefender Core instances share the same database), along with traditional standalone database mode (every MetaDefender Core instance has its own database to manage).

- Standalone mode
- Shared Database Mode

Installation

1. Download the package of your choice from My OPSWAT
2. Install the package on your computer via the Command Line or via the Wizard
3. Open a web browser and point to http://<server name or IP>:<port>
    - The default port is **8008**
    - In case of problem check Inaccessible Management Console page
4. Complete the required steps of the basic configuration wizard
5. You must activate this deployment to use its features

Installation notes

- If the MetaDefender Core package **dependencies are not installed** on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.
- During installation the databases might need to be upgraded. This could take noticeable time depending on database size (eg.: length of scan history).

Standalone mode

Wizard Installation

The Install Wizard is only for the Windows installer (.msi file).

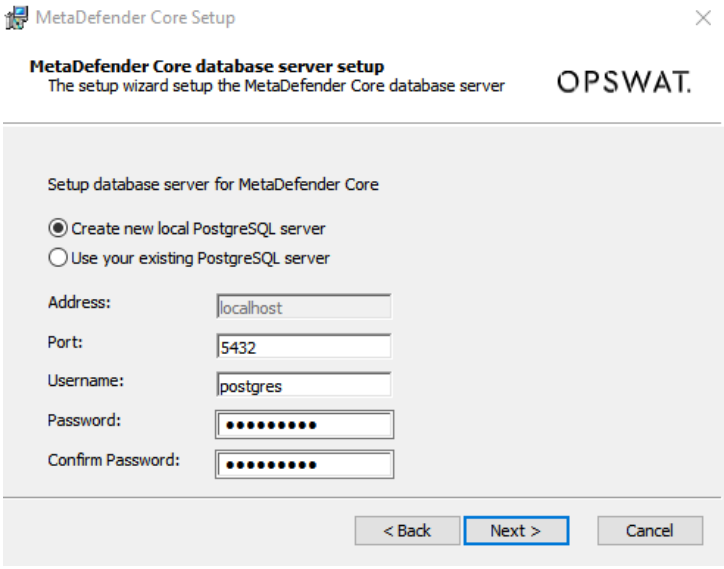
Since MetaDefender Core 4.19.0, we have switched to using PostgreSQL DBMS which needs a dedicated database server. MetaDefender Core supports two modes to setup PostgreSQL server:

Local (Bundle) PostgreSQL server

MetaDefender Core will install a new PostgreSQL server locally in the same box with the product.

> **PostgreSQL user credentials**
> Unicode characters are NOT supported for "Username" and "Password"
>
> "Password" does not accept # (hash) as starting character (e.g. #ABC)

> **The ignition file**
> If the ignition file exist in the designated location, then MetaDefender Core will use the details defined in the ignition file, rather than what is passed via the installation wizard.
>
> More info about the ignition file based installation: Command Line Installation

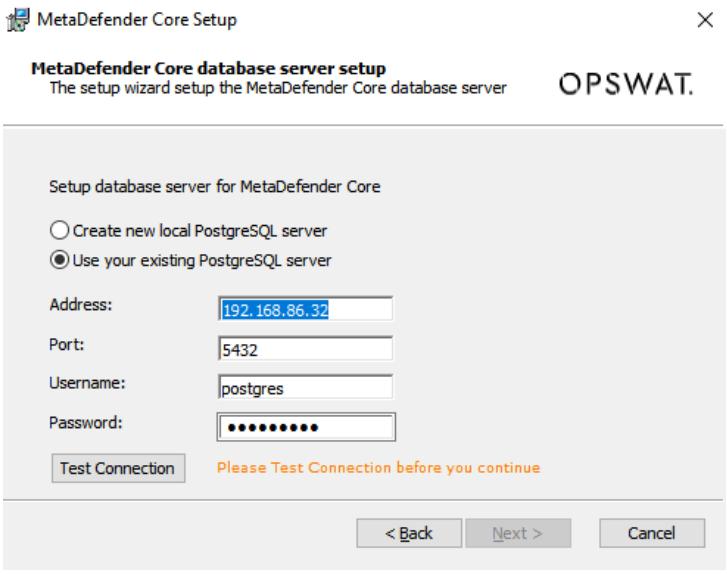


Remote (Existing) PostgreSQL server

MetaDefender Core will connect to remote pre-installed PostgreSQL running remotely.

> **PostgreSQL connection configuration**
> "Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

> **PostgreSQL user privilege requirements**
> MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.
>
> Details: PostgreSQL User Privilege Requirements



Command Line Installation

Ignition File

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Make sure to create the ignition file, if it doesn't exists before you go ahead and install MetaDefender Core via command line.

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

MetaDefender Core supports two modes to setup PostgreSQL server:

1) Local PostgreSQL server

- MetaDefender Core will install a new PostgreSQL server locally, next to the product.
  - A sample iginition file for PostgreSQL server information (***Only non-Unicode characters supported for "user" and "password"__***):

ometascan.conf bash

```
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
```

2) Remote PostgreSQL server

> **Info**
>
> **PostgreSQL version 12 (at least 12.3) or version 14 is required.**
>
> According to PostgreSQL notice, version 12 will stop receiving fixes on November 14, 2024, so we suggest that you use version 14.

MetaDefender Core will connect to a pre-installed PostgreSQL server running remotely and will setup its database.

Make sure to follow steps at Open Connection On PostgreSQL Server first, so that your remote existing PostgreSQL server to accept upcoming connection from MetaDefender Core.

A sample ignition file for PostgreSQL server information:

ometascan.conf bash

```
[dbserver]
type=remote
host=192.168.86.32
port=5432
user=existingserver_admin_user
password=existing_server_pass
```

After installing MetaDefender Core successfully, you may want to remove all credentials info created for [dbserver] section in ignition file for security reason.

> **Warning**
> If the MetaDefender Core package dependencies are not installed on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.

Debian package (.deb)

bash

```
sudo dpkg -i <filename> || sudo apt-get install -f
```

On Red Hat Enterprise 7.x Linux / CentOS package (.rpm)

bash

```
sudo yum install <filename>
```

For systems which enabled GPD check flag:

bash

```
sudo yum install --nogpgcheck <filename>
```

Red Hat Enterprise Linux 8.x package (.rpm)

Install the latest PostgreSQL repository RPM:

> **Note**
> Due to the shorter support cycle on Red Hat Enterprise Linux, all supported versions of PostgreSQL are not available on this platform. The PostgreSQL Yum Repository will integrate with the normal systems and patch management, and provide automatic updates for all supported versions of PostgreSQL throughout the supported lifetime of PostgreSQL. Reference: https://www.postgresql.org/download/linux/redhat/

bash

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

In case installing a remote PostgreSQL on either the same MetaDefender Core server or another server, you need to install the following additional package. PostgreSQL version 12.3 or newer is recommended.

bash

```
sudo dnf install -y postgresql12-contrib
```

Install MetaDefender Core:

bash

```
sudo dnf install -y <filename>
```

Windows package (.msi)

On Windows systems it is possible to install the product by running the corresponding .msi file.

From command line interface it is also possible to install the product by executing

bash

```
msiexec /qn /i <msi file name> <option key>=<option value>
```

where the possible keys and their default values are the following:

Key	Default Value	Description
INSTALLFOLDER	\Program Files\OPSWAT\MetaDefender Core	Customize installation folder for product Example: `INSTALLFOLDER="D:\Products"`
RESTADDRESS	*	REST interface binding IPv4 or IPv6 address ('*' means that service listens on all IPv4 and IPv6 interfaces)
RESTPORT	8008	REST interface binding port
REPORT_ENGINE_ISSUE	true	Enable reporting of engine issue count. (possible values: "true" or "false").

Note

Reporting of engine issue count

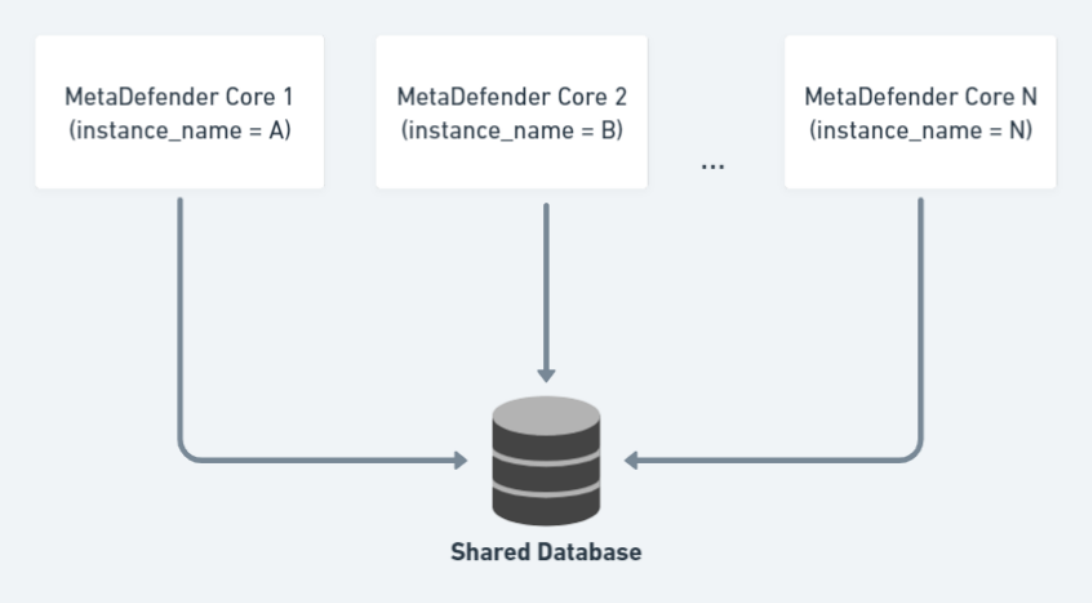
If reporting of engine issue count is enabled, MetaDefender Core server will send only the **number** of initialization errors and **number** of unexpected stops for the specific db/engine version. This information is sent over a HTTPS channel when the product downloads the latest package descriptors. This information is used for early detection of any specific 3rd party engine quality issues.

For details on using msiexec please consult Windows installer documentation.

Shared Database Mode

Starting with MetaDefender 4.20.0, you can use a centralized database deployment model, where all MetaDefender Core instances can share the same external database.

That helps auto-sync all product configurations (**except Single-Sign-On** that requires unique settings for each Core), processing history, live/on-going scan result query, audit, statistics data.



This article demonstrates how to install and setup MetaDefender Core 4.20.0 or newer with shared database mode.

Important notes & current limitations:

Currently not supported

- Does not support Central Management use-case with shared DB mode.
- Does not support to switch from shared database mode back to standalone database mode.
- Does not support to change instance_name of MetaDefender Core in shared database deployment model after installation.
- Does not support upgrading older MetaDefender Core from non-persistent mode to using the shared database mode.
- Does not support upgrading older MetaDefender Core from standalone mode to using the shared database mode.
- Does not support to upgrade multiple MetaDefender Core instances at once in shared database mode
    - Stop all Core instances' scanning service before performing the upgrade.
    - Upgrade each Core server one by one instead.

- For each Core, when the upgrade finished, then you can start using that Core for scanning without waiting for the other Core instances' upgrade to be done.

> **Disclaimer**
> MetaDefender Core does not support mode switching.

Requirements for shared database setup:

- The central remote PostgreSQL server (for shared database) could be utilized with a built-in (native) PostgreSQL database server installed with one of the MetaDefender Core instances in the network. Or using a dedicated existing PostgreSQL database server (see detailed requirements for PostgreSQL version at Recommended System Configuration).
- Before setting up, make sure to follow steps at Open Connection On PostgreSQL Server so that all MetaDefender Core instances could connect to the central remote PostgreSQL server.
- MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core, details: PostgreSQL User Privilege Requirements
- All MetaDefender Core instances must be installed on the same Operating System (Windows / Linux).
- The first MetaDefender Core instance connects to the shared database must be ready first (license activated, users/groups/roles created) before setting up other MetaDefender Core instances to connect and use that shared database.

Engine update:

- If engine updates is configured to be via Folder source, all MetaDefender Core instances must be able to access to the configured update folder, and the option DELETE FILES AFTER IMPORT will be unchecked and prevented for modification.

Authentication:

- Single-Sign-On (SSO) authentication settings (if using) cannot be shared among MetaDefender Core instances because SSO requires an unique login URL for every different Core.
- If an user is locked for login on a certain MetaDefender Core instance(e.g. exceeded the number of failed login times), that user is not prevented for logging in on any other instances.
- If an user is locked for sign-in on a certain MetaDefender Core instance (e.g. exceeded the number of failed login times), if the admin releases lockout on that user on that MetaDefender Core instance, that user is still not able to login on other MetaDefender Core instances.

Sanitized / DLP / Quarantined location changed after MetaDefender Core install:

Windows:

Shared storage path will be stored in 3 strings under key global in registry: quarantinepath, dlppath, sanitizepath as below

(Default)	REG_SZ	(value not set)
dbmode	REG_SZ	4
dlppath	REG_SZ	\\10.40.50.100\test
instance_name	REG_SZ	481rvj
quarantinepath	REG_SZ	\\10.40.50.100\test
restaddress	REG_SZ	*
restport	REG_SZ	8008
sanitizepath	REG_SZ	\\10.40.50.100\test

You can set either 1, 2 or 3 different paths for the 3 settings above (means that quarantined, DLP and sanitized files will be stored in same or different storages corresponding). In case shared storage is not set in installation, values of 3 settings above will be empty.

Restart MetaDefender Core service after changed.

Linux:

Open file `/etc/ometascan/ometascan.conf` and set shared storage path for 3 keys quarantinepath, dlppath, sanitizepath under section [global]

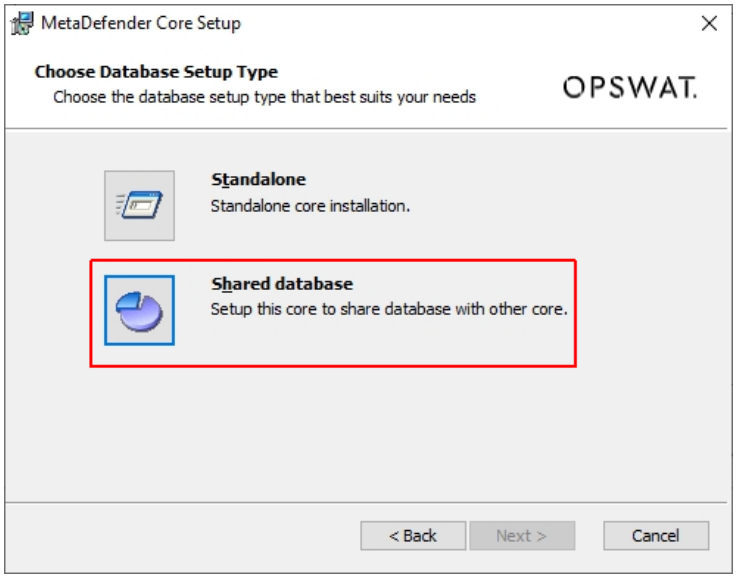
Restart ometascan service after changed.

Wizard Installation (Windows Only)

> **Info**
> Only applicable to MetaDefender Core 4.20.0 or newer using wizard on Windows OS.

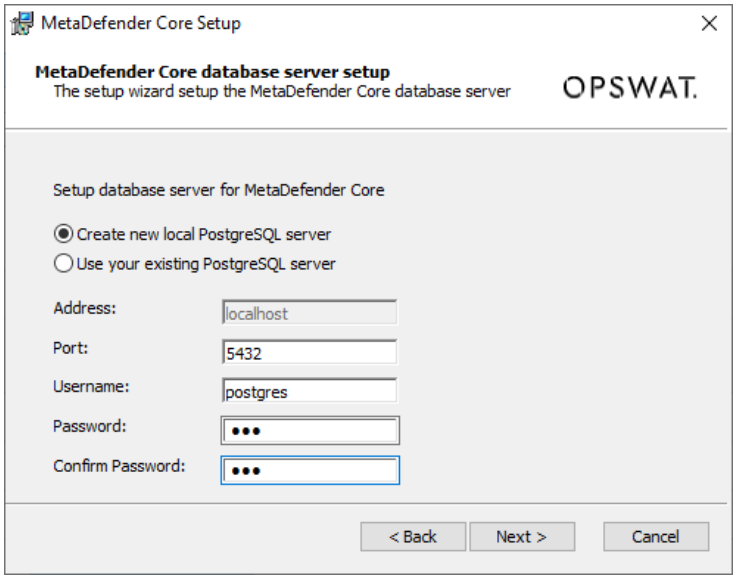
Clean install scenario

1. **Choose Shared database option:**

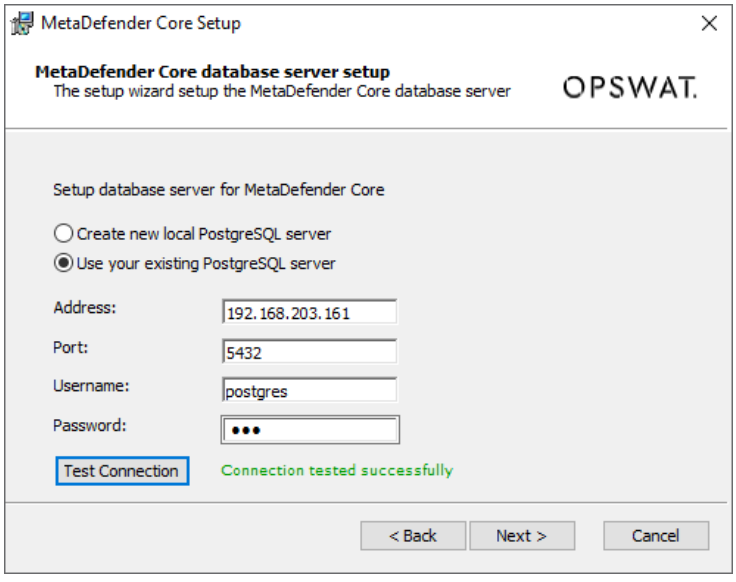


2. **Setup database server for MetaDefender Core:**

You can choose to create a native / built-in PostgreSQL server and setup the shared database on (then other MetaDefender Core instances will want to connect to that PostgreSQL server afterwards)



Or choose to connect to a pre-setup existing PostgreSQL server:



PostgreSQL connection configuration

"Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

PostgreSQL user privilege requirements

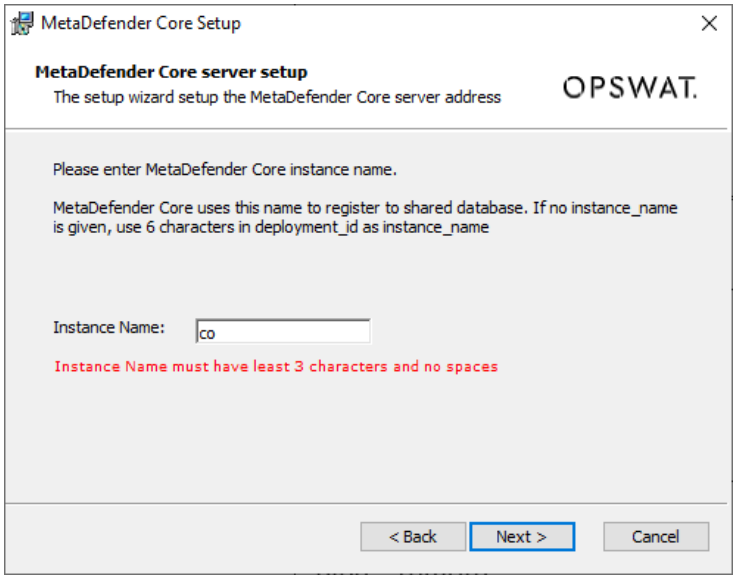
MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.

Details: PostgreSQL User Privilege Requirements

3. **Create an unique instance name for that MetaDefender Core (different from other MetaDefender Core instances):**

Instance name must contain at least 3 characters and no white space.

If you choose to leave it blank, which means it will use the default instance name with 6 characters extracted from that MetaDefender Core deployment ID.

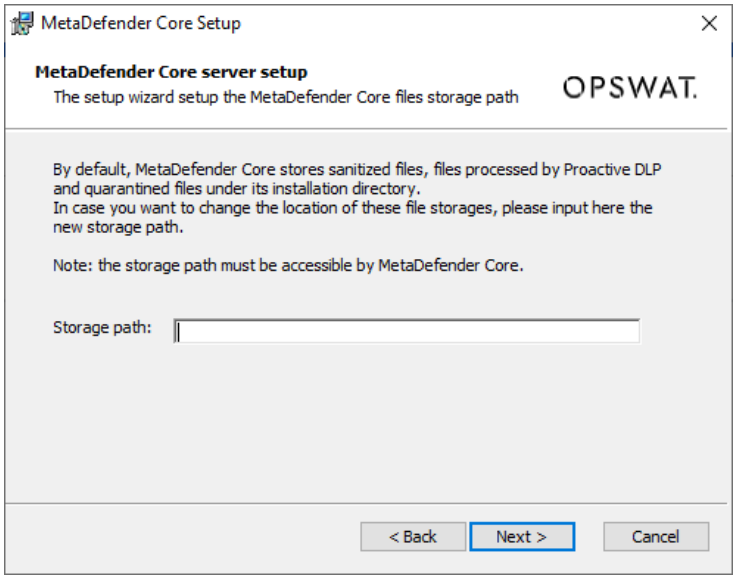


4. **Input the file storage path (sanitized, DLP, quarantined files)**

In the shared database model, all MetaDefender Core instances could use the same folder path to store all their own sanitized, DLP, quarantined files after processed.

The default location is: ***<MetaDefender Core installation folder>\data\ {sanitized|dlp|quarantine}***

If you wish to use that default location, just leave the input box blank and press Next.



Otherwise, you can specify a different location of sanitized/dlp/quarantined files. It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core.

For example:

- C:\new_folder
- \192.168.1.1\new_folder

Command Line Installation

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Clean install scenario

1. Create ignition file if not existed

- Windows: **C:\OPSWAT\ometascan.conf**
- Linux: ***/etc/opswat/ometascan.conf***

An example of ignition file:

ometascan.conf bash

```
eula=true

[global]
dbmode=4
instance_name=MetaDefenderCoreA
storage_path=D:\path\

[user]
name=admin
password=admin
email=admin@local

[dbserver]
type=remote
host=192.168.1.1
port=5432
user=postgres
password=your_own_password
```

Field	Required	Mapping to wizard UI	Possible values
global/dbmode	optional default is 1 (standalone)		1 - Standalone 2 - Central Hub deployment 3 - Non-persistent **4 - Shared**
global/instance_name	optional default is 6 characters of deployment ID		Unicode and number At least 3 characters No whitespace
global/storage_path	optional default is empty		Path to storages (DLP, quarantined, sanitized files) It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core. For example: • C:\folder • \192.168.1.1\folder

2. Run installers using command line to install

Please refer to Command Line Installation for installation command.

Deployment automation support

Note

An Ignition file could be utilized for either / both of following use-cases:

1. Pre-define PostgreSQL server information for MetaDefender Core to connect and setup database: only required on Linux installation, or Windows with command line installation. Learn more details at:
    a. Standalone Mode > Command Line Installation (applicable on version 4.19.0 or newer).
    b. Shared Database Mode > Command Line Installation (only applicable on version 4.20.0 or newer).
2. Automating to accept EULA, create a default local admin user and import MetaDefender Core configurations upon installed: Those steps could be automated using ignition file described in this page.

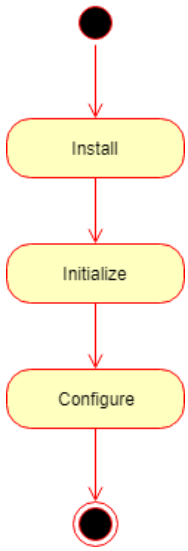
Ignition file location to create:

- **Windows**: `C:\OPSWAT\ometascan.conf`
- **Linux**: `/etc/opswat/ometascan.conf`

The product supports fully automated deployment. It means that it can be installed and configured with no human interaction.

The automated deployment can be split to three steps on a high level:

1. Installation
2. Initialization
3. Configuration



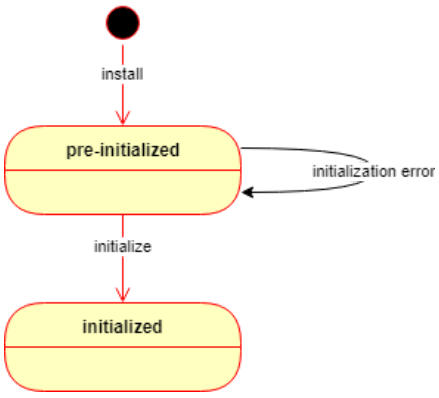
Installation

To automate the installation, install the product from the command line and provide the installation-time options as parameters to the installer. For further details see Command Line Installation

After the installation is complete, the product starts up and waits in a pre-initialized status. The product may be initialized in two ways:

1. Manually using the Wizard Setup or
2. Automatically using an *ignition file* (see below).

> **Warning**
> If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried after fixing the problem (e.g. placing the ignition file in its lookup location) and restarting the OPSWAT MetaDefender Core service.
>
> Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

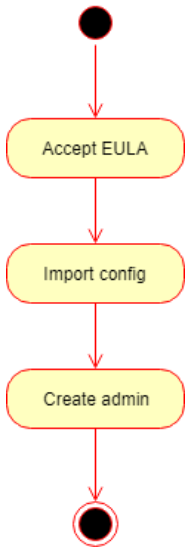


Initialization

Initialization is the process of bringing the product to an operable status.

Basically the initialization consists of the following steps:

1. Accept the End User License Agreement (EULA),
2. Import product configuration and
3. Create the first administrator user account.



Ignition file

The initialization process can be configured in a file called the *ignition file*.

> **Warning**
> In Linux, the ignition file and the folder /etc/opswat should have the permission 755

Ignition file location

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

The ignition file must be in conf format.

ometascan.conf bash

```
eula=true
[user]
name=admin
password=admin
email=admin@local
apikey=44c96cbb4e4e6c0a1d16bcb3c3da24def317
[config]
import=config_package.zip
import_password=123
[security]
session_cookie_samesite=Lax
[internal]
skip_startup_pages=true
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
private_username=my_internal_db_user
private_password=mypassword
[ocm]
ocm_url=https://my.opswat.com
reg_code=XXXXXXXX
```

Ignition file fields

The ignition file accepts the following fields:

Section	Key	Required	Accepted Values	Description
	eula	Optional (mandatory when having [user] section)	true false (default)	Whether to accept the End User License Agreement. This key must be set at the top of ignition file. This key must be set to "true" to accept the EULA. Any other value will cause the initialization to fail. When having [user] section defined, then this key must be set.
[user]		Optional		Initial administrator user account properties. The Administrator role is granted to the account.
	name	Optional	string	User name for the initial administrator user account.
	password	Optional	string	Password for the initial administrator user account. WARNING! Clear text password The password in this configuration file must be stored in its clear-text format and as so it may be visible for unauthorized parties.
	email	Optional	string	E-mail address for the initial administrator user account.
	apikey	Optional	string	API key for the initial administrator user account.
[config]		Optional		
	import	Optional	string	Path to a file in json/zip format that contains a previously exported configuration to be imported. Zip-format file contains a full configuration including user management settings, so it will overwrite [user] section.
	import_password	Optional	string	Password to decrypt .zip configuration package.
[security]		Optional		

Section	Key	Required	Accepted Values	Description
	session_cookie_samesite	Optional	Strict Lax (default) None	Declare if session cookie should be restricted to a first-party or same-site context.
[internal]		Optional		
	skip_startup_pages	Optional	true false (default)	Skip all welcome and other startup pages (e.g. upgrade result) when installing / upgrading MetaDefender, and going straight to the product index default page.
[dbserver]		Optional		
	type	Mandatory when having [dbserver] section	local remote	Where the PostgreSQL database server should physically locate. "remote" means PostgreSQL server and MetaDefender Core are not in the same machine.
	host	Mandatory when having [dbserver] section	string	IP address / domain name of the server where PostgreSQL server locates. "localhost" can be used when applicable.
	port	Mandatory when having [dbserver] section	number	Port of PostgreSQL server is listening for connections from clients (i.e. MetaDefender Core).
	user	Mandatory when having [dbserver] section	string	PostgreSQL server's user. SUPERUSER privilege is required for MetaDefender Core to setup its database and extensions for the first time. **Only non-Unicode characters supported.**

Section	Key	Required	Accepted Values	Description
	password	Mandatory when having [dbserver] section	string	PostgreSQL server's user credentials. **Only non-Unicode characters supported.**
	private_username	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
	private_password	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
[ocm]		Optional		
	ocm_url	Optional	string	URL to My OPSWAT server.
	reg_code	Optional	string	A registration code to enroll to My OPSWAT server.

Detailed initialization process

1) After the product has been started, it looks for the ignition file in the designated location.

Windows: C:\OPSWAT\ometascan.conf ** Linux: /etc/opswat/ometascan.conf

** 2) If an ignition file is found, then

2.1) It gets validated, and if it is valid, then based on the information found in the ignition file:

- The EULA is accepted,
- The configuration is imported,
- The administrator account is created.
- If any of the above steps fails, then the error is logged, and the initialization gets terminated.

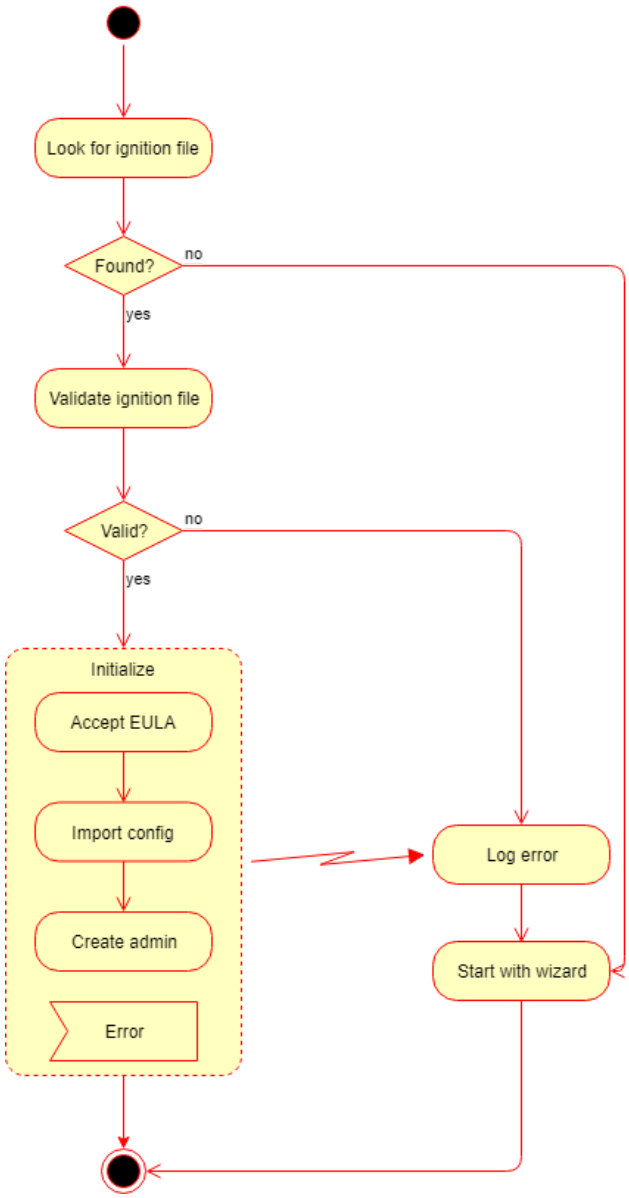
> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

2.2) If it is not valid, then the error is logged, and the initialization gets terminated.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

3) If there is no ignition file, then no initialization is performed.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.



Warning

If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried fixing the problem (e.g. placing the ignition file to its lookup location) and restarting the OPSWAT MetaDefender Core service.

Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

Configuration

After the initialization is complete, the product is ready with the default and the imported configuration.

This configuration can be later changed calling the configuration API functions. For further details about the API see Configuration related APIs API

CIS Level 1 Guidelines

> **Support Red Hat Enterprise Linux 9**
> For more details about Center for Internet Security (CIS) please refer to this document

Instruction steps

I. Install OpenSCAP

bash bash

```
yum install openscap-scanner scap-security-guide
```

II. Generate a result file and a HTML report using OpenSCAP scanner tool

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

III. Remediation of CIS Level 1 issues

Generate a remediation script based on the `ssg-rhel9-ds.xml` file:

bash

```
oscap xccdf generate fix --profile
xccdf_org.ssgproject.content_profile_cis_server_l1 --fix-type bash --output
remediations.sh /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

And execute remediation script.

bash

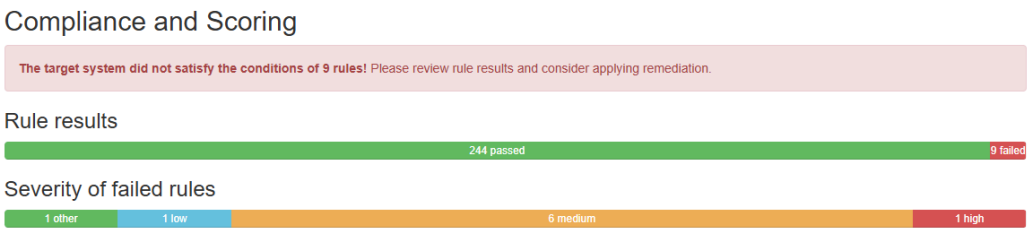
```
./remediations.sh
```

IV. Review the results after remediation

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

Example:



Notes

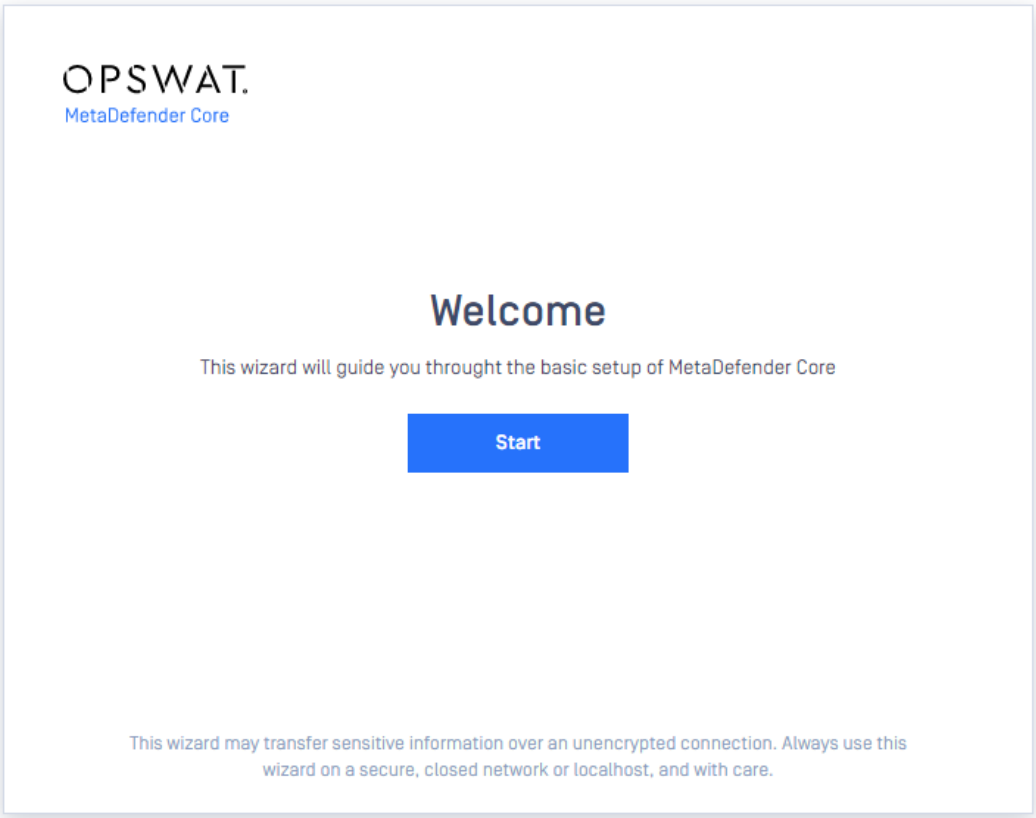
- CIS Level 1 requires `/tmp` folder to be mounted in a separate partition. Please ensure that that new partition have enough disk space for MetaDefender Core to run.
- CIS Level 1 requires that "Ensure No World-Writable Files Exist".
    - For now, when freshly installing MetaDefender Core, all its binary files meet the requirement.
    - When installing/updating engines, some engines might create additional files for its operation, and it might violate this requirement. In this case, you need to again execute the remediation script in the step III.

Wizard Setup

Introduction

Access the MetaDefender Core management console for the first time (i.e. http://localhost:8008), you are to complete a quick walkthrough for wizard setup in order to be able to use the product.

The administrative management console will be available only upon you have successfully finished this wizard stage.

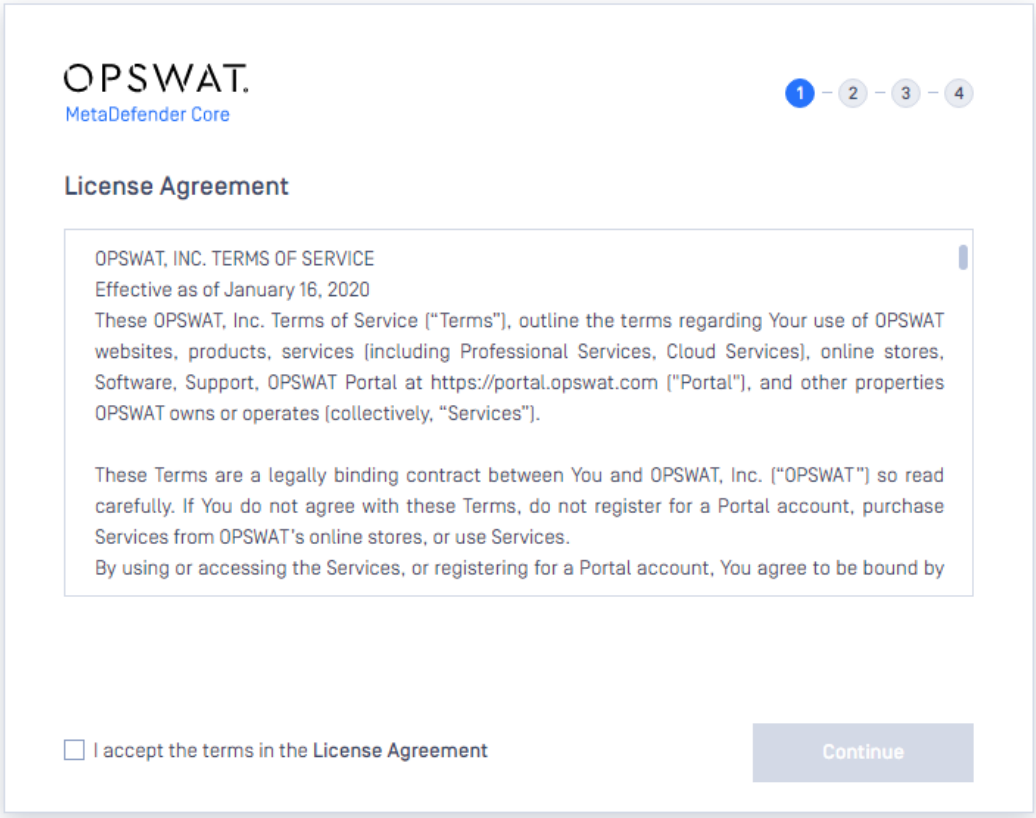


Warning

This wizard may transfer sensitive information over an unencrypted connection. Always use this wizard on a secure, closed network or localhost, and with care.

Basic configuration steps

End-User License Agreement



In the first page you can find the End-User License Agreement. You have to accept the terms before moving on.

Please read through the EULA (https://docs.opswat.com/policies/privacy-policy/portal-privacy-policy) carefully and if you agree with it, check `I accept the terms in the **License Agreement**` and click `Continue` to proceed.

Local Admin User Setup

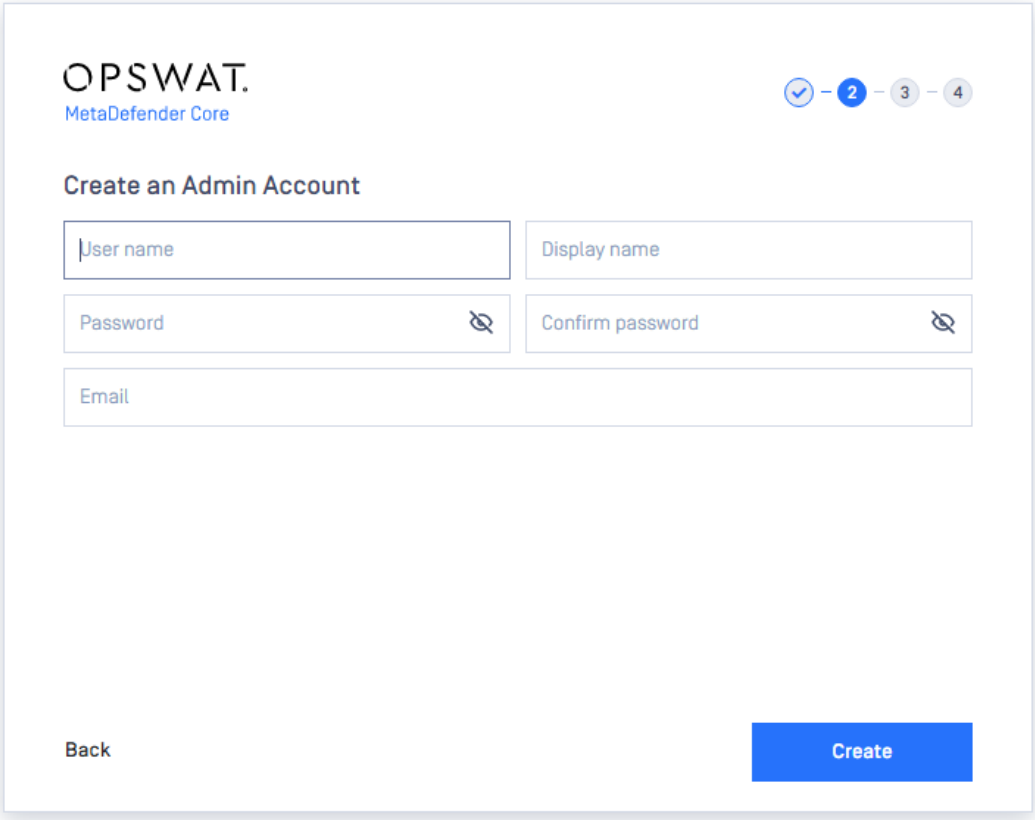
The next step is to set up a default local administrator account.

This account will be the first one being able to access the administrative management console, and to create accounts for other users. You have to fill out all fields in this page to be able to move forward. When you are done, click NEXT to continue.

> **Info - User Directory**
> The administrator account, that is created via the basic configuration wizard, is always added to the LOCAL user directory as a member.
>
> Learn more about user directory: User directories

The following information is required for the administrator account:

ACCOUNT NAME	The unique name of the account that is used at the time of login and in log messages for accountability.
ACCOUNT DISPLAY NAME	Name of the person bound to this account. This name (appended to the name of the account's user directory) is displayed in the top right corner of the administrative management console.
PASSWORD	Password of the user bound to this account that is used at the time of login. Passwords sent clear-text As long as TLS is not configured for the basic configuration wizard, passwords are sent clear-text over the network and may be disclosed to unauthorized parties. As a mitigation action: 1. Either use the wizard on *localhost* or on a direct network connection, or 2. Enabling HTTPS as soon as possible and change the password immediately if it has already been set.
EMAIL	Email address of the person bound to this account.



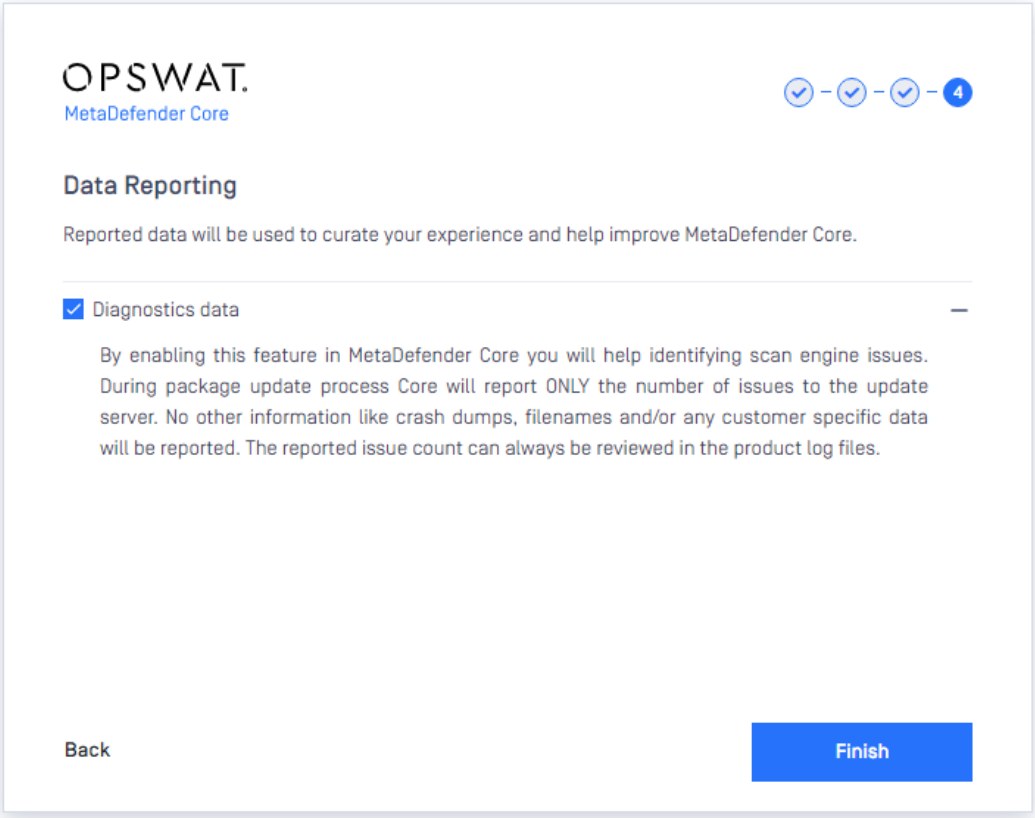
License activation

For license activation details see License Activation

Data Reporting

Enabling this option to allow MetaDefender Core collect its own engine issues and report back to OPSWAT for further analysis.

MetaDefender will not collect any other information.

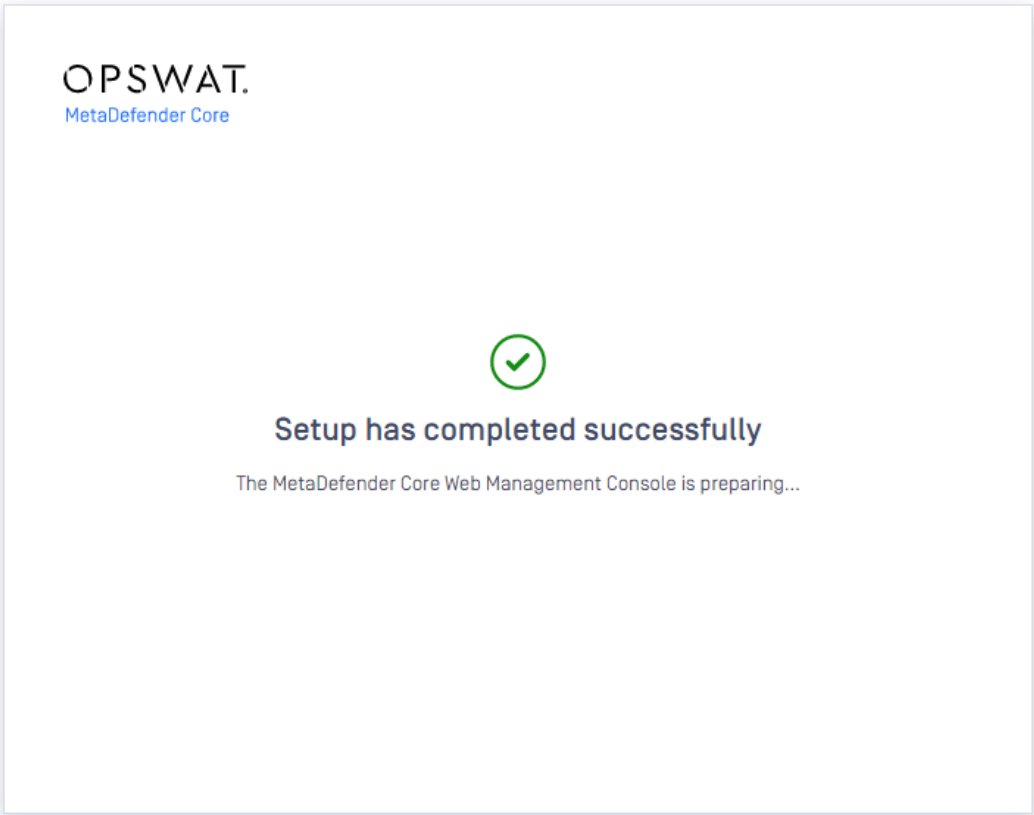


Wizard completion

After you have completed every steps you are ready to finish the wizard and start using the product. Click the `Fishish` button to complete the wizard.

> **Warning**
> The product's service will be restarted and the browser will be redirected to the administrative management console. This could take several seconds.

You can login to the Web Management Console with the administrator user that have just been created in the previous steps.



Setup HTTPS

Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network. Websites, like the Web Management Console, are able to use TLS to secure all communications between their servers and web browsers.

The TLS protocol aims primarily to provide confidentiality (privacy) and data integrity between two communicating computer applications.

Warning

No TLS for the wizard

By default, TLS is not enabled for the basic configuration wizard. As a consequence sessions between the wizard's backend and the browser may be insecure.

Info

Performing the same steps as for the Web Management Console, it is possible to set up TLS for the basic configuration wizard. Remember completing the TLS setup before launching the wizard.

For instructions to set up TLS see Enabling HTTPS

License Activation

##

To activate your installation go to the **Settings > License** menu in the Web Management Console. If you have no valid license, you will only see your installation's **deployment ID**. You will also see a warning in the Web Management Console header.

After successful activation, the product will start downloading the latest available scan engines and malware databases. You can follow the status of the scan engine installation on the **Inventory > Modules** page.

When all the licensed engines are installed, you can start using the MetaDefender Core to process files - File Processing.

Deployment ID

Each MetaDefender Core instance represented by an unique system fingerprint called `Deployment ID`

MetaDefender Core licensing is purely based on deployment ID activated. Therefore, everytime your MetaDefender Core is activated with a different deployment ID, then it will be captured as a different activation record accumulated into your total activation times allowed.

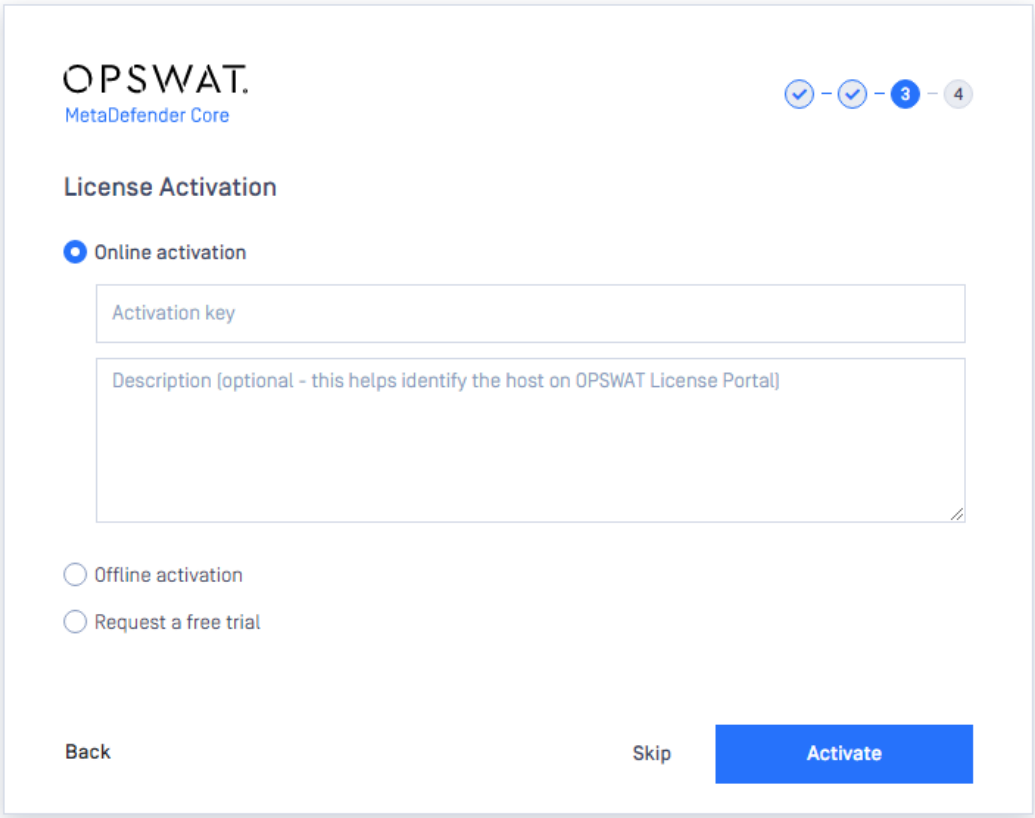
> **Notice**
> Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you alter the system where MetaDefender Core relies on (e.g. network card, OS hostname).
>
> Deployment ID is an UUID version 4 random based string. Learn more at: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you update the system where MetaDefender Core relies on.

Activation Modes

There are three different options for the product activation:

- Online activation: MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.
- Offline activation: Requiring you to upload a license file (.yml). That license file could be retrieved by activating your license via OPSWAT Portal. Follow the displayed instructions for details.
- Request trial key online: For evaluation purpose, you can receive an trial Activation key via email. Follow the displayed instructions for details.



Online activation

MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.

Required to input your Activation License key, and optional for description (but we recommend this for troubleshooting and auditing purpose if needed).

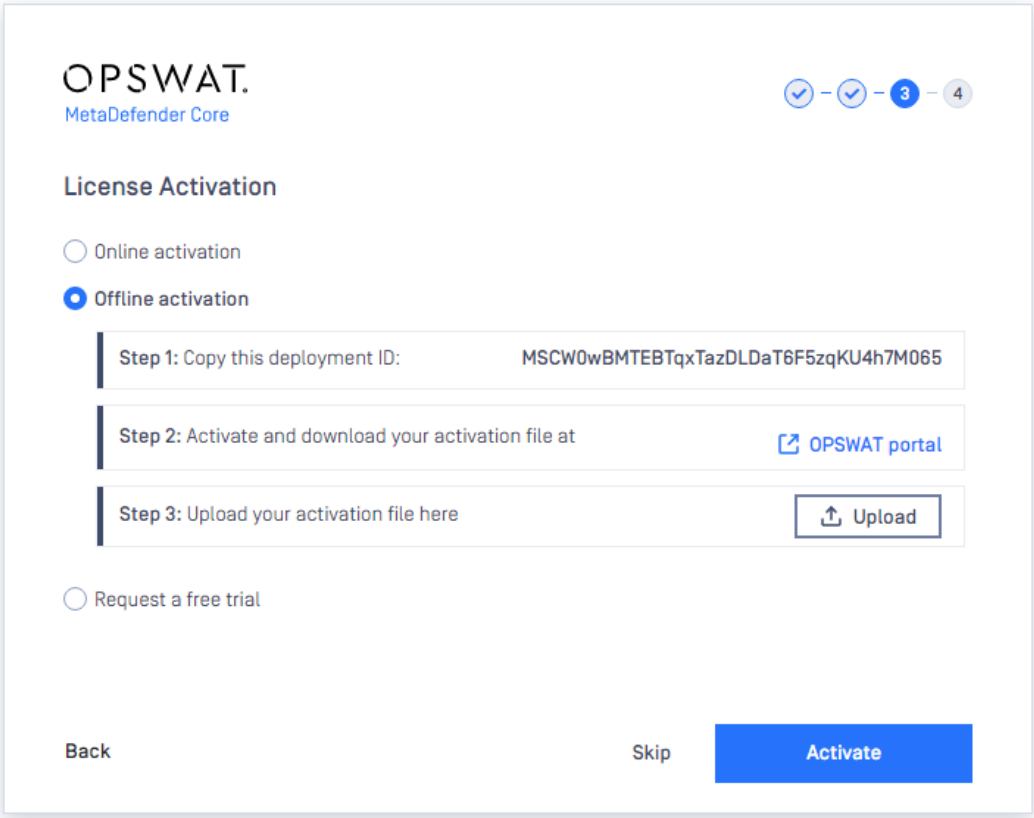
Warning

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts' for license activation:

- https://activation.dl.opswat.com Note: *IP address-based whitelisting on your firewall might fail after some time since OPSWAT uses CDN (Content Delivery Network) to faster delivery updates over the world, and IP address of edge servers might change over time.*

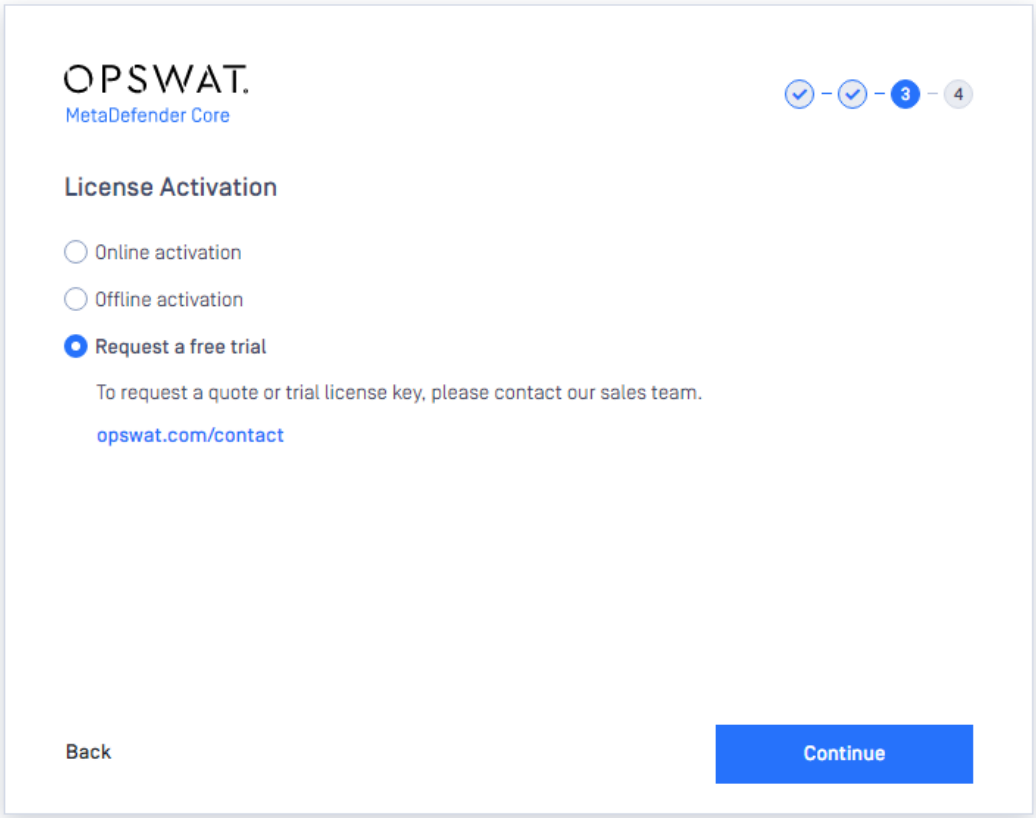
Offline activation

With no internet connection on the server the Metadefender v4 instance may be activated indirectly from a different machine, that has internet connection. The Deployment ID of the Metadefender v4 instance and the the Activation key received at the time of purchasing the product will be required. Follow the steps on the screen to activate the product offline.



Request trial key online

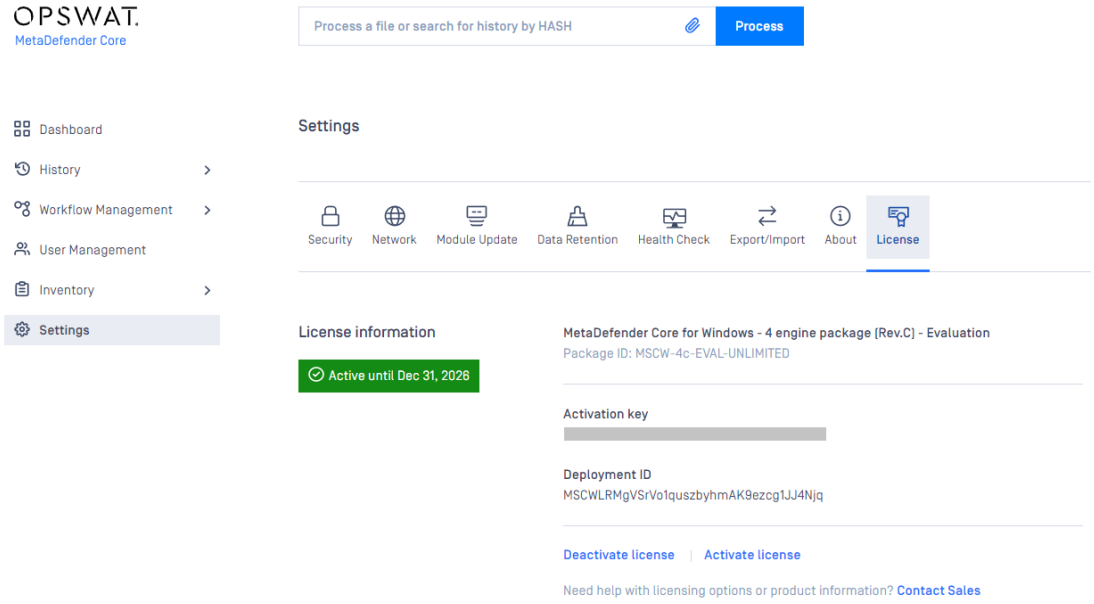
An evaluation license could be acquired by contacting our sales team.



Check Your License Details

For more license details and activating your MetaDefender Core, go to Settings > License menu tab on the Web Management Console:

- Product name: product name as on your order (e.g. MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation)
- Package ID: product identification as on your order (e.g. MSCW-4c-EVAL-UNLIMITED)
- Expiration: last day of license validity
- Activation key: key used for activating product
- Deployment ID: identification of this installation


OPSWAT.
MetaDefender Core
Process a file or search for history by HASH
Process
Dashboard
History
Workflow Management
User Management
Inventory
Settings
Settings
Security
Network
Module Update
Data Retention
Health Check
Export/Import
About
License
License information
Active until Dec 31, 2026
MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation
Package ID: MSCW-4c-EVAL-UNLIMITED
Activation key
Deployment ID
MSCWLRMgVSrVo1quszbyhmAK9ezcg1JJ4Njq
Deactivate license
Activate license
Need help with licensing options or product information? Contact Sales

Scripted license management

Using REST API calls there is an option for scripted activation and deactivation for MetaDefender Core servers with **no Internet connection or if your infrastructure is using automation to create/destroy MetaDefender Core instances**. In this chapter the steps of these two scenarios are described.

Requirements

- an installed MetaDefender Core instance without Internet connection
- another computer that has Internet connection and can run your activation scripts
- a manual or automated way to transfer data between the two computers

Activation steps

1. In order to activate MetaDefender Core, deployment ID and activation key are needed to generate the license file.
    a. Activation key should be purchased from OPSWAT.
    b. MetaDefender Core deployment ID can be queried by using REST API (for details see GET Get Current License Information API page)

> **Info**
> Save this deployment ID in your system, you might need this ID when the instance is unavailable at the time of deactivation.

2. Activate license and get the license file using the following URL, where <quantity> is 1.

activation call bash

```
curl -X GET https://activation.dl.opswat.com/activation?key=<activation-key>&deployment=<deployment unique-ID>&quantity=<quantity>
```

If the activation is successfully, a license file is downloaded. Save this file.

HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	license file	Activation was successfully.
200 Activation failed	error: '<user conform error message>'	Failed activation
200 Invalid parameter	error: 'Could not activate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not activate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
200 Invalid parameter	error: 'Could not activate your product because the quantity you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key, quantity or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Info
In case of any activation issue, contact OPSWAT support for help

3. Upload license file to MetaDefender Core: The license file should be uploaded to the MetaDefender Core v4 to activate the product. For details see page: POST Upload license key API

Deactivation steps

1. For deactivation, following infomation is required:
    a. Activation key
    b. MetaDefender Core deployment ID: can be queried by using REST API (for details see GET Get Current License Information API page)
2. Deactivate license using the following URL:

deactivation call bash

```
curl -X GET https://activation.dl.opswat.com/deactivation?key=<activation-
key>&deployment=<deployment unique-ID>
```

Response HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	result: ok	Successful deactivation
200 No active license found	error: 'Could not found any active license with the given parameters'	The license has not been activated yet or it has been deactivated already.
200 Invalid parameter	error: 'Could not deactivate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not deactivate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Important notes

Warning

Product activation is tied to several hardware, operating system and software parameters. In case of one or more major hardware or software parameter change the product might turn into deactivated status. Operating system updates, other software updates should not affecting the activation status.

If the product is online activated and has live Internet connection then in case of deployment ID change the product reactivates itself. If the product doesn't have Internet connection then the administrator is responsible to reactivate the product.

Because of the above administrator should consider licensing restrictions of the offline deployments if using any virtualization and/or containerization technologies.

File Processing

MetaDefender Core provides REST API interface for file processing business.

There are several ways to process files with MetaDefender Core:

- Analyze Files via Web Interface
- Process Files via REST API

Product Upgrading

> **Info**
> Simply run installer file, details could be found at Installation

> **Info**
> For command line mode, upgrading (e.g. on **Linux**), prior to running the installer file, please make sure to **create the ignition file** with PostgreSQL server information included. Details:
>
> - Standalone DB mode: Command Line Installation (Windows and Linux)
> - Shared DB mode: Command Line Installation

Recommendation for MetaDefender Core upgrade

Our goal is to help you execute the product upgrade continuously, efficiently, and correctly.

PostgreSQL service upgrade

- **Bundled PostgreSQL Service**: If you are using the bundled PostgreSQL server included in the installer, the PostgreSQL service will be automatically upgraded to the newer version (if available). No additional action is required for this process.
- **Remote PostgreSQL Service**: For remote PostgreSQL servers, customers are responsible for managing and upgrading the service independently. Upgrading MetaDefender Core does not update the remote PostgreSQL service.

Before the upgrade

Prepare a proper disaster recovery plan (DRP) including some common practices:

- Backup, or take a snapshot of your current working MetaDefender Core server.
  - This is to help quickly rollback for any upgrade failure might cause later for any reason.
- Optionally backup your PostgreSQL database, especially when using remote shared database mode.
  - Reference: https://www.postgresql.org/docs/current/backup.html
  - Please do it after stopping all linked MetaDefender Core instances to avoid potential data collision.
- Lastly, if backing up entire server is not feasible, then at least export MetaDefender Core configuration package Import/Export configuration just in case you'll need to start off a new server with current MetaDefender Core configuration to keep the scanning service running without any impact.

On the PostgreSQL server, double check server resource to ensure sufficient RAM memory and disk space to be preserved for the upgrade process.

- 8GB RAM available at least during the upgrade.

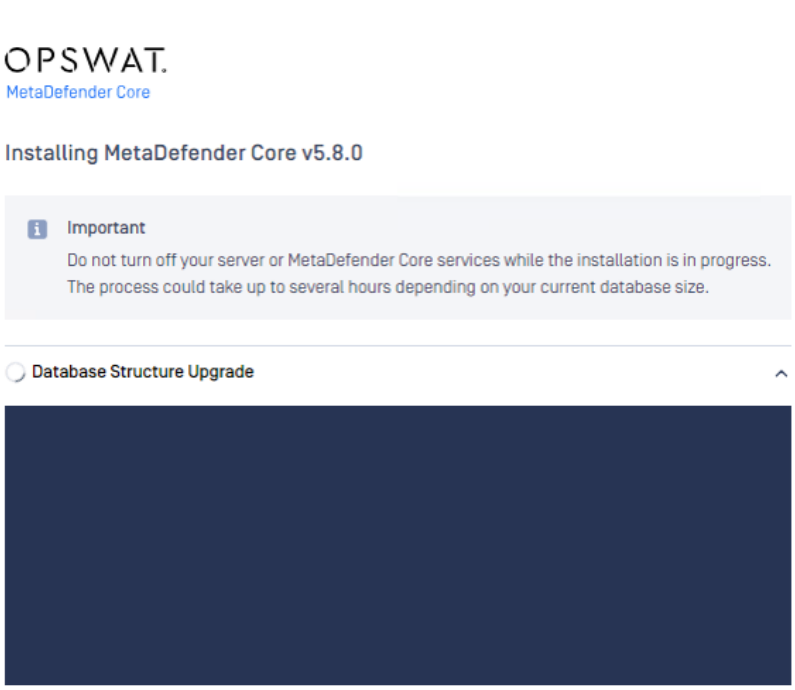
- Disk availability is greater than MetaDefender Core database size.

While doing the upgrade

Applicable for shared database deployment:

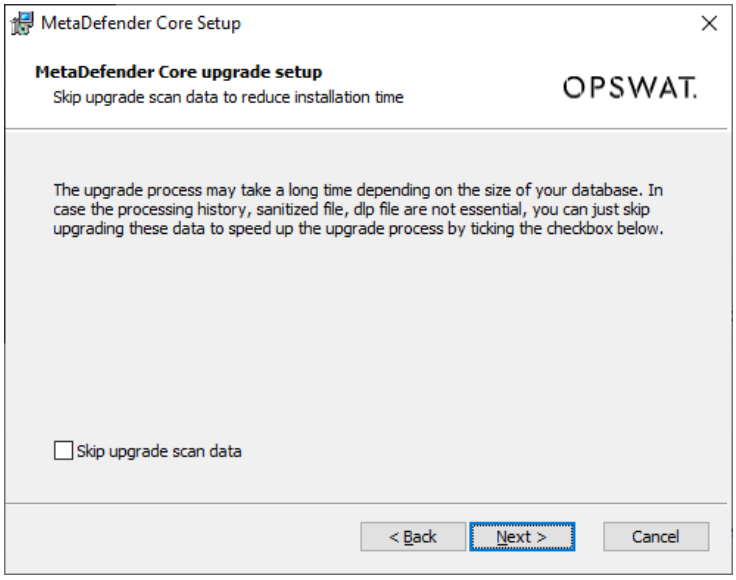
- Stop all MetaDefender Core instances' scanning service before performing the upgrade.
- Upgrade one (1) MetaDefender Core instance at a time.
- Proceed to upgrade MetaDefender Core instance where the bundle local PostgreSQL server is used for shared database setup.
- Otherwise is using remote PostgreSQL then the upgrade order does not matter.

For large size database, mostly for scan history, expecting long time to be spent on the upgrade process.



The product supports an option to skip data migration for scan history to expedite the process drastically and mitigate any failure risk. You can manually backup all data from your PostgreSQL server for offline usage if any.

Check "Skip upgrade scan data" to enable this feature while doing the upgrade.



or add following key in the ignition file:

none

```
[internal]
skip_upgrade_scan_data = 0
```

Troubleshooting

If the upgrade process ever runs into an error:

- Double check your network and services, applicable to remote PostgreSQL server.
- Optionally configure MetaDefender Core to skip migrating scan history.
  - Windows
    - Open Windows registry, and navigate to HKLM\Software\OPSWAT\Metascan
    - Add a new key `internal/skip_upgrade_scan_data` with value = 1
  - Linux
    - Open MetaDefender Core configuration file `/etc/ometascan/ometascan.conf`
    - Add new key into the file
      - [internal]
      - skip_upgrade_scan_data=1
  - Note: This key will be auto removed upon the upgrade finishes.
  - Then restart MetaDefender Core service to retry the upgrade.
    - Might retry this a few times until the upgrade is done.

Just in case the upgrade process still fails, please contact OPSWAT support with the support package - How to Create Support Package? This will help us analyze the root cause and figure out an optimal solution for your system failure.

Using external Load Balancer

Application Load Balancing (Layer 7)

This page provides information about recommended ways to use sessions with a Layer 7 load-balancer.

Most load-balancers have the ability to provide cookies for pinning subsequent traffic from a client to the appropriate server. This method is called sticky session, session persistence or session affinity. Using cookies efficiently requires the client to know when a cookie should be sent, should not be sent or should be deleted.

Sticky session load balancing

Each Metadefender Core instance has its own databases and application sessions that cannot be seen by other instances. Therefore, in order to get the related data/response to our queries we should ask the appropriate Metadefender Core v5 server. On the other hand, to keep the advantages of the used load balancing method, cookies should not be sent if it is not necessary.

Single file scanning

Step	Stage	Task	Cookie usage
1.	Sending file	Initiate processing a file on the client side. Send file through the load-balancer. (See POST File Analysis API)	Cookie should not be sent
2.	Sending file	Save the cookie and the data_id you got from the load-balancer	Save cookie
3.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Send cookie saved in step 2
4.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Send cookie saved in step 2

Batch scanning

Step	Stage	Task	Cookie usage
1.	Open batch	Initiate processing file(s) in batch. Request a batch ID through the load-balancer. (See POST Initiate Batch API)	Cookie should not be sent
2.	Open batch	Save the cookie and the batch_id you got from a Core server through the load-balancer.	Save cookie
3.	Sending files	Send file through the load-balancer. (See POST File Analysis (using batch header) API)	Send cookie saved in step 2
4.	Sending files	Save the data_id you got from the load-balancer.	-
	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Send cookie saved in step 2
5.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
6.	Getting batch status	See GET Batch Status API	Send cookie saved in step 2
7.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Send cookie saved in step 2
8.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API	Send cookie saved in step 2

If it does not matter which upstream server responds, the queries should be sent without cookie.

> **Info**
> It is recommended not to send cookies when it's not necessary to allow load-balancer to use its own method to share the load between Metadefender Core v5 servers.

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration through load-balancer

OPSWAT products that support HTTP load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	4.3.4	-
OPSWAT Client	Windows: 7.6.247.0 Mac: 10.4.243.0	-
MetaDefender Email Security	4.3.0	-
MetaDefender ICAP Server	4.3.0	-
MetaDefender Vault	1.3.0	-
MetaDefender for Secure Storage	2.0.7	-

DNS Load Balancing

Using this method is logically similar to Layer 7 load-balancing.

Briefly how it works

Client uses a domain name to send a query to a server. Client's DNS server has more "A" records for that name with different IPs. When a client resolves the server's domain name DNS server randomly chooses an IP for that name to send back. When a session is used on the application layer, client should know the IP address of the Core server that handles that specific session. In every other case, client should resolve the domain name with DNS query to let requests to be balanced between Metadefender Core servers.

Single file scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Sending file	Initiate processing a file on the client side. (See POST File Analysis API)	Use IP saved in step 2
4.	Sending file	Save the data_id got from the Core server	-
5.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Use IP saved in step 2
6.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Use IP saved in step 2

Batch scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Open batch	Initiate processing file(s) in batch. Request a batch ID. (See POST Initiate Batch API)	Use IP saved in step 2
4.	Sending files	Send file to the specific Core server with the basch ID saved in step 3. (See POST File Analysis (using batch header) API)	Use IP saved in step 2
5.	Sending files	Save the data_id you got from the Core server.	-
6.	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Use IP saved in step 2
7.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
8.	Getting batch status	See GET Batch Status API	Use IP saved in step 2
9.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Use IP saved in step 2
10.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API).	Use IP saved in step 2

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration via DNS load-balancing

OPSWAT products that support DNS load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	does not support yet	-
MetaDefender Client	does not support yet	-
MetaDefender Email Security	does not support yet	-
MetaDefender ICAP Server	does not support yet	-
MetaDefender Vault	does not support yet	-

SSL connection for PostgreSQL communication

> **Note**
> The SSL connection configuration is possible thanks to the support made by PostgreSQL. For reference: https://www.postgresql.org/docs/14/ssl-tcp.html
>
> MetaDefender Core is tested against this mode on particular version 4.21.1

Step 1: Generate SSL certificate for PostgreSQL server

> **Note**
> Following instructions are for self-sign certificate. For production environment, making sure to use the certificate signed by your trusted CA.

1) Generate a private key

bash

```
openssl genrsa -des3 -out server.key 1024
```

Sample output:

bash

```
Generating RSA private key, 1024 bit long modulus ........++++++
...............++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

Type your desired password / pass phrase to encrypt the private key

2) Remove the pass phrase to automatically start up the PostgreSQL server

bash

```
openssl rsa -in server.key -out server.key
```

> **Note**
> On Linux, make sure to set permission on the server.key file

bash

```
chmod og-rwx server.key
```

3) Create a self-signed certificate

bash

```
openssl req -new -key server.key -days 3650 -out server.crt -x509
```

You will be prompted to enter detailed information which is incorporated into your self-signed certificate request.

none

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:CA
Locality Name (eg, city) [Default City]:San Francisco
Organization Name (eg, company) [Default Company Ltd]:OPSWAT
Unit Name (eg, section) []:OPSWAT
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:dummy_email@opswat.com
```

For self-signed certificate, use the server ceriticate as the trusted root certificate:

bash

```
cp server.crt root.crt
```

Copy **server.key**, **server.crt**, **root.crt** to PostgreSQL data folder. For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data**
- Linux: **/var/lib/ometascan/pg_data/**

Step 2: Configure PostgreSQL server for SSL authentication connection

1) Create a custom config file (e.g. ssl.conf) for PostgreSQL.

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\postgres\ssl.conf**
- Linux: **/var/lib/ometascan/postgres/ssl.conf**

2) Content of ssl.conf:

none

```
ssl = on
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
```

3) Modify pg_hba.conf

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data\pg_hba.conf**
- Linux: **/var/lib/ometascan/pg_data/pg_hba.conf**

Add following (modify **::1/128** if needed to match with your environment requirement)

none

```
hostssl all all      ::1/128      scram-sha-256
```

After the modify, the content of pg_hba.conf should be like this (example):

> **Note**
> In a Linux environment, a local connection is required to upgrade from version 5.5.1 or earlier to version 5.6.0 or later.

none

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local         DATABASE  USER  METHOD  [OPTIONS]
# host          DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl     DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostgssenc    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnogssenc  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
local    all all                    scram-sha-256
hostssl all all      ::1/128      scram-sha-256

#More content.......
```

4) Reload the services

- Standalone DB mode: Restart **ometascan** service (MetaDefender Core service)
- Shared DB mode: Restart **ometascan-pg** / **ometascan-postgresql** service (MetaDefender Core PostgreSQL service) and then **ometascan** service (MetaDefender Core service)

Test your SSL database connection:

bash

```
psql.exe "sslmode=require dbname=postgres user=postgres host=localhost port=5432"
Password for user postgres:
psql (12.7)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
Type "help" for help.
```

This message indicate that you are now using SSL connection:

bash

```
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
```

Performance and Load Estimation

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Some factors that affect performance

- MetaDefender Core version
- MetaDefender Core engine package and configuration
  - set of engines (which and how many)
  - product configuration (e.g., thread pool size)
- System environment
  - server profile (CPU, RAM, hard disk)
  - client application location - remote or local
  - system caching and engine level caching
- Dataset
  - encrypted or decrypted
  - file types
    - different file types (e.g., document, image, executable)
    - archive file or compound document format files
  - file size
  - bad or unknown (assume to be clean)
- Performance tool

How test results are calculated

Performance (mainly scanning speed) is measured by throughput rather than unit speed. For example, if it takes 10 seconds to process 1 file, and it also takes 10 seconds to process 10 files, then performance is quantified as 1 second per file, rather than 10 seconds.

- total time / total number of files processed: 10 seconds / 10 files = 1 second / file.

Standalone DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.2.416.0

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

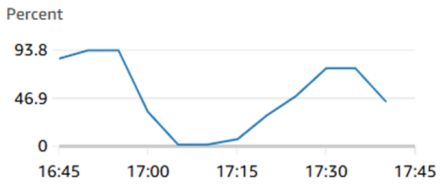
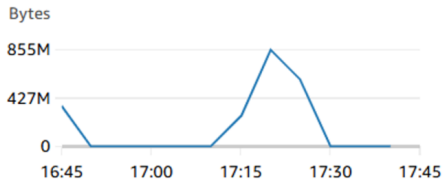
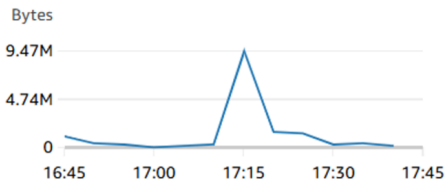
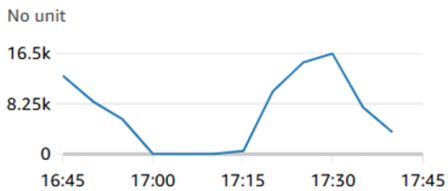
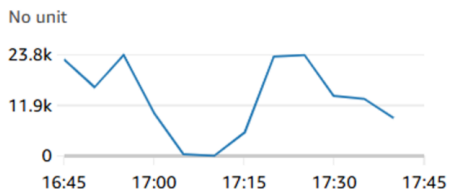
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV)	24.2	47.2 / 80	37.3 / 41.1		315,310	0.0114
Deep CDR	19.9	48.9 / 81.8	32.6 / 37		383,442	0.0094
Proactive DLP	17.1	36.8 / 58.2	30.4 / 33.6		446,228	0.0081
Vulnerability	10.1	59.2 / 85	29.3 / 33.3		755,495	0.0048

MetaDefender Core with common engine packages:

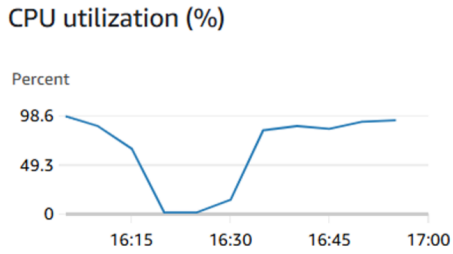
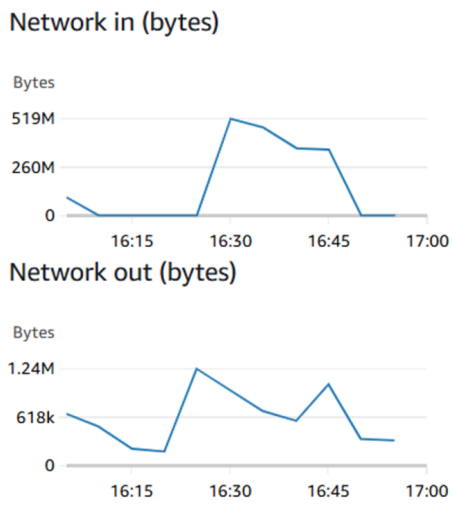
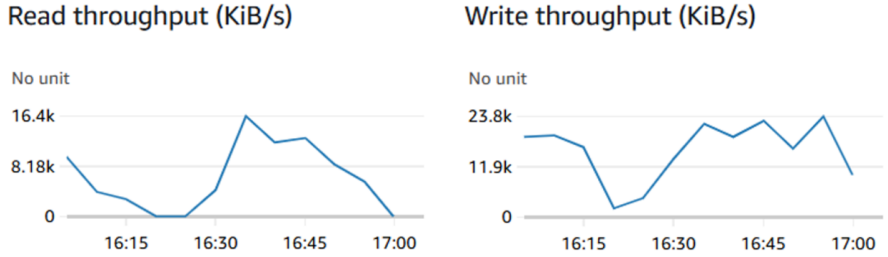
Use case	Scan duration	Avg./Max CPU usage	Avg./Max RAM usage	Avg./Max Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV) + Deep CDR	30.8	68.1 / 88	40.1 / 46.8		247,744	0.0145
Metascan (8 AV) + Deep CDR + Proactive DLP	31.2	74.2 / 98.6	41.5 / 51.1		244,567.3	0.0147
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	31.8	73.5 / 98	40.6 / 51.5		239,953	0.015

System resource utilization charts

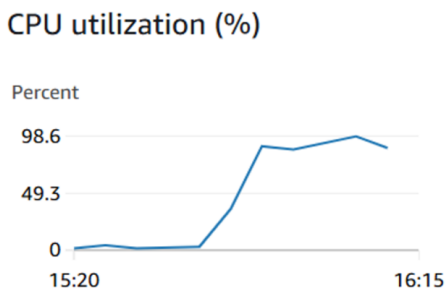
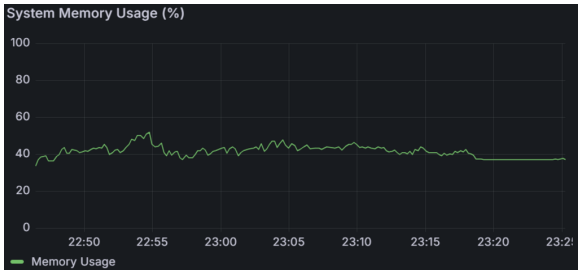
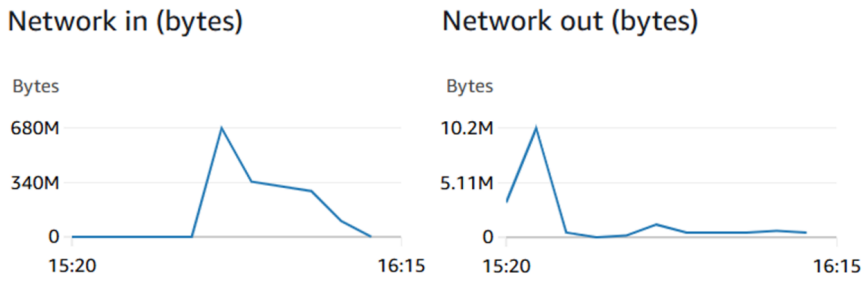
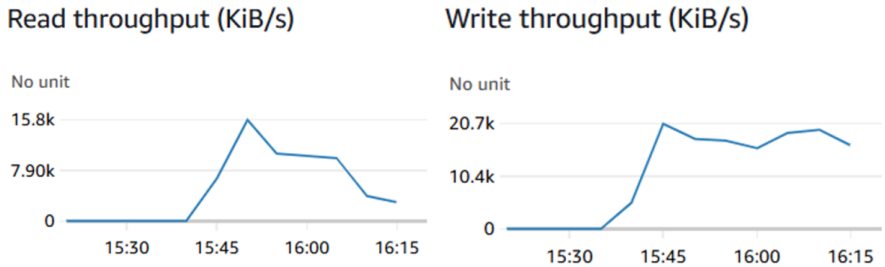
MetaDefender Core with Metascan (8 AV engines):

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

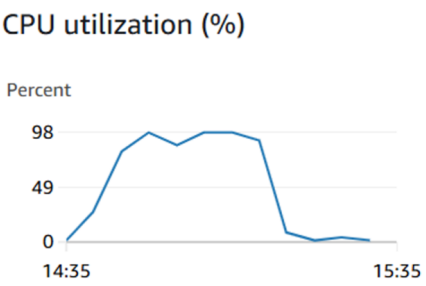
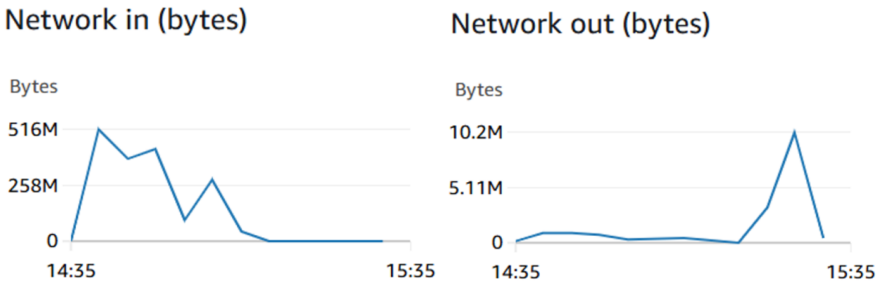
MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

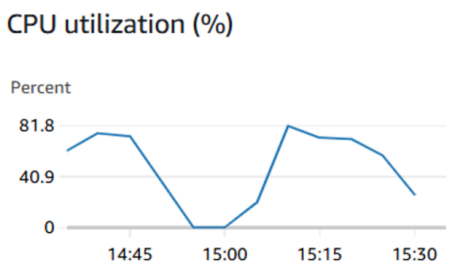
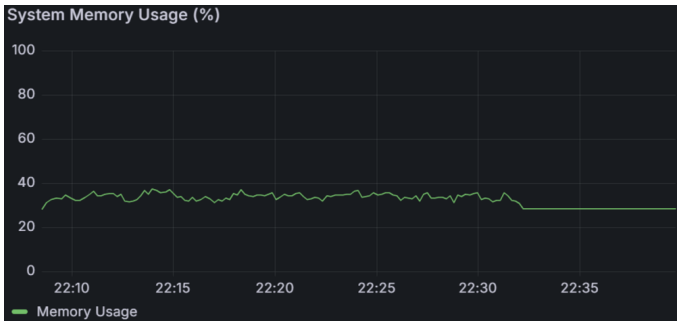
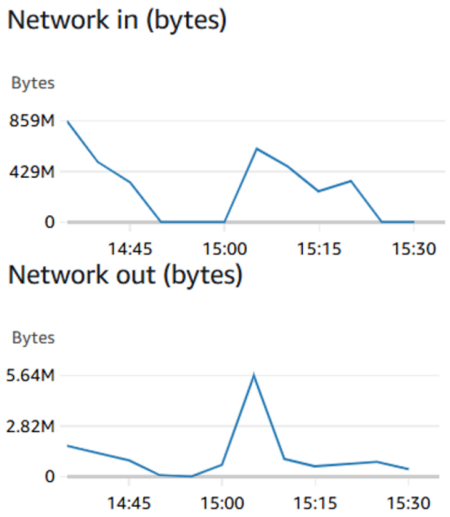
MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

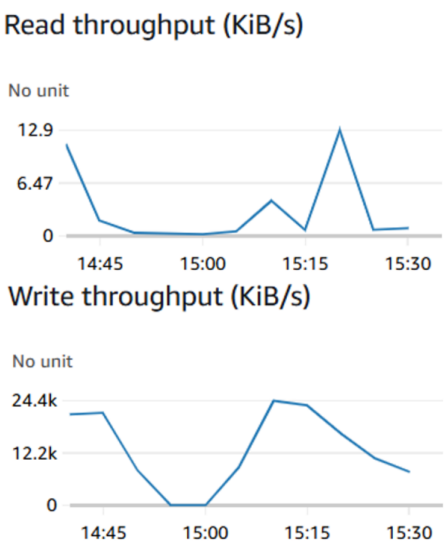
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

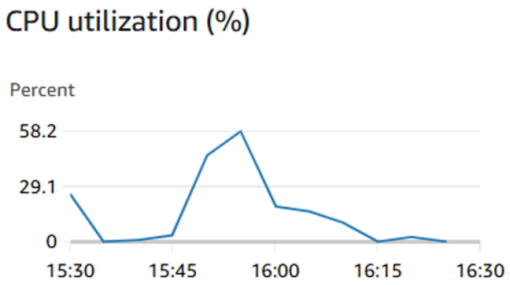
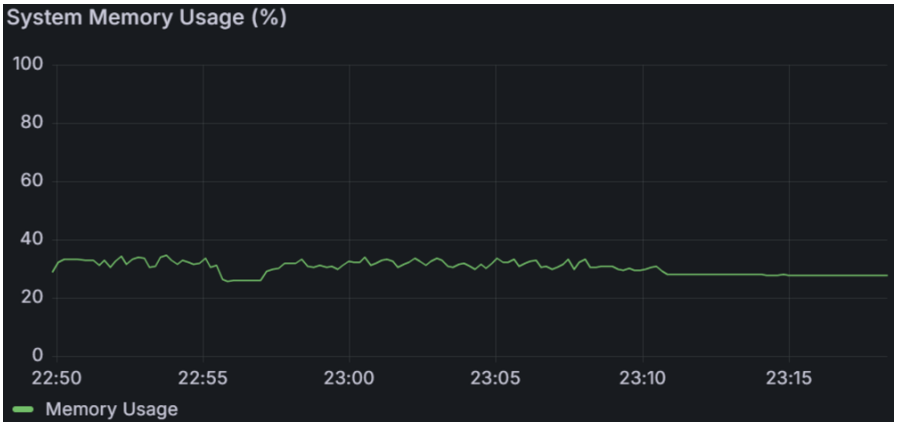
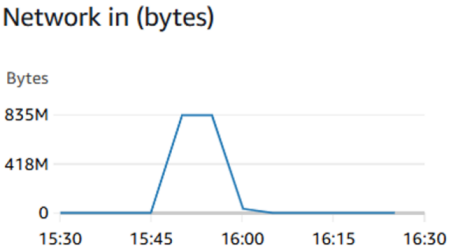
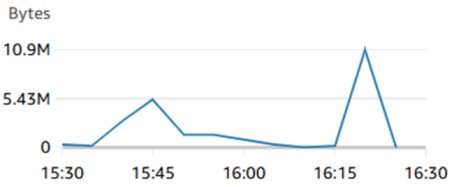
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

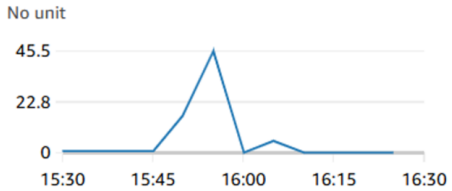
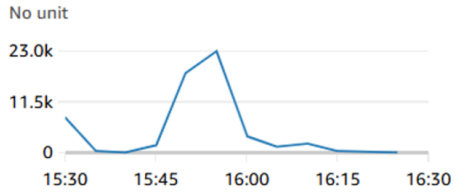
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

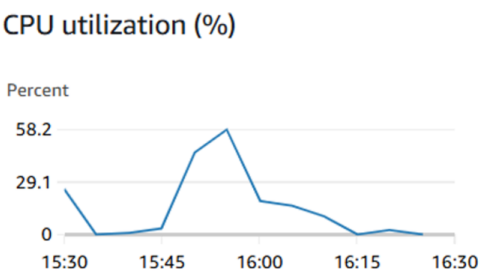
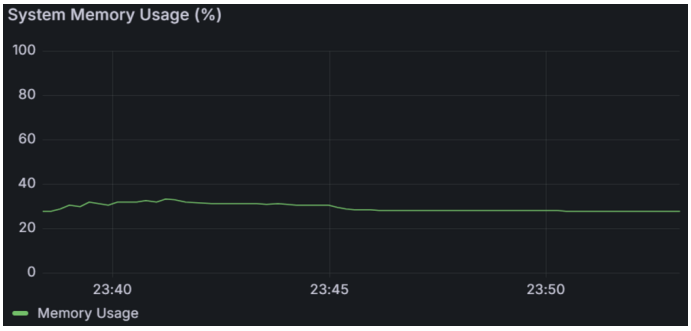
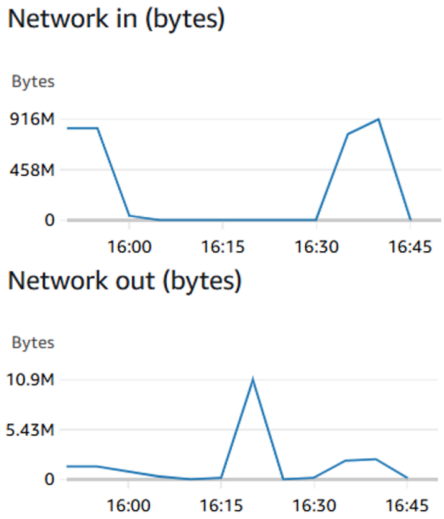
	Utilization charts
Disk usage	

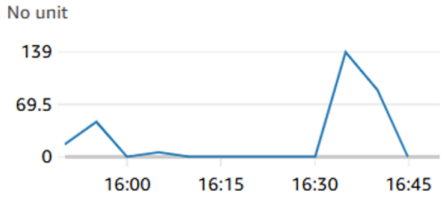
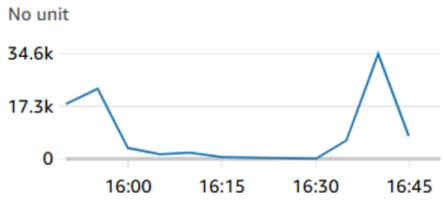
MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	 

	Utilization charts
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

MetaDefender Core with Vulnerability:

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

Standalone DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Ubuntu 22.04	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.57-236

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

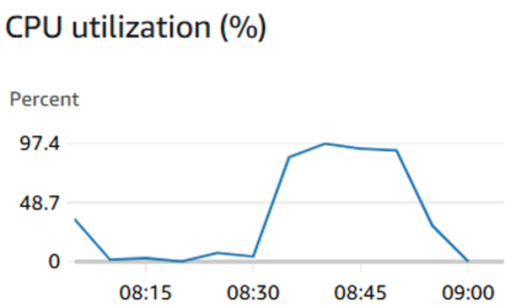
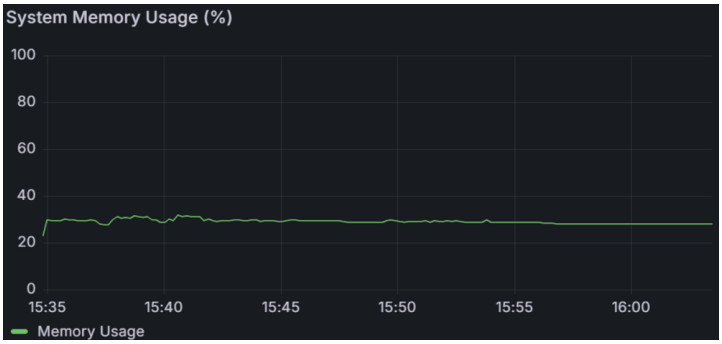
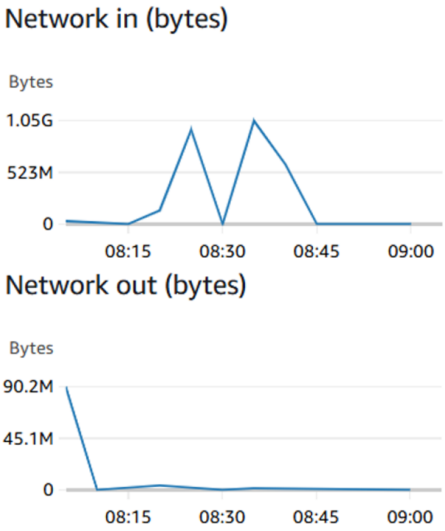
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV)	30.2	69 / 98	29 / 32		252,666	0.0142
Deep CDR	19.4	48 / 62	36 / 49		393,325	0.009
Proactive DLP	16.5	34 / 39	32 / 48		462,455	0.008
Vulnerability	12	50 / 65	46 / 46		635,875	0.006

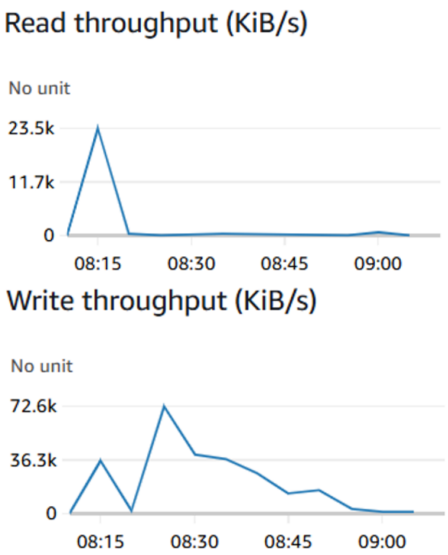
MetaDefender Core with common engine packages:

Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV) + Deep CDR	39.8	78 / 99	62 / 73		191,721	0.0188
Metascan (10 AV) + Deep CDR + Proactive DLP	43.5	61 / 84	47 / 67		175,414	0.0205
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	44.8	62 / 88	69 / 83		170,324	0.0211

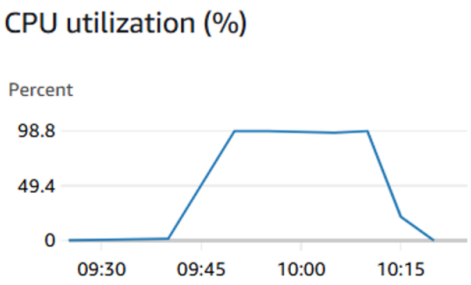
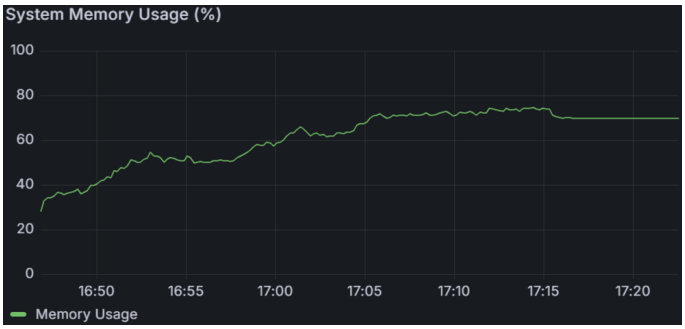
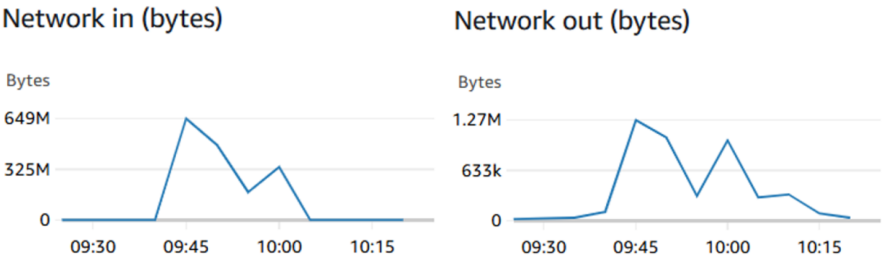
System resource utilization charts

MetaDefender Core with Metascan (10 AV engines) :

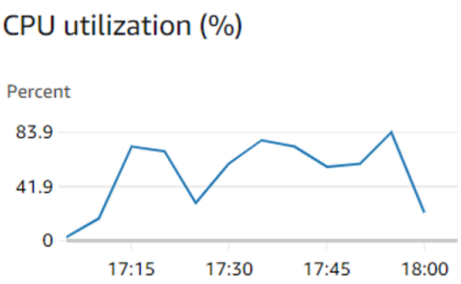
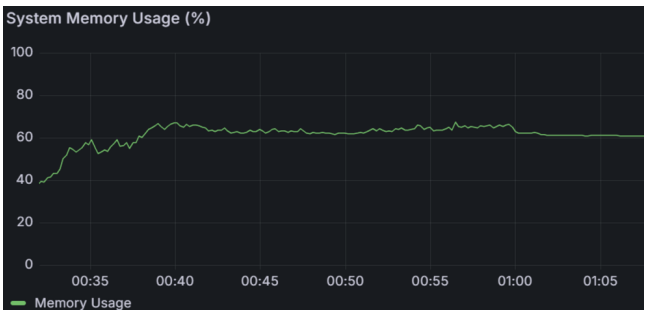
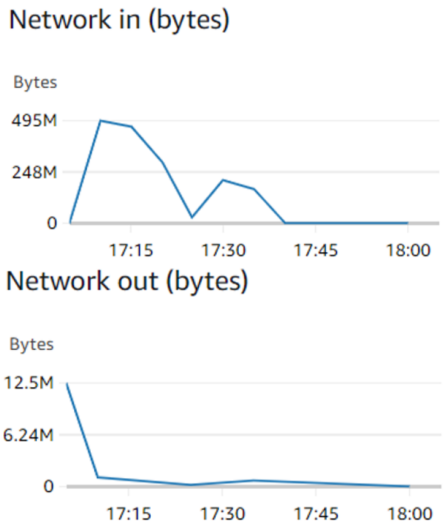
	Utilization charts
CPU usage	
Memory usage	
Network usage	

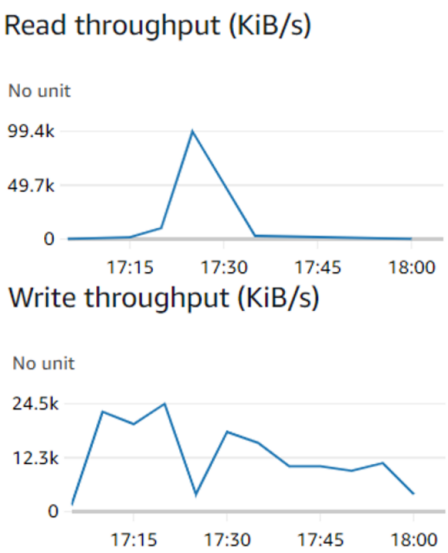
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR :

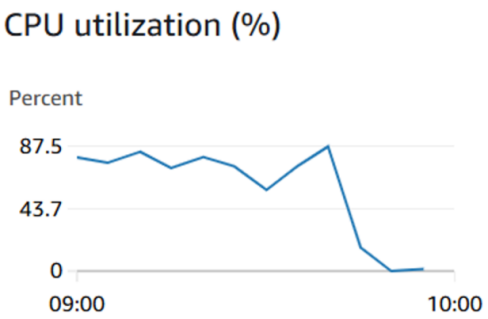
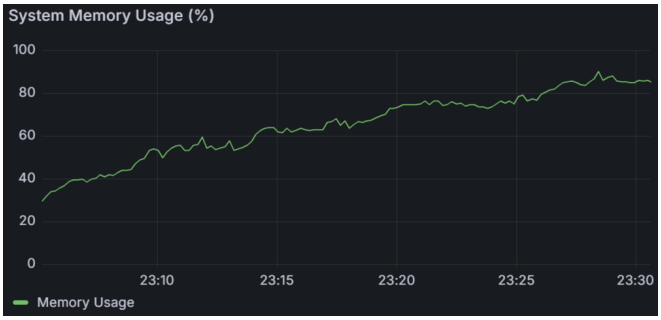
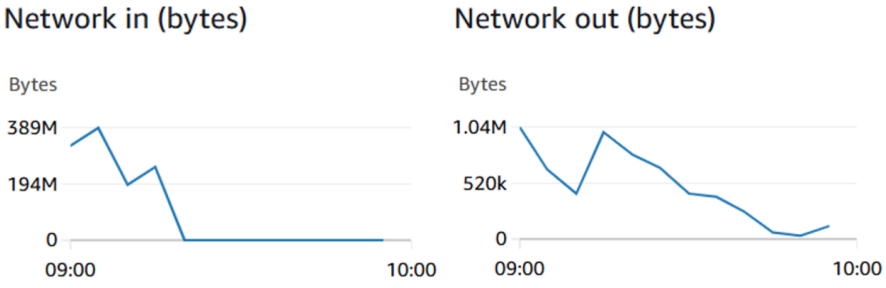
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP :

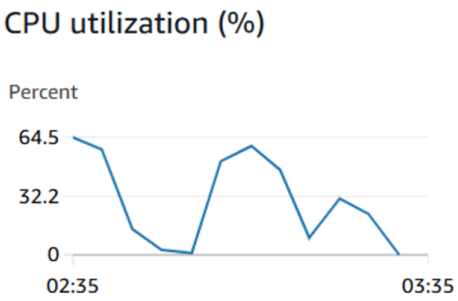
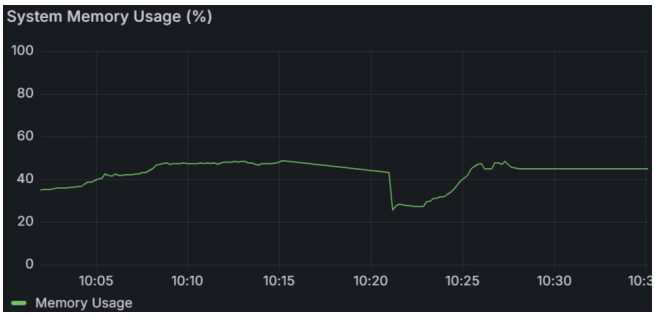
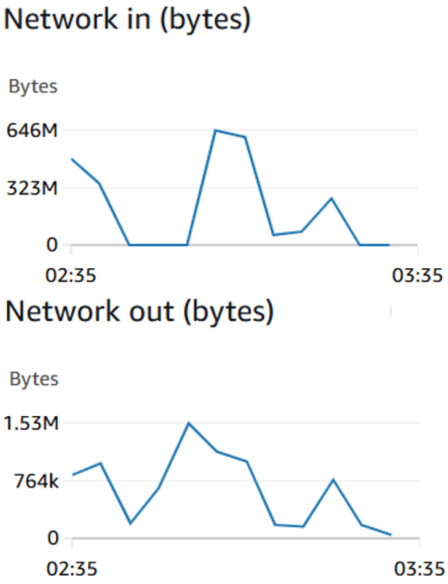
	Utilization charts
CPU usage	
Memory usage	
Network usage	

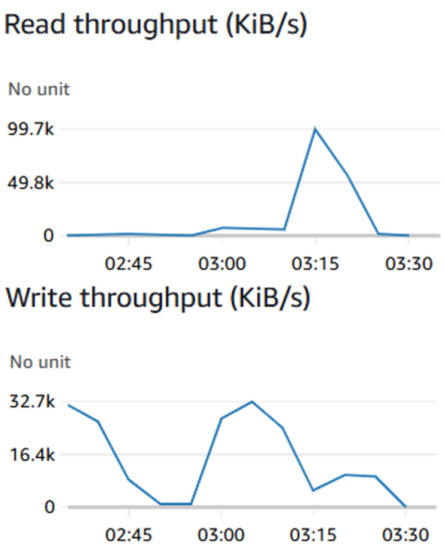
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP + Vulnerability :

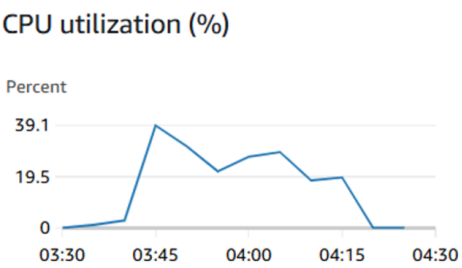
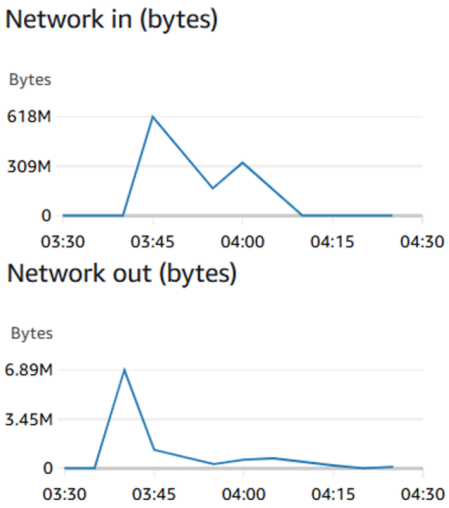
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

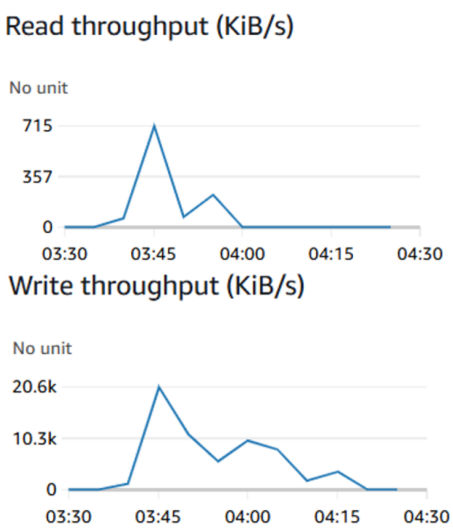
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

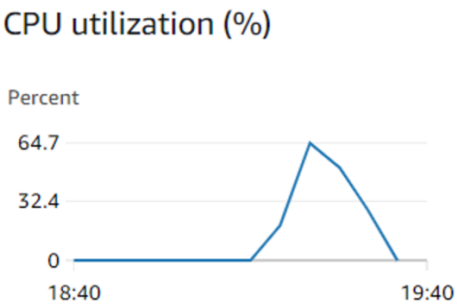
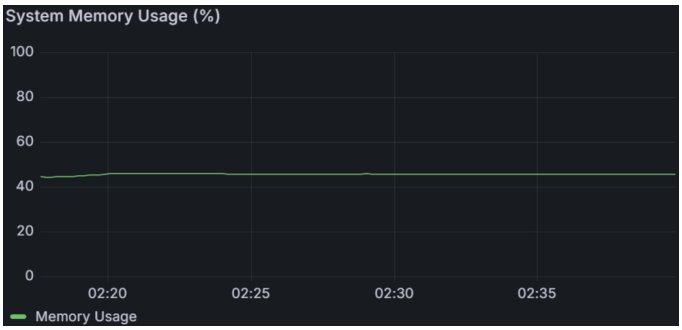
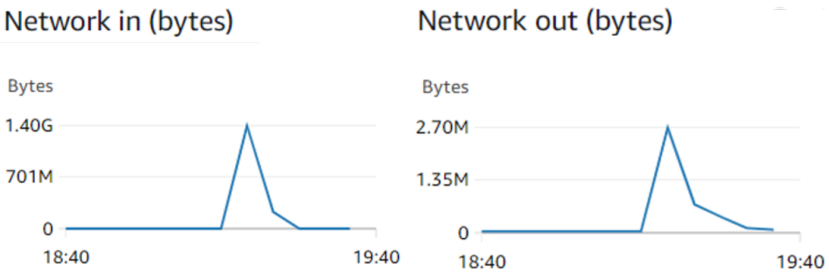
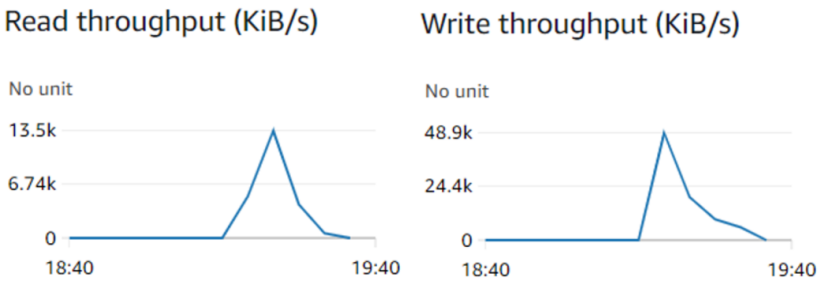
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability :

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

Shared DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

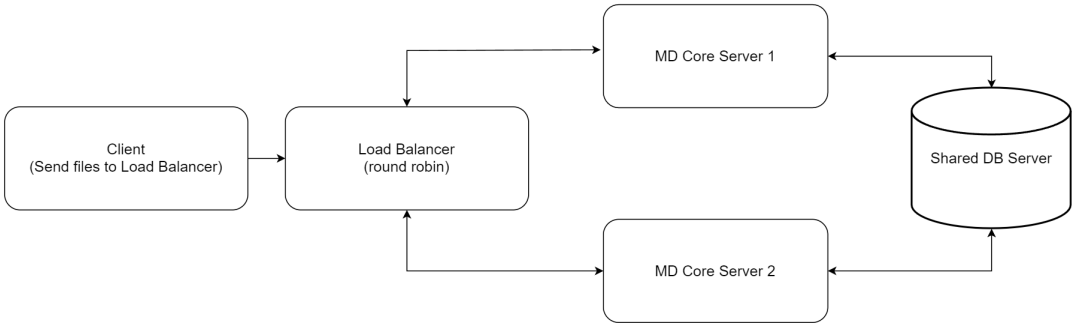
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV)	11.9	37.0	41.9	30.1	30.4	Receive: 6441.1 Send: 769.6	Receive: 6200.2 Send: 693.7	640,821.2	0.0
Deep CDR	10.5	60.1	66.7	30.3	32.2	Receive: 6379.8 Send: 594.8	Receive: 7146.1 Send: 609.6	729,368.5	0.0
Proactive DLP	7.8	38.7	37.5	29.7	31.0	Receive: 8842.6 Send: 791.8	Receive: 8635.6 Send: 729.1	965,455.7	0.0
Vulnerability	4.4	52.2	53.0	26.0	26.9	Receive: 16,215.5 Send: 1006.8	Receive: 15,785.7 Send: 997.2	1714,449.3	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV) + Deep CDR	15.4	72.4	75	34.3	35.4	Receive: 5436.6 Send: 611.3	Receive: 5589.3 Send: 607.2	495,697.4	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP	16.9	77.2	80.2	36.4	37.9	Receive: 5254 Send: 585.4	Receive: 5041.3 Send: 576.1	450,657.6	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	18.0	72.1	80.4	38.6	38.0	Receive: 4504.3 Send: 574.1	Receive: 4649.5 Send: 571.1	423,010	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

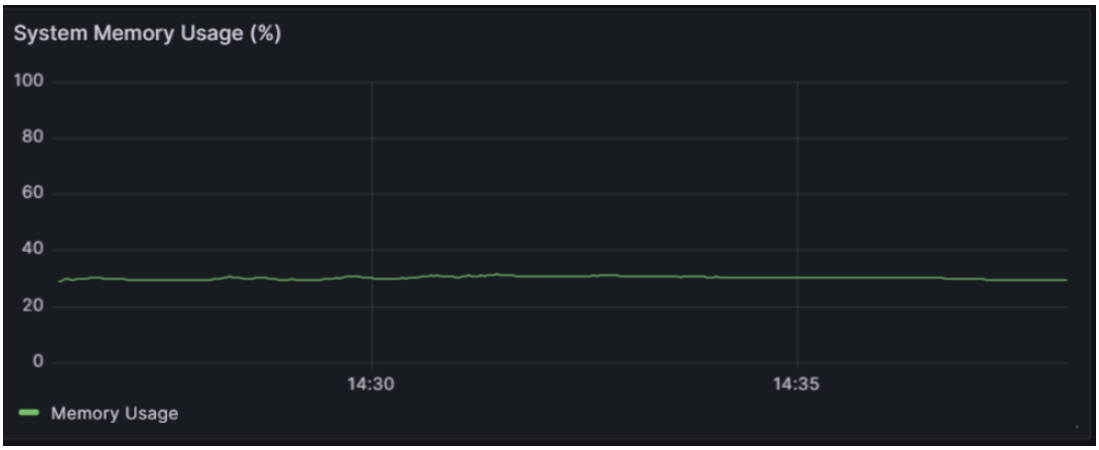
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2

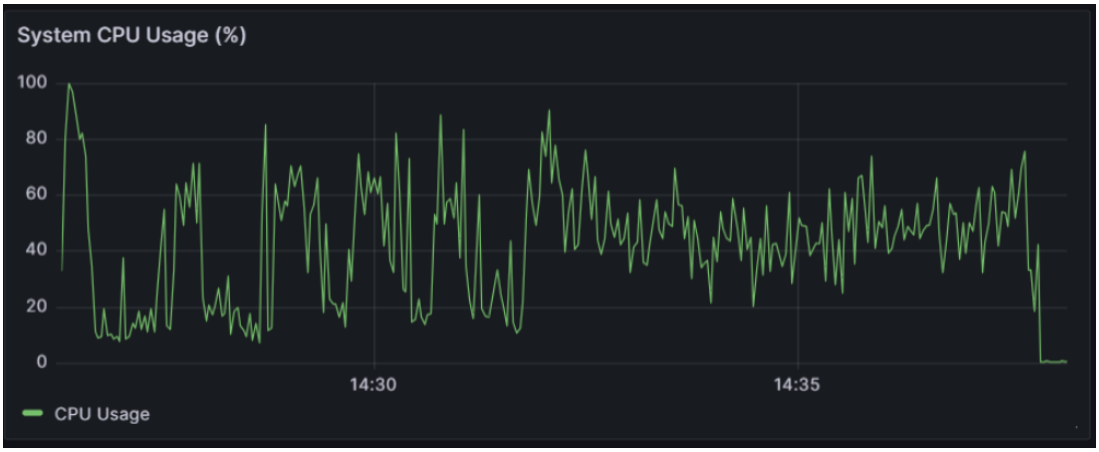


CPU usage

MD Core #1

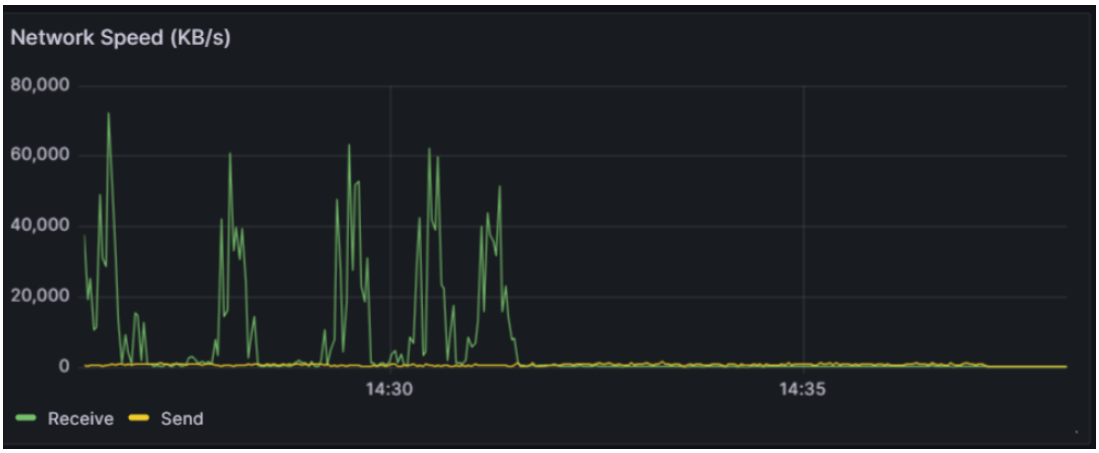


MD Core #2

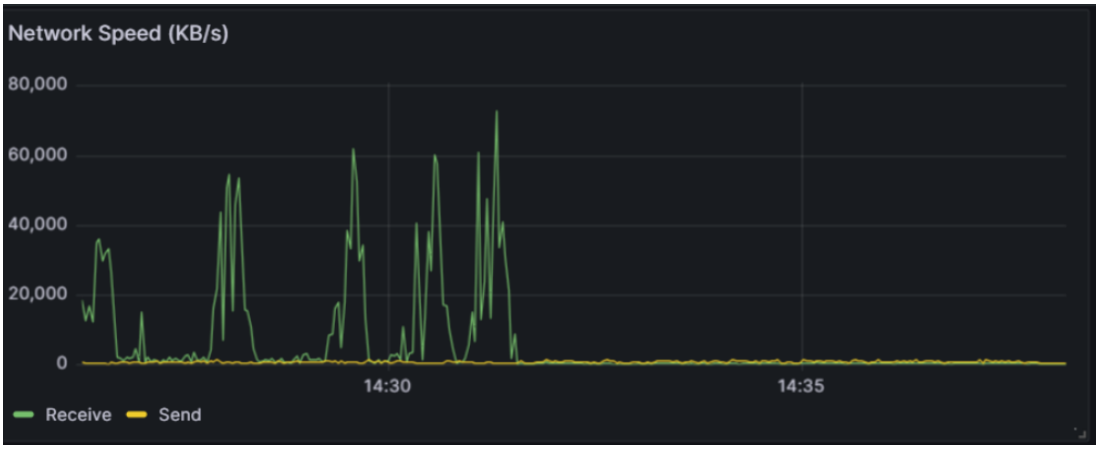


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2

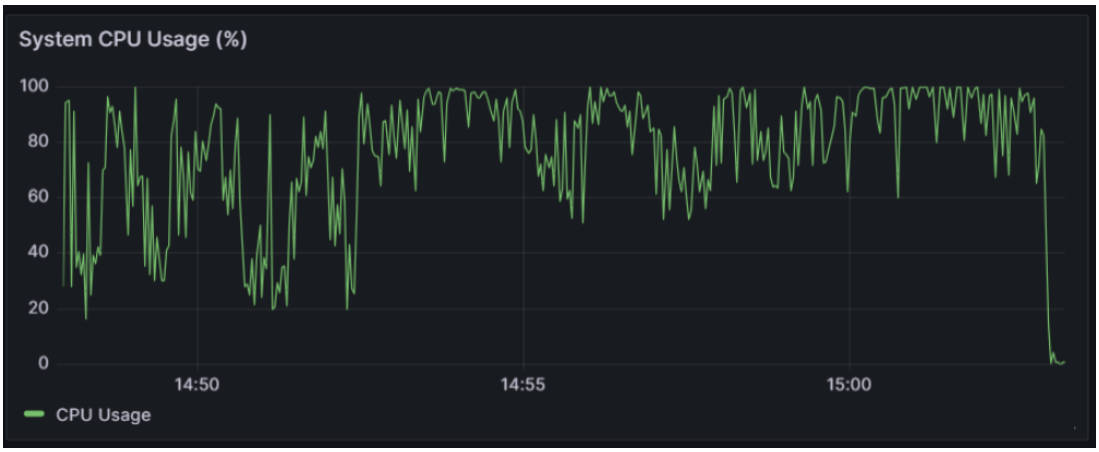


CPU usage

MD Core #1

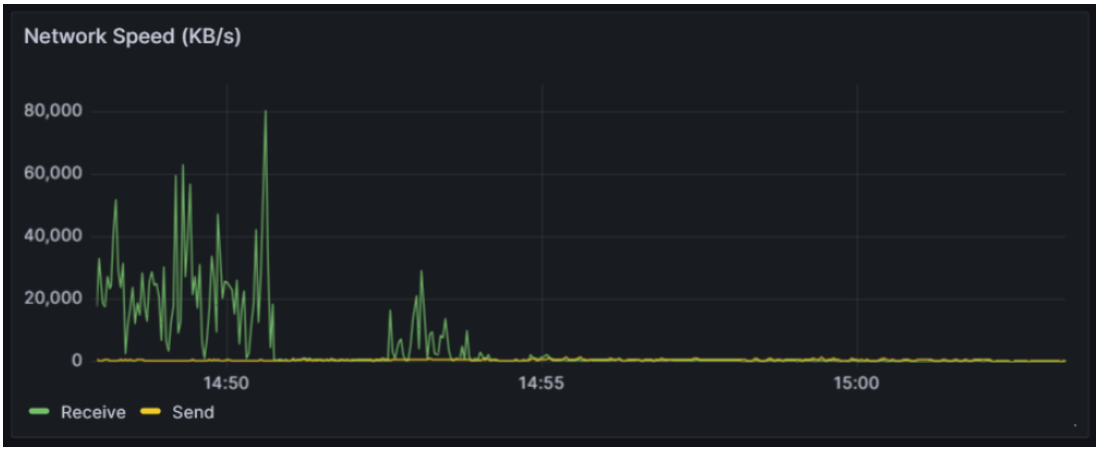


MD Core #2



Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

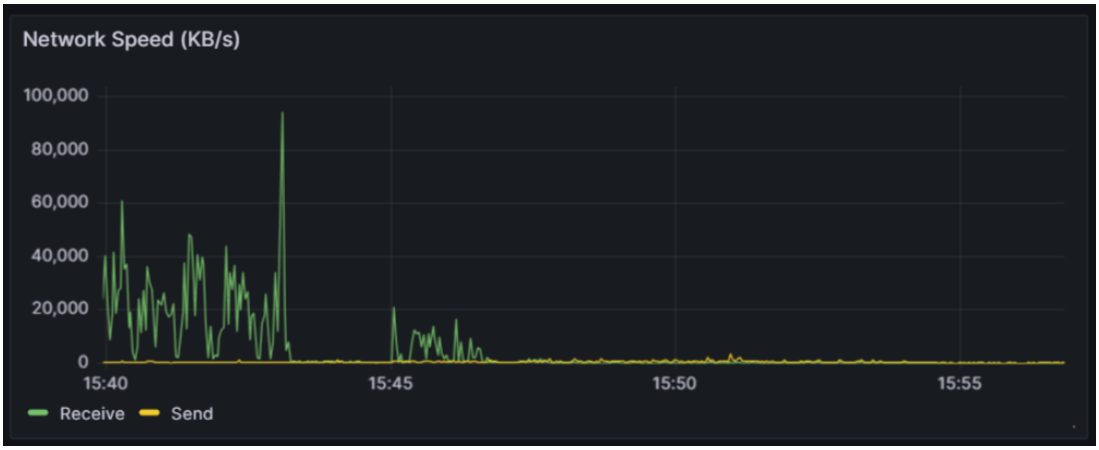


MD Core #2

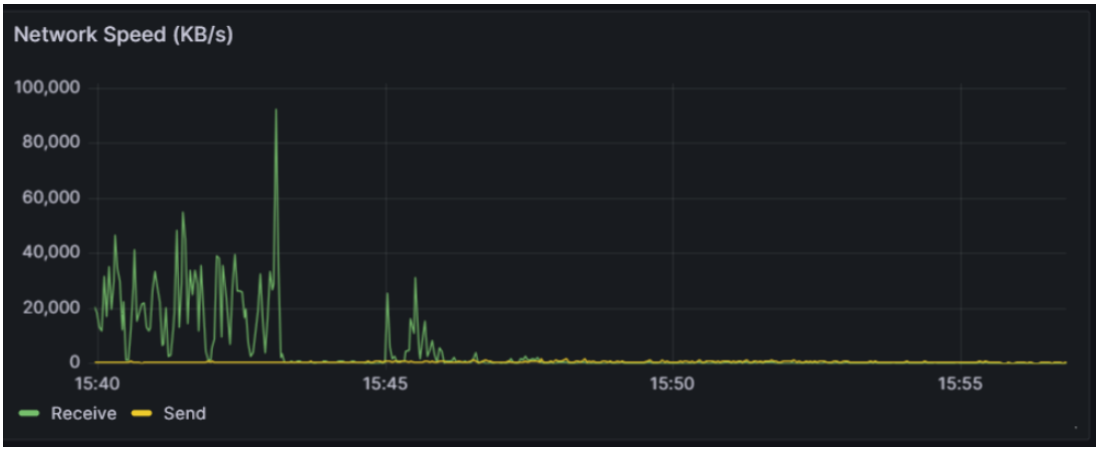


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1

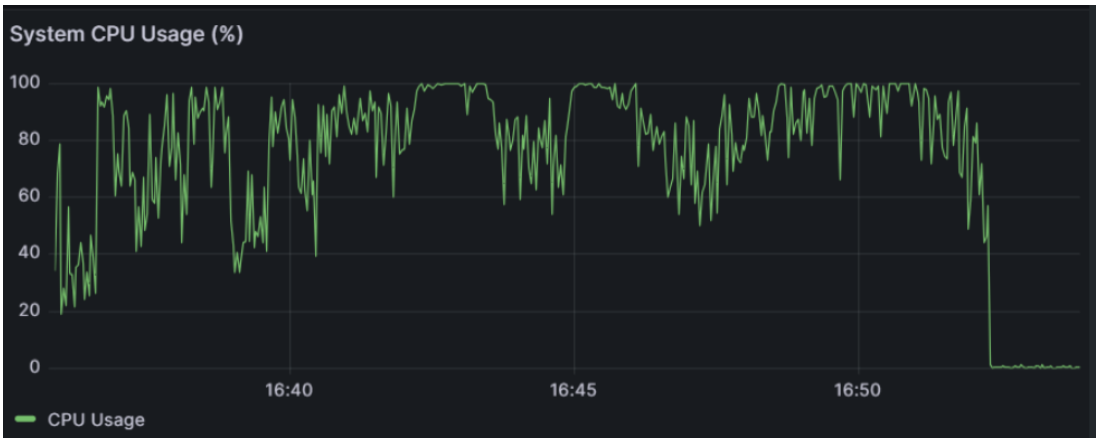


MD Core #2



CPU usage

MD Core #1

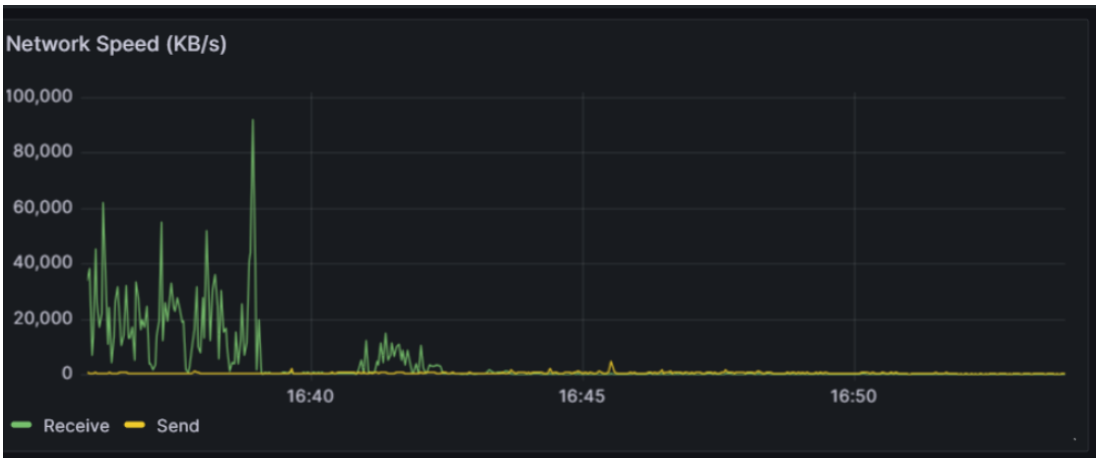


MD Core #2

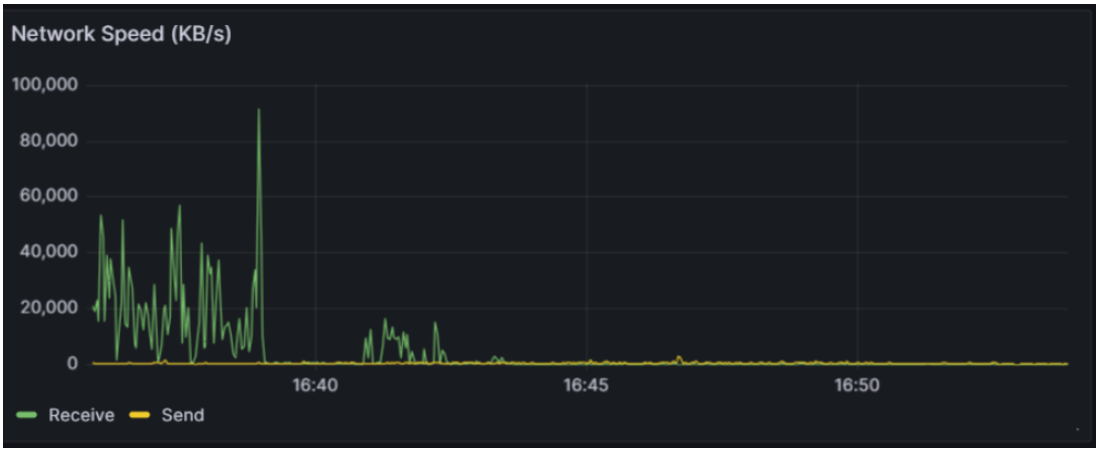


Network usage

MD Core #1



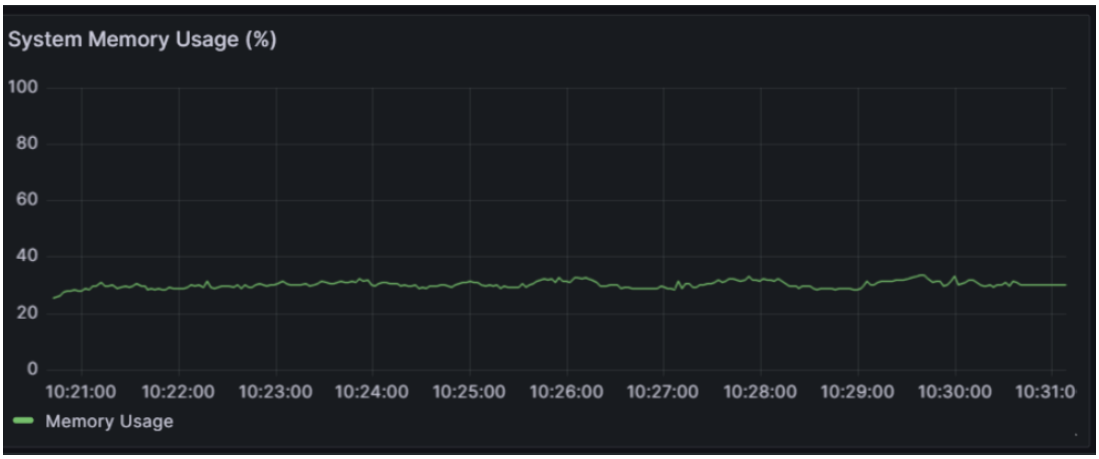
MD Core #2



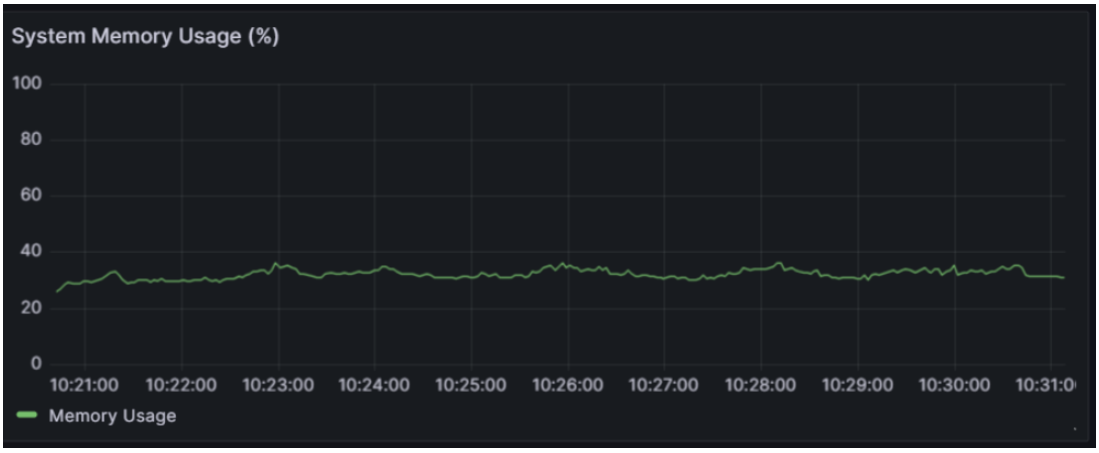
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

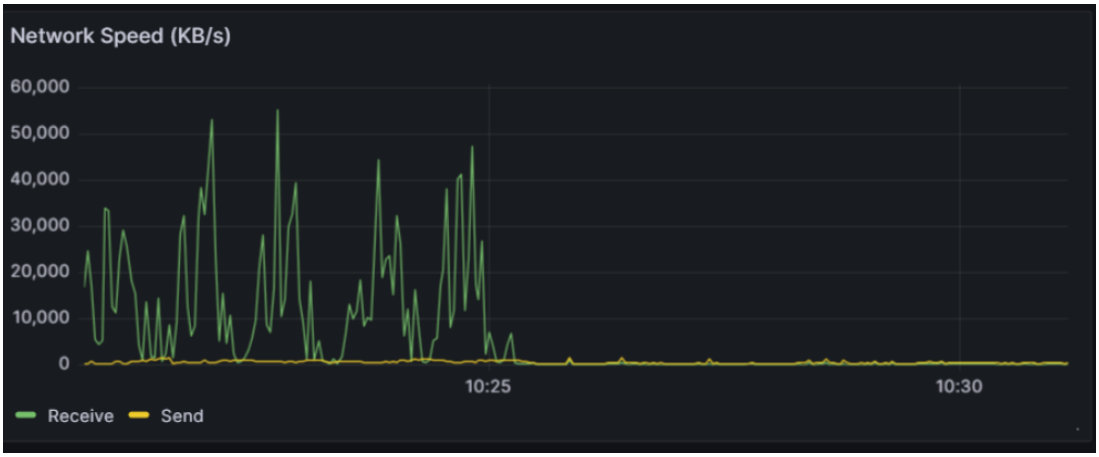


MD Core #2

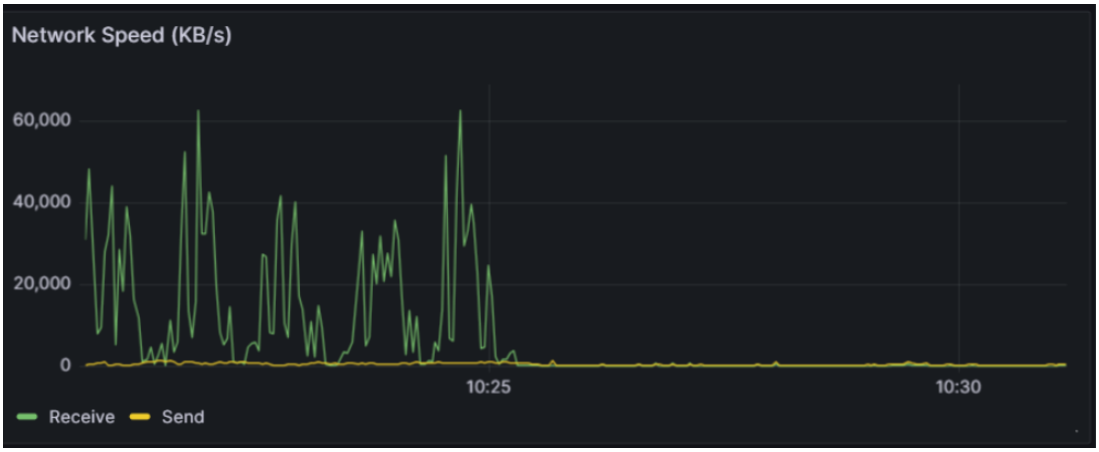


Network usage

MD Core #1



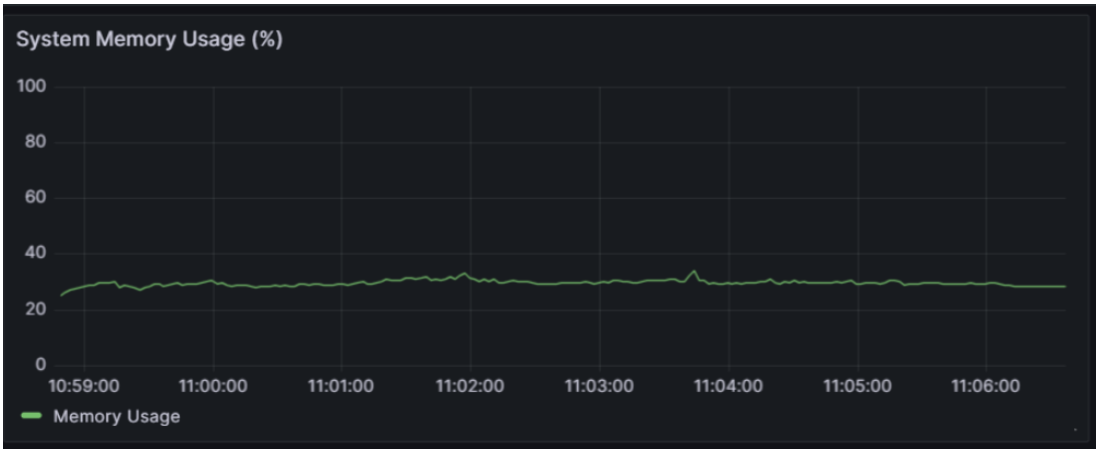
MD Core #2



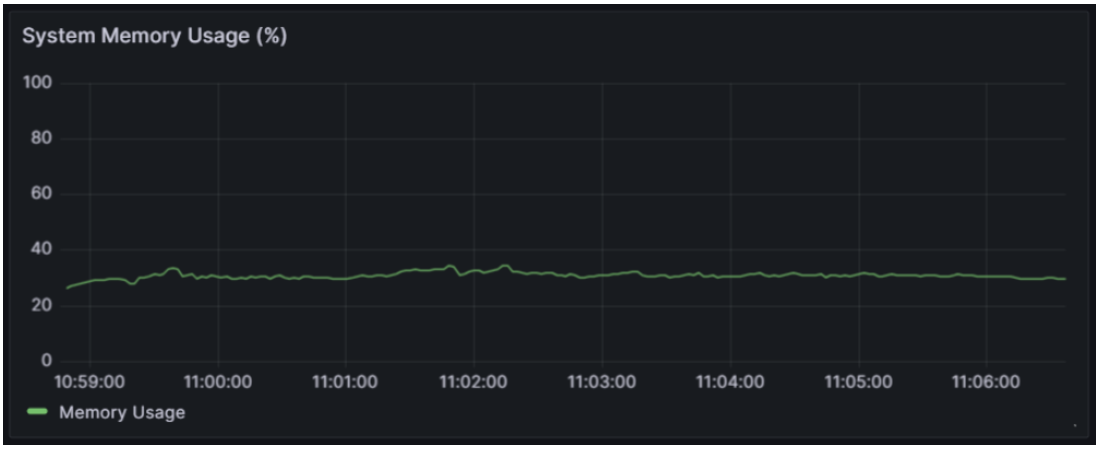
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

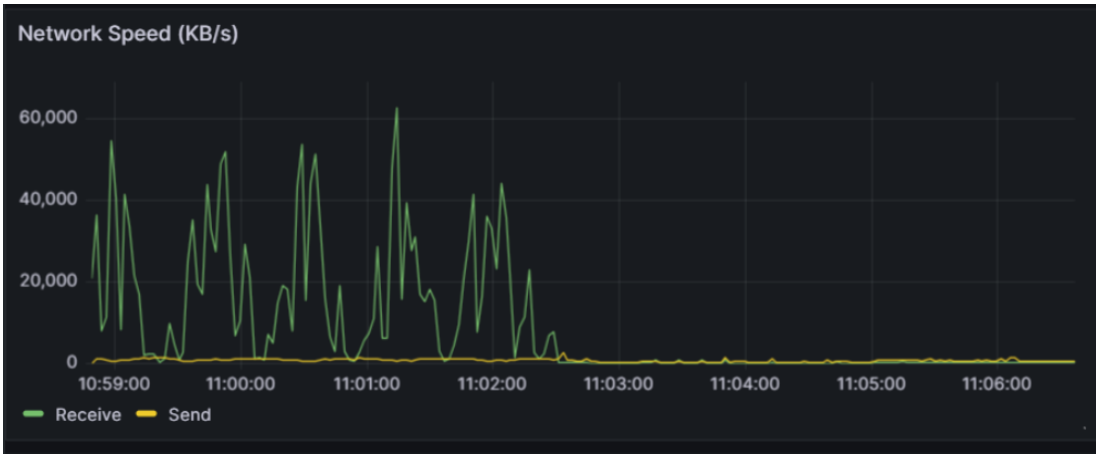


MD Core #2

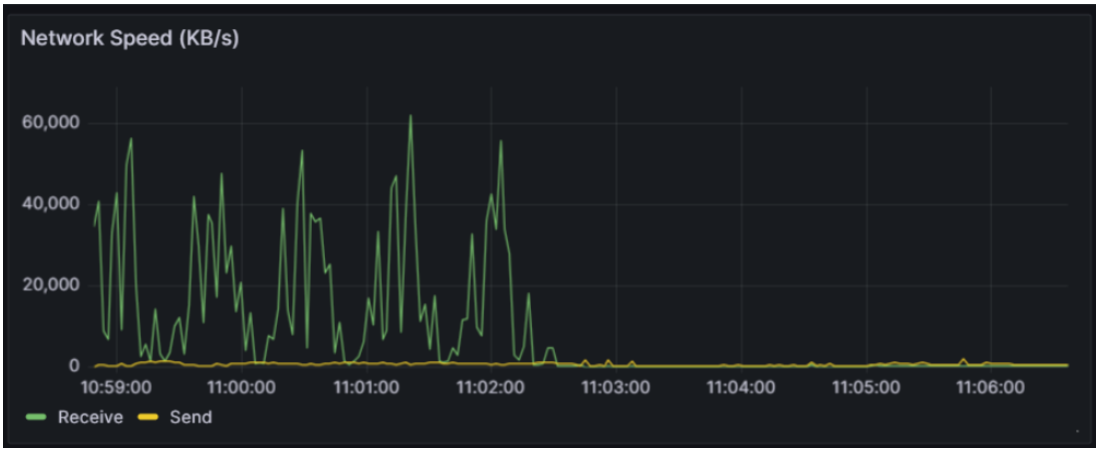


Network usage

MD Core #1



MD Core #2



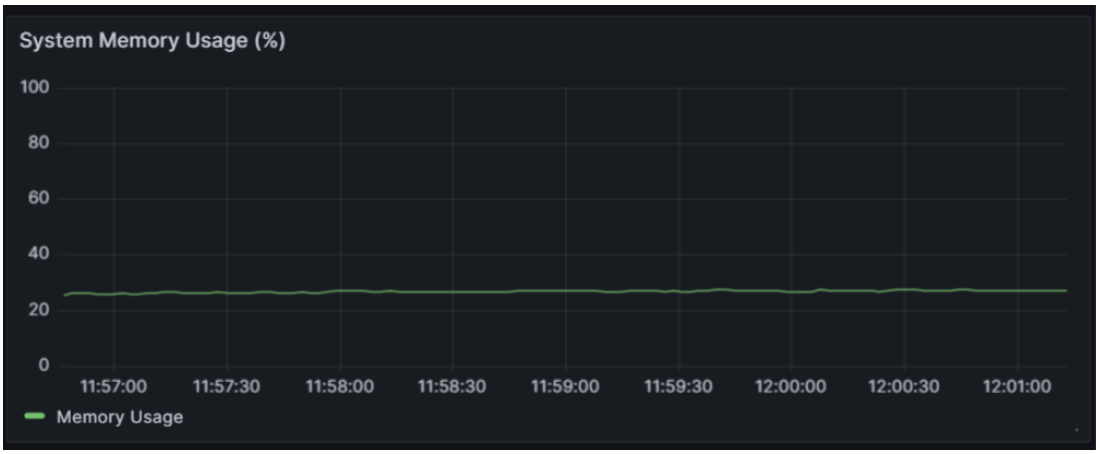
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

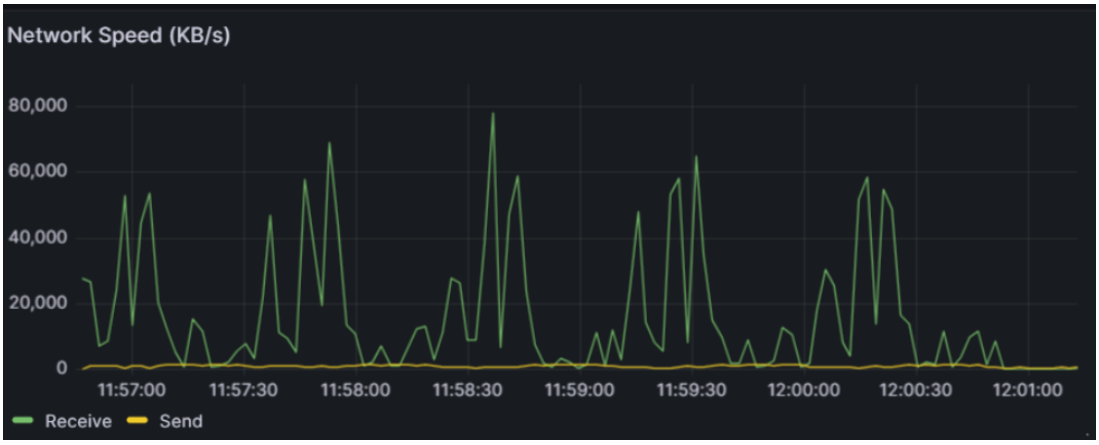


MD Core #2

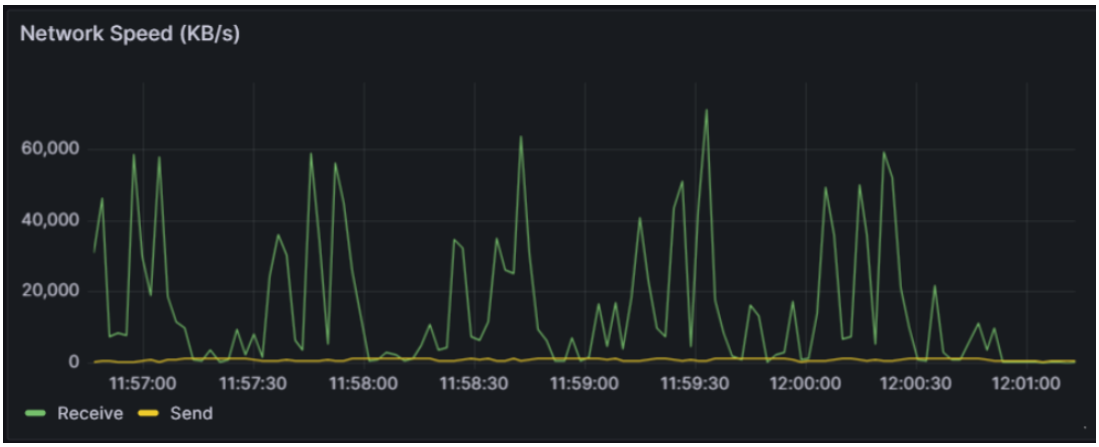


Network usage

MD Core #1



MD Core #2



Shared DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

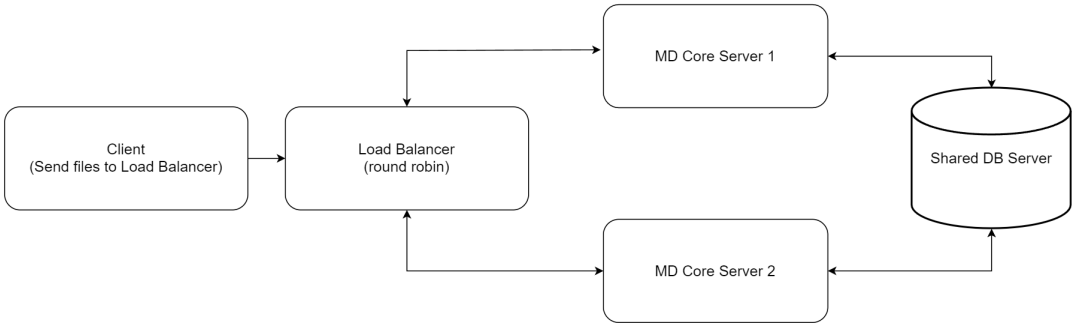
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV)	15.5	63.1	73.3	25.2	25.8	Receive: 4555.9 Send: 694.4	Receive: 4516.2 Send: 692.4	492,620.5	0.0
Deep CDR	9.70	50.7	51.2	31.9	32.5	Receive: 7062.9 Send: 623.2	Receive: 6653 Send: 654.1	786,189.6	0.0
Proactive DLP	7.2	35.2	35.0	27.9	27.5	Receive: 9047.4 Send: 802.4	Receive: 9859.8 Send: 787.5	1,055,110.8	0.0
Vulnerability	5.0	45.7	49.3	24.4	24.5	Receive: 13,585.7 Send: 966.7	Receive: 13,505.8 Send: 987.4	1,519,562.1	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV) + Deep CDR	19.9	79.1	77.6	34.0	34.2	Receive: 3785.9 Send: 573.8	Receive: 3663.1 Send: 577.7	382,716	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP	21.6	80.3	81.5	37.7	44.4	Receive: 3305.6 Send: 543.3	Receive: 3414.3 Send: 556.2	353,694.9	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	22.5	84.2	79.8	37.9	44.7	Receive: 3267.9 Send: 554.9	Receive: 3398.2 Send: 553.8	338,507.6	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

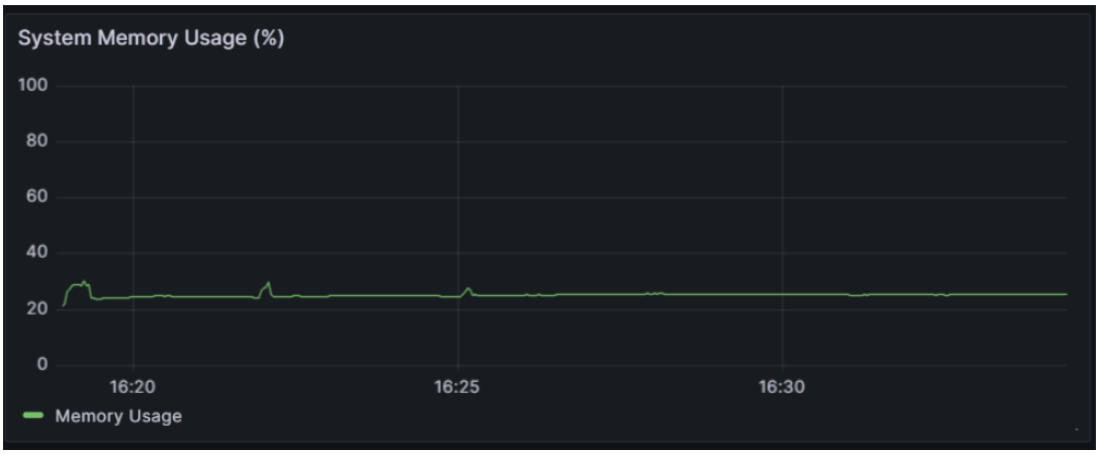
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

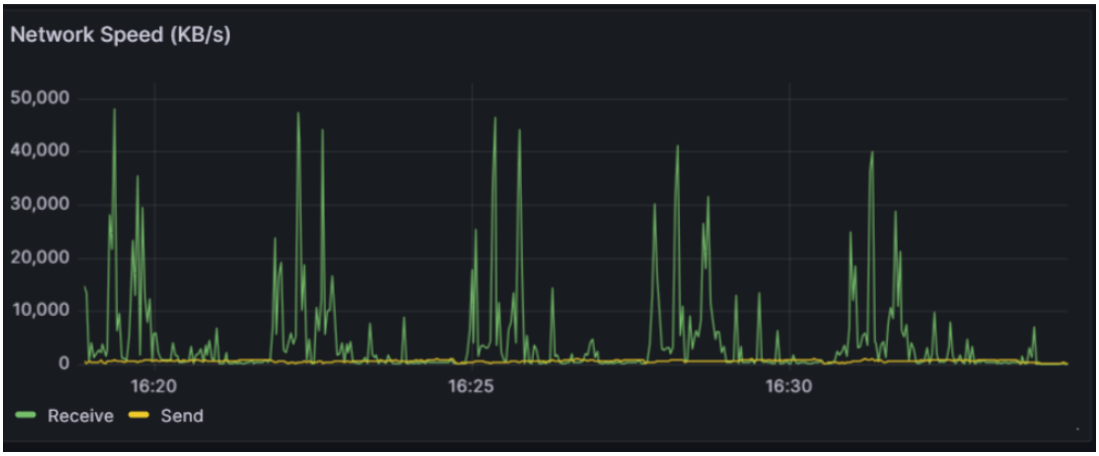


MD Core #2

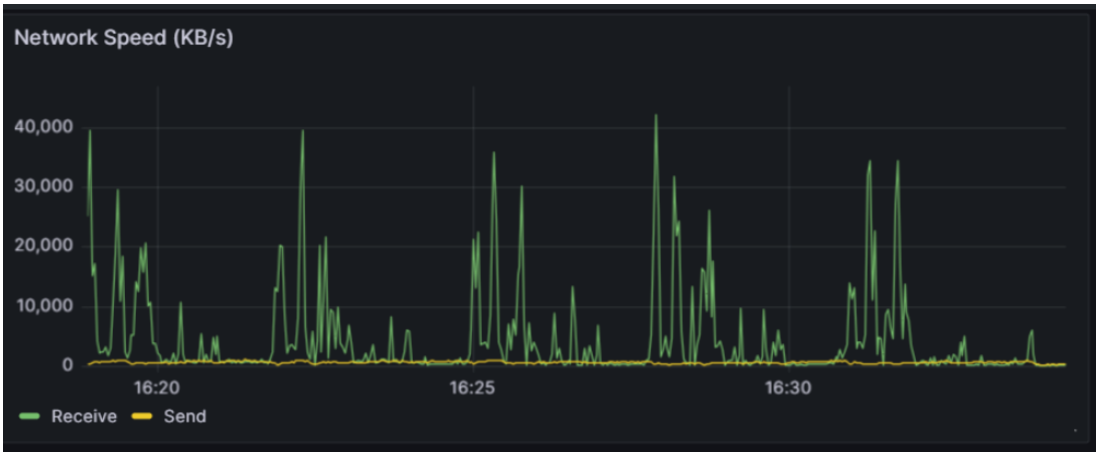


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

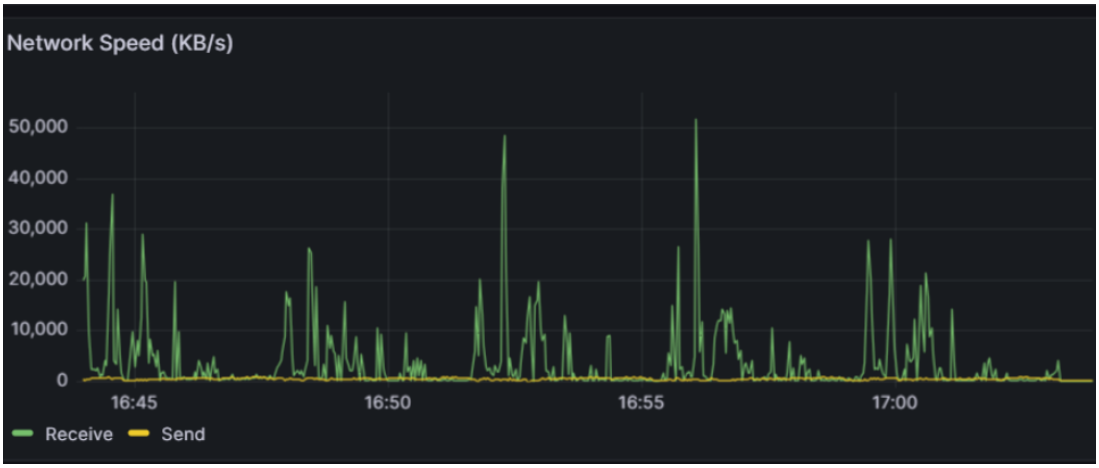


MD Core #2

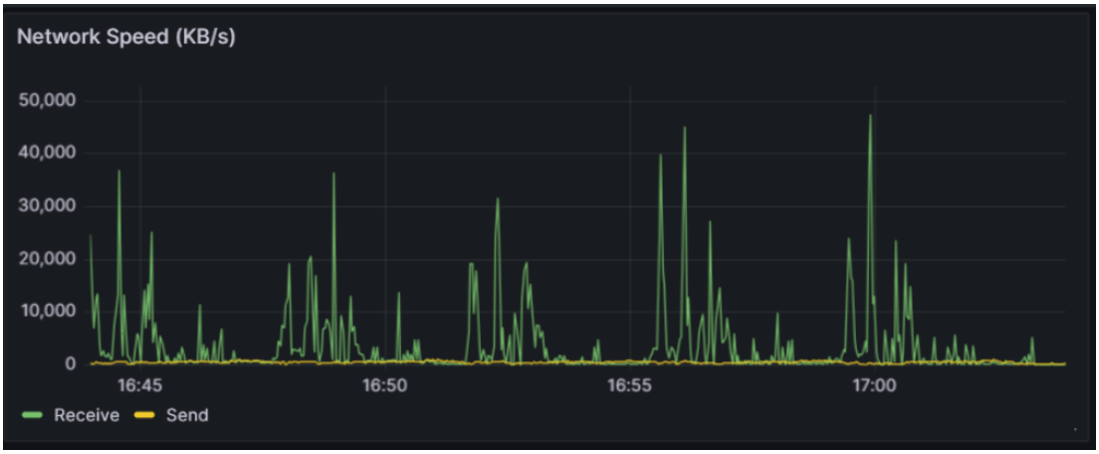


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

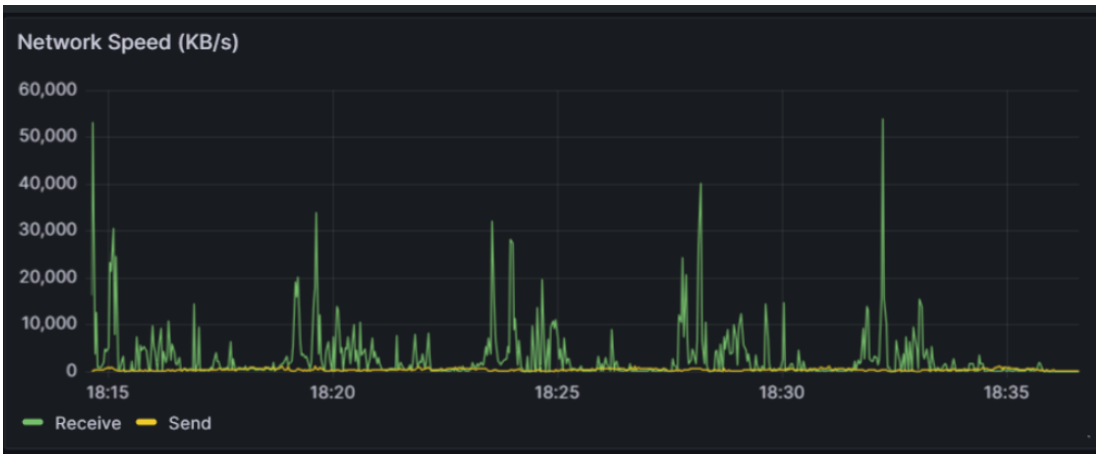


MD Core #2

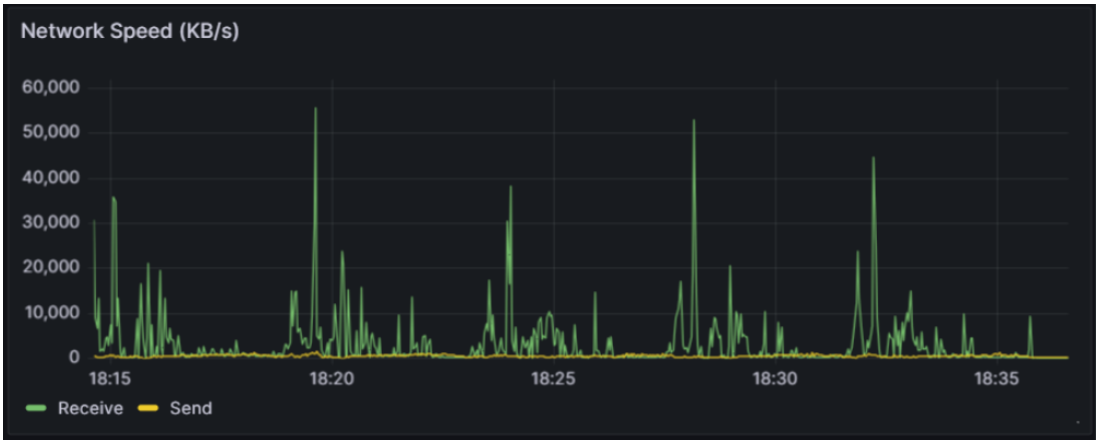


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

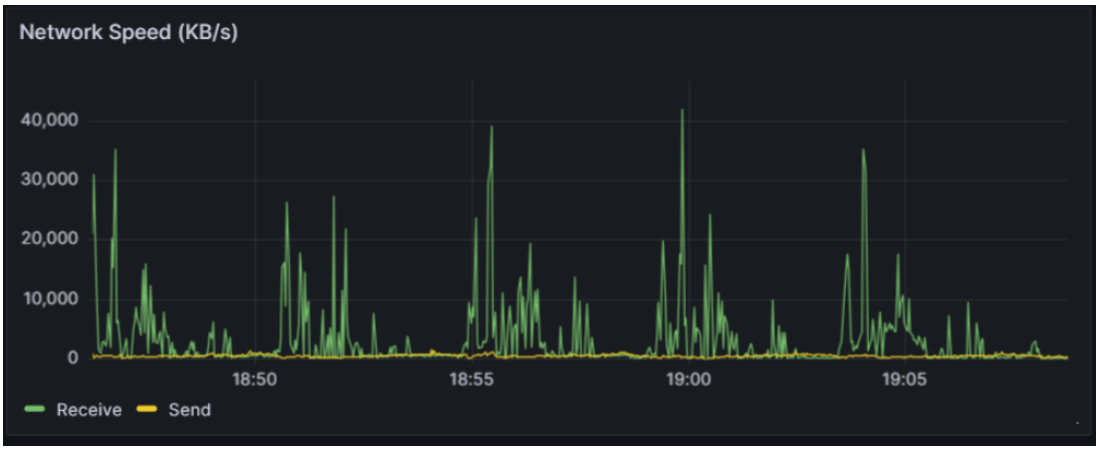


MD Core #2

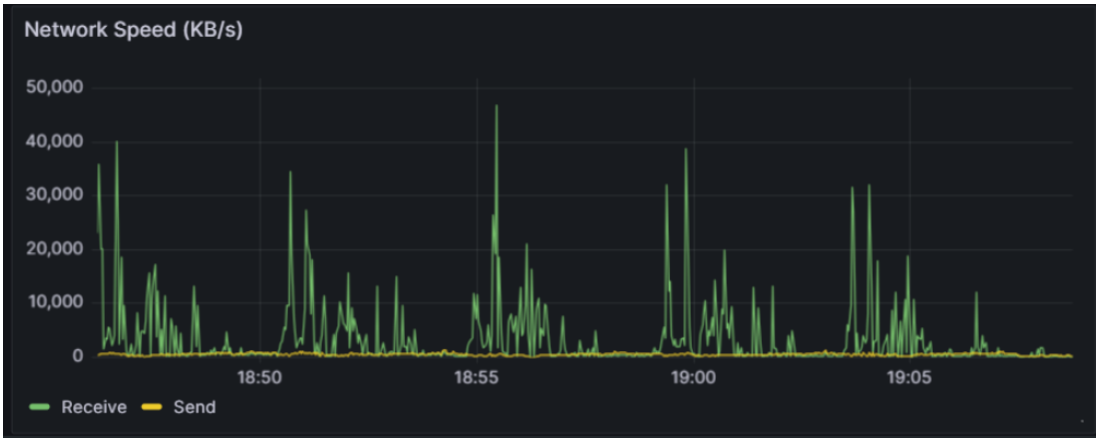


Network usage

MD Core #1



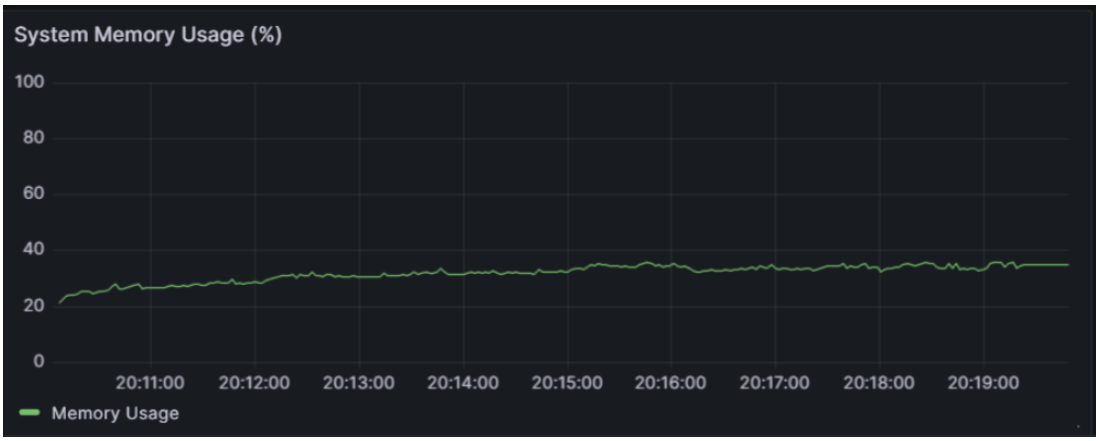
MD Core #2



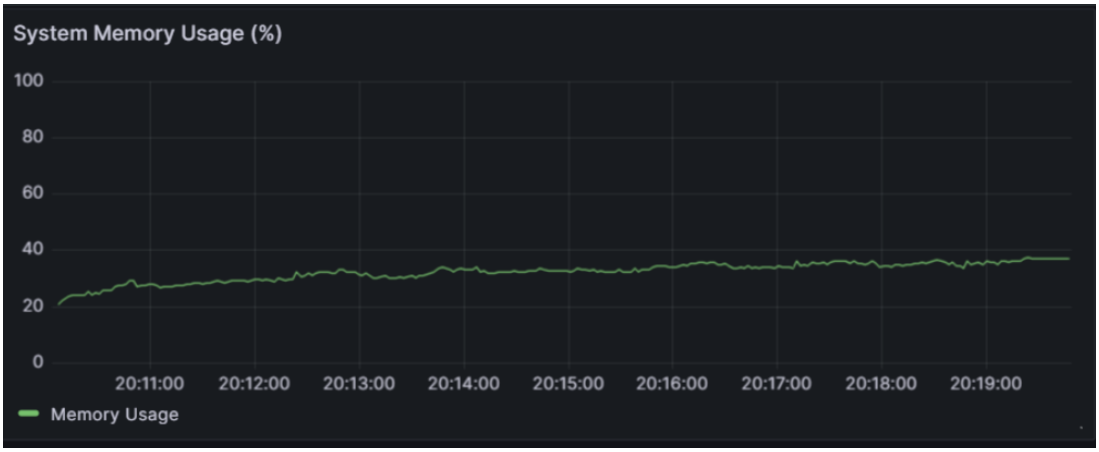
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1

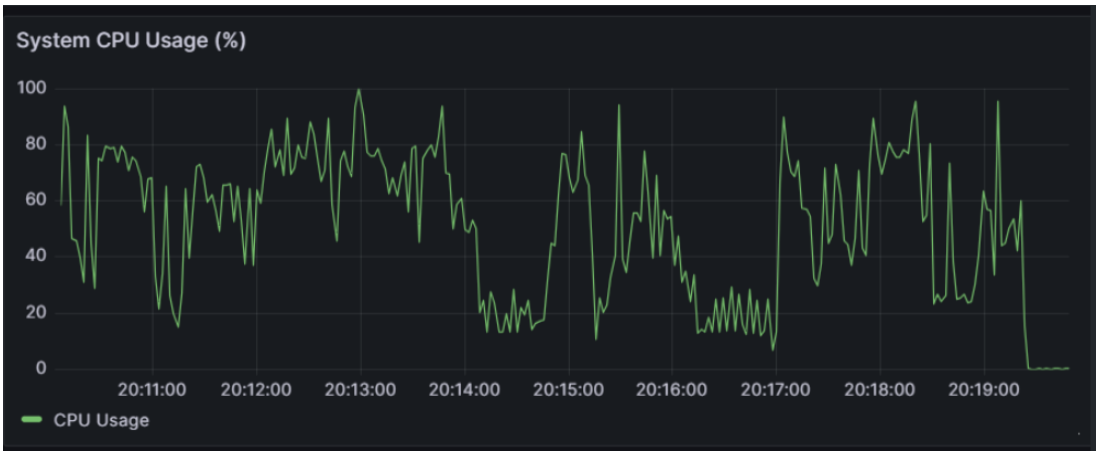


MD Core #2



CPU usage

MD Core #1

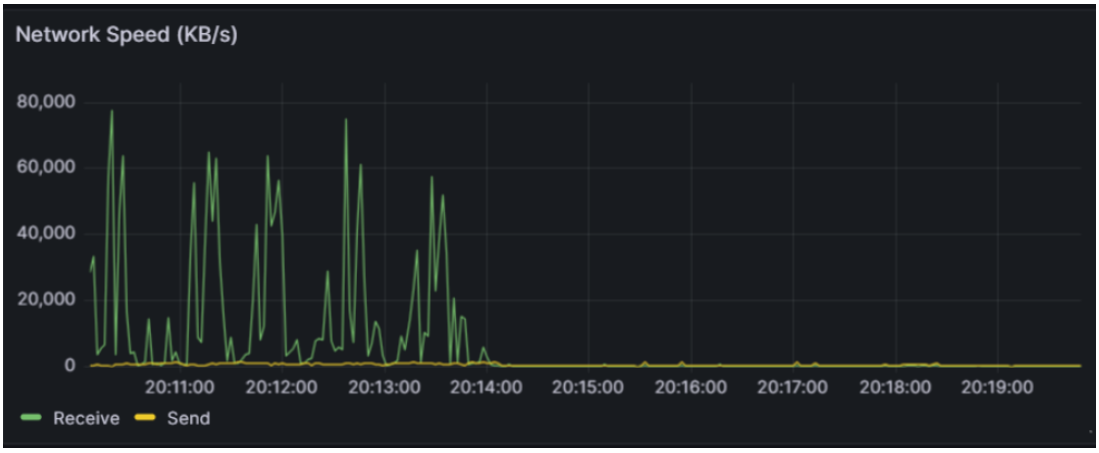


MD Core #2

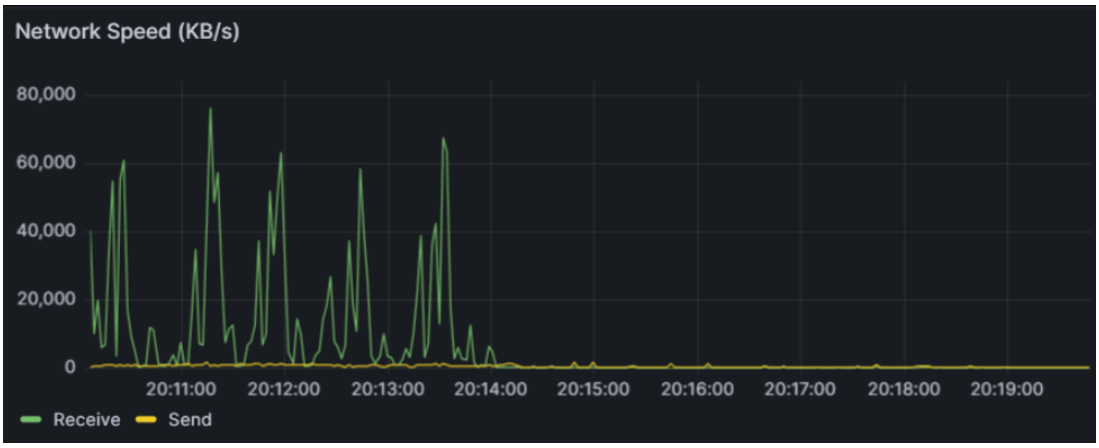


Network usage

MD Core #1



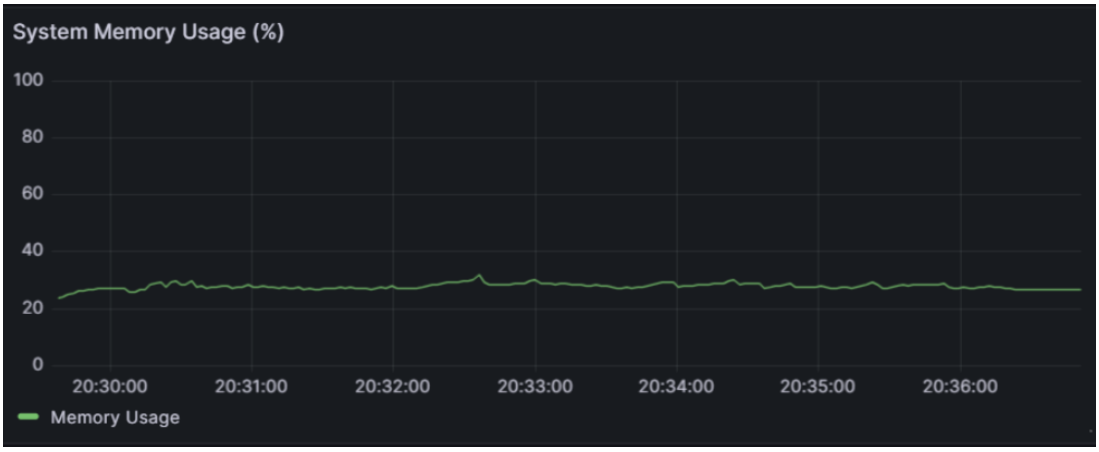
MD Core #2



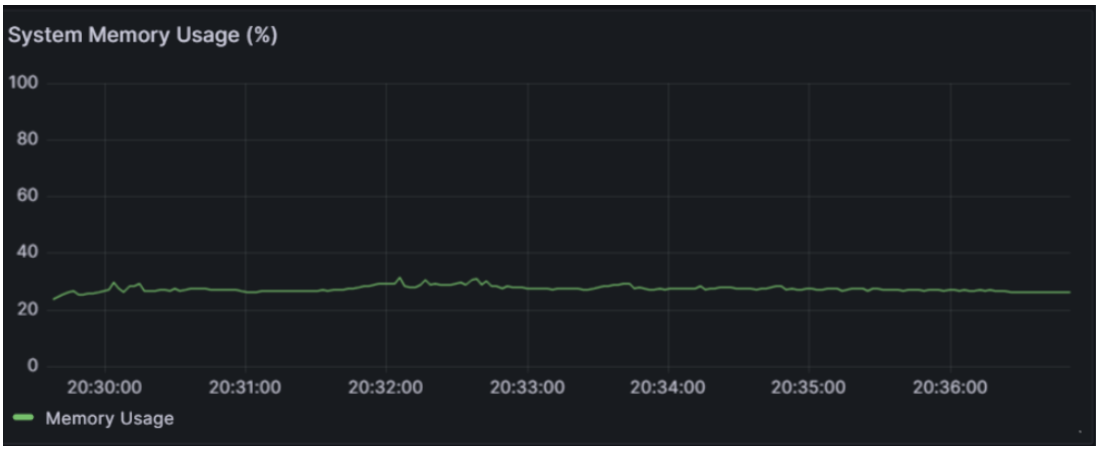
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1

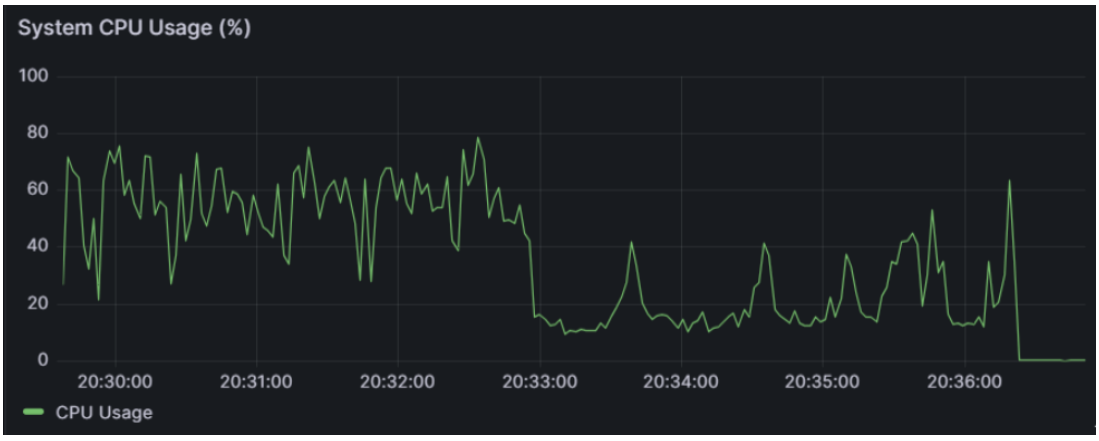


MD Core #2

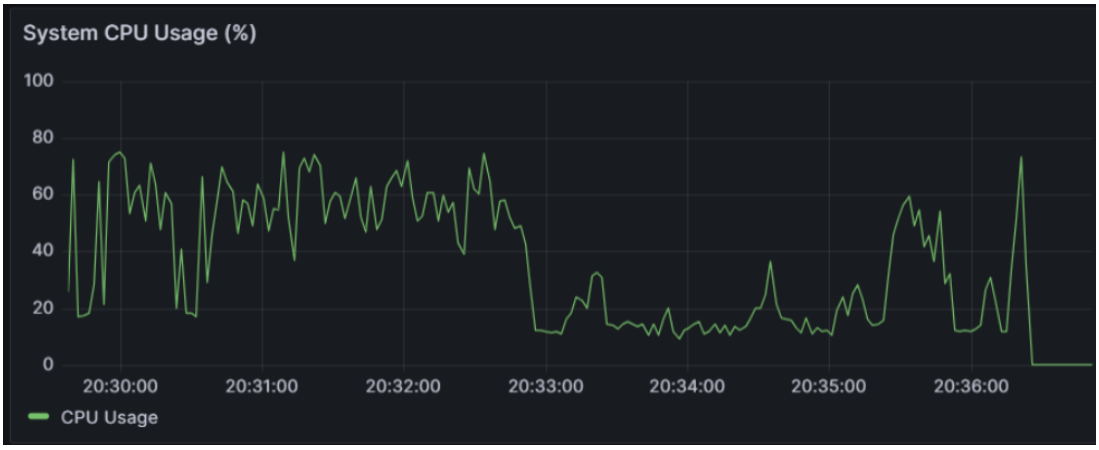


CPU usage

MD Core #1

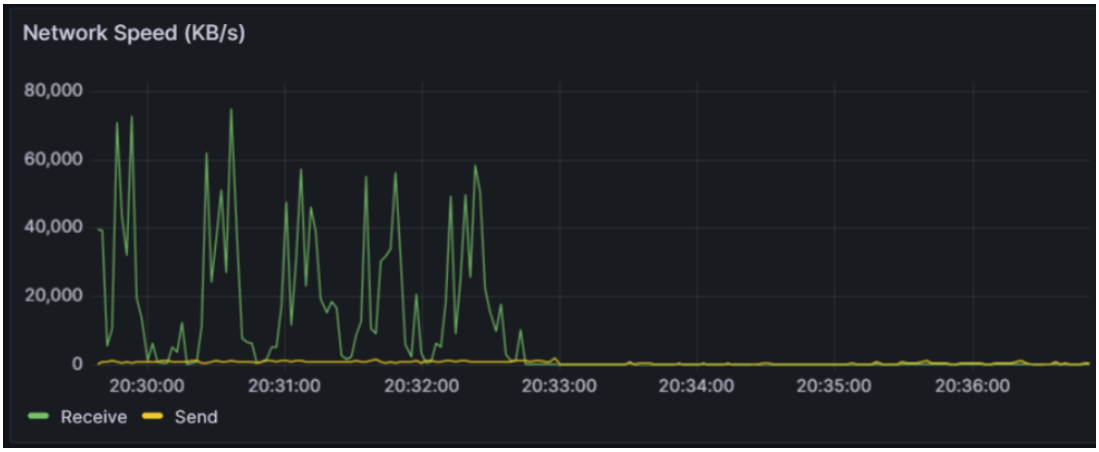


MD Core #2

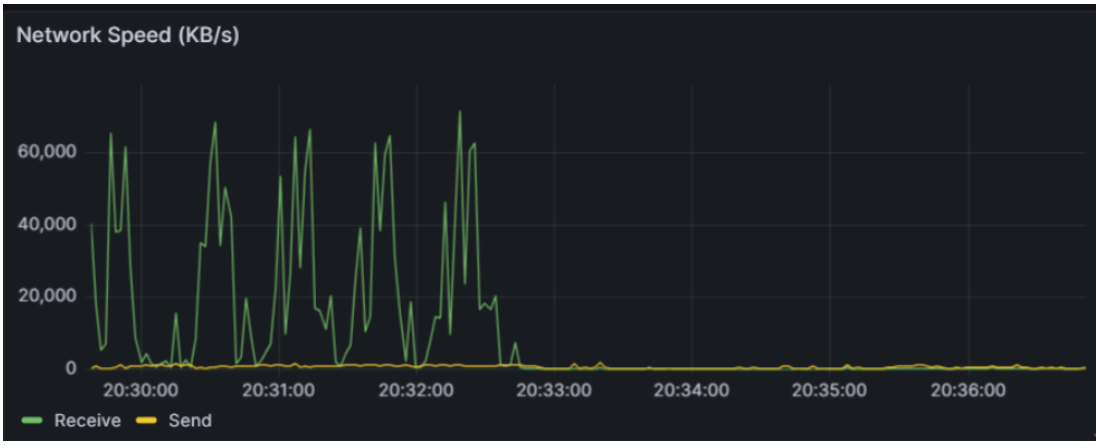


Network usage

MD Core #1



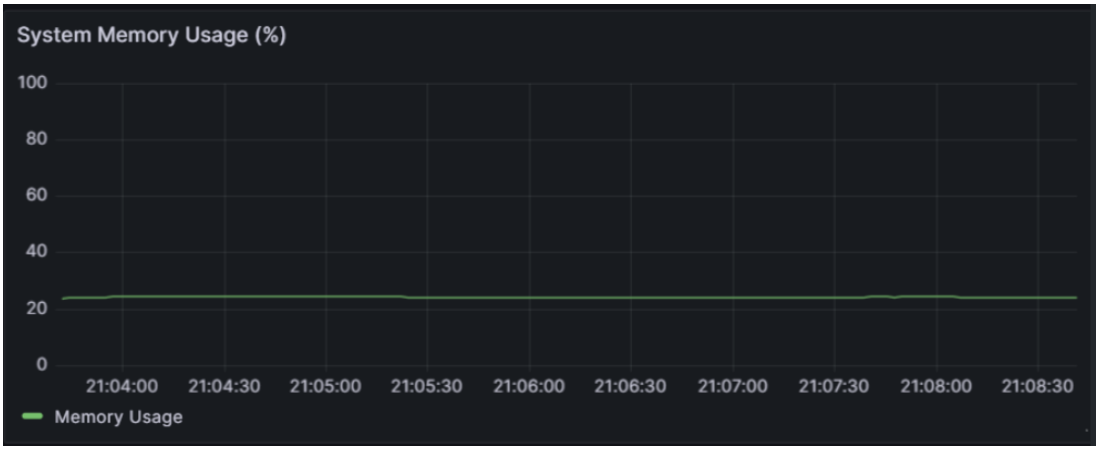
MD Core #2



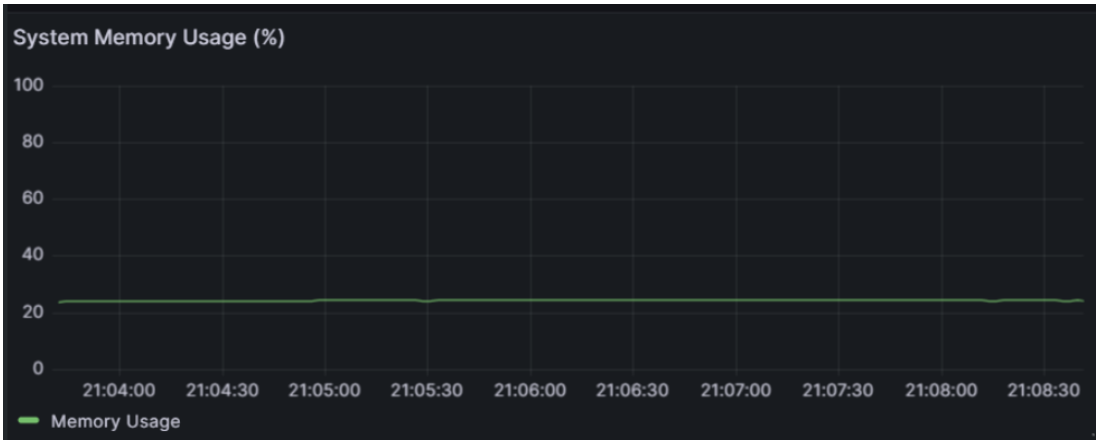
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

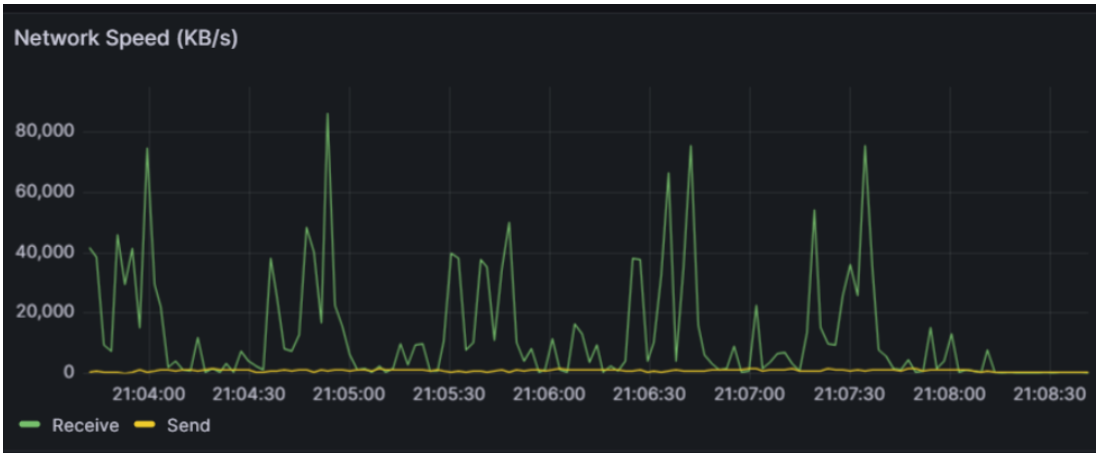


MD Core #2

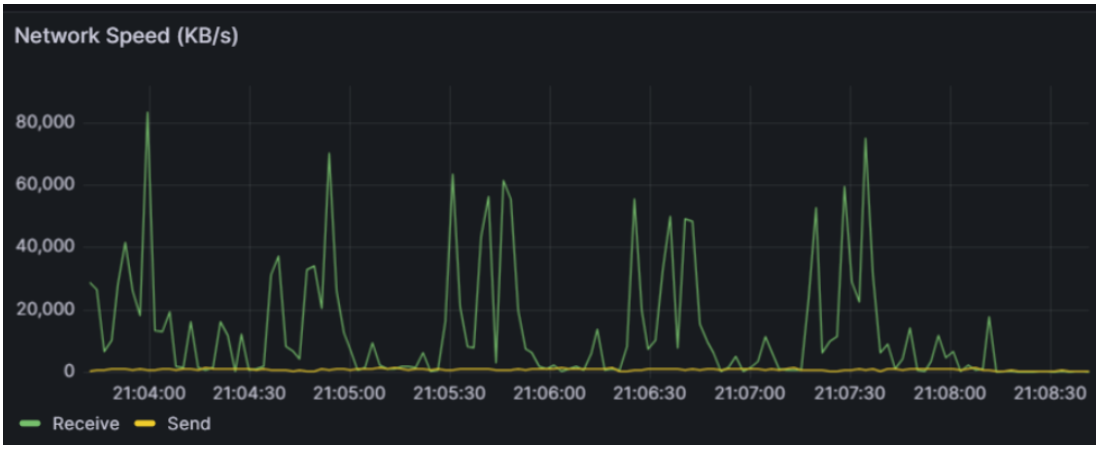


Network usage

MD Core #1



MD Core #2

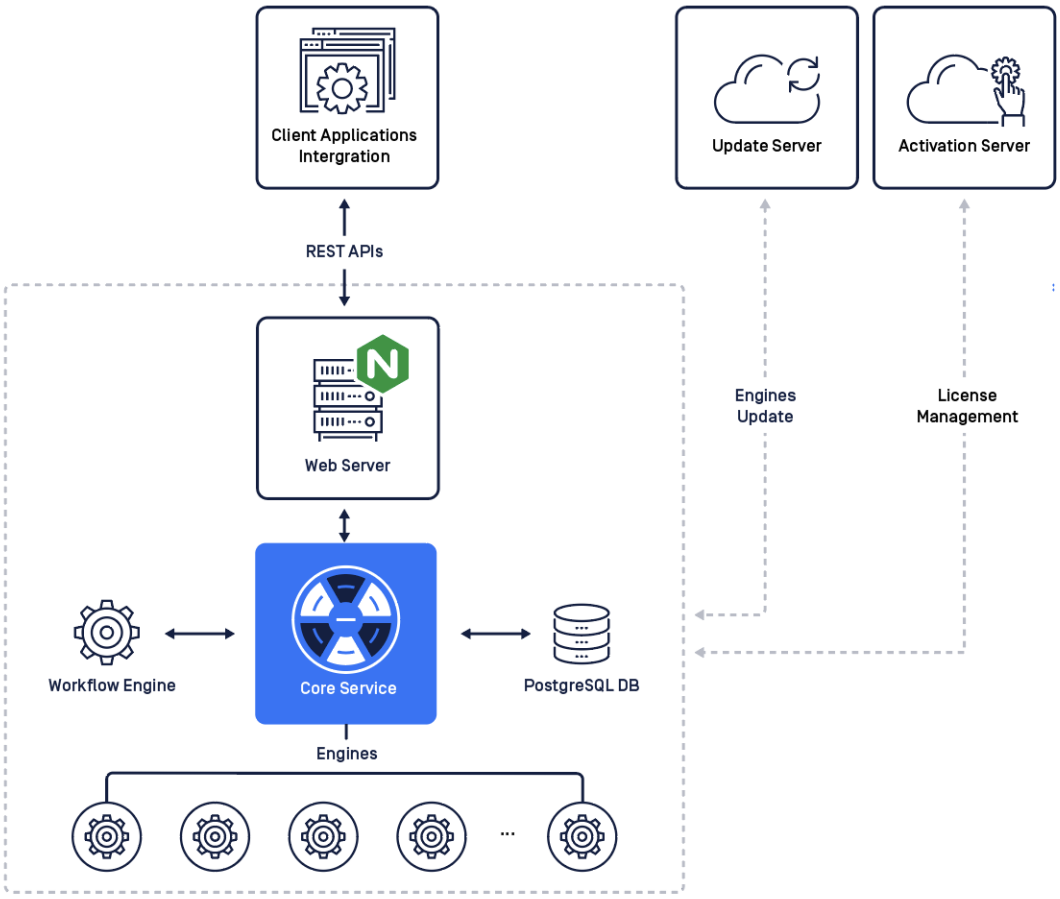

Network Speed (KB/s)
80,000
60,000
40,000
20,000
0
21:04:00
21:04:30
21:05:00
21:05:30
21:06:00
21:06:30
21:07:00
21:07:30
21:08:00
21:08:30
Receive
Send

MetaDefender Core Documentation

The users can consult this web page or, alternatively, they can download the manual in pdf format from the link below:

MetaDefender Core manual (SHA256:).

Recommended System Configuration



MetaDefender Core supports a wide range of deployment options, including air-gapped environments, providing the flexibility to choose the environment that best suits customers' requirements and infrastructure.

- Physical server.
- Virtualization platforms like VMware, Hyper-V and XenServer.
- Cloud Deployment.
- Container Deployment.

> **Note**
> Before installing MetaDefender Core, please refer to the **recommended minimum system configuration** listed below. The requirements are based on a standardized customer deployment and may require more or less resources to achieve the level of performance desired in your environment. Please note that the server specifications are built to allow a high volume daily processing. Please refer to Performance and Load Estimation for more details.

The recommendations below are for MetaDefender Core, API usage only. For any other use cases, please consult the user guide of the licensed products for accurate recommendations.

For certain use cases these might be adjusted and customized on specific needs and SLAs. We highly recommend to engage our ProServ team to assist in fine tuning MetaDefender and get the maximum performance out of your systems.

Microsoft Windows Deployments

[Windows] Supported Operating Systems

Only support 64-bit platforms.

- Windows 10, Windows 11 23H2, 11 IoT (22H2) only for End-User systems (e.g. Kiosk deployments)
  - Based on Microsoft License Terms, you're not allowed to use Windows Desktop as a server, therefore, MetaDefender Core can't be used on these systems for server use cases.
- Windows Server 2016, 2019, 2022

> **Disclaimer**
> End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

> **Plan for legacy OS support**
> OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:
>
> - Ubuntu 20.04 after **April 2025**.
> - Amazon Linux 2 after **June 2025**.
> - Windows 10 after **October 2025**.
>
> If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

[Windows] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 8	8	8 GB	16 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 12	16	16 GB	24 GB + 50 GB + 100 GB
MetaDefender Core 16	16	16 GB	32 GB + 50 GB + 100 GB
MetaDefender Core 20	32	16 GB	40 GB + 50 GB + 100 GB
MetaDefender Core MAX	32	32 GB	120 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core is installed.
- For a better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.
- SBOM processing requires more disk space, please allow at least 15GB free disk for the engine to work properly.

[Windows] Third Party Dependencies

- .NET framework 4.5 or above
- Microsoft Visual C++ Redistributable for Visual Studio 2015-2022 (version 14.38 or higher); x64 version.
- PostgreSQL: Only applicable when using a pre-installed PostgreSQL server running remotely.
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Some engines also have dependencies as described below:

Engine Name	Dependency
Deep CDR	Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 5.8.0 or above)
ESET	MetaDefender Core v4 temporary directory should have more than 200MB free disk space
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
Proactive DLP	• Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 2.0 or above) • .NET 6 dependencies • CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
Scrutiny	Microsoft Visual C++ 2015 - 2019 Redistributable x86 and x64
Systweak	.NET runtime 6.0, 7.0 or 8.0
Bkav Pro	Microsoft Visual C++ 2015 - 2022 Redistributable x64

[Windows] Installation Details

MetaDefender Core on Windows uses "C:\Program Files\OPSWAT" folder for storing resources or the installation directory.

MetaDefender will use its resources folder to store temp files as part of the analysis. **It's recommended to exclude this folder from real-time protection monitoring**. Knowledge base:

- How to set up exclusions on anti-malware software to prevent disruption
- Mitigating scan failure triggered by real-time protection service
- Can local AVs interrupt ongoing scans?

Linux Deployments

[Linux] Supported Operating Systems

Only support 64-bit platforms.

- Red Hat Enterprise Linux 8, 9
- Rocky Linux 9
- Debian 11, 12
- Ubuntu 20.04, 22.04
- Amazon Linux 2 is supported via Amazon Machine Image (AMI)

Plan for legacy OS support

OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:

- Ubuntu 20.04 after **April 2025**.
- Amazon Linux 2 after **June 2025**.
- Windows 10 after **October 2025**.

If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

For Hardended Linux OS

Some AV engines could be failed to initialize and run on MetaDefender Core (e.g. ESET, Kaspersky) due to execution permission required by them on `/var/tmp/ometascan` folder.

Please follow steps at Why have the ESET and Kaspersky scan engines failed to initialize on the hardened Linux OS? for remediation details.

Disclaimer

End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

[Linux] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 5	4	4 GB	10 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 10	8	8 GB	20 GB + 50 GB + 100 GB
MetaDefender Core MAX	16	16 GB	40 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core resides on.
- For better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.

[Linux] Third Party Dependencies

Dependencies list:

- openssl
- grep
- lib32stdc++6 (>= 4.5)
- libc6-i386 (>= 2.10)
- procps
- zlib1g
- libcurl3 (>= 7.19.7)
- libcurl4
- postgresql:
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Not all the above dependencies will need to be installed, it is dependent on different Unix distro and version

Some engines also have dependencies as described below:

Engine name	Dependency
Archive engine	• To extract password protected MS Office files, you need to install .NET 6 dependencies, libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • Rocky Linux 9.x/Red Hat 9.x: libnsl
Deep CDR	• Rocky Linux 9.x: libgomp, ncurses-compat-libs, libnsl • CentOS: libgomp, ncurses-compat-libs • Red Hat 8.x: libgomp, ncurses-compat-libs, libnsl • Red Hat 9.x: libgomp, ncurses-compat-libs, libnsl, compat-openssl11 • Ubuntu/Debian: libgomp1, libncurses5 • libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • .NET 6 dependencies
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
FileType engine	Rocky Linux 9.x/Red Hat 9.x: libnsl
Proactive DLP	libgdiplus, mscorefonts .NET 6 dependencies CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
RocketCyber	libgomp

Note: Some dependencies may need to be installed from external repositories, not from the OS default repositories.

[Linux] Installation Details

MetaDefender Core default installation path is using /var folder for storing resources:

- `/var/lib/ometascan` : installation folder with all its resources (database, DLP processed / CDR sanitized / quarantined file storage, engine package and definition updates).
- `/var/log/ometascan` : application logs.
- `/var/run/ometascan` : runtime folder for Core related processes.
- `/var/lib/ometascan` : all successful deployed engines data files.
- `/usr/lib/ometascan` : all shared libraries required by MetaDefender Core, bundled PostgreSQL server related binaries and libraries.
- `/var/tmp/ometascan/resources` : all temporary processing files for MetaDefender Core to process.

Storage Usage Information

Based on the configuration, MetaDefender Core will need additional disk space to store analysis/result data, quarantined files and temporary files:

	Data Storage
Analysis Reports (Processing History)	Approximately 4.5GB is required for every 1M analysis reports stored. • 100 GB is recommended as a baseline. Database storage requirements are affected by Data Retention settings, engine settings and scanning load.
Quarantined Files	Depends on the customers' dataset and configuration
Sanitized Files	Depends on the customers' dataset and configuration
Temporary processing files	Depends on the customers' dataset and configuration • A 1GB non-archive file requires 1GB free disk space at least. • A 1GB archive file requires (1 * 2)GB free disk space at least. • 50GB is a baseline and should be adjusted based on observation of the solution within the environment, with consideration to peak times and overall file makeup (e.g. complexity, size, archives).
Note	Options are available to control how large and/or complex files are handled to limit the amount of system resources required. See Workflow Configuration Template for more information.

Custom Engines

The recommendations above are specific for MetaDefender pre-packaged bundles.

However for additional Custom Engines, please review the Knowledge base to review additional requirements (if any) for the selected engine.

Browser Requirements for the MetaDefender Core Management Console

One of the following browsers is suggested to view the MetaDefender Core Management Console:

- Microsoft Edge
- Chrome
- Firefox
- Safari

Chrome, Firefox, Safari and Microsoft Edge browsers are tested with the latest available version at the time of release.

Network Bandwidth

In online environment where MetaDefender Core is configured to download engine update packages directly from OPSWAT update server source, then using gigabit network speed (125 MB/sec) is highly recommended to avoid potential network related issue while downloading files, or at least 100 Mbps (12.5 MB/sec) network speed.

Allowlisting URLs

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts':

- https://activation.dl.opswat.com - for license online activation
- https://update.dl.opswat.com - for engines and databases download

Note: *IP address-based allowlisting on your firewall might fail after some time since OPSWAT uses CDN (AWS Cloudfront) to faster delivery updates over the world, and IP address of edge servers might change over time.*

Depending on your location, connections with OPSWAT cloud-based servers be forwarded to specific AWS Cloudfront's edge locations, each is assigned with its own unique IP address ranges. You might also need to allow correct AWS Cloudfront's corresponding IP address ranges by referring to https://ip-ranges.amazonaws.com/ip-ranges.json

Warning

Even the OPSWAT update servers host updates for all of the available engines we support, sometimes custom engines might try to connect to their own cloud for updates, but this can be disabled in firewall and they will be updated just from OPSWAT cloud servers instead.

Differences Between MetaDefender Core v4 and v5

We are very excited to inform you about the release of version 5.0.0 of our MetaDefender Core!

The new version is designed to enhance the product performance, user experience, as well as to provide more flexibility, security, and scalability for our customers.

At the same time, we are also committed to keeping backward compatibility for whom running MetaDefender Core v4, and avoid the need for updating your client integration to move forward with MetaDefender Core v5.

v4 REST API Integration Supported
All public REST API endpoints available on MetaDefender Core v4 will be continously supported on v5.

v4 Product Major Features Supported
All product major features available on MetaDefender Core 4.21.2 (the latest one on v4) will be continously supported on v5.

Seamless Product Upgrade
Upgrading from MetaDefender Core v4 to v5 will be supported and covered with data migration without additional manual steps from users.

Licensing information will be also migrated.

Below are some highlights of the first v5 release, MetaDefender Core 5.0.0:

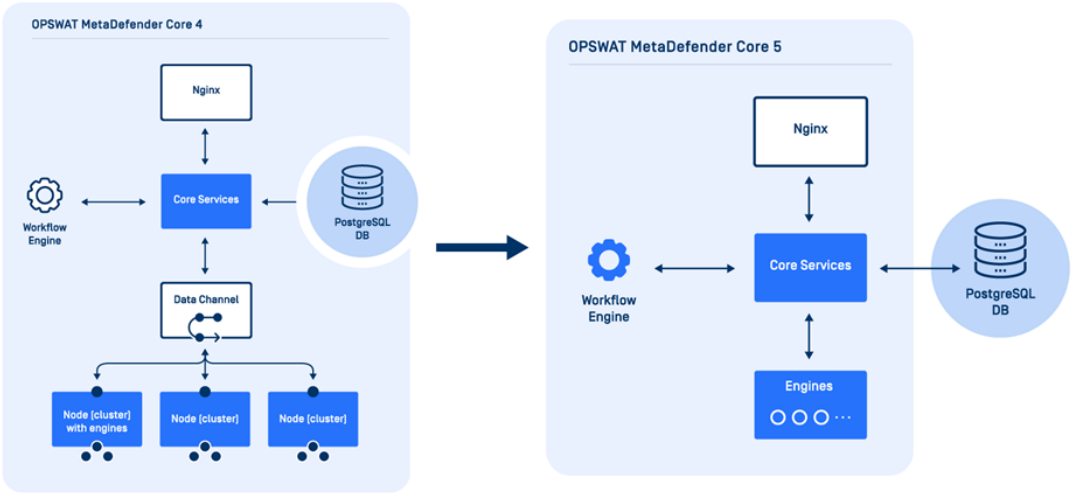
New product architecture

This version introduces a simpler product architecture that speeds up integration and automation, file processing flow, and immensely enhance the product performance and efficiency.

Earlier versions utilized MetaDefender Node concept, designed to support scaling of the scanning infrastructure by distributing scan requests among several different processing Nodes. Product licensing was based on number of Node instances. The benefit of having such a distributed infrastructure is that based on Node loads, Metadefender Core server can always choose the most appropriate Node to assign a new scan task to. However, having nodes in the v4 architecture caused more complexity and possible latency in the processing flow and entailed maintenance cost.

In this release, we retired multi-Node deployment and restructured MetaDefender Core's backend. This alteration significantly improves file processing speed and product efficiency, as well as eliminates extra cost for maintaining node services. Users now can simply install additional MetaDefender Core instances for scalability requirement,

as well as configure their own load balancer to coordinate tasks among multiple MetaDefender Core instances. Accordingly, product licensing is now based on number of the product instances.



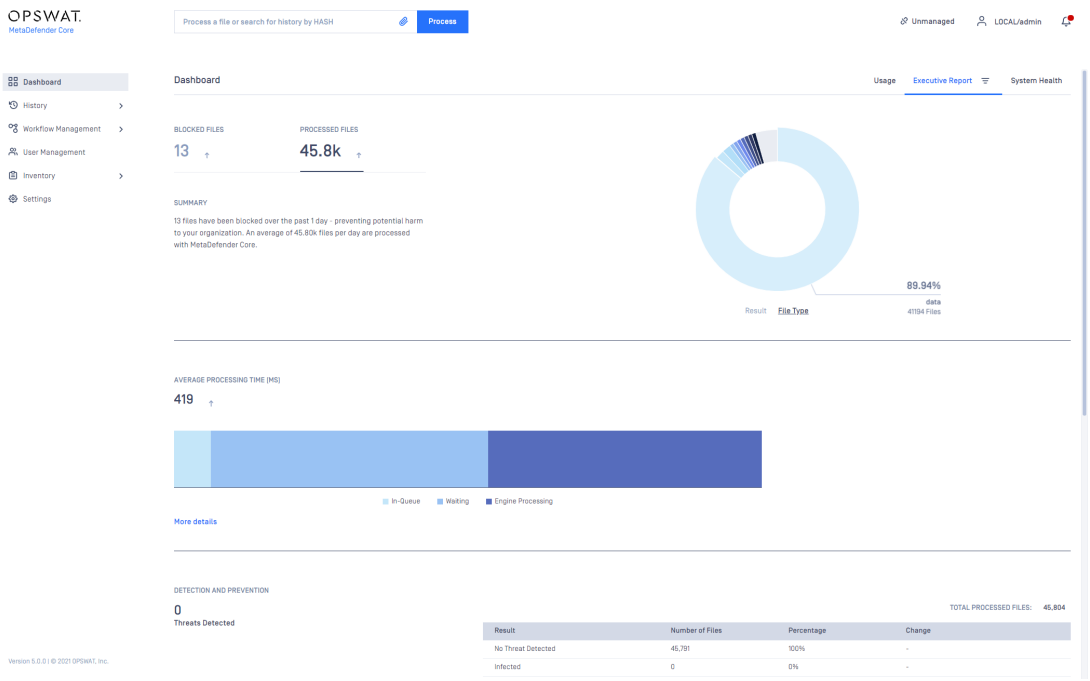
Please be informed that

- All existing administrative APIs related to "Node" on v4 will be kept as it is on v5 to ensure backward compatibility with existing client integrations.
- Applicable settings defined in Windows registry (Windows) or ometascan-node.conf config file (Linux) for Node will be migrated to Core. Details: MetaDefender Configuration

Brand-new product user interface

The second notable update in this version is a brand-new UI for the management console, which was designed with the focus on the quality and thoughtfulness of the user experience. The new administrative console UI provides an advanced set of Core features and observable processing status.

Comprehensive executive dashboard reporting offers dynamic, real-time and actionable threat insights. It also provides visibility of your security posture allowing you to paint a picture of security that aligns with the overall business. The report includes easy-to-read charts, graphs, and statistic explanation that help your security professionals reduce clutter, gather information quicker, spot threat activity early and act swiftly.



We're continuing to improve our product UI to provide users an excellent experience with the application and overall usability. Please expect more powerful UI features coming in next releases.

New compression archive engine for archive sanitization

In previous version, MetaDefender Core's archive engine implemented four functions including decryption, extraction, encryption and compression that with a high volume of files it takes time to process them all. For the purpose of enhancing archive handling performance, this release comes with a separated compression archive engine, which dedicatedly processes archive sanitization (compression and encryption). It is also created to facilitate maintenance and further feature development.

Archive sanitization feature now requires Deep CDR license

As to respect sanitization practice and ensure unknown threats prevention as designed, now starting on version 5, it's mandatory to have Deep CDR licensed for archive sanitization feature enablement on MetaDefender Core workflow configurations.

Archive handing related engines changed

The legacy "Archive" engine on MetaDefender Core v4 will be changed its name to "Archive Extraction" engine, even when you are still on MetaDefender Core v4.

A brand new "compression archive engine" is to dedicatedly handle archive sanitization in v5 (archive compression), that helps relevant archive handling performance improvement, easier for maintainance and further feature development.

Notes when you are still on MetaDefender Core v4 (not upgraded yet)

The legacy "Archive" engine (now name changed to "Archive Extraction" engine) on MetaDefender Core v4.x will still cover both extraction and compression archive tasks as usual.

Even though your MetaDefender Core v4.x might now have a new "Archive Compression" engine auto downloaded, but that engine cannot be deployed (marked as "permanently failed" in relevant engine status check API). This behavior is expected, and not causing any impact to MetaDefender Core processing flow, or its performance though.

- Just in case your client integration has a check for engine status before triggering scan request, then you might want to disable "Archive Compression" engine status on MetaDefender Core v4 using REST API. Detailed instruction: Archive Compression Engine Permanently Failed on MetaDefender Core v4
- When you later upgrade to MetaDefender Core v5 from v4, then you might need to manually re-enable "Archive Compression" engine to make it active and running on MetaDefender Core v5.

Notes for licensing

For offline deployment (MetaDefender Core is not connected to the Internet directly for online update), you will need to re-activate again on OPSWAT portal to retrieve latest engine list in .yml file including "Archive Compression" engine if you are currently licensed with Deep CDR.

Meanwhile, for online deployment, this manual procedure is not required since MetaDefender Core will cover that itself.

Import & export configurations

Not only workflow settings like in previous versions, but also all other general settings (update, security, data retention, server, hashes to skip) and most importantly, user settings, will be able to export and import on MetaDefender Core version 5.

Learn more at Import/Export configuration

Backward compatibility supported

The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

Windows application event logging disabled by default

Applicable only to Windows OS based MetaDefender Core.

Help avoid duplicated logging with separate file option, and improve overall processing performance. Of course, when being still valid to you, you are still supported to enable it back.

CSRF token header renamed

CSRF token originally created for security strengthening and being used by MetaDefender Core management console UI since version 4.21.0 via header `oms_csrf_token`

To get it compliant with some load balancers which do not accept underscore in header name by default, and save you from additional load balancer configuration step, we have now changed the header name to `oms-csrf-token`

This change does not impact to any REST API based client integration.

More comprehensive and accurate error messages on REST APIs

Impacting on following REST API endpoints:

- `PUT /admin/config/session`
- `PUT /admin/config/update`
- `PUT /admin/config/scan`
- `PUT /admin/config/skip`
- `PUT /admin/config/auditlog`
- `PUT /admin/config/webhook`
- `PUT /admin/config/rule`

Upload time and queue time logic updated in JSON scan response

Impacting on following REST API endpoints:

- `GET /file/{data_id}`
- `GET /hash/{md5/sha1/sha256}`
- `GET /archive/{data_id}`
- `POST /file/{async}`

Added new field:

- `receive_data_timestamp`: The timestamp when upload progress started (first byte received) (in milliseconds)
- `upload_time`: Total time elapsed for upload process (in milliseconds)

Modified existing field (new logic):

> **Backward compatibility supported**
> The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

- `upload_timestamp`: The timestamp when upload progress finished (all bytes received) (in milliseconds)
- `queue_time`: Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue time = start_time - upload_ timestamp) (in milliseconds)

Release version 5.0.0 also comes with variety of outstanding features and improvements.

For detailed information about our latest release please view our Release notes

Installation

Starting with MetaDefender Core 4.20.0, the centralized/shared database deployment model is supported (all MetaDefender Core instances share the same database), along with traditional standalone database mode (every MetaDefender Core instance has its own database to manage).

- Standalone mode
- Shared Database Mode

Installation

1. Download the package of your choice from My OPSWAT
2. Install the package on your computer via the Command Line or via the Wizard
3. Open a web browser and point to http://<server name or IP>:<port>
    - The default port is **8008**
    - In case of problem check Inaccessible Management Console page
4. Complete the required steps of the basic configuration wizard
5. You must activate this deployment to use its features

Installation notes

- If the MetaDefender Core package **dependencies are not installed** on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.
- During installation the databases might need to be upgraded. This could take noticeable time depending on database size (eg.: length of scan history).

Standalone mode

Wizard Installation

The Install Wizard is only for the Windows installer (.msi file).

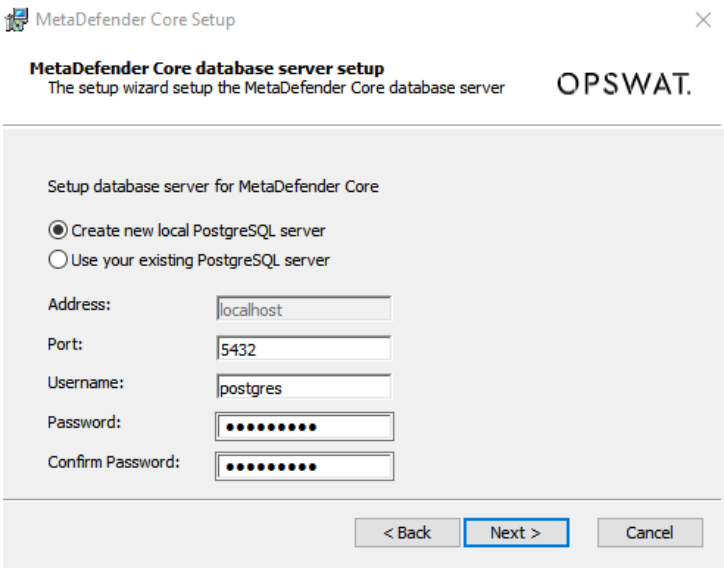
Since MetaDefender Core 4.19.0, we have switched to using PostgreSQL DBMS which needs a dedicated database server. MetaDefender Core supports two modes to setup PostgreSQL server:

Local (Bundle) PostgreSQL server

MetaDefender Core will install a new PostgreSQL server locally in the same box with the product.

> **PostgreSQL user credentials**
> Unicode characters are NOT supported for "Username" and "Password"
>
> "Password" does not accept # (hash) as starting character (e.g. #ABC)

> **The ignition file**
> If the ignition file exist in the designated location, then MetaDefender Core will use the details defined in the ignition file, rather than what is passed via the installation wizard.
>
> More info about the ignition file based installation: Command Line Installation

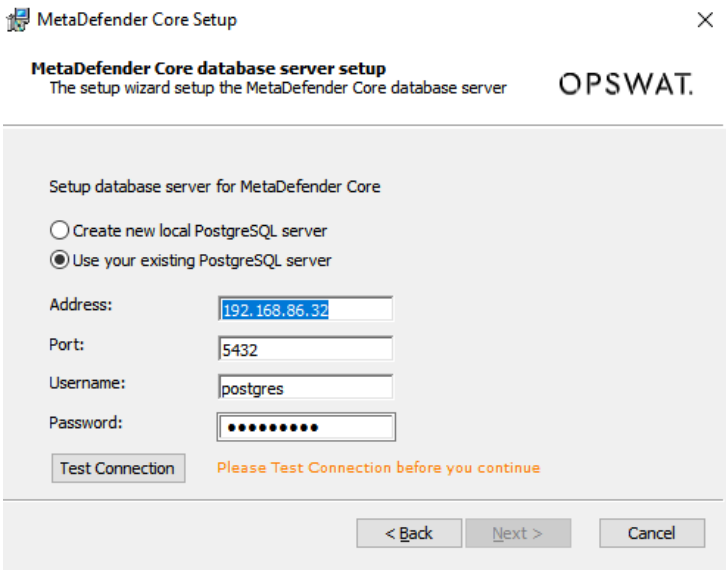


Remote (Existing) PostgreSQL server

MetaDefender Core will connect to remote pre-installed PostgreSQL running remotely.

> **PostgreSQL connection configuration**
> "Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

> **PostgreSQL user privilege requirements**
> MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.
>
> Details: PostgreSQL User Privilege Requirements



Command Line Installation

Ignition File

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Make sure to create the ignition file, if it doesn't exists before you go ahead and install MetaDefender Core via command line.

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

MetaDefender Core supports two modes to setup PostgreSQL server:

1) Local PostgreSQL server

- MetaDefender Core will install a new PostgreSQL server locally, next to the product.
  - A sample iginition file for PostgreSQL server information (***Only non-Unicode characters supported for "user" and "password"__***):

ometascan.conf bash

```
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
```

2) Remote PostgreSQL server

> **Info**
>
> **PostgreSQL version 12 (at least 12.3) or version 14 is required.**
>
> According to PostgreSQL notice, version 12 will stop receiving fixes on November 14, 2024, so we suggest that you use version 14.

MetaDefender Core will connect to a pre-installed PostgreSQL server running remotely and will setup its database.

Make sure to follow steps at Open Connection On PostgreSQL Server first, so that your remote existing PostgreSQL server to accept upcoming connection from MetaDefender Core.

A sample ignition file for PostgreSQL server information:

ometascan.conf bash

```
[dbserver]
type=remote
host=192.168.86.32
port=5432
user=existingserver_admin_user
password=existing_server_pass
```

After installing MetaDefender Core successfully, you may want to remove all credentials info created for [dbserver] section in ignition file for security reason.

> **Warning**
> If the MetaDefender Core package dependencies are not installed on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.

Debian package (.deb)

bash

```
sudo dpkg -i <filename> || sudo apt-get install -f
```

On Red Hat Enterprise 7.x Linux / CentOS package (.rpm)

bash

```
sudo yum install <filename>
```

For systems which enabled GPD check flag:

bash

```
sudo yum install --nogpgcheck <filename>
```

Red Hat Enterprise Linux 8.x package (.rpm)

Install the latest PostgreSQL repository RPM:

> **Note**
> Due to the shorter support cycle on Red Hat Enterprise Linux, all supported versions of PostgreSQL are not available on this platform. The PostgreSQL Yum Repository will integrate with the normal systems and patch management, and provide automatic updates for all supported versions of PostgreSQL throughout the supported lifetime of PostgreSQL. Reference: https://www.postgresql.org/download/linux/redhat/

bash

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

In case installing a remote PostgreSQL on either the same MetaDefender Core server or another server, you need to install the following additional package. PostgreSQL version 12.3 or newer is recommended.

bash

```
sudo dnf install -y postgresql12-contrib
```

Install MetaDefender Core:

bash

```
sudo dnf install -y <filename>
```

Windows package (.msi)

On Windows systems it is possible to install the product by running the corresponding .msi file.

From command line interface it is also possible to install the product by executing

bash

```
msiexec /qn /i <msi file name> <option key>=<option value>
```

where the possible keys and their default values are the following:

Key	Default Value	Description
INSTALLFOLDER	\Program Files\OPSWAT\MetaDefender Core	Customize installation folder for product Example: `INSTALLFOLDER="D:\Products"`
RESTADDRESS	*	REST interface binding IPv4 or IPv6 address ('*' means that service listens on all IPv4 and IPv6 interfaces)
RESTPORT	8008	REST interface binding port
REPORT_ENGINE_ISSUE	true	Enable reporting of engine issue count. (possible values: "true" or "false").

Note

Reporting of engine issue count

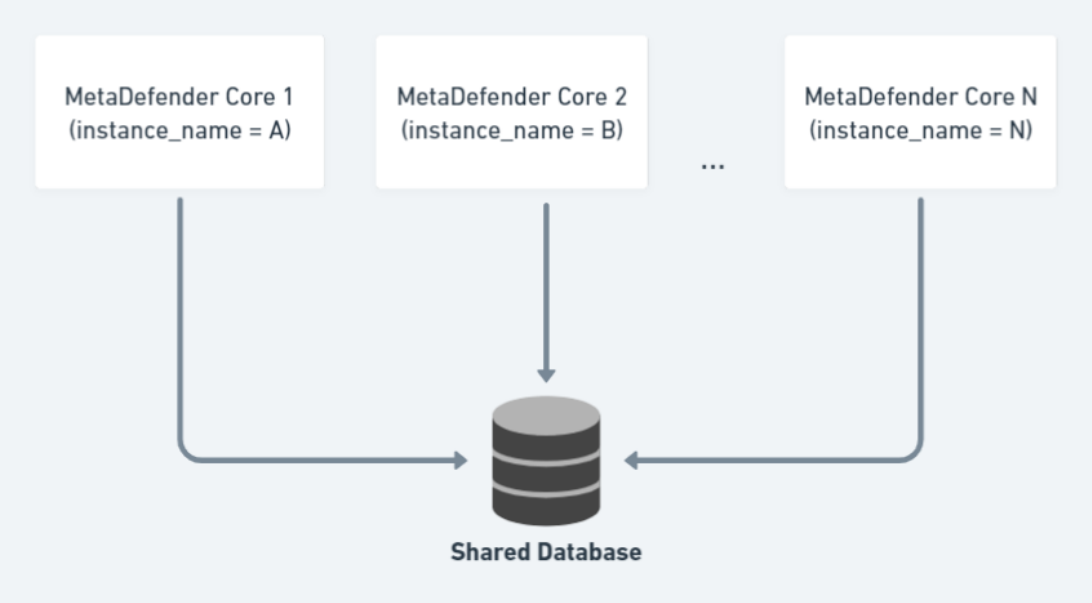
If reporting of engine issue count is enabled, MetaDefender Core server will send only the **number** of initialization errors and **number** of unexpected stops for the specific db/engine version. This information is sent over a HTTPS channel when the product downloads the latest package descriptors. This information is used for early detection of any specific 3rd party engine quality issues.

For details on using msiexec please consult Windows installer documentation.

Shared Database Mode

Starting with MetaDefender 4.20.0, you can use a centralized database deployment model, where all MetaDefender Core instances can share the same external database.

That helps auto-sync all product configurations (**except Single-Sign-On** that requires unique settings for each Core), processing history, live/on-going scan result query, audit, statistics data.



This article demonstrates how to install and setup MetaDefender Core 4.20.0 or newer with shared database mode.

Important notes & current limitations:

Currently not supported

- Does not support Central Management use-case with shared DB mode.
- Does not support to switch from shared database mode back to standalone database mode.
- Does not support to change instance_name of MetaDefender Core in shared database deployment model after installation.
- Does not support upgrading older MetaDefender Core from non-persistent mode to using the shared database mode.
- Does not support upgrading older MetaDefender Core from standalone mode to using the shared database mode.
- Does not support to upgrade multiple MetaDefender Core instances at once in shared database mode
    - Stop all Core instances' scanning service before performing the upgrade.
    - Upgrade each Core server one by one instead.

- For each Core, when the upgrade finished, then you can start using that Core for scanning without waiting for the other Core instances' upgrade to be done.

> **Disclaimer**
> MetaDefender Core does not support mode switching.

Requirements for shared database setup:

- The central remote PostgreSQL server (for shared database) could be utilized with a built-in (native) PostgreSQL database server installed with one of the MetaDefender Core instances in the network. Or using a dedicated existing PostgreSQL database server (see detailed requirements for PostgreSQL version at Recommended System Configuration).
- Before setting up, make sure to follow steps at Open Connection On PostgreSQL Server so that all MetaDefender Core instances could connect to the central remote PostgreSQL server.
- MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core, details: PostgreSQL User Privilege Requirements
- All MetaDefender Core instances must be installed on the same Operating System (Windows / Linux).
- The first MetaDefender Core instance connects to the shared database must be ready first (license activated, users/groups/roles created) before setting up other MetaDefender Core instances to connect and use that shared database.

Engine update:

- If engine updates is configured to be via Folder source, all MetaDefender Core instances must be able to access to the configured update folder, and the option DELETE FILES AFTER IMPORT will be unchecked and prevented for modification.

Authentication:

- Single-Sign-On (SSO) authentication settings (if using) cannot be shared among MetaDefender Core instances because SSO requires an unique login URL for every different Core.
- If an user is locked for login on a certain MetaDefender Core instance(e.g. exceeded the number of failed login times), that user is not prevented for logging in on any other instances.
- If an user is locked for sign-in on a certain MetaDefender Core instance (e.g. exceeded the number of failed login times), if the admin releases lockout on that user on that MetaDefender Core instance, that user is still not able to login on other MetaDefender Core instances.

Sanitized / DLP / Quarantined location changed after MetaDefender Core install:

Windows:

Shared storage path will be stored in 3 strings under key global in registry: quarantinepath, dlppath, sanitizepath as below

(Default)	REG_SZ	(value not set)
dbmode	REG_SZ	4
dlppath	REG_SZ	\\10.40.50.100\test
instance_name	REG_SZ	481rvj
quarantinepath	REG_SZ	\\10.40.50.100\test
restaddress	REG_SZ	*
restport	REG_SZ	8008
sanitizepath	REG_SZ	\\10.40.50.100\test

You can set either 1, 2 or 3 different paths for the 3 settings above (means that quarantined, DLP and sanitized files will be stored in same or different storages corresponding). In case shared storage is not set in installation, values of 3 settings above will be empty.

Restart MetaDefender Core service after changed.

Linux:

Open file `/etc/ometascan/ometascan.conf` and set shared storage path for 3 keys quarantinepath, dlppath, sanitizepath under section [global]

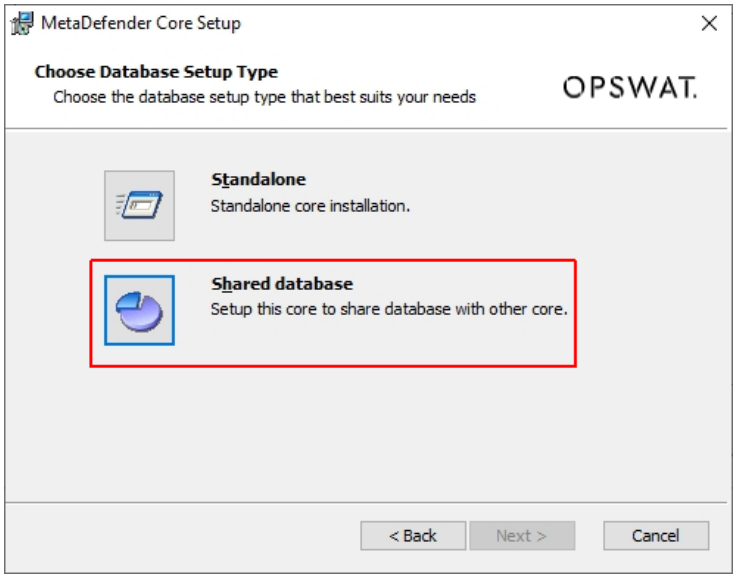
Restart ometascan service after changed.

Wizard Installation (Windows Only)

> **Info**
> Only applicable to MetaDefender Core 4.20.0 or newer using wizard on Windows OS.

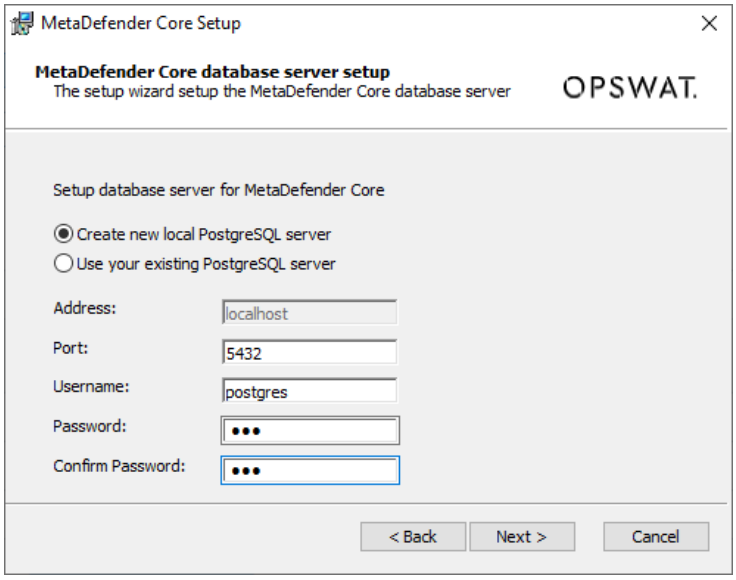
Clean install scenario

1. **Choose Shared database option:**

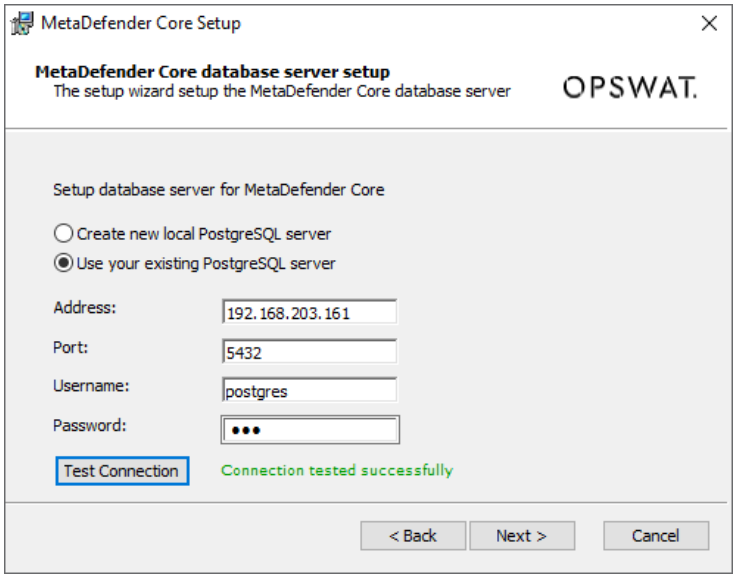


2. **Setup database server for MetaDefender Core:**

You can choose to create a native / built-in PostgreSQL server and setup the shared database on (then other MetaDefender Core instances will want to connect to that PostgreSQL server afterwards)



Or choose to connect to a pre-setup existing PostgreSQL server:



PostgreSQL connection configuration

"Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

PostgreSQL user privilege requirements

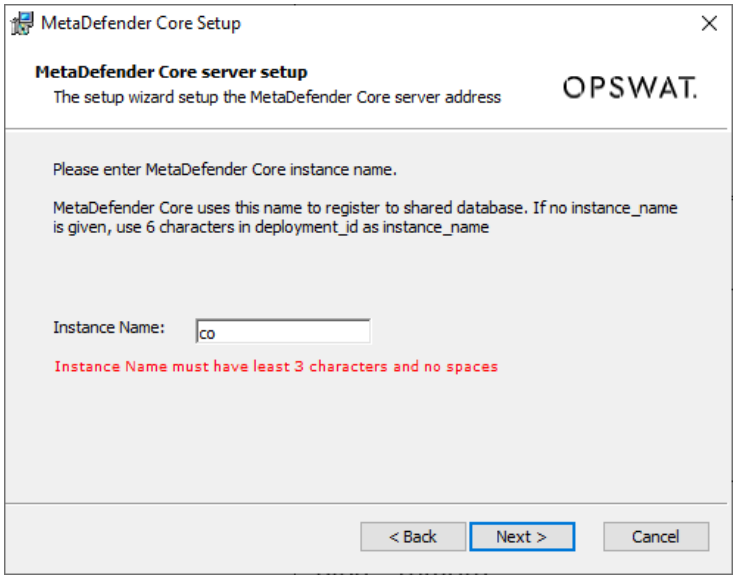
MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.

Details: PostgreSQL User Privilege Requirements

3. **Create an unique instance name for that MetaDefender Core (different from other MetaDefender Core instances):**

Instance name must contain at least 3 characters and no white space.

If you choose to leave it blank, which means it will use the default instance name with 6 characters extracted from that MetaDefender Core deployment ID.

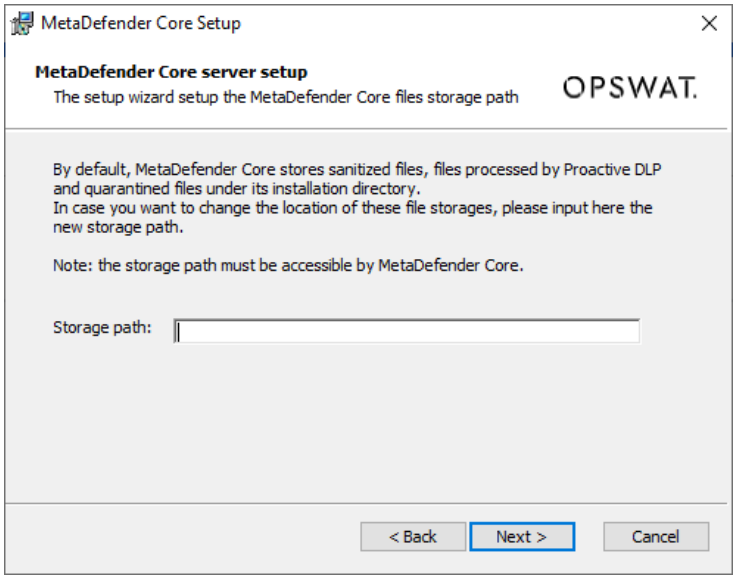


4. **Input the file storage path (sanitized, DLP, quarantined files)**

In the shared database model, all MetaDefender Core instances could use the same folder path to store all their own sanitized, DLP, quarantined files after processed.

The default location is: ***<MetaDefender Core installation folder>\data\ {sanitized|dlp|quarantine}***

If you wish to use that default location, just leave the input box blank and press Next.



Otherwise, you can specify a different location of sanitized/dlp/quarantined files. It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core.

For example:

- C:\new_folder
- \192.168.1.1\new_folder

Command Line Installation

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Clean install scenario

1. Create ignition file if not existed

- Windows: **C:\OPSWAT\ometascan.conf**
- Linux: ***/etc/opswat/ometascan.conf***

An example of ignition file:

ometascan.conf bash

```
eula=true

[global]
dbmode=4
instance_name=MetaDefenderCoreA
storage_path=D:\path\

[user]
name=admin
password=admin
email=admin@local

[dbserver]
type=remote
host=192.168.1.1
port=5432
user=postgres
password=your_own_password
```

Field	Required	Mapping to wizard UI	Possible values
global/dbmode	optional default is 1 (standalone)		1 - Standalone 2 - Central Hub deployment 3 - Non-persistent **4 - Shared**
global/instance_name	optional default is 6 characters of deployment ID		Unicode and number At least 3 characters No whitespace
global/storage_path	optional default is empty		Path to storages (DLP, quarantined, sanitized files) It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core. For example: • C:\folder • \192.168.1.1\folder

2. Run installers using command line to install

Please refer to Command Line Installation for installation command.

Deployment automation support

Note

An Ignition file could be utilized for either / both of following use-cases:

1. Pre-define PostgreSQL server information for MetaDefender Core to connect and setup database: only required on Linux installation, or Windows with command line installation. Learn more details at:
   a. Standalone Mode > Command Line Installation (applicable on version 4.19.0 or newer).
   b. Shared Database Mode > Command Line Installation (only applicable on version 4.20.0 or newer).
2. Automating to accept EULA, create a default local admin user and import MetaDefender Core configurations upon installed: Those steps could be automated using ignition file described in this page.

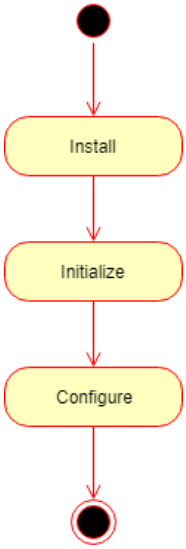
Ignition file location to create:

- **Windows**: `C:\OPSWAT\ometascan.conf`
- **Linux**: `/etc/opswat/ometascan.conf`

The product supports fully automated deployment. It means that it can be installed and configured with no human interaction.

The automated deployment can be split to three steps on a high level:

1. Installation
2. Initialization
3. Configuration



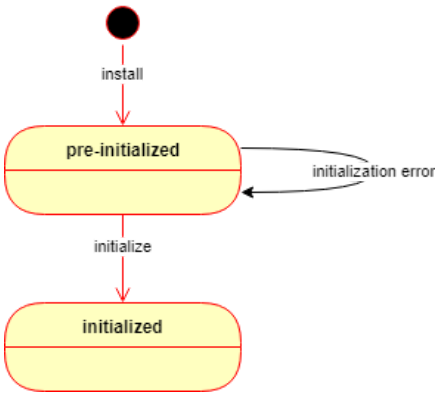
Installation

To automate the installation, install the product from the command line and provide the installation-time options as parameters to the installer. For further details see Command Line Installation

After the installation is complete, the product starts up and waits in a pre-initialized status. The product may be initialized in two ways:

1. Manually using the Wizard Setup or
2. Automatically using an *ignition file* (see below).

> **Warning**
> If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried after fixing the problem (e.g. placing the ignition file in its lookup location) and restarting the OPSWAT MetaDefender Core service.
>
> Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

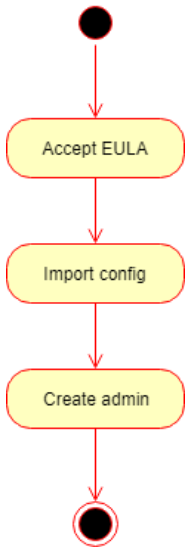


Initialization

Initialization is the process of bringing the product to an operable status.

Basically the initialization consists of the following steps:

1. Accept the End User License Agreement (EULA),
2. Import product configuration and
3. Create the first administrator user account.



Ignition file

The initialization process can be configured in a file called the *ignition file*.

> **Warning**
> In Linux, the ignition file and the folder /etc/opswat should have the permission 755

Ignition file location

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

The ignition file must be in conf format.

ometascan.conf bash

```
eula=true
[user]
name=admin
password=admin
email=admin@local
apikey=44c96cbb4e4e6c0a1d16bcb3c3da24def317
[config]
import=config_package.zip
import_password=123
[security]
session_cookie_samesite=Lax
[internal]
skip_startup_pages=true
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
private_username=my_internal_db_user
private_password=mypassword
[ocm]
ocm_url=https://my.opswat.com
reg_code=XXXXXXXX
```

Ignition file fields

The ignition file accepts the following fields:

Section	Key	Required	Accepted Values	Description
	eula	Optional (mandatory when having [user] section)	true false (default)	Whether to accept the End User License Agreement. This key must be set at the top of ignition file. This key must be set to "true" to accept the EULA. Any other value will cause the initialization to fail. When having [user] section defined, then this key must be set.
[user]		Optional		Initial administrator user account properties. The Administrator role is granted to the account.
	name	Optional	string	User name for the initial administrator user account.
	password	Optional	string	Password for the initial administrator user account. WARNING! Clear text password The password in this configuration file must be stored in its clear-text format and as so it may be visible for unauthorized parties.
	email	Optional	string	E-mail address for the initial administrator user account.
	apikey	Optional	string	API key for the initial administrator user account.
[config]		Optional		
	import	Optional	string	Path to a file in json/zip format that contains a previously exported configuration to be imported. Zip-format file contains a full configuration including user management settings, so it will overwrite [user] section.
	import_password	Optional	string	Password to decrypt .zip configuration package.
[security]		Optional		

Section	Key	Required	Accepted Values	Description
	session_cookie_samesite	Optional	Strict Lax (default) None	Declare if session cookie should be restricted to a first-party or same-site context.
[internal]		Optional		
	skip_startup_pages	Optional	true false (default)	Skip all welcome and other startup pages (e.g. upgrade result) when installing / upgrading MetaDefender, and going straight to the product index default page.
[dbserver]		Optional		
	type	Mandatory when having [dbserver] section	local remote	Where the PostgreSQL database server should physically locate. "remote" means PostgreSQL server and MetaDefender Core are not in the same machine.
	host	Mandatory when having [dbserver] section	string	IP address / domain name of the server where PostgreSQL server locates. "localhost" can be used when applicable.
	port	Mandatory when having [dbserver] section	number	Port of PostgreSQL server is listening for connections from clients (i.e. MetaDefender Core).
	user	Mandatory when having [dbserver] section	string	PostgreSQL server's user. SUPERUSER privilege is required for MetaDefender Core to setup its database and extensions for the first time. **Only non-Unicode characters supported.**

Section	Key	Required	Accepted Values	Description
	password	Mandatory when having [dbserver] section	string	PostgreSQL server's user credentials. **Only non-Unicode characters supported.**
	private_username	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
	private_password	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
[ocm]		Optional		
	ocm_url	Optional	string	URL to My OPSWAT server.
	reg_code	Optional	string	A registration code to enroll to My OPSWAT server.

Detailed initialization process

1) After the product has been started, it looks for the ignition file in the designated location.

Windows: C:\OPSWAT\ometascan.conf ** Linux: /etc/opswat/ometascan.conf

** 2) If an ignition file is found, then

2.1) It gets validated, and if it is valid, then based on the information found in the ignition file:

- The EULA is accepted,
- The configuration is imported,
- The administrator account is created.
- If any of the above steps fails, then the error is logged, and the initialization gets terminated.

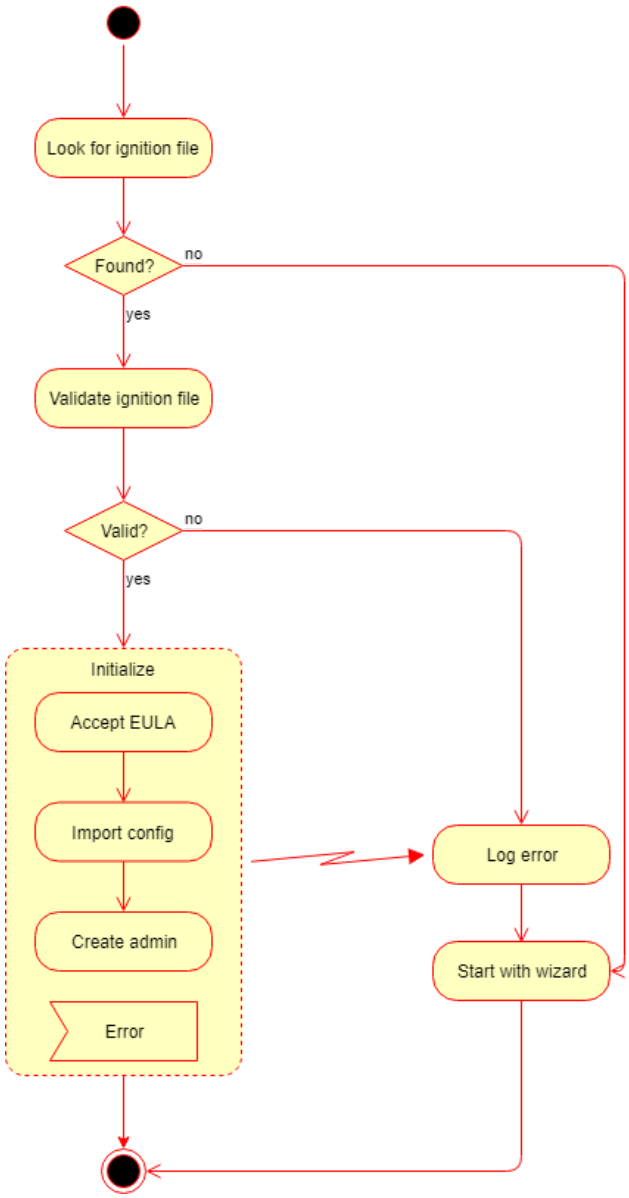
> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

2.2) If it is not valid, then the error is logged, and the initialization gets terminated.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

3) If there is no ignition file, then no initialization is performed.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.



Warning

If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried fixing the problem (e.g. placing the ignition file to its lookup location) and restarting the OPSWAT MetaDefender Core service.

Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

Configuration

After the initialization is complete, the product is ready with the default and the imported configuration.

This configuration can be later changed calling the configuration API functions. For further details about the API see Configuration related APIs API

CIS Level 1 Guidelines

Support Red Hat Enterprise Linux 9
For more details about Center for Internet Security (CIS) please refer to this document

Instruction steps

I. Install OpenSCAP

bash bash

```
yum install openscap-scanner scap-security-guide
```

II. Generate a result file and a HTML report using OpenSCAP scanner tool

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

III. Remediation of CIS Level 1 issues

Generate a remediation script based on the `ssg-rhel9-ds.xml` file:

bash

```
oscap xccdf generate fix --profile
xccdf_org.ssgproject.content_profile_cis_server_l1 --fix-type bash --output
remediations.sh /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

And execute remediation script.

bash

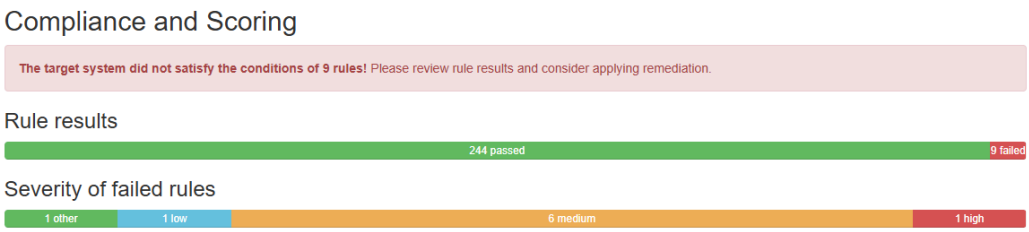
```
./remediations.sh
```

IV. Review the results after remediation

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

Example:



Notes

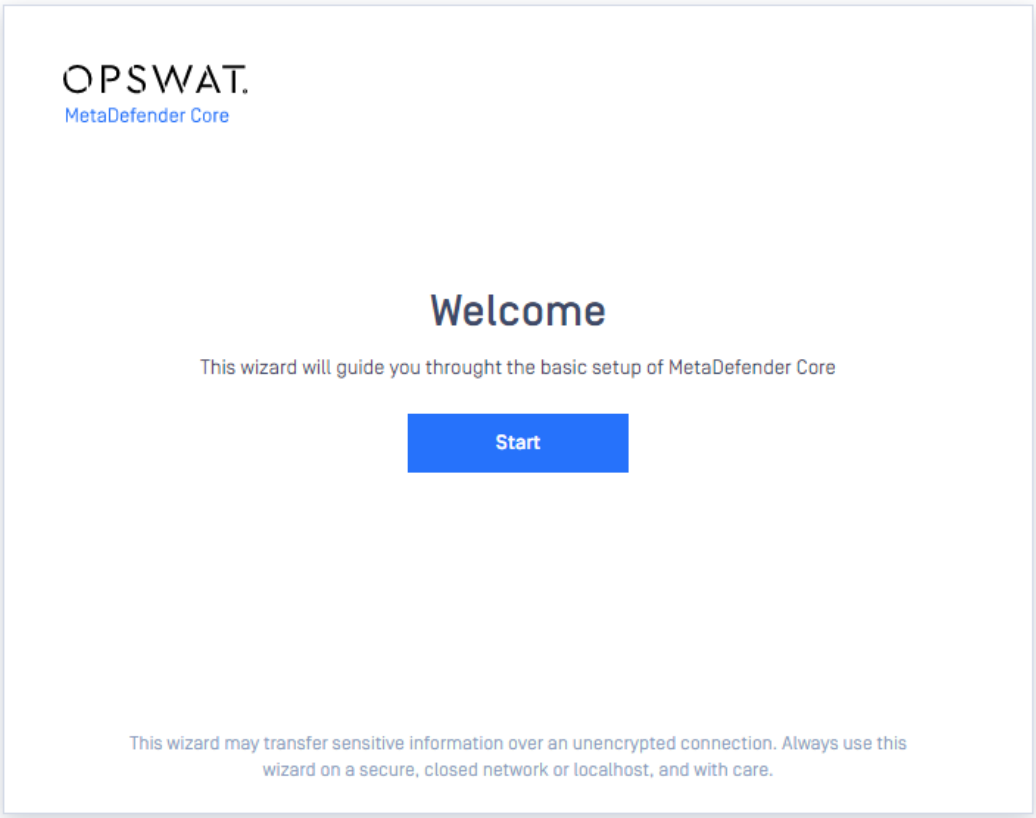
- CIS Level 1 requires `/tmp` folder to be mounted in a separate partition. Please ensure that that new partition have enough disk space for MetaDefender Core to run.
- CIS Level 1 requires that "Ensure No World-Writable Files Exist".
    - For now, when freshly installing MetaDefender Core, all its binary files meet the requirement.
    - When installing/updating engines, some engines might create additional files for its operation, and it might violate this requirement. In this case, you need to again execute the remediation script in the step III.

Wizard Setup

Introduction

Access the MetaDefender Core management console for the first time (i.e. http://localhost:8008), you are to complete a quick walkthrough for wizard setup in order to be able to use the product.

The administrative management console will be available only upon you have successfully finished this wizard stage.

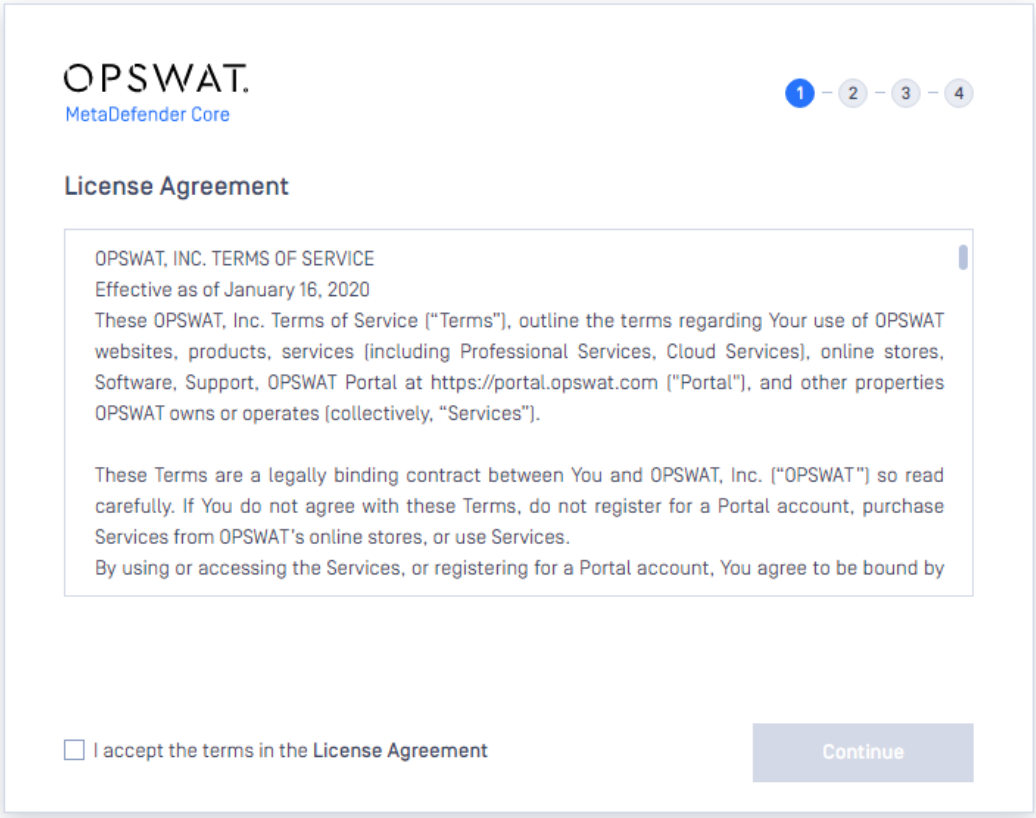


Warning

This wizard may transfer sensitive information over an unencrypted connection. Always use this wizard on a secure, closed network or localhost, and with care.

Basic configuration steps

End-User License Agreement



In the first page you can find the End-User License Agreement. You have to accept the terms before moving on.

Please read through the EULA (https://docs.opswat.com/policies/privacy-policy/portal-privacy-policy) carefully and if you agree with it, check `I accept the terms in the **License Agreement**` and click `Continue` to proceed.

Local Admin User Setup

The next step is to set up a default local administrator account.

This account will be the first one being able to access the administrative management console, and to create accounts for other users. You have to fill out all fields in this page to be able to move forward. When you are done, click NEXT to continue.

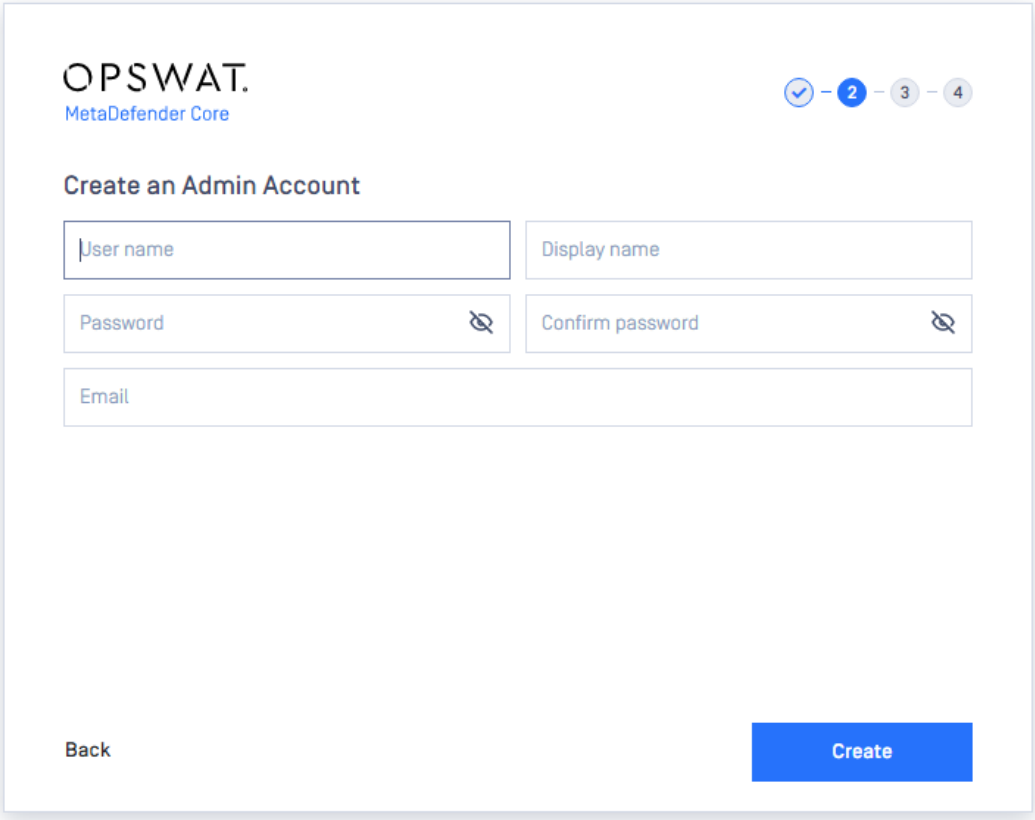
Info - User Directory

The administrator account, that is created via the basic configuration wizard, is always added to the LOCAL user directory as a member.

Learn more about user directory: User directories

The following information is required for the administrator account:

ACCOUNT NAME	The unique name of the account that is used at the time of login and in log messages for accountability.
ACCOUNT DISPLAY NAME	Name of the person bound to this account. This name (appended to the name of the account's user directory) is displayed in the top right corner of the administrative management console.
PASSWORD	Password of the user bound to this account that is used at the time of login. Passwords sent clear-text As long as TLS is not configured for the basic configuration wizard, passwords are sent clear-text over the network and may be disclosed to unauthorized parties. As a mitigation action: 1. Either use the wizard on *localhost* or on a direct network connection, or 2. Enabling HTTPS as soon as possible and change the password immediately if it has already been set.
EMAIL	Email address of the person bound to this account.



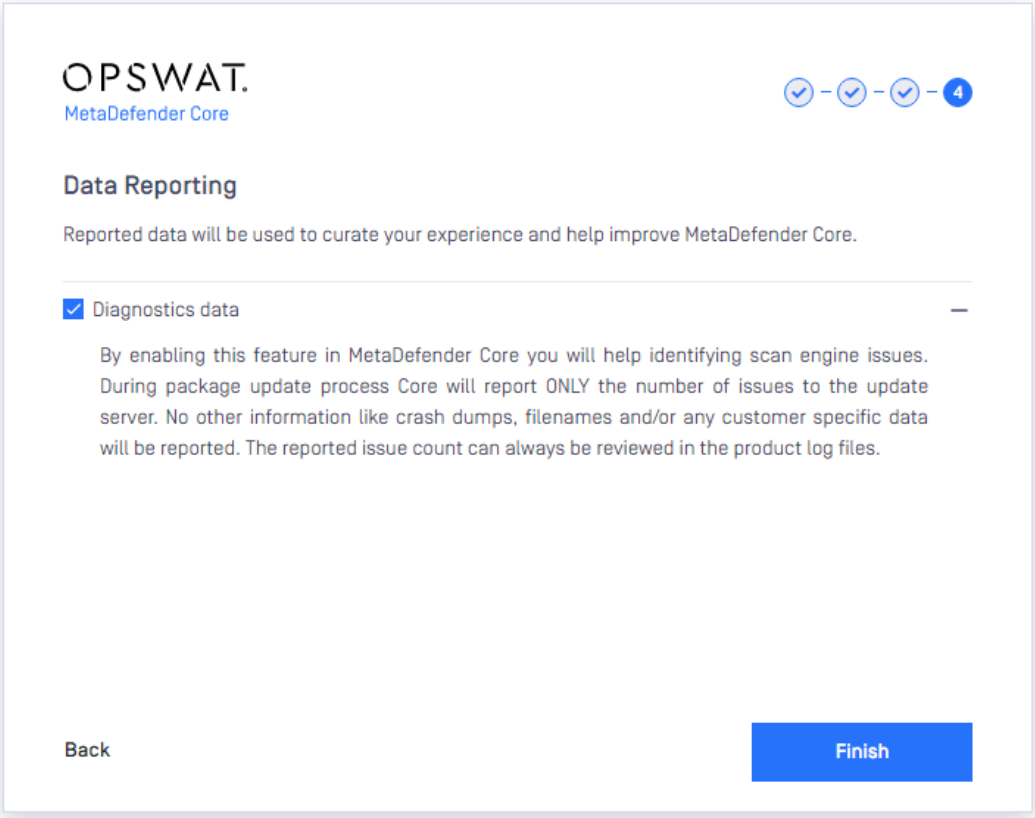
License activation

For license activation details see License Activation

Data Reporting

Enabling this option to allow MetaDefender Core collect its own engine issues and report back to OPSWAT for further analysis.

MetaDefender will not collect any other information.

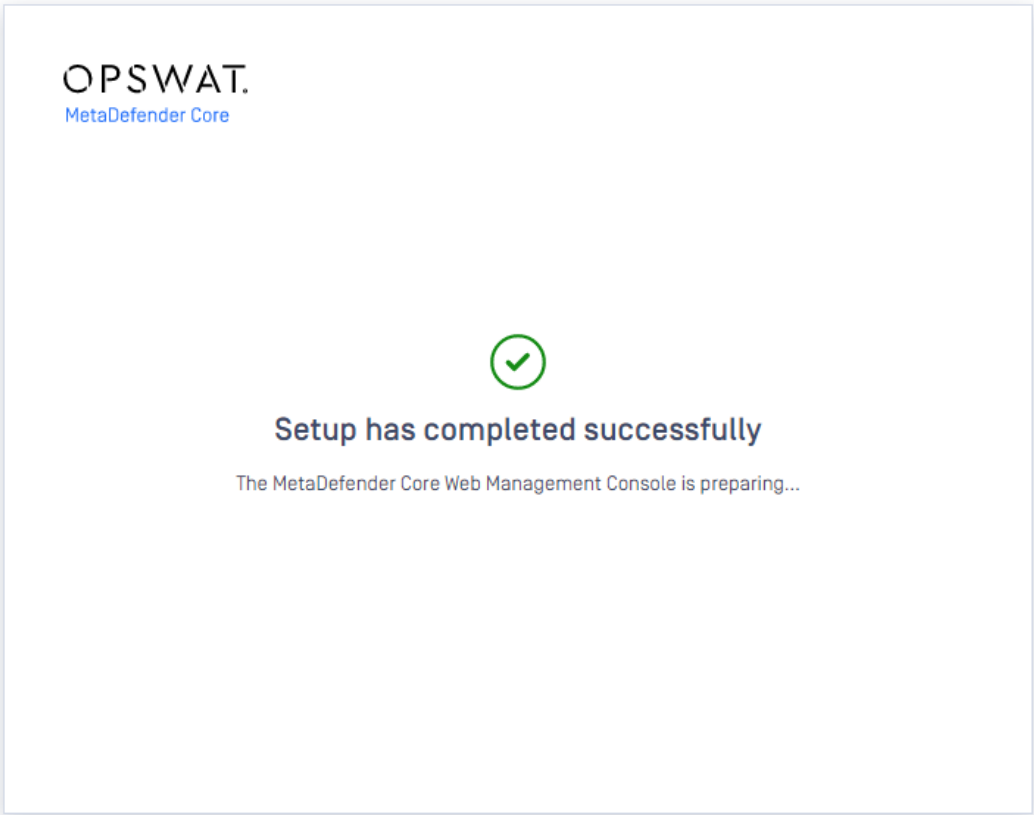


Wizard completion

After you have completed every steps you are ready to finish the wizard and start using the product. Click the `Fishish` button to complete the wizard.

> **Warning**
> The product's service will be restarted and the browser will be redirected to the administrative management console. This could take several seconds.

You can login to the Web Management Console with the administrator user that have just been created in the previous steps.



Setup HTTPS

Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network. Websites, like the Web Management Console, are able to use TLS to secure all communications between their servers and web browsers.

The TLS protocol aims primarily to provide confidentiality (privacy) and data integrity between two communicating computer applications.

Warning

No TLS for the wizard

By default, TLS is not enabled for the basic configuration wizard. As a consequence sessions between the wizard's backend and the browser may be insecure.

Info

Performing the same steps as for the Web Management Console, it is possible to set up TLS for the basic configuration wizard. Remember completing the TLS setup before launching the wizard.

For instructions to set up TLS see Enabling HTTPS

License Activation

##

To activate your installation go to the **Settings > License** menu in the Web Management Console. If you have no valid license, you will only see your installation's **deployment ID**. You will also see a warning in the Web Management Console header.

After successful activation, the product will start downloading the latest available scan engines and malware databases. You can follow the status of the scan engine installation on the **Inventory > Modules** page.

When all the licensed engines are installed, you can start using the MetaDefender Core to process files - File Processing.

Deployment ID

Each MetaDefender Core instance represented by an unique system fingerprint called `Deployment ID`

MetaDefender Core licensing is purely based on deployment ID activated. Therefore, everytime your MetaDefender Core is activated with a different deployment ID, then it will be captured as a different activation record accumulated into your total activation times allowed.

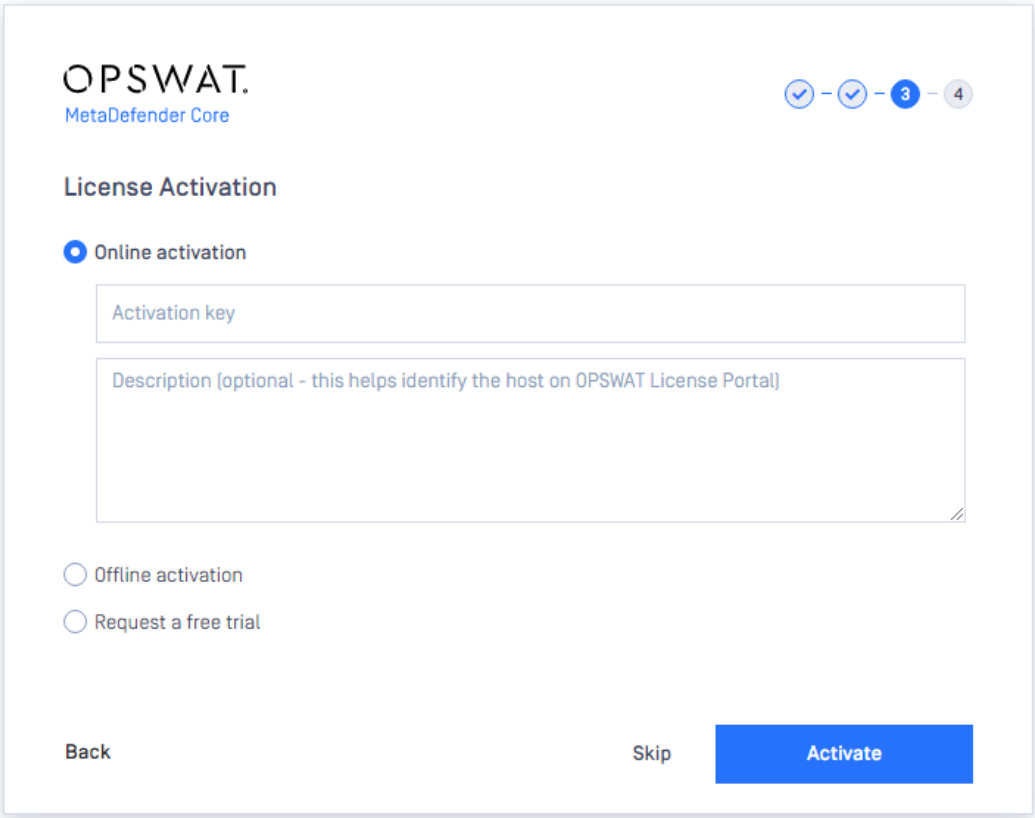
> **Notice**
> Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you alter the system where MetaDefender Core relies on (e.g. network card, OS hostname).
>
> Deployment ID is an UUID version 4 random based string. Learn more at: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you update the system where MetaDefender Core relies on.

Activation Modes

There are three different options for the product activation:

- Online activation: MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.
- Offline activation: Requiring you to upload a license file (.yml). That license file could be retrieved by activating your license via OPSWAT Portal. Follow the displayed instructions for details.
- Request trial key online: For evaluation purpose, you can receive an trial Activation key via email. Follow the displayed instructions for details.



Online activation

MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.

Required to input your Activation License key, and optional for description (but we recommend this for troubleshooting and auditing purpose if needed).

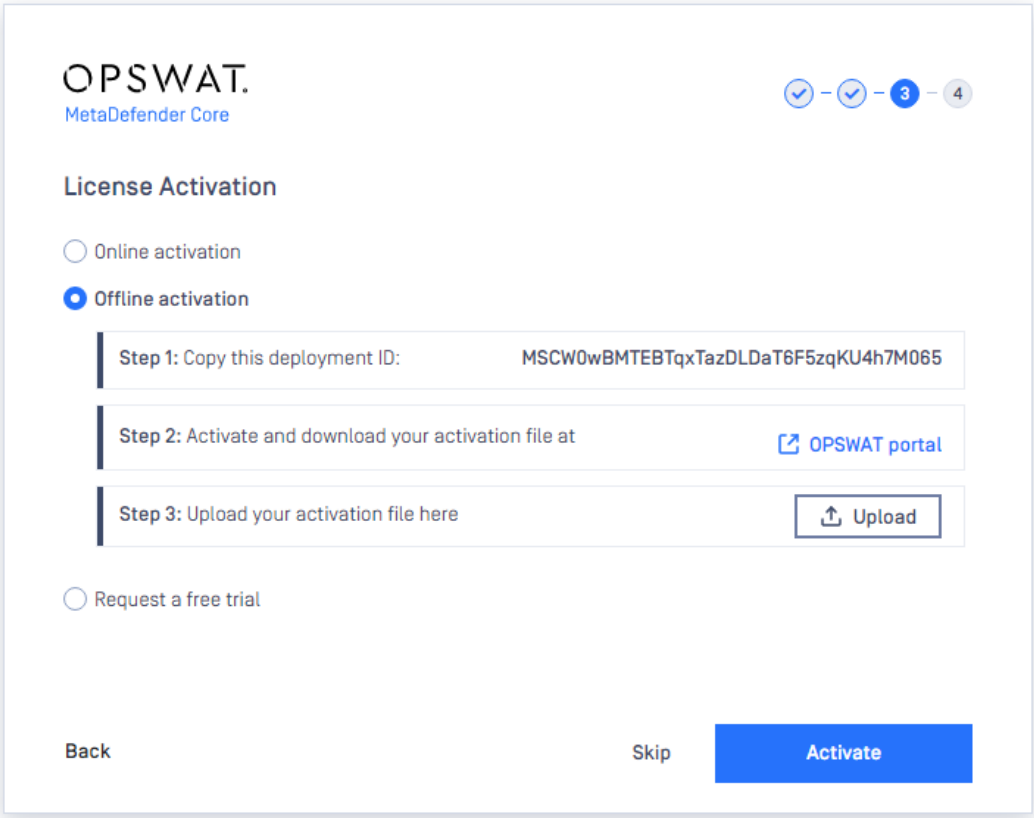
Warning

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts' for license activation:

- https://activation.dl.opswat.com Note: *IP address-based whitelisting on your firewall might fail after some time since OPSWAT uses CDN (Content Delivery Network) to faster delivery updates over the world, and IP address of edge servers might change over time.*

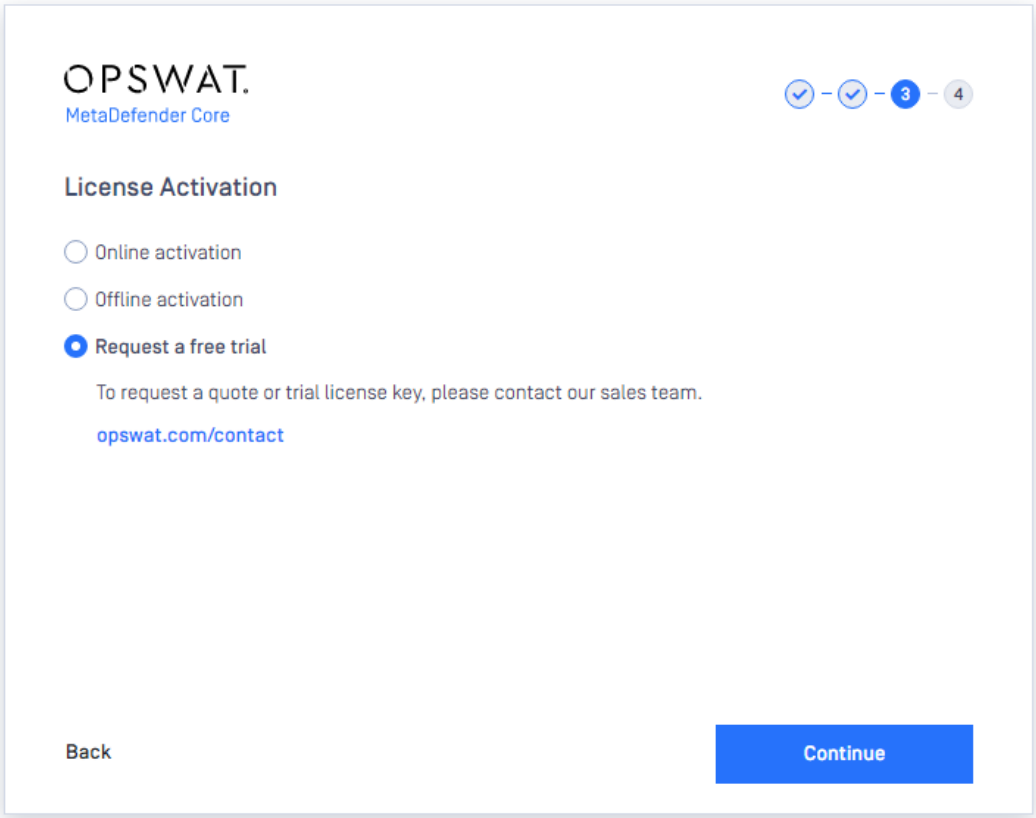
Offline activation

With no internet connection on the server the Metadefender v4 instance may be activated indirectly from a different machine, that has internet connection. The Deployment ID of the Metadefender v4 instance and the the Activation key received at the time of purchasing the product will be required. Follow the steps on the screen to activate the product offline.



Request trial key online

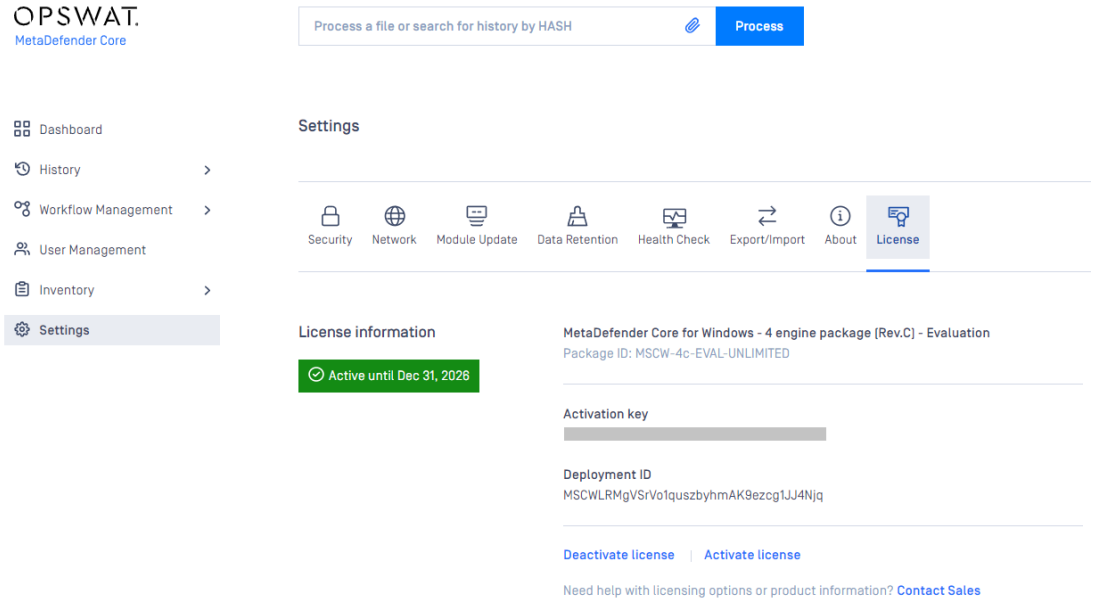
An evaluation license could be acquired by contacting our sales team.



Check Your License Details

For more license details and activating your MetaDefender Core, go to Settings > License menu tab on the Web Management Console:

- Product name: product name as on your order (e.g. MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation)
- Package ID: product identification as on your order (e.g. MSCW-4c-EVAL-UNLIMITED)
- Expiration: last day of license validity
- Activation key: key used for activating product
- Deployment ID: identification of this installation


OPSWAT.
MetaDefender Core
Process a file or search for history by HASH
Process
Dashboard
History
Workflow Management
User Management
Inventory
Settings
Settings
Security
Network
Module Update
Data Retention
Health Check
Export/Import
About
License
License information
Active until Dec 31, 2026
MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation
Package ID: MSCW-4c-EVAL-UNLIMITED
Activation key
Deployment ID
MSCWLRMgVSrVo1quszbyhmAK9ezcg1JJ4Njq
Deactivate license
Activate license
Need help with licensing options or product information? Contact Sales

Scripted license management

Using REST API calls there is an option for scripted activation and deactivation for MetaDefender Core servers with **no Internet connection or if your infrastructure is using automation to create/destroy MetaDefender Core instances**. In this chapter the steps of these two scenarios are described.

Requirements

- an installed MetaDefender Core instance without Internet connection
- another computer that has Internet connection and can run your activation scripts
- a manual or automated way to transfer data between the two computers

Activation steps

1. In order to activate MetaDefender Core, deployment ID and activation key are needed to generate the license file.
    a. Activation key should be purchased from OPSWAT.
    b. MetaDefender Core deployment ID can be queried by using REST API (for details see GET Get Current License Information API page)

> **Info**
> Save this deployment ID in your system, you might need this ID when the instance is unavailable at the time of deactivation.

2. Activate license and get the license file using the following URL, where <quantity> is 1.

activation call bash

```
curl -X GET https://activation.dl.opswat.com/activation?key=<activation-key>&deployment=<deployment unique-ID>&quantity=<quantity>
```

If the activation is successfully, a license file is downloaded. Save this file.

HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	license file	Activation was successfully.
200 Activation failed	error: '<user conform error message>'	Failed activation
200 Invalid parameter	error: 'Could not activate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not activate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
200 Invalid parameter	error: 'Could not activate your product because the quantity you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key, quantity or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Info
In case of any activation issue, contact OPSWAT support for help

3. Upload license file to MetaDefender Core: The license file should be uploaded to the MetaDefender Core v4 to activate the product. For details see page: POST Upload license key API

Deactivation steps

1. For deactivation, following infomation is required:
    a. Activation key
    b. MetaDefender Core deployment ID: can be queried by using REST API (for details see GET Get Current License Information API page)
2. Deactivate license using the following URL:

deactivation call bash

```
curl -X GET https://activation.dl.opswat.com/deactivation?key=<activation-
key>&deployment=<deployment unique-ID>
```

Response HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	result: ok	Successful deactivation
200 No active license found	error: 'Could not found any active license with the given parameters'	The license has not been activated yet or it has been deactivated already.
200 Invalid parameter	error: 'Could not deactivate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not deactivate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Important notes

Warning

Product activation is tied to several hardware, operating system and software parameters. In case of one or more major hardware or software parameter change the product might turn into deactivated status. Operating system updates, other software updates should not affecting the activation status.

If the product is online activated and has live Internet connection then in case of deployment ID change the product reactivates itself. If the product doesn't have Internet connection then the administrator is responsible to reactivate the product.

Because of the above administrator should consider licensing restrictions of the offline deployments if using any virtualization and/or containerization technologies.

File Processing

MetaDefender Core provides REST API interface for file processing business.

There are several ways to process files with MetaDefender Core:

- Analyze Files via Web Interface
- Process Files via REST API

Product Upgrading

> **Info**
> Simply run installer file, details could be found at Installation

> **Info**
> For command line mode, upgrading (e.g. on **Linux**), prior to running the installer file, please make sure to **create the ignition file** with PostgreSQL server information included. Details:
>
> - Standalone DB mode: Command Line Installation (Windows and Linux)
> - Shared DB mode: Command Line Installation

Recommendation for MetaDefender Core upgrade

Our goal is to help you execute the product upgrade continuously, efficiently, and correctly.

PostgreSQL service upgrade

- **Bundled PostgreSQL Service**: If you are using the bundled PostgreSQL server included in the installer, the PostgreSQL service will be automatically upgraded to the newer version (if available). No additional action is required for this process.
- **Remote PostgreSQL Service**: For remote PostgreSQL servers, customers are responsible for managing and upgrading the service independently. Upgrading MetaDefender Core does not update the remote PostgreSQL service.

Before the upgrade

Prepare a proper disaster recovery plan (DRP) including some common practices:

- Backup, or take a snapshot of your current working MetaDefender Core server.
  - This is to help quickly rollback for any upgrade failure might cause later for any reason.
- Optionally backup your PostgreSQL database, especially when using remote shared database mode.
  - Reference: https://www.postgresql.org/docs/current/backup.html
  - Please do it after stopping all linked MetaDefender Core instances to avoid potential data collision.
- Lastly, if backing up entire server is not feasible, then at least export MetaDefender Core configuration package Import/Export configuration just in case you'll need to start off a new server with current MetaDefender Core configuration to keep the scanning service running without any impact.

On the PostgreSQL server, double check server resource to ensure sufficient RAM memory and disk space to be preserved for the upgrade process.

- 8GB RAM available at least during the upgrade.

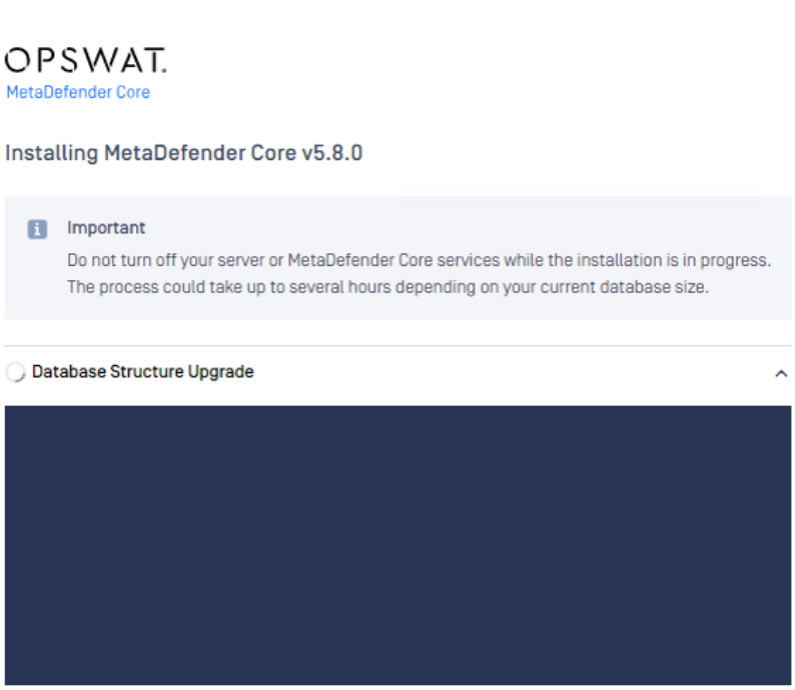
- Disk availability is greater than MetaDefender Core database size.

While doing the upgrade

Applicable for shared database deployment:

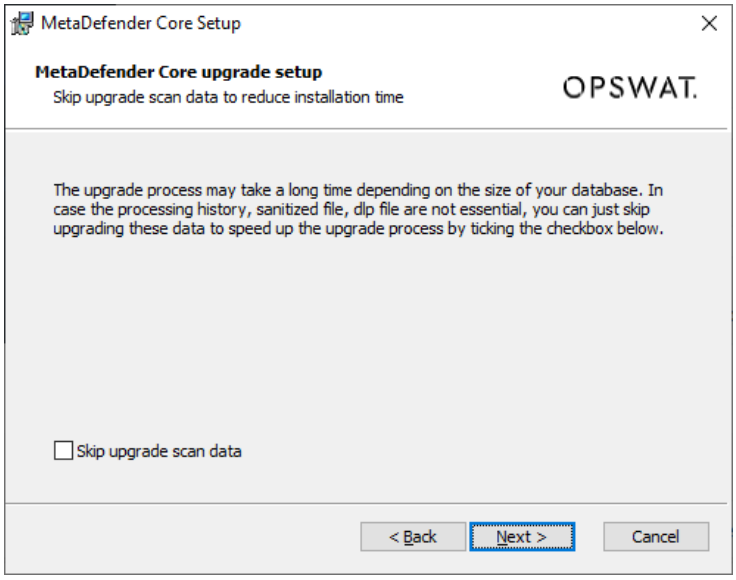
- Stop all MetaDefender Core instances' scanning service before performing the upgrade.
- Upgrade one (1) MetaDefender Core instance at a time.
- Proceed to upgrade MetaDefender Core instance where the bundle local PostgreSQL server is used for shared database setup.
- Otherwise is using remote PostgreSQL then the upgrade order does not matter.

For large size database, mostly for scan history, expecting long time to be spent on the upgrade process.



The product supports an option to skip data migration for scan history to expedite the process drastically and mitigate any failure risk. You can manually backup all data from your PostgreSQL server for offline usage if any.

Check "Skip upgrade scan data" to enable this feature while doing the upgrade.



or add following key in the ignition file:

none

```
[internal]
skip_upgrade_scan_data = 0
```

Troubleshooting

If the upgrade process ever runs into an error:

- Double check your network and services, applicable to remote PostgreSQL server.
- Optionally configure MetaDefender Core to skip migrating scan history.
  - Windows
    - Open Windows registry, and navigate to HKLM\Software\OPSWAT\Metascan
    - Add a new key `internal/skip_upgrade_scan_data` with value = 1
  - Linux
    - Open MetaDefender Core configuration file `/etc/ometascan/ometascan.conf`
    - Add new key into the file
      - [internal]
      - skip_upgrade_scan_data=1
  - Note: This key will be auto removed upon the upgrade finishes.
  - Then restart MetaDefender Core service to retry the upgrade.
    - Might retry this a few times until the upgrade is done.

Just in case the upgrade process still fails, please contact OPSWAT support with the support package - How to Create Support Package? This will help us analyze the root cause and figure out an optimal solution for your system failure.

Using external Load Balancer

Application Load Balancing (Layer 7)

This page provides information about recommended ways to use sessions with a Layer 7 load-balancer.

Most load-balancers have the ability to provide cookies for pinning subsequent traffic from a client to the appropriate server. This method is called sticky session, session persistence or session affinity. Using cookies efficiently requires the client to know when a cookie should be sent, should not be sent or should be deleted.

Sticky session load balancing

Each Metadefender Core instance has its own databases and application sessions that cannot be seen by other instances. Therefore, in order to get the related data/response to our queries we should ask the appropriate Metadefender Core v5 server. On the other hand, to keep the advantages of the used load balancing method, cookies should not be sent if it is not necessary.

Single file scanning

Step	Stage	Task	Cookie usage
1.	Sending file	Initiate processing a file on the client side. Send file through the load-balancer. (See POST File Analysis API)	Cookie should not be sent
2.	Sending file	Save the cookie and the data_id you got from the load-balancer	Save cookie
3.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Send cookie saved in step 2
4.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Send cookie saved in step 2

Batch scanning

Step	Stage	Task	Cookie usage
1.	Open batch	Initiate processing file(s) in batch. Request a batch ID through the load-balancer. (See POST Initiate Batch API)	Cookie should not be sent
2.	Open batch	Save the cookie and the batch_id you got from a Core server through the load-balancer.	Save cookie
3.	Sending files	Send file through the load-balancer. (See POST File Analysis (using batch header) API)	Send cookie saved in step 2
4.	Sending files	Save the data_id you got from the load-balancer.	-
	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Send cookie saved in step 2
5.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
6.	Getting batch status	See GET Batch Status API	Send cookie saved in step 2
7.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Send cookie saved in step 2
8.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API	Send cookie saved in step 2

If it does not matter which upstream server responds, the queries should be sent without cookie.

> **Info**
> It is recommended not to send cookies when it's not necessary to allow load-balancer to use its own method to share the load between Metadefender Core v5 servers.

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration through load-balancer

OPSWAT products that support HTTP load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	4.3.4	-
OPSWAT Client	Windows: 7.6.247.0 Mac: 10.4.243.0	-
MetaDefender Email Security	4.3.0	-
MetaDefender ICAP Server	4.3.0	-
MetaDefender Vault	1.3.0	-
MetaDefender for Secure Storage	2.0.7	-

DNS Load Balancing

Using this method is logically similar to Layer 7 load-balancing.

Briefly how it works

Client uses a domain name to send a query to a server. Client's DNS server has more "A" records for that name with different IPs. When a client resolves the server's domain name DNS server randomly chooses an IP for that name to send back. When a session is used on the application layer, client should know the IP address of the Core server that handles that specific session. In every other case, client should resolve the domain name with DNS query to let requests to be balanced between Metadefender Core servers.

Single file scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Sending file	Initiate processing a file on the client side. (See POST File Analysis API)	Use IP saved in step 2
4.	Sending file	Save the data_id got from the Core server	-
5.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Use IP saved in step 2
6.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Use IP saved in step 2

Batch scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Open batch	Initiate processing file(s) in batch. Request a batch ID. (See POST Initiate Batch API)	Use IP saved in step 2
4.	Sending files	Send file to the specific Core server with the basch ID saved in step 3. (See POST File Analysis (using batch header) API)	Use IP saved in step 2
5.	Sending files	Save the data_id you got from the Core server.	-
6.	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Use IP saved in step 2
7.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
8.	Getting batch status	See GET Batch Status API	Use IP saved in step 2
9.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Use IP saved in step 2
10.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API).	Use IP saved in step 2

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration via DNS load-balancing

OPSWAT products that support DNS load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	does not support yet	-
MetaDefender Client	does not support yet	-
MetaDefender Email Security	does not support yet	-
MetaDefender ICAP Server	does not support yet	-
MetaDefender Vault	does not support yet	-

SSL connection for PostgreSQL communication

> **Note**
> The SSL connection configuration is possible thanks to the support made by PostgreSQL. For reference: https://www.postgresql.org/docs/14/ssl-tcp.html
>
> MetaDefender Core is tested against this mode on particular version 4.21.1

Step 1: Generate SSL certificate for PostgreSQL server

> **Note**
> Following instructions are for self-sign certificate. For production environment, making sure to use the certificate signed by your trusted CA.

1) Generate a private key

bash

```
openssl genrsa -des3 -out server.key 1024
```

Sample output:

bash

```
Generating RSA private key, 1024 bit long modulus ........++++++
...............++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

Type your desired password / pass phrase to encrypt the private key

2) Remove the pass phrase to automatically start up the PostgreSQL server

bash

```
openssl rsa -in server.key -out server.key
```

> **Note**
> On Linux, make sure to set permission on the server.key file

bash

```
chmod og-rwx server.key
```

3) Create a self-signed certificate

bash

```
openssl req -new -key server.key -days 3650 -out server.crt -x509
```

You will be prompted to enter detailed information which is incorporated into your self-signed certificate request.

none

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:CA
Locality Name (eg, city) [Default City]:San Francisco
Organization Name (eg, company) [Default Company Ltd]:OPSWAT
Unit Name (eg, section) []:OPSWAT
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:dummy_email@opswat.com
```

For self-signed certificate, use the server ceriticate as the trusted root certificate:

bash

```
cp server.crt root.crt
```

Copy **server.key**, **server.crt**, **root.crt** to PostgreSQL data folder. For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data**
- Linux: **/var/lib/ometascan/pg_data/**

Step 2: Configure PostgreSQL server for SSL authentication connection

1) Create a custom config file (e.g. ssl.conf) for PostgreSQL.

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\postgres\ssl.conf**
- Linux: **/var/lib/ometascan/postgres/ssl.conf**

2) Content of ssl.conf:

none

```
ssl = on
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
```

3) Modify pg_hba.conf

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data\pg_hba.conf**
- Linux: **/var/lib/ometascan/pg_data/pg_hba.conf**

Add following (modify **::1/128** if needed to match with your environment requirement)

none

```
hostssl all all      ::1/128      scram-sha-256
```

After the modify, the content of pg_hba.conf should be like this (example):

> **Note**
> In a Linux environment, a local connection is required to upgrade from version 5.5.1 or earlier to version 5.6.0 or later.

none

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local         DATABASE  USER  METHOD  [OPTIONS]
# host          DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl     DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostgssenc    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnogssenc  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
local    all all                    scram-sha-256
hostssl all all      ::1/128      scram-sha-256

#More content.......
```

4) Reload the services

- Standalone DB mode: Restart **ometascan** service (MetaDefender Core service)
- Shared DB mode: Restart **ometascan-pg** / **ometascan-postgresql** service (MetaDefender Core PostgreSQL service) and then **ometascan** service (MetaDefender Core service)

Test your SSL database connection:

bash

```
psql.exe "sslmode=require dbname=postgres user=postgres host=localhost port=5432"
Password for user postgres:
psql (12.7)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
Type "help" for help.
```

This message indicate that you are now using SSL connection:

bash

```
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
```

Performance and Load Estimation

> **Disclaimer**
> These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Some factors that affect performance

- MetaDefender Core version
- MetaDefender Core engine package and configuration
  - set of engines (which and how many)
  - product configuration (e.g., thread pool size)
- System environment
  - server profile (CPU, RAM, hard disk)
  - client application location - remote or local
  - system caching and engine level caching
- Dataset
  - encrypted or decrypted
  - file types
    - different file types (e.g., document, image, executable)
    - archive file or compound document format files
  - file size
  - bad or unknown (assume to be clean)
- Performance tool

How test results are calculated

Performance (mainly scanning speed) is measured by throughput rather than unit speed. For example, if it takes 10 seconds to process 1 file, and it also takes 10 seconds to process 10 files, then performance is quantified as 1 second per file, rather than 10 seconds.

- total time / total number of files processed: 10 seconds / 10 files = 1 second / file.

Standalone DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.2.416.0

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

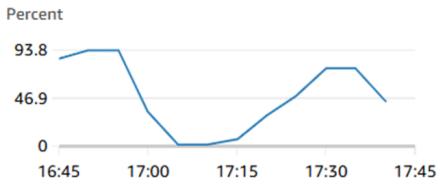
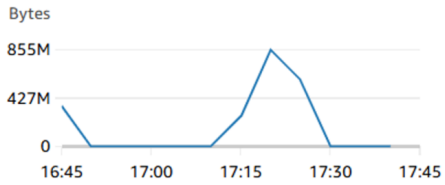
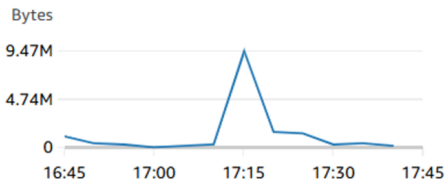
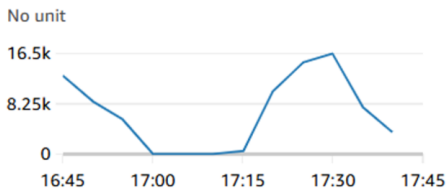
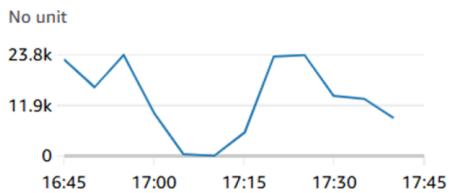
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV)	24.2	47.2 / 80	37.3 / 41.1		315,310	0.0114
Deep CDR	19.9	48.9 / 81.8	32.6 / 37		383,442	0.0094
Proactive DLP	17.1	36.8 / 58.2	30.4 / 33.6		446,228	0.0081
Vulnerability	10.1	59.2 / 85	29.3 / 33.3		755,495	0.0048

MetaDefender Core with common engine packages:

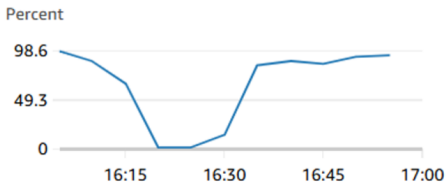
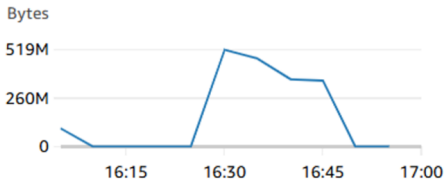
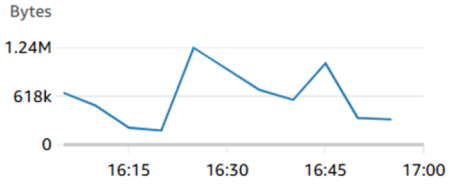
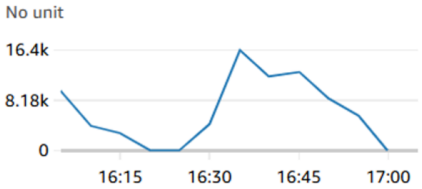
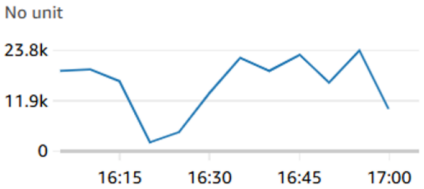
Use case	Scan duration	Avg./Max CPU usage	Avg./Max RAM usage	Avg./Max Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV) + Deep CDR	30.8	68.1 / 88	40.1 / 46.8		247,744	0.0145
Metascan (8 AV) + Deep CDR + Proactive DLP	31.2	74.2 / 98.6	41.5 / 51.1		244,567.3	0.0147
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	31.8	73.5 / 98	40.6 / 51.5		239,953	0.015

System resource utilization charts

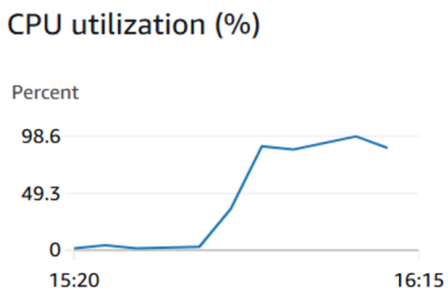
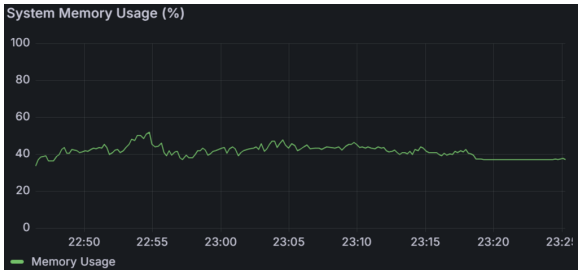
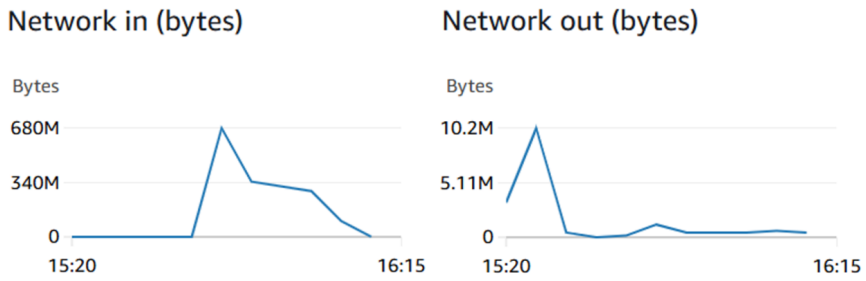
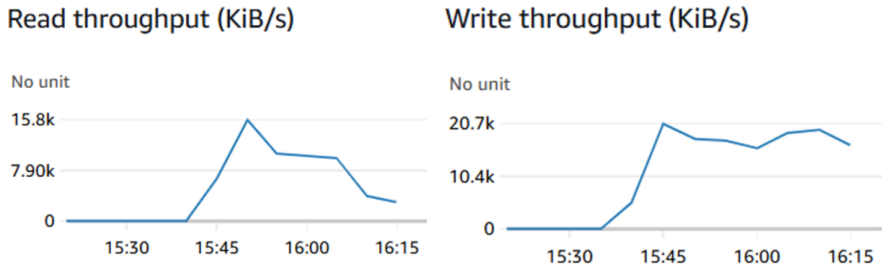
MetaDefender Core with Metascan (8 AV engines):

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

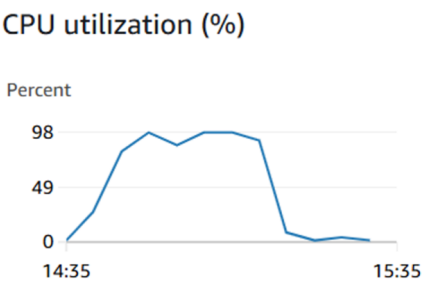
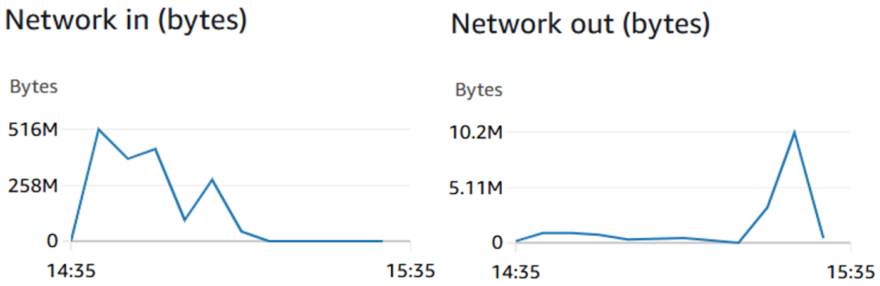
MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

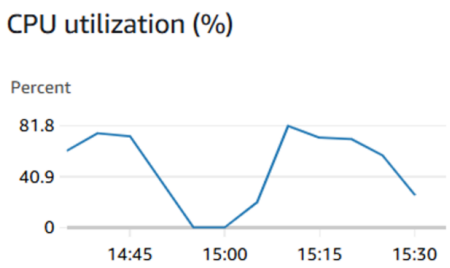
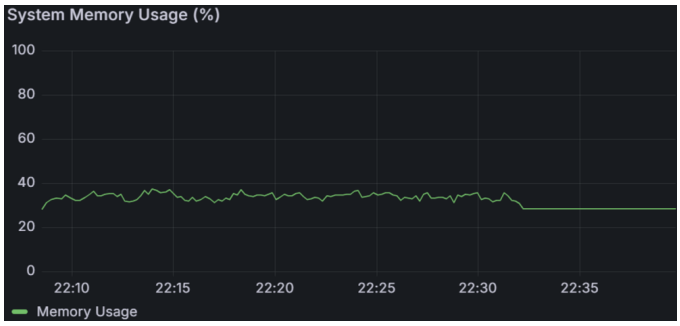
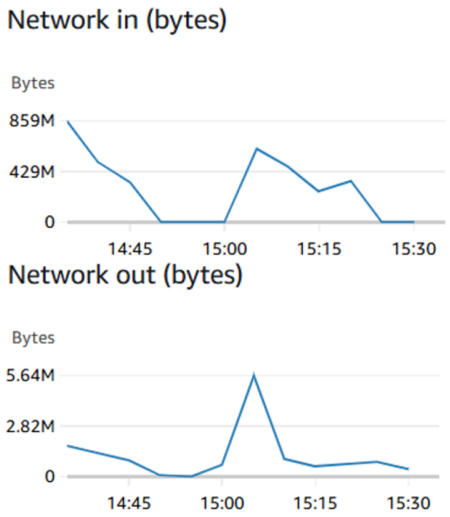
MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

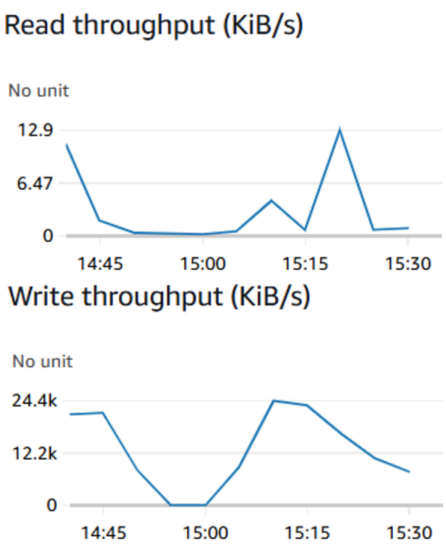
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

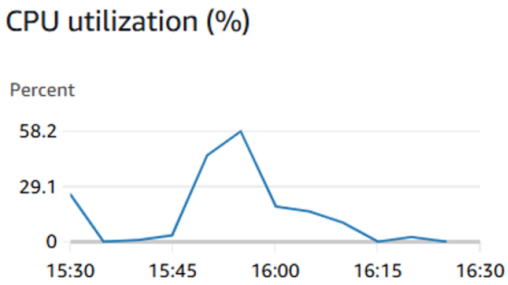
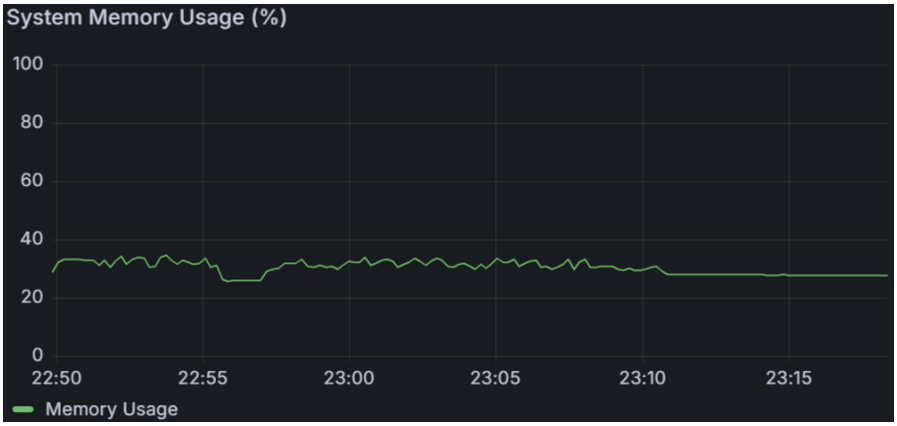
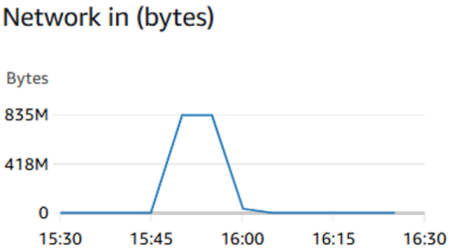
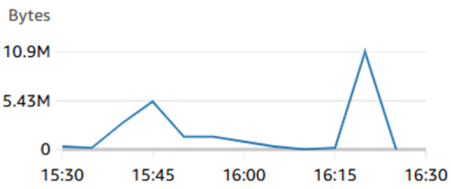
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

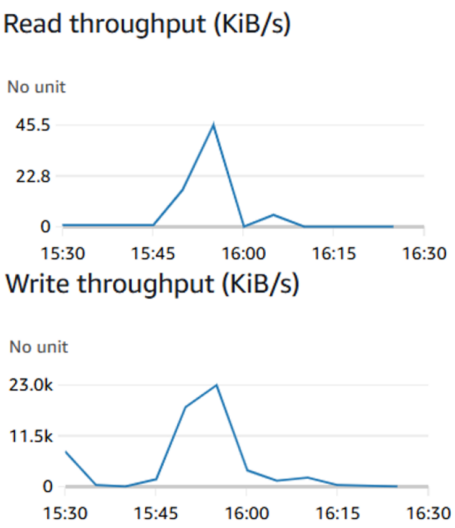
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

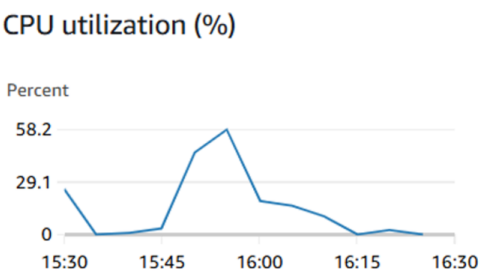
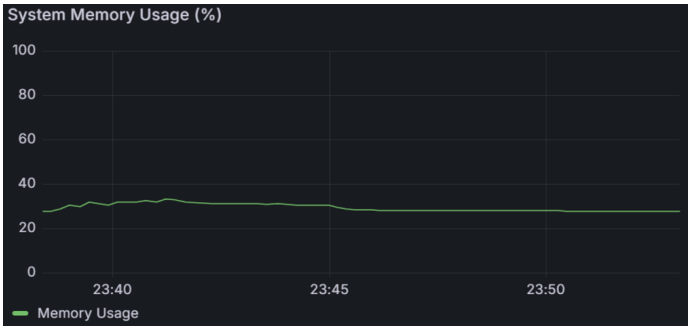
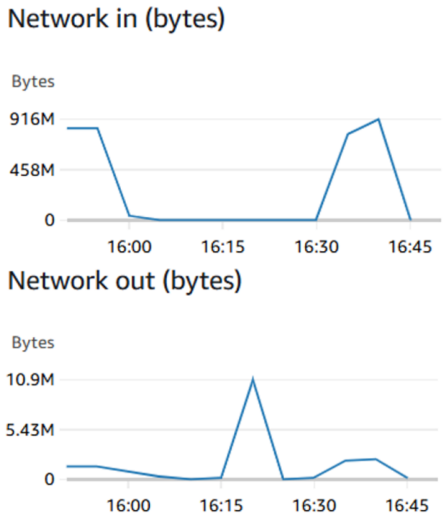
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	 

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability:

	Utilization charts
CPU usage	
Memory usage	
Network usage	

Utilization charts

Disk usage



Standalone DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Ubuntu 22.04	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.57-236

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

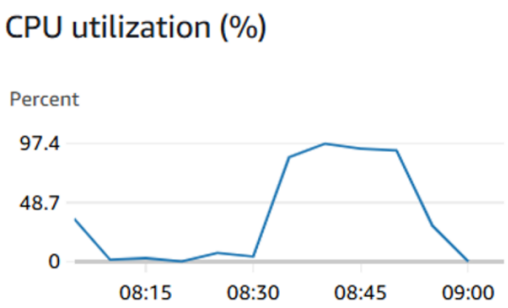
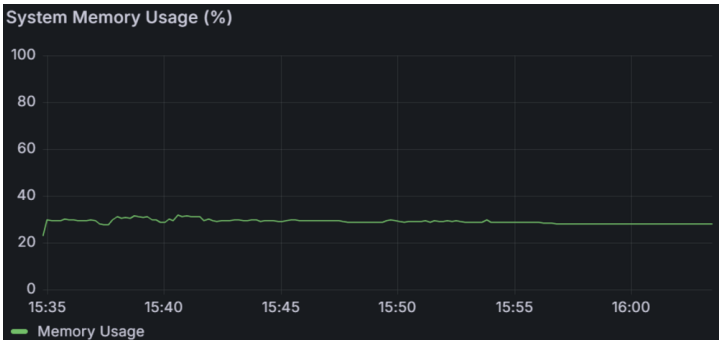
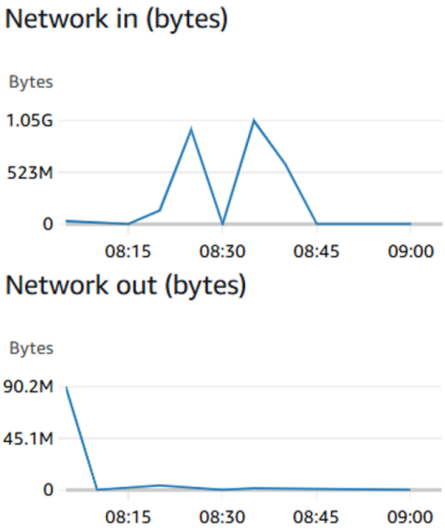
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV)	30.2	69 / 98	29 / 32		252,666	0.0142
Deep CDR	19.4	48 / 62	36 / 49		393,325	0.009
Proactive DLP	16.5	34 / 39	32 / 48		462,455	0.008
Vulnerability	12	50 / 65	46 / 46		635,875	0.006

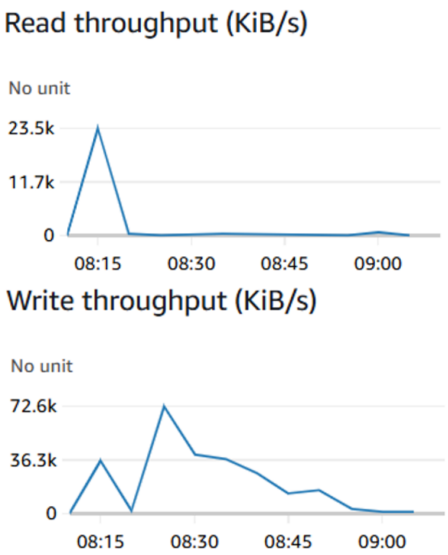
MetaDefender Core with common engine packages:

Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV) + Deep CDR	39.8	78 / 99	62 / 73		191,721	0.0188
Metascan (10 AV) + Deep CDR + Proactive DLP	43.5	61 / 84	47 / 67		175,414	0.0205
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	44.8	62 / 88	69 / 83		170,324	0.0211

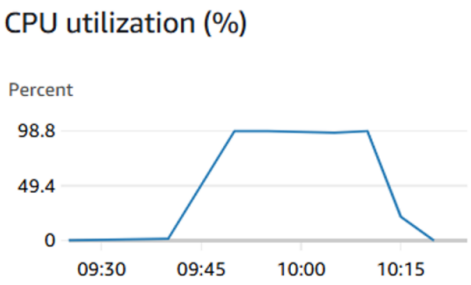
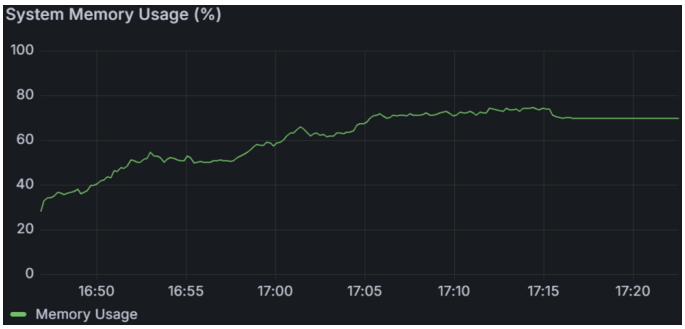
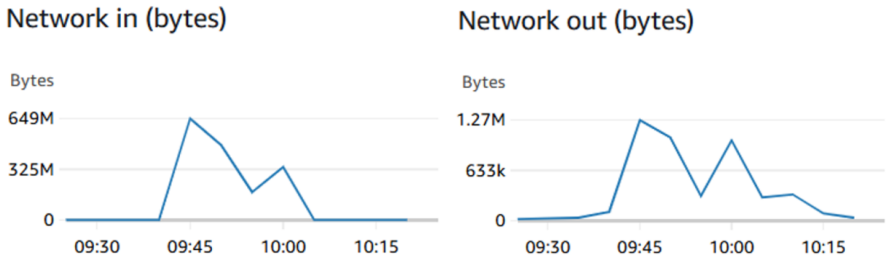
System resource utilization charts

MetaDefender Core with Metascan (10 AV engines) :

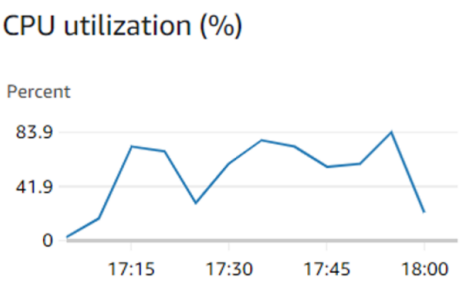
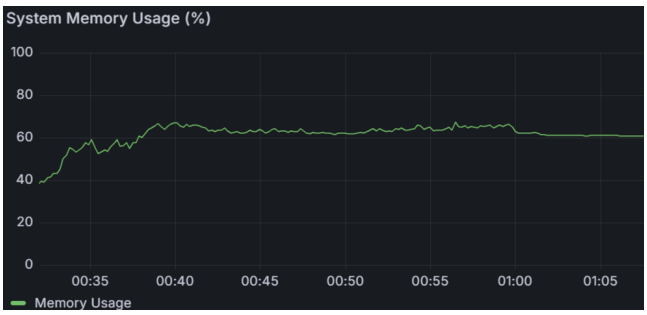
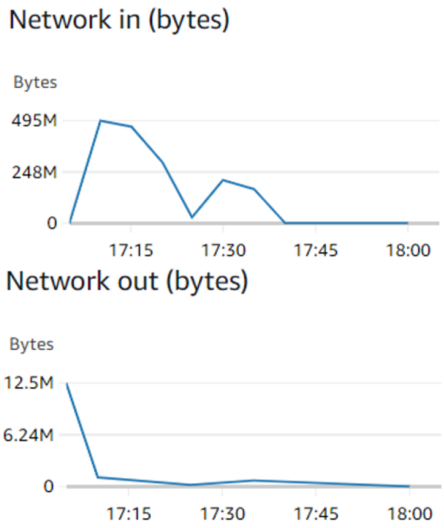
	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR :

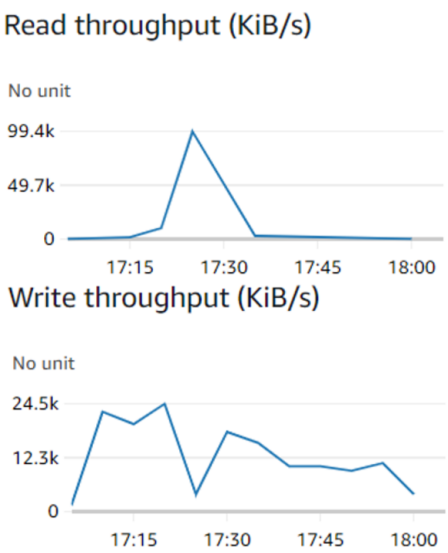
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP :

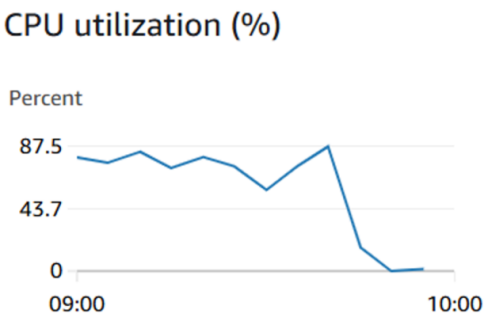
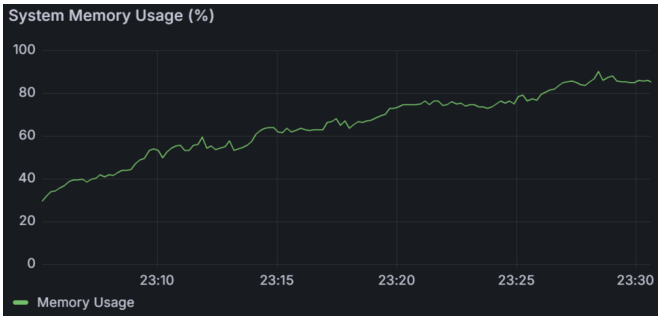
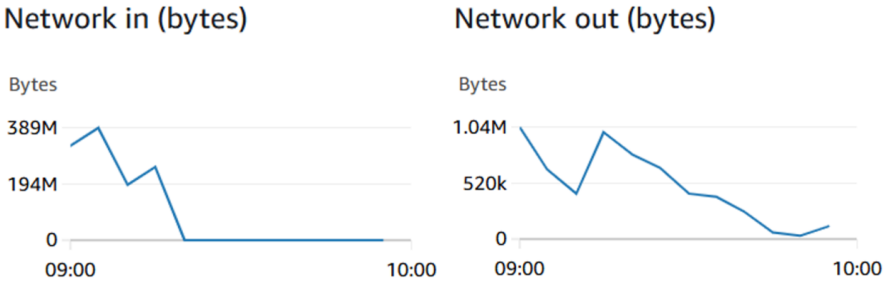
	Utilization charts
CPU usage	
Memory usage	
Network usage	

Utilization charts

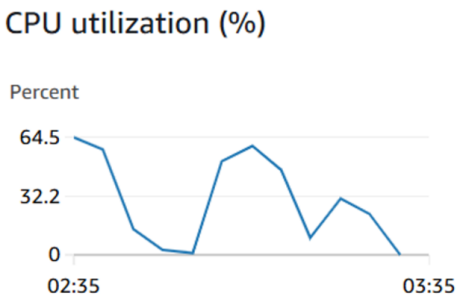
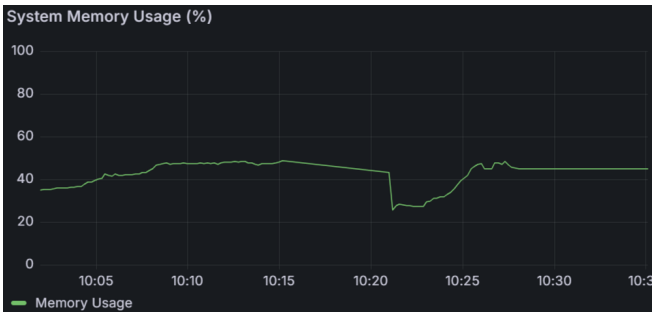
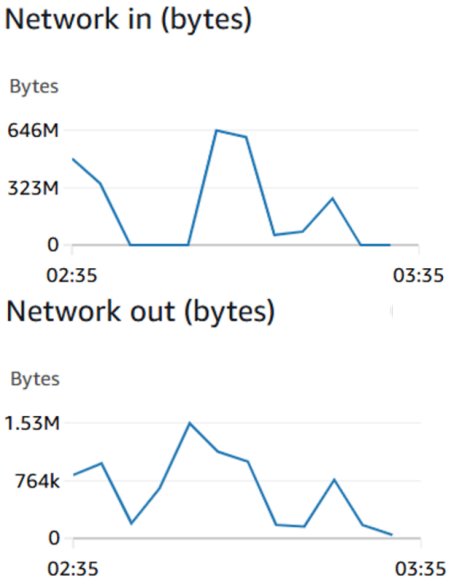
Disk usage

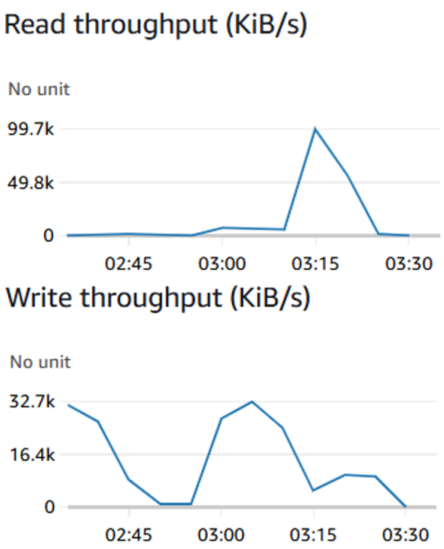


MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP + Vulnerability :

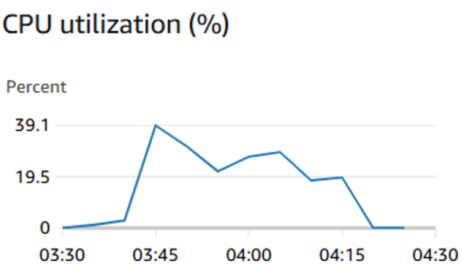
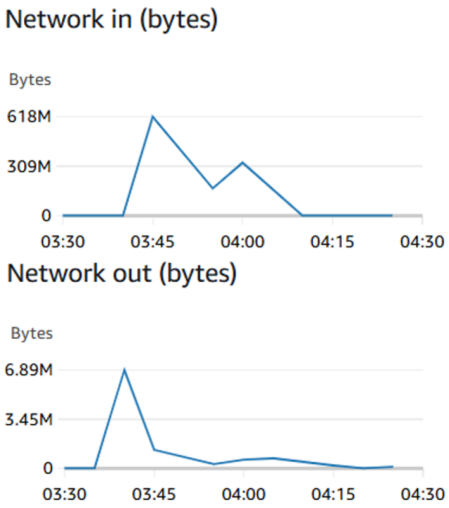
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

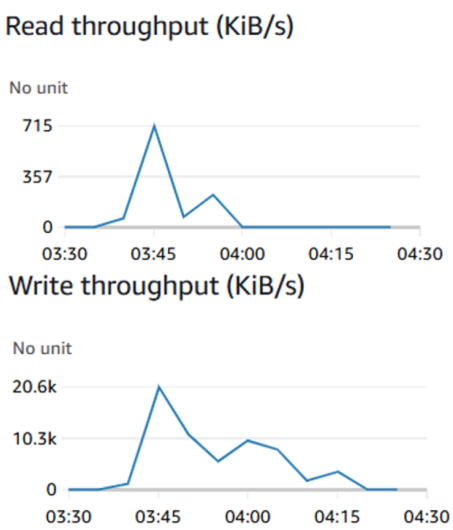
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

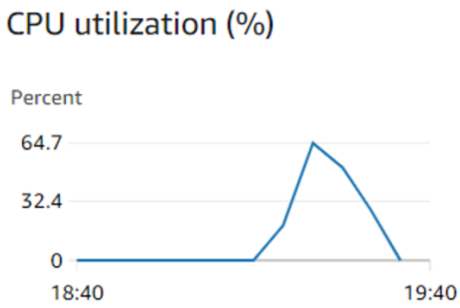
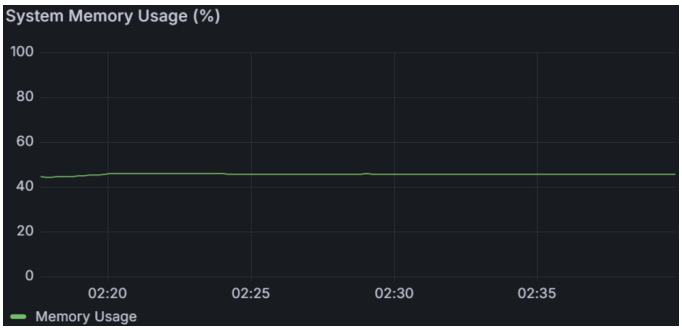
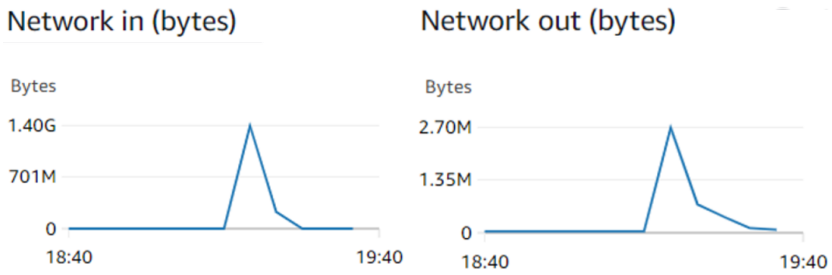
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability :

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

Shared DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

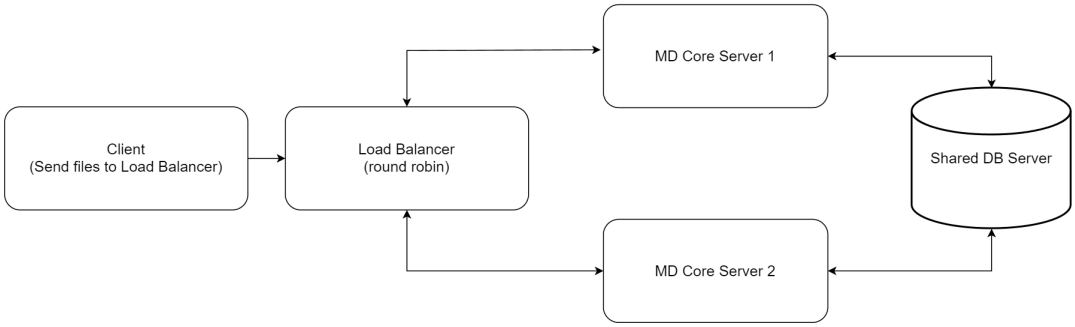
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV)	11.9	37.0	41.9	30.1	30.4	Receive: 6441.1 Send: 769.6	Receive: 6200.2 Send: 693.7	640,821.2	0.0
Deep CDR	10.5	60.1	66.7	30.3	32.2	Receive: 6379.8 Send: 594.8	Receive: 7146.1 Send: 609.6	729,368.5	0.0
Proactive DLP	7.8	38.7	37.5	29.7	31.0	Receive: 8842.6 Send: 791.8	Receive: 8635.6 Send: 729.1	965,455.7	0.0
Vulnerability	4.4	52.2	53.0	26.0	26.9	Receive: 16,215.5 Send: 1006.8	Receive: 15,785.7 Send: 997.2	1714,449.3	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV) + Deep CDR	15.4	72.4	75	34.3	35.4	Receive: 5436.6 Send: 611.3	Receive: 5589.3 Send: 607.2	495,697.4	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP	16.9	77.2	80.2	36.4	37.9	Receive: 5254 Send: 585.4	Receive: 5041.3 Send: 576.1	450,657.6	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	18.0	72.1	80.4	38.6	38.0	Receive: 4504.3 Send: 574.1	Receive: 4649.5 Send: 571.1	423,010	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

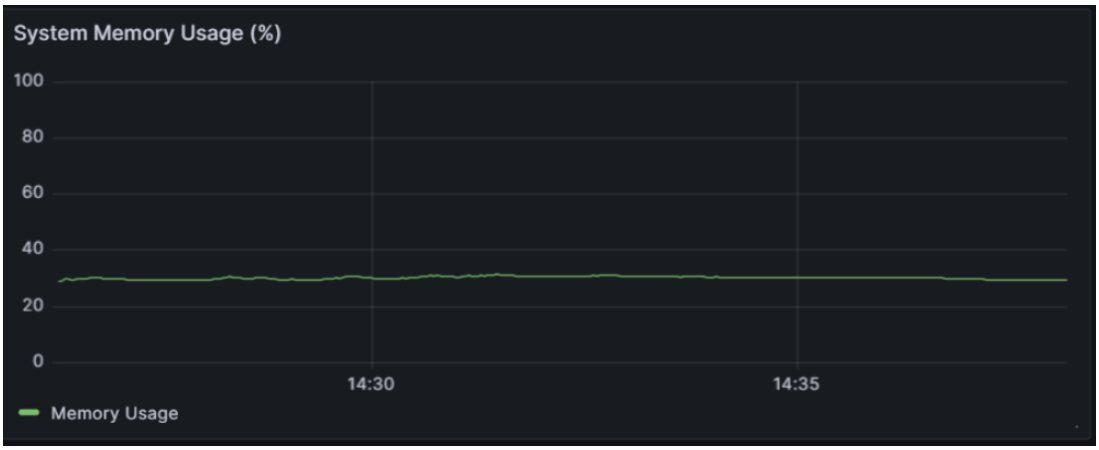
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2

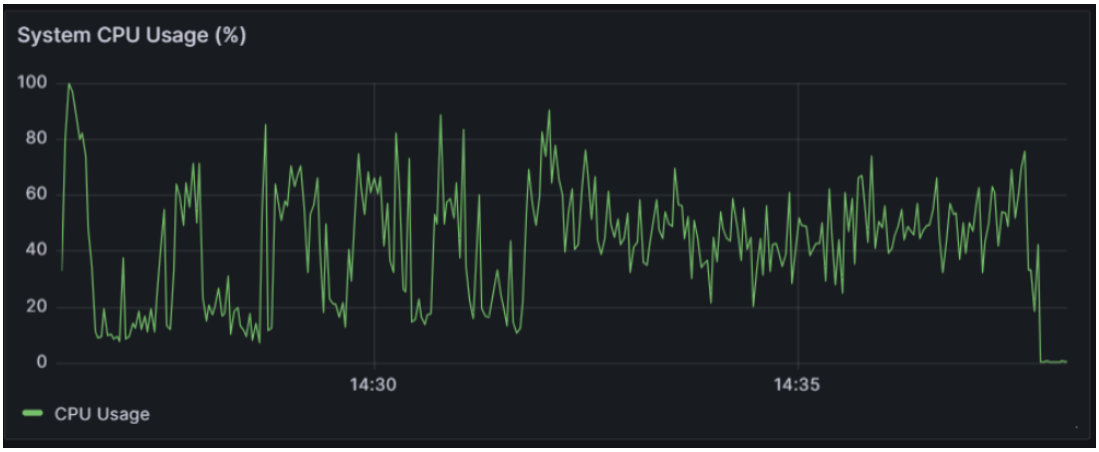


CPU usage

MD Core #1

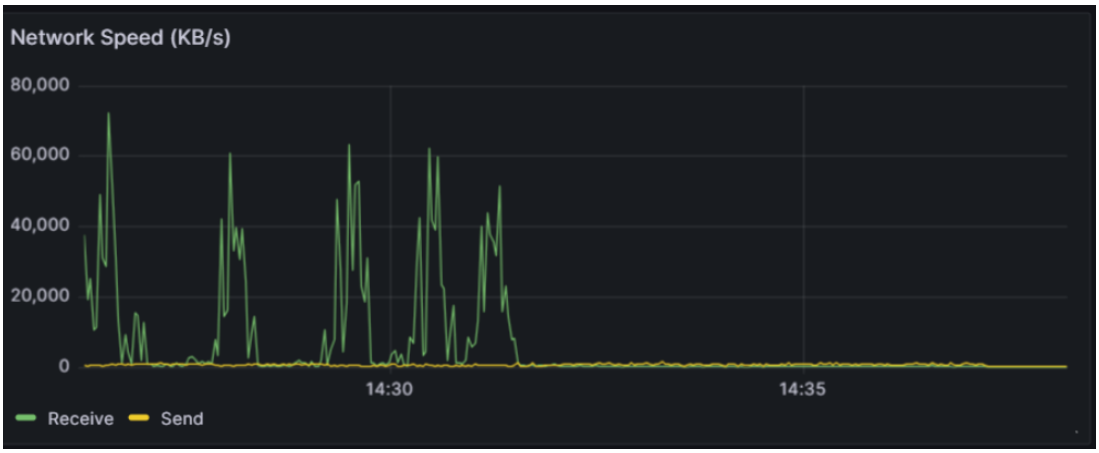


MD Core #2

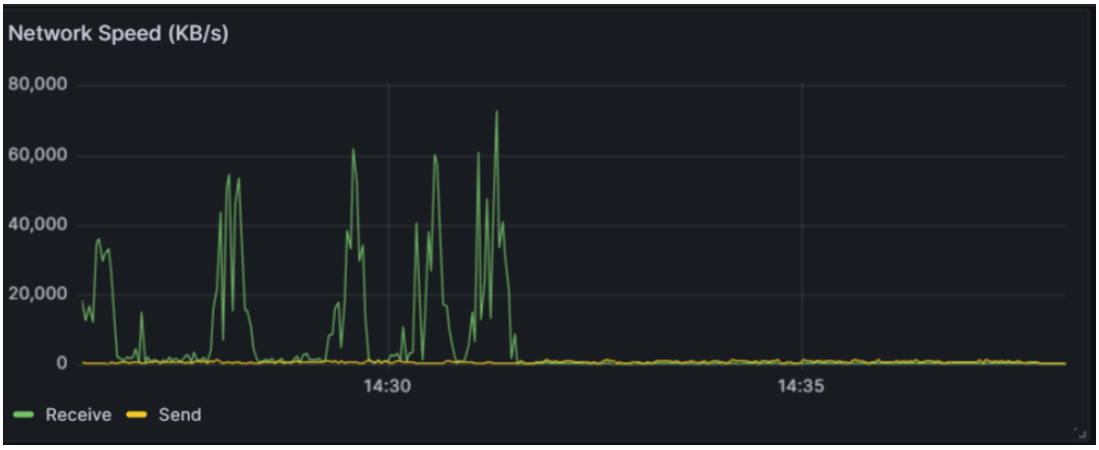


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2

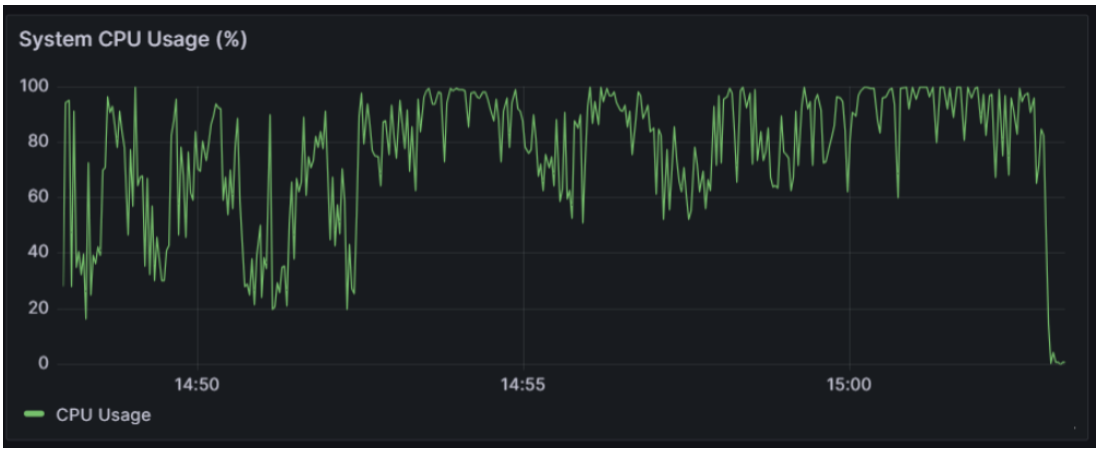


CPU usage

MD Core #1

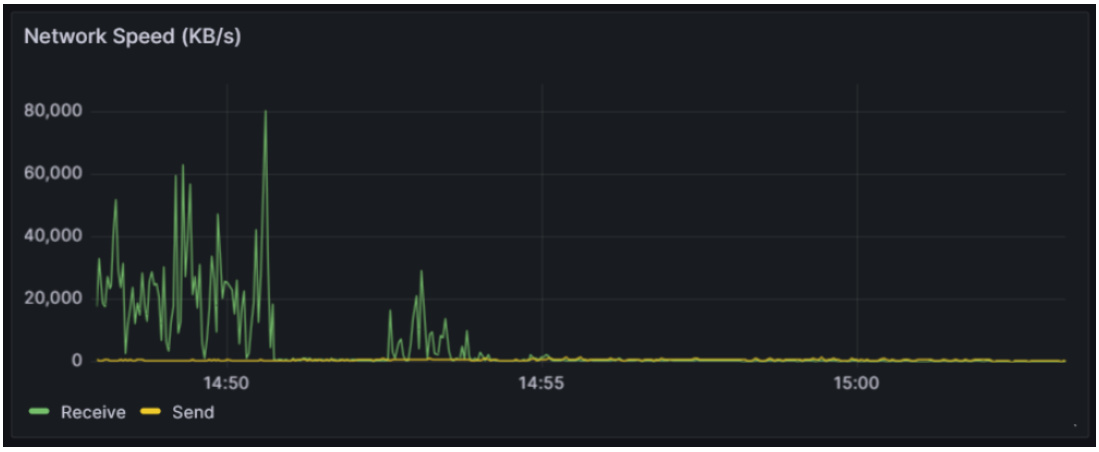


MD Core #2



Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

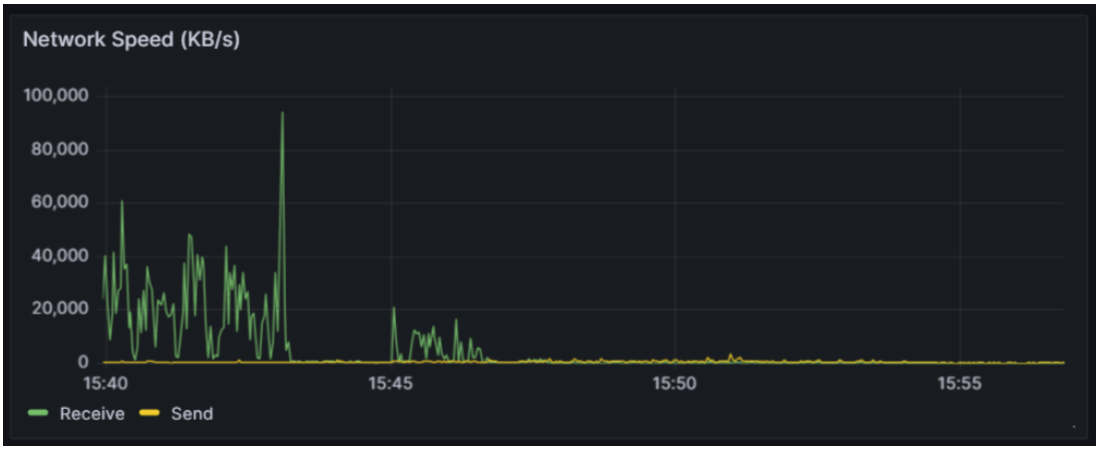


MD Core #2

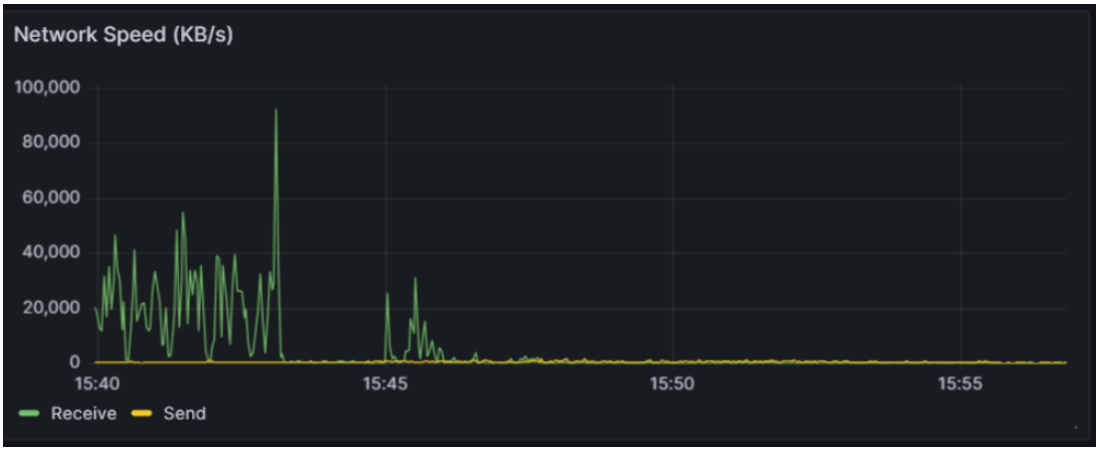


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1

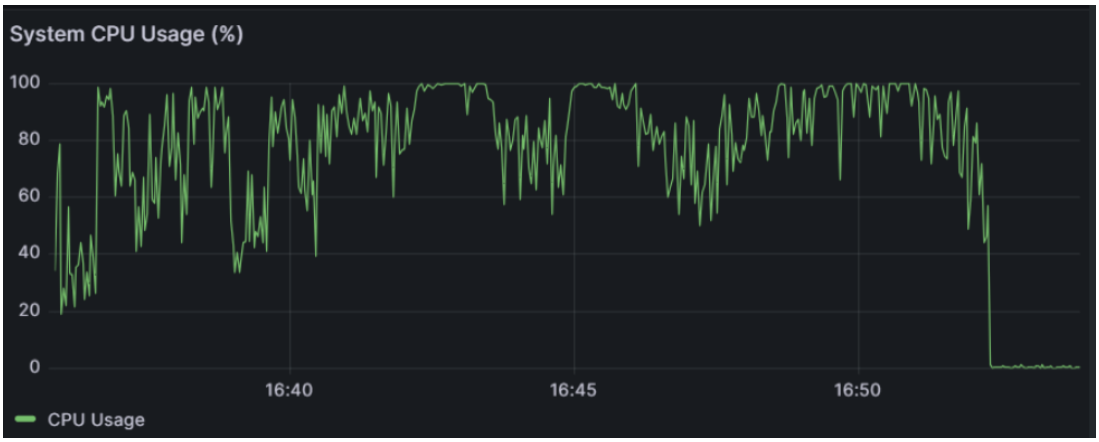


MD Core #2



CPU usage

MD Core #1

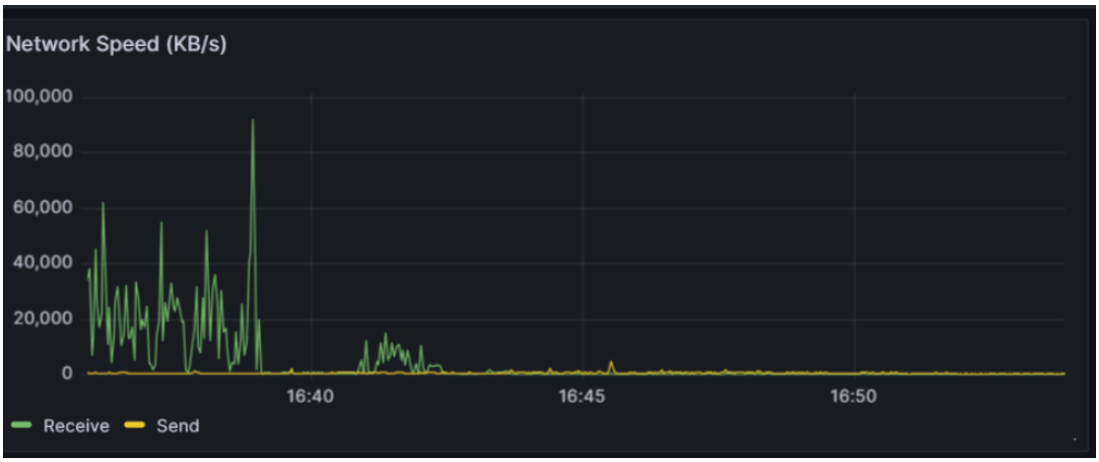


MD Core #2

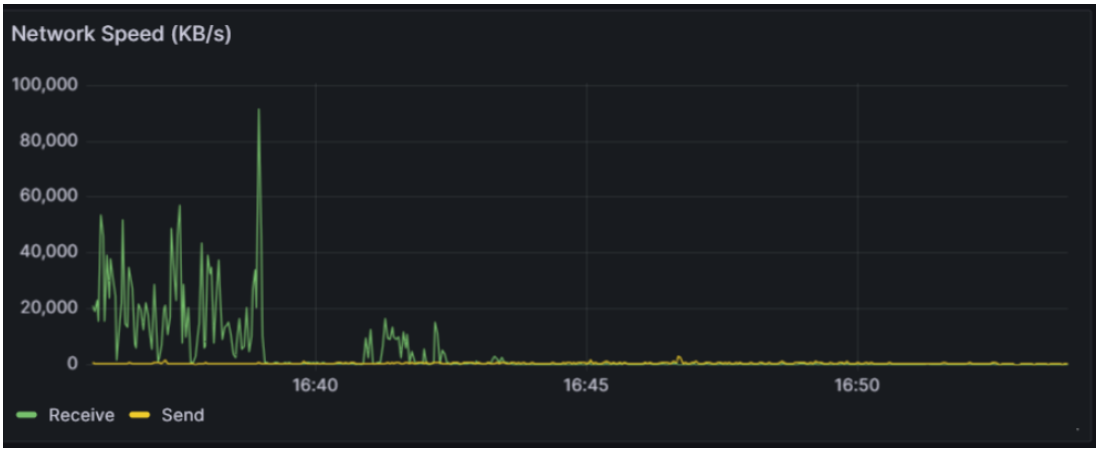


Network usage

MD Core #1



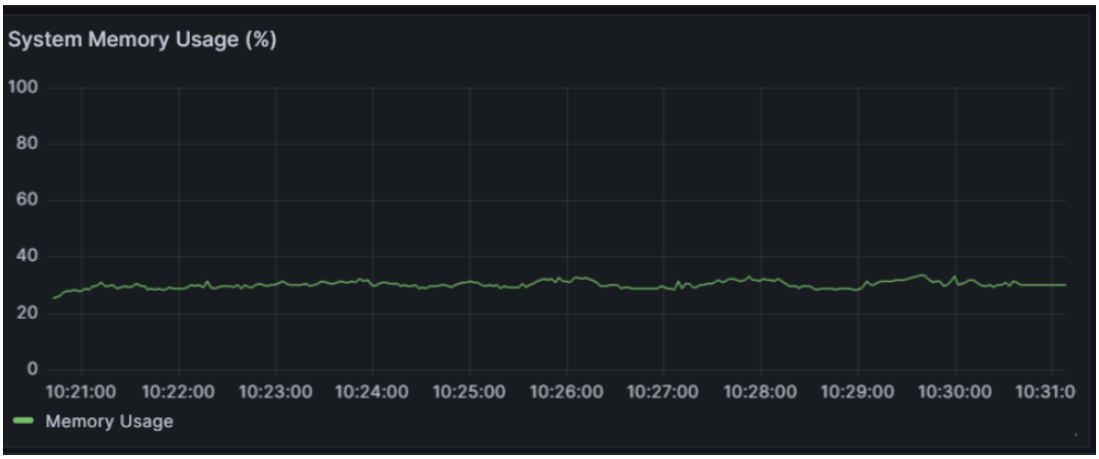
MD Core #2



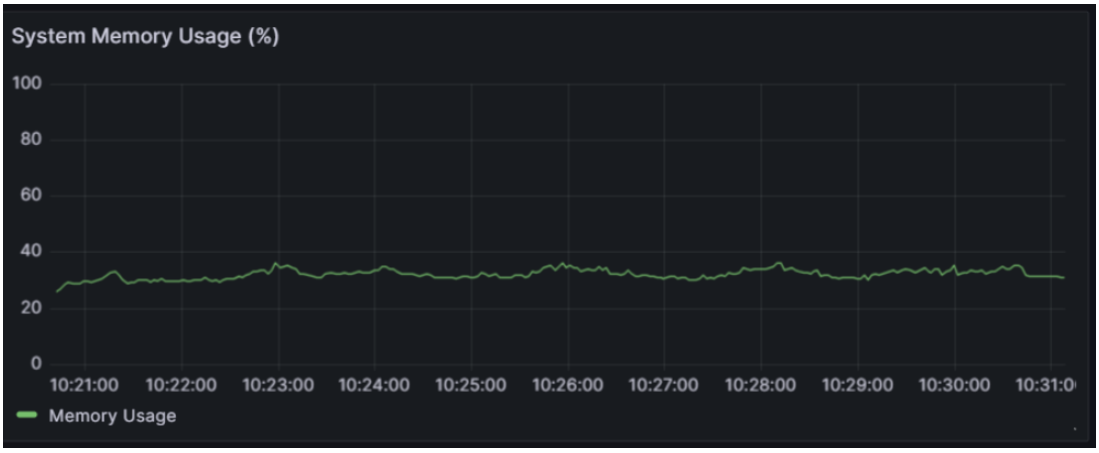
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

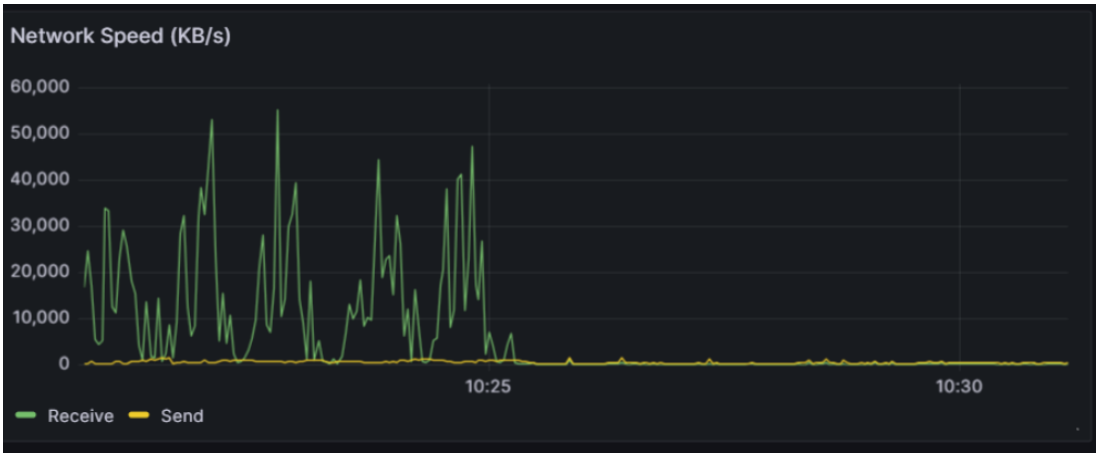


MD Core #2

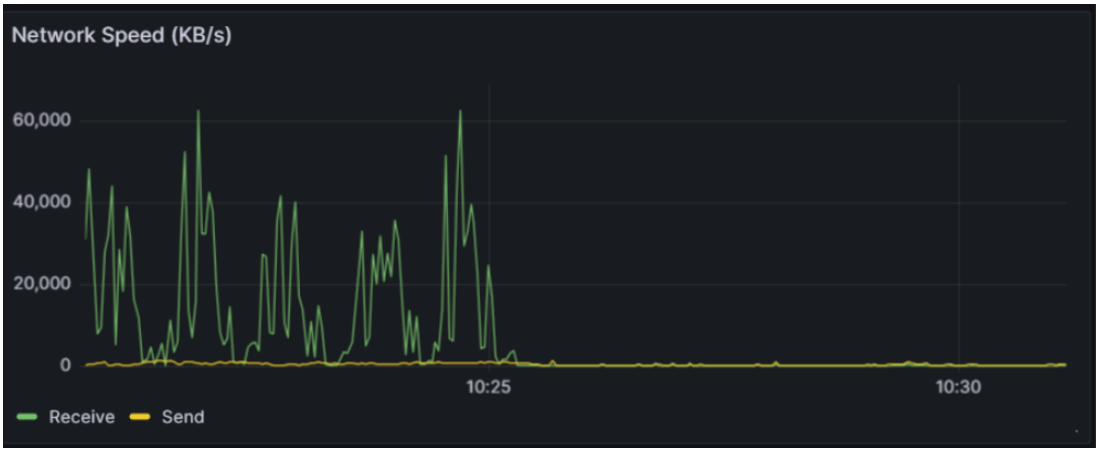


Network usage

MD Core #1



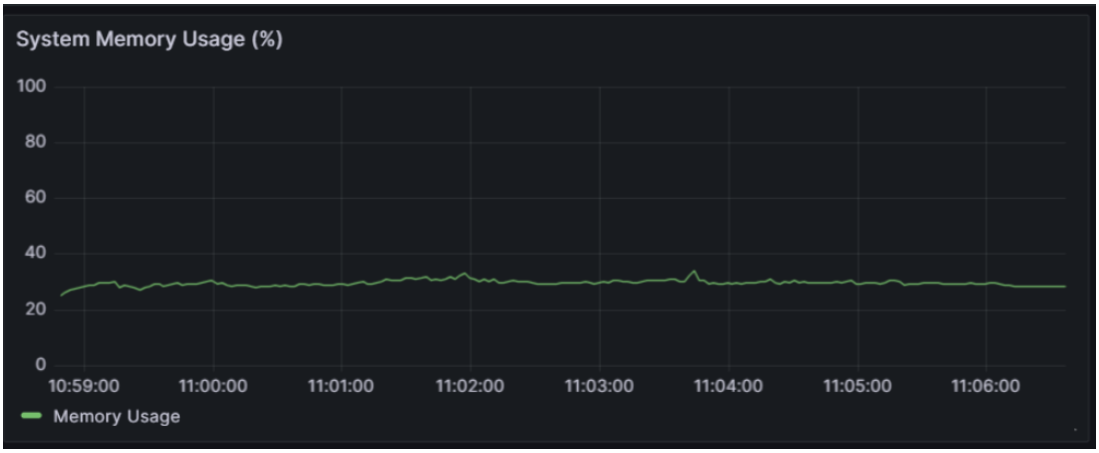
MD Core #2



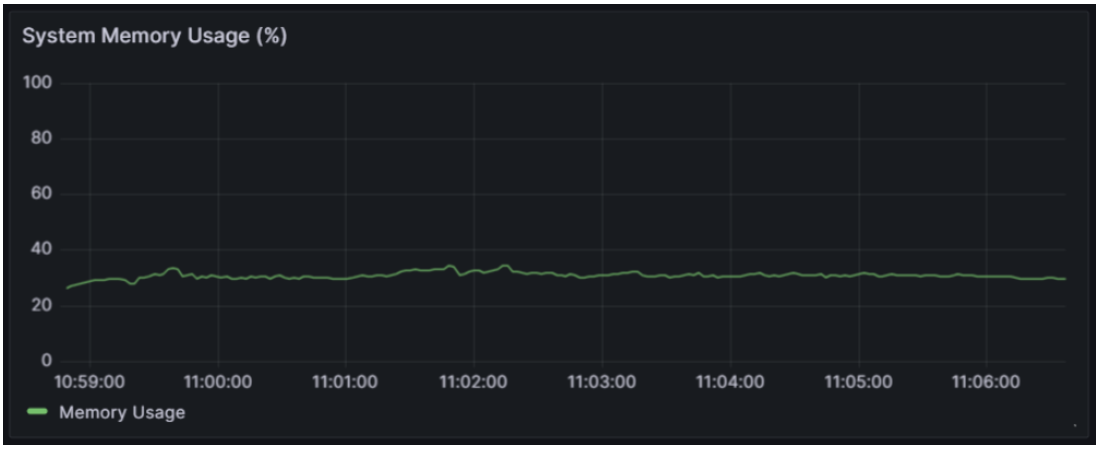
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

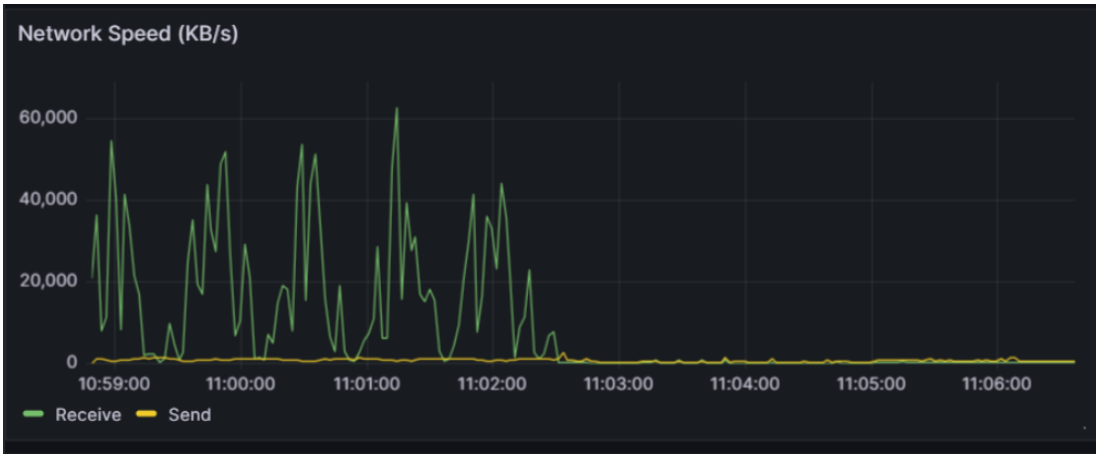


MD Core #2

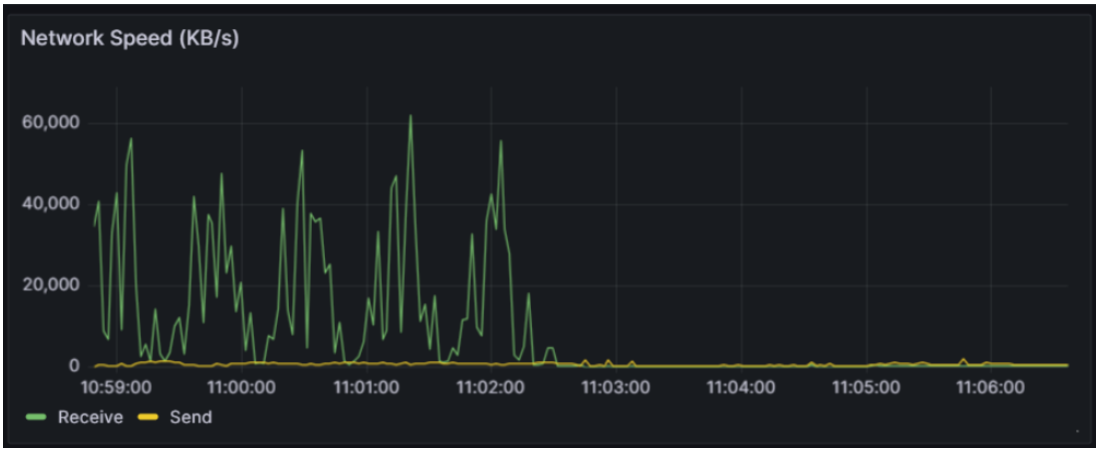


Network usage

MD Core #1



MD Core #2



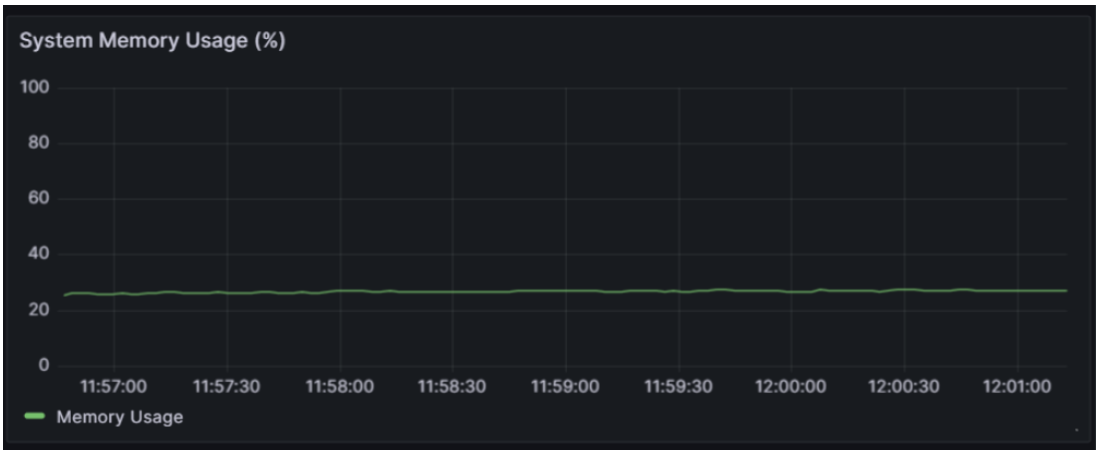
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

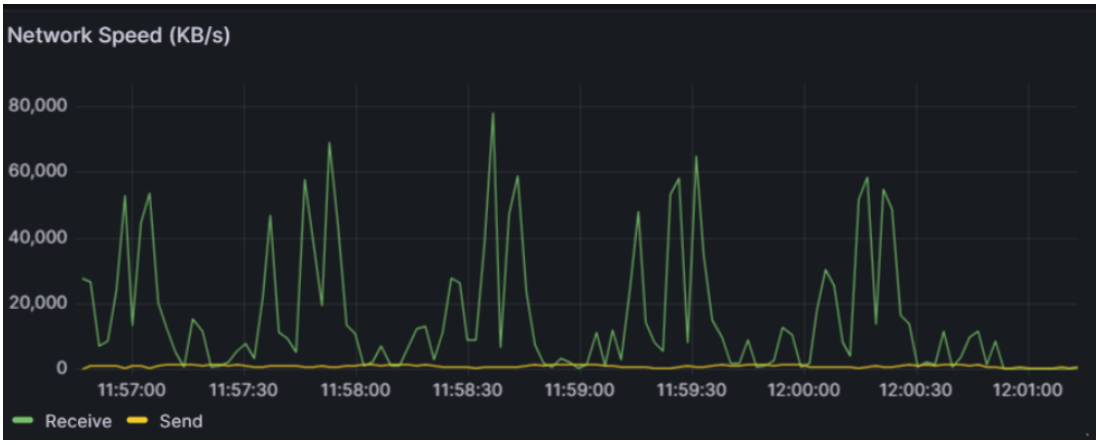


MD Core #2

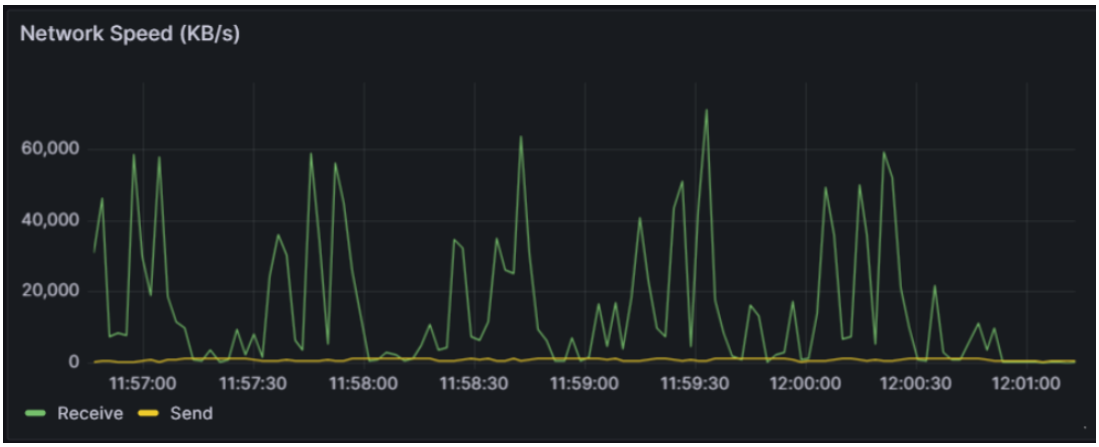


Network usage

MD Core #1



MD Core #2



Shared DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

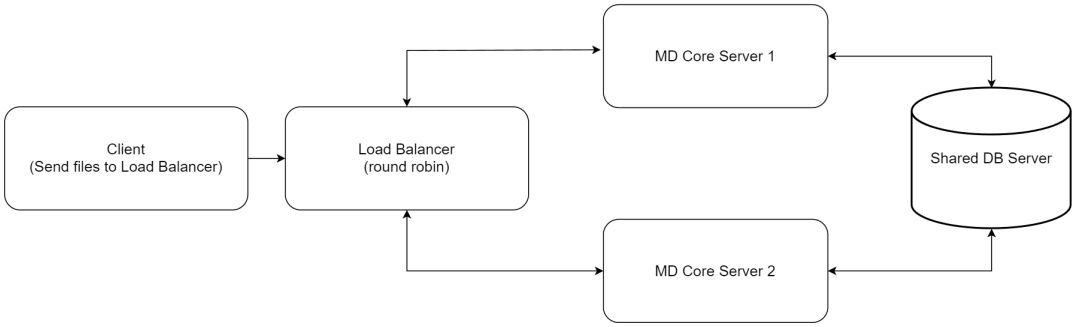
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV)	15.5	63.1	73.3	25.2	25.8	Receive: 4555.9 Send: 694.4	Receive: 4516.2 Send: 692.4	492,620.5	0.0
Deep CDR	9.70	50.7	51.2	31.9	32.5	Receive: 7062.9 Send: 623.2	Receive: 6653 Send: 654.1	786,189.6	0.0
Proactive DLP	7.2	35.2	35.0	27.9	27.5	Receive: 9047.4 Send: 802.4	Receive: 9859.8 Send: 787.5	1,055,110.8	0.0
Vulnerability	5.0	45.7	49.3	24.4	24.5	Receive: 13,585.7 Send: 966.7	Receive: 13,505.8 Send: 987.4	1,519,562.1	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV) + Deep CDR	19.9	79.1	77.6	34.0	34.2	Receive: 3785.9 Send: 573.8	Receive: 3663.1 Send: 577.7	382,716	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP	21.6	80.3	81.5	37.7	44.4	Receive: 3305.6 Send: 543.3	Receive: 3414.3 Send: 556.2	353,694.9	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	22.5	84.2	79.8	37.9	44.7	Receive: 3267.9 Send: 554.9	Receive: 3398.2 Send: 553.8	338,507.6	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

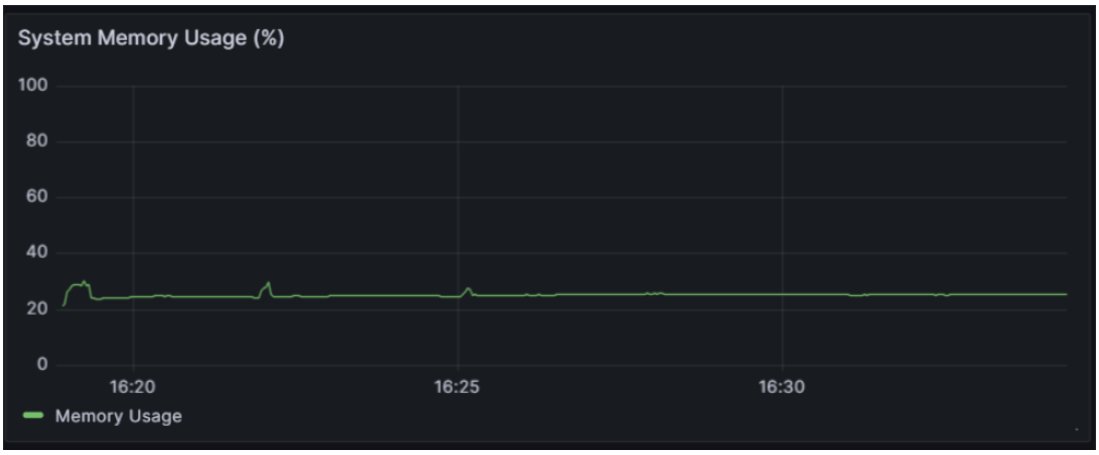
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

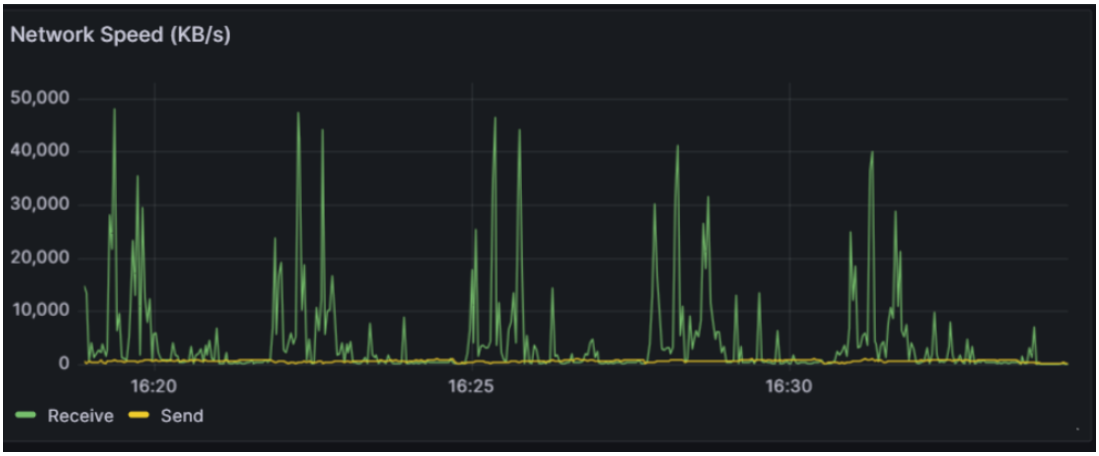


MD Core #2

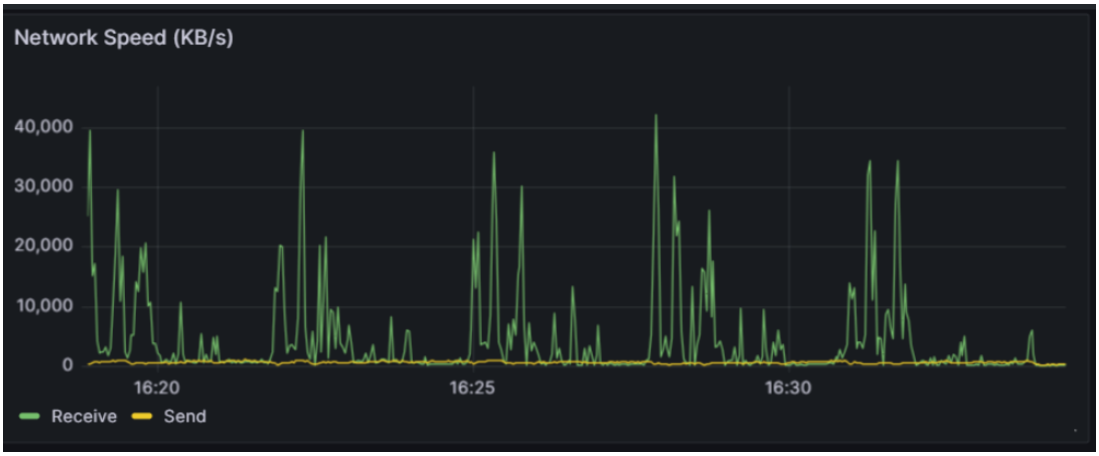


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

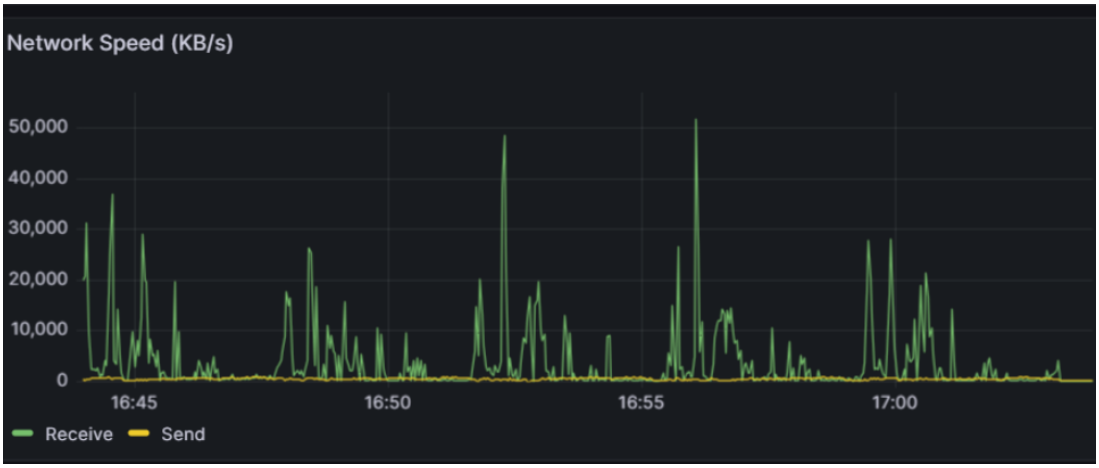


MD Core #2

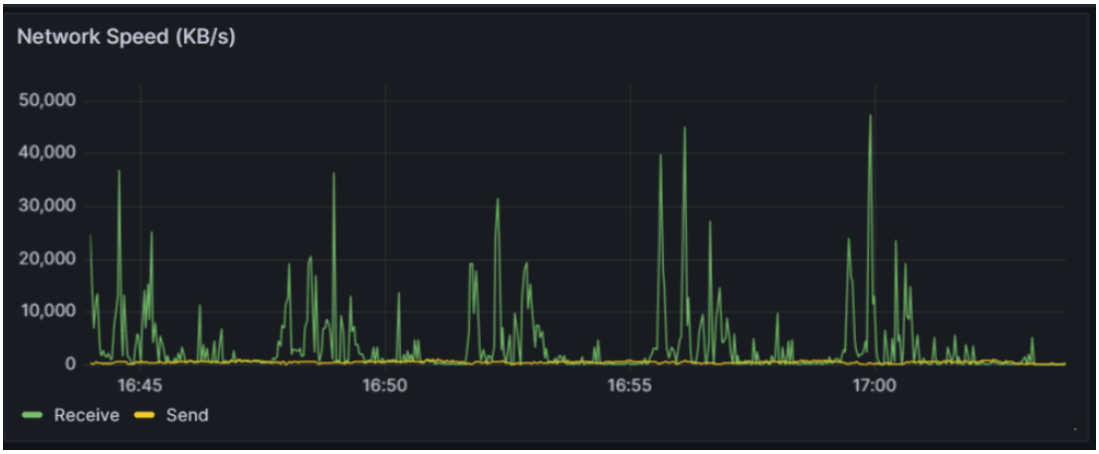


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1

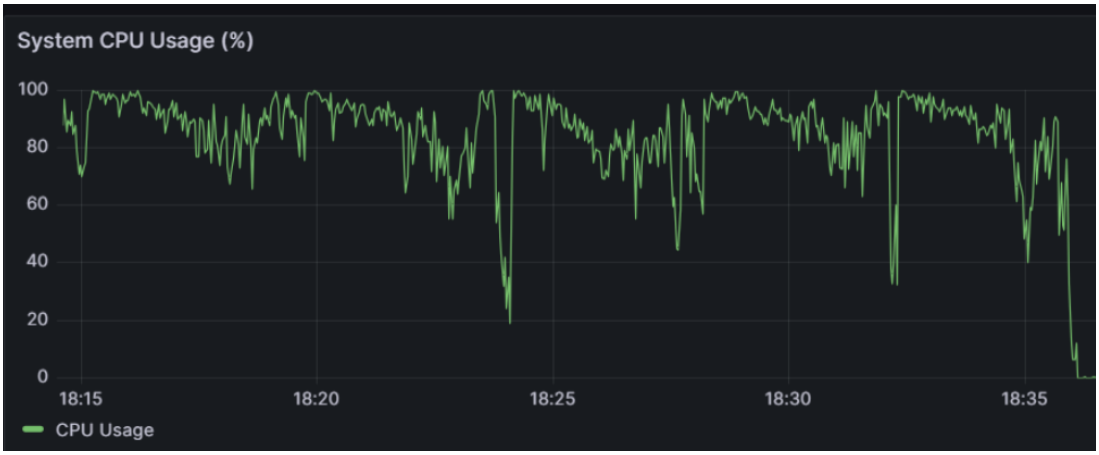


MD Core #2



CPU usage

MD Core #1

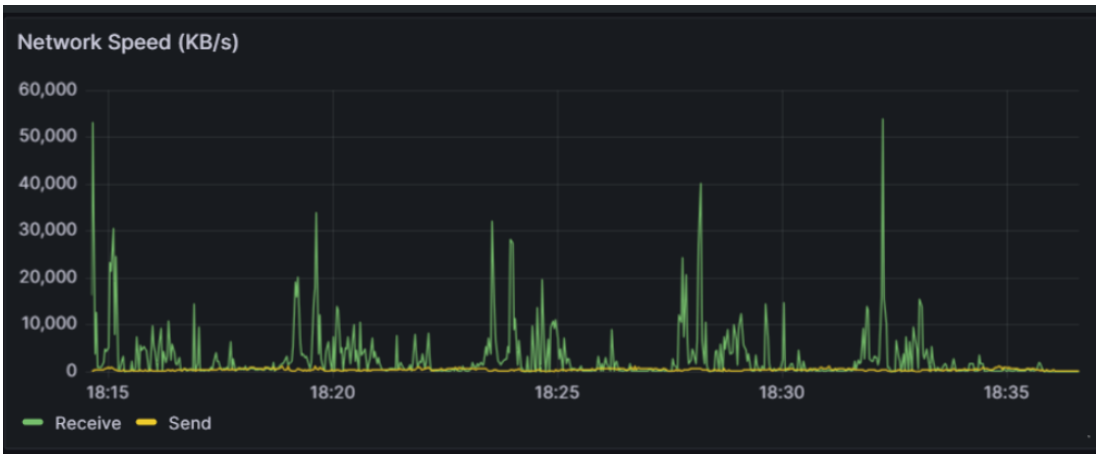


MD Core #2

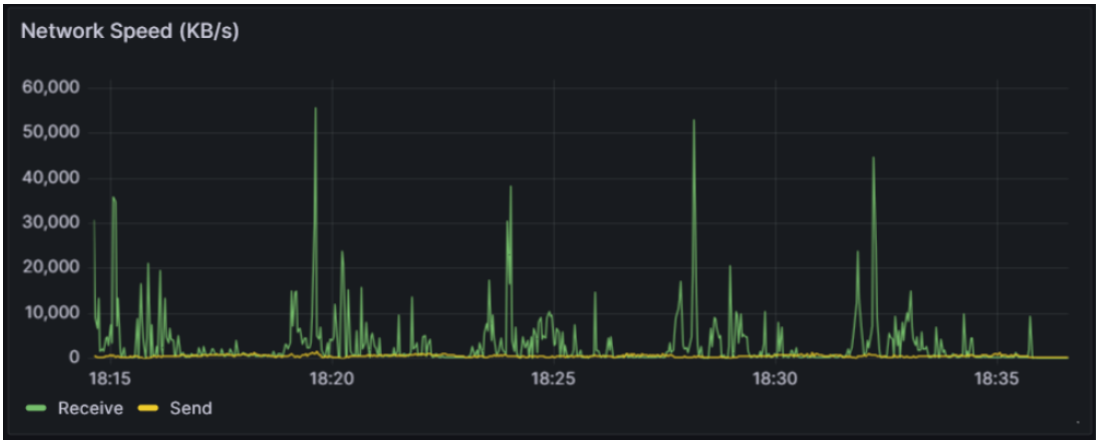


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

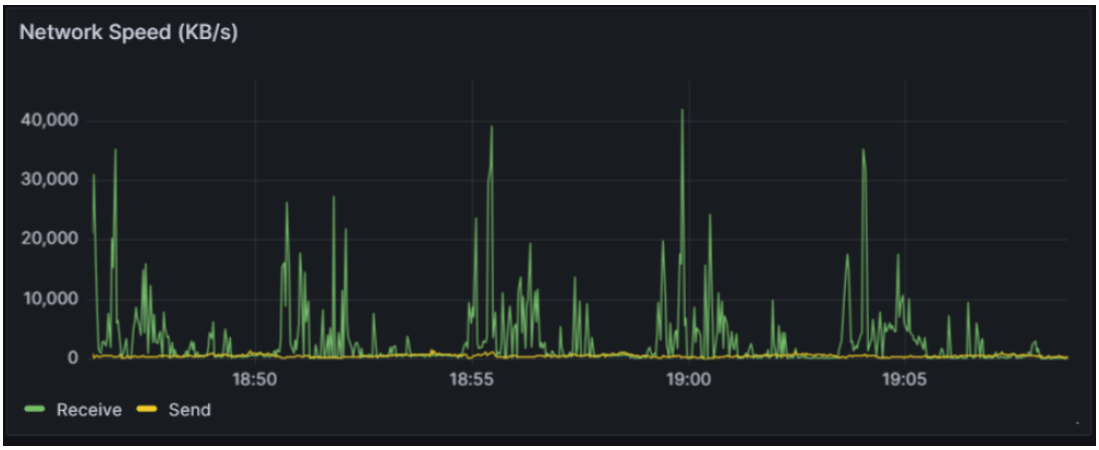


MD Core #2

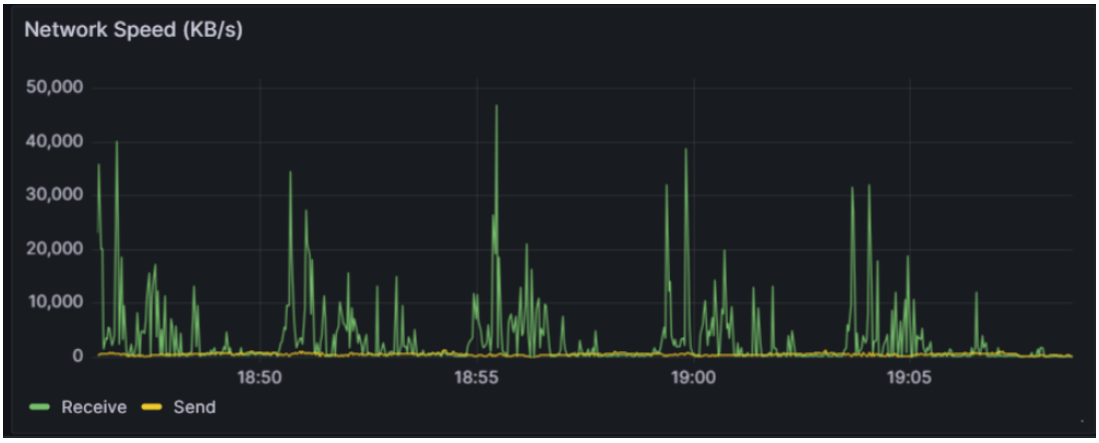


Network usage

MD Core #1



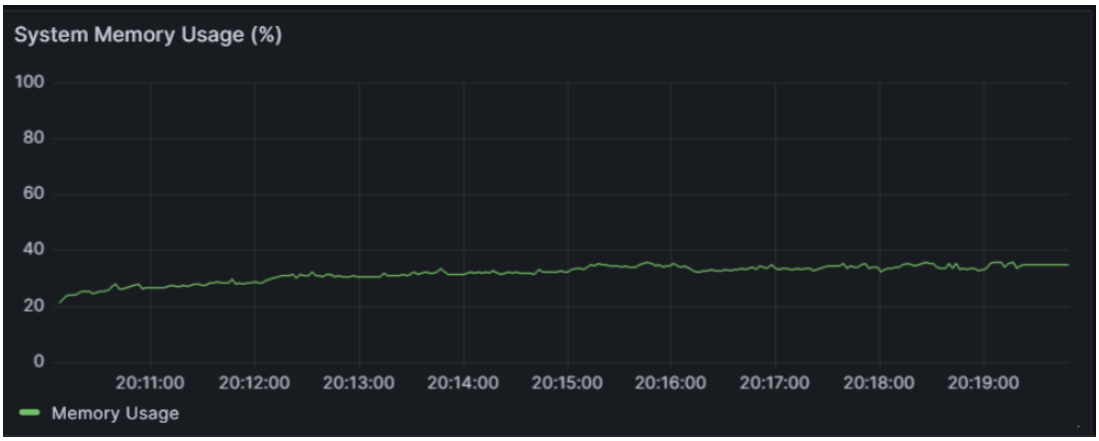
MD Core #2



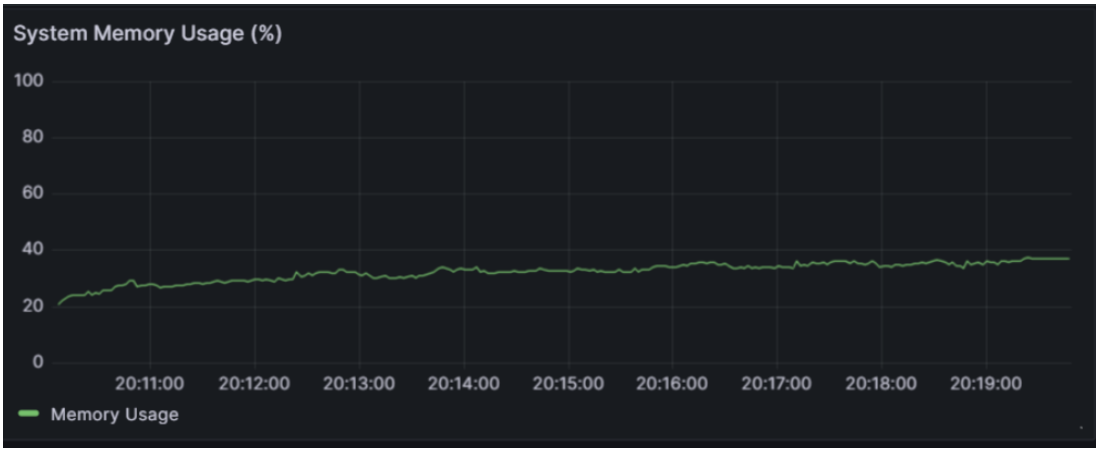
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1

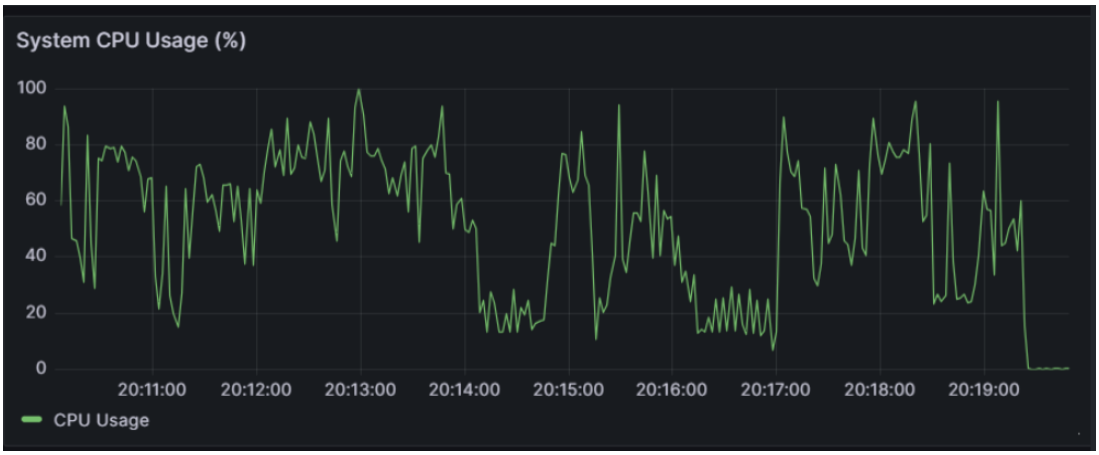


MD Core #2



CPU usage

MD Core #1

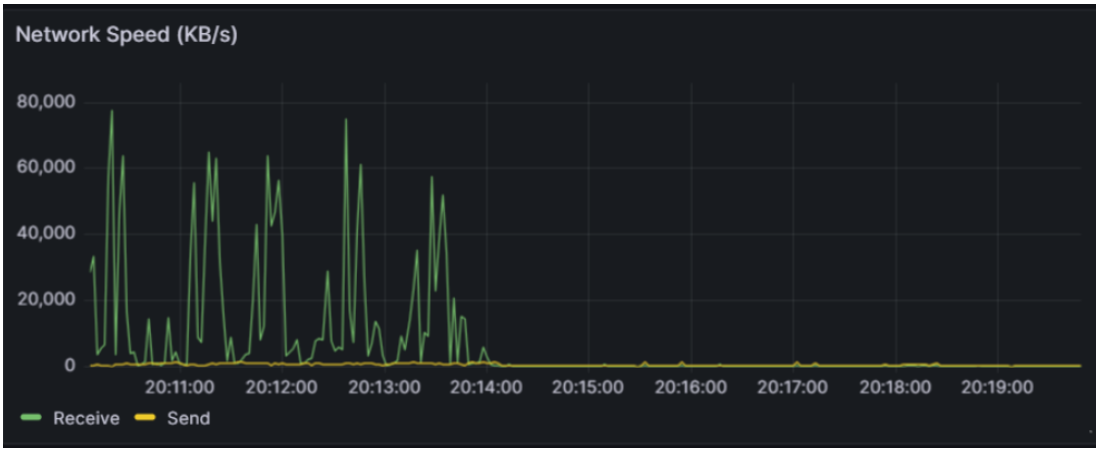


MD Core #2

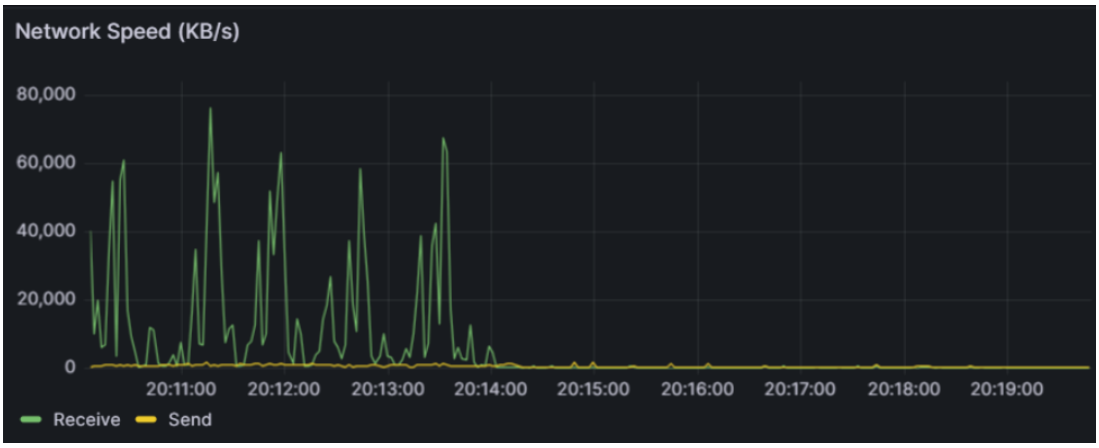


Network usage

MD Core #1



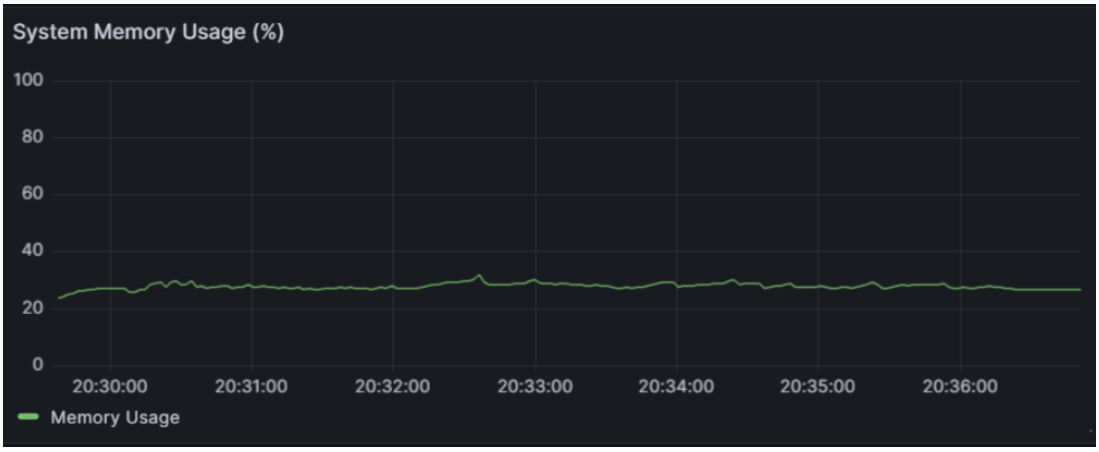
MD Core #2



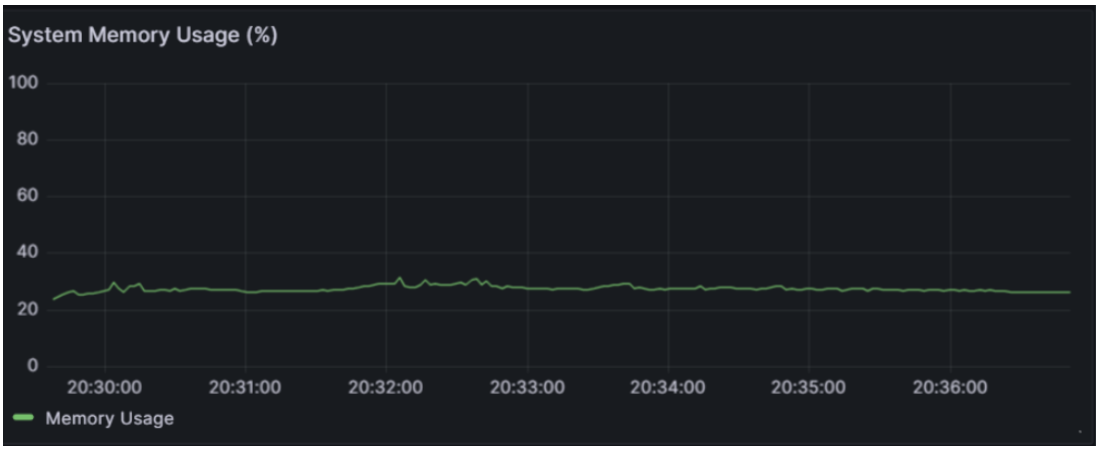
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1

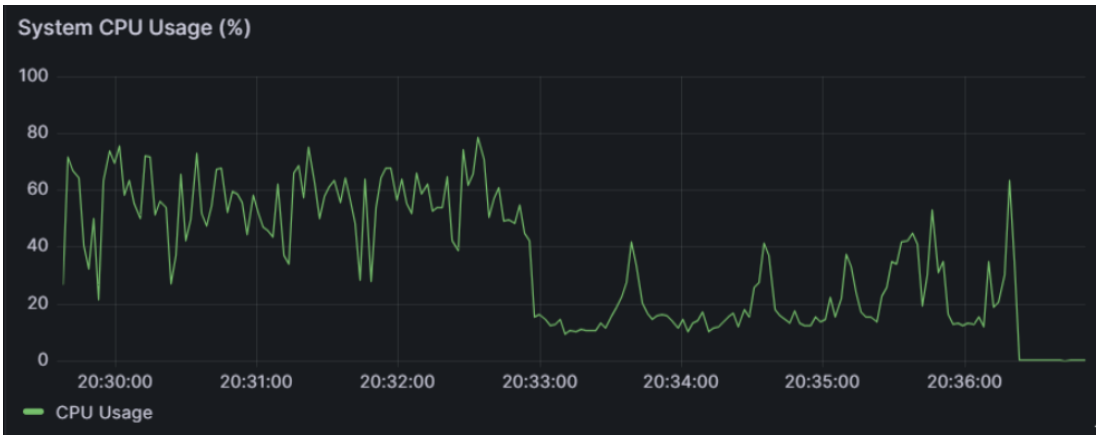


MD Core #2

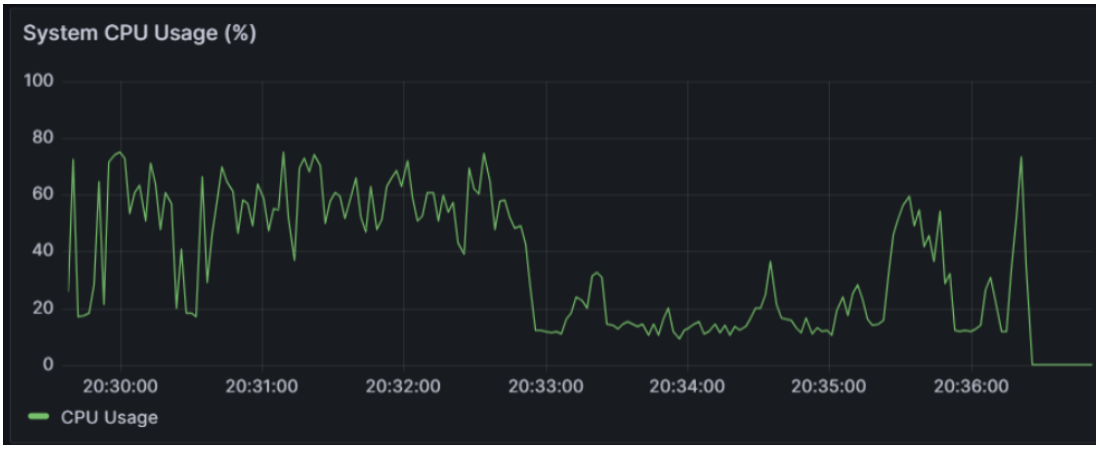


CPU usage

MD Core #1

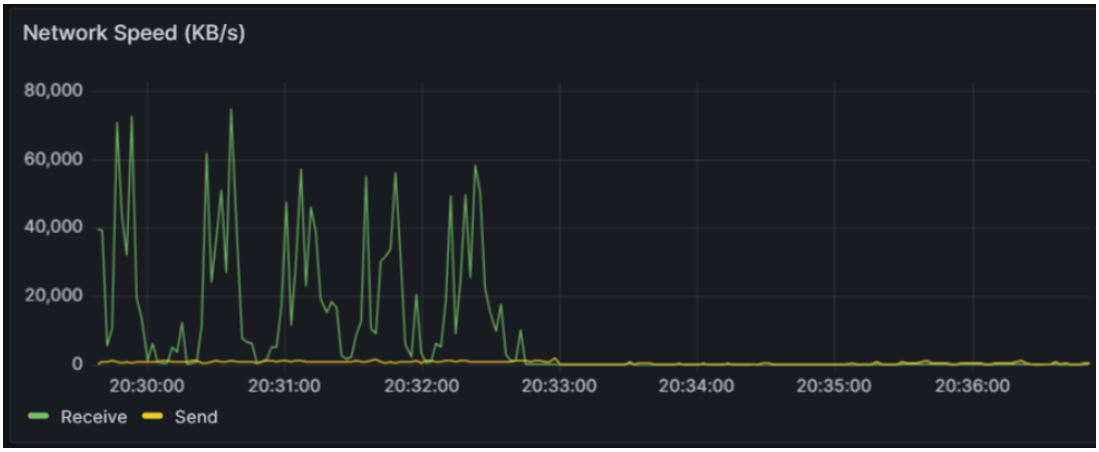


MD Core #2

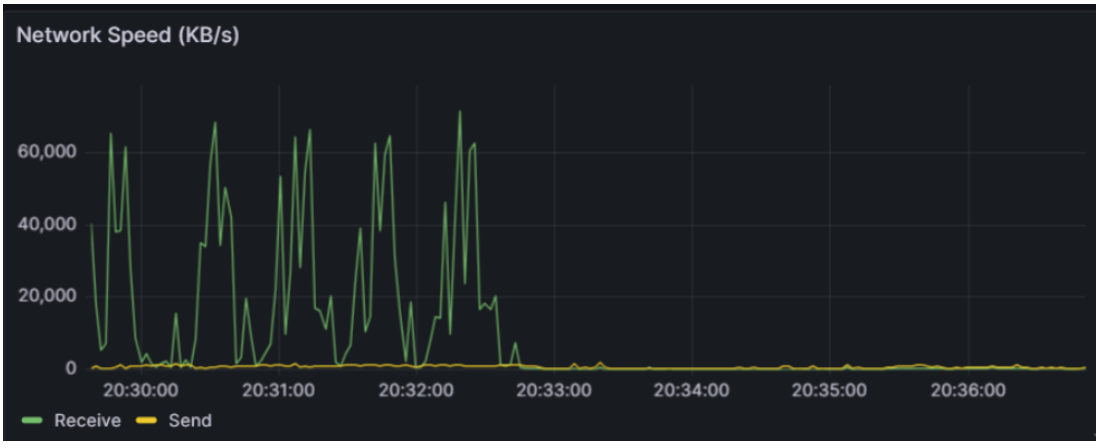


Network usage

MD Core #1



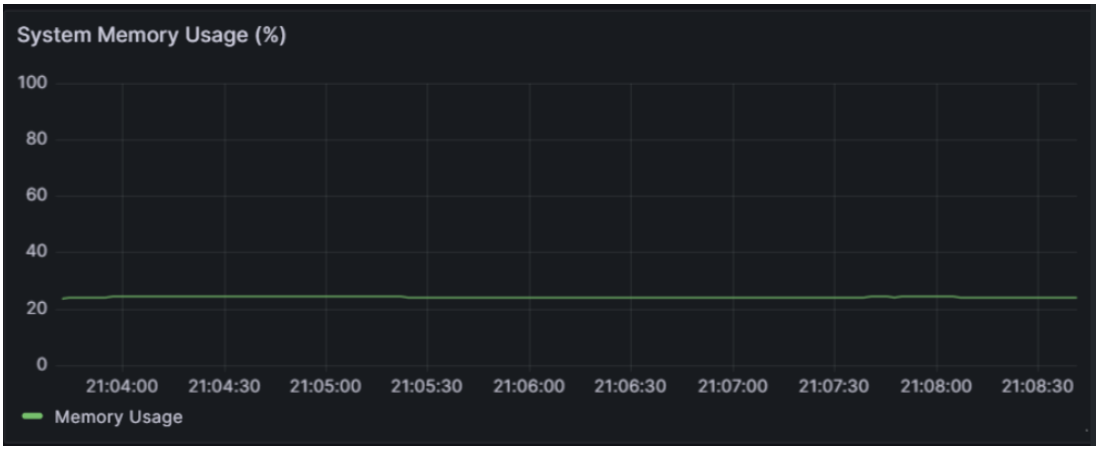
MD Core #2



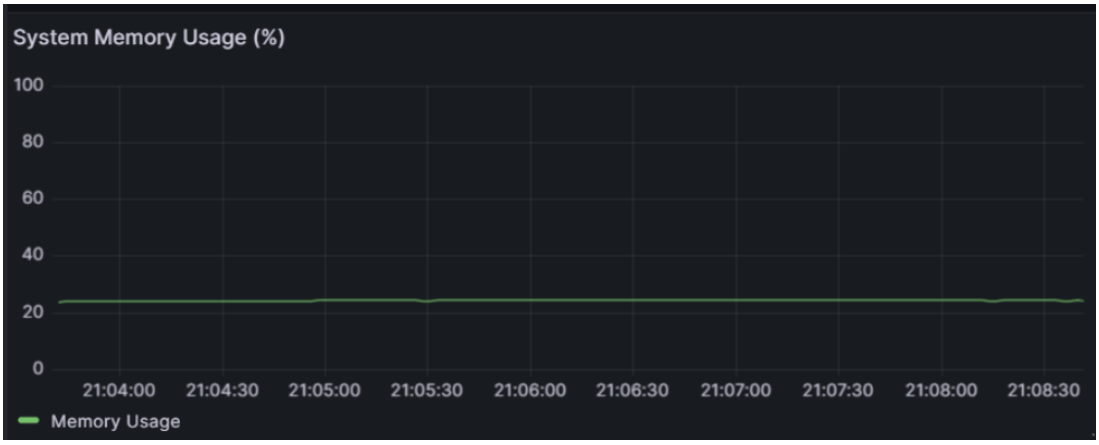
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

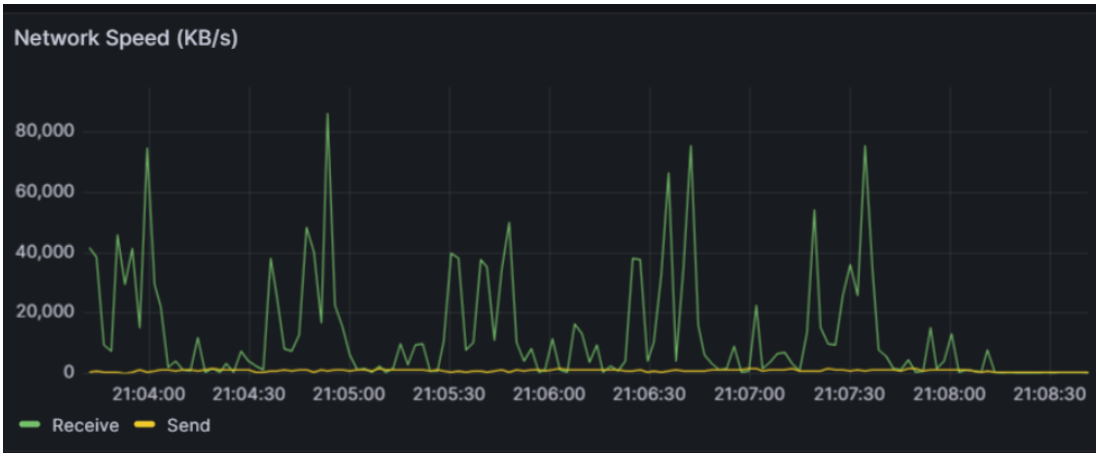


MD Core #2

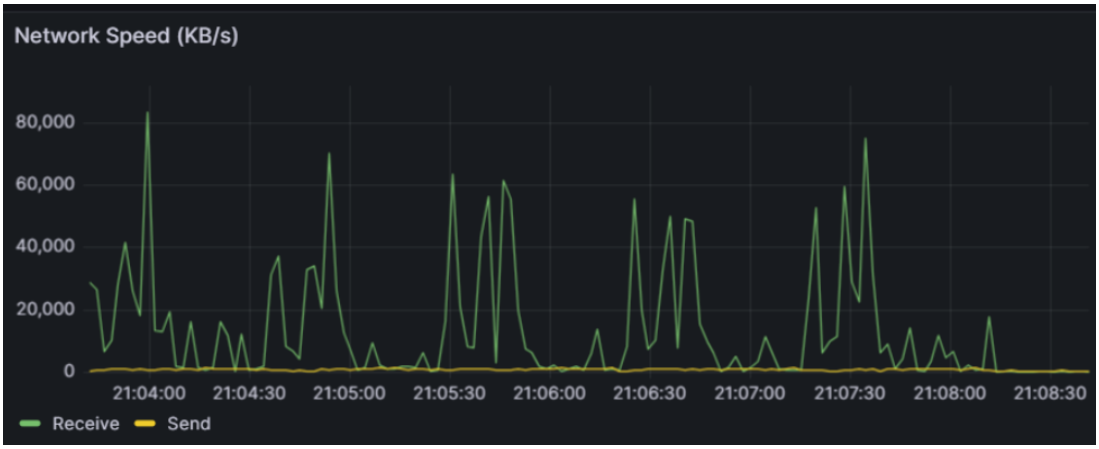


Network usage

MD Core #1



MD Core #2

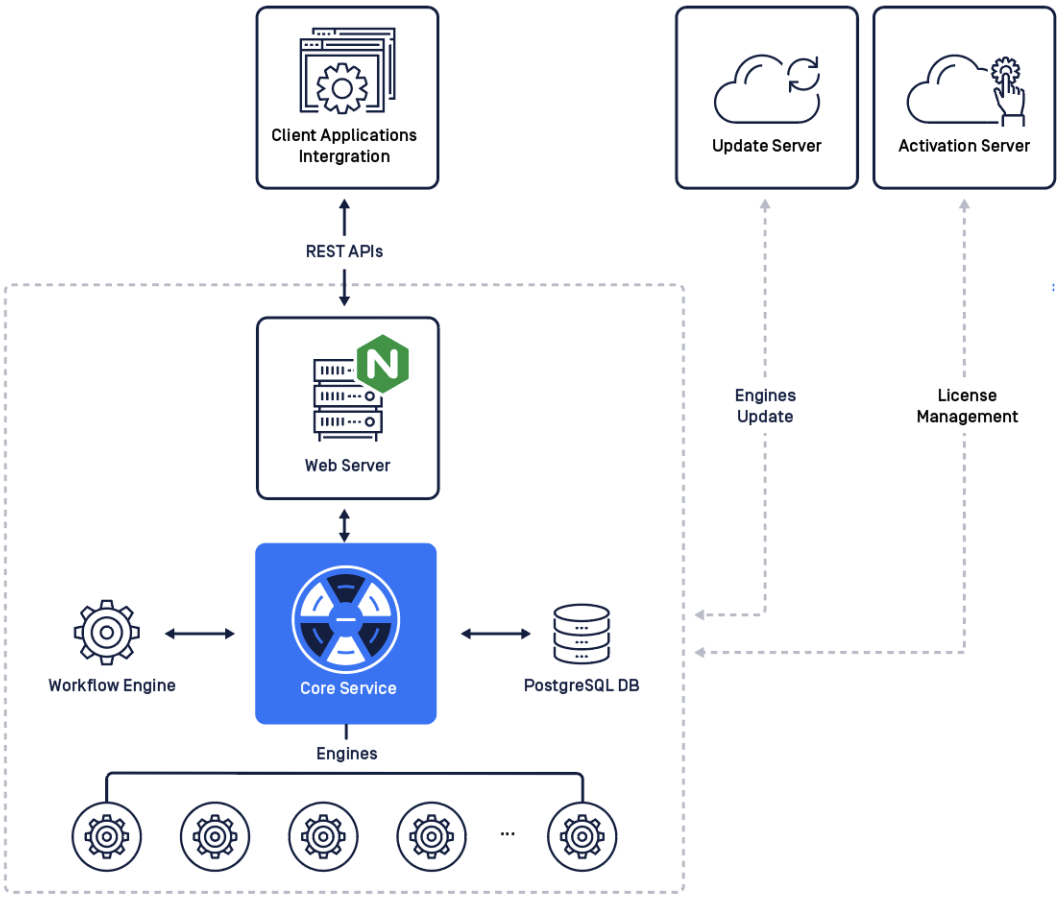

Network Speed (KB/s)
80,000
60,000
40,000
20,000
0
21:04:00
21:04:30
21:05:00
21:05:30
21:06:00
21:06:30
21:07:00
21:07:30
21:08:00
21:08:30
Receive
Send

MetaDefender Core Documentation

The users can consult this web page or, alternatively, they can download the manual in pdf format from the link below:

MetaDefender Core manual (SHA256:).

Recommended System Configuration



MetaDefender Core supports a wide range of deployment options, including air-gapped environments, providing the flexibility to choose the environment that best suits customers' requirements and infrastructure.

- Physical server.
- Virtualization platforms like VMware, Hyper-V and XenServer.
- Cloud Deployment.
- Container Deployment.

> **Note**
> Before installing MetaDefender Core, please refer to the **recommended minimum system configuration** listed below. The requirements are based on a standardized customer deployment and may require more or less resources to achieve the level of performance desired in your environment. Please note that the server specifications are built to allow a high volume daily processing. Please refer to Performance and Load Estimation for more details.

The recommendations below are for MetaDefender Core, API usage only. For any other use cases, please consult the user guide of the licensed products for accurate recommendations.

For certain use cases these might be adjusted and customized on specific needs and SLAs. We highly recommend to engage our ProServ team to assist in fine tuning MetaDefender and get the maximum performance out of your systems.

Microsoft Windows Deployments

[Windows] Supported Operating Systems

Only support 64-bit platforms.

- Windows 10, Windows 11 23H2, 11 IoT (22H2) only for End-User systems (e.g. Kiosk deployments)
  - Based on Microsoft License Terms, you're not allowed to use Windows Desktop as a server, therefore, MetaDefender Core can't be used on these systems for server use cases.
- Windows Server 2016, 2019, 2022

> **Disclaimer**
> End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

> **Plan for legacy OS support**
> OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:
>
> - Ubuntu 20.04 after **April 2025**.
> - Amazon Linux 2 after **June 2025**.
> - Windows 10 after **October 2025**.
>
> If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

[Windows] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 8	8	8 GB	16 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 12	16	16 GB	24 GB + 50 GB + 100 GB
MetaDefender Core 16	16	16 GB	32 GB + 50 GB + 100 GB
MetaDefender Core 20	32	16 GB	40 GB + 50 GB + 100 GB
MetaDefender Core MAX	32	32 GB	120 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core is installed.
- For a better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.
- SBOM processing requires more disk space, please allow at least 15GB free disk for the engine to work properly.

[Windows] Third Party Dependencies

- .NET framework 4.5 or above
- Microsoft Visual C++ Redistributable for Visual Studio 2015-2022 (version 14.38 or higher); x64 version.
- PostgreSQL: Only applicable when using a pre-installed PostgreSQL server running remotely.
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Some engines also have dependencies as described below:

Engine Name	Dependency
Deep CDR	Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 5.8.0 or above)
ESET	MetaDefender Core v4 temporary directory should have more than 200MB free disk space
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
Proactive DLP	• Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 2.0 or above) • .NET 6 dependencies • CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
Scrutiny	Microsoft Visual C++ 2015 - 2019 Redistributable x86 and x64
Systweak	.NET runtime 6.0, 7.0 or 8.0
Bkav Pro	Microsoft Visual C++ 2015 - 2022 Redistributable x64

[Windows] Installation Details

MetaDefender Core on Windows uses "C:\Program Files\OPSWAT" folder for storing resources or the installation directory.

MetaDefender will use its resources folder to store temp files as part of the analysis. **It's recommended to exclude this folder from real-time protection monitoring**. Knowledge base:

- How to set up exclusions on anti-malware software to prevent disruption
- Mitigating scan failure triggered by real-time protection service
- Can local AVs interrupt ongoing scans?

Linux Deployments

[Linux] Supported Operating Systems

Only support 64-bit platforms.

- Red Hat Enterprise Linux 8, 9
- Rocky Linux 9
- Debian 11, 12
- Ubuntu 20.04, 22.04
- Amazon Linux 2 is supported via Amazon Machine Image (AMI)

Plan for legacy OS support

OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:

- Ubuntu 20.04 after **April 2025**.
- Amazon Linux 2 after **June 2025**.
- Windows 10 after **October 2025**.

If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

For Hardended Linux OS

Some AV engines could be failed to initialize and run on MetaDefender Core (e.g. ESET, Kaspersky) due to execution permission required by them on `/var/tmp/ometascan` folder.

Please follow steps at Why have the ESET and Kaspersky scan engines failed to initialize on the hardened Linux OS? for remediation details.

Disclaimer

End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

[Linux] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 5	4	4 GB	10 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 10	8	8 GB	20 GB + 50 GB + 100 GB
MetaDefender Core MAX	16	16 GB	40 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core resides on.
- For better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.

[Linux] Third Party Dependencies

Dependencies list:

- openssl
- grep
- lib32stdc++6 (>= 4.5)
- libc6-i386 (>= 2.10)
- procps
- zlib1g
- libcurl3 (>= 7.19.7)
- libcurl4
- postgresql:
  - From 14.5 to 14.15
  - From 15.7
  - From 16.4

Not all the above dependencies will need to be installed, it is dependent on different Unix distro and version

Some engines also have dependencies as described below:

Engine name	Dependency
Archive engine	• To extract password protected MS Office files, you need to install .NET 6 dependencies, libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • Rocky Linux 9.x/Red Hat 9.x: libnsl
Deep CDR	• Rocky Linux 9.x: libgomp, ncurses-compat-libs, libnsl • CentOS: libgomp, ncurses-compat-libs • Red Hat 8.x: libgomp, ncurses-compat-libs, libnsl • Red Hat 9.x: libgomp, ncurses-compat-libs, libnsl, compat-openssl11 • Ubuntu/Debian: libgomp1, libncurses5 • libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • .NET 6 dependencies
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
FileType engine	Rocky Linux 9.x/Red Hat 9.x: libnsl
Proactive DLP	libgdiplus, mscorefonts .NET 6 dependencies CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
RocketCyber	libgomp

Note: Some dependencies may need to be installed from external repositories, not from the OS default repositories.

[Linux] Installation Details

MetaDefender Core default installation path is using /var folder for storing resources:

- `/var/lib/ometascan` : installation folder with all its resources (database, DLP processed / CDR sanitized / quarantined file storage, engine package and definition updates).
- `/var/log/ometascan` : application logs.
- `/var/run/ometascan` : runtime folder for Core related processes.
- `/var/lib/ometascan` : all successful deployed engines data files.
- `/usr/lib/ometascan` : all shared libraries required by MetaDefender Core, bundled PostgreSQL server related binaries and libraries.
- `/var/tmp/ometascan/resources` : all temporary processing files for MetaDefender Core to process.

Storage Usage Information

Based on the configuration, MetaDefender Core will need additional disk space to store analysis/result data, quarantined files and temporary files:

	Data Storage
Analysis Reports (Processing History)	Approximately 4.5GB is required for every 1M analysis reports stored. • 100 GB is recommended as a baseline. Database storage requirements are affected by Data Retention settings, engine settings and scanning load.
Quarantined Files	Depends on the customers' dataset and configuration
Sanitized Files	Depends on the customers' dataset and configuration
Temporary processing files	Depends on the customers' dataset and configuration • A 1GB non-archive file requires 1GB free disk space at least. • A 1GB archive file requires (1 * 2)GB free disk space at least. • 50GB is a baseline and should be adjusted based on observation of the solution within the environment, with consideration to peak times and overall file makeup (e.g. complexity, size, archives).
Note	Options are available to control how large and/or complex files are handled to limit the amount of system resources required. See Workflow Configuration Template for more information.

Custom Engines

The recommendations above are specific for MetaDefender pre-packaged bundles.

However for additional Custom Engines, please review the Knowledge base to review additional requirements (if any) for the selected engine.

Browser Requirements for the MetaDefender Core Management Console

One of the following browsers is suggested to view the MetaDefender Core Management Console:

- Microsoft Edge
- Chrome
- Firefox
- Safari

Chrome, Firefox, Safari and Microsoft Edge browsers are tested with the latest available version at the time of release.

Network Bandwidth

In online environment where MetaDefender Core is configured to download engine update packages directly from OPSWAT update server source, then using gigabit network speed (125 MB/sec) is highly recommended to avoid potential network related issue while downloading files, or at least 100 Mbps (12.5 MB/sec) network speed.

Allowlisting URLs

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts':

- https://activation.dl.opswat.com - for license online activation
- https://update.dl.opswat.com - for engines and databases download

Note: *IP address-based allowlisting on your firewall might fail after some time since OPSWAT uses CDN (AWS Cloudfront) to faster delivery updates over the world, and IP address of edge servers might change over time.*

Depending on your location, connections with OPSWAT cloud-based servers be forwarded to specific AWS Cloudfront's edge locations, each is assigned with its own unique IP address ranges. You might also need to allow correct AWS Cloudfront's corresponding IP address ranges by referring to https://ip-ranges.amazonaws.com/ip-ranges.json

Warning

Even the OPSWAT update servers host updates for all of the available engines we support, sometimes custom engines might try to connect to their own cloud for updates, but this can be disabled in firewall and they will be updated just from OPSWAT cloud servers instead.

Differences Between MetaDefender Core v4 and v5

We are very excited to inform you about the release of version 5.0.0 of our MetaDefender Core!

The new version is designed to enhance the product performance, user experience, as well as to provide more flexibility, security, and scalability for our customers.

At the same time, we are also committed to keeping backward compatibility for whom running MetaDefender Core v4, and avoid the need for updating your client integration to move forward with MetaDefender Core v5.

v4 REST API Integration Supported
All public REST API endpoints available on MetaDefender Core v4 will be continously supported on v5.

v4 Product Major Features Supported
All product major features available on MetaDefender Core 4.21.2 (the latest one on v4) will be continously supported on v5.

Seamless Product Upgrade
Upgrading from MetaDefender Core v4 to v5 will be supported and covered with data migration without additional manual steps from users.

Licensing information will be also migrated.

Below are some highlights of the first v5 release, MetaDefender Core 5.0.0:

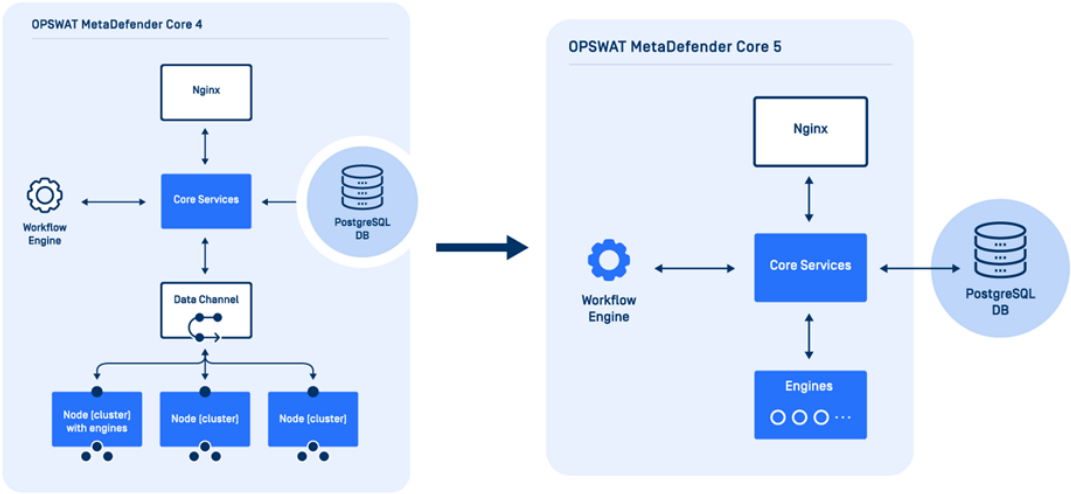
New product architecture

This version introduces a simpler product architecture that speeds up integration and automation, file processing flow, and immensely enhance the product performance and efficiency.

Earlier versions utilized MetaDefender Node concept, designed to support scaling of the scanning infrastructure by distributing scan requests among several different processing Nodes. Product licensing was based on number of Node instances. The benefit of having such a distributed infrastructure is that based on Node loads, Metadefender Core server can always choose the most appropriate Node to assign a new scan task to. However, having nodes in the v4 architecture caused more complexity and possible latency in the processing flow and entailed maintenance cost.

In this release, we retired multi-Node deployment and restructured MetaDefender Core's backend. This alteration significantly improves file processing speed and product efficiency, as well as eliminates extra cost for maintaining node services. Users now can simply install additional MetaDefender Core instances for scalability requirement,

as well as configure their own load balancer to coordinate tasks among multiple MetaDefender Core instances. Accordingly, product licensing is now based on number of the product instances.



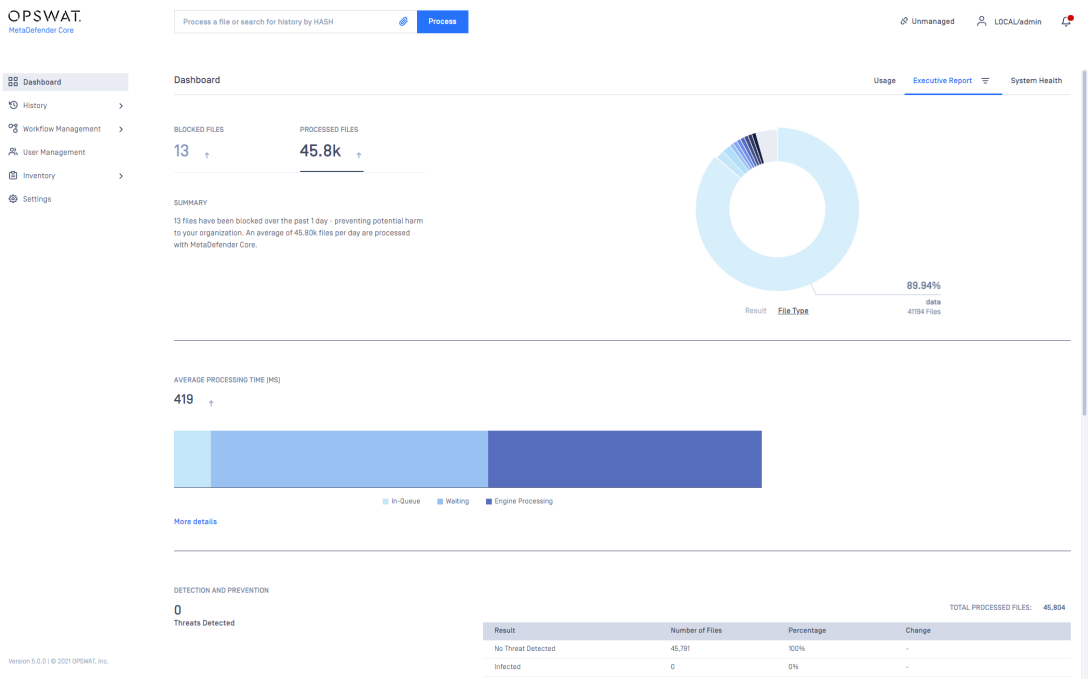
Please be informed that

- All existing administrative APIs related to "Node" on v4 will be kept as it is on v5 to ensure backward compatibility with existing client integrations.
- Applicable settings defined in Windows registry (Windows) or ometascan-node.conf config file (Linux) for Node will be migrated to Core. Details: MetaDefender Configuration

Brand-new product user interface

The second notable update in this version is a brand-new UI for the management console, which was designed with the focus on the quality and thoughtfulness of the user experience. The new administrative console UI provides an advanced set of Core features and observable processing status.

Comprehensive executive dashboard reporting offers dynamic, real-time and actionable threat insights. It also provides visibility of your security posture allowing you to paint a picture of security that aligns with the overall business. The report includes easy-to-read charts, graphs, and statistic explanation that help your security professionals reduce clutter, gather information quicker, spot threat activity early and act swiftly.



We're continuing to improve our product UI to provide users an excellent experience with the application and overall usability. Please expect more powerful UI features coming in next releases.

New compression archive engine for archive sanitization

In previous version, MetaDefender Core's archive engine implemented four functions including decryption, extraction, encryption and compression that with a high volume of files it takes time to process them all. For the purpose of enhancing archive handling performance, this release comes with a separated compression archive engine, which dedicatedly processes archive sanitization (compression and encryption). It is also created to facilitate maintenance and further feature development.

Archive sanitization feature now requires Deep CDR license

As to respect sanitization practice and ensure unknown threats prevention as designed, now starting on version 5, it's mandatory to have Deep CDR licensed for archive sanitization feature enablement on MetaDefender Core workflow configurations.

Archive handing related engines changed

The legacy "Archive" engine on MetaDefender Core v4 will be changed its name to "Archive Extraction" engine, even when you are still on MetaDefender Core v4.

A brand new "compression archive engine" is to dedicatedly handle archive sanitization in v5 (archive compression), that helps relevant archive handling performance improvement, easier for maintainance and further feature development.

Notes when you are still on MetaDefender Core v4 (not upgraded yet)

The legacy "Archive" engine (now name changed to "Archive Extraction" engine) on MetaDefender Core v4.x will still cover both extraction and compression archive tasks as usual.

Even though your MetaDefender Core v4.x might now have a new "Archive Compression" engine auto downloaded, but that engine cannot be deployed (marked as "permanently failed" in relevant engine status check API). This behavior is expected, and not causing any impact to MetaDefender Core processing flow, or its performance though.

- Just in case your client integration has a check for engine status before triggering scan request, then you might want to disable "Archive Compression" engine status on MetaDefender Core v4 using REST API. Detailed instruction: Archive Compression Engine Permanently Failed on MetaDefender Core v4
- When you later upgrade to MetaDefender Core v5 from v4, then you might need to manually re-enable "Archive Compression" engine to make it active and running on MetaDefender Core v5.

Notes for licensing

For offline deployment (MetaDefender Core is not connected to the Internet directly for online update), you will need to re-activate again on OPSWAT portal to retrieve latest engine list in .yml file including "Archive Compression" engine if you are currently licensed with Deep CDR.

Meanwhile, for online deployment, this manual procedure is not required since MetaDefender Core will cover that itself.

Import & export configurations

Not only workflow settings like in previous versions, but also all other general settings (update, security, data retention, server, hashes to skip) and most importantly, user settings, will be able to export and import on MetaDefender Core version 5.

Learn more at Import/Export configuration

Backward compatibility supported

The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

Windows application event logging disabled by default

Applicable only to Windows OS based MetaDefender Core.

Help avoid duplicated logging with separate file option, and improve overall processing performance. Of course, when being still valid to you, you are still supported to enable it back.

CSRF token header renamed

CSRF token originally created for security strengthening and being used by MetaDefender Core management console UI since version 4.21.0 via header `oms_csrf_token`

To get it compliant with some load balancers which do not accept underscore in header name by default, and save you from additional load balancer configuration step, we have now changed the header name to `oms-csrf-token`

This change does not impact to any REST API based client integration.

More comprehensive and accurate error messages on REST APIs

Impacting on following REST API endpoints:

- `PUT /admin/config/session`
- `PUT /admin/config/update`
- `PUT /admin/config/scan`
- `PUT /admin/config/skip`
- `PUT /admin/config/auditlog`
- `PUT /admin/config/webhook`
- `PUT /admin/config/rule`

Upload time and queue time logic updated in JSON scan response

Impacting on following REST API endpoints:

- `GET /file/{data_id}`
- `GET /hash/{md5/sha1/sha256}`
- `GET /archive/{data_id}`
- `POST /file/{async}`

Added new field:

- `receive_data_timestamp`: The timestamp when upload progress started (first byte received) (in milliseconds)
- `upload_time`: Total time elapsed for upload process (in milliseconds)

Modified existing field (new logic):

> **Backward compatibility supported**
> The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

- `upload_timestamp`: The timestamp when upload progress finished (all bytes received) (in milliseconds)
- `queue_time`: Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue time = start_time - upload_ timestamp) (in milliseconds)

Release version 5.0.0 also comes with variety of outstanding features and improvements.

For detailed information about our latest release please view our Release notes

Installation

Starting with MetaDefender Core 4.20.0, the centralized/shared database deployment model is supported (all MetaDefender Core instances share the same database), along with traditional standalone database mode (every MetaDefender Core instance has its own database to manage).

- Standalone mode
- Shared Database Mode

Installation

1. Download the package of your choice from My OPSWAT
2. Install the package on your computer via the Command Line or via the Wizard
3. Open a web browser and point to http://<server name or IP>:<port>
    - The default port is **8008**
    - In case of problem check Inaccessible Management Console page
4. Complete the required steps of the basic configuration wizard
5. You must activate this deployment to use its features

Installation notes

- If the MetaDefender Core package **dependencies are not installed** on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.
- During installation the databases might need to be upgraded. This could take noticeable time depending on database size (eg.: length of scan history).

Standalone mode

Wizard Installation

The Install Wizard is only for the Windows installer (.msi file).

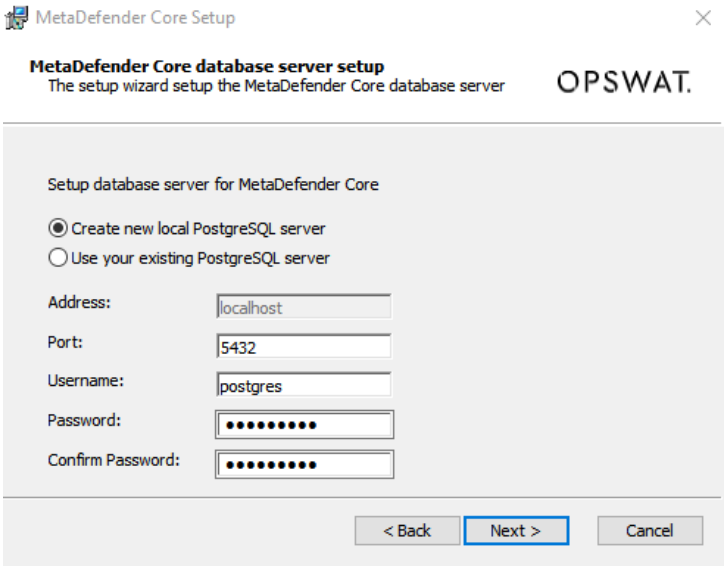
Since MetaDefender Core 4.19.0, we have switched to using PostgreSQL DBMS which needs a dedicated database server. MetaDefender Core supports two modes to setup PostgreSQL server:

Local (Bundle) PostgreSQL server

MetaDefender Core will install a new PostgreSQL server locally in the same box with the product.

> **PostgreSQL user credentials**
> Unicode characters are NOT supported for "Username" and "Password"
>
> "Password" does not accept # (hash) as starting character (e.g. #ABC)

> **The ignition file**
> If the ignition file exist in the designated location, then MetaDefender Core will use the details defined in the ignition file, rather than what is passed via the installation wizard.
>
> More info about the ignition file based installation: Command Line Installation

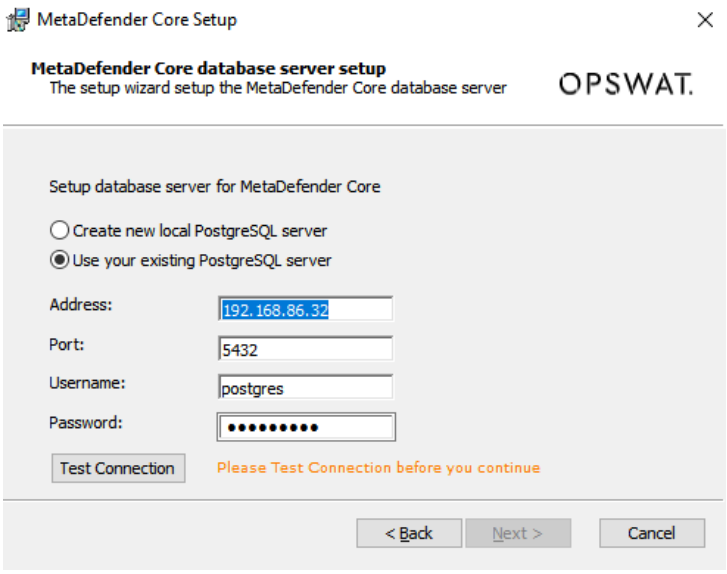


Remote (Existing) PostgreSQL server

MetaDefender Core will connect to remote pre-installed PostgreSQL running remotely.

> **PostgreSQL connection configuration**
> "Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

> **PostgreSQL user privilege requirements**
> MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.
>
> Details: PostgreSQL User Privilege Requirements



Command Line Installation

Ignition File

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Make sure to create the ignition file, if it doesn't exists before you go ahead and install MetaDefender Core via command line.

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

MetaDefender Core supports two modes to setup PostgreSQL server:

1) Local PostgreSQL server

- MetaDefender Core will install a new PostgreSQL server locally, next to the product.
  - A sample iginition file for PostgreSQL server information (***Only non-Unicode characters supported for "user" and "password"__***):

ometascan.conf bash

```
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
```

2) Remote PostgreSQL server

> **Info**
>
> **PostgreSQL version 12 (at least 12.3) or version 14 is required.**
>
> According to PostgreSQL notice, version 12 will stop receiving fixes on November 14, 2024, so we suggest that you use version 14.

MetaDefender Core will connect to a pre-installed PostgreSQL server running remotely and will setup its database.

Make sure to follow steps at Open Connection On PostgreSQL Server first, so that your remote existing PostgreSQL server to accept upcoming connection from MetaDefender Core.

A sample ignition file for PostgreSQL server information:

ometascan.conf bash

```
[dbserver]
type=remote
host=192.168.86.32
port=5432
user=existingserver_admin_user
password=existing_server_pass
```

After installing MetaDefender Core successfully, you may want to remove all credentials info created for [dbserver] section in ignition file for security reason.

> **Warning**
> If the MetaDefender Core package dependencies are not installed on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.

Debian package (.deb)

bash

```
sudo dpkg -i <filename> || sudo apt-get install -f
```

On Red Hat Enterprise 7.x Linux / CentOS package (.rpm)

bash

```
sudo yum install <filename>
```

For systems which enabled GPD check flag:

bash

```
sudo yum install --nogpgcheck <filename>
```

Red Hat Enterprise Linux 8.x package (.rpm)

Install the latest PostgreSQL repository RPM:

> **Note**
> Due to the shorter support cycle on Red Hat Enterprise Linux, all supported versions of PostgreSQL are not available on this platform. The PostgreSQL Yum Repository will integrate with the normal systems and patch management, and provide automatic updates for all supported versions of PostgreSQL throughout the supported lifetime of PostgreSQL. Reference: https://www.postgresql.org/download/linux/redhat/

bash

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

In case installing a remote PostgreSQL on either the same MetaDefender Core server or another server, you need to install the following additional package. PostgreSQL version 12.3 or newer is recommended.

bash

```
sudo dnf install -y postgresql12-contrib
```

Install MetaDefender Core:

bash

```
sudo dnf install -y <filename>
```

Windows package (.msi)

On Windows systems it is possible to install the product by running the corresponding .msi file.

From command line interface it is also possible to install the product by executing

bash

```
msiexec /qn /i <msi file name> <option key>=<option value>
```

where the possible keys and their default values are the following:

Key	Default Value	Description
INSTALLFOLDER	\Program Files\OPSWAT\MetaDefender Core	Customize installation folder for product Example: `INSTALLFOLDER="D:\Products"`
RESTADDRESS	*	REST interface binding IPv4 or IPv6 address ('*' means that service listens on all IPv4 and IPv6 interfaces)
RESTPORT	8008	REST interface binding port
REPORT_ENGINE_ISSUE	true	Enable reporting of engine issue count. (possible values: "true" or "false").

Note

Reporting of engine issue count

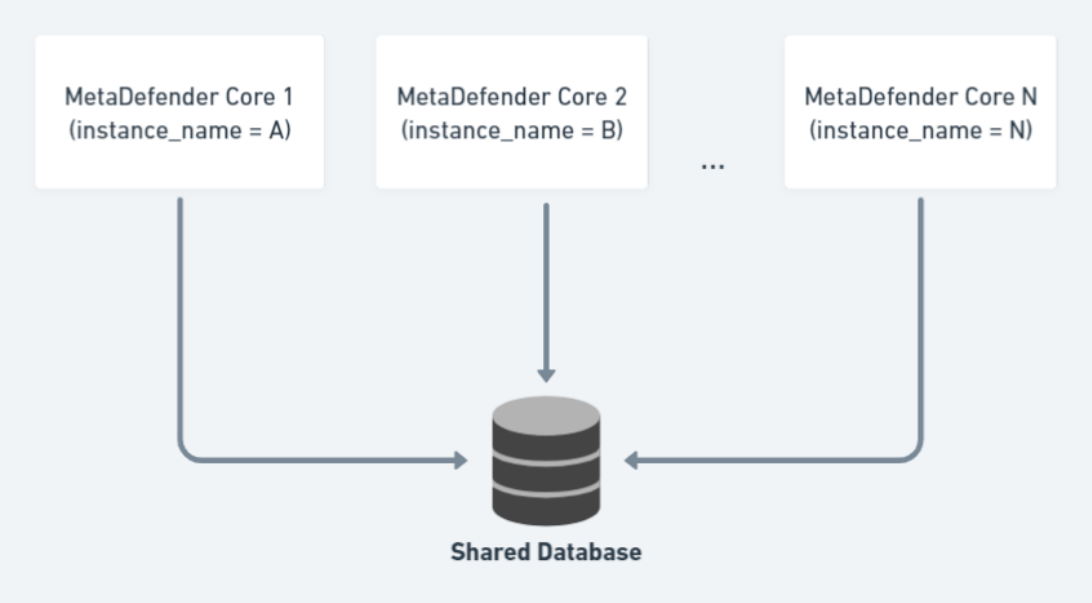
If reporting of engine issue count is enabled, MetaDefender Core server will send only the **number** of initialization errors and **number** of unexpected stops for the specific db/engine version. This information is sent over a HTTPS channel when the product downloads the latest package descriptors. This information is used for early detection of any specific 3rd party engine quality issues.

For details on using msiexec please consult Windows installer documentation.

Shared Database Mode

Starting with MetaDefender 4.20.0, you can use a centralized database deployment model, where all MetaDefender Core instances can share the same external database.

That helps auto-sync all product configurations (**except Single-Sign-On** that requires unique settings for each Core), processing history, live/on-going scan result query, audit, statistics data.



This article demonstrates how to install and setup MetaDefender Core 4.20.0 or newer with shared database mode.

Important notes & current limitations:

Currently not supported

- Does not support Central Management use-case with shared DB mode.
- Does not support to switch from shared database mode back to standalone database mode.
- Does not support to change instance_name of MetaDefender Core in shared database deployment model after installation.
- Does not support upgrading older MetaDefender Core from non-persistent mode to using the shared database mode.
- Does not support upgrading older MetaDefender Core from standalone mode to using the shared database mode.
- Does not support to upgrade multiple MetaDefender Core instances at once in shared database mode
    - Stop all Core instances' scanning service before performing the upgrade.
    - Upgrade each Core server one by one instead.

- For each Core, when the upgrade finished, then you can start using that Core for scanning without waiting for the other Core instances' upgrade to be done.

> **Disclaimer**
> MetaDefender Core does not support mode switching.

Requirements for shared database setup:

- The central remote PostgreSQL server (for shared database) could be utilized with a built-in (native) PostgreSQL database server installed with one of the MetaDefender Core instances in the network. Or using a dedicated existing PostgreSQL database server (see detailed requirements for PostgreSQL version at Recommended System Configuration).
- Before setting up, make sure to follow steps at Open Connection On PostgreSQL Server so that all MetaDefender Core instances could connect to the central remote PostgreSQL server.
- MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core, details: PostgreSQL User Privilege Requirements
- All MetaDefender Core instances must be installed on the same Operating System (Windows / Linux).
- The first MetaDefender Core instance connects to the shared database must be ready first (license activated, users/groups/roles created) before setting up other MetaDefender Core instances to connect and use that shared database.

Engine update:

- If engine updates is configured to be via Folder source, all MetaDefender Core instances must be able to access to the configured update folder, and the option DELETE FILES AFTER IMPORT will be unchecked and prevented for modification.

Authentication:

- Single-Sign-On (SSO) authentication settings (if using) cannot be shared among MetaDefender Core instances because SSO requires an unique login URL for every different Core.
- If an user is locked for login on a certain MetaDefender Core instance(e.g. exceeded the number of failed login times), that user is not prevented for logging in on any other instances.
- If an user is locked for sign-in on a certain MetaDefender Core instance (e.g. exceeded the number of failed login times), if the admin releases lockout on that user on that MetaDefender Core instance, that user is still not able to login on other MetaDefender Core instances.

Sanitized / DLP / Quarantined location changed after MetaDefender Core install:

Windows:

Shared storage path will be stored in 3 strings under key global in registry: quarantinepath, dlppath, sanitizepath as below

(Default)	REG_SZ	(value not set)
dbmode	REG_SZ	4
dlppath	REG_SZ	\\10.40.50.100\test
instance_name	REG_SZ	481rvj
quarantinepath	REG_SZ	\\10.40.50.100\test
restaddress	REG_SZ	*
restport	REG_SZ	8008
sanitizepath	REG_SZ	\\10.40.50.100\test

You can set either 1, 2 or 3 different paths for the 3 settings above (means that quarantined, DLP and sanitized files will be stored in same or different storages corresponding). In case shared storage is not set in installation, values of 3 settings above will be empty.

Restart MetaDefender Core service after changed.

Linux:

Open file `/etc/ometascan/ometascan.conf` and set shared storage path for 3 keys quarantinepath, dlppath, sanitizepath under section [global]

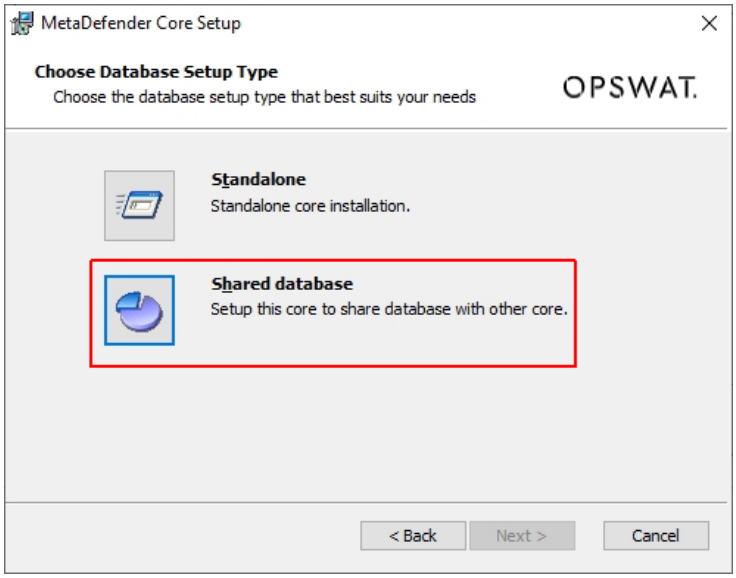
Restart ometascan service after changed.

Wizard Installation (Windows Only)

> **Info**
> Only applicable to MetaDefender Core 4.20.0 or newer using wizard on Windows OS.

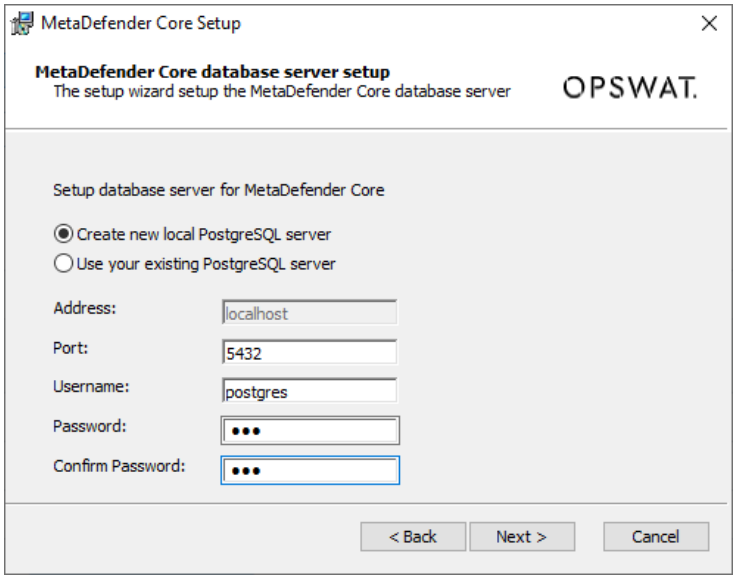
Clean install scenario

1. **Choose Shared database option:**

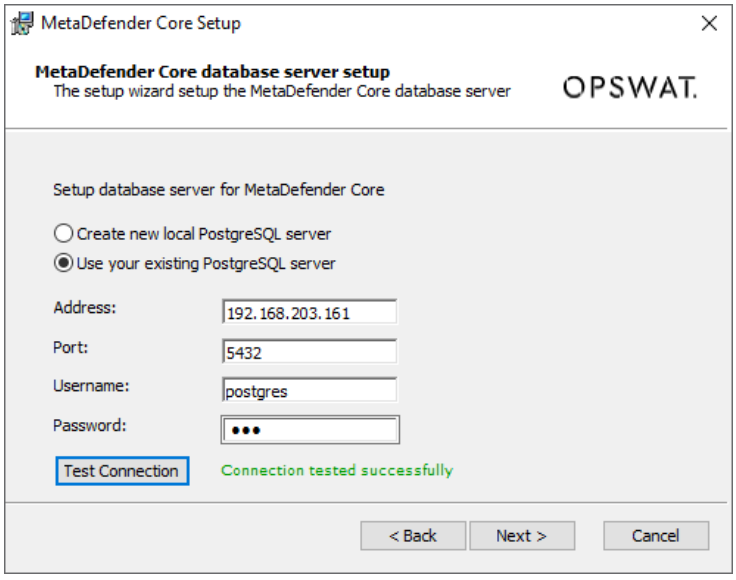


2. **Setup database server for MetaDefender Core:**

You can choose to create a native / built-in PostgreSQL server and setup the shared database on (then other MetaDefender Core instances will want to connect to that PostgreSQL server afterwards)



Or choose to connect to a pre-setup existing PostgreSQL server:



PostgreSQL connection configuration

"Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

PostgreSQL user privilege requirements

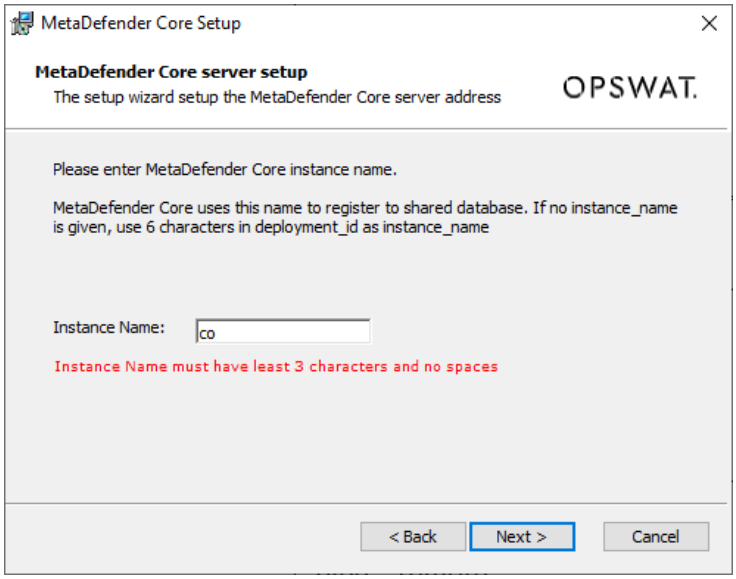
MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.

Details: PostgreSQL User Privilege Requirements

3. **Create an unique instance name for that MetaDefender Core (different from other MetaDefender Core instances):**

Instance name must contain at least 3 characters and no white space.

If you choose to leave it blank, which means it will use the default instance name with 6 characters extracted from that MetaDefender Core deployment ID.

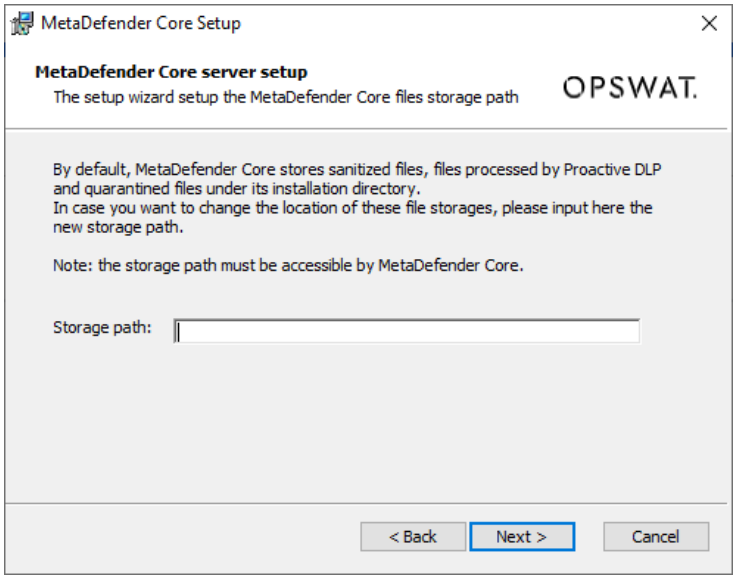


4. **Input the file storage path (sanitized, DLP, quarantined files)**

In the shared database model, all MetaDefender Core instances could use the same folder path to store all their own sanitized, DLP, quarantined files after processed.

The default location is: ***<MetaDefender Core installation folder>\data\ {sanitized|dlp|quarantine}***

If you wish to use that default location, just leave the input box blank and press Next.



Otherwise, you can specify a different location of sanitized/dlp/quarantined files. It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core.

For example:

- C:\new_folder
- \192.168.1.1\new_folder

Command Line Installation

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Clean install scenario

1. Create ignition file if not existed

- Windows: **C:\OPSWAT\ometascan.conf**
- Linux: ***/etc/opswat/ometascan.conf***

An example of ignition file:

ometascan.conf bash

```
eula=true

[global]
dbmode=4
instance_name=MetaDefenderCoreA
storage_path=D:\path\

[user]
name=admin
password=admin
email=admin@local

[dbserver]
type=remote
host=192.168.1.1
port=5432
user=postgres
password=your_own_password
```

Field	Required	Mapping to wizard UI	Possible values
global/dbmode	optional default is 1 (standalone)		1 - Standalone 2 - Central Hub deployment 3 - Non-persistent **4 - Shared**
global/instance_name	optional default is 6 characters of deployment ID		Unicode and number At least 3 characters No whitespace
global/storage_path	optional default is empty		Path to storages (DLP, quarantined, sanitized files) It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core. For example: • C:\folder • \192.168.1.1\folder

2. Run installers using command line to install

Please refer to Command Line Installation for installation command.

Deployment automation support

Note

An Ignition file could be utilized for either / both of following use-cases:

1. Pre-define PostgreSQL server information for MetaDefender Core to connect and setup database: only required on Linux installation, or Windows with command line installation. Learn more details at:
    a. Standalone Mode > Command Line Installation (applicable on version 4.19.0 or newer).
    b. Shared Database Mode > Command Line Installation (only applicable on version 4.20.0 or newer).
2. Automating to accept EULA, create a default local admin user and import MetaDefender Core configurations upon installed: Those steps could be automated using ignition file described in this page.

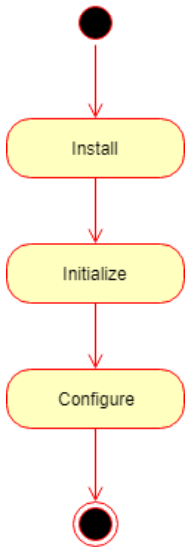
Ignition file location to create:

- **Windows**: `C:\OPSWAT\ometascan.conf`
- **Linux**: `/etc/opswat/ometascan.conf`

The product supports fully automated deployment. It means that it can be installed and configured with no human interaction.

The automated deployment can be split to three steps on a high level:

1. Installation
2. Initialization
3. Configuration



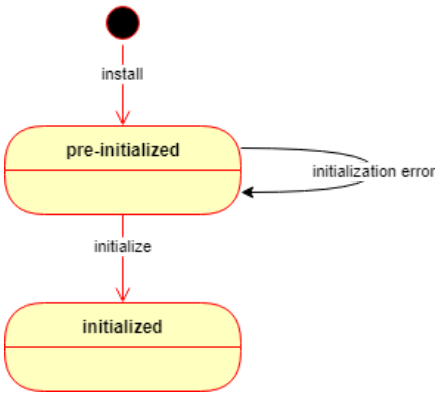
Installation

To automate the installation, install the product from the command line and provide the installation-time options as parameters to the installer. For further details see Command Line Installation

After the installation is complete, the product starts up and waits in a pre-initialized status. The product may be initialized in two ways:

1. Manually using the Wizard Setup or
2. Automatically using an *ignition file* (see below).

> **Warning**
> If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried after fixing the problem (e.g. placing the ignition file in its lookup location) and restarting the OPSWAT MetaDefender Core service.
>
> Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

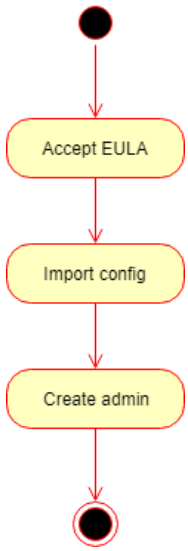


Initialization

Initialization is the process of bringing the product to an operable status.

Basically the initialization consists of the following steps:

1. Accept the End User License Agreement (EULA),
2. Import product configuration and
3. Create the first administrator user account.



Ignition file

The initialization process can be configured in a file called the *ignition file*.

> **Warning**
> In Linux, the ignition file and the folder /etc/opswat should have the permission 755

Ignition file location

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

The ignition file must be in conf format.

ometascan.conf bash

```
eula=true
[user]
name=admin
password=admin
email=admin@local
apikey=44c96cbb4e4e6c0a1d16bcb3c3da24def317
[config]
import=config_package.zip
import_password=123
[security]
session_cookie_samesite=Lax
[internal]
skip_startup_pages=true
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
private_username=my_internal_db_user
private_password=mypassword
[ocm]
ocm_url=https://my.opswat.com
reg_code=XXXXXXXX
```

Ignition file fields

The ignition file accepts the following fields:

Section	Key	Required	Accepted Values	Description
	eula	Optional (mandatory when having [user] section)	true false (default)	Whether to accept the End User License Agreement. This key must be set at the top of ignition file. This key must be set to "true" to accept the EULA. Any other value will cause the initialization to fail. When having [user] section defined, then this key must be set.
[user]		Optional		Initial administrator user account properties. The Administrator role is granted to the account.
	name	Optional	string	User name for the initial administrator user account.
	password	Optional	string	Password for the initial administrator user account. WARNING! Clear text password The password in this configuration file must be stored in its clear-text format and as so it may be visible for unauthorized parties.
	email	Optional	string	E-mail address for the initial administrator user account.
	apikey	Optional	string	API key for the initial administrator user account.
[config]		Optional		
	import	Optional	string	Path to a file in json/zip format that contains a previously exported configuration to be imported. Zip-format file contains a full configuration including user management settings, so it will overwrite [user] section.
	import_password	Optional	string	Password to decrypt .zip configuration package.
[security]		Optional		

Section	Key	Required	Accepted Values	Description
	session_cookie_samesite	Optional	Strict Lax (default) None	Declare if session cookie should be restricted to a first-party or same-site context.
[internal]		Optional		
	skip_startup_pages	Optional	true false (default)	Skip all welcome and other startup pages (e.g. upgrade result) when installing / upgrading MetaDefender, and going straight to the product index default page.
[dbserver]		Optional		
	type	Mandatory when having [dbserver] section	local remote	Where the PostgreSQL database server should physically locate. "remote" means PostgreSQL server and MetaDefender Core are not in the same machine.
	host	Mandatory when having [dbserver] section	string	IP address / domain name of the server where PostgreSQL server locates. "localhost" can be used when applicable.
	port	Mandatory when having [dbserver] section	number	Port of PostgreSQL server is listening for connections from clients (i.e. MetaDefender Core).
	user	Mandatory when having [dbserver] section	string	PostgreSQL server's user. SUPERUSER privilege is required for MetaDefender Core to setup its database and extensions for the first time. **Only non-Unicode characters supported.**

Section	Key	Required	Accepted Values	Description
	password	Mandatory when having [dbserver] section	string	PostgreSQL server's user credentials. **Only non-Unicode characters supported.**
	private_username	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
	private_password	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
[ocm]		Optional		
	ocm_url	Optional	string	URL to My OPSWAT server.
	reg_code	Optional	string	A registration code to enroll to My OPSWAT server.

Detailed initialization process

1) After the product has been started, it looks for the ignition file in the designated location.

Windows: C:\OPSWAT\ometascan.conf ** Linux: /etc/opswat/ometascan.conf

** 2) If an ignition file is found, then

2.1) It gets validated, and if it is valid, then based on the information found in the ignition file:

- The EULA is accepted,
- The configuration is imported,
- The administrator account is created.
- If any of the above steps fails, then the error is logged, and the initialization gets terminated.

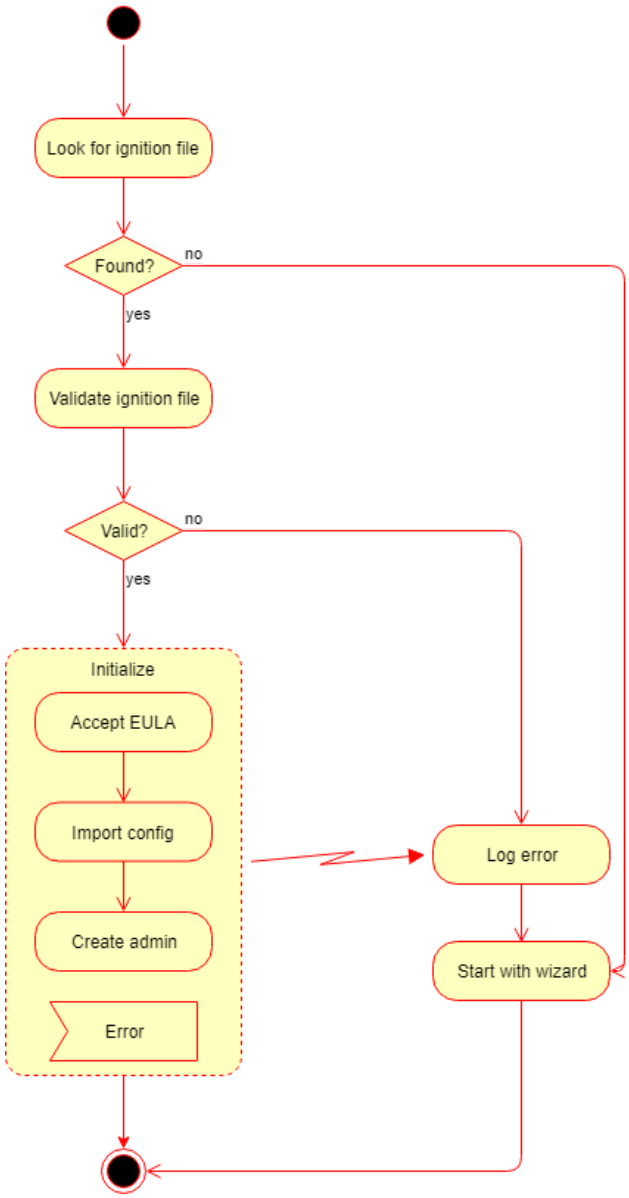
> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

2.2) If it is not valid, then the error is logged, and the initialization gets terminated.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

3) If there is no ignition file, then no initialization is performed.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.



Warning

If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried fixing the problem (e.g. placing the ignition file to its lookup location) and restarting the OPSWAT MetaDefender Core service.

Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

Configuration

After the initialization is complete, the product is ready with the default and the imported configuration.

This configuration can be later changed calling the configuration API functions. For further details about the API see Configuration related APIs API

CIS Level 1 Guidelines

Support Red Hat Enterprise Linux 9
For more details about Center for Internet Security (CIS) please refer to this document

Instruction steps

I. Install OpenSCAP

bash bash

```
yum install openscap-scanner scap-security-guide
```

II. Generate a result file and a HTML report using OpenSCAP scanner tool

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

III. Remediation of CIS Level 1 issues

Generate a remediation script based on the `ssg-rhel9-ds.xml` file:

bash

```
oscap xccdf generate fix --profile
xccdf_org.ssgproject.content_profile_cis_server_l1 --fix-type bash --output
remediations.sh /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

And execute remediation script.

bash

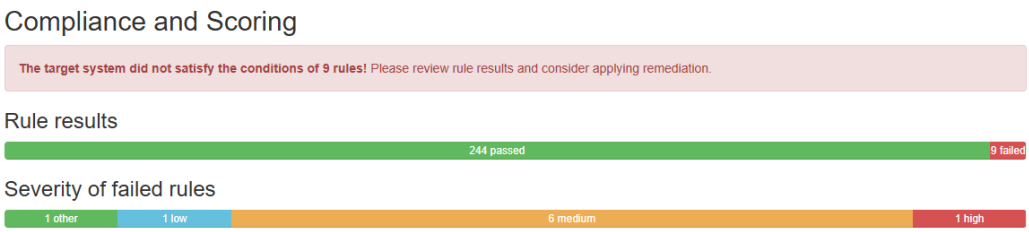
```
./remediations.sh
```

IV. Review the results after remediation

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

Example:



Notes

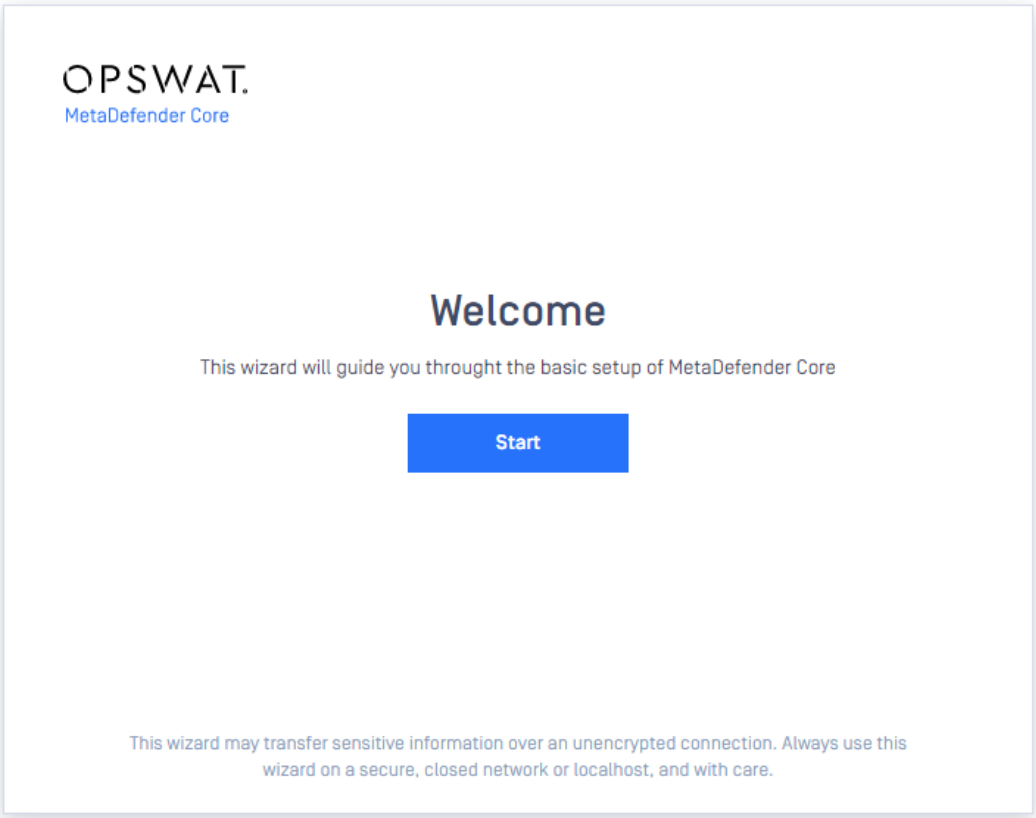
- CIS Level 1 requires `/tmp` folder to be mounted in a separate partition. Please ensure that that new partition have enough disk space for MetaDefender Core to run.
- CIS Level 1 requires that "Ensure No World-Writable Files Exist".
    - For now, when freshly installing MetaDefender Core, all its binary files meet the requirement.
    - When installing/updating engines, some engines might create additional files for its operation, and it might violate this requirement. In this case, you need to again execute the remediation script in the step III.

Wizard Setup

Introduction

Access the MetaDefender Core management console for the first time (i.e. http://localhost:8008), you are to complete a quick walkthrough for wizard setup in order to be able to use the product.

The administrative management console will be available only upon you have successfully finished this wizard stage.

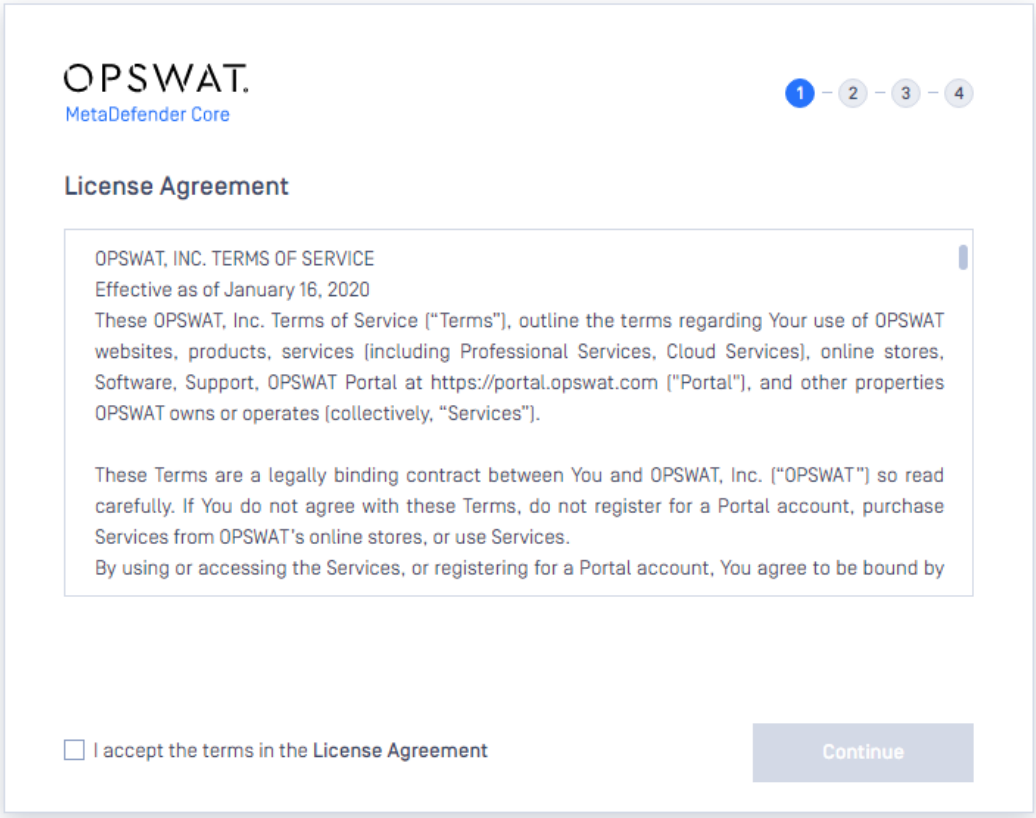


Warning

This wizard may transfer sensitive information over an unencrypted connection. Always use this wizard on a secure, closed network or localhost, and with care.

Basic configuration steps

End-User License Agreement



In the first page you can find the End-User License Agreement. You have to accept the terms before moving on.

Please read through the EULA (https://docs.opswat.com/policies/privacy-policy/portal-privacy-policy) carefully and if you agree with it, check `I accept the terms in the **License Agreement**` and click `Continue` to proceed.

Local Admin User Setup

The next step is to set up a default local administrator account.

This account will be the first one being able to access the administrative management console, and to create accounts for other users. You have to fill out all fields in this page to be able to move forward. When you are done, click NEXT to continue.

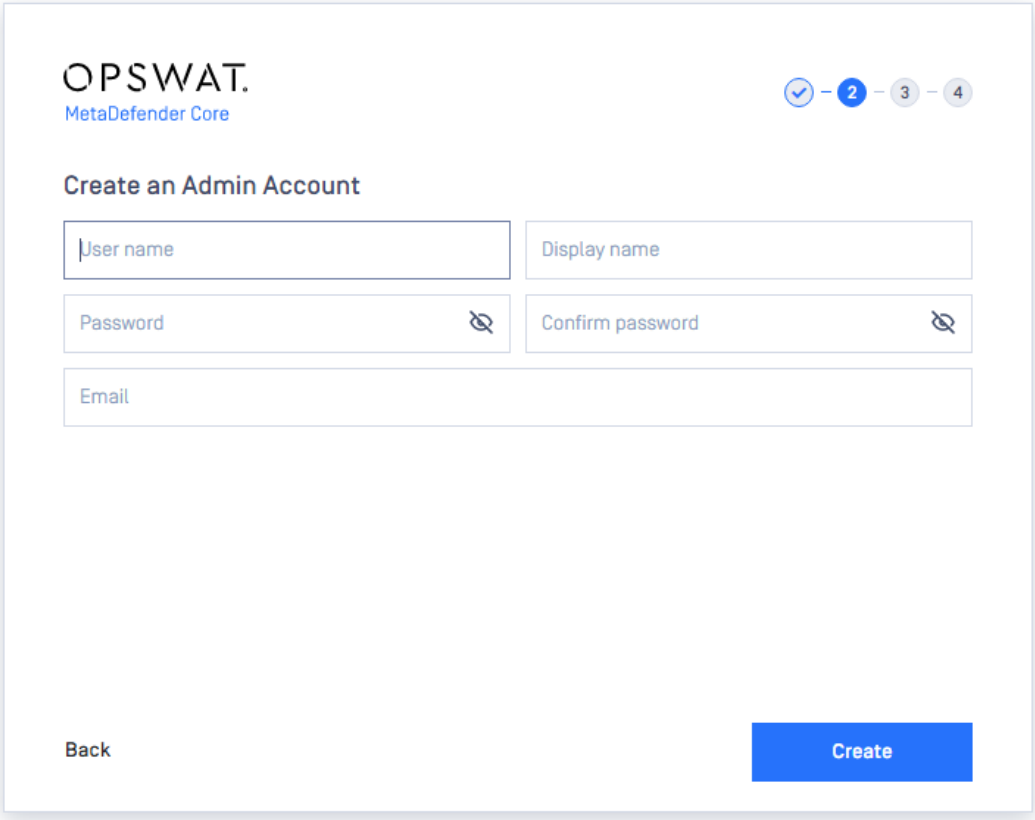
Info - User Directory

The administrator account, that is created via the basic configuration wizard, is always added to the LOCAL user directory as a member.

Learn more about user directory: User directories

The following information is required for the administrator account:

ACCOUNT NAME	The unique name of the account that is used at the time of login and in log messages for accountability.
ACCOUNT DISPLAY NAME	Name of the person bound to this account. This name (appended to the name of the account's user directory) is displayed in the top right corner of the administrative management console.
PASSWORD	Password of the user bound to this account that is used at the time of login. Passwords sent clear-text As long as TLS is not configured for the basic configuration wizard, passwords are sent clear-text over the network and may be disclosed to unauthorized parties. As a mitigation action: 1. Either use the wizard on *localhost* or on a direct network connection, or 2. Enabling HTTPS as soon as possible and change the password immediately if it has already been set.
EMAIL	Email address of the person bound to this account.



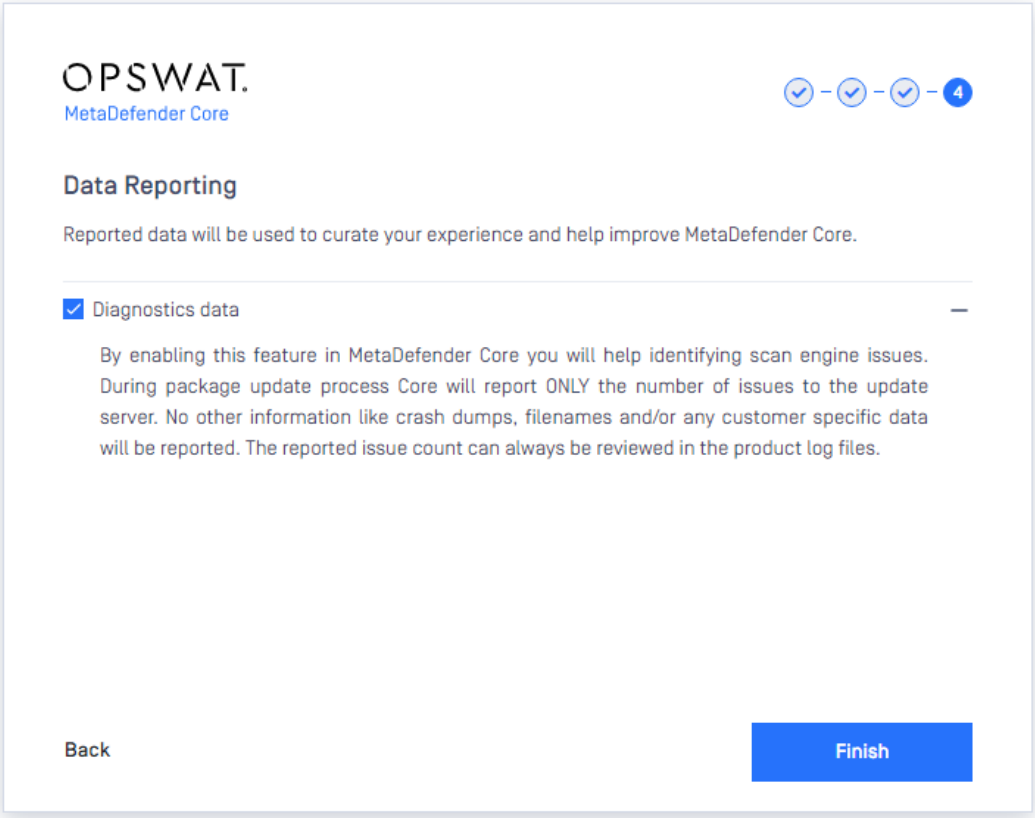
License activation

For license activation details see License Activation

Data Reporting

Enabling this option to allow MetaDefender Core collect its own engine issues and report back to OPSWAT for further analysis.

MetaDefender will not collect any other information.

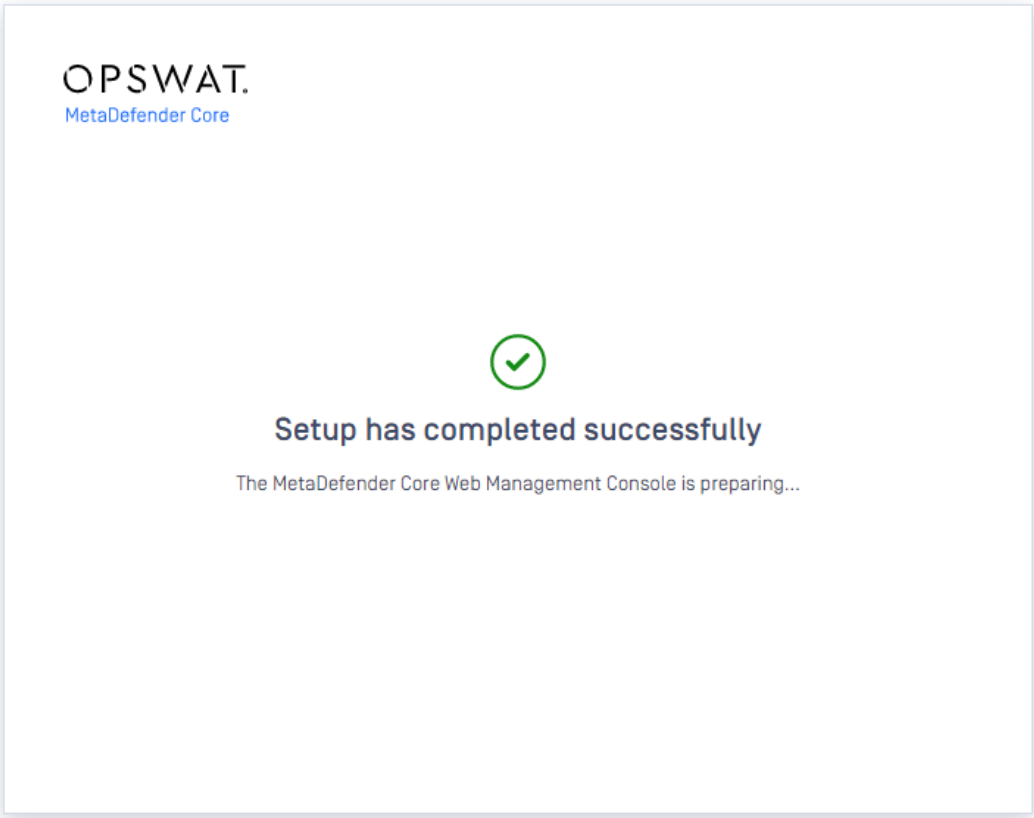


Wizard completion

After you have completed every steps you are ready to finish the wizard and start using the product. Click the `Fishish` button to complete the wizard.

> **Warning**
> The product's service will be restarted and the browser will be redirected to the administrative management console. This could take several seconds.

You can login to the Web Management Console with the administrator user that have just been created in the previous steps.



Setup HTTPS

Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network. Websites, like the Web Management Console, are able to use TLS to secure all communications between their servers and web browsers.

The TLS protocol aims primarily to provide confidentiality (privacy) and data integrity between two communicating computer applications.

Warning

No TLS for the wizard

By default, TLS is not enabled for the basic configuration wizard. As a consequence sessions between the wizard's backend and the browser may be insecure.

Info

Performing the same steps as for the Web Management Console, it is possible to set up TLS for the basic configuration wizard. Remember completing the TLS setup before launching the wizard.

For instructions to set up TLS see Enabling HTTPS

License Activation

##

To activate your installation go to the **Settings > License** menu in the Web Management Console. If you have no valid license, you will only see your installation's **deployment ID**. You will also see a warning in the Web Management Console header.

After successful activation, the product will start downloading the latest available scan engines and malware databases. You can follow the status of the scan engine installation on the **Inventory > Modules** page.

When all the licensed engines are installed, you can start using the MetaDefender Core to process files - File Processing.

Deployment ID

Each MetaDefender Core instance represented by an unique system fingerprint called `Deployment ID`

MetaDefender Core licensing is purely based on deployment ID activated. Therefore, everytime your MetaDefender Core is activated with a different deployment ID, then it will be captured as a different activation record accumulated into your total activation times allowed.

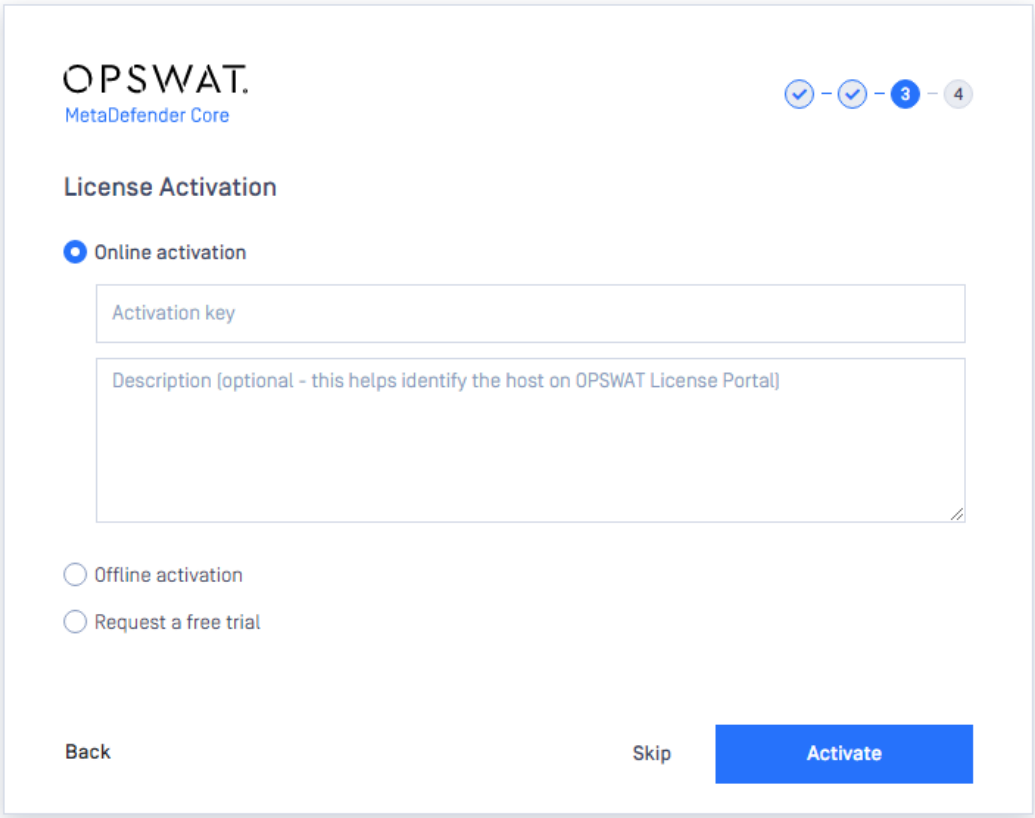
> **Notice**
> Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you alter the system where MetaDefender Core relies on (e.g. network card, OS hostname).
>
> Deployment ID is an UUID version 4 random based string. Learn more at: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you update the system where MetaDefender Core relies on.

Activation Modes

There are three different options for the product activation:

- Online activation: MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.
- Offline activation: Requiring you to upload a license file (.yml). That license file could be retrieved by activating your license via OPSWAT Portal. Follow the displayed instructions for details.
- Request trial key online: For evaluation purpose, you can receive an trial Activation key via email. Follow the displayed instructions for details.



Online activation

MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.

Required to input your Activation License key, and optional for description (but we recommend this for troubleshooting and auditing purpose if needed).

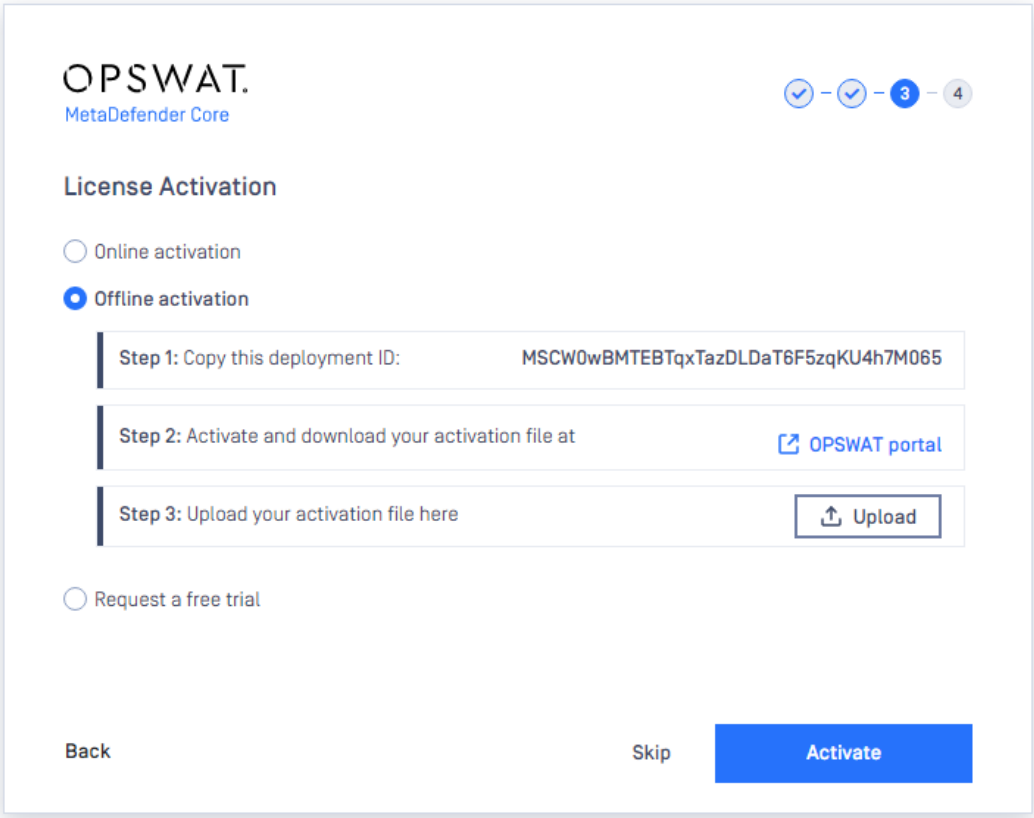
Warning

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts' for license activation:

- https://activation.dl.opswat.com Note: *IP address-based whitelisting on your firewall might fail after some time since OPSWAT uses CDN (Content Delivery Network) to faster delivery updates over the world, and IP address of edge servers might change over time.*

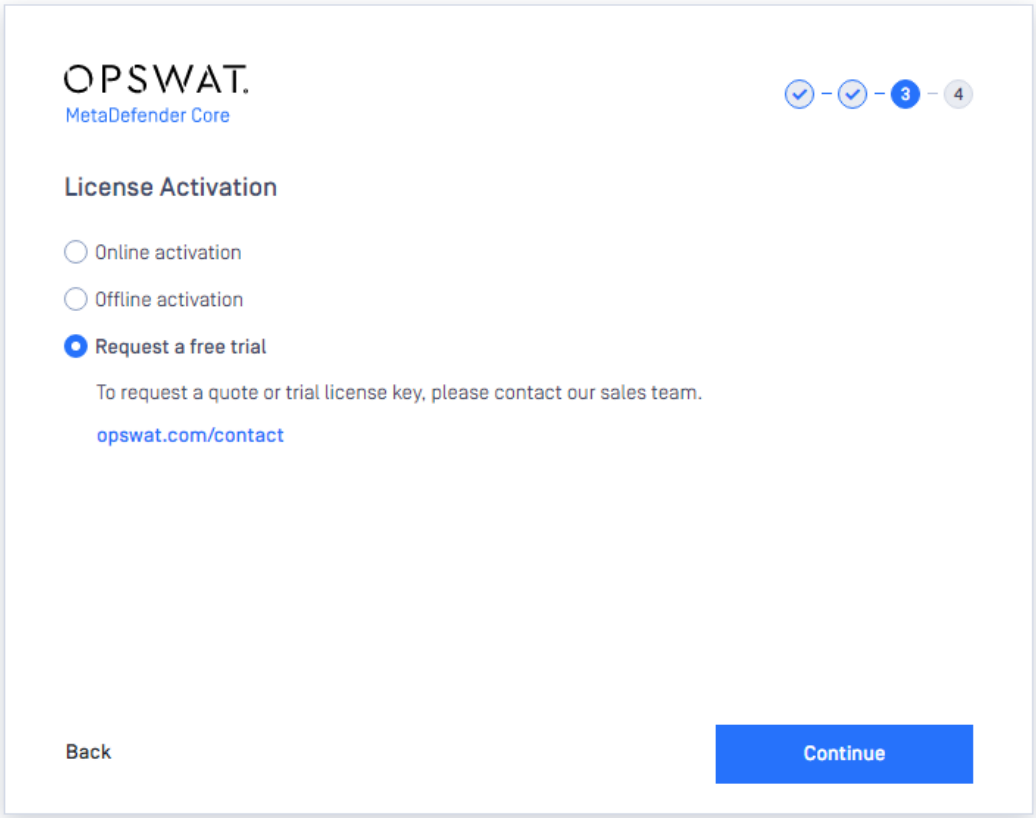
Offline activation

With no internet connection on the server the Metadefender v4 instance may be activated indirectly from a different machine, that has internet connection. The Deployment ID of the Metadefender v4 instance and the the Activation key received at the time of purchasing the product will be required. Follow the steps on the screen to activate the product offline.



Request trial key online

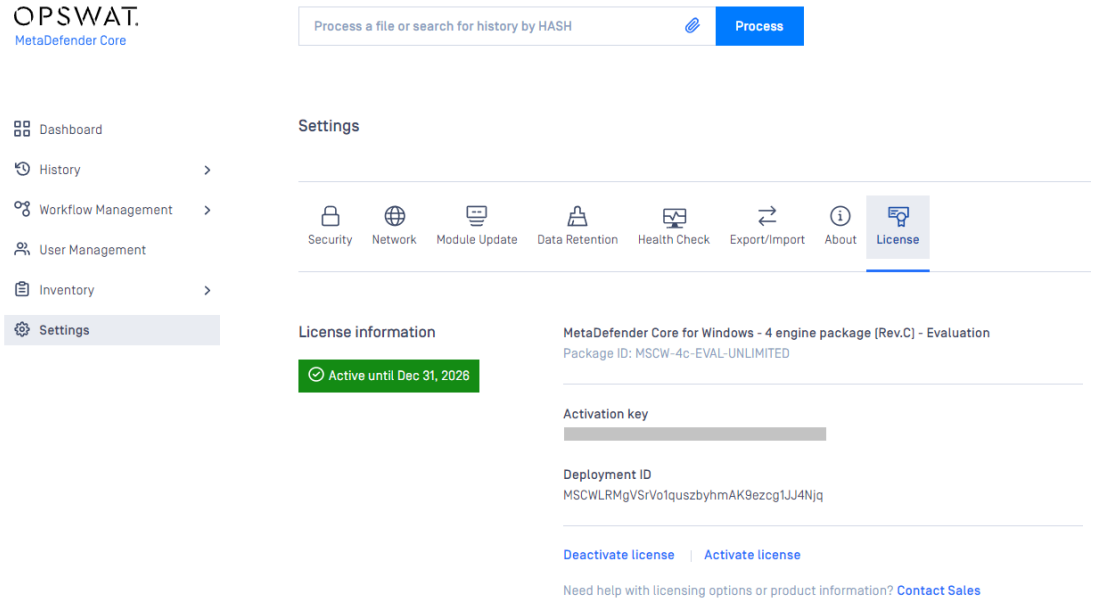
An evaluation license could be acquired by contacting our sales team.



Check Your License Details

For more license details and activating your MetaDefender Core, go to Settings > License menu tab on the Web Management Console:

- Product name: product name as on your order (e.g. MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation)
- Package ID: product identification as on your order (e.g. MSCW-4c-EVAL-UNLIMITED)
- Expiration: last day of license validity
- Activation key: key used for activating product
- Deployment ID: identification of this installation


OPSWAT.
MetaDefender Core
Process a file or search for history by HASH
Process
Dashboard
History
Workflow Management
User Management
Inventory
Settings
Settings
Security
Network
Module Update
Data Retention
Health Check
Export/Import
About
License
License information
Active until Dec 31, 2026
MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation
Package ID: MSCW-4c-EVAL-UNLIMITED
Activation key
Deployment ID
MSCWLRMgVSrVo1quszbyhmAK9ezcg1JJ4Njq
Deactivate license
Activate license
Need help with licensing options or product information? Contact Sales

Scripted license management

Using REST API calls there is an option for scripted activation and deactivation for MetaDefender Core servers with **no Internet connection or if your infrastructure is using automation to create/destroy MetaDefender Core instances**. In this chapter the steps of these two scenarios are described.

Requirements

- an installed MetaDefender Core instance without Internet connection
- another computer that has Internet connection and can run your activation scripts
- a manual or automated way to transfer data between the two computers

Activation steps

1. In order to activate MetaDefender Core, deployment ID and activation key are needed to generate the license file.
    a. Activation key should be purchased from OPSWAT.
    b. MetaDefender Core deployment ID can be queried by using REST API (for details see GET Get Current License Information API page)

> **Info**
> Save this deployment ID in your system, you might need this ID when the instance is unavailable at the time of deactivation.

2. Activate license and get the license file using the following URL, where <quantity> is 1.

activation call bash

```
curl -X GET https://activation.dl.opswat.com/activation?key=<activation-key>&deployment=<deployment unique-ID>&quantity=<quantity>
```

If the activation is successfully, a license file is downloaded. Save this file.

HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	license file	Activation was successfully.
200 Activation failed	error: '<user conform error message>'	Failed activation
200 Invalid parameter	error: 'Could not activate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not activate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
200 Invalid parameter	error: 'Could not activate your product because the quantity you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key, quantity or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Info
In case of any activation issue, contact OPSWAT support for help

3. Upload license file to MetaDefender Core: The license file should be uploaded to the MetaDefender Core v4 to activate the product. For details see page: POST Upload license key API

Deactivation steps

1. For deactivation, following infomation is required:
    a. Activation key
    b. MetaDefender Core deployment ID: can be queried by using REST API (for details see GET Get Current License Information API page)
2. Deactivate license using the following URL:

deactivation call bash

```
curl -X GET https://activation.dl.opswat.com/deactivation?key=<activation-
key>&deployment=<deployment unique-ID>
```

Response HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	result: ok	Successful deactivation
200 No active license found	error: 'Could not found any active license with the given parameters'	The license has not been activated yet or it has been deactivated already.
200 Invalid parameter	error: 'Could not deactivate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not deactivate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Important notes

Warning

Product activation is tied to several hardware, operating system and software parameters. In case of one or more major hardware or software parameter change the product might turn into deactivated status. Operating system updates, other software updates should not affecting the activation status.

If the product is online activated and has live Internet connection then in case of deployment ID change the product reactivates itself. If the product doesn't have Internet connection then the administrator is responsible to reactivate the product.

Because of the above administrator should consider licensing restrictions of the offline deployments if using any virtualization and/or containerization technologies.

File Processing

MetaDefender Core provides REST API interface for file processing business.

There are several ways to process files with MetaDefender Core:

- Analyze Files via Web Interface
- Process Files via REST API

Product Upgrading

> **Info**
> Simply run installer file, details could be found at Installation

> **Info**
> For command line mode, upgrading (e.g. on **Linux**), prior to running the installer file, please make sure to **create the ignition file** with PostgreSQL server information included. Details:
>
> - Standalone DB mode: Command Line Installation (Windows and Linux)
> - Shared DB mode: Command Line Installation

Recommendation for MetaDefender Core upgrade

Our goal is to help you execute the product upgrade continuously, efficiently, and correctly.

PostgreSQL service upgrade

- **Bundled PostgreSQL Service**: If you are using the bundled PostgreSQL server included in the installer, the PostgreSQL service will be automatically upgraded to the newer version (if available). No additional action is required for this process.
- **Remote PostgreSQL Service**: For remote PostgreSQL servers, customers are responsible for managing and upgrading the service independently. Upgrading MetaDefender Core does not update the remote PostgreSQL service.

Before the upgrade

Prepare a proper disaster recovery plan (DRP) including some common practices:

- Backup, or take a snapshot of your current working MetaDefender Core server.
  - This is to help quickly rollback for any upgrade failure might cause later for any reason.
- Optionally backup your PostgreSQL database, especially when using remote shared database mode.
  - Reference: https://www.postgresql.org/docs/current/backup.html
  - Please do it after stopping all linked MetaDefender Core instances to avoid potential data collision.
- Lastly, if backing up entire server is not feasible, then at least export MetaDefender Core configuration package Import/Export configuration just in case you'll need to start off a new server with current MetaDefender Core configuration to keep the scanning service running without any impact.

On the PostgreSQL server, double check server resource to ensure sufficient RAM memory and disk space to be preserved for the upgrade process.

- 8GB RAM available at least during the upgrade.

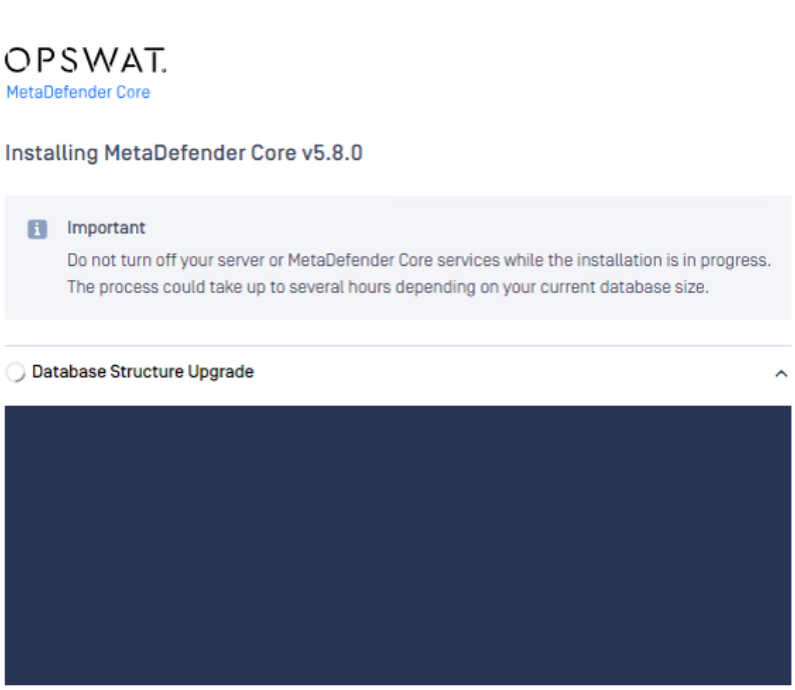
- Disk availability is greater than MetaDefender Core database size.

While doing the upgrade

Applicable for shared database deployment:

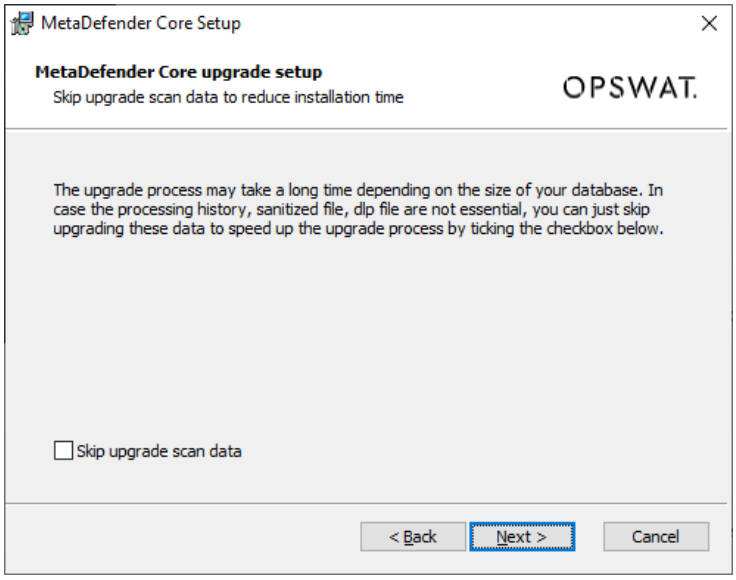
- Stop all MetaDefender Core instances' scanning service before performing the upgrade.
- Upgrade one (1) MetaDefender Core instance at a time.
- Proceed to upgrade MetaDefender Core instance where the bundle local PostgreSQL server is used for shared database setup.
- Otherwise is using remote PostgreSQL then the upgrade order does not matter.

For large size database, mostly for scan history, expecting long time to be spent on the upgrade process.



The product supports an option to skip data migration for scan history to expedite the process drastically and mitigate any failure risk. You can manually backup all data from your PostgreSQL server for offline usage if any.

Check "Skip upgrade scan data" to enable this feature while doing the upgrade.



or add following key in the ignition file:

none

```
[internal]
skip_upgrade_scan_data = 0
```

Troubleshooting

If the upgrade process ever runs into an error:

- Double check your network and services, applicable to remote PostgreSQL server.
- Optionally configure MetaDefender Core to skip migrating scan history.
  - Windows
    - Open Windows registry, and navigate to HKLM\Software\OPSWAT\Metascan
    - Add a new key `internal/skip_upgrade_scan_data` with value = 1
  - Linux
    - Open MetaDefender Core configuration file `/etc/ometascan/ometascan.conf`
    - Add new key into the file
      - [internal]
      - skip_upgrade_scan_data=1
  - Note: This key will be auto removed upon the upgrade finishes.
  - Then restart MetaDefender Core service to retry the upgrade.
    - Might retry this a few times until the upgrade is done.

Just in case the upgrade process still fails, please contact OPSWAT support with the support package - How to Create Support Package? This will help us analyze the root cause and figure out an optimal solution for your system failure.

Using external Load Balancer

Application Load Balancing (Layer 7)

This page provides information about recommended ways to use sessions with a Layer 7 load-balancer.

Most load-balancers have the ability to provide cookies for pinning subsequent traffic from a client to the appropriate server. This method is called sticky session, session persistence or session affinity. Using cookies efficiently requires the client to know when a cookie should be sent, should not be sent or should be deleted.

Sticky session load balancing

Each Metadefender Core instance has its own databases and application sessions that cannot be seen by other instances. Therefore, in order to get the related data/response to our queries we should ask the appropriate Metadefender Core v5 server. On the other hand, to keep the advantages of the used load balancing method, cookies should not be sent if it is not necessary.

Single file scanning

Step	Stage	Task	Cookie usage
1.	Sending file	Initiate processing a file on the client side. Send file through the load-balancer. (See POST File Analysis API)	Cookie should not be sent
2.	Sending file	Save the cookie and the data_id you got from the load-balancer	Save cookie
3.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Send cookie saved in step 2
4.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Send cookie saved in step 2

Batch scanning

Step	Stage	Task	Cookie usage
1.	Open batch	Initiate processing file(s) in batch. Request a batch ID through the load-balancer. (See POST Initiate Batch API)	Cookie should not be sent
2.	Open batch	Save the cookie and the batch_id you got from a Core server through the load-balancer.	Save cookie
3.	Sending files	Send file through the load-balancer. (See POST File Analysis (using batch header) API)	Send cookie saved in step 2
4.	Sending files	Save the data_id you got from the load-balancer.	-
	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Send cookie saved in step 2
5.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
6.	Getting batch status	See GET Batch Status API	Send cookie saved in step 2
7.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Send cookie saved in step 2
8.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API	Send cookie saved in step 2

If it does not matter which upstream server responds, the queries should be sent without cookie.

> **Info**
> It is recommended not to send cookies when it's not necessary to allow load-balancer to use its own method to share the load between Metadefender Core v5 servers.

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration through load-balancer

OPSWAT products that support HTTP load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	4.3.4	-
OPSWAT Client	Windows: 7.6.247.0 Mac: 10.4.243.0	-
MetaDefender Email Security	4.3.0	-
MetaDefender ICAP Server	4.3.0	-
MetaDefender Vault	1.3.0	-
MetaDefender for Secure Storage	2.0.7	-

DNS Load Balancing

Using this method is logically similar to Layer 7 load-balancing.

Briefly how it works

Client uses a domain name to send a query to a server. Client's DNS server has more "A" records for that name with different IPs. When a client resolves the server's domain name DNS server randomly chooses an IP for that name to send back. When a session is used on the application layer, client should know the IP address of the Core server that handles that specific session. In every other case, client should resolve the domain name with DNS query to let requests to be balanced between Metadefender Core servers.

Single file scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Sending file	Initiate processing a file on the client side. (See POST File Analysis API)	Use IP saved in step 2
4.	Sending file	Save the data_id got from the Core server	-
5.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Use IP saved in step 2
6.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Use IP saved in step 2

Batch scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Open batch	Initiate processing file(s) in batch. Request a batch ID. (See POST Initiate Batch API)	Use IP saved in step 2
4.	Sending files	Send file to the specific Core server with the basch ID saved in step 3. (See POST File Analysis (using batch header) API)	Use IP saved in step 2
5.	Sending files	Save the data_id you got from the Core server.	-
6.	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Use IP saved in step 2
7.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
8.	Getting batch status	See GET Batch Status API	Use IP saved in step 2
9.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Use IP saved in step 2
10.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API).	Use IP saved in step 2

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration via DNS load-balancing

OPSWAT products that support DNS load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	does not support yet	-
MetaDefender Client	does not support yet	-
MetaDefender Email Security	does not support yet	-
MetaDefender ICAP Server	does not support yet	-
MetaDefender Vault	does not support yet	-

SSL connection for PostgreSQL communication

> **Note**
> The SSL connection configuration is possible thanks to the support made by PostgreSQL. For reference: https://www.postgresql.org/docs/14/ssl-tcp.html
>
> MetaDefender Core is tested against this mode on particular version 4.21.1

Step 1: Generate SSL certificate for PostgreSQL server

> **Note**
> Following instructions are for self-sign certificate. For production environment, making sure to use the certificate signed by your trusted CA.

1) Generate a private key

bash

```
openssl genrsa -des3 -out server.key 1024
```

Sample output:

bash

```
Generating RSA private key, 1024 bit long modulus ........++++++
..............++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

Type your desired password / pass phrase to encrypt the private key

2) Remove the pass phrase to automatically start up the PostgreSQL server

bash

```
openssl rsa -in server.key -out server.key
```

> **Note**
> On Linux, make sure to set permission on the server.key file

bash

```
chmod og-rwx server.key
```

3) Create a self-signed certificate

bash

```
openssl req -new -key server.key -days 3650 -out server.crt -x509
```

You will be prompted to enter detailed information which is incorporated into your self-signed certificate request.

none

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:CA
Locality Name (eg, city) [Default City]:San Francisco
Organization Name (eg, company) [Default Company Ltd]:OPSWAT
Unit Name (eg, section) []:OPSWAT
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:dummy_email@opswat.com
```

For self-signed certificate, use the server ceriticate as the trusted root certificate:

bash

```
cp server.crt root.crt
```

Copy **server.key**, **server.crt**, **root.crt** to PostgreSQL data folder. For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data**
- Linux: **/var/lib/ometascan/pg_data/**

Step 2: Configure PostgreSQL server for SSL authentication connection

1) Create a custom config file (e.g. ssl.conf) for PostgreSQL.

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\postgres\ssl.conf**
- Linux: **/var/lib/ometascan/postgres/ssl.conf**

2) Content of ssl.conf:

none

```
ssl = on
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
```

3) Modify pg_hba.conf

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data\pg_hba.conf**
- Linux: **/var/lib/ometascan/pg_data/pg_hba.conf**

Add following (modify **::1/128** if needed to match with your environment requirement)

none

```
hostssl all all      ::1/128      scram-sha-256
```

After the modify, the content of pg_hba.conf should be like this (example):

> **Note**
> In a Linux environment, a local connection is required to upgrade from version 5.5.1 or earlier to version 5.6.0 or later.

none

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local         DATABASE  USER  METHOD  [OPTIONS]
# host          DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl     DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostgssenc    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnogssenc  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
local    all all                    scram-sha-256
hostssl all all      ::1/128      scram-sha-256

#More content.......
```

4) Reload the services

- Standalone DB mode: Restart **ometascan** service (MetaDefender Core service)
- Shared DB mode: Restart **ometascan-pg** / **ometascan-postgresql** service (MetaDefender Core PostgreSQL service) and then **ometascan** service (MetaDefender Core service)

Test your SSL database connection:

bash

```
psql.exe "sslmode=require dbname=postgres user=postgres host=localhost port=5432"
Password for user postgres:
psql (12.7)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
Type "help" for help.
```

This message indicate that you are now using SSL connection:

bash

```
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
```

Performance and Load Estimation

> **Disclaimer**
> These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Some factors that affect performance

- MetaDefender Core version
- MetaDefender Core engine package and configuration
  - set of engines (which and how many)
  - product configuration (e.g., thread pool size)
- System environment
  - server profile (CPU, RAM, hard disk)
  - client application location - remote or local
  - system caching and engine level caching
- Dataset
  - encrypted or decrypted
  - file types
    - different file types (e.g., document, image, executable)
    - archive file or compound document format files
  - file size
  - bad or unknown (assume to be clean)
- Performance tool

How test results are calculated

Performance (mainly scanning speed) is measured by throughput rather than unit speed. For example, if it takes 10 seconds to process 1 file, and it also takes 10 seconds to process 10 files, then performance is quantified as 1 second per file, rather than 10 seconds.

- total time / total number of files processed: 10 seconds / 10 files = 1 second / file.

Standalone DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.2.416.0

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

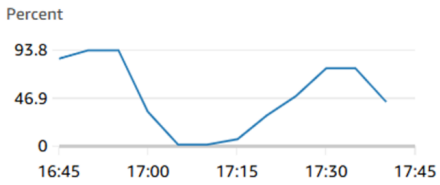
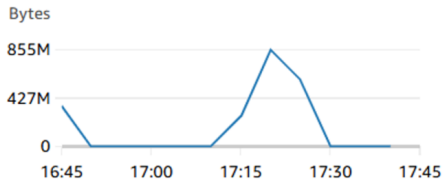
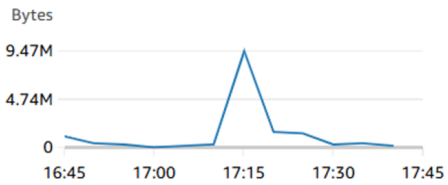
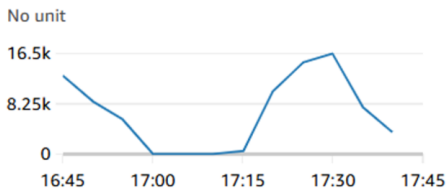
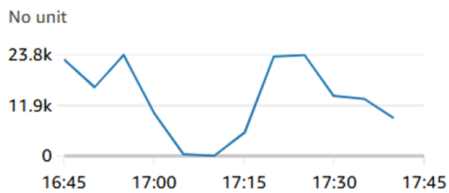
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV)	24.2	47.2 / 80	37.3 / 41.1		315,310	0.0114
Deep CDR	19.9	48.9 / 81.8	32.6 / 37		383,442	0.0094
Proactive DLP	17.1	36.8 / 58.2	30.4 / 33.6		446,228	0.0081
Vulnerability	10.1	59.2 / 85	29.3 / 33.3		755,495	0.0048

MetaDefender Core with common engine packages:

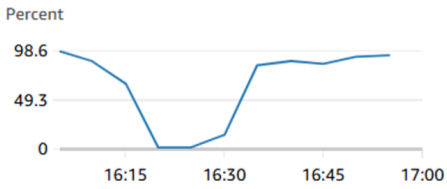
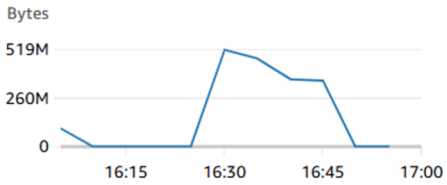
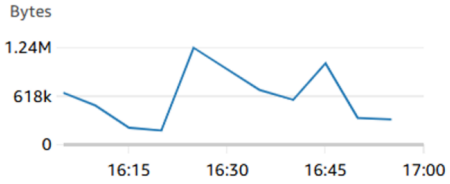
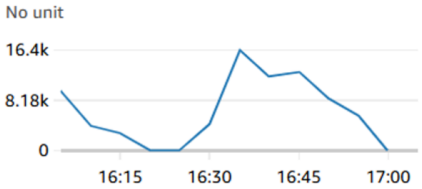
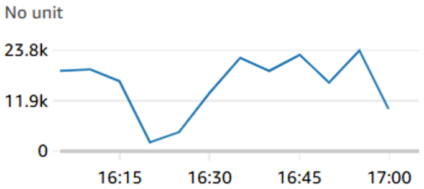
Use case	Scan duration	Avg./Max CPU usage	Avg./Max RAM usage	Avg./Max Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV) + Deep CDR	30.8	68.1 / 88	40.1 / 46.8		247,744	0.0145
Metascan (8 AV) + Deep CDR + Proactive DLP	31.2	74.2 / 98.6	41.5 / 51.1		244,567.3	0.0147
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	31.8	73.5 / 98	40.6 / 51.5		239,953	0.015

System resource utilization charts

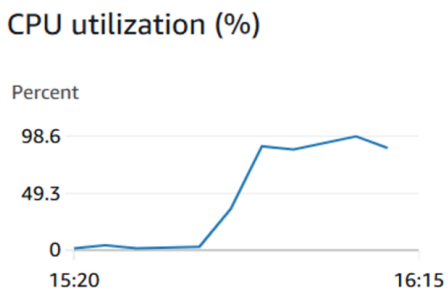
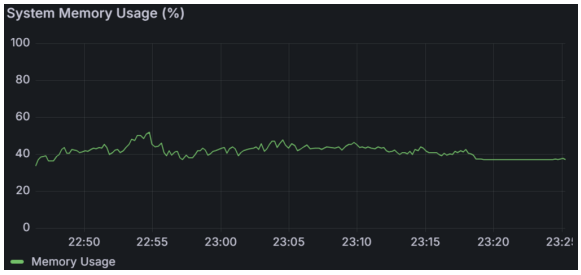
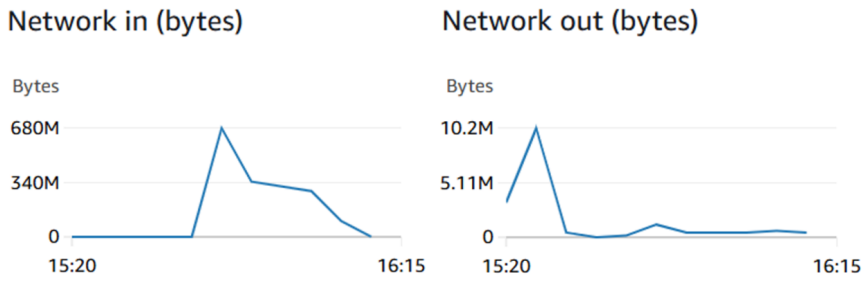
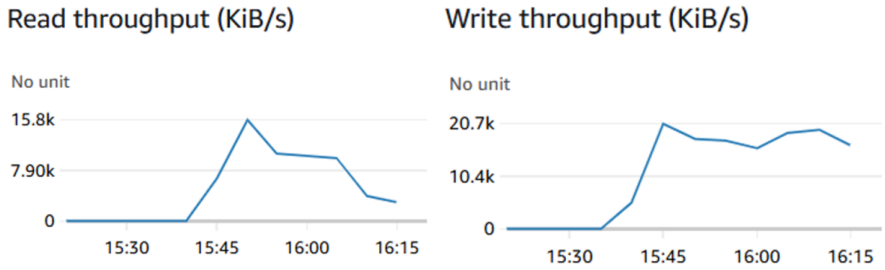
MetaDefender Core with Metascan (8 AV engines):

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

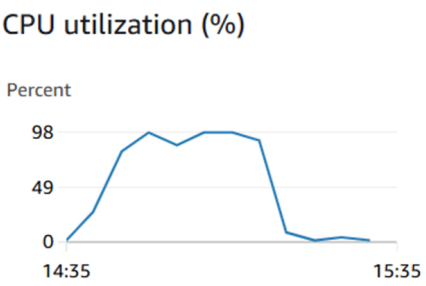
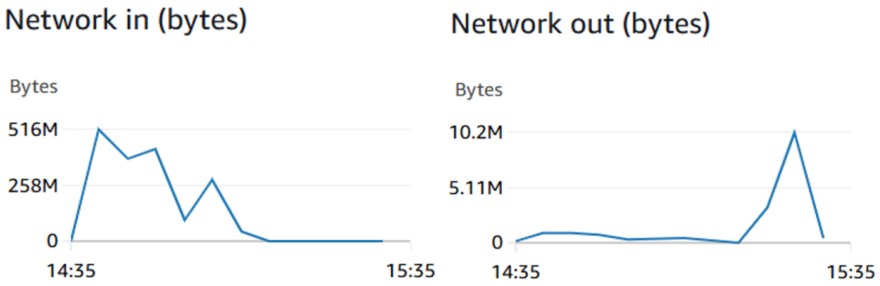
MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

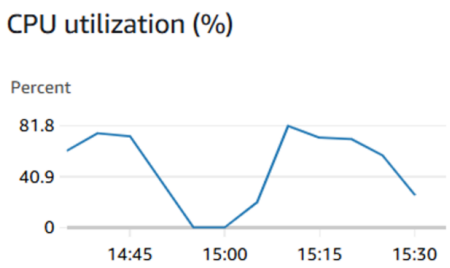
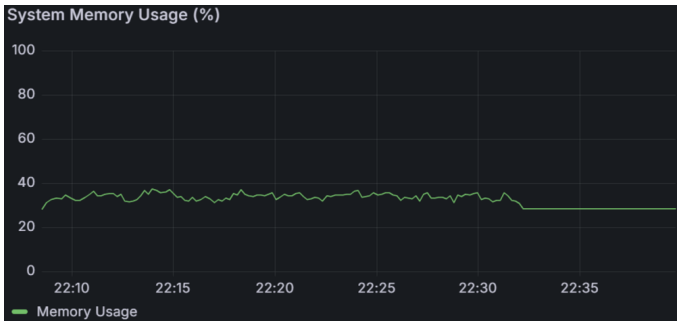
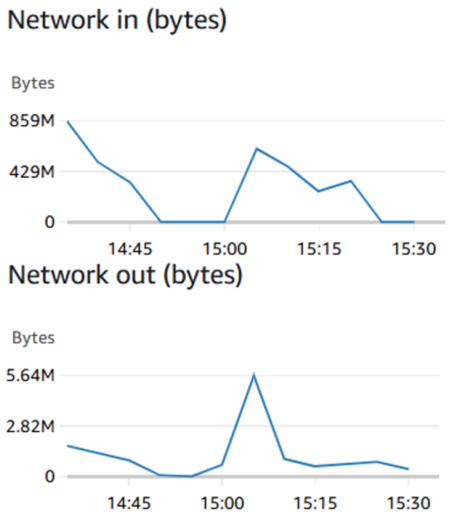
MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

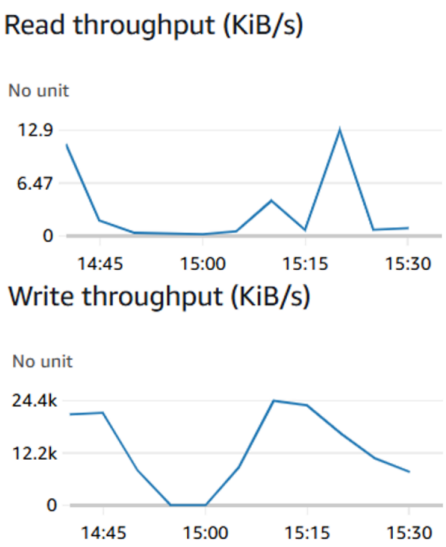
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

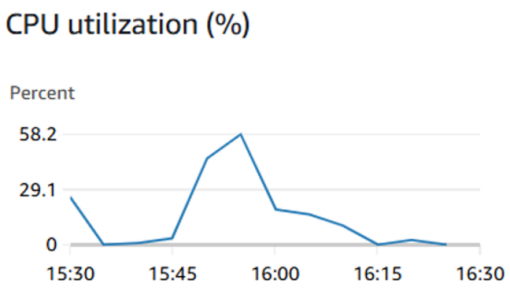
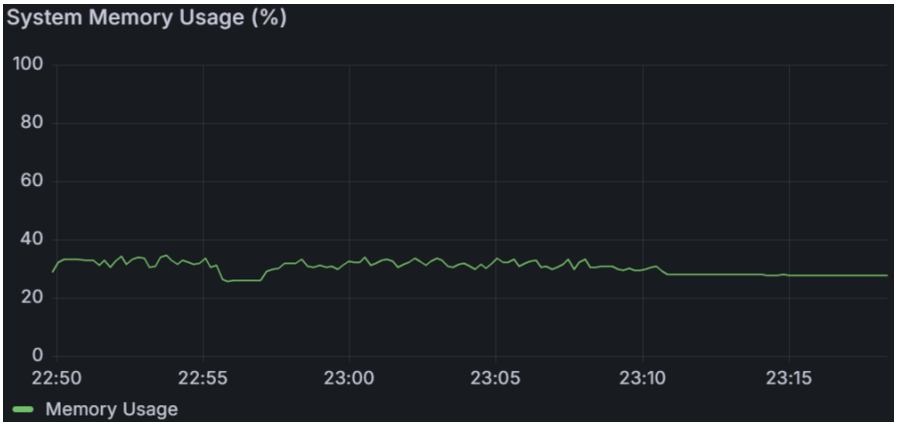
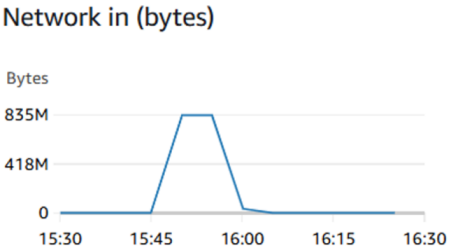
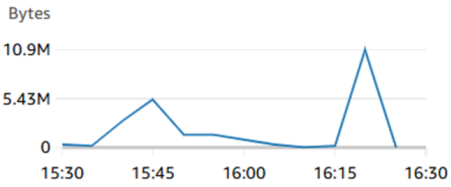
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

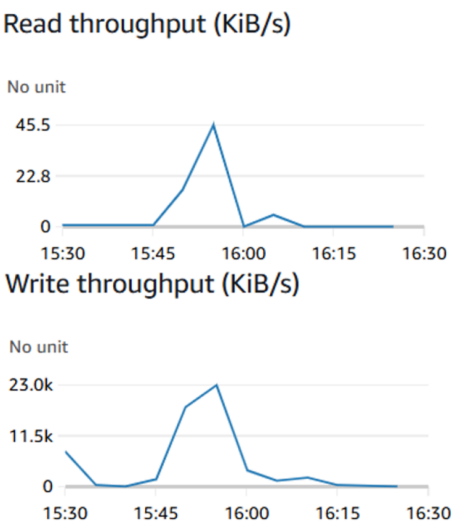
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

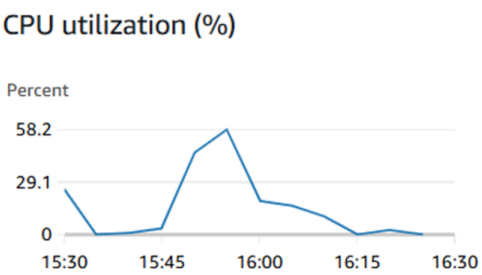
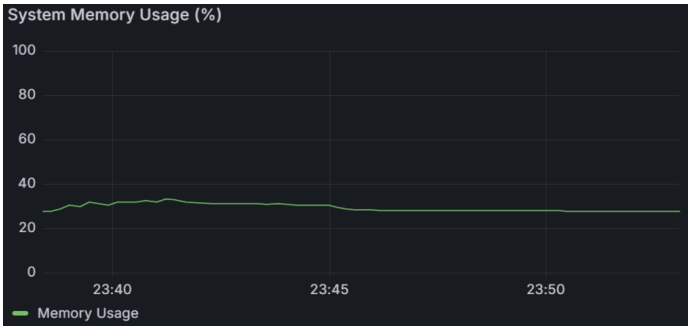
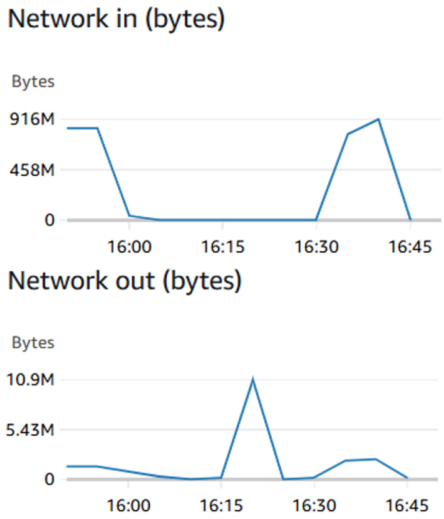
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	 

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability:

	Utilization charts
CPU usage	
Memory usage	
Network usage	

Utilization charts

Disk usage



Standalone DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Ubuntu 22.04	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.57-236

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

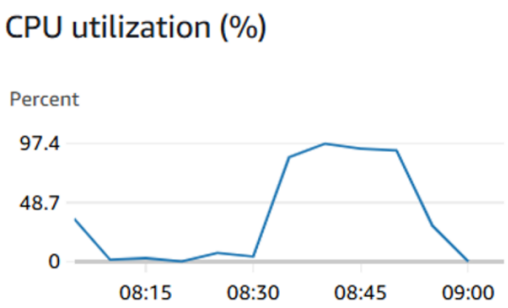
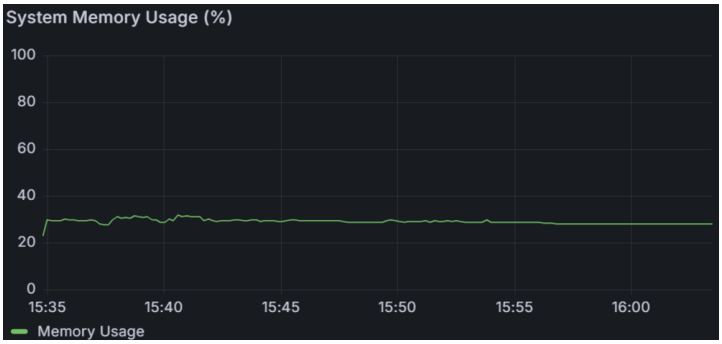
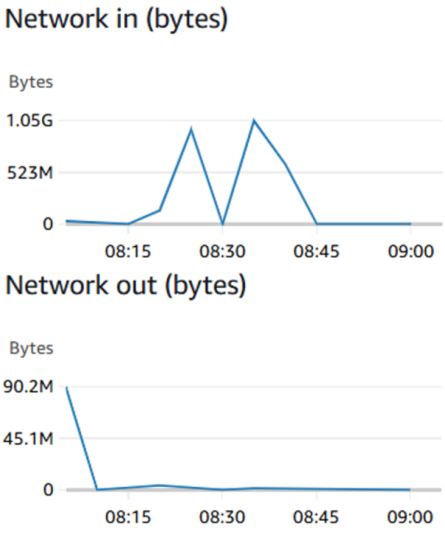
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV)	30.2	69 / 98	29 / 32		252,666	0.0142
Deep CDR	19.4	48 / 62	36 / 49		393,325	0.009
Proactive DLP	16.5	34 / 39	32 / 48		462,455	0.008
Vulnerability	12	50 / 65	46 / 46		635,875	0.006

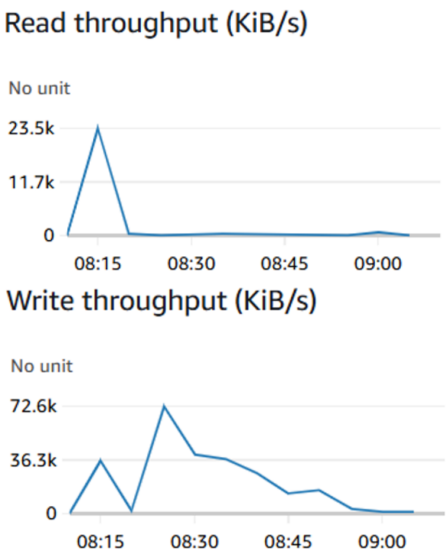
MetaDefender Core with common engine packages:

Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV) + Deep CDR	39.8	78 / 99	62 / 73		191,721	0.0188
Metascan (10 AV) + Deep CDR + Proactive DLP	43.5	61 / 84	47 / 67		175,414	0.0205
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	44.8	62 / 88	69 / 83		170,324	0.0211

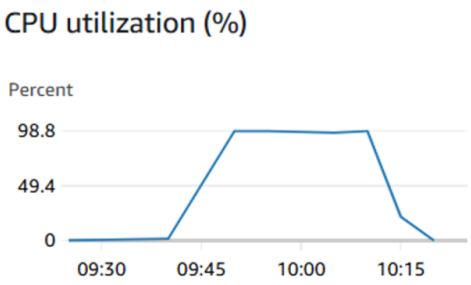
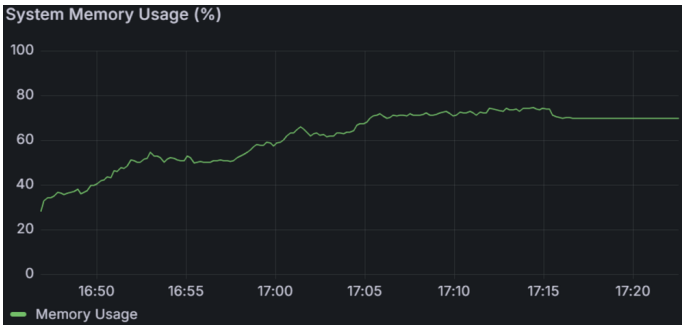
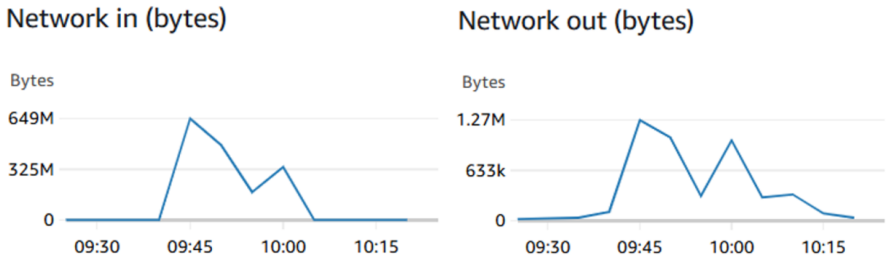
System resource utilization charts

MetaDefender Core with Metascan (10 AV engines) :

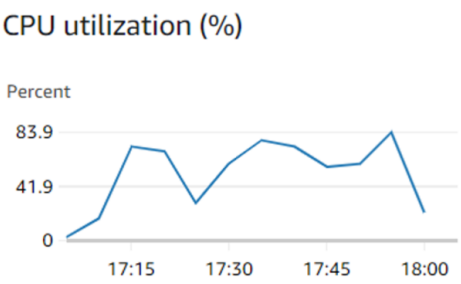
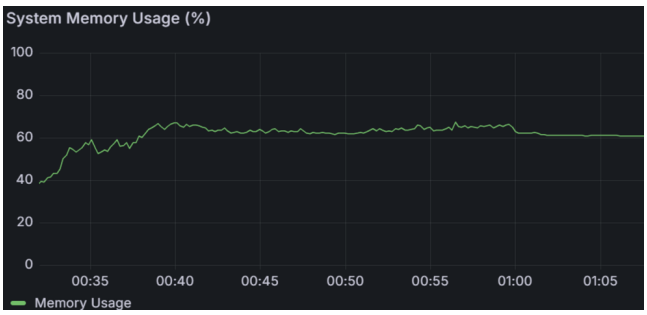
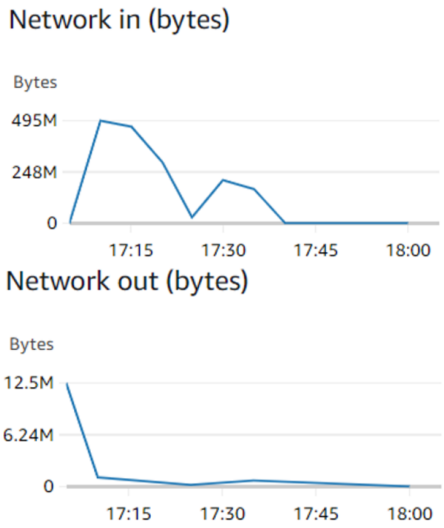
	Utilization charts
CPU usage	
Memory usage	
Network usage	

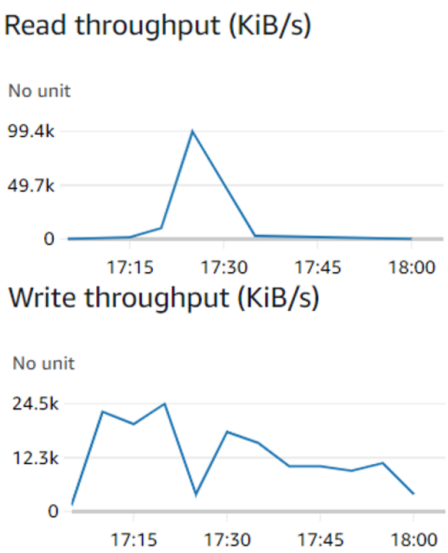
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR :

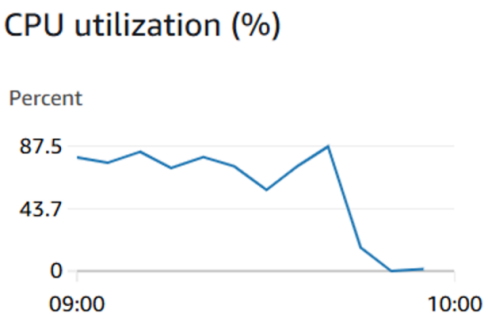
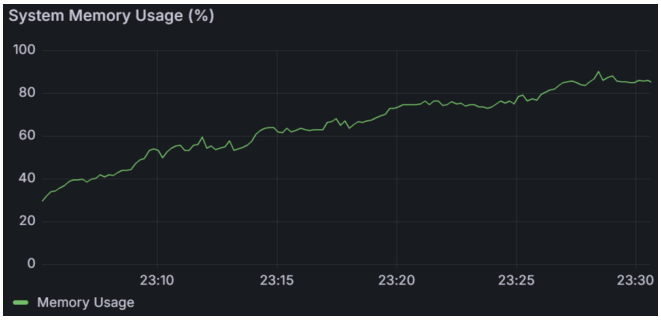
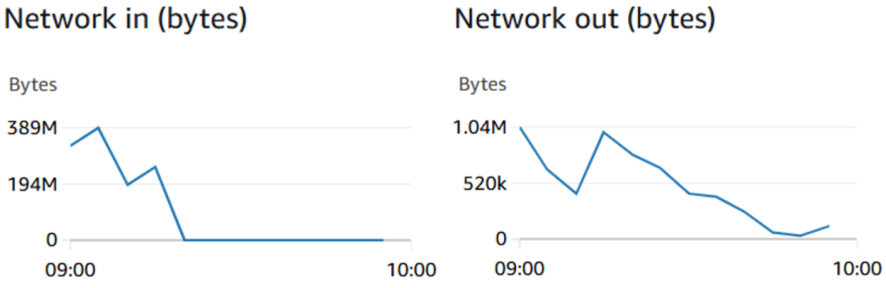
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP :

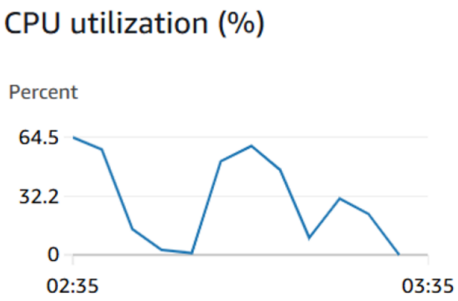
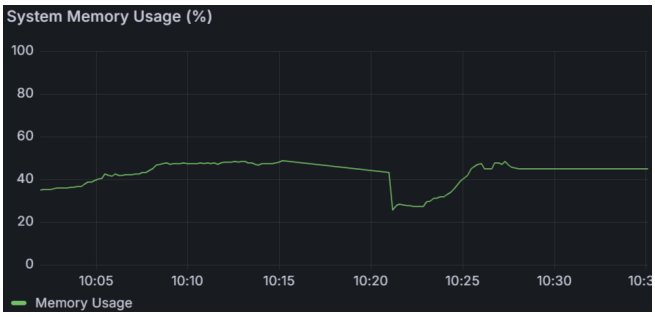
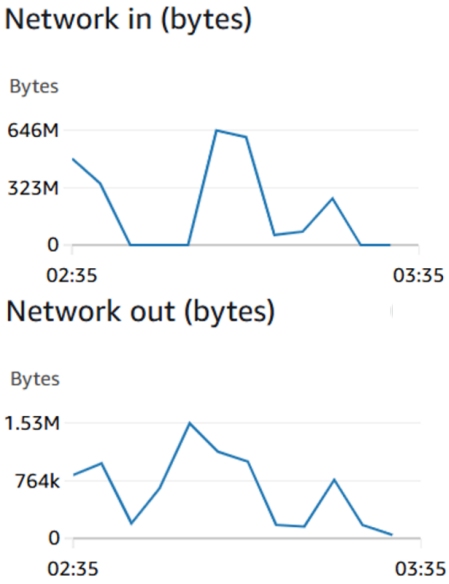
	Utilization charts
CPU usage	
Memory usage	
Network usage	

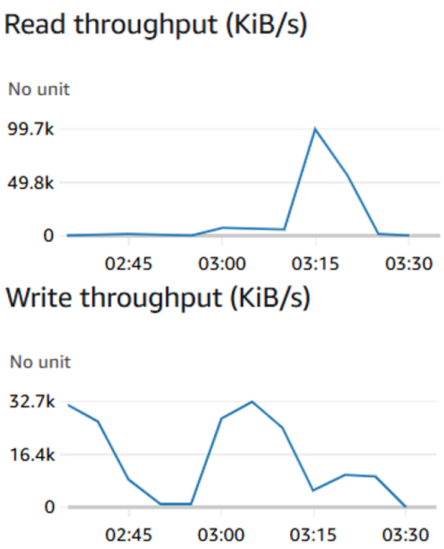
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP + Vulnerability :

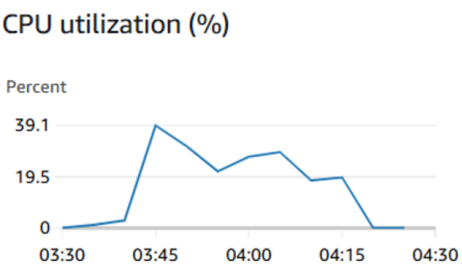
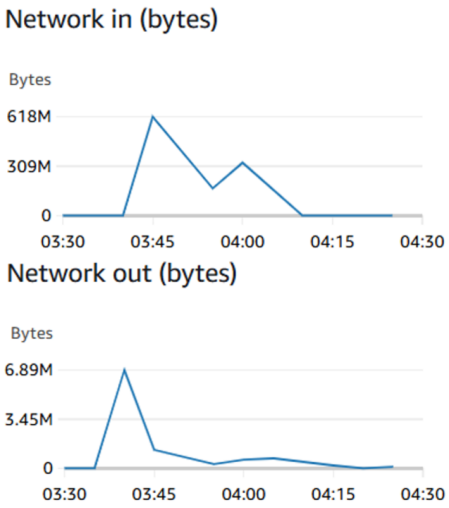
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

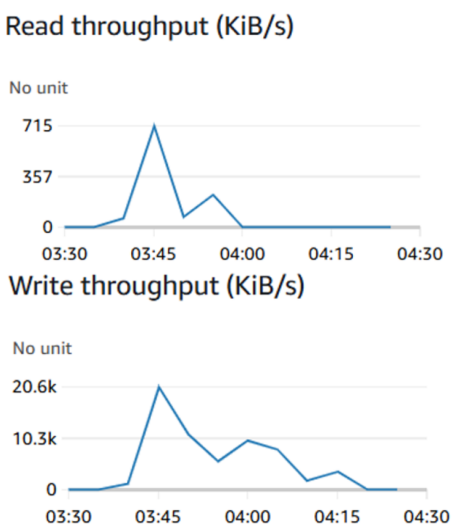
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

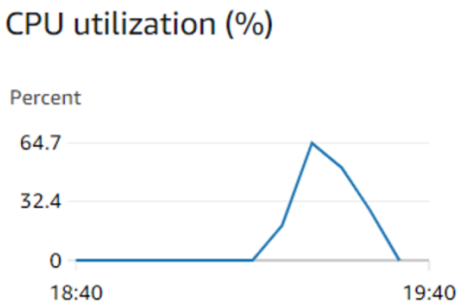
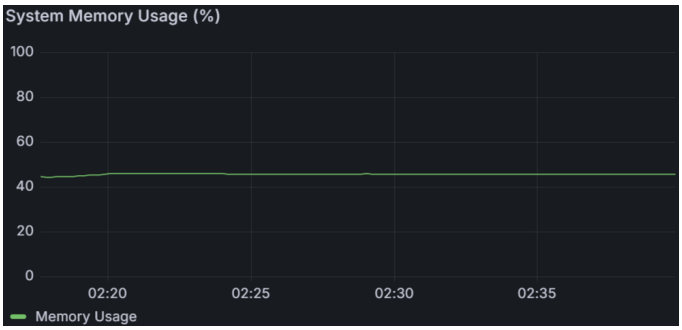
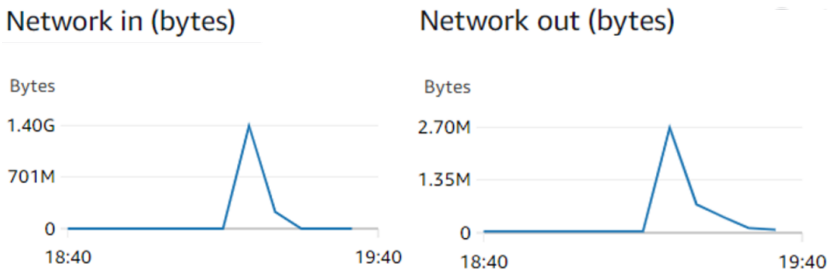
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability :

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

Shared DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

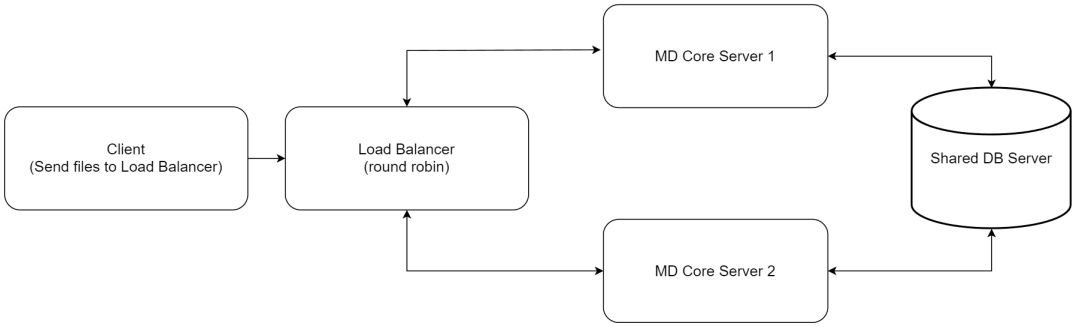
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV)	11.9	37.0	41.9	30.1	30.4	Receive: 6441.1 Send: 769.6	Receive: 6200.2 Send: 693.7	640,821.2	0.0
Deep CDR	10.5	60.1	66.7	30.3	32.2	Receive: 6379.8 Send: 594.8	Receive: 7146.1 Send: 609.6	729,368.5	0.0
Proactive DLP	7.8	38.7	37.5	29.7	31.0	Receive: 8842.6 Send: 791.8	Receive: 8635.6 Send: 729.1	965,455.7	0.0
Vulnerability	4.4	52.2	53.0	26.0	26.9	Receive: 16,215.5 Send: 1006.8	Receive: 15,785.7 Send: 997.2	1714,449.3	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV) + Deep CDR	15.4	72.4	75	34.3	35.4	Receive: 5436.6 Send: 611.3	Receive: 5589.3 Send: 607.2	495,697.4	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP	16.9	77.2	80.2	36.4	37.9	Receive: 5254 Send: 585.4	Receive: 5041.3 Send: 576.1	450,657.6	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	18.0	72.1	80.4	38.6	38.0	Receive: 4504.3 Send: 574.1	Receive: 4649.5 Send: 571.1	423,010	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

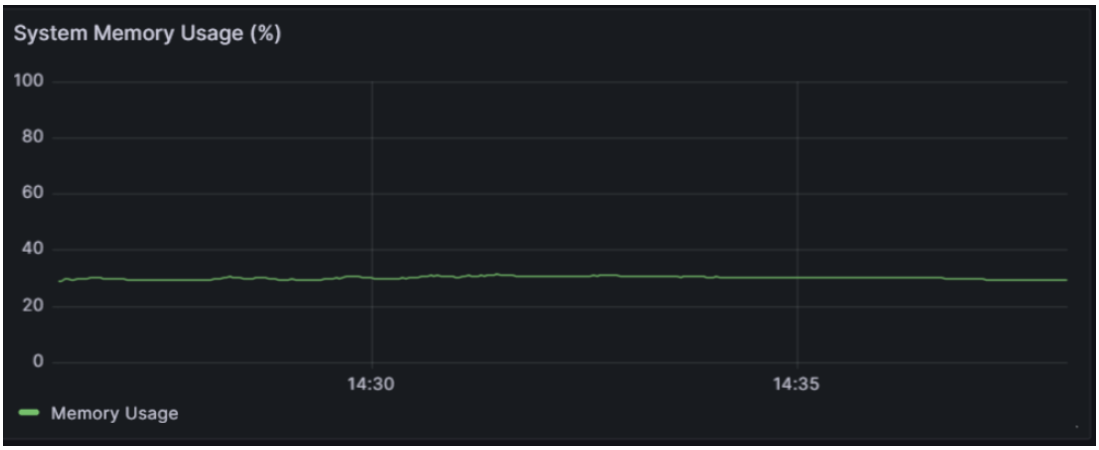
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2

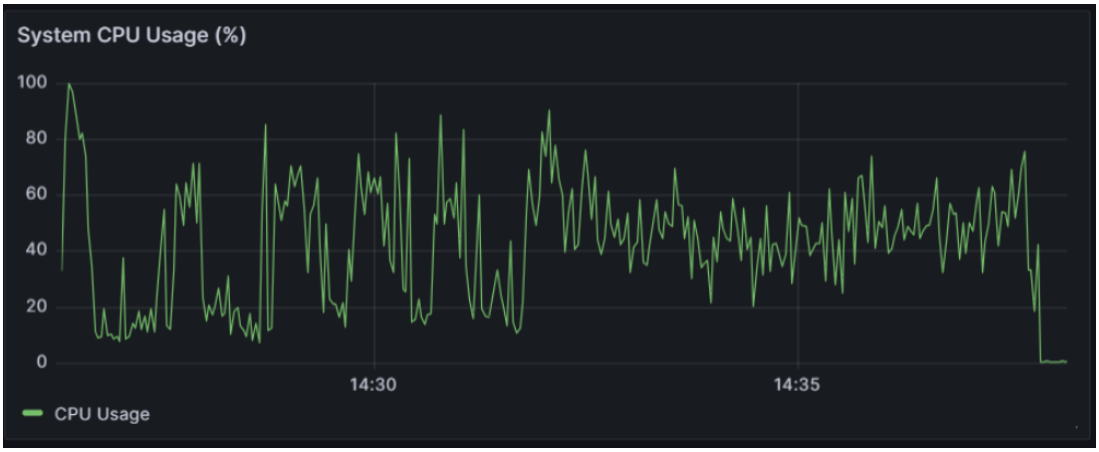


CPU usage

MD Core #1

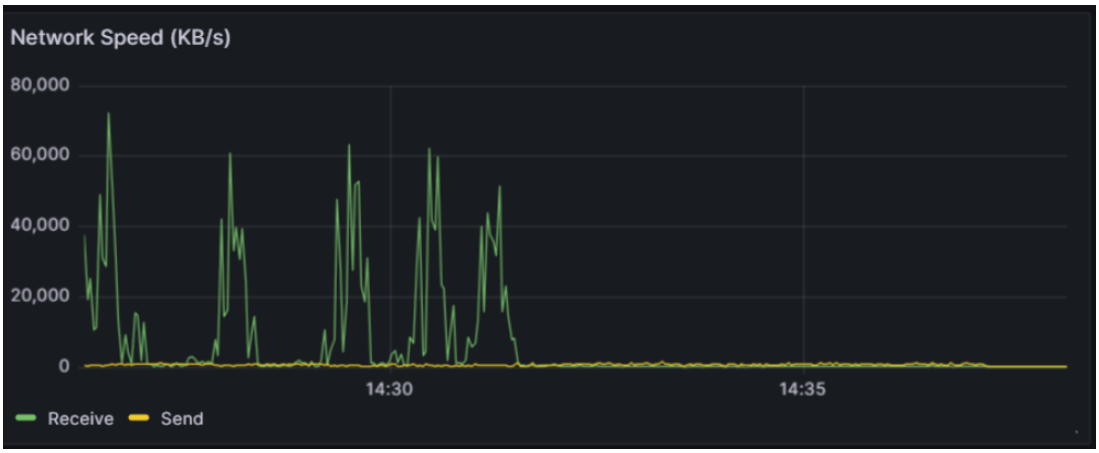


MD Core #2

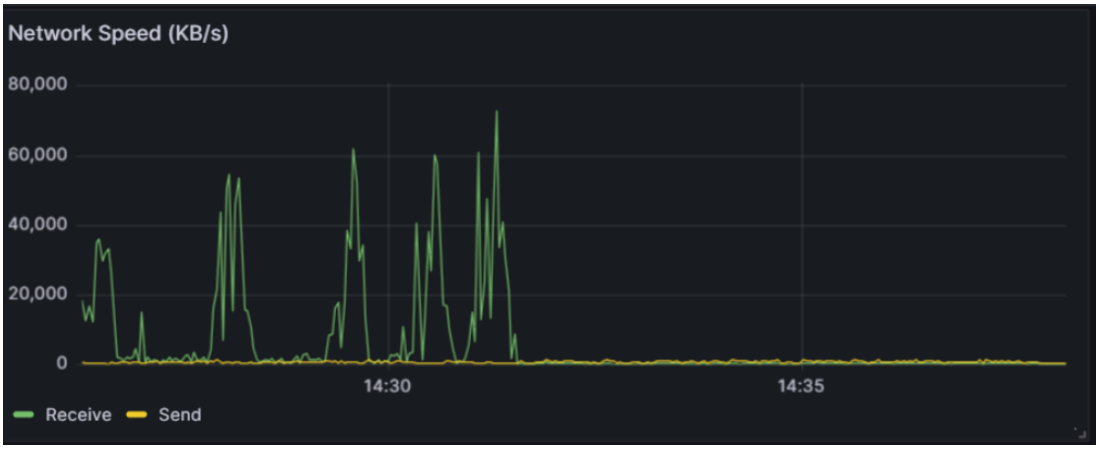


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2

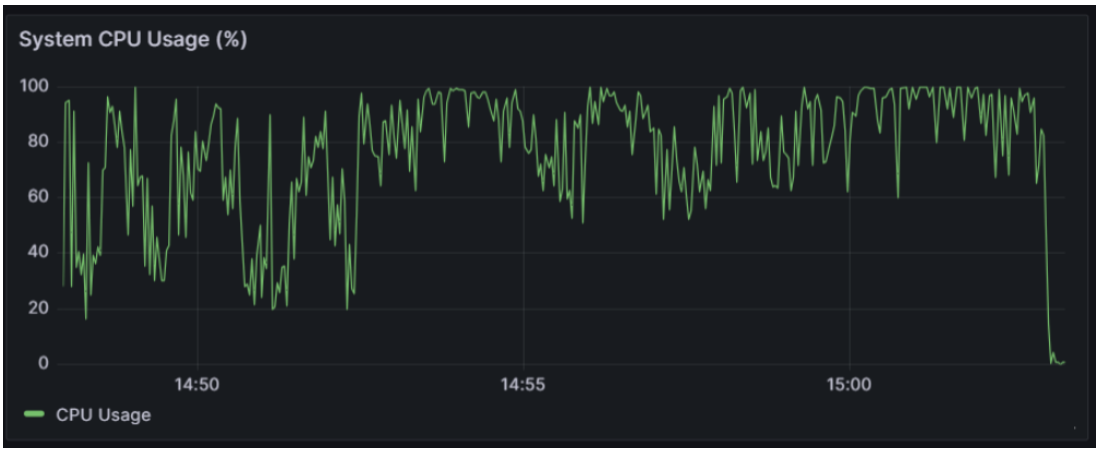


CPU usage

MD Core #1

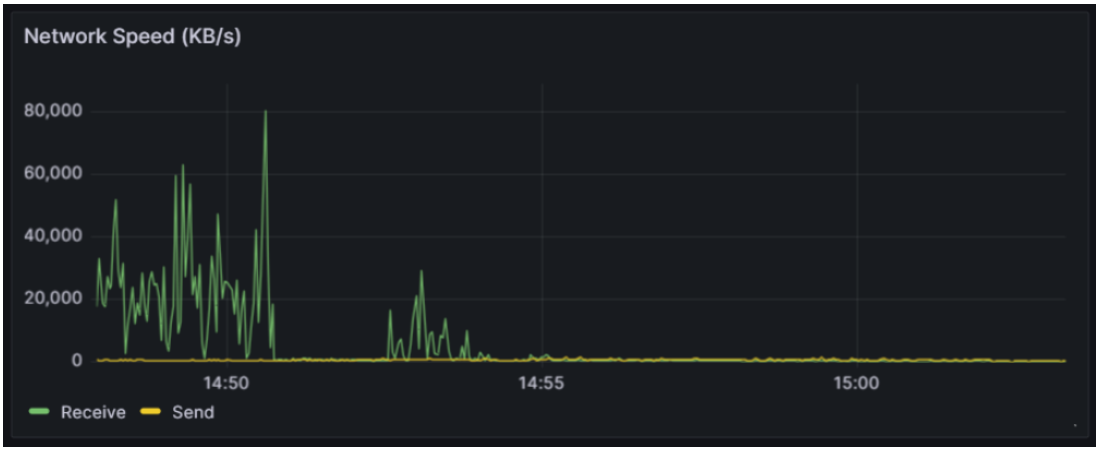


MD Core #2



Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

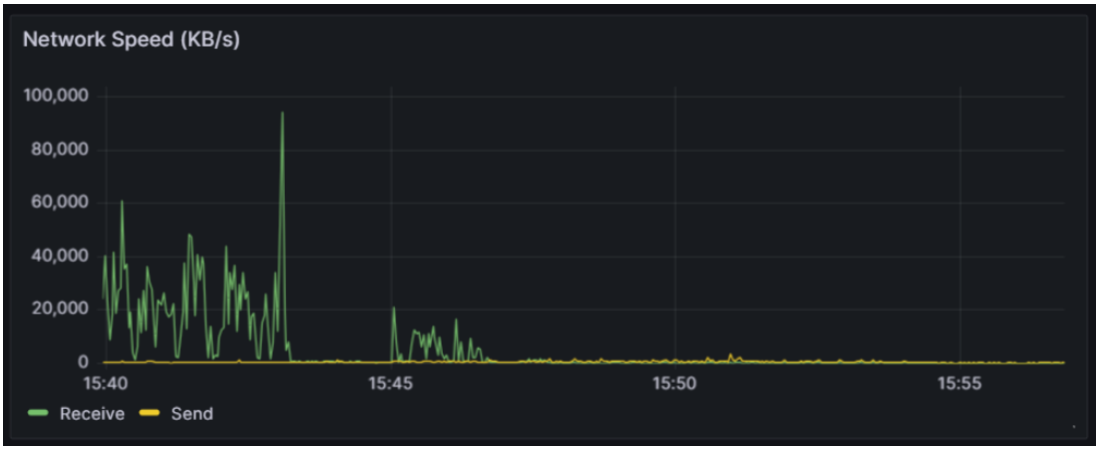


MD Core #2

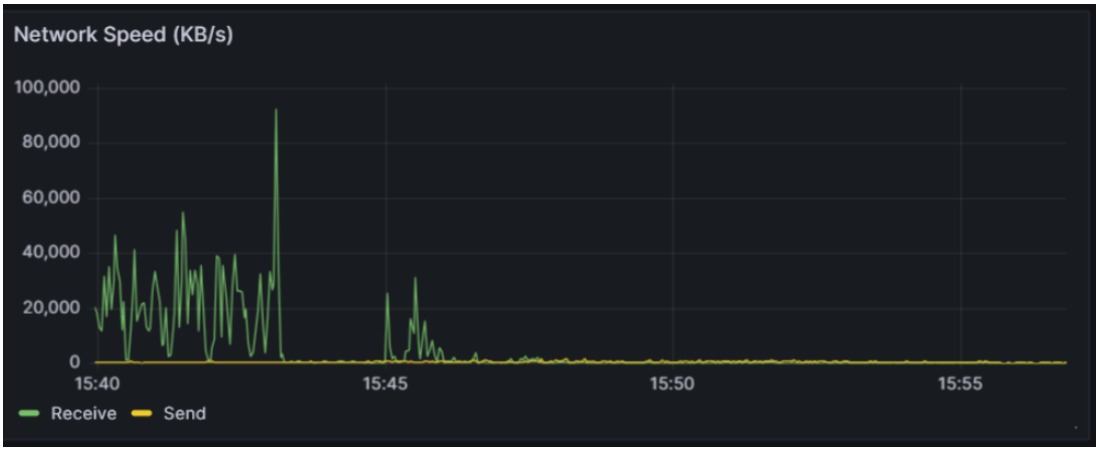


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1

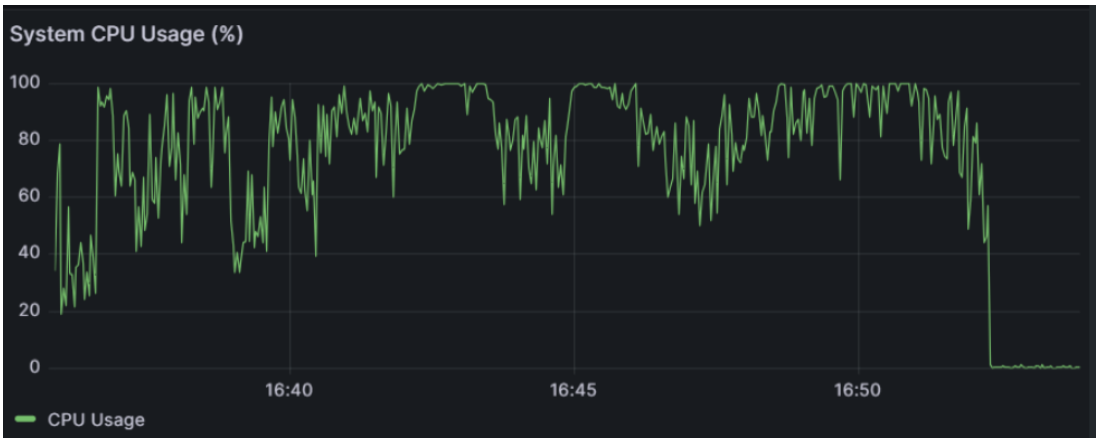


MD Core #2



CPU usage

MD Core #1

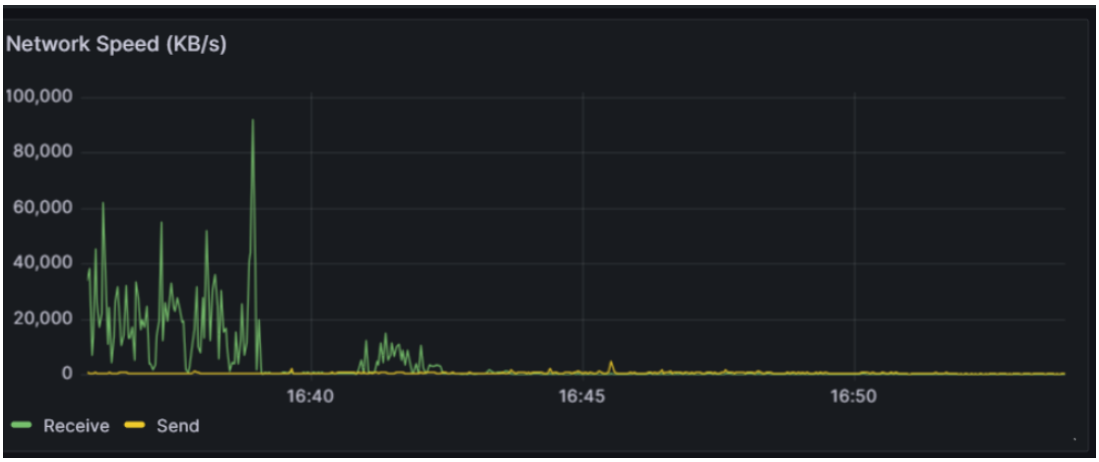


MD Core #2

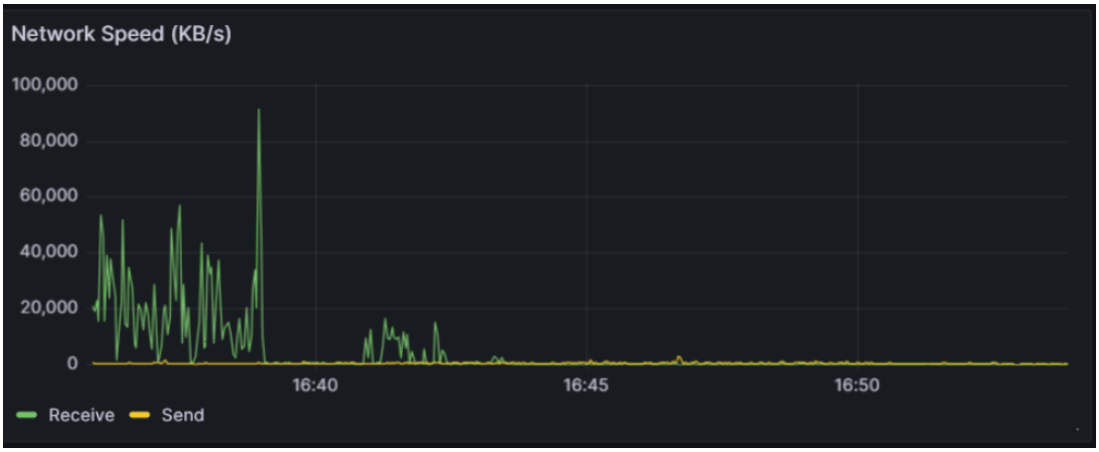


Network usage

MD Core #1



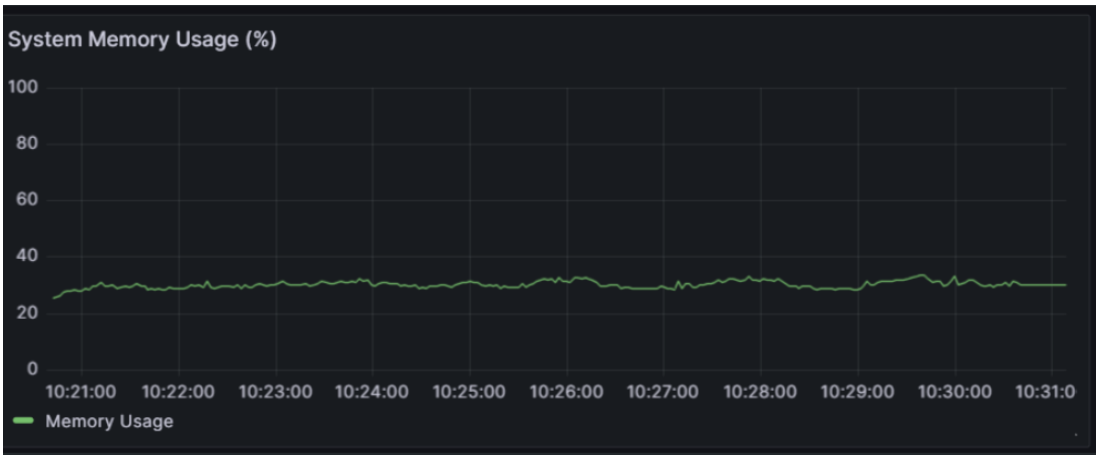
MD Core #2



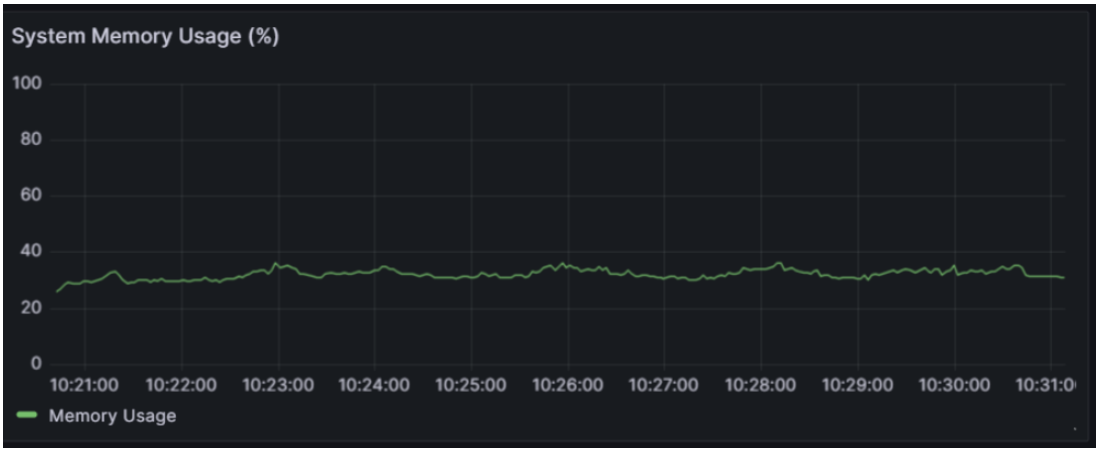
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

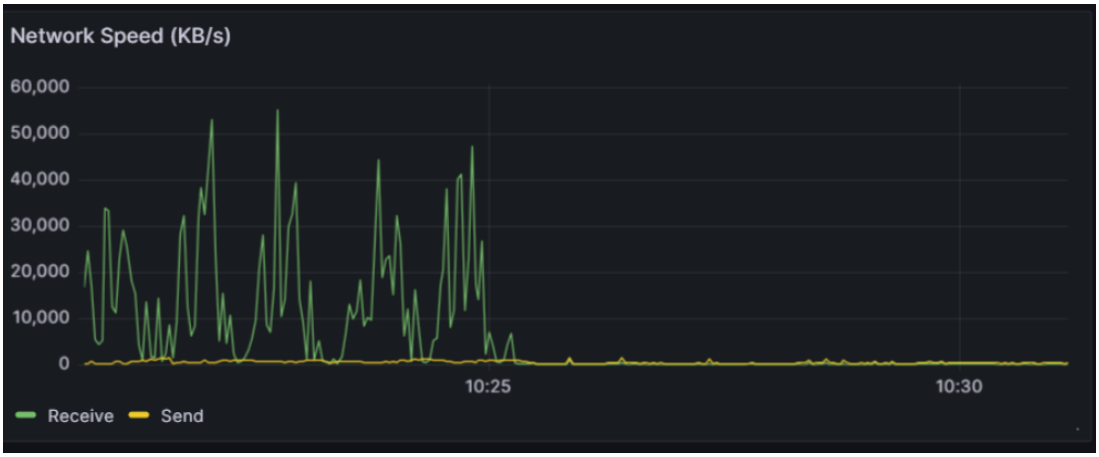


MD Core #2

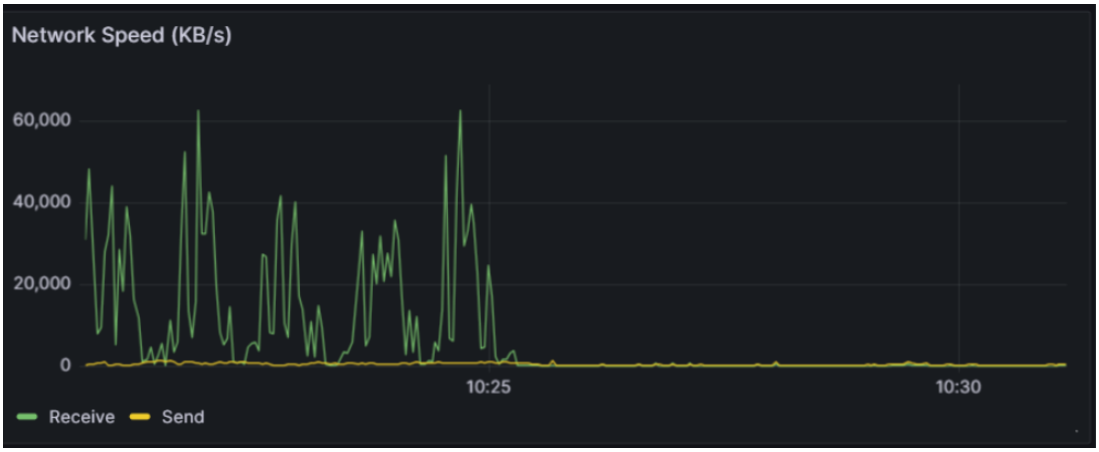


Network usage

MD Core #1



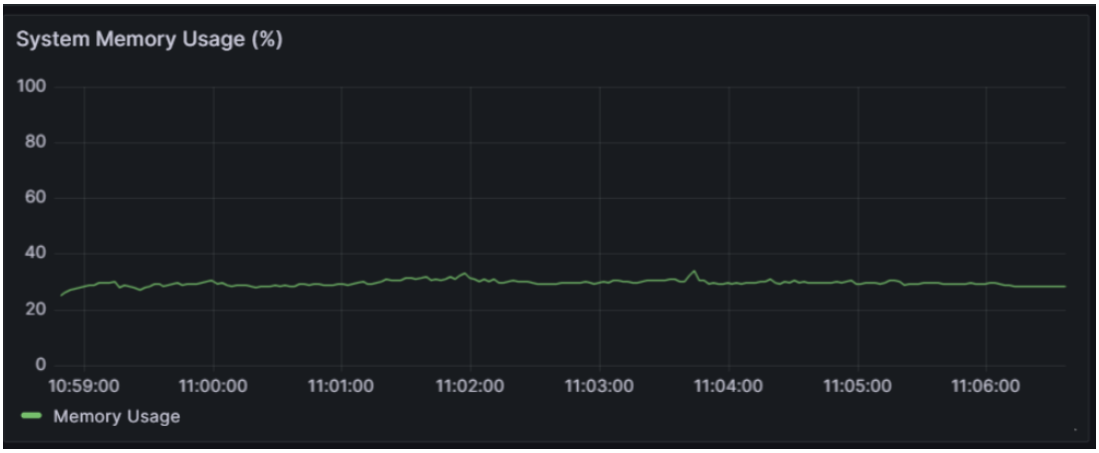
MD Core #2



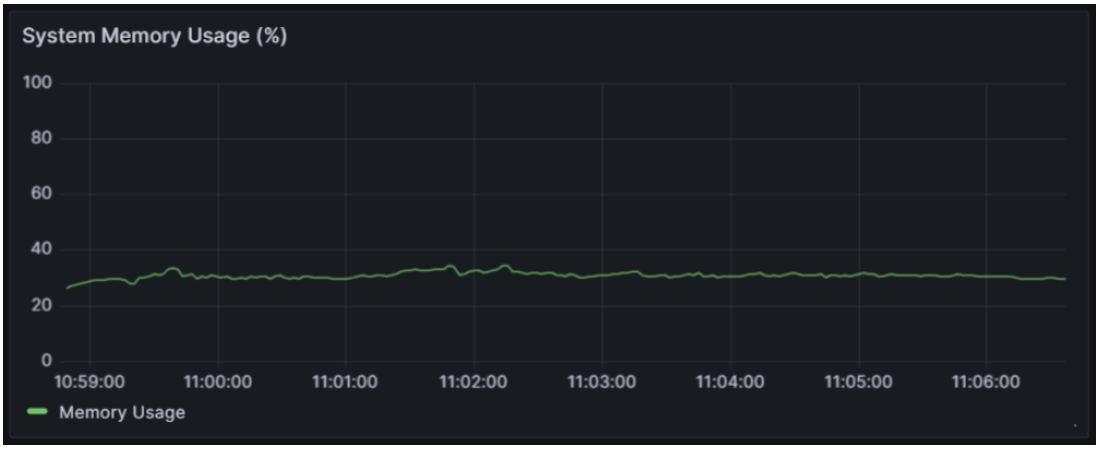
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

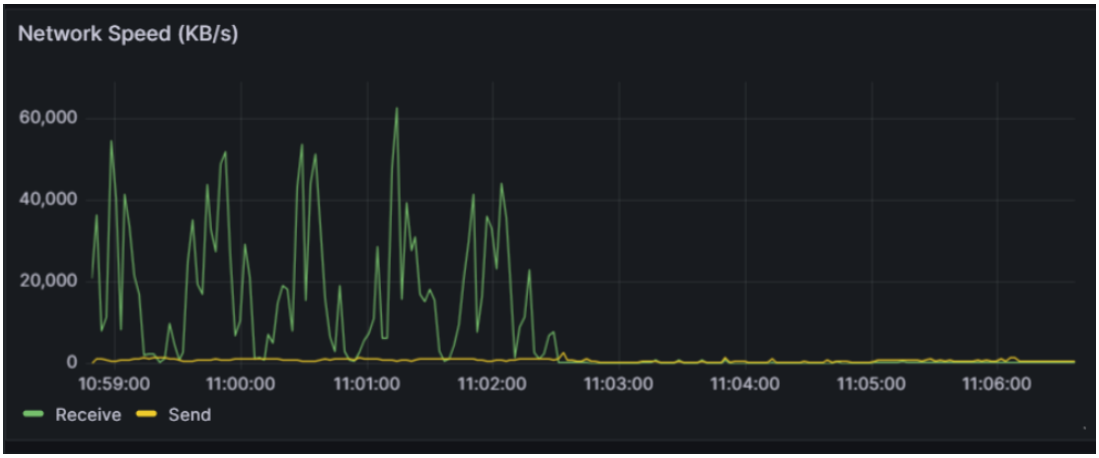


MD Core #2

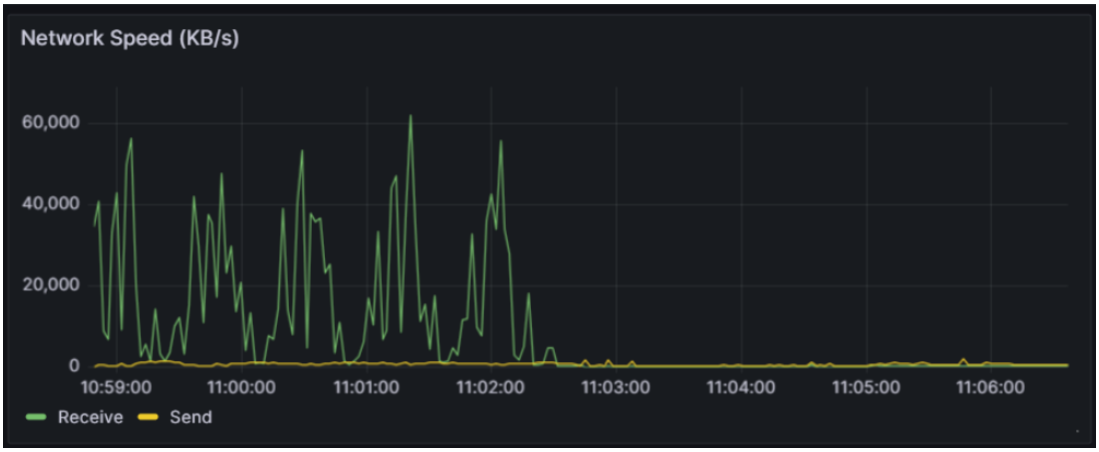


Network usage

MD Core #1



MD Core #2



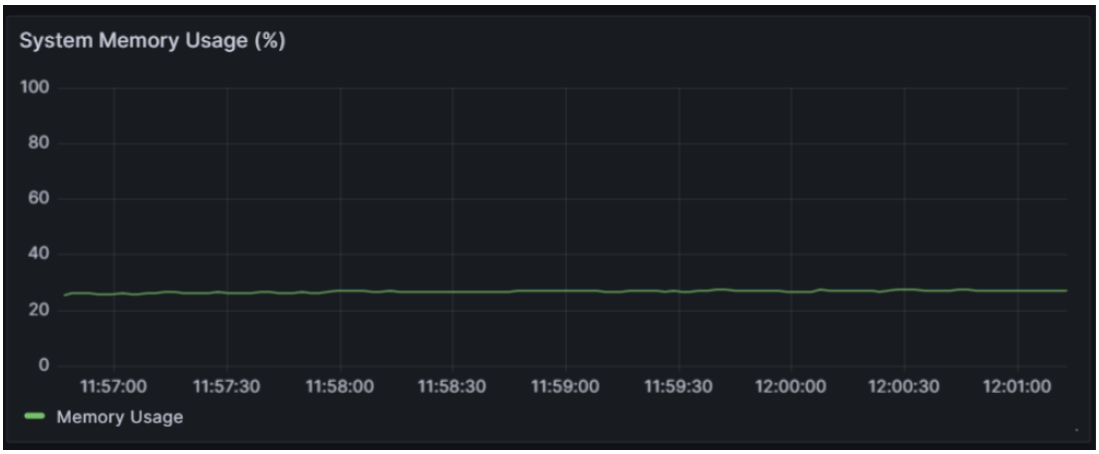
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

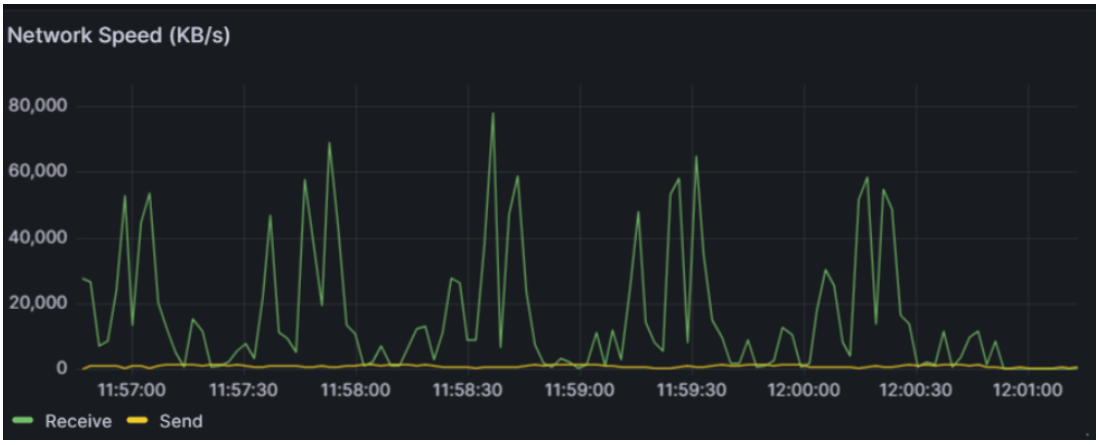


MD Core #2

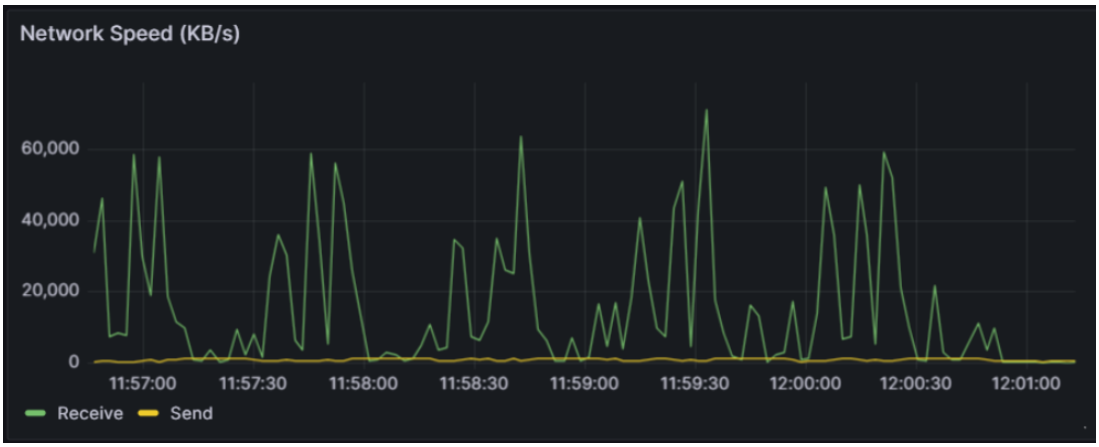


Network usage

MD Core #1



MD Core #2



Shared DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

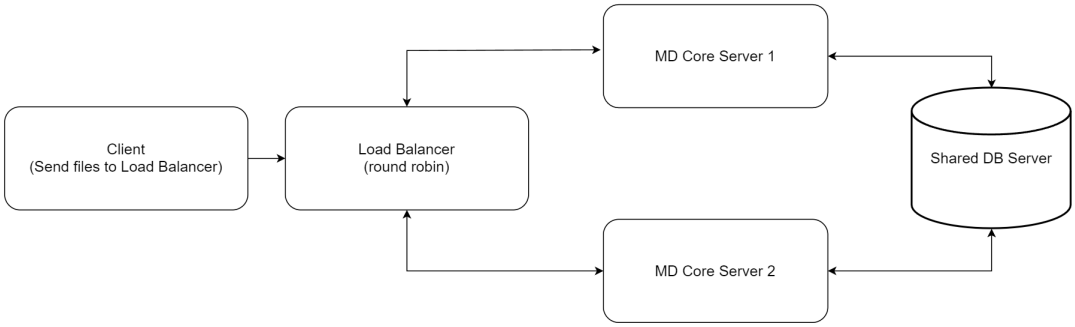
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV)	15.5	63.1	73.3	25.2	25.8	Receive: 4555.9 Send: 694.4	Receive: 4516.2 Send: 692.4	492,620.5	0.0
Deep CDR	9.70	50.7	51.2	31.9	32.5	Receive: 7062.9 Send: 623.2	Receive: 6653 Send: 654.1	786,189.6	0.0
Proactive DLP	7.2	35.2	35.0	27.9	27.5	Receive: 9047.4 Send: 802.4	Receive: 9859.8 Send: 787.5	1,055,110.8	0.0
Vulnerability	5.0	45.7	49.3	24.4	24.5	Receive: 13,585.7 Send: 966.7	Receive: 13,505.8 Send: 987.4	1,519,562.1	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV) + Deep CDR	19.9	79.1	77.6	34.0	34.2	Receive: 3785.9 Send: 573.8	Receive: 3663.1 Send: 577.7	382,716	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP	21.6	80.3	81.5	37.7	44.4	Receive: 3305.6 Send: 543.3	Receive: 3414.3 Send: 556.2	353,694.9	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	22.5	84.2	79.8	37.9	44.7	Receive: 3267.9 Send: 554.9	Receive: 3398.2 Send: 553.8	338,507.6	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

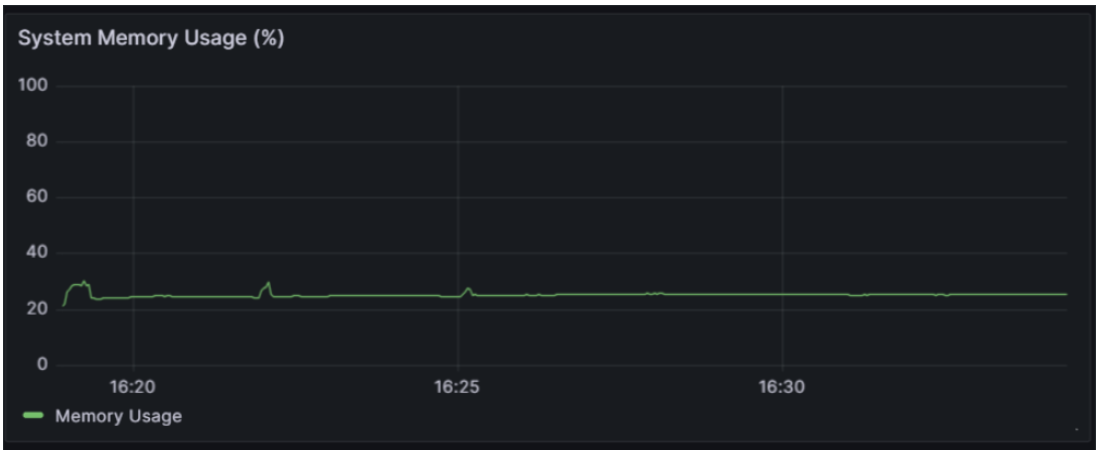
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

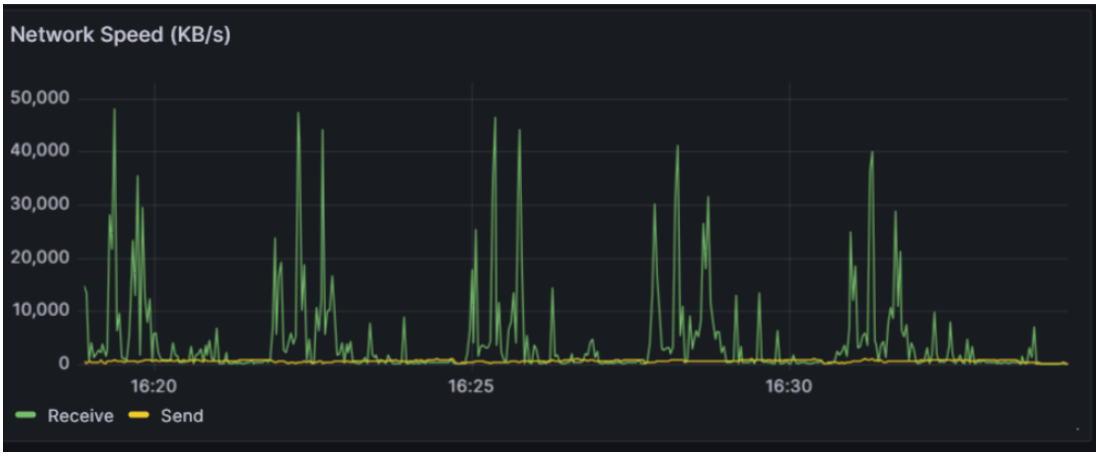


MD Core #2

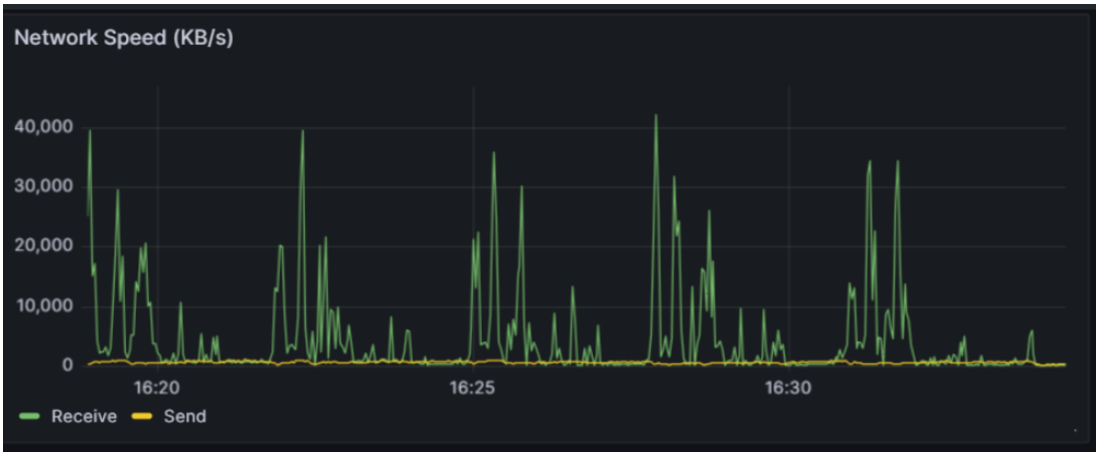


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

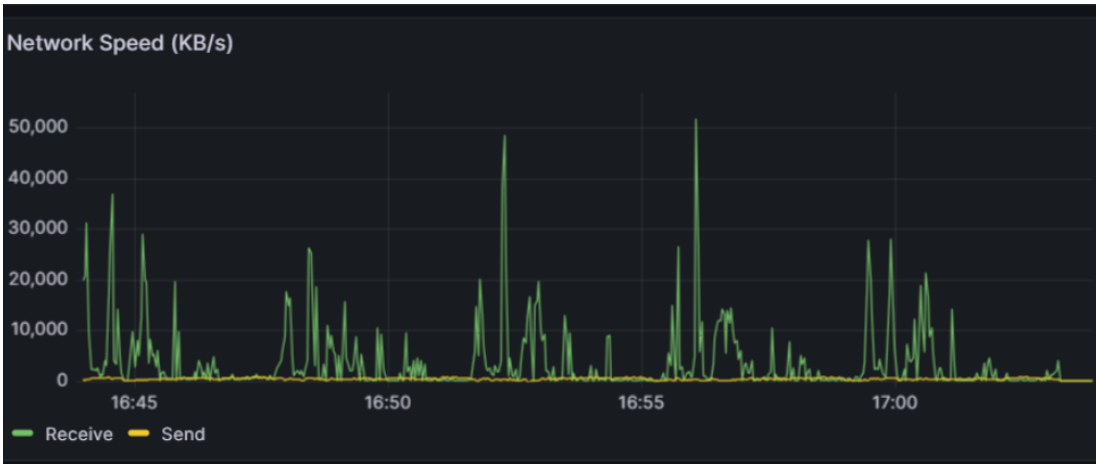


MD Core #2

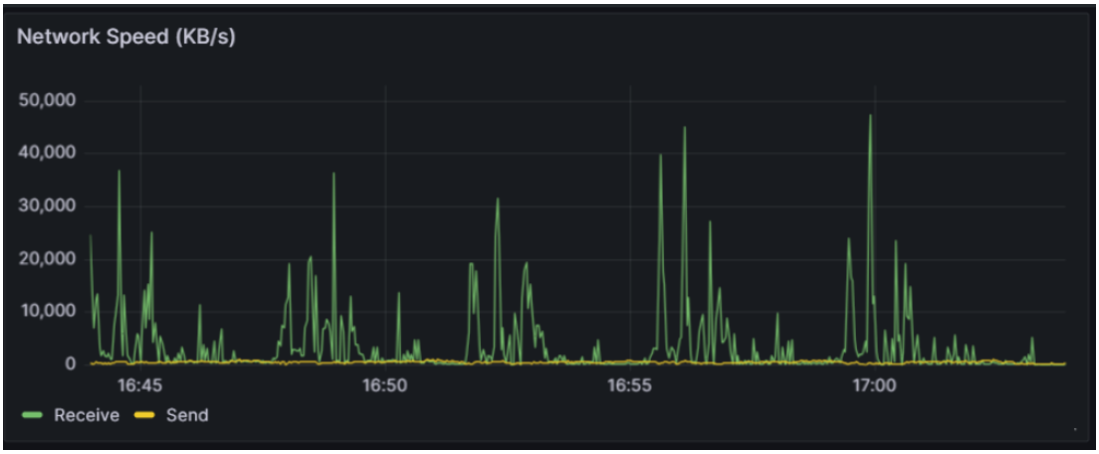


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1

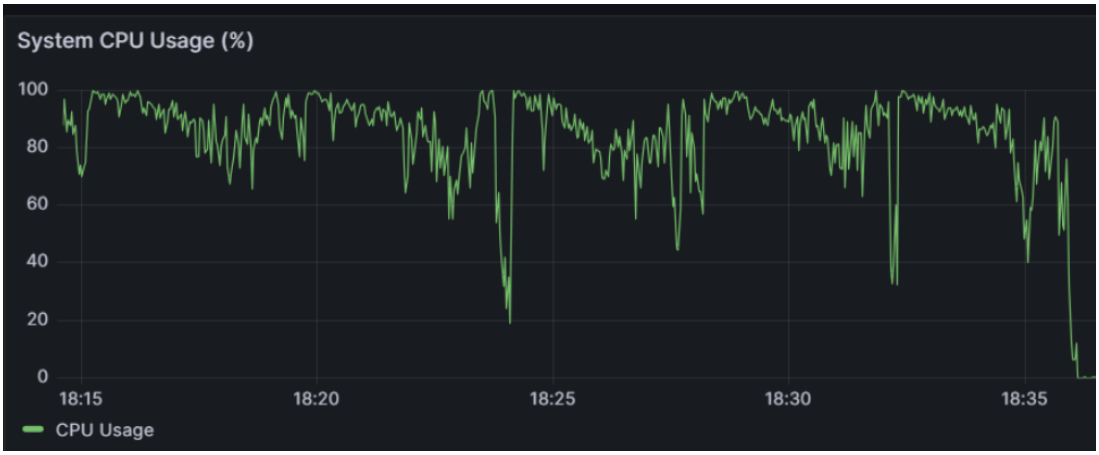


MD Core #2



CPU usage

MD Core #1

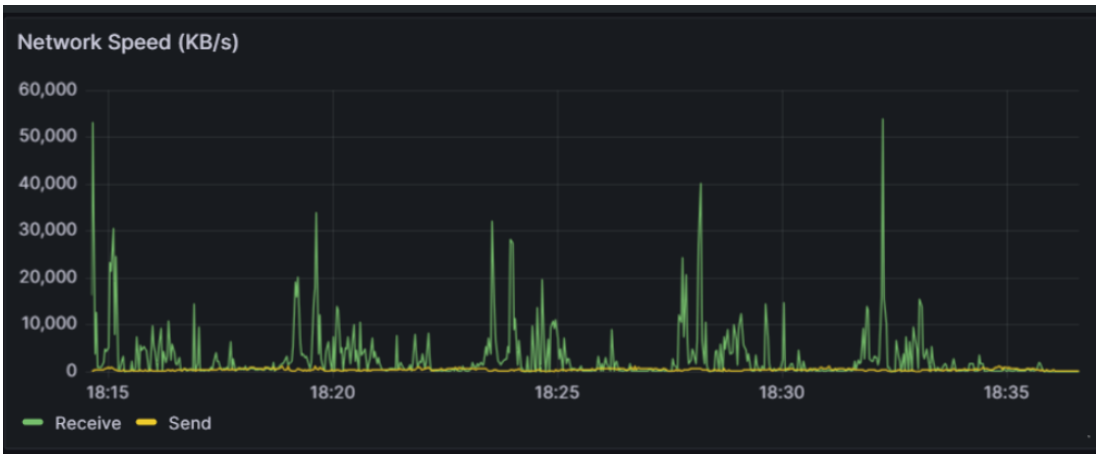


MD Core #2

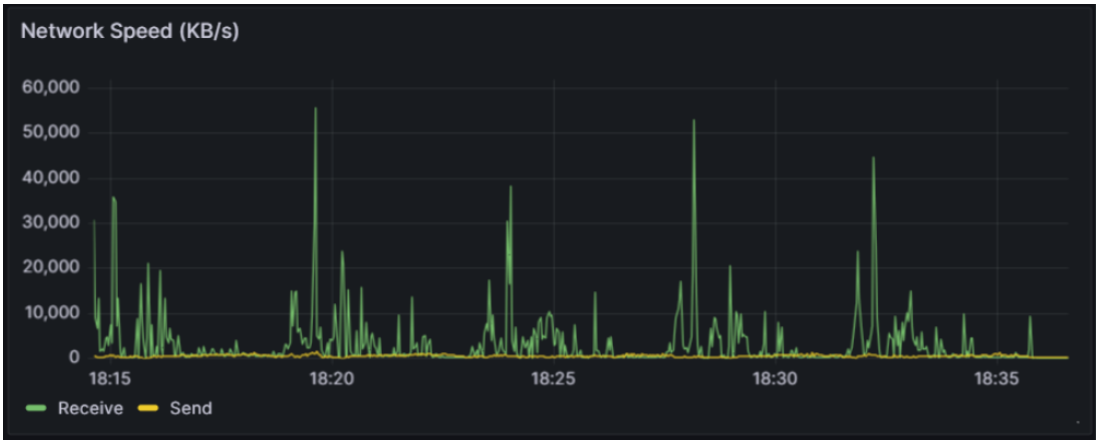


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

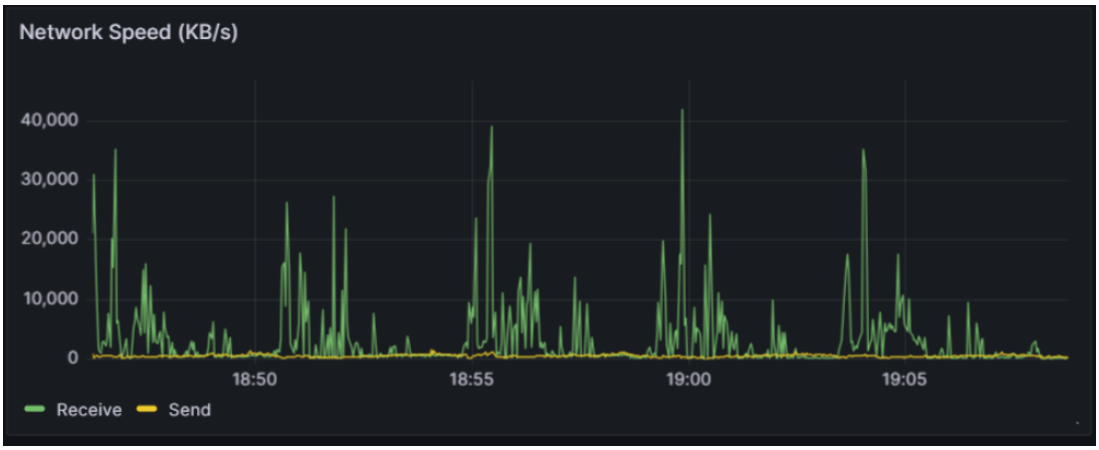


MD Core #2

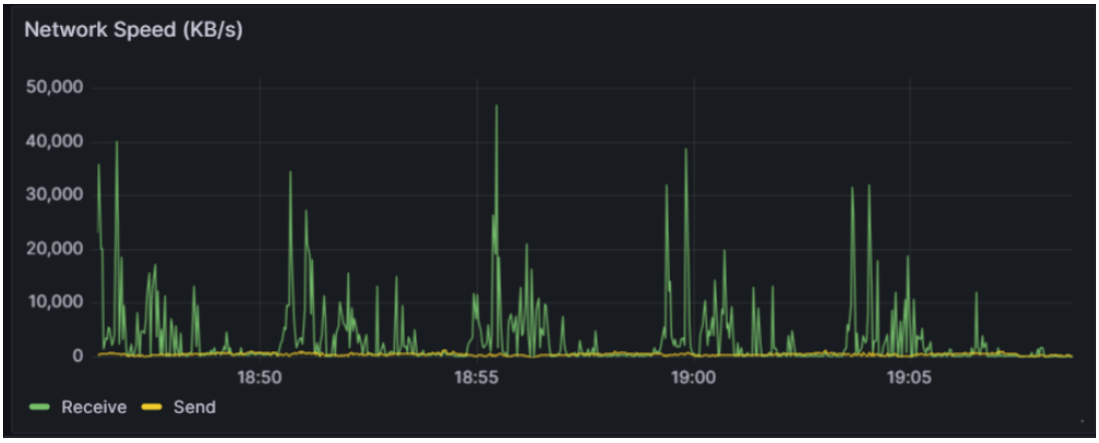


Network usage

MD Core #1



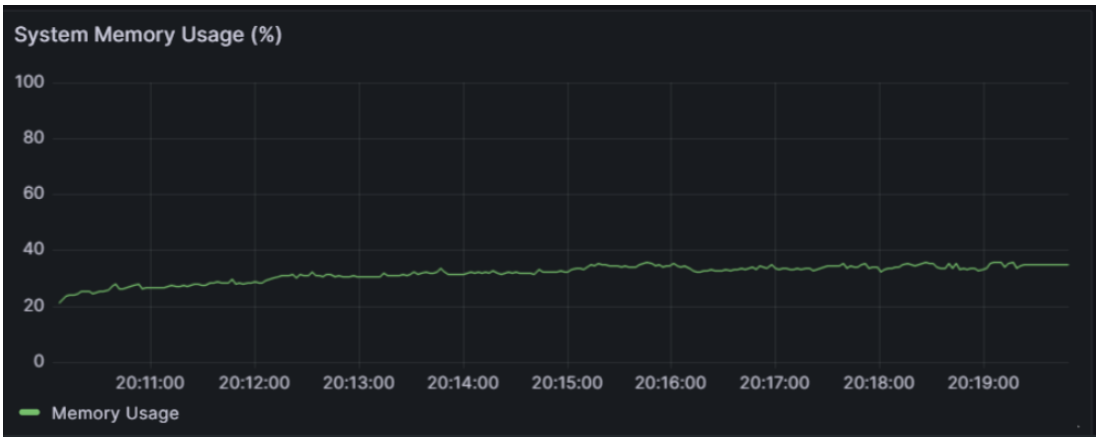
MD Core #2



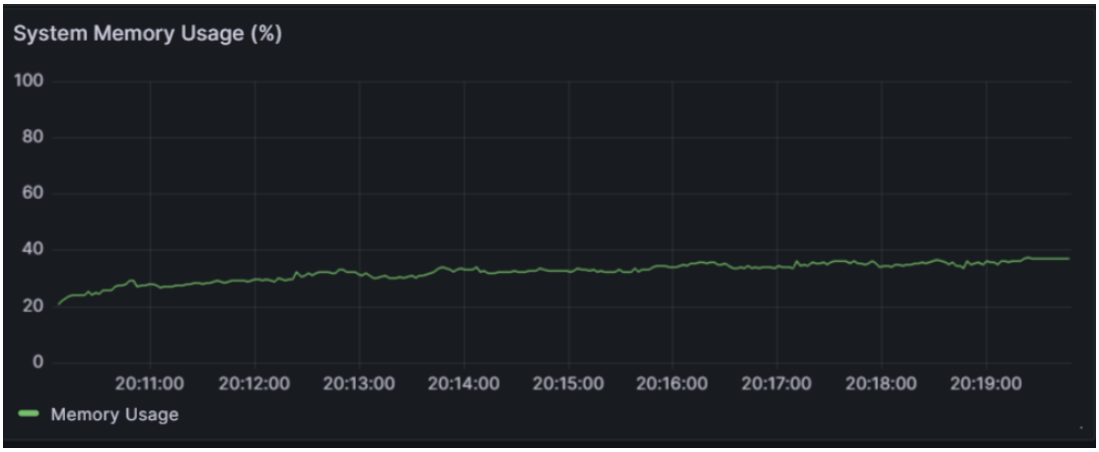
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1

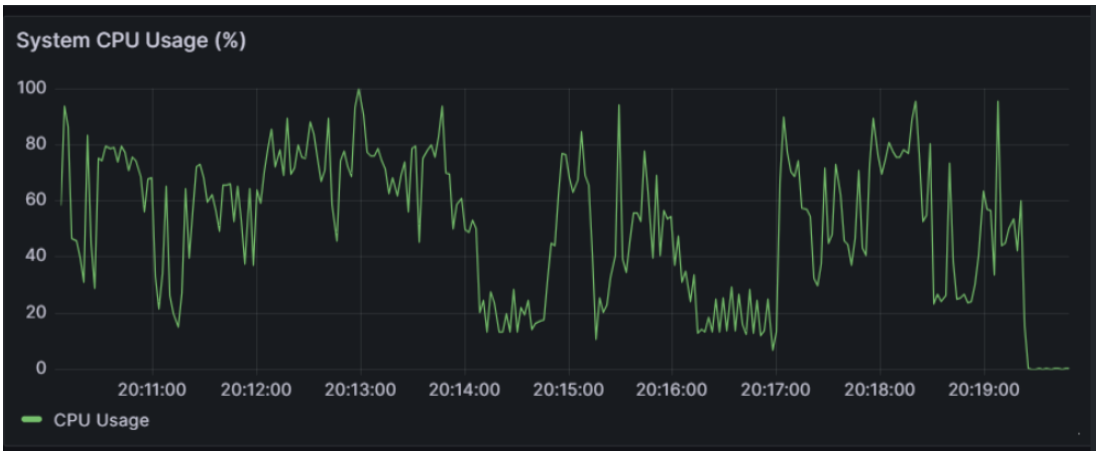


MD Core #2



CPU usage

MD Core #1

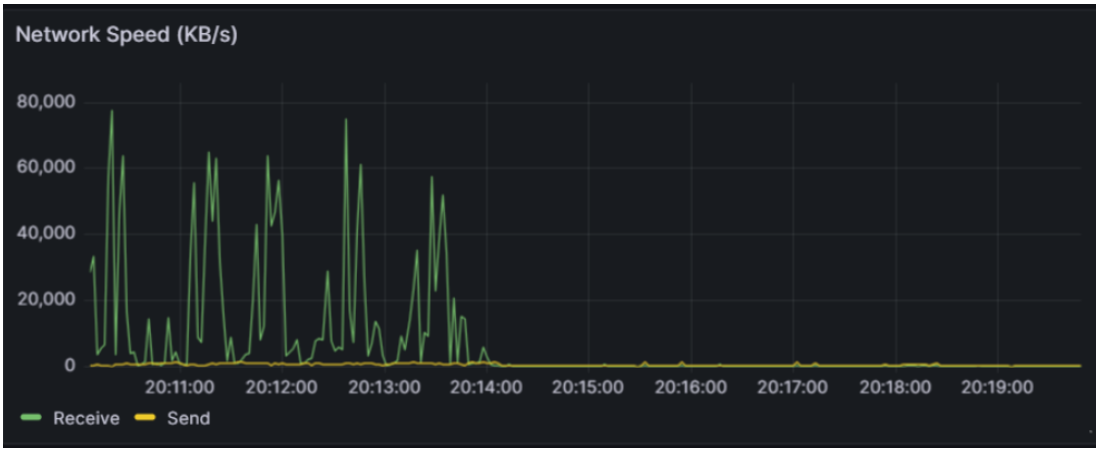


MD Core #2

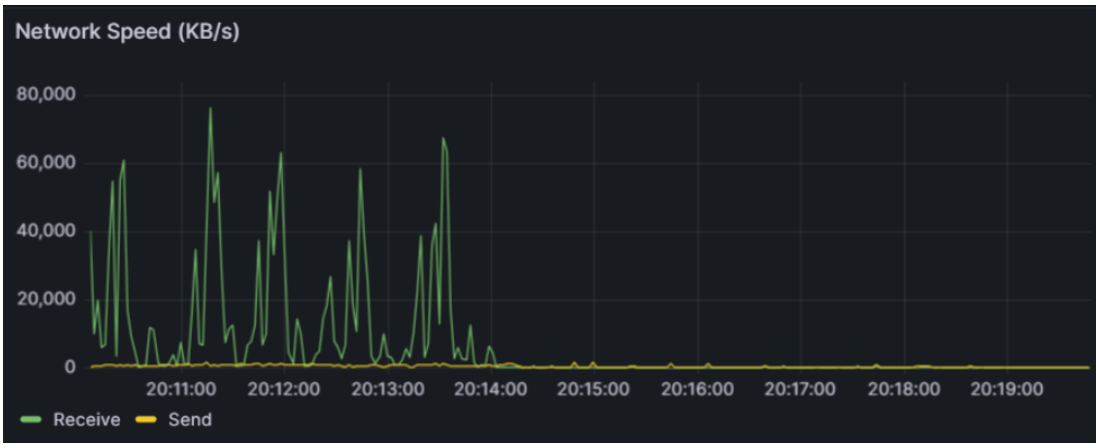


Network usage

MD Core #1



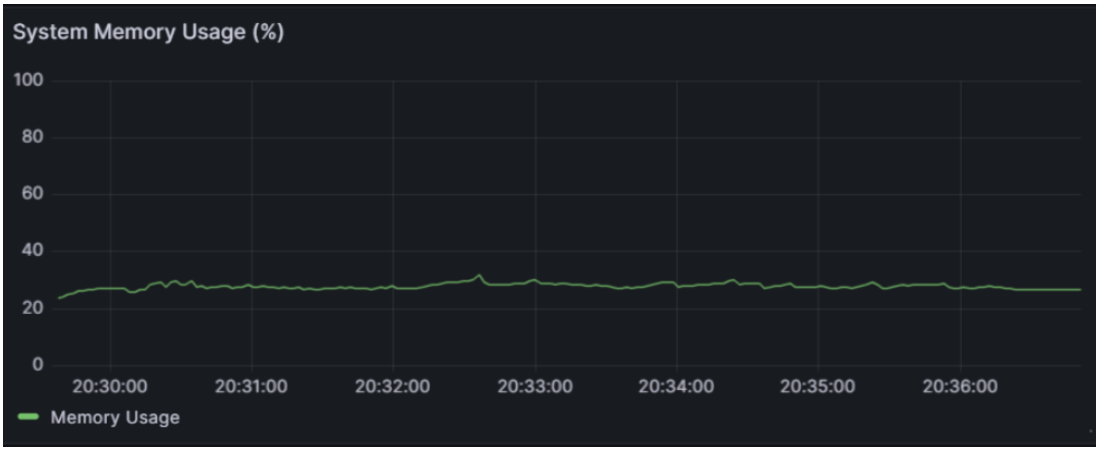
MD Core #2



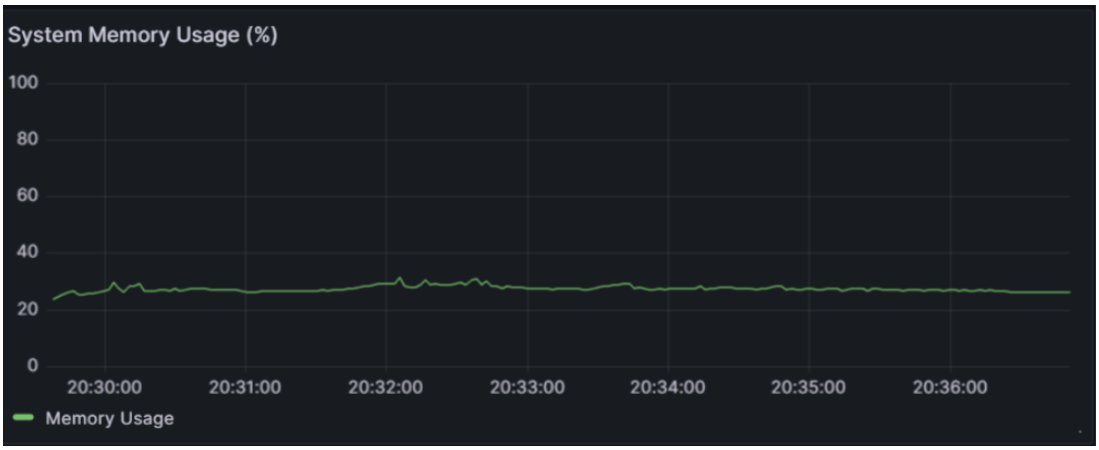
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1

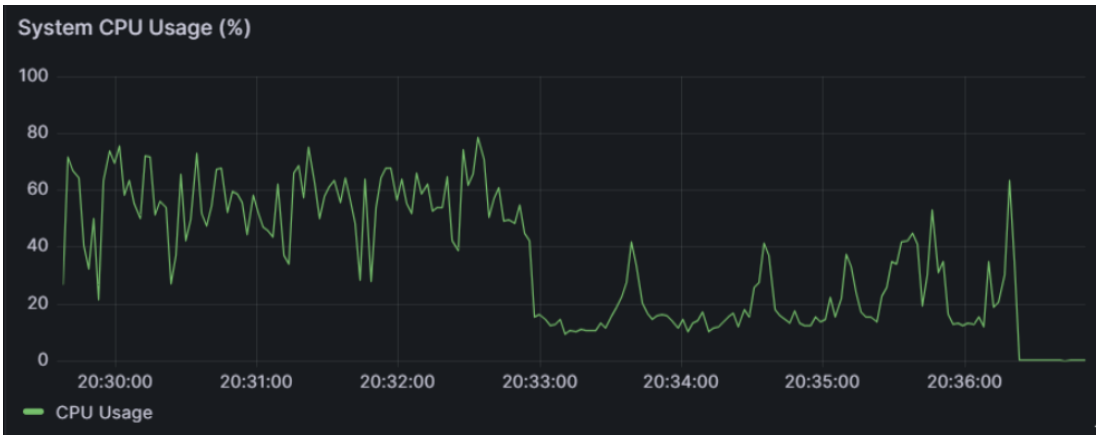


MD Core #2

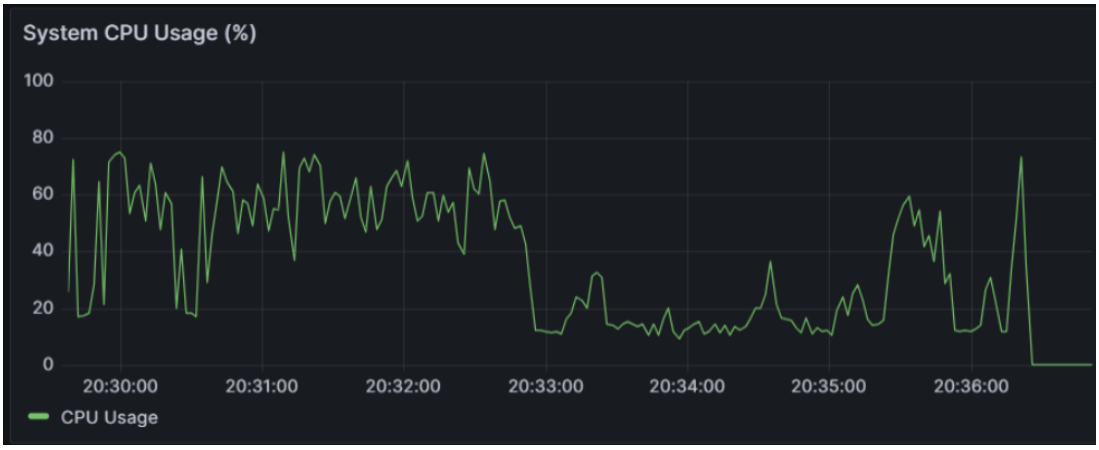


CPU usage

MD Core #1

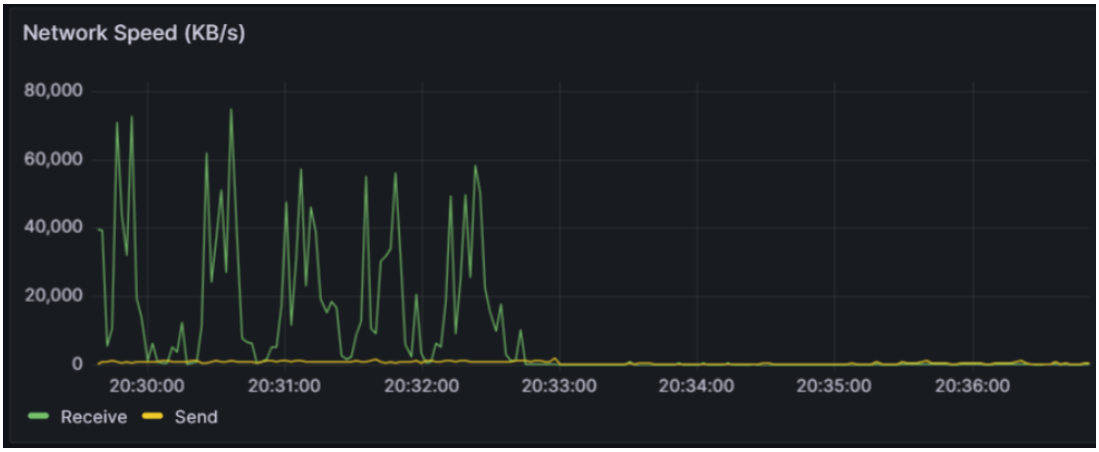


MD Core #2

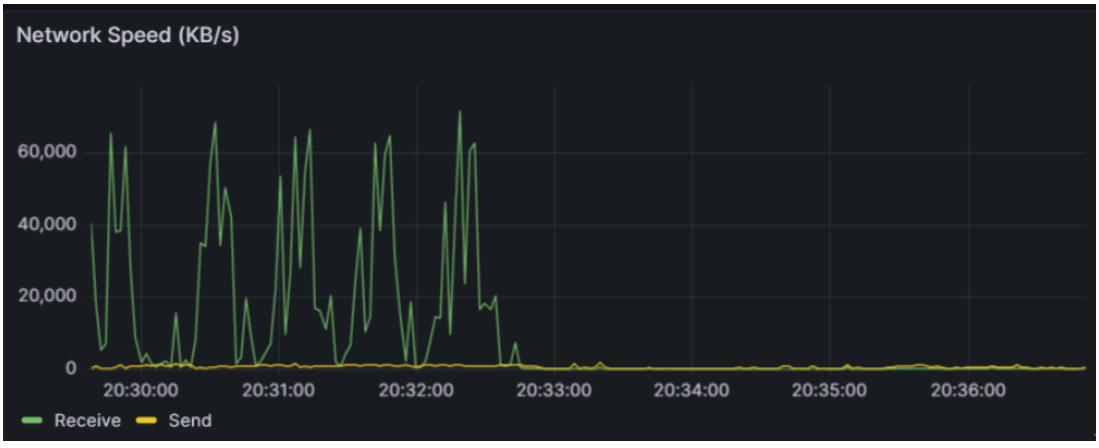


Network usage

MD Core #1



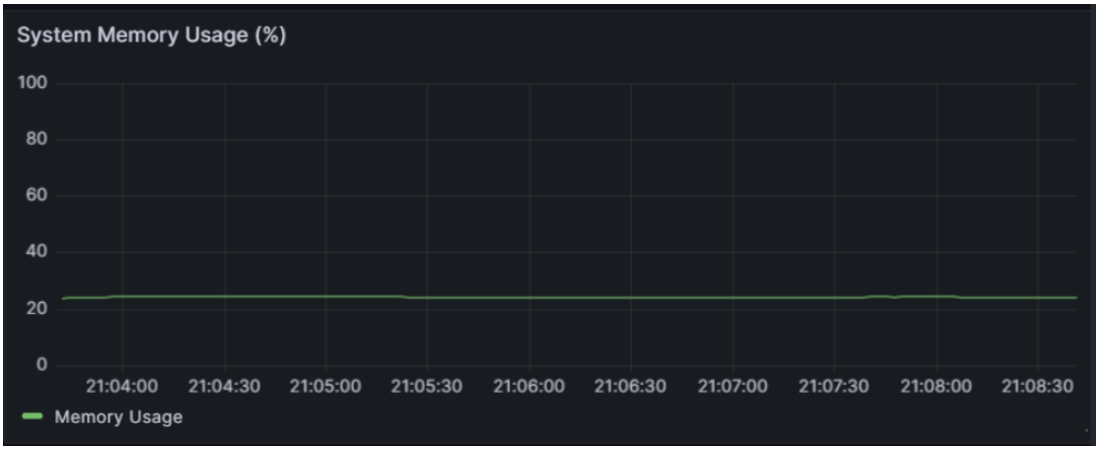
MD Core #2



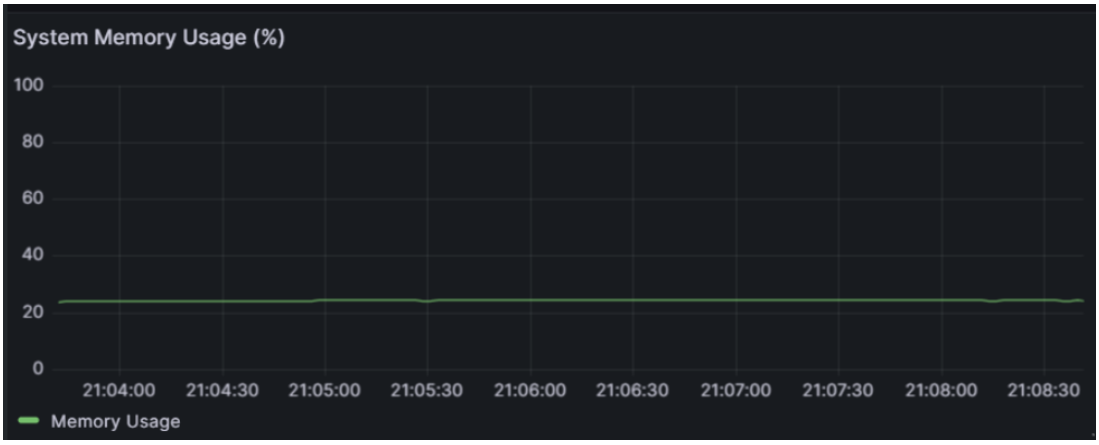
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

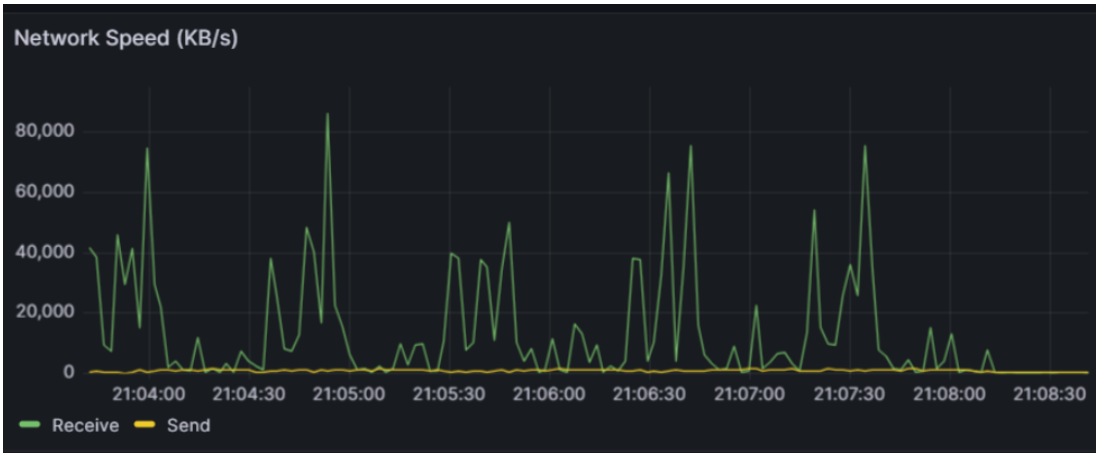


MD Core #2

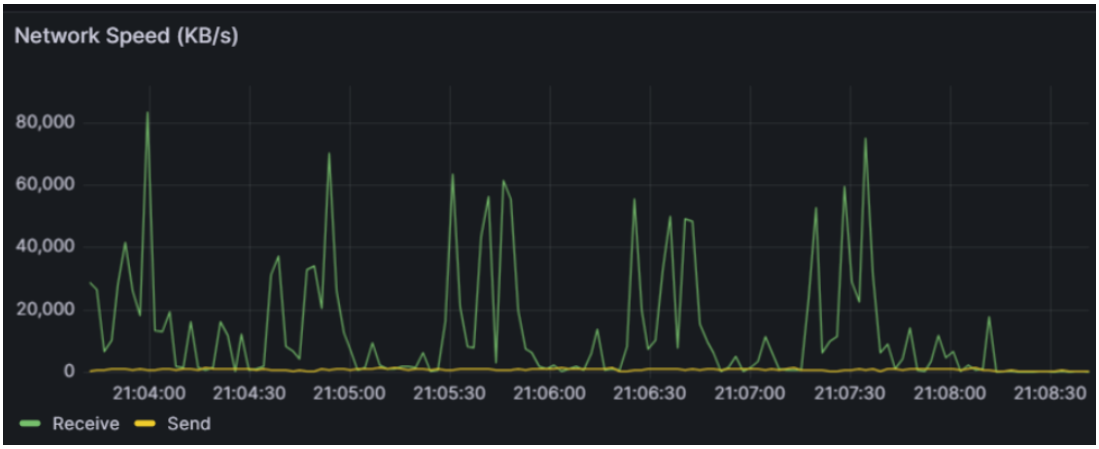


Network usage

MD Core #1



MD Core #2

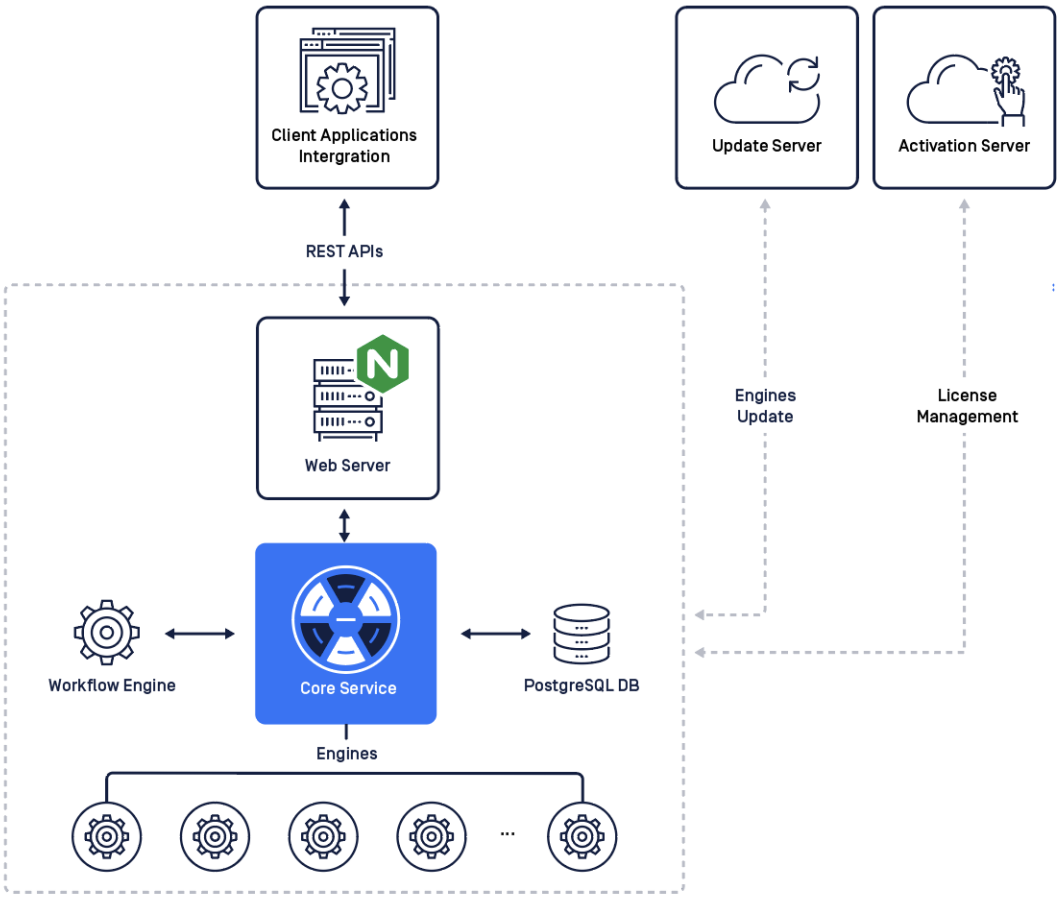

Network Speed (KB/s)
80,000
60,000
40,000
20,000
0
21:04:00
21:04:30
21:05:00
21:05:30
21:06:00
21:06:30
21:07:00
21:07:30
21:08:00
21:08:30
Receive
Send

MetaDefender Core Documentation

The users can consult this web page or, alternatively, they can download the manual in pdf format from the link below:

MetaDefender Core manual (SHA256:).

Recommended System Configuration



MetaDefender Core supports a wide range of deployment options, including air-gapped environments, providing the flexibility to choose the environment that best suits customers' requirements and infrastructure.

- Physical server.
- Virtualization platforms like VMware, Hyper-V and XenServer.
- Cloud Deployment.
- Container Deployment.

> **Note**
> Before installing MetaDefender Core, please refer to the **recommended minimum system configuration** listed below. The requirements are based on a standardized customer deployment and may require more or less resources to achieve the level of performance desired in your environment. Please note that the server specifications are built to allow a high volume daily processing. Please refer to Performance and Load Estimation for more details.

The recommendations below are for MetaDefender Core, API usage only. For any other use cases, please consult the user guide of the licensed products for accurate recommendations.

For certain use cases these might be adjusted and customized on specific needs and SLAs. We highly recommend to engage our ProServ team to assist in fine tuning MetaDefender and get the maximum performance out of your systems.

Microsoft Windows Deployments

[Windows] Supported Operating Systems

Only support 64-bit platforms.

- Windows 10, Windows 11 23H2, 11 IoT (22H2) only for End-User systems (e.g. Kiosk deployments)
  - Based on Microsoft License Terms, you're not allowed to use Windows Desktop as a server, therefore, MetaDefender Core can't be used on these systems for server use cases.
- Windows Server 2016, 2019, 2022

> **Disclaimer**
> End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

> **Plan for legacy OS support**
> OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:
>
> - Ubuntu 20.04 after **April 2025**.
> - Amazon Linux 2 after **June 2025**.
> - Windows 10 after **October 2025**.
>
> If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

[Windows] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 8	8	8 GB	16 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 12	16	16 GB	24 GB + 50 GB + 100 GB
MetaDefender Core 16	16	16 GB	32 GB + 50 GB + 100 GB
MetaDefender Core 20	32	16 GB	40 GB + 50 GB + 100 GB
MetaDefender Core MAX	32	32 GB	120 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core is installed.
- For a better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.
- SBOM processing requires more disk space, please allow at least 15GB free disk for the engine to work properly.

[Windows] Third Party Dependencies

- .NET framework 4.5 or above
- Microsoft Visual C++ Redistributable for Visual Studio 2015-2022 (version 14.38 or higher); x64 version.
- PostgreSQL: Only applicable when using a pre-installed PostgreSQL server running remotely.
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Some engines also have dependencies as described below:

Engine Name	Dependency
Deep CDR	Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 5.8.0 or above)
ESET	MetaDefender Core v4 temporary directory should have more than 200MB free disk space
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
Proactive DLP	• Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 2.0 or above) • .NET 6 dependencies • CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
Scrutiny	Microsoft Visual C++ 2015 - 2019 Redistributable x86 and x64
Systweak	.NET runtime 6.0, 7.0 or 8.0
Bkav Pro	Microsoft Visual C++ 2015 - 2022 Redistributable x64

[Windows] Installation Details

MetaDefender Core on Windows uses "C:\Program Files\OPSWAT" folder for storing resources or the installation directory.

MetaDefender will use its resources folder to store temp files as part of the analysis. **It's recommended to exclude this folder from real-time protection monitoring**. Knowledge base:

- How to set up exclusions on anti-malware software to prevent disruption
- Mitigating scan failure triggered by real-time protection service
- Can local AVs interrupt ongoing scans?

Linux Deployments

[Linux] Supported Operating Systems

Only support 64-bit platforms.

- Red Hat Enterprise Linux 8, 9
- Rocky Linux 9
- Debian 11, 12
- Ubuntu 20.04, 22.04
- Amazon Linux 2 is supported via Amazon Machine Image (AMI)

Plan for legacy OS support

OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:

- Ubuntu 20.04 after **April 2025**.
- Amazon Linux 2 after **June 2025**.
- Windows 10 after **October 2025**.

If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

For Hardended Linux OS

Some AV engines could be failed to initialize and run on MetaDefender Core (e.g. ESET, Kaspersky) due to execution permission required by them on `/var/tmp/ometascan` folder.

Please follow steps at Why have the ESET and Kaspersky scan engines failed to initialize on the hardened Linux OS? for remediation details.

Disclaimer

End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

[Linux] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 5	4	4 GB	10 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 10	8	8 GB	20 GB + 50 GB + 100 GB
MetaDefender Core MAX	16	16 GB	40 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core resides on.
- For better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.

[Linux] Third Party Dependencies

Dependencies list:

- openssl
- grep
- lib32stdc++6 (>= 4.5)
- libc6-i386 (>= 2.10)
- procps
- zlib1g
- libcurl3 (>= 7.19.7)
- libcurl4
- postgresql:
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Not all the above dependencies will need to be installed, it is dependent on different Unix distro and version

Some engines also have dependencies as described below:

Engine name	Dependency
Archive engine	• To extract password protected MS Office files, you need to install .NET 6 dependencies, libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • Rocky Linux 9.x/Red Hat 9.x: libnsl
Deep CDR	• Rocky Linux 9.x: libgomp, ncurses-compat-libs, libnsl • CentOS: libgomp, ncurses-compat-libs • Red Hat 8.x: libgomp, ncurses-compat-libs, libnsl • Red Hat 9.x: libgomp, ncurses-compat-libs, libnsl, compat-openssl11 • Ubuntu/Debian: libgomp1, libncurses5 • libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • .NET 6 dependencies
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
FileType engine	Rocky Linux 9.x/Red Hat 9.x: libnsl
Proactive DLP	libgdiplus, mscorefonts .NET 6 dependencies CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
RocketCyber	libgomp

Note: Some dependencies may need to be installed from external repositories, not from the OS default repositories.

[Linux] Installation Details

MetaDefender Core default installation path is using /var folder for storing resources:

- `/var/lib/ometascan` : installation folder with all its resources (database, DLP processed / CDR sanitized / quarantined file storage, engine package and definition updates).
- `/var/log/ometascan` : application logs.
- `/var/run/ometascan` : runtime folder for Core related processes.
- `/var/lib/ometascan` : all successful deployed engines data files.
- `/usr/lib/ometascan` : all shared libraries required by MetaDefender Core, bundled PostgreSQL server related binaries and libraries.
- `/var/tmp/ometascan/resources` : all temporary processing files for MetaDefender Core to process.

Storage Usage Information

Based on the configuration, MetaDefender Core will need additional disk space to store analysis/result data, quarantined files and temporary files:

	Data Storage
Analysis Reports (Processing History)	Approximately 4.5GB is required for every 1M analysis reports stored. • 100 GB is recommended as a baseline. Database storage requirements are affected by Data Retention settings, engine settings and scanning load.
Quarantined Files	Depends on the customers' dataset and configuration
Sanitized Files	Depends on the customers' dataset and configuration
Temporary processing files	Depends on the customers' dataset and configuration • A 1GB non-archive file requires 1GB free disk space at least. • A 1GB archive file requires (1 * 2)GB free disk space at least. • 50GB is a baseline and should be adjusted based on observation of the solution within the environment, with consideration to peak times and overall file makeup (e.g. complexity, size, archives).
Note	Options are available to control how large and/or complex files are handled to limit the amount of system resources required. See Workflow Configuration Template for more information.

Custom Engines

The recommendations above are specific for MetaDefender pre-packaged bundles.

However for additional Custom Engines, please review the Knowledge base to review additional requirements (if any) for the selected engine.

Browser Requirements for the MetaDefender Core Management Console

One of the following browsers is suggested to view the MetaDefender Core Management Console:

- Microsoft Edge
- Chrome
- Firefox
- Safari

Chrome, Firefox, Safari and Microsoft Edge browsers are tested with the latest available version at the time of release.

Network Bandwidth

In online environment where MetaDefender Core is configured to download engine update packages directly from OPSWAT update server source, then using gigabit network speed (125 MB/sec) is highly recommended to avoid potential network related issue while downloading files, or at least 100 Mbps (12.5 MB/sec) network speed.

Allowlisting URLs

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts':

- https://activation.dl.opswat.com - for license online activation
- https://update.dl.opswat.com - for engines and databases download

Note: *IP address-based allowlisting on your firewall might fail after some time since OPSWAT uses CDN (AWS Cloudfront) to faster delivery updates over the world, and IP address of edge servers might change over time.*

Depending on your location, connections with OPSWAT cloud-based servers be forwarded to specific AWS Cloudfront's edge locations, each is assigned with its own unique IP address ranges. You might also need to allow correct AWS Cloudfront's corresponding IP address ranges by referring to https://ip-ranges.amazonaws.com/ip-ranges.json

Warning

Even the OPSWAT update servers host updates for all of the available engines we support, sometimes custom engines might try to connect to their own cloud for updates, but this can be disabled in firewall and they will be updated just from OPSWAT cloud servers instead.

Differences Between MetaDefender Core v4 and v5

We are very excited to inform you about the release of version 5.0.0 of our MetaDefender Core!

The new version is designed to enhance the product performance, user experience, as well as to provide more flexibility, security, and scalability for our customers.

At the same time, we are also committed to keeping backward compatibility for whom running MetaDefender Core v4, and avoid the need for updating your client integration to move forward with MetaDefender Core v5.

v4 REST API Integration Supported
All public REST API endpoints available on MetaDefender Core v4 will be continously supported on v5.

v4 Product Major Features Supported
All product major features available on MetaDefender Core 4.21.2 (the latest one on v4) will be continously supported on v5.

Seamless Product Upgrade
Upgrading from MetaDefender Core v4 to v5 will be supported and covered with data migration without additional manual steps from users.

Licensing information will be also migrated.

Below are some highlights of the first v5 release, MetaDefender Core 5.0.0:

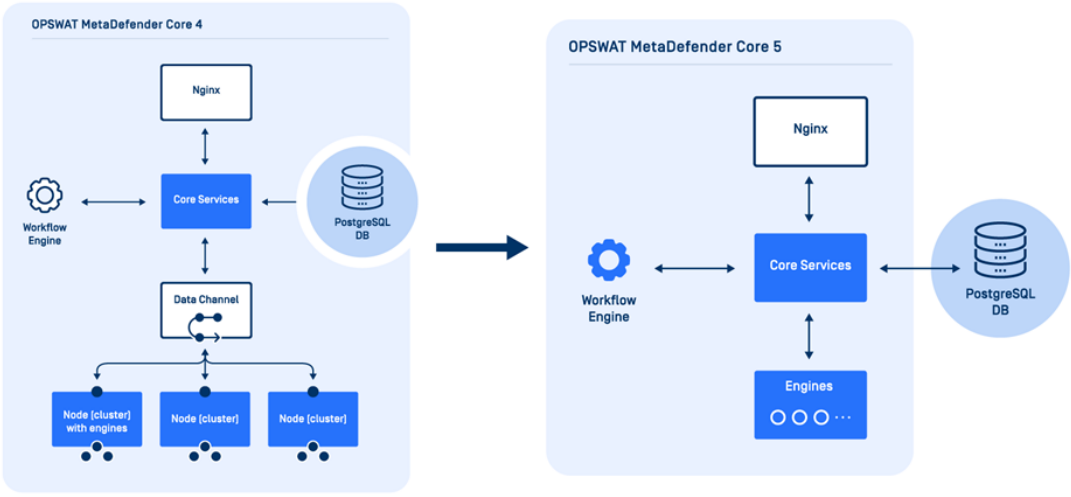
New product architecture

This version introduces a simpler product architecture that speeds up integration and automation, file processing flow, and immensely enhance the product performance and efficiency.

Earlier versions utilized MetaDefender Node concept, designed to support scaling of the scanning infrastructure by distributing scan requests among several different processing Nodes. Product licensing was based on number of Node instances. The benefit of having such a distributed infrastructure is that based on Node loads, Metadefender Core server can always choose the most appropriate Node to assign a new scan task to. However, having nodes in the v4 architecture caused more complexity and possible latency in the processing flow and entailed maintenance cost.

In this release, we retired multi-Node deployment and restructured MetaDefender Core's backend. This alteration significantly improves file processing speed and product efficiency, as well as eliminates extra cost for maintaining node services. Users now can simply install additional MetaDefender Core instances for scalability requirement,

as well as configure their own load balancer to coordinate tasks among multiple MetaDefender Core instances. Accordingly, product licensing is now based on number of the product instances.



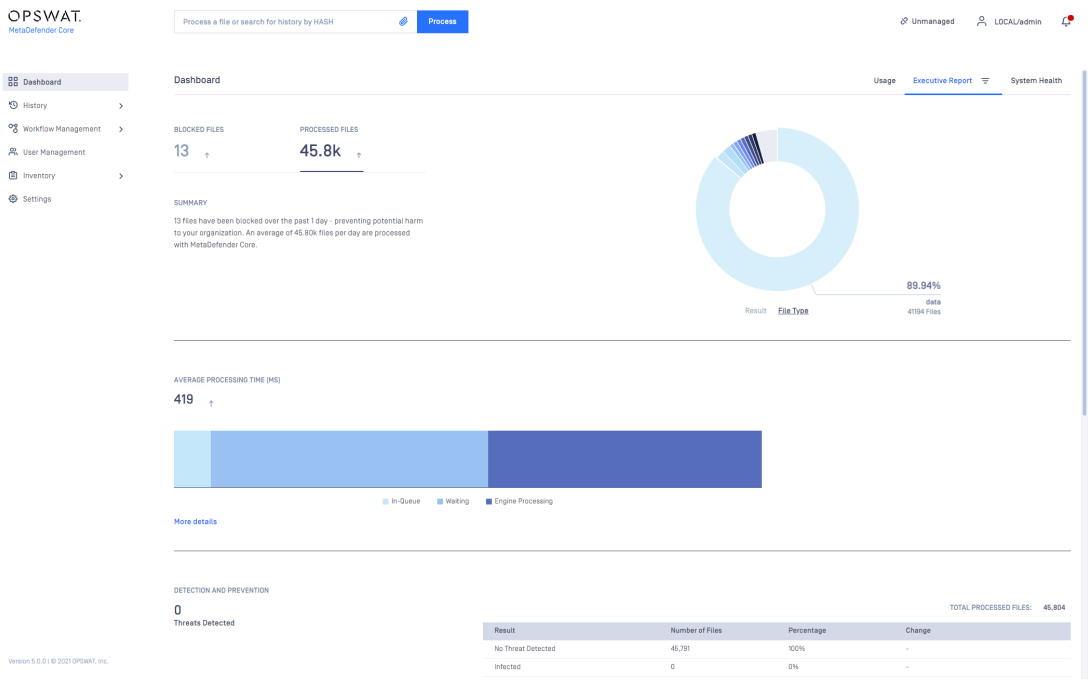
Please be informed that

- All existing administrative APIs related to "Node" on v4 will be kept as it is on v5 to ensure backward compatibility with existing client integrations.
- Applicable settings defined in Windows registry (Windows) or ometascan-node.conf config file (Linux) for Node will be migrated to Core. Details: MetaDefender Configuration

Brand-new product user interface

The second notable update in this version is a brand-new UI for the management console, which was designed with the focus on the quality and thoughtfulness of the user experience. The new administrative console UI provides an advanced set of Core features and observable processing status.

Comprehensive executive dashboard reporting offers dynamic, real-time and actionable threat insights. It also provides visibility of your security posture allowing you to paint a picture of security that aligns with the overall business. The report includes easy-to-read charts, graphs, and statistic explanation that help your security professionals reduce clutter, gather information quicker, spot threat activity early and act swiftly.



We're continuing to improve our product UI to provide users an excellent experience with the application and overall usability. Please expect more powerful UI features coming in next releases.

New compression archive engine for archive sanitization

In previous version, MetaDefender Core's archive engine implemented four functions including decryption, extraction, encryption and compression that with a high volume of files it takes time to process them all. For the purpose of enhancing archive handling performance, this release comes with a separated compression archive engine, which dedicatedly processes archive sanitization (compression and encryption). It is also created to facilitate maintenance and further feature development.

Archive sanitization feature now requires Deep CDR license

As to respect sanitization practice and ensure unknown threats prevention as designed, now starting on version 5, it's mandatory to have Deep CDR licensed for archive sanitization feature enablement on MetaDefender Core workflow configurations.

Archive handing related engines changed

The legacy "Archive" engine on MetaDefender Core v4 will be changed its name to "Archive Extraction" engine, even when you are still on MetaDefender Core v4.

A brand new "compression archive engine" is to dedicatedly handle archive sanitization in v5 (archive compression), that helps relevant archive handling performance improvement, easier for maintainance and further feature development.

Notes when you are still on MetaDefender Core v4 (not upgraded yet)

The legacy "Archive" engine (now name changed to "Archive Extraction" engine) on MetaDefender Core v4.x will still cover both extraction and compression archive tasks as usual.

Even though your MetaDefender Core v4.x might now have a new "Archive Compression" engine auto downloaded, but that engine cannot be deployed (marked as "permanently failed" in relevant engine status check API). This behavior is expected, and not causing any impact to MetaDefender Core processing flow, or its performance though.

- Just in case your client integration has a check for engine status before triggering scan request, then you might want to disable "Archive Compression" engine status on MetaDefender Core v4 using REST API. Detailed instruction: Archive Compression Engine Permanently Failed on MetaDefender Core v4
- When you later upgrade to MetaDefender Core v5 from v4, then you might need to manually re-enable "Archive Compression" engine to make it active and running on MetaDefender Core v5.

Notes for licensing

For offline deployment (MetaDefender Core is not connected to the Internet directly for online update), you will need to re-activate again on OPSWAT portal to retrieve latest engine list in .yml file including "Archive Compression" engine if you are currently licensed with Deep CDR.

Meanwhile, for online deployment, this manual procedure is not required since MetaDefender Core will cover that itself.

Import & export configurations

Not only workflow settings like in previous versions, but also all other general settings (update, security, data retention, server, hashes to skip) and most importantly, user settings, will be able to export and import on MetaDefender Core version 5.

Learn more at Import/Export configuration

Backward compatibility supported

The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

Windows application event logging disabled by default

Applicable only to Windows OS based MetaDefender Core.

Help avoid duplicated logging with separate file option, and improve overall processing performance. Of course, when being still valid to you, you are still supported to enable it back.

CSRF token header renamed

CSRF token originally created for security strengthening and being used by MetaDefender Core management console UI since version 4.21.0 via header `oms_csrf_token`

To get it compliant with some load balancers which do not accept underscore in header name by default, and save you from additional load balancer configuration step, we have now changed the header name to `oms-csrf-token`

This change does not impact to any REST API based client integration.

More comprehensive and accurate error messages on REST APIs

Impacting on following REST API endpoints:

- `PUT /admin/config/session`
- `PUT /admin/config/update`
- `PUT /admin/config/scan`
- `PUT /admin/config/skip`
- `PUT /admin/config/auditlog`
- `PUT /admin/config/webhook`
- `PUT /admin/config/rule`

Upload time and queue time logic updated in JSON scan response

Impacting on following REST API endpoints:

- `GET /file/{data_id}`
- `GET /hash/{md5/sha1/sha256}`
- `GET /archive/{data_id}`
- `POST /file/{async}`

Added new field:

- `receive_data_timestamp`: The timestamp when upload progress started (first byte received) (in milliseconds)
- `upload_time`: Total time elapsed for upload process (in milliseconds)

Modified existing field (new logic):

> **Backward compatibility supported**
> The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

- `upload_timestamp`: The timestamp when upload progress finished (all bytes received) (in milliseconds)
- `queue_time`: Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue time = start_time - upload_ timestamp) (in milliseconds)

Release version 5.0.0 also comes with variety of outstanding features and improvements.

For detailed information about our latest release please view our Release notes

Installation

Starting with MetaDefender Core 4.20.0, the centralized/shared database deployment model is supported (all MetaDefender Core instances share the same database), along with traditional standalone database mode (every MetaDefender Core instance has its own database to manage).

- Standalone mode
- Shared Database Mode

Installation

1. Download the package of your choice from My OPSWAT
2. Install the package on your computer via the Command Line or via the Wizard
3. Open a web browser and point to http://<server name or IP>:<port>
    - The default port is **8008**
    - In case of problem check Inaccessible Management Console page
4. Complete the required steps of the basic configuration wizard
5. You must activate this deployment to use its features

Installation notes

- If the MetaDefender Core package **dependencies are not installed** on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.
- During installation the databases might need to be upgraded. This could take noticeable time depending on database size (eg.: length of scan history).

Standalone mode

Wizard Installation

The Install Wizard is only for the Windows installer (.msi file).

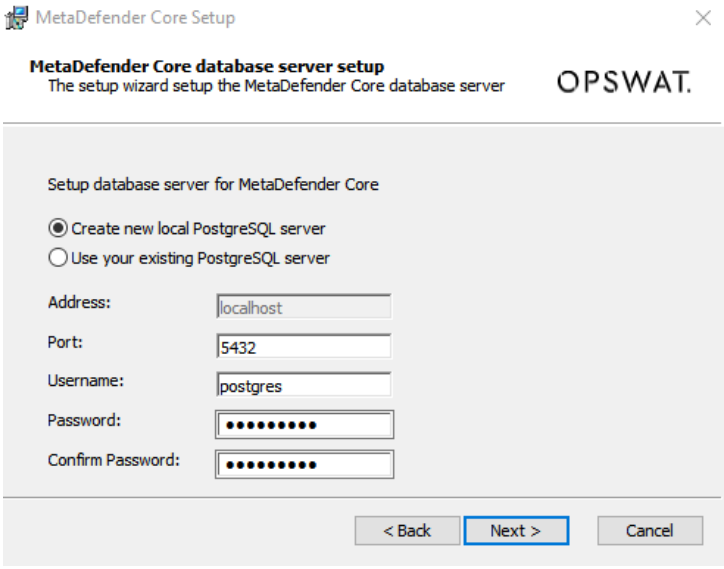
Since MetaDefender Core 4.19.0, we have switched to using PostgreSQL DBMS which needs a dedicated database server. MetaDefender Core supports two modes to setup PostgreSQL server:

Local (Bundle) PostgreSQL server

MetaDefender Core will install a new PostgreSQL server locally in the same box with the product.

> **PostgreSQL user credentials**
> Unicode characters are NOT supported for "Username" and "Password"
>
> "Password" does not accept # (hash) as starting character (e.g. #ABC)

> **The ignition file**
> If the ignition file exist in the designated location, then MetaDefender Core will use the details defined in the ignition file, rather than what is passed via the installation wizard.
>
> More info about the ignition file based installation: Command Line Installation

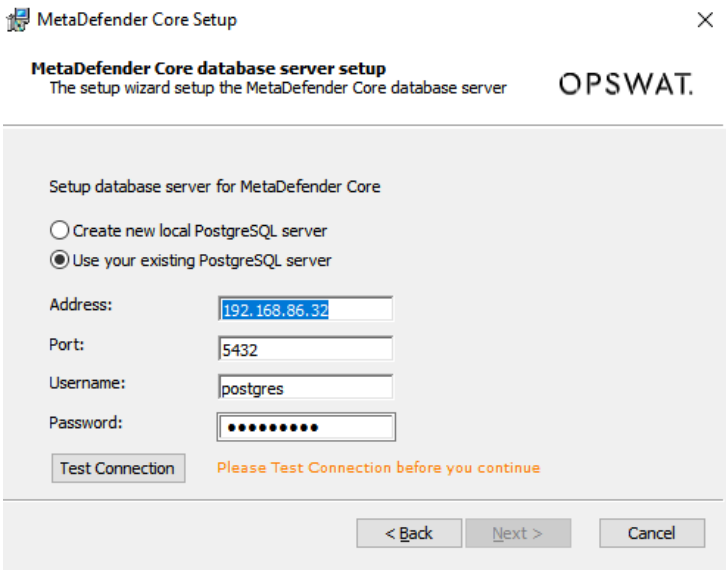


Remote (Existing) PostgreSQL server

MetaDefender Core will connect to remote pre-installed PostgreSQL running remotely.

> **PostgreSQL connection configuration**
> "Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

> **PostgreSQL user privilege requirements**
> MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.
>
> Details: PostgreSQL User Privilege Requirements



Command Line Installation

Ignition File

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Make sure to create the ignition file, if it doesn't exists before you go ahead and install MetaDefender Core via command line.

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

MetaDefender Core supports two modes to setup PostgreSQL server:

1) Local PostgreSQL server

- MetaDefender Core will install a new PostgreSQL server locally, next to the product.
  - A sample iginition file for PostgreSQL server information (***Only non-Unicode characters supported for "user" and "password"__***):

ometascan.conf bash

```
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
```

2) Remote PostgreSQL server

> **Info**
>
> **PostgreSQL version 12 (at least 12.3) or version 14 is required.**
>
> According to PostgreSQL notice, version 12 will stop receiving fixes on November 14, 2024, so we suggest that you use version 14.

MetaDefender Core will connect to a pre-installed PostgreSQL server running remotely and will setup its database.

Make sure to follow steps at Open Connection On PostgreSQL Server first, so that your remote existing PostgreSQL server to accept upcoming connection from MetaDefender Core.

A sample ignition file for PostgreSQL server information:

ometascan.conf bash

```
[dbserver]
type=remote
host=192.168.86.32
port=5432
user=existingserver_admin_user
password=existing_server_pass
```

After installing MetaDefender Core successfully, you may want to remove all credentials info created for [dbserver] section in ignition file for security reason.

> **Warning**
> If the MetaDefender Core package dependencies are not installed on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.

Debian package (.deb)

bash

```
sudo dpkg -i <filename> || sudo apt-get install -f
```

On Red Hat Enterprise 7.x Linux / CentOS package (.rpm)

bash

```
sudo yum install <filename>
```

For systems which enabled GPD check flag:

bash

```
sudo yum install --nogpgcheck <filename>
```

Red Hat Enterprise Linux 8.x package (.rpm)

Install the latest PostgreSQL repository RPM:

> **Note**
> Due to the shorter support cycle on Red Hat Enterprise Linux, all supported versions of PostgreSQL are not available on this platform. The PostgreSQL Yum Repository will integrate with the normal systems and patch management, and provide automatic updates for all supported versions of PostgreSQL throughout the supported lifetime of PostgreSQL. Reference: https://www.postgresql.org/download/linux/redhat/

bash

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

In case installing a remote PostgreSQL on either the same MetaDefender Core server or another server, you need to install the following additional package. PostgreSQL version 12.3 or newer is recommended.

bash

```
sudo dnf install -y postgresql12-contrib
```

Install MetaDefender Core:

bash

```
sudo dnf install -y <filename>
```

Windows package (.msi)

On Windows systems it is possible to install the product by running the corresponding .msi file.

From command line interface it is also possible to install the product by executing

bash

```
msiexec /qn /i <msi file name> <option key>=<option value>
```

where the possible keys and their default values are the following:

Key	Default Value	Description
INSTALLFOLDER	\Program Files\OPSWAT\MetaDefender Core	Customize installation folder for product Example: `INSTALLFOLDER="D:\Products"`
RESTADDRESS	*	REST interface binding IPv4 or IPv6 address ('*' means that service listens on all IPv4 and IPv6 interfaces)
RESTPORT	8008	REST interface binding port
REPORT_ENGINE_ISSUE	true	Enable reporting of engine issue count. (possible values: "true" or "false").

Note

Reporting of engine issue count

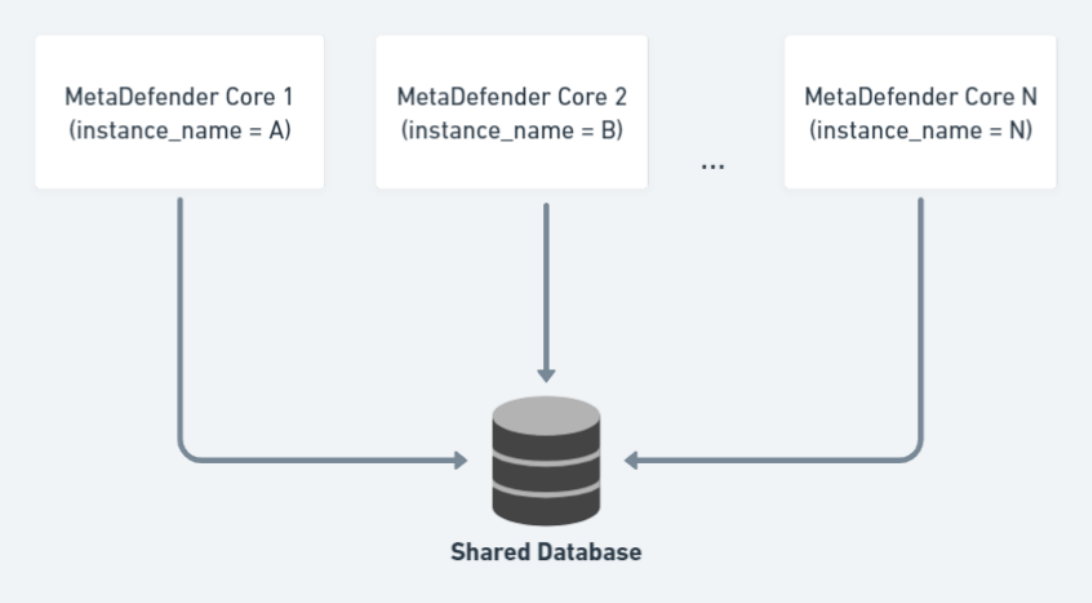
If reporting of engine issue count is enabled, MetaDefender Core server will send only the **number** of initialization errors and **number** of unexpected stops for the specific db/engine version. This information is sent over a HTTPS channel when the product downloads the latest package descriptors. This information is used for early detection of any specific 3rd party engine quality issues.

For details on using msiexec please consult Windows installer documentation.

Shared Database Mode

Starting with MetaDefender 4.20.0, you can use a centralized database deployment model, where all MetaDefender Core instances can share the same external database.

That helps auto-sync all product configurations (**except Single-Sign-On** that requires unique settings for each Core), processing history, live/on-going scan result query, audit, statistics data.



This article demonstrates how to install and setup MetaDefender Core 4.20.0 or newer with shared database mode.

Important notes & current limitations:

Currently not supported

- Does not support Central Management use-case with shared DB mode.
- Does not support to switch from shared database mode back to standalone database mode.
- Does not support to change instance_name of MetaDefender Core in shared database deployment model after installation.
- Does not support upgrading older MetaDefender Core from non-persistent mode to using the shared database mode.
- Does not support upgrading older MetaDefender Core from standalone mode to using the shared database mode.
- Does not support to upgrade multiple MetaDefender Core instances at once in shared database mode
    - Stop all Core instances' scanning service before performing the upgrade.
    - Upgrade each Core server one by one instead.

- For each Core, when the upgrade finished, then you can start using that Core for scanning without waiting for the other Core instances' upgrade to be done.

> **Disclaimer**
> MetaDefender Core does not support mode switching.

Requirements for shared database setup:

- The central remote PostgreSQL server (for shared database) could be utilized with a built-in (native) PostgreSQL database server installed with one of the MetaDefender Core instances in the network. Or using a dedicated existing PostgreSQL database server (see detailed requirements for PostgreSQL version at Recommended System Configuration).
- Before setting up, make sure to follow steps at Open Connection On PostgreSQL Server so that all MetaDefender Core instances could connect to the central remote PostgreSQL server.
- MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core, details: PostgreSQL User Privilege Requirements
- All MetaDefender Core instances must be installed on the same Operating System (Windows / Linux).
- The first MetaDefender Core instance connects to the shared database must be ready first (license activated, users/groups/roles created) before setting up other MetaDefender Core instances to connect and use that shared database.

Engine update:

- If engine updates is configured to be via Folder source, all MetaDefender Core instances must be able to access to the configured update folder, and the option DELETE FILES AFTER IMPORT will be unchecked and prevented for modification.

Authentication:

- Single-Sign-On (SSO) authentication settings (if using) cannot be shared among MetaDefender Core instances because SSO requires an unique login URL for every different Core.
- If an user is locked for login on a certain MetaDefender Core instance(e.g. exceeded the number of failed login times), that user is not prevented for logging in on any other instances.
- If an user is locked for sign-in on a certain MetaDefender Core instance (e.g. exceeded the number of failed login times), if the admin releases lockout on that user on that MetaDefender Core instance, that user is still not able to login on other MetaDefender Core instances.

Sanitized / DLP / Quarantined location changed after MetaDefender Core install:

Windows:

Shared storage path will be stored in 3 strings under key global in registry: quarantinepath, dlppath, sanitizepath as below

(Default)	REG_SZ	(value not set)
dbmode	REG_SZ	4
dlppath	REG_SZ	\\10.40.50.100\test
instance_name	REG_SZ	481rvj
quarantinepath	REG_SZ	\\10.40.50.100\test
restaddress	REG_SZ	*
restport	REG_SZ	8008
sanitizepath	REG_SZ	\\10.40.50.100\test

You can set either 1, 2 or 3 different paths for the 3 settings above (means that quarantined, DLP and sanitized files will be stored in same or different storages corresponding). In case shared storage is not set in installation, values of 3 settings above will be empty.

Restart MetaDefender Core service after changed.

Linux:

Open file `/etc/ometascan/ometascan.conf` and set shared storage path for 3 keys quarantinepath, dlppath, sanitizepath under section [global]

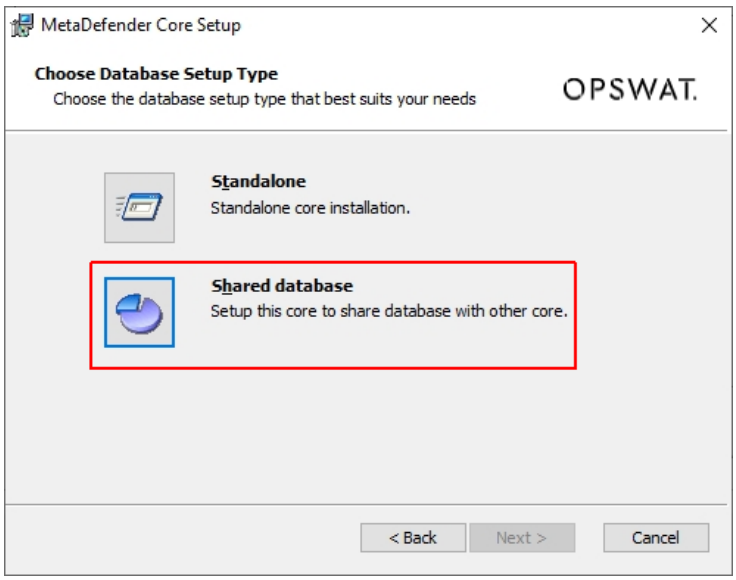
Restart ometascan service after changed.

Wizard Installation (Windows Only)

> **Info**
> Only applicable to MetaDefender Core 4.20.0 or newer using wizard on Windows OS.

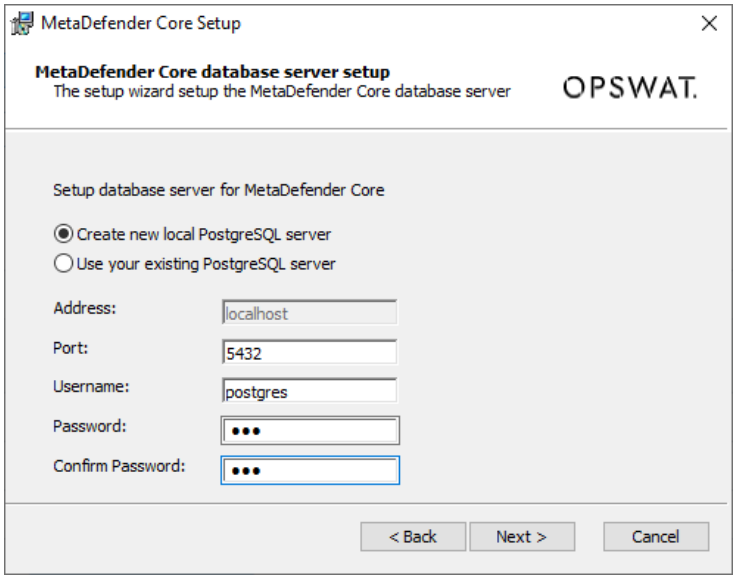
Clean install scenario

1. **Choose Shared database option:**

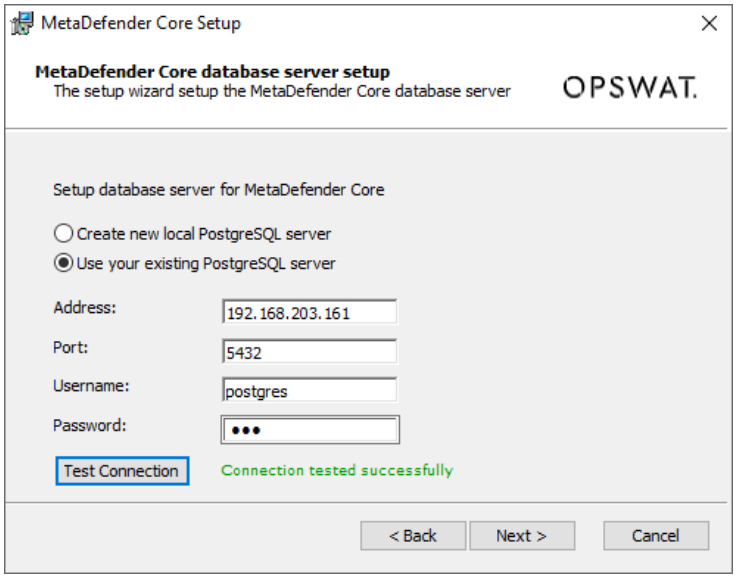


2. **Setup database server for MetaDefender Core:**

You can choose to create a native / built-in PostgreSQL server and setup the shared database on (then other MetaDefender Core instances will want to connect to that PostgreSQL server afterwards)



Or choose to connect to a pre-setup existing PostgreSQL server:



PostgreSQL connection configuration

"Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

PostgreSQL user privilege requirements

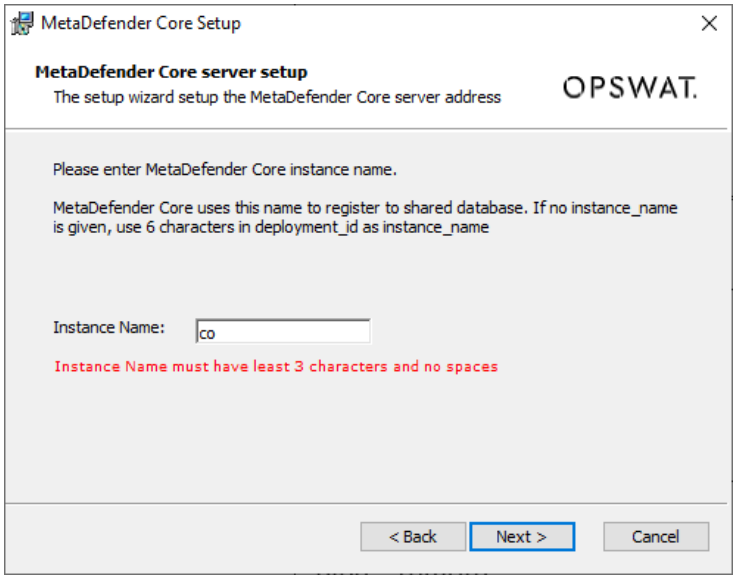
MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.

Details: PostgreSQL User Privilege Requirements

3. **Create an unique instance name for that MetaDefender Core (different from other MetaDefender Core instances):**

Instance name must contain at least 3 characters and no white space.

If you choose to leave it blank, which means it will use the default instance name with 6 characters extracted from that MetaDefender Core deployment ID.

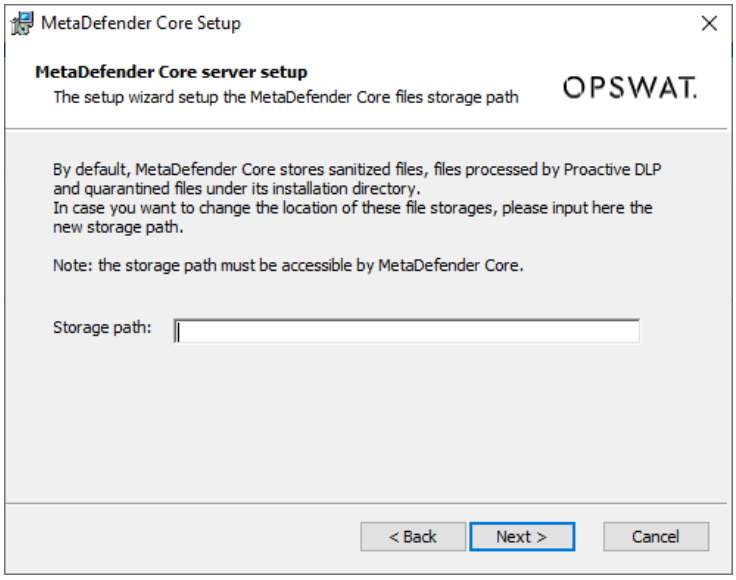


4. **Input the file storage path (sanitized, DLP, quarantined files)**

In the shared database model, all MetaDefender Core instances could use the same folder path to store all their own sanitized, DLP, quarantined files after processed.

The default location is: ***<MetaDefender Core installation folder>\data\ {sanitized|dlp|quarantine}***

If you wish to use that default location, just leave the input box blank and press Next.



Otherwise, you can specify a different location of sanitized/dlp/quarantined files. It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core.

For example:

- C:\new_folder
- \192.168.1.1\new_folder

Command Line Installation

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Clean install scenario

1. Create ignition file if not existed

- Windows: **C:\OPSWAT\ometascan.conf**
- Linux: ***/etc/opswat/ometascan.conf***

An example of ignition file:

ometascan.conf bash

```
eula=true

[global]
dbmode=4
instance_name=MetaDefenderCoreA
storage_path=D:\path\

[user]
name=admin
password=admin
email=admin@local

[dbserver]
type=remote
host=192.168.1.1
port=5432
user=postgres
password=your_own_password
```

Field	Required	Mapping to wizard UI	Possible values
global/dbmode	optional default is 1 (standalone)		1 - Standalone 2 - Central Hub deployment 3 - Non-persistent **4 - Shared**
global/instance_name	optional default is 6 characters of deployment ID		Unicode and number At least 3 characters No whitespace
global/storage_path	optional default is empty		Path to storages (DLP, quarantined, sanitized files) It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core. For example: • C:\folder • \192.168.1.1\folder

2. Run installers using command line to install

Please refer to Command Line Installation for installation command.

Deployment automation support

Note

An Ignition file could be utilized for either / both of following use-cases:

1. Pre-define PostgreSQL server information for MetaDefender Core to connect and setup database: only required on Linux installation, or Windows with command line installation. Learn more details at:
    a. Standalone Mode > Command Line Installation (applicable on version 4.19.0 or newer).
    b. Shared Database Mode > Command Line Installation (only applicable on version 4.20.0 or newer).
2. Automating to accept EULA, create a default local admin user and import MetaDefender Core configurations upon installed: Those steps could be automated using ignition file described in this page.

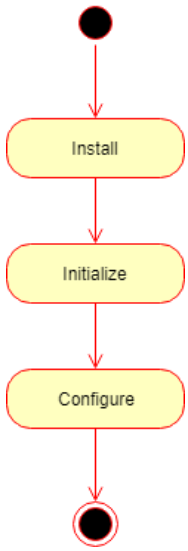
Ignition file location to create:

- **Windows**: `C:\OPSWAT\ometascan.conf`
- **Linux**: `/etc/opswat/ometascan.conf`

The product supports fully automated deployment. It means that it can be installed and configured with no human interaction.

The automated deployment can be split to three steps on a high level:

1. Installation
2. Initialization
3. Configuration



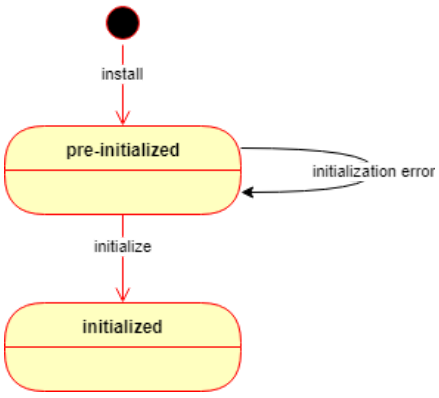
Installation

To automate the installation, install the product from the command line and provide the installation-time options as parameters to the installer. For further details see Command Line Installation

After the installation is complete, the product starts up and waits in a pre-initialized status. The product may be initialized in two ways:

1. Manually using the Wizard Setup or
2. Automatically using an *ignition file* (see below).

> **Warning**
> If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried after fixing the problem (e.g. placing the ignition file in its lookup location) and restarting the OPSWAT MetaDefender Core service.
>
> Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

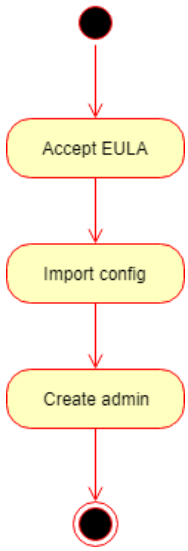


Initialization

Initialization is the process of bringing the product to an operable status.

Basically the initialization consists of the following steps:

1. Accept the End User License Agreement (EULA),
2. Import product configuration and
3. Create the first administrator user account.



Ignition file

The initialization process can be configured in a file called the *ignition file*.

> **Warning**
> In Linux, the ignition file and the folder /etc/opswat should have the permission 755

Ignition file location

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

The ignition file must be in conf format.

ometascan.conf bash

```
eula=true
[user]
name=admin
password=admin
email=admin@local
apikey=44c96cbb4e4e6c0a1d16bcb3c3da24def317
[config]
import=config_package.zip
import_password=123
[security]
session_cookie_samesite=Lax
[internal]
skip_startup_pages=true
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
private_username=my_internal_db_user
private_password=mypassword
[ocm]
ocm_url=https://my.opswat.com
reg_code=XXXXXXXX
```

Ignition file fields

The ignition file accepts the following fields:

Section	Key	Required	Accepted Values	Description
	eula	Optional (mandatory when having [user] section)	true false (default)	Whether to accept the End User License Agreement. This key must be set at the top of ignition file. This key must be set to "true" to accept the EULA. Any other value will cause the initialization to fail. When having [user] section defined, then this key must be set.
[user]		Optional		Initial administrator user account properties. The Administrator role is granted to the account.
	name	Optional	string	User name for the initial administrator user account.
	password	Optional	string	Password for the initial administrator user account. WARNING! Clear text password The password in this configuration file must be stored in its clear-text format and as so it may be visible for unauthorized parties.
	email	Optional	string	E-mail address for the initial administrator user account.
	apikey	Optional	string	API key for the initial administrator user account.
[config]		Optional		
	import	Optional	string	Path to a file in json/zip format that contains a previously exported configuration to be imported. Zip-format file contains a full configuration including user management settings, so it will overwrite [user] section.
	import_password	Optional	string	Password to decrypt .zip configuration package.
[security]		Optional		

Section	Key	Required	Accepted Values	Description
	session_cookie_samesite	Optional	Strict Lax (default) None	Declare if session cookie should be restricted to a first-party or same-site context.
[internal]		Optional		
	skip_startup_pages	Optional	true false (default)	Skip all welcome and other startup pages (e.g. upgrade result) when installing / upgrading MetaDefender, and going straight to the product index default page.
[dbserver]		Optional		
	type	Mandatory when having [dbserver] section	local remote	Where the PostgreSQL database server should physically locate. "remote" means PostgreSQL server and MetaDefender Core are not in the same machine.
	host	Mandatory when having [dbserver] section	string	IP address / domain name of the server where PostgreSQL server locates. "localhost" can be used when applicable.
	port	Mandatory when having [dbserver] section	number	Port of PostgreSQL server is listening for connections from clients (i.e. MetaDefender Core).
	user	Mandatory when having [dbserver] section	string	PostgreSQL server's user. SUPERUSER privilege is required for MetaDefender Core to setup its database and extensions for the first time. **Only non-Unicode characters supported.**

Section	Key	Required	Accepted Values	Description
	password	Mandatory when having [dbserver] section	string	PostgreSQL server's user credentials. **Only non-Unicode characters supported.**
	private_username	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
	private_password	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
[ocm]		Optional		
	ocm_url	Optional	string	URL to My OPSWAT server.
	reg_code	Optional	string	A registration code to enroll to My OPSWAT server.

Detailed initialization process

1) After the product has been started, it looks for the ignition file in the designated location.

Windows: C:\OPSWAT\ometascan.conf ** Linux: /etc/opswat/ometascan.conf

** 2) If an ignition file is found, then

2.1) It gets validated, and if it is valid, then based on the information found in the ignition file:

- The EULA is accepted,
- The configuration is imported,
- The administrator account is created.
- If any of the above steps fails, then the error is logged, and the initialization gets terminated.

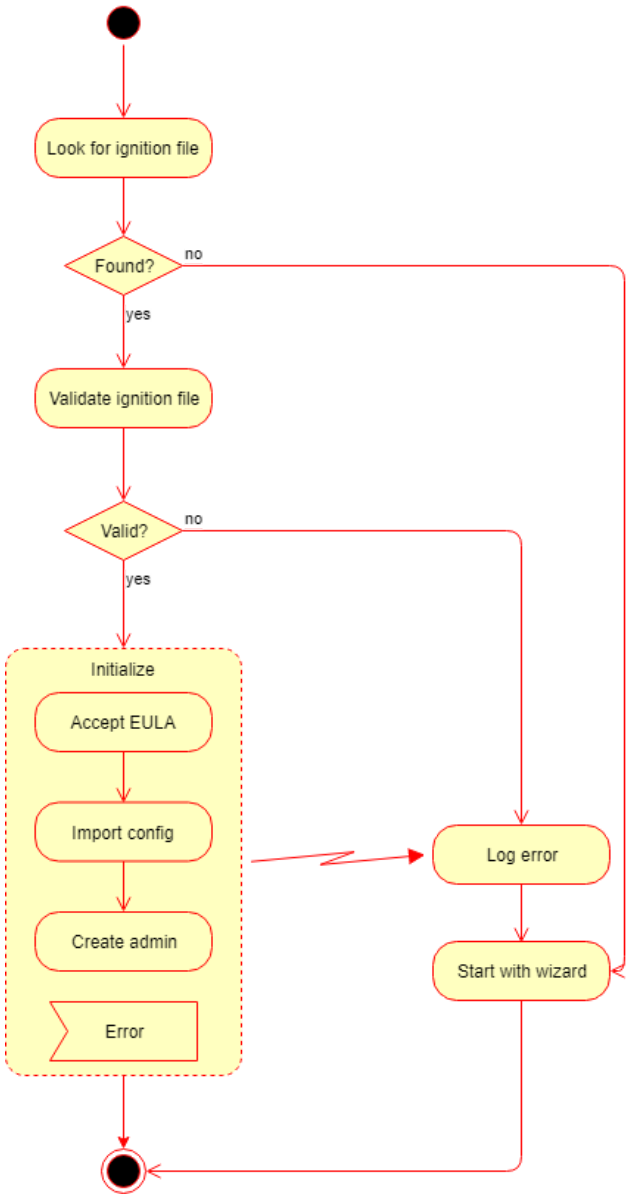
> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

2.2) If it is not valid, then the error is logged, and the initialization gets terminated.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

3) If there is no ignition file, then no initialization is performed.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.



Warning

If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried fixing the problem (e.g. placing the ignition file to its lookup location) and restarting the OPSWAT MetaDefender Core service.

Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

Configuration

After the initialization is complete, the product is ready with the default and the imported configuration.

This configuration can be later changed calling the configuration API functions. For further details about the API see Configuration related APIs API

CIS Level 1 Guidelines

Support Red Hat Enterprise Linux 9
For more details about Center for Internet Security (CIS) please refer to this document

Instruction steps

I. Install OpenSCAP

bash bash

```
yum install openscap-scanner scap-security-guide
```

II. Generate a result file and a HTML report using OpenSCAP scanner tool

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

III. Remediation of CIS Level 1 issues

Generate a remediation script based on the `ssg-rhel9-ds.xml` file:

bash

```
oscap xccdf generate fix --profile
xccdf_org.ssgproject.content_profile_cis_server_l1 --fix-type bash --output
remediations.sh /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

And execute remediation script.

bash

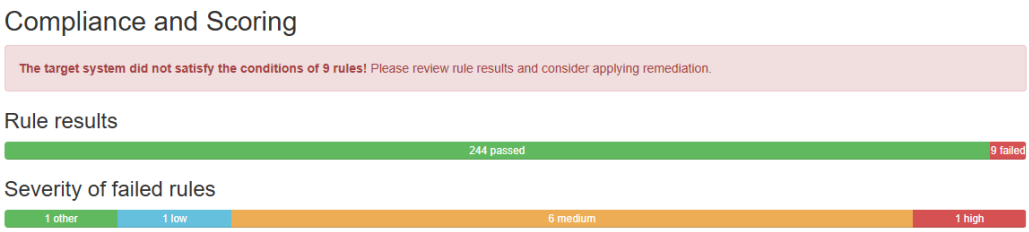
```
./remediations.sh
```

IV. Review the results after remediation

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

Example:



Notes

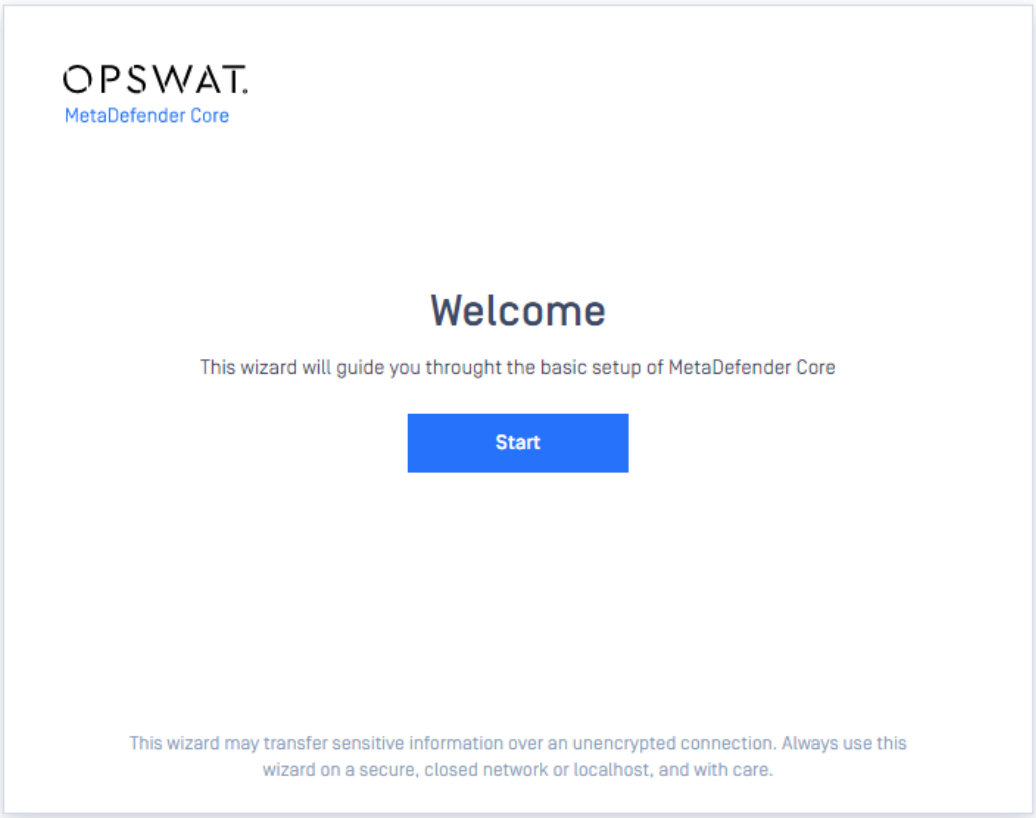
- CIS Level 1 requires `/tmp` folder to be mounted in a separate partition. Please ensure that that new partition have enough disk space for MetaDefender Core to run.
- CIS Level 1 requires that "Ensure No World-Writable Files Exist".
    - For now, when freshly installing MetaDefender Core, all its binary files meet the requirement.
    - When installing/updating engines, some engines might create additional files for its operation, and it might violate this requirement. In this case, you need to again execute the remediation script in the step III.

Wizard Setup

Introduction

Access the MetaDefender Core management console for the first time (i.e. http://localhost:8008), you are to complete a quick walkthrough for wizard setup in order to be able to use the product.

The administrative management console will be available only upon you have successfully finished this wizard stage.

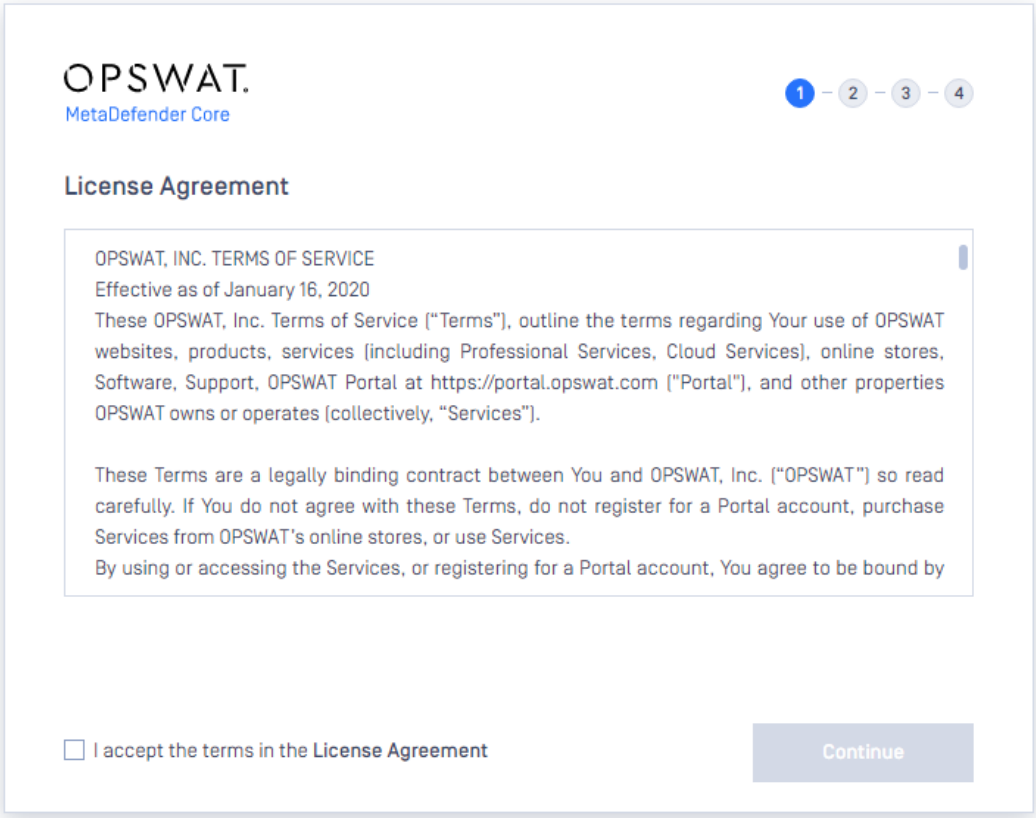


Warning

This wizard may transfer sensitive information over an unencrypted connection. Always use this wizard on a secure, closed network or localhost, and with care.

Basic configuration steps

End-User License Agreement



In the first page you can find the End-User License Agreement. You have to accept the terms before moving on.

Please read through the EULA (https://docs.opswat.com/policies/privacy-policy/portal-privacy-policy) carefully and if you agree with it, check `I accept the terms in the **License Agreement**` and click `Continue` to proceed.

Local Admin User Setup

The next step is to set up a default local administrator account.

This account will be the first one being able to access the administrative management console, and to create accounts for other users. You have to fill out all fields in this page to be able to move forward. When you are done, click NEXT to continue.

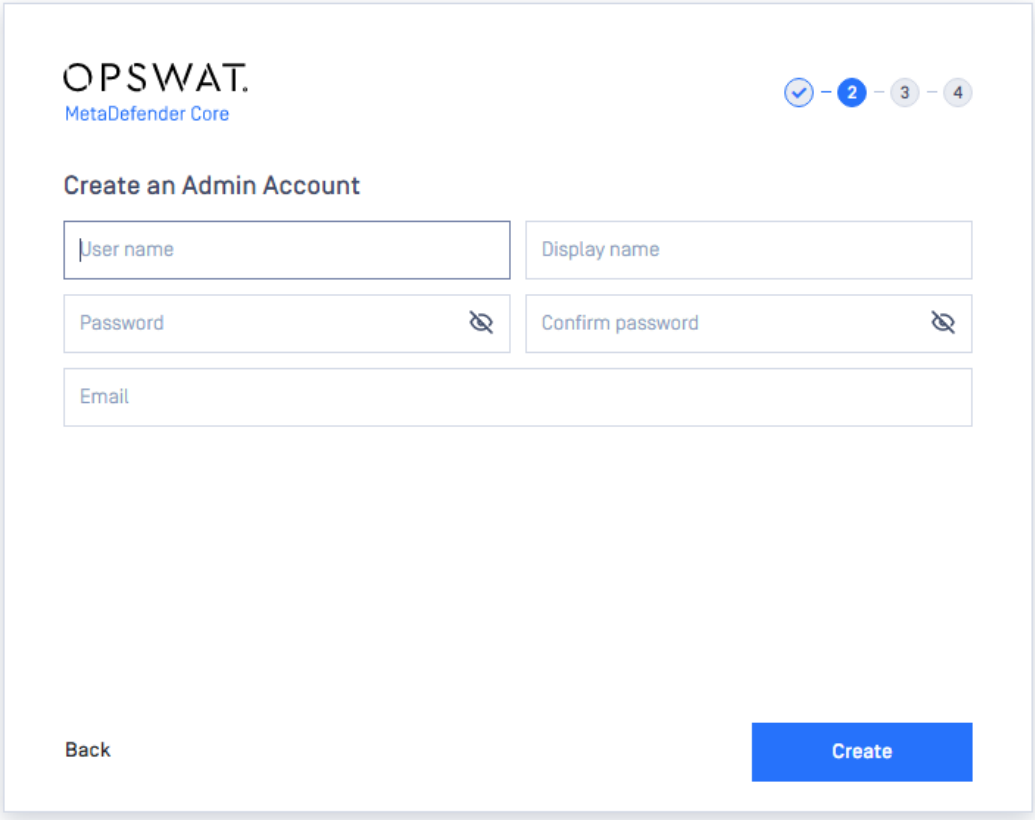
Info - User Directory

The administrator account, that is created via the basic configuration wizard, is always added to the LOCAL user directory as a member.

Learn more about user directory: User directories

The following information is required for the administrator account:

ACCOUNT NAME	The unique name of the account that is used at the time of login and in log messages for accountability.
ACCOUNT DISPLAY NAME	Name of the person bound to this account. This name (appended to the name of the account's user directory) is displayed in the top right corner of the administrative management console.
PASSWORD	Password of the user bound to this account that is used at the time of login. Passwords sent clear-text As long as TLS is not configured for the basic configuration wizard, passwords are sent clear-text over the network and may be disclosed to unauthorized parties. As a mitigation action: 1. Either use the wizard on *localhost* or on a direct network connection, or 2. Enabling HTTPS as soon as possible and change the password immediately if it has already been set.
EMAIL	Email address of the person bound to this account.



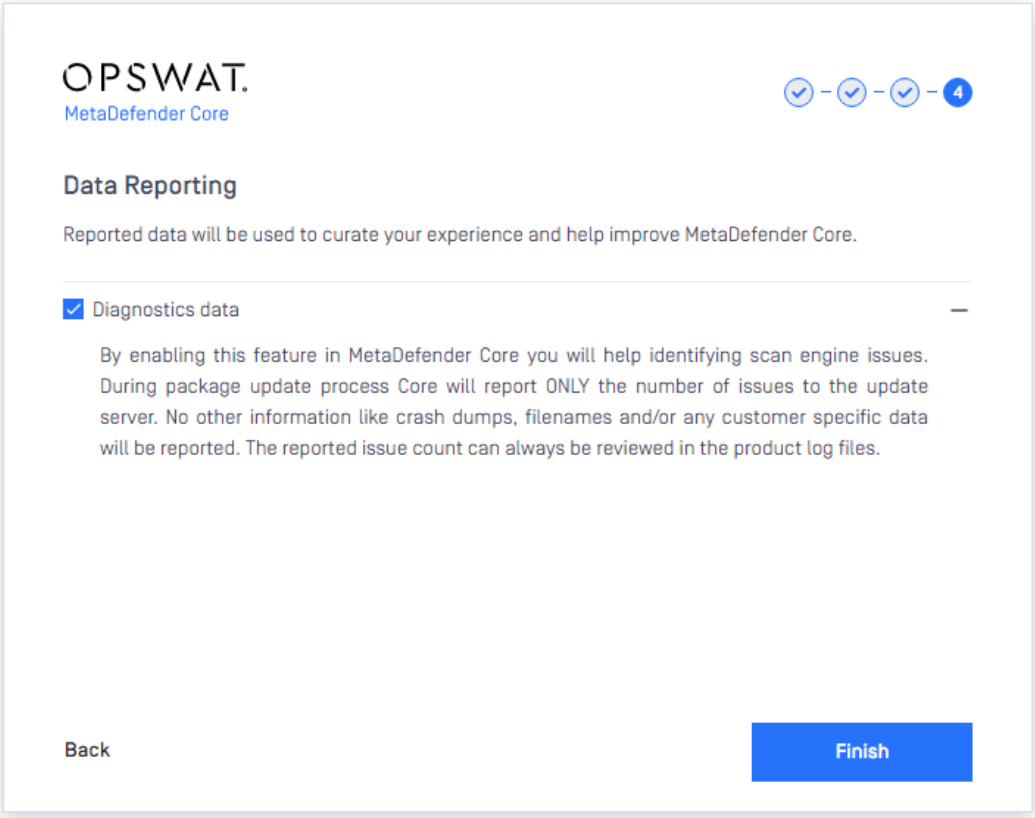
License activation

For license activation details see License Activation

Data Reporting

Enabling this option to allow MetaDefender Core collect its own engine issues and report back to OPSWAT for further analysis.

MetaDefender will not collect any other information.

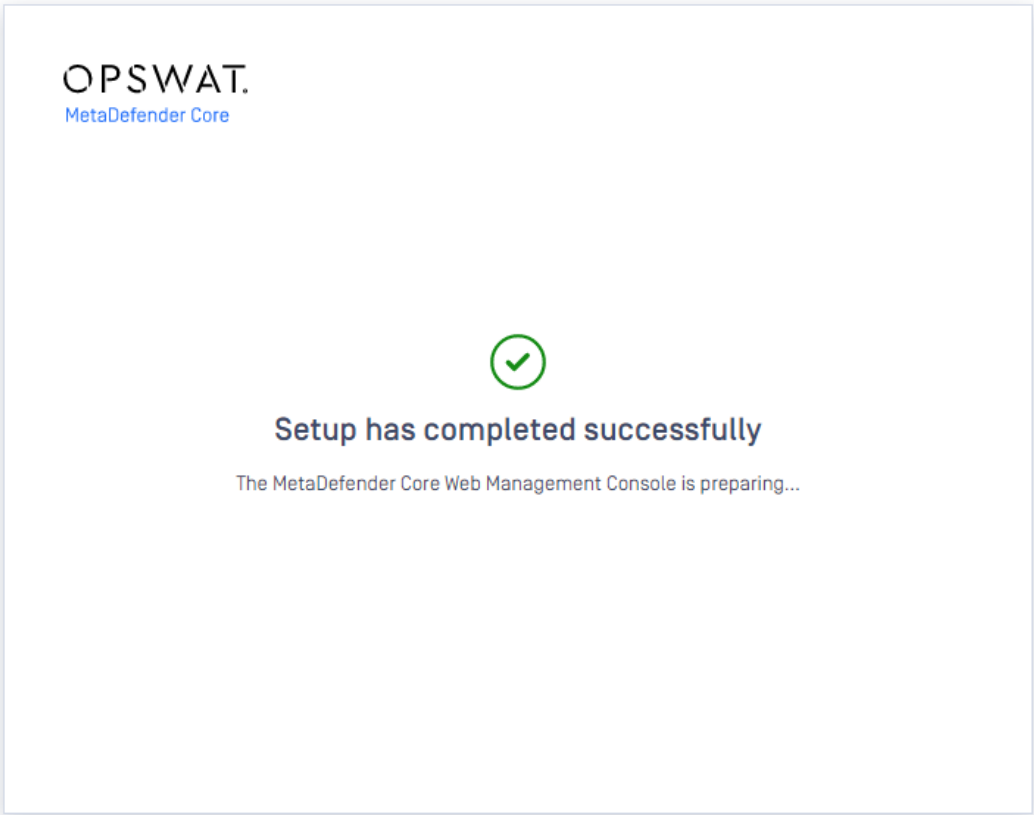


Wizard completion

After you have completed every steps you are ready to finish the wizard and start using the product. Click the `Fishish` button to complete the wizard.

> **Warning**
> The product's service will be restarted and the browser will be redirected to the administrative management console. This could take several seconds.

You can login to the Web Management Console with the administrator user that have just been created in the previous steps.



Setup HTTPS

Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network. Websites, like the Web Management Console, are able to use TLS to secure all communications between their servers and web browsers.

The TLS protocol aims primarily to provide confidentiality (privacy) and data integrity between two communicating computer applications.

Warning

No TLS for the wizard

By default, TLS is not enabled for the basic configuration wizard. As a consequence sessions between the wizard's backend and the browser may be insecure.

Info

Performing the same steps as for the Web Management Console, it is possible to set up TLS for the basic configuration wizard. Remember completing the TLS setup before launching the wizard.

For instructions to set up TLS see Enabling HTTPS

License Activation

##

To activate your installation go to the **Settings > License** menu in the Web Management Console. If you have no valid license, you will only see your installation's **deployment ID**. You will also see a warning in the Web Management Console header.

After successful activation, the product will start downloading the latest available scan engines and malware databases. You can follow the status of the scan engine installation on the **Inventory > Modules** page.

When all the licensed engines are installed, you can start using the MetaDefender Core to process files - File Processing.

Deployment ID

Each MetaDefender Core instance represented by an unique system fingerprint called `Deployment ID`

MetaDefender Core licensing is purely based on deployment ID activated. Therefore, everytime your MetaDefender Core is activated with a different deployment ID, then it will be captured as a different activation record accumulated into your total activation times allowed.

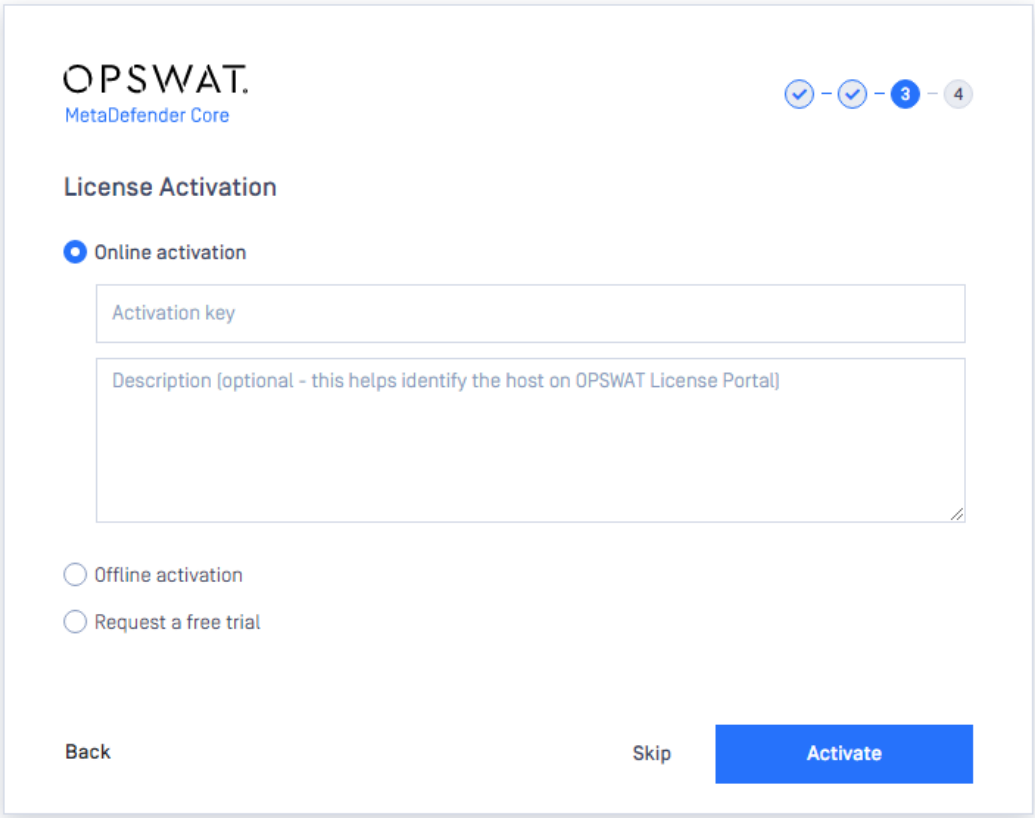
> **Notice**
> Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you alter the system where MetaDefender Core relies on (e.g. network card, OS hostname).
>
> Deployment ID is an UUID version 4 random based string. Learn more at: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you update the system where MetaDefender Core relies on.

Activation Modes

There are three different options for the product activation:

- Online activation: MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.
- Offline activation: Requiring you to upload a license file (.yml). That license file could be retrieved by activating your license via OPSWAT Portal. Follow the displayed instructions for details.
- Request trial key online: For evaluation purpose, you can receive an trial Activation key via email. Follow the displayed instructions for details.



Online activation

MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.

Required to input your Activation License key, and optional for description (but we recommend this for troubleshooting and auditing purpose if needed).

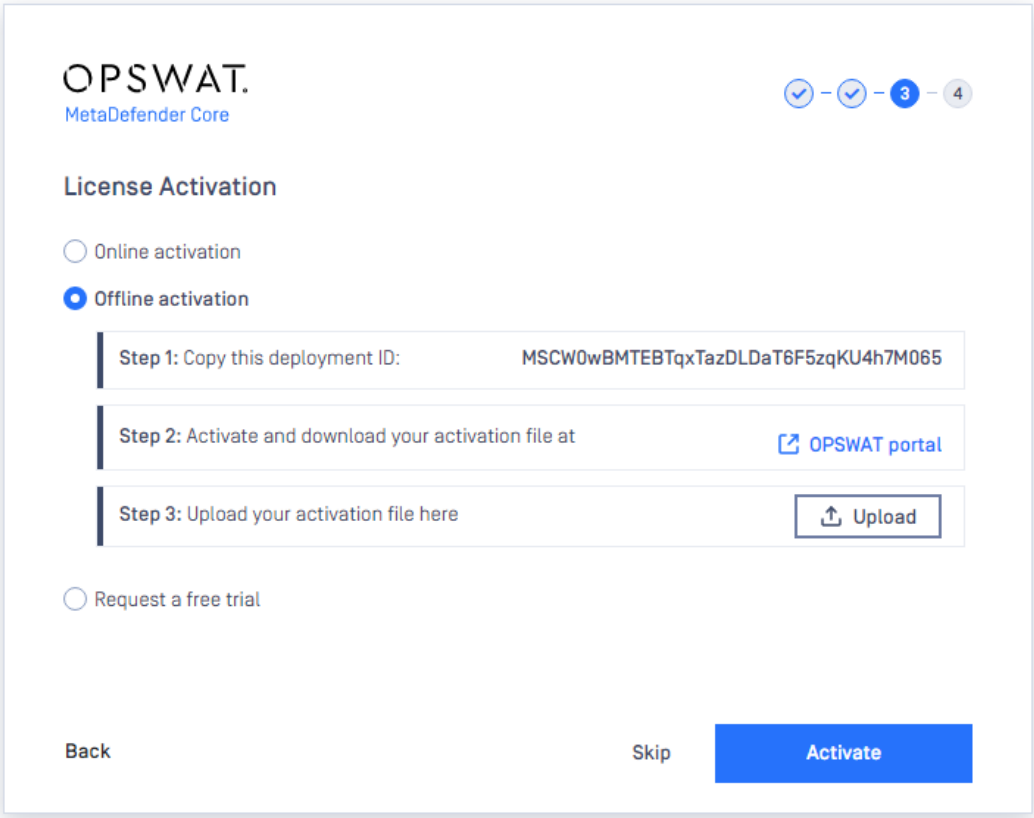
Warning

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts' for license activation:

- https://activation.dl.opswat.com Note: *IP address-based whitelisting on your firewall might fail after some time since OPSWAT uses CDN (Content Delivery Network) to faster delivery updates over the world, and IP address of edge servers might change over time.*

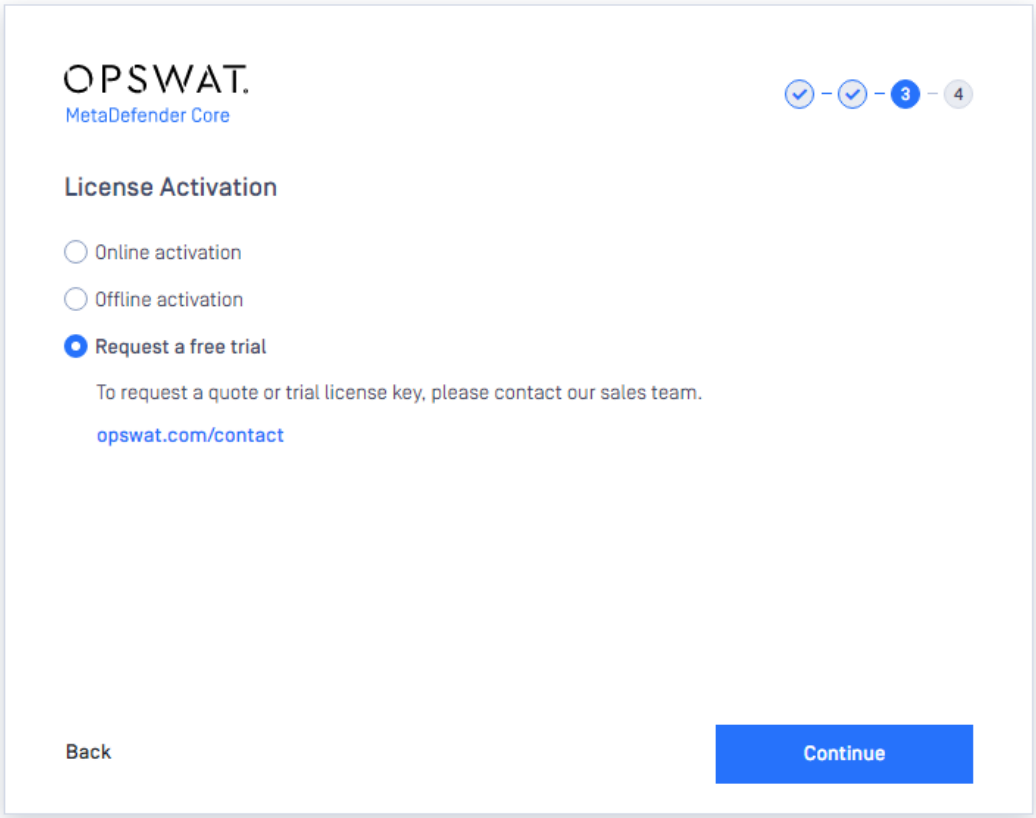
Offline activation

With no internet connection on the server the Metadefender v4 instance may be activated indirectly from a different machine, that has internet connection. The Deployment ID of the Metadefender v4 instance and the the Activation key received at the time of purchasing the product will be required. Follow the steps on the screen to activate the product offline.



Request trial key online

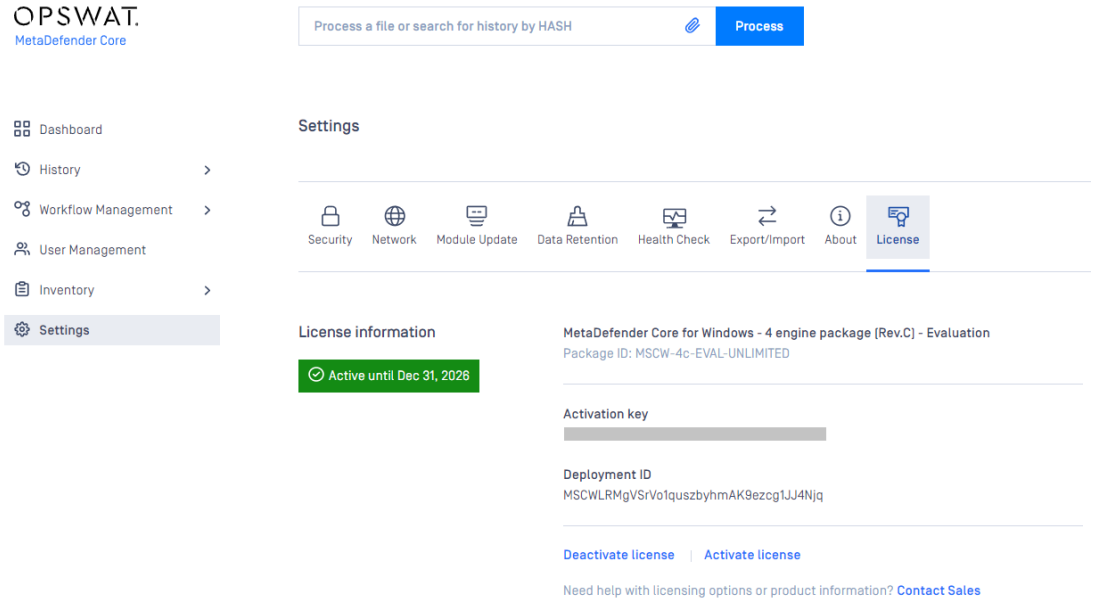
An evaluation license could be acquired by contacting our sales team.



Check Your License Details

For more license details and activating your MetaDefender Core, go to Settings > License menu tab on the Web Management Console:

- Product name: product name as on your order (e.g. MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation)
- Package ID: product identification as on your order (e.g. MSCW-4c-EVAL-UNLIMITED)
- Expiration: last day of license validity
- Activation key: key used for activating product
- Deployment ID: identification of this installation


OPSWAT.
MetaDefender Core
Process a file or search for history by HASH
Process
Dashboard
History
Workflow Management
User Management
Inventory
Settings
Settings
Security
Network
Module Update
Data Retention
Health Check
Export/Import
About
License
License information
Active until Dec 31, 2026
MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation
Package ID: MSCW-4c-EVAL-UNLIMITED
Activation key
Deployment ID
MSCWLRMgVSrVo1quszbyhmAK9ezcg1JJ4Njq
Deactivate license
Activate license
Need help with licensing options or product information? Contact Sales

Scripted license management

Using REST API calls there is an option for scripted activation and deactivation for MetaDefender Core servers with **no Internet connection or if your infrastructure is using automation to create/destroy MetaDefender Core instances**. In this chapter the steps of these two scenarios are described.

Requirements

- an installed MetaDefender Core instance without Internet connection
- another computer that has Internet connection and can run your activation scripts
- a manual or automated way to transfer data between the two computers

Activation steps

1. In order to activate MetaDefender Core, deployment ID and activation key are needed to generate the license file.
    a. Activation key should be purchased from OPSWAT.
    b. MetaDefender Core deployment ID can be queried by using REST API (for details see GET Get Current License Information API page)

> **Info**
> Save this deployment ID in your system, you might need this ID when the instance is unavailable at the time of deactivation.

2. Activate license and get the license file using the following URL, where <quantity> is 1.

activation call bash

```
curl -X GET https://activation.dl.opswat.com/activation?key=<activation-key>&deployment=<deployment unique-ID>&quantity=<quantity>
```

If the activation is successfully, a license file is downloaded. Save this file.

HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	license file	Activation was successfully.
200 Activation failed	error: '<user conform error message>'	Failed activation
200 Invalid parameter	error: 'Could not activate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not activate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
200 Invalid parameter	error: 'Could not activate your product because the quantity you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key, quantity or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Info
In case of any activation issue, contact OPSWAT support for help

3. Upload license file to MetaDefender Core: The license file should be uploaded to the MetaDefender Core v4 to activate the product. For details see page: POST Upload license key API

Deactivation steps

1. For deactivation, following infomation is required:
    a. Activation key
    b. MetaDefender Core deployment ID: can be queried by using REST API (for details see GET Get Current License Information API page)
2. Deactivate license using the following URL:

deactivation call bash

```
curl -X GET https://activation.dl.opswat.com/deactivation?key=<activation-
key>&deployment=<deployment unique-ID>
```

Response HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	result: ok	Successful deactivation
200 No active license found	error: 'Could not found any active license with the given parameters'	The license has not been activated yet or it has been deactivated already.
200 Invalid parameter	error: 'Could not deactivate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not deactivate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Important notes

Warning

Product activation is tied to several hardware, operating system and software parameters. In case of one or more major hardware or software parameter change the product might turn into deactivated status. Operating system updates, other software updates should not affecting the activation status.

If the product is online activated and has live Internet connection then in case of deployment ID change the product reactivates itself. If the product doesn't have Internet connection then the administrator is responsible to reactivate the product.

Because of the above administrator should consider licensing restrictions of the offline deployments if using any virtualization and/or containerization technologies.

File Processing

MetaDefender Core provides REST API interface for file processing business.

There are several ways to process files with MetaDefender Core:

- Analyze Files via Web Interface
- Process Files via REST API

Product Upgrading

> **Info**
> Simply run installer file, details could be found at Installation

> **Info**
> For command line mode, upgrading (e.g. on **Linux**), prior to running the installer file, please make sure to **create the ignition file** with PostgreSQL server information included. Details:
>
> - Standalone DB mode: Command Line Installation (Windows and Linux)
> - Shared DB mode: Command Line Installation

Recommendation for MetaDefender Core upgrade

Our goal is to help you execute the product upgrade continuously, efficiently, and correctly.

PostgreSQL service upgrade

- **Bundled PostgreSQL Service**: If you are using the bundled PostgreSQL server included in the installer, the PostgreSQL service will be automatically upgraded to the newer version (if available). No additional action is required for this process.
- **Remote PostgreSQL Service**: For remote PostgreSQL servers, customers are responsible for managing and upgrading the service independently. Upgrading MetaDefender Core does not update the remote PostgreSQL service.

Before the upgrade

Prepare a proper disaster recovery plan (DRP) including some common practices:

- Backup, or take a snapshot of your current working MetaDefender Core server.
  - This is to help quickly rollback for any upgrade failure might cause later for any reason.
- Optionally backup your PostgreSQL database, especially when using remote shared database mode.
  - Reference: https://www.postgresql.org/docs/current/backup.html
  - Please do it after stopping all linked MetaDefender Core instances to avoid potential data collision.
- Lastly, if backing up entire server is not feasible, then at least export MetaDefender Core configuration package Import/Export configuration just in case you'll need to start off a new server with current MetaDefender Core configuration to keep the scanning service running without any impact.

On the PostgreSQL server, double check server resource to ensure sufficient RAM memory and disk space to be preserved for the upgrade process.

- 8GB RAM available at least during the upgrade.

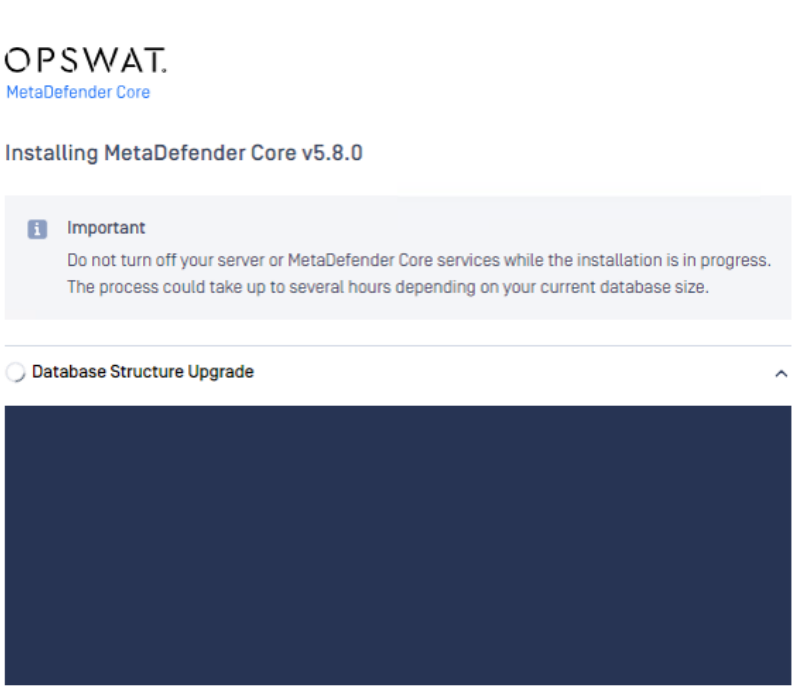
- Disk availability is greater than MetaDefender Core database size.

While doing the upgrade

Applicable for shared database deployment:

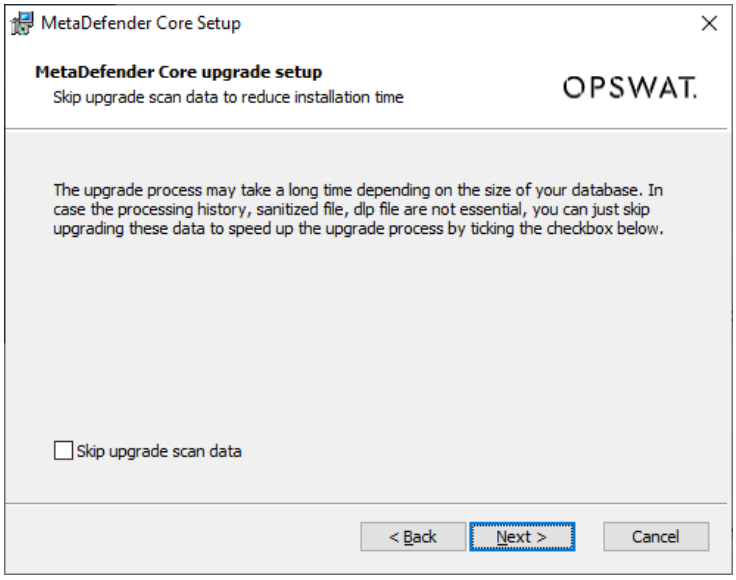
- Stop all MetaDefender Core instances' scanning service before performing the upgrade.
- Upgrade one (1) MetaDefender Core instance at a time.
- Proceed to upgrade MetaDefender Core instance where the bundle local PostgreSQL server is used for shared database setup.
- Otherwise is using remote PostgreSQL then the upgrade order does not matter.

For large size database, mostly for scan history, expecting long time to be spent on the upgrade process.



The product supports an option to skip data migration for scan history to expedite the process drastically and mitigate any failure risk. You can manually backup all data from your PostgreSQL server for offline usage if any.

Check "Skip upgrade scan data" to enable this feature while doing the upgrade.



or add following key in the ignition file:

none

```
[internal]
skip_upgrade_scan_data = 0
```

Troubleshooting

If the upgrade process ever runs into an error:

- Double check your network and services, applicable to remote PostgreSQL server.
- Optionally configure MetaDefender Core to skip migrating scan history.
  - Windows
    - Open Windows registry, and navigate to HKLM\Software\OPSWAT\Metascan
    - Add a new key `internal/skip_upgrade_scan_data` with value = 1
  - Linux
    - Open MetaDefender Core configuration file `/etc/ometascan/ometascan.conf`
    - Add new key into the file
      - [internal]
      - skip_upgrade_scan_data=1
  - Note: This key will be auto removed upon the upgrade finishes.
  - Then restart MetaDefender Core service to retry the upgrade.
    - Might retry this a few times until the upgrade is done.

Just in case the upgrade process still fails, please contact OPSWAT support with the support package - How to Create Support Package? This will help us analyze the root cause and figure out an optimal solution for your system failure.

Using external Load Balancer

Application Load Balancing (Layer 7)

This page provides information about recommended ways to use sessions with a Layer 7 load-balancer.

Most load-balancers have the ability to provide cookies for pinning subsequent traffic from a client to the appropriate server. This method is called sticky session, session persistence or session affinity. Using cookies efficiently requires the client to know when a cookie should be sent, should not be sent or should be deleted.

Sticky session load balancing

Each Metadefender Core instance has its own databases and application sessions that cannot be seen by other instances. Therefore, in order to get the related data/response to our queries we should ask the appropriate Metadefender Core v5 server. On the other hand, to keep the advantages of the used load balancing method, cookies should not be sent if it is not necessary.

Single file scanning

Step	Stage	Task	Cookie usage
1.	Sending file	Initiate processing a file on the client side. Send file through the load-balancer. (See POST File Analysis API)	Cookie should not be sent
2.	Sending file	Save the cookie and the data_id you got from the load-balancer	Save cookie
3.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Send cookie saved in step 2
4.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Send cookie saved in step 2

Batch scanning

Step	Stage	Task	Cookie usage
1.	Open batch	Initiate processing file(s) in batch. Request a batch ID through the load-balancer. (See POST Initiate Batch API)	Cookie should not be sent
2.	Open batch	Save the cookie and the batch_id you got from a Core server through the load-balancer.	Save cookie
3.	Sending files	Send file through the load-balancer. (See POST File Analysis (using batch header) API)	Send cookie saved in step 2
4.	Sending files	Save the data_id you got from the load-balancer.	-
	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Send cookie saved in step 2
5.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
6.	Getting batch status	See GET Batch Status API	Send cookie saved in step 2
7.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Send cookie saved in step 2
8.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API	Send cookie saved in step 2

If it does not matter which upstream server responds, the queries should be sent without cookie.

> **Info**
> It is recommended not to send cookies when it's not necessary to allow load-balancer to use its own method to share the load between Metadefender Core v5 servers.

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration through load-balancer

OPSWAT products that support HTTP load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	4.3.4	-
OPSWAT Client	Windows: 7.6.247.0 Mac: 10.4.243.0	-
MetaDefender Email Security	4.3.0	-
MetaDefender ICAP Server	4.3.0	-
MetaDefender Vault	1.3.0	-
MetaDefender for Secure Storage	2.0.7	-

DNS Load Balancing

Using this method is logically similar to Layer 7 load-balancing.

Briefly how it works

Client uses a domain name to send a query to a server. Client's DNS server has more "A" records for that name with different IPs. When a client resolves the server's domain name DNS server randomly chooses an IP for that name to send back. When a session is used on the application layer, client should know the IP address of the Core server that handles that specific session. In every other case, client should resolve the domain name with DNS query to let requests to be balanced between Metadefender Core servers.

Single file scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Sending file	Initiate processing a file on the client side. (See POST File Analysis API)	Use IP saved in step 2
4.	Sending file	Save the data_id got from the Core server	-
5.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Use IP saved in step 2
6.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Use IP saved in step 2

Batch scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Open batch	Initiate processing file(s) in batch. Request a batch ID. (See POST Initiate Batch API)	Use IP saved in step 2
4.	Sending files	Send file to the specific Core server with the basch ID saved in step 3. (See POST File Analysis (using batch header) API)	Use IP saved in step 2
5.	Sending files	Save the data_id you got from the Core server.	-
6.	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Use IP saved in step 2
7.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
8.	Getting batch status	See GET Batch Status API	Use IP saved in step 2
9.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Use IP saved in step 2
10.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API).	Use IP saved in step 2

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration via DNS load-balancing

OPSWAT products that support DNS load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	does not support yet	-
MetaDefender Client	does not support yet	-
MetaDefender Email Security	does not support yet	-
MetaDefender ICAP Server	does not support yet	-
MetaDefender Vault	does not support yet	-

SSL connection for PostgreSQL communication

> **Note**
> The SSL connection configuration is possible thanks to the support made by PostgreSQL. For reference: https://www.postgresql.org/docs/14/ssl-tcp.html
>
> MetaDefender Core is tested against this mode on particular version 4.21.1

Step 1: Generate SSL certificate for PostgreSQL server

> **Note**
> Following instructions are for self-sign certificate. For production environment, making sure to use the certificate signed by your trusted CA.

1) Generate a private key

bash

```
openssl genrsa -des3 -out server.key 1024
```

Sample output:

bash

```
Generating RSA private key, 1024 bit long modulus ........++++++
...............++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

Type your desired password / pass phrase to encrypt the private key

2) Remove the pass phrase to automatically start up the PostgreSQL server

bash

```
openssl rsa -in server.key -out server.key
```

> **Note**
> On Linux, make sure to set permission on the server.key file

bash

```
chmod og-rwx server.key
```

3) Create a self-signed certificate

bash

```
openssl req -new -key server.key -days 3650 -out server.crt -x509
```

You will be prompted to enter detailed information which is incorporated into your self-signed certificate request.

none

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:CA
Locality Name (eg, city) [Default City]:San Francisco
Organization Name (eg, company) [Default Company Ltd]:OPSWAT
Unit Name (eg, section) []:OPSWAT
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:dummy_email@opswat.com
```

For self-signed certificate, use the server ceriticate as the trusted root certificate:

bash

```
cp server.crt root.crt
```

Copy **server.key**, **server.crt**, **root.crt** to PostgreSQL data folder. For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data**
- Linux: **/var/lib/ometascan/pg_data/**

Step 2: Configure PostgreSQL server for SSL authentication connection

1) Create a custom config file (e.g. ssl.conf) for PostgreSQL.

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\postgres\ssl.conf**
- Linux: **/var/lib/ometascan/postgres/ssl.conf**

2) Content of ssl.conf:

none

```
ssl = on
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
```

3) Modify pg_hba.conf

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data\pg_hba.conf**
- Linux: **/var/lib/ometascan/pg_data/pg_hba.conf**

Add following (modify **::1/128** if needed to match with your environment requirement)

none

```
hostssl all all      ::1/128      scram-sha-256
```

After the modify, the content of pg_hba.conf should be like this (example):

> **Note**
> In a Linux environment, a local connection is required to upgrade from version 5.5.1 or earlier to version 5.6.0 or later.

none

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local         DATABASE  USER  METHOD  [OPTIONS]
# host          DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl     DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostgssenc    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnogssenc  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
local    all all                    scram-sha-256
hostssl all all      ::1/128      scram-sha-256

#More content.......
```

4) Reload the services

- Standalone DB mode: Restart **ometascan** service (MetaDefender Core service)
- Shared DB mode: Restart **ometascan-pg** / **ometascan-postgresql** service (MetaDefender Core PostgreSQL service) and then **ometascan** service (MetaDefender Core service)

Test your SSL database connection:

bash

```
psql.exe "sslmode=require dbname=postgres user=postgres host=localhost port=5432"
Password for user postgres:
psql (12.7)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
Type "help" for help.
```

This message indicate that you are now using SSL connection:

bash

```
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
```

Performance and Load Estimation

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Some factors that affect performance

- MetaDefender Core version
- MetaDefender Core engine package and configuration
  - set of engines (which and how many)
  - product configuration (e.g., thread pool size)
- System environment
  - server profile (CPU, RAM, hard disk)
  - client application location - remote or local
  - system caching and engine level caching
- Dataset
  - encrypted or decrypted
  - file types
    - different file types (e.g., document, image, executable)
    - archive file or compound document format files
  - file size
  - bad or unknown (assume to be clean)
- Performance tool

How test results are calculated

Performance (mainly scanning speed) is measured by throughput rather than unit speed. For example, if it takes 10 seconds to process 1 file, and it also takes 10 seconds to process 10 files, then performance is quantified as 1 second per file, rather than 10 seconds.

- total time / total number of files processed: 10 seconds / 10 files = 1 second / file.

Standalone DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.2.416.0

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

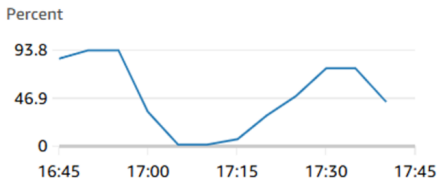
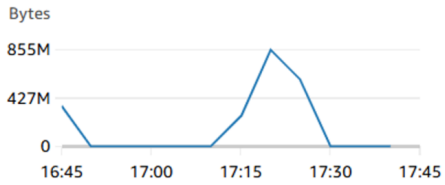
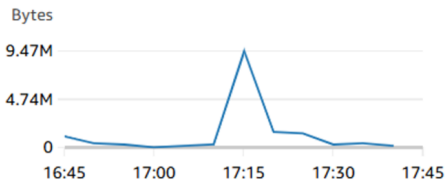
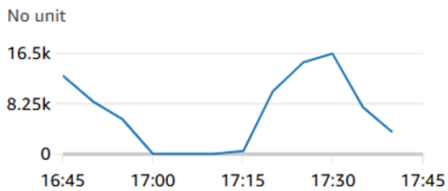
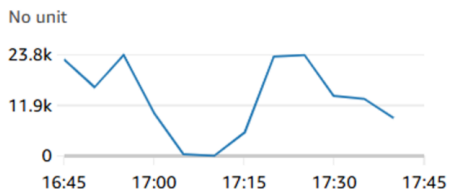
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV)	24.2	47.2 / 80	37.3 / 41.1		315,310	0.0114
Deep CDR	19.9	48.9 / 81.8	32.6 / 37		383,442	0.0094
Proactive DLP	17.1	36.8 / 58.2	30.4 / 33.6		446,228	0.0081
Vulnerability	10.1	59.2 / 85	29.3 / 33.3		755,495	0.0048

MetaDefender Core with common engine packages:

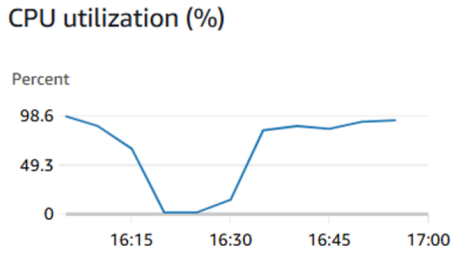
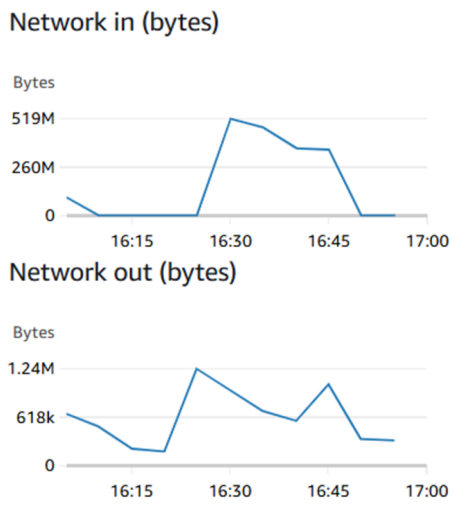
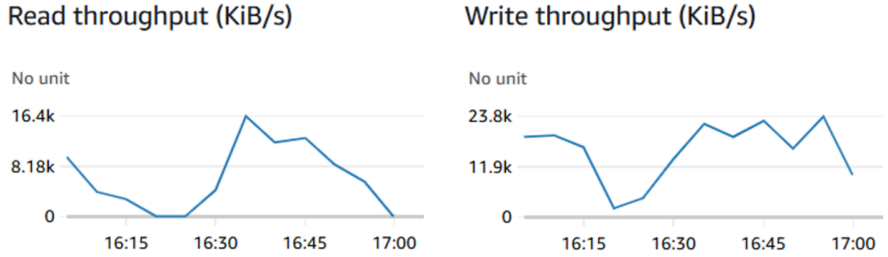
Use case	Scan duration	Avg./Max CPU usage	Avg./Max RAM usage	Avg./Max Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV) + Deep CDR	30.8	68.1 / 88	40.1 / 46.8		247,744	0.0145
Metascan (8 AV) + Deep CDR + Proactive DLP	31.2	74.2 / 98.6	41.5 / 51.1		244,567.3	0.0147
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	31.8	73.5 / 98	40.6 / 51.5		239,953	0.015

System resource utilization charts

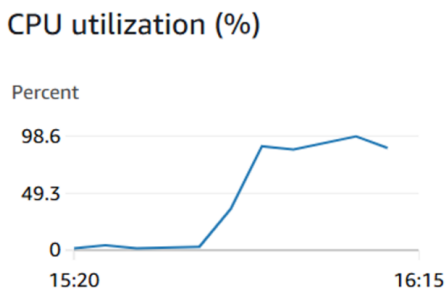
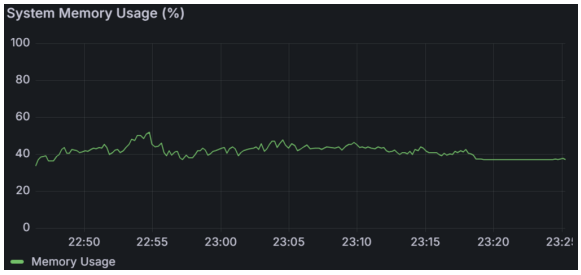
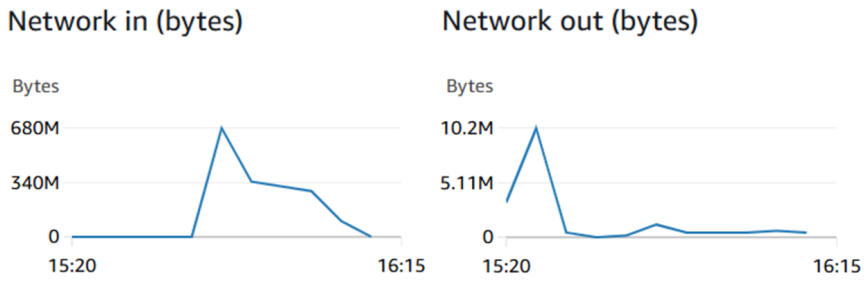
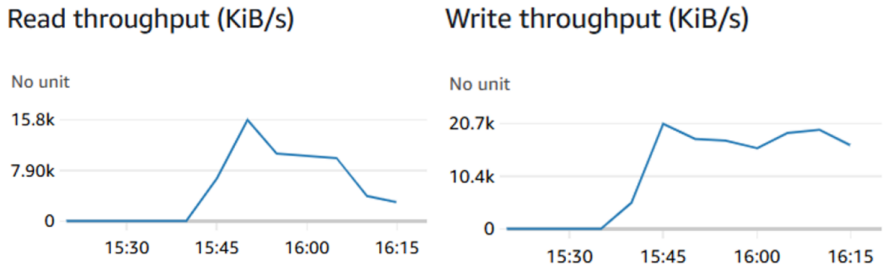
MetaDefender Core with Metascan (8 AV engines):

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

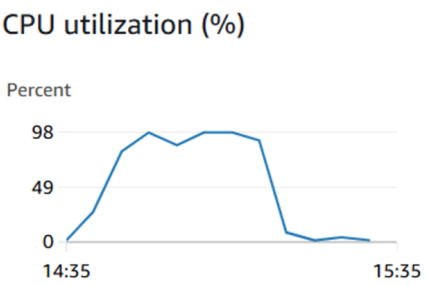
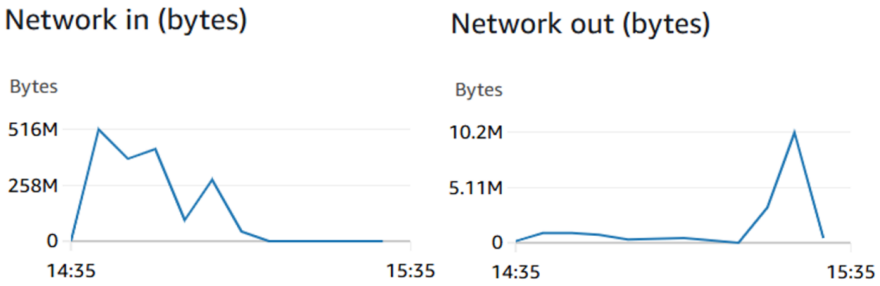
MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

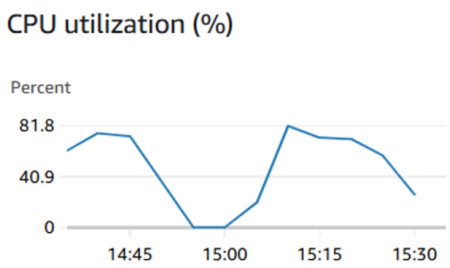
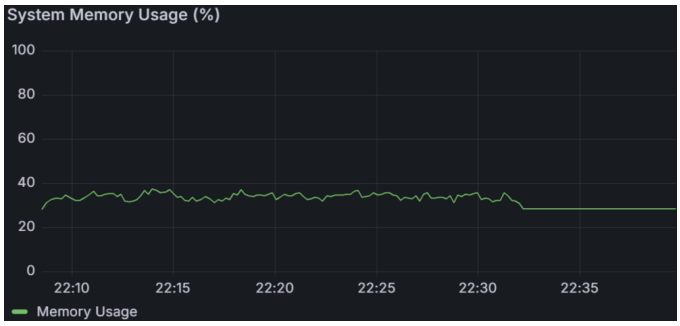
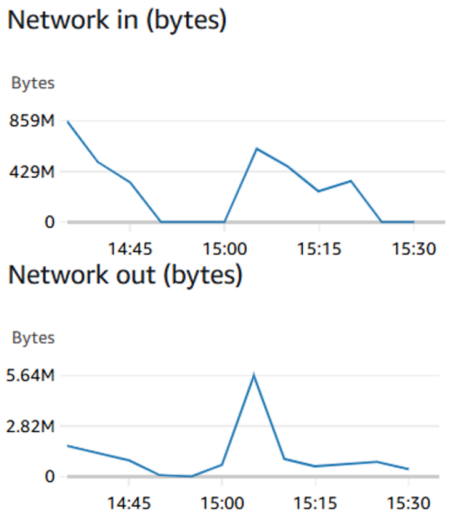
MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

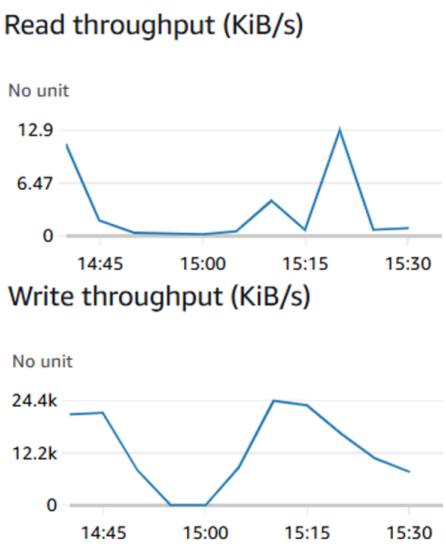
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

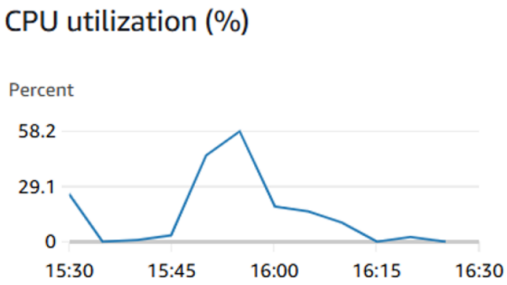
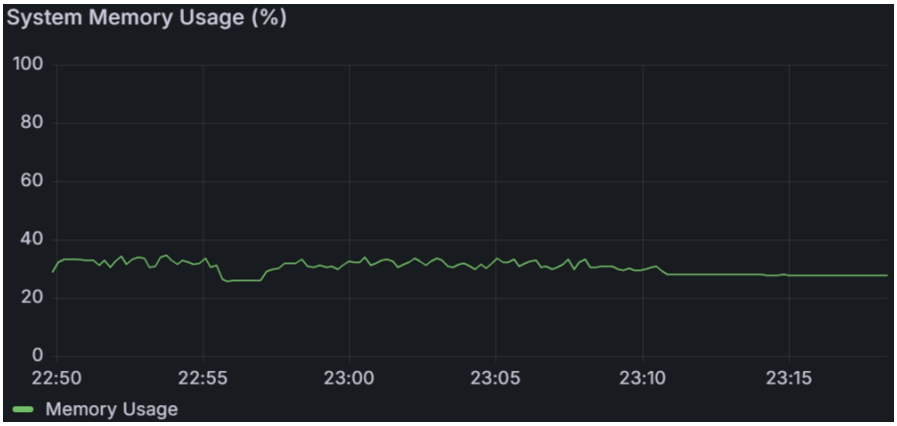
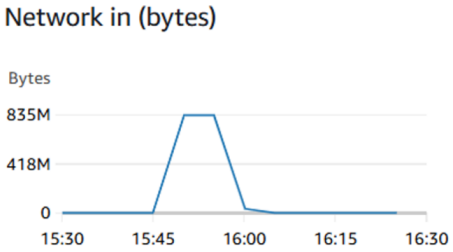
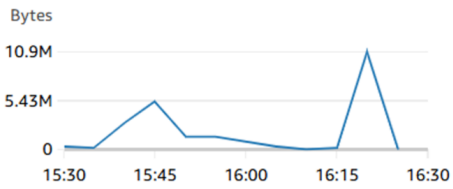
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

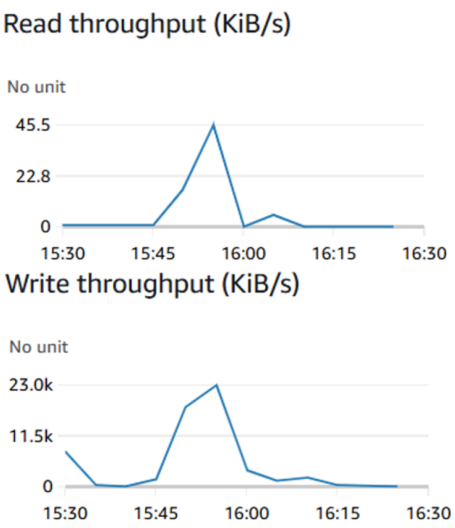
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

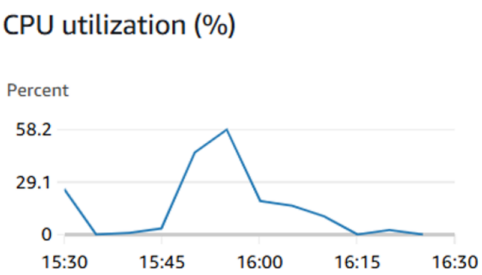
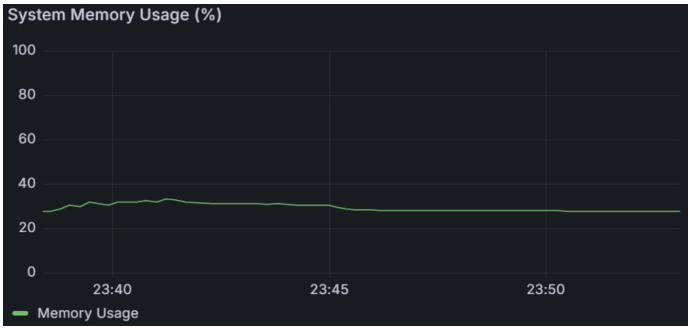
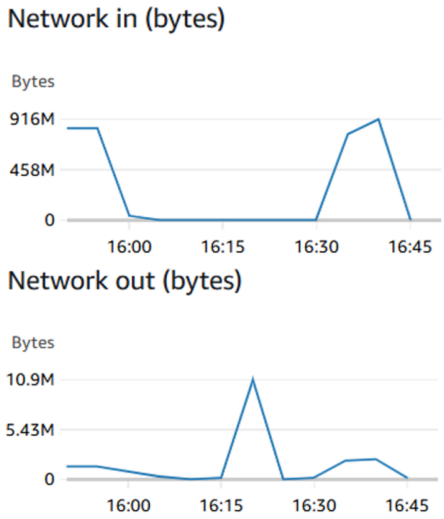
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	 

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability:

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

Standalone DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Ubuntu 22.04	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.57-236

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

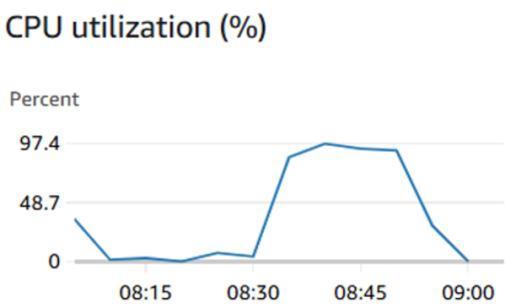
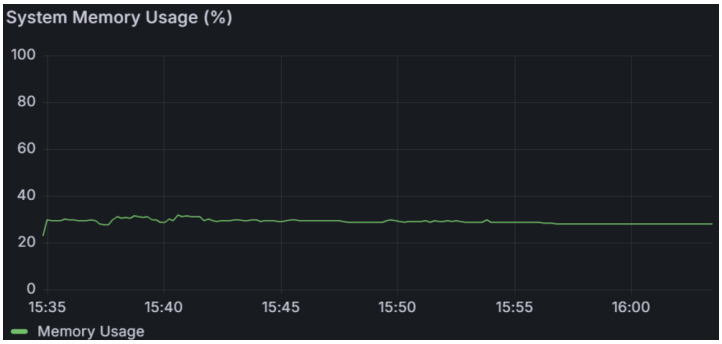
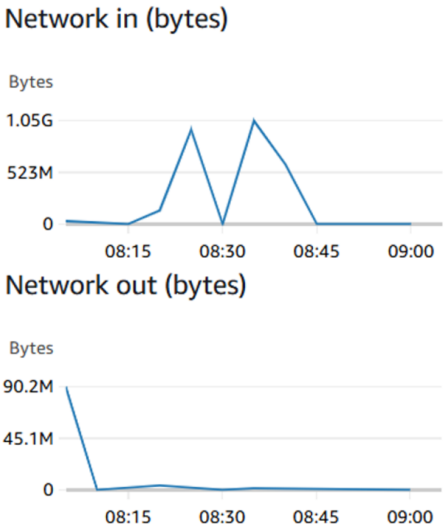
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV)	30.2	69 / 98	29 / 32		252,666	0.0142
Deep CDR	19.4	48 / 62	36 / 49		393,325	0.009
Proactive DLP	16.5	34 / 39	32 / 48		462,455	0.008
Vulnerability	12	50 / 65	46 / 46		635,875	0.006

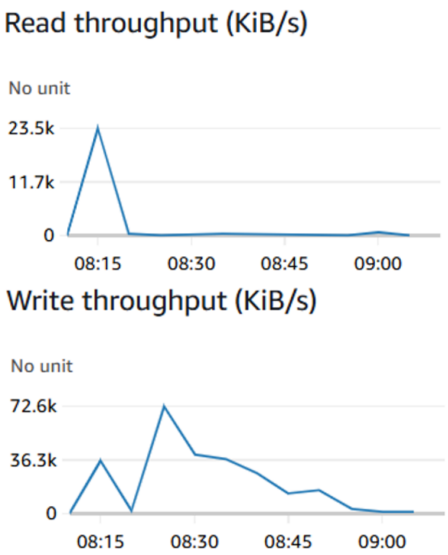
MetaDefender Core with common engine packages:

Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV) + Deep CDR	39.8	78 / 99	62 / 73		191,721	0.0188
Metascan (10 AV) + Deep CDR + Proactive DLP	43.5	61 / 84	47 / 67		175,414	0.0205
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	44.8	62 / 88	69 / 83		170,324	0.0211

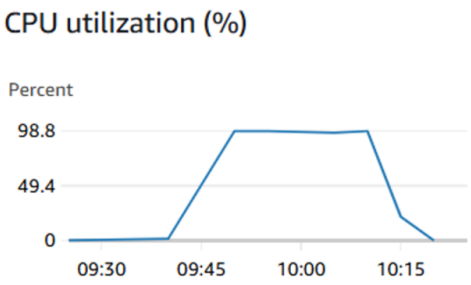
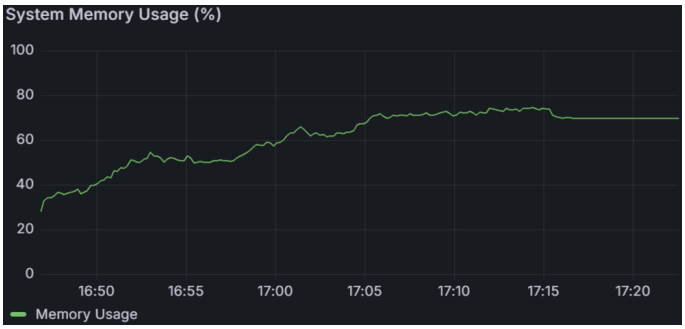
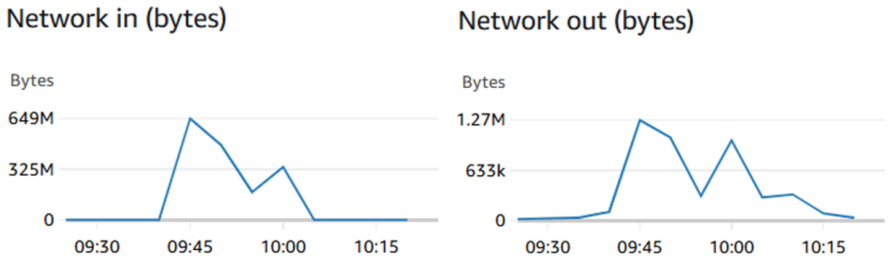
System resource utilization charts

MetaDefender Core with Metascan (10 AV engines) :

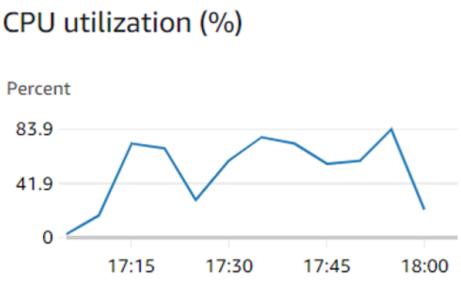
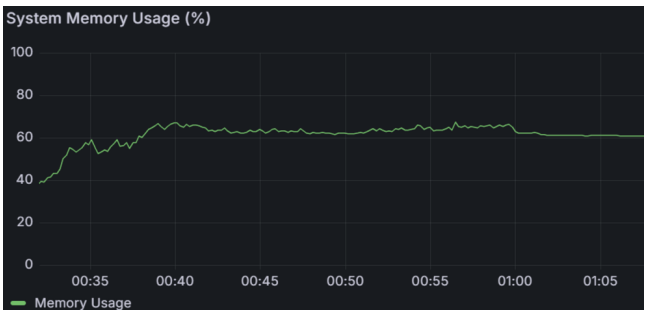
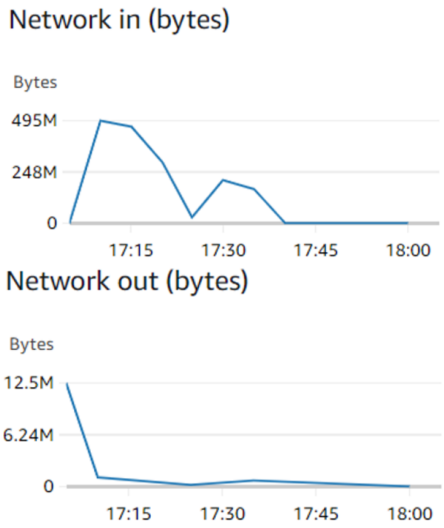
	Utilization charts
CPU usage	
Memory usage	
Network usage	

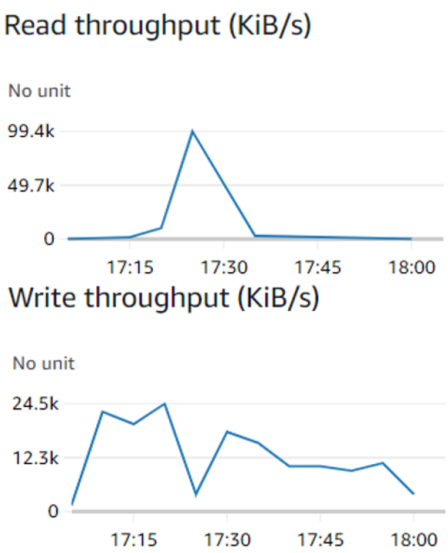
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR :

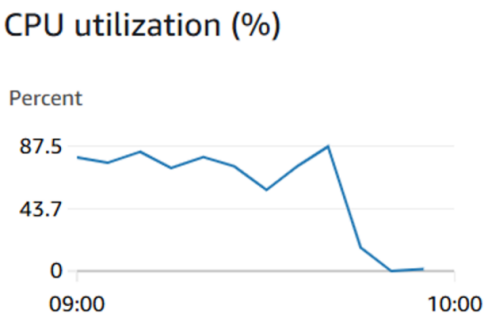
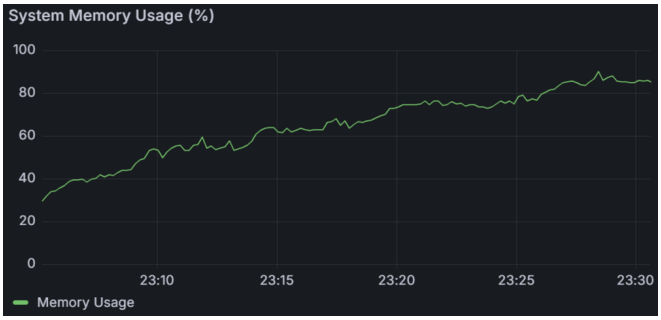
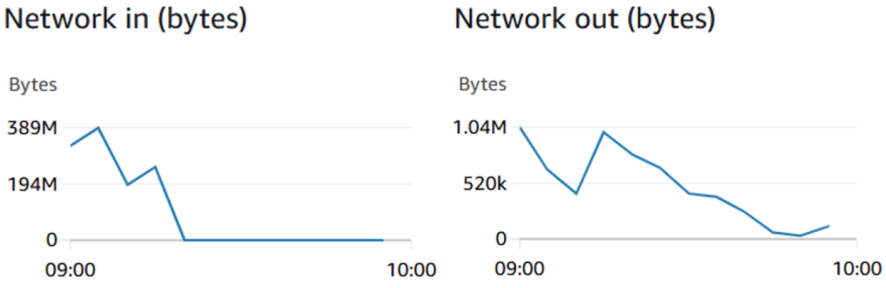
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP :

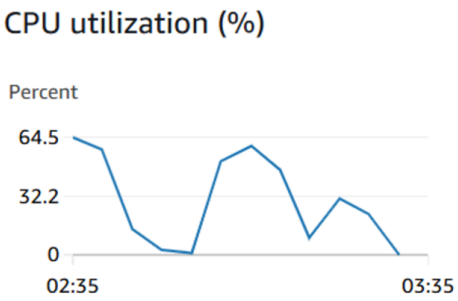
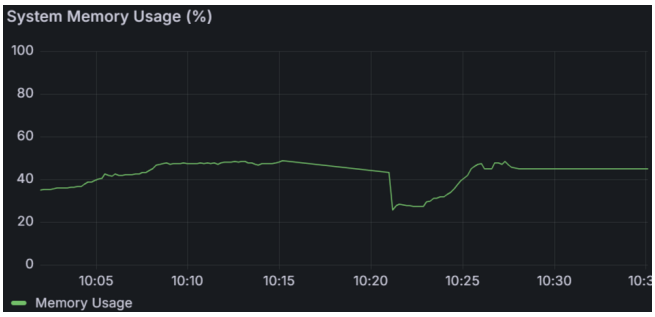
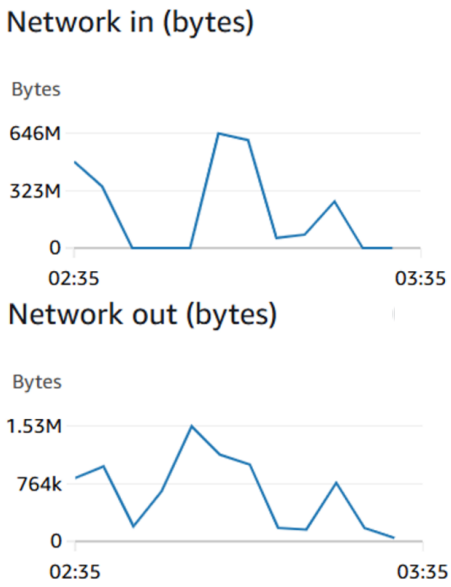
	Utilization charts
CPU usage	
Memory usage	
Network usage	

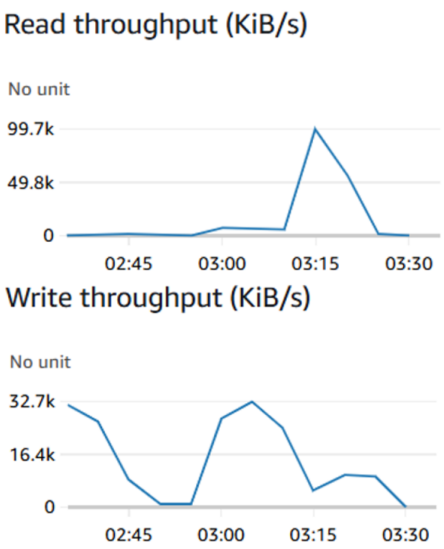
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP + Vulnerability :

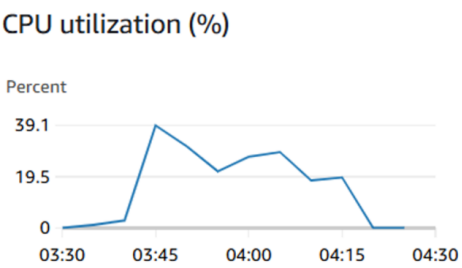
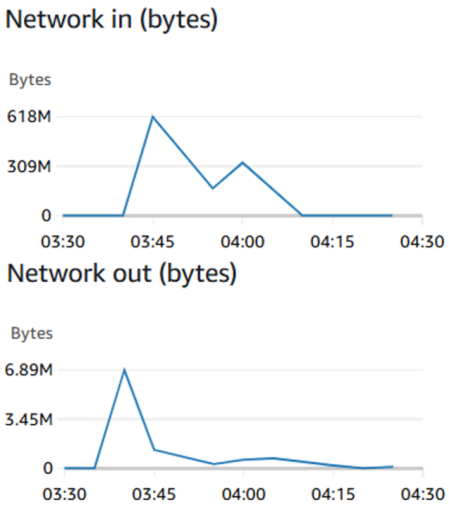
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

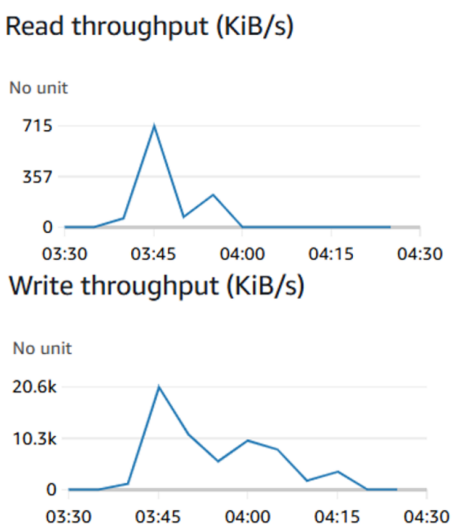
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

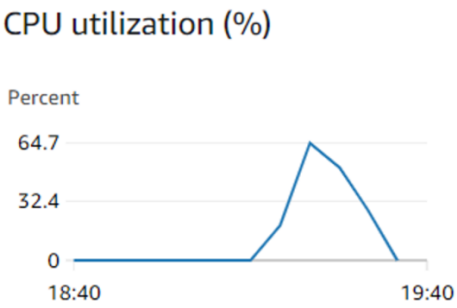
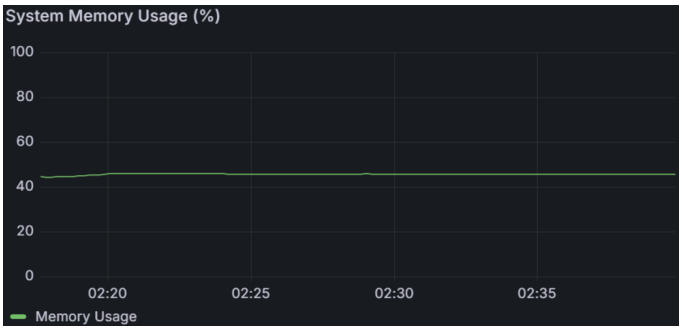
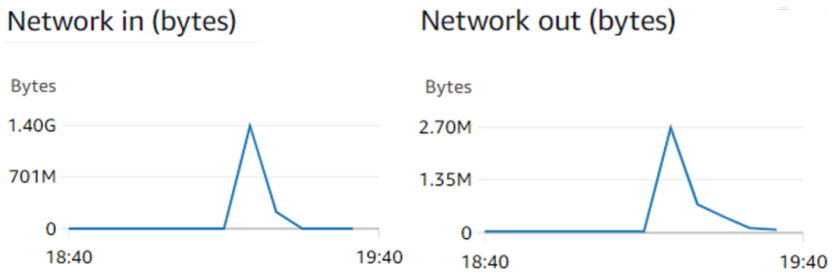
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability :

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

Shared DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

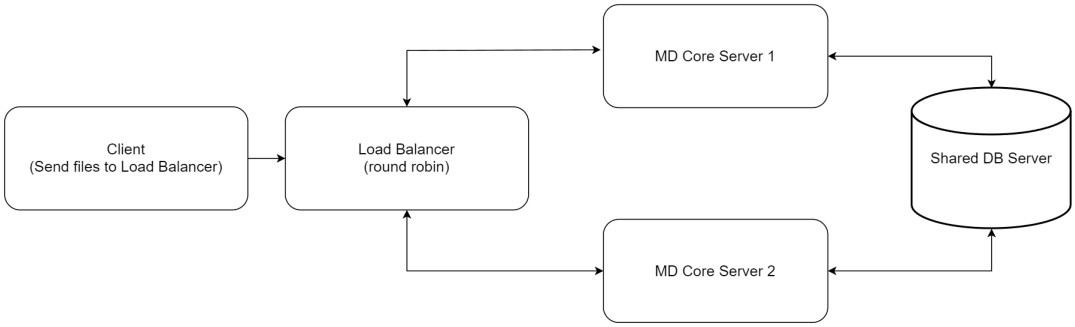
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV)	11.9	37.0	41.9	30.1	30.4	Receive: 6441.1 Send: 769.6	Receive: 6200.2 Send: 693.7	640,821.2	0.0
Deep CDR	10.5	60.1	66.7	30.3	32.2	Receive: 6379.8 Send: 594.8	Receive: 7146.1 Send: 609.6	729,368.5	0.0
Proactive DLP	7.8	38.7	37.5	29.7	31.0	Receive: 8842.6 Send: 791.8	Receive: 8635.6 Send: 729.1	965,455.7	0.0
Vulnerability	4.4	52.2	53.0	26.0	26.9	Receive: 16,215.5 Send: 1006.8	Receive: 15,785.7 Send: 997.2	1714,449.3	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV) + Deep CDR	15.4	72.4	75	34.3	35.4	Receive: 5436.6 Send: 611.3	Receive: 5589.3 Send: 607.2	495,697.4	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP	16.9	77.2	80.2	36.4	37.9	Receive: 5254 Send: 585.4	Receive: 5041.3 Send: 576.1	450,657.6	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	18.0	72.1	80.4	38.6	38.0	Receive: 4504.3 Send: 574.1	Receive: 4649.5 Send: 571.1	423,010	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

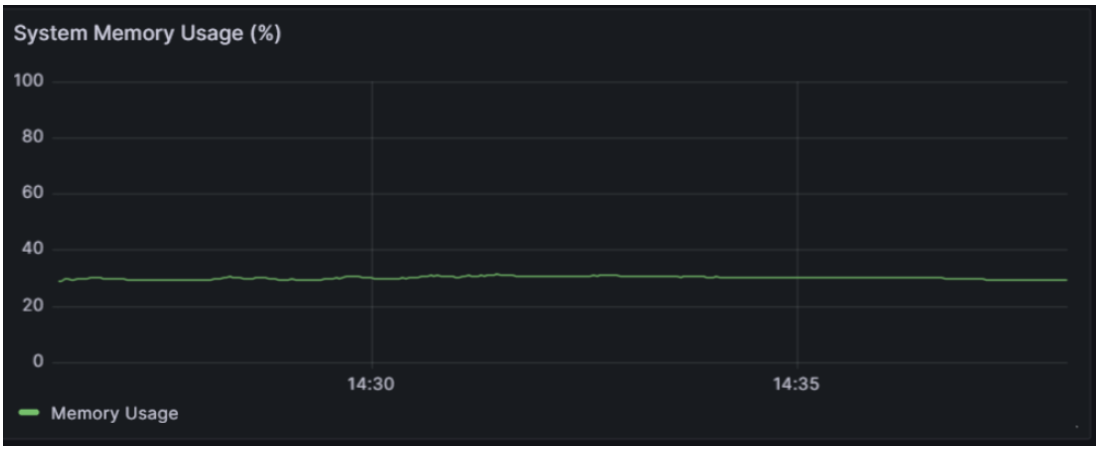
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2

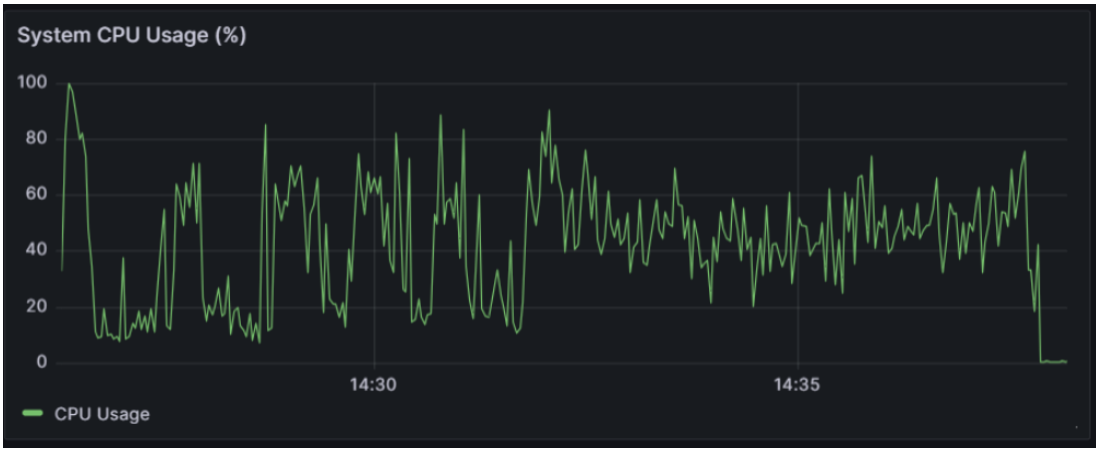


CPU usage

MD Core #1

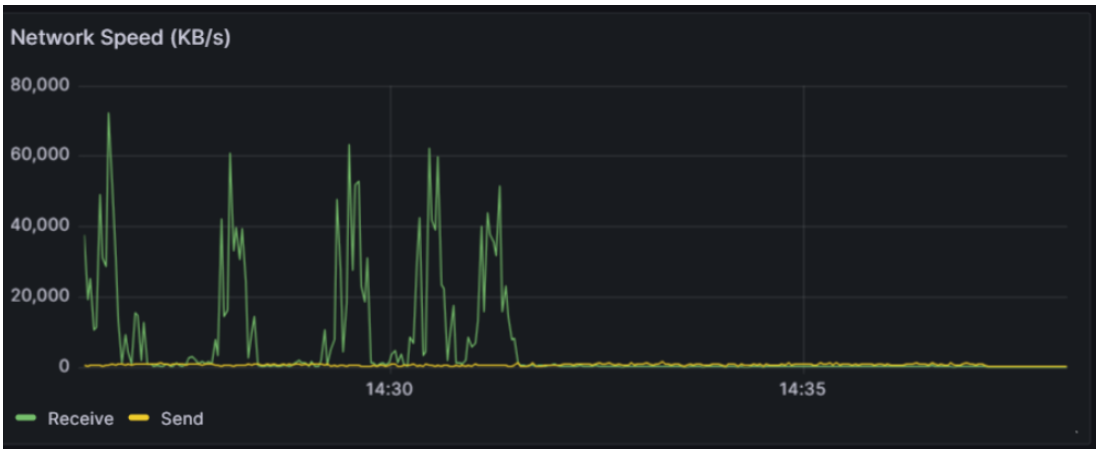


MD Core #2

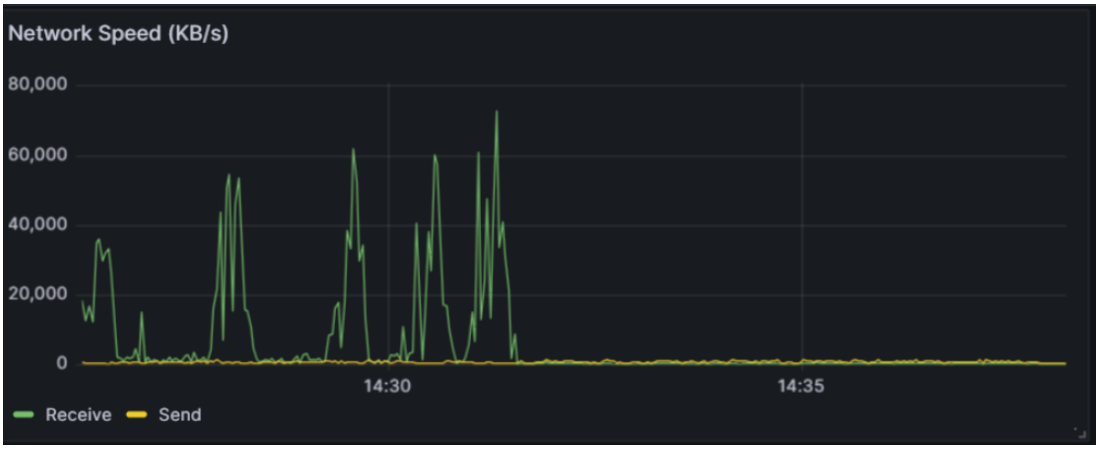


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2

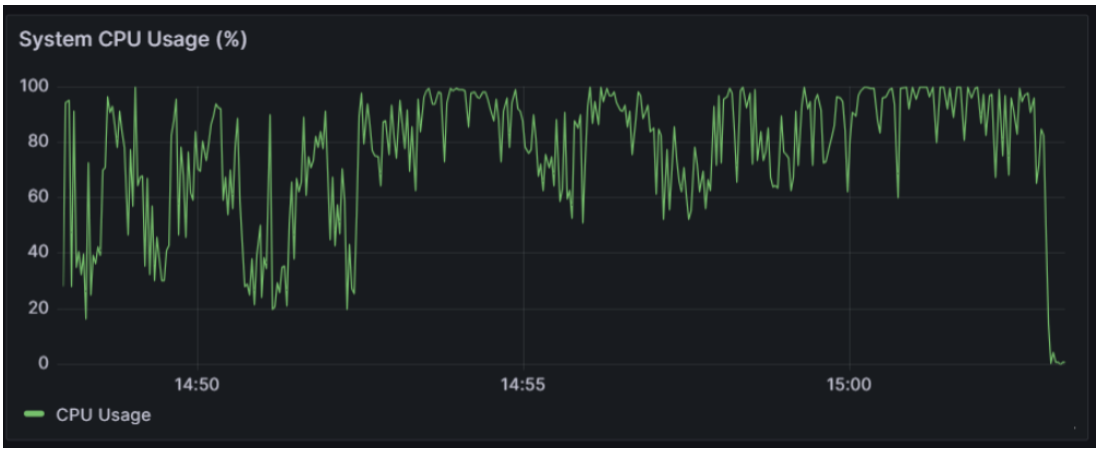


CPU usage

MD Core #1

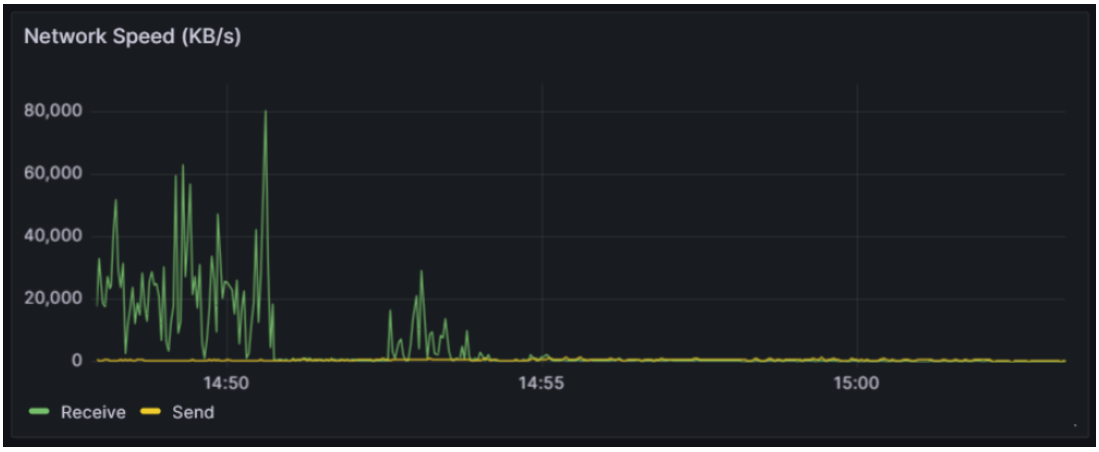


MD Core #2



Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

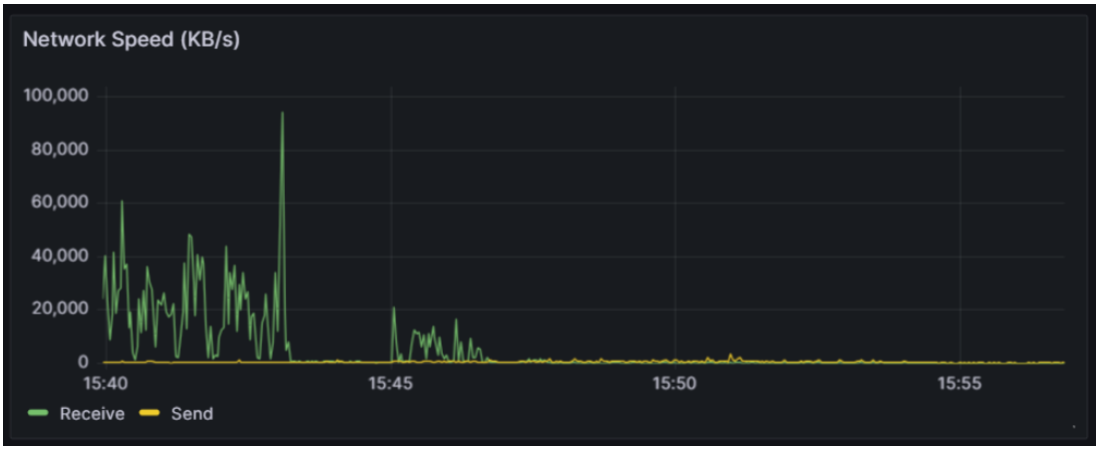


MD Core #2

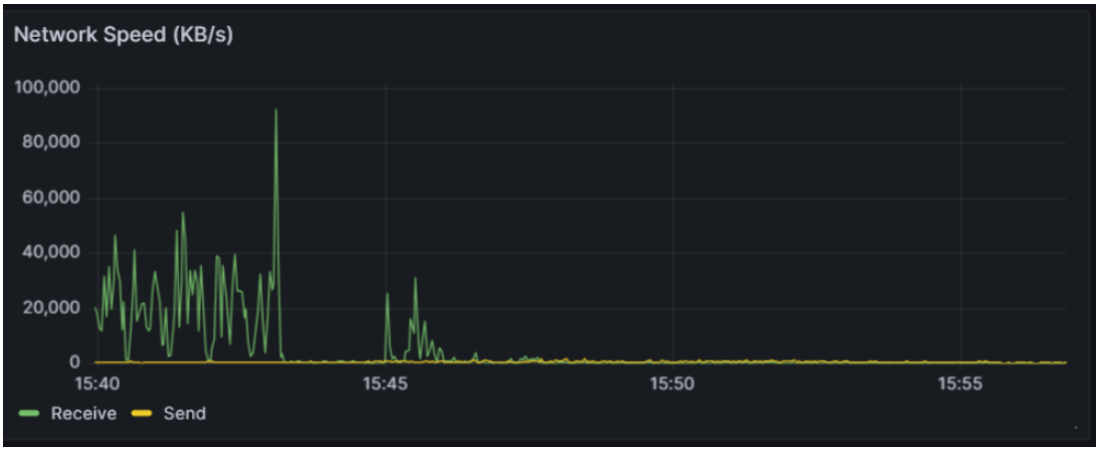


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1

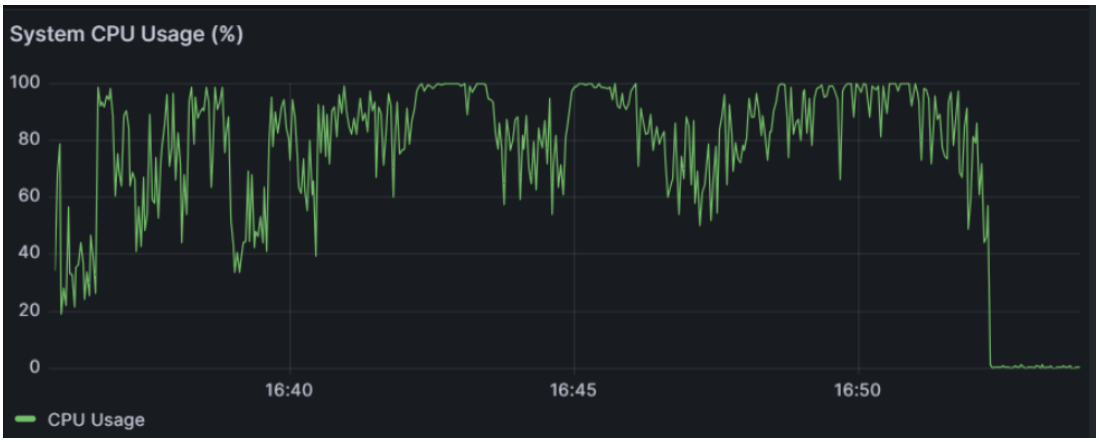


MD Core #2



CPU usage

MD Core #1

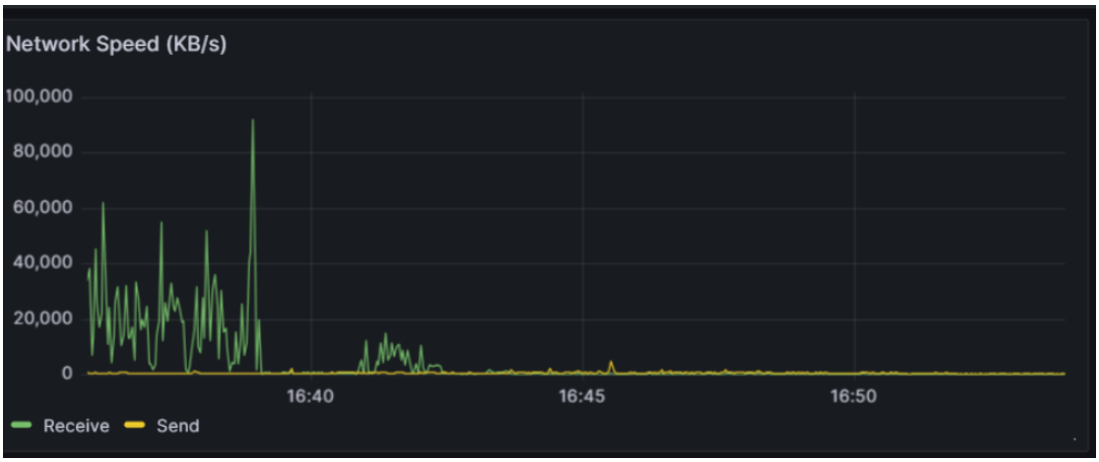


MD Core #2

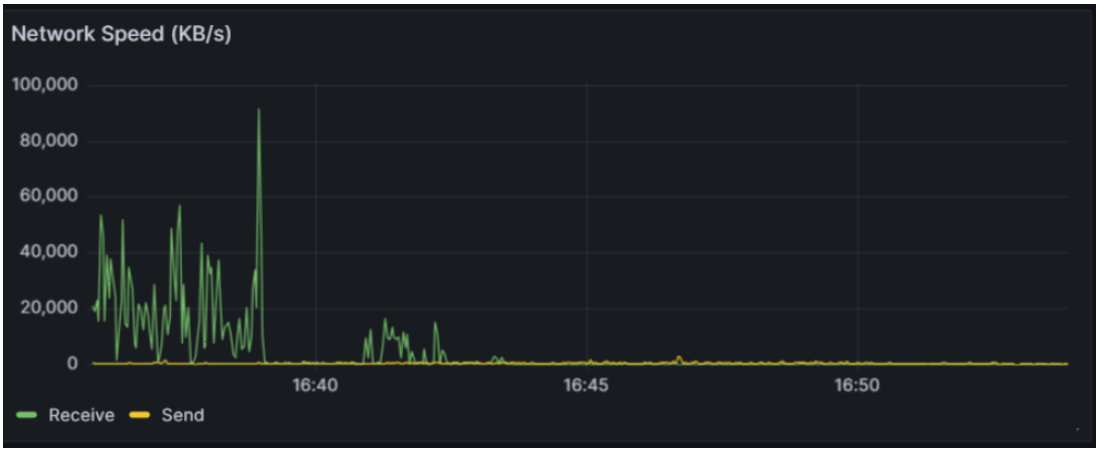


Network usage

MD Core #1



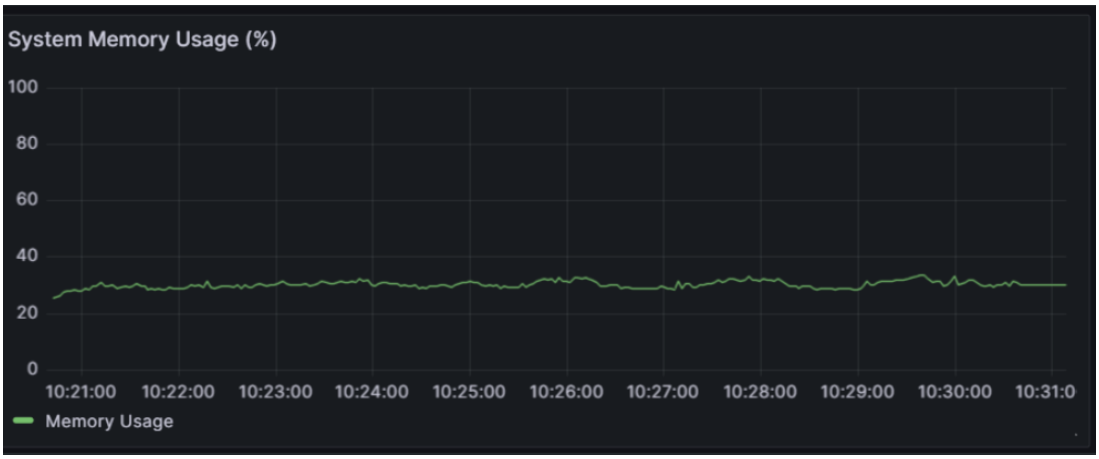
MD Core #2



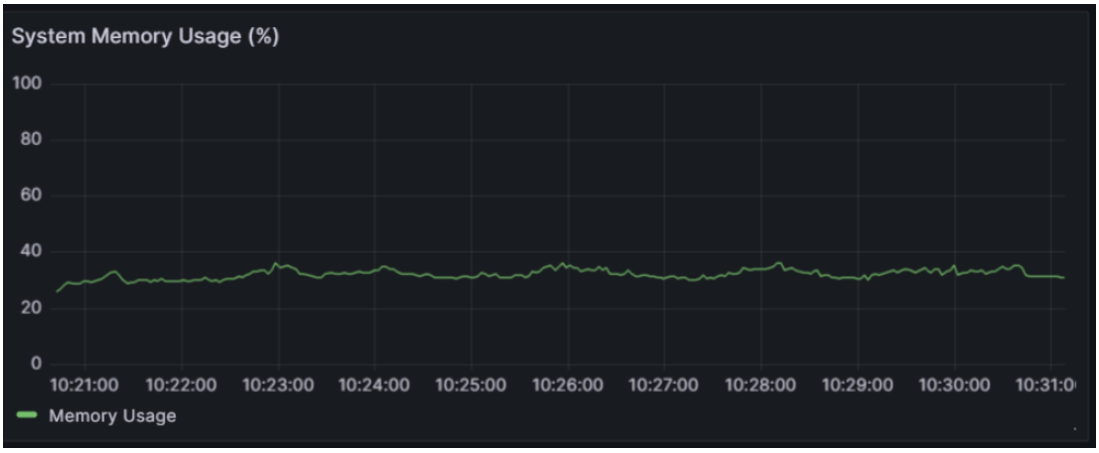
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

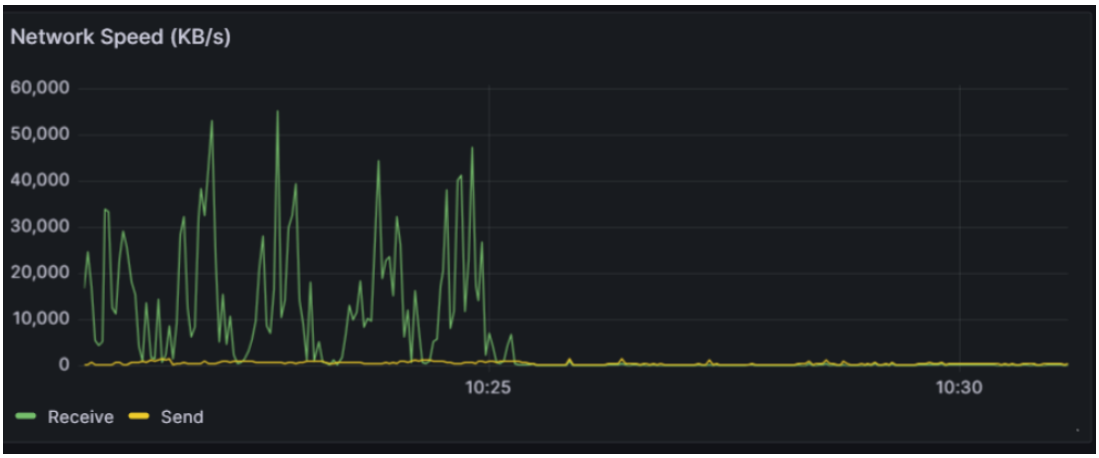


MD Core #2

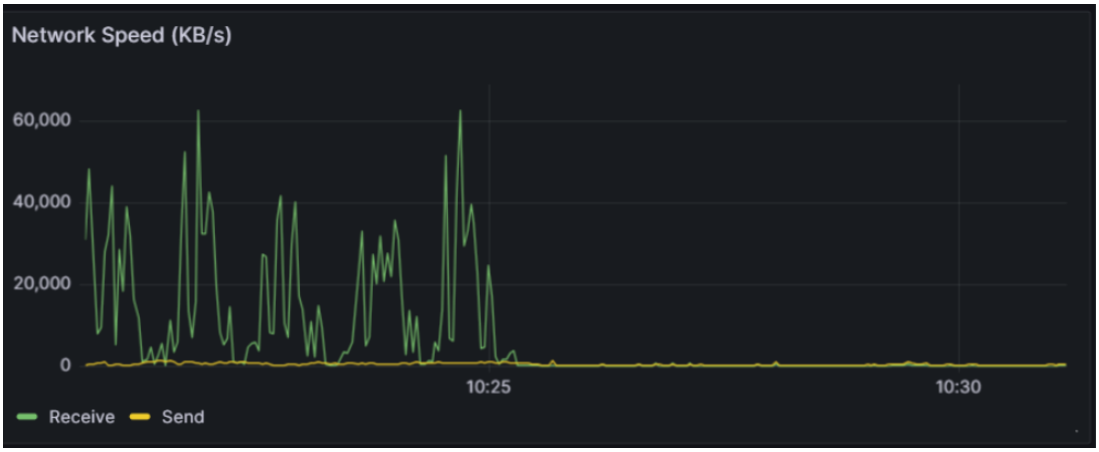


Network usage

MD Core #1



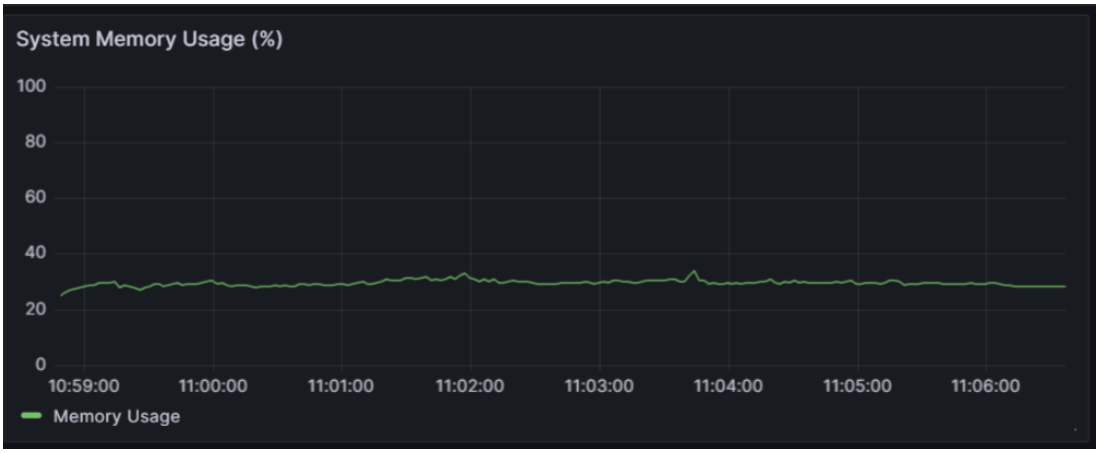
MD Core #2



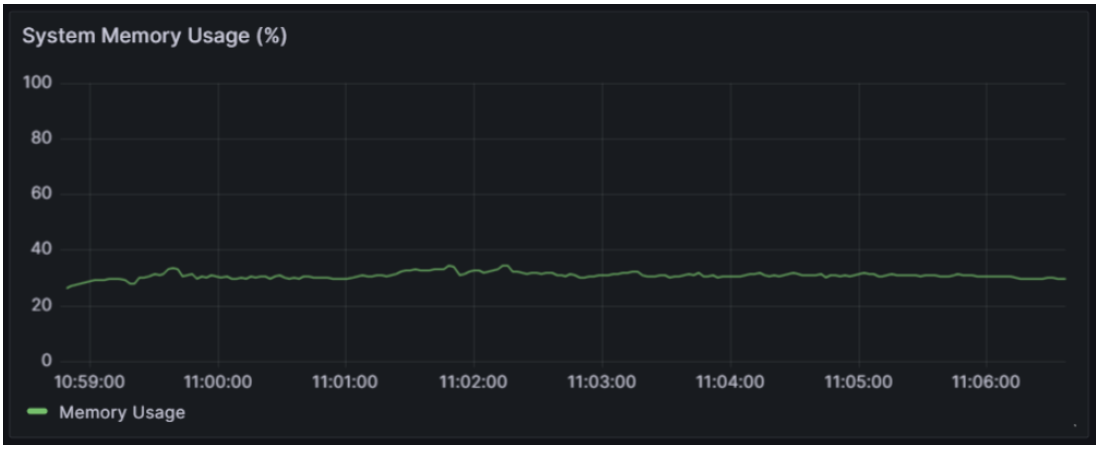
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

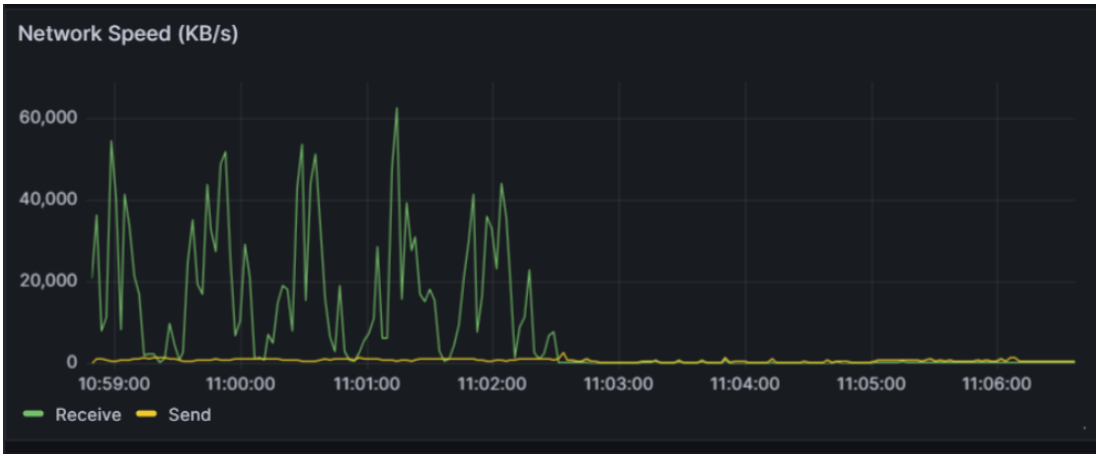


MD Core #2

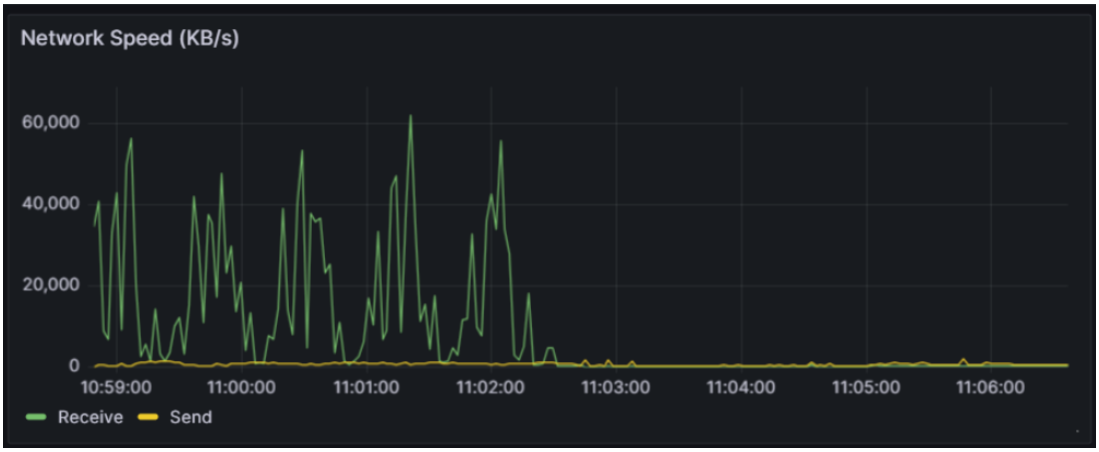


Network usage

MD Core #1



MD Core #2



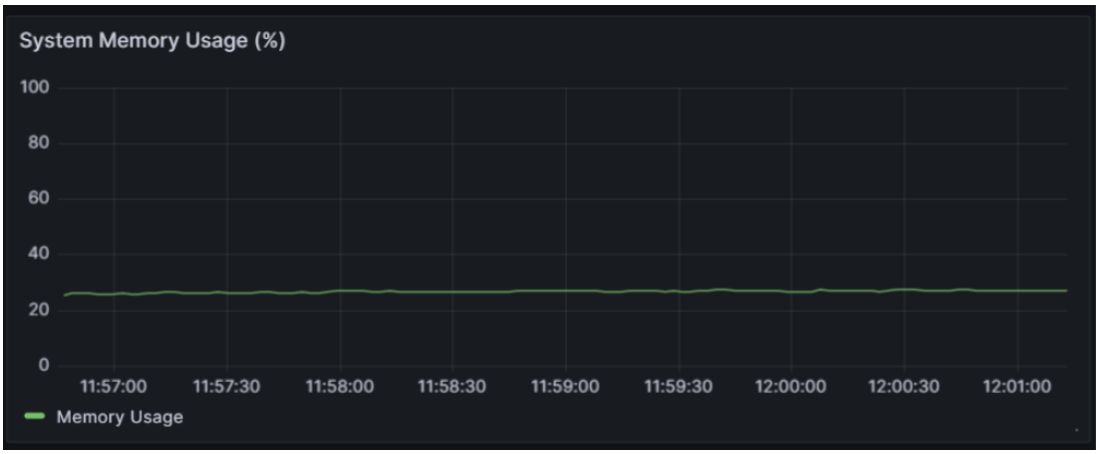
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

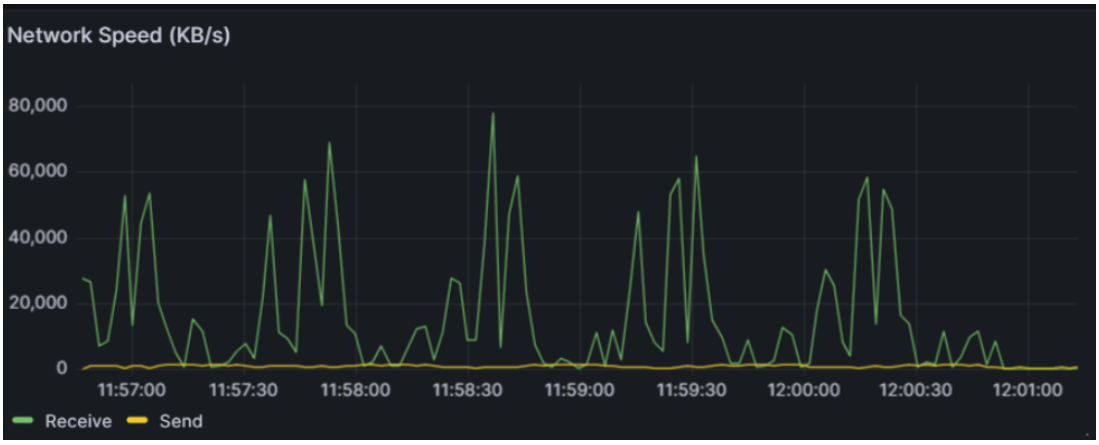


MD Core #2

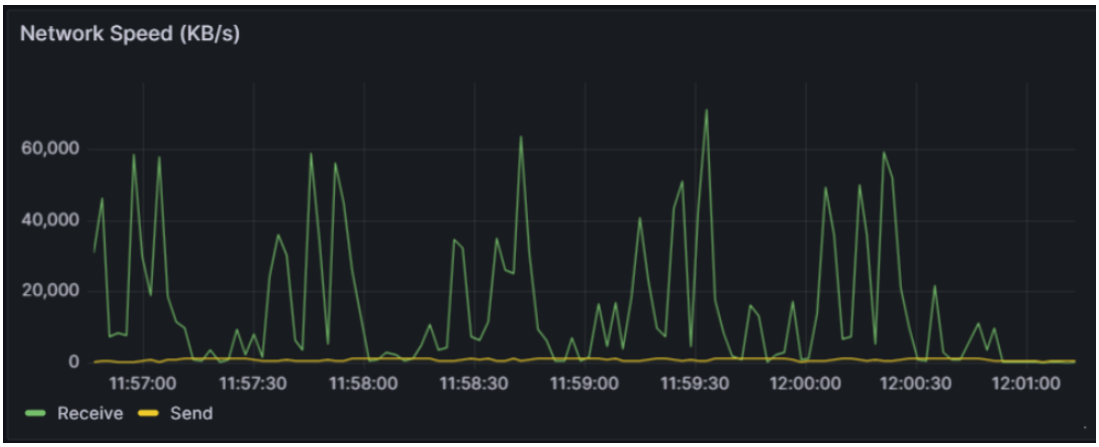


Network usage

MD Core #1



MD Core #2



Shared DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

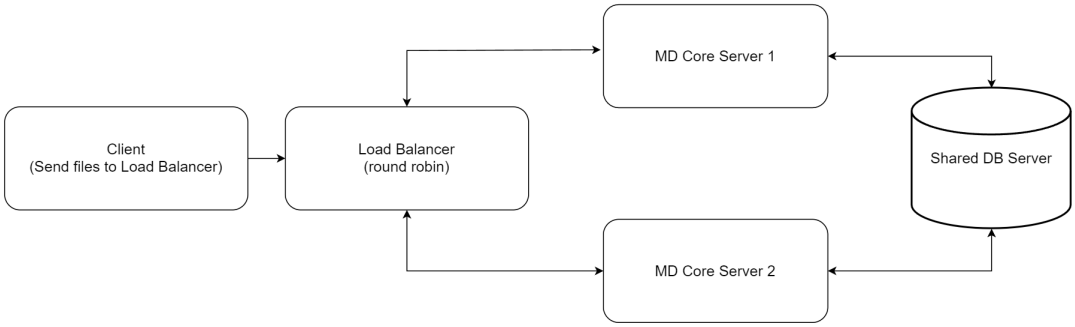
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV)	15.5	63.1	73.3	25.2	25.8	Receive: 4555.9 Send: 694.4	Receive: 4516.2 Send: 692.4	492,620.5	0.0
Deep CDR	9.70	50.7	51.2	31.9	32.5	Receive: 7062.9 Send: 623.2	Receive: 6653 Send: 654.1	786,189.6	0.0
Proactive DLP	7.2	35.2	35.0	27.9	27.5	Receive: 9047.4 Send: 802.4	Receive: 9859.8 Send: 787.5	1,055,110.8	0.0
Vulnerability	5.0	45.7	49.3	24.4	24.5	Receive: 13,585.7 Send: 966.7	Receive: 13,505.8 Send: 987.4	1,519,562.1	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV) + Deep CDR	19.9	79.1	77.6	34.0	34.2	Receive: 3785.9 Send: 573.8	Receive: 3663.1 Send: 577.7	382,716	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP	21.6	80.3	81.5	37.7	44.4	Receive: 3305.6 Send: 543.3	Receive: 3414.3 Send: 556.2	353,694.9	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	22.5	84.2	79.8	37.9	44.7	Receive: 3267.9 Send: 554.9	Receive: 3398.2 Send: 553.8	338,507.6	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

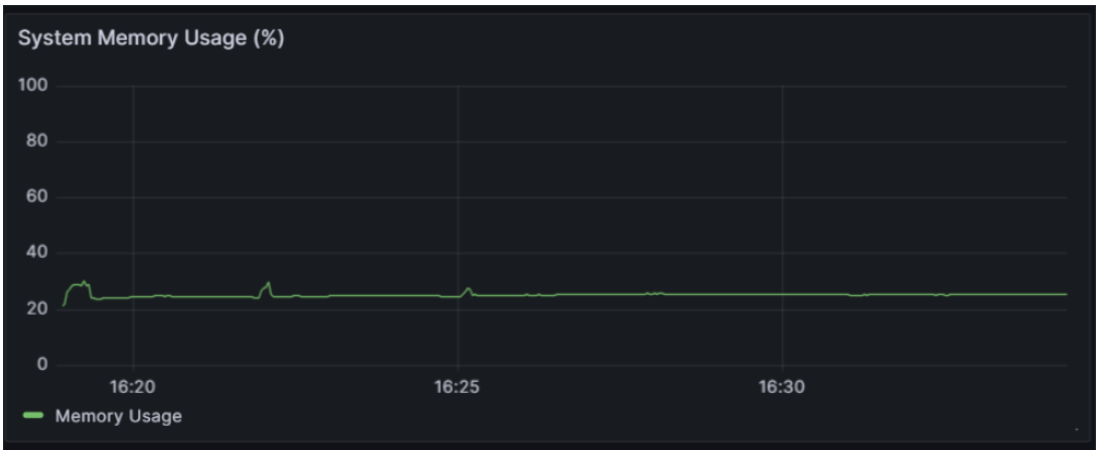
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

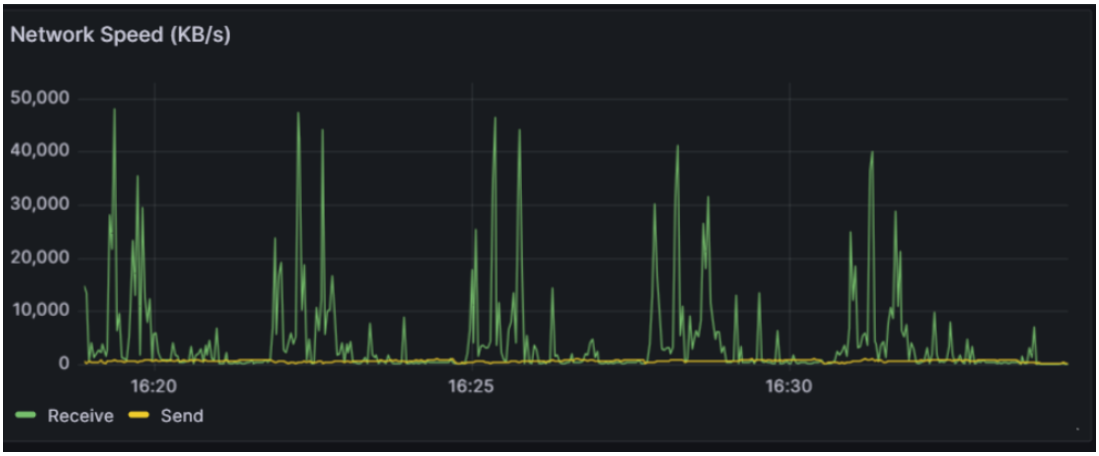


MD Core #2

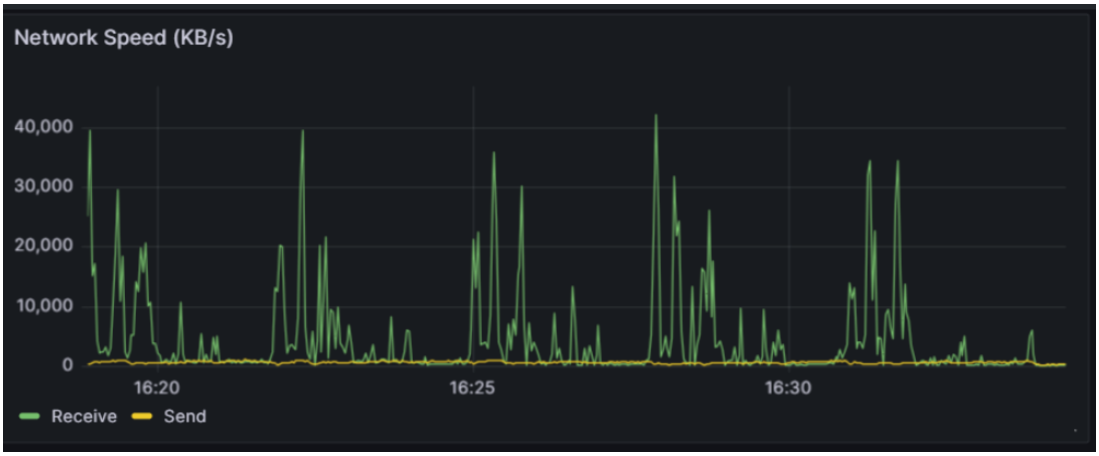


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

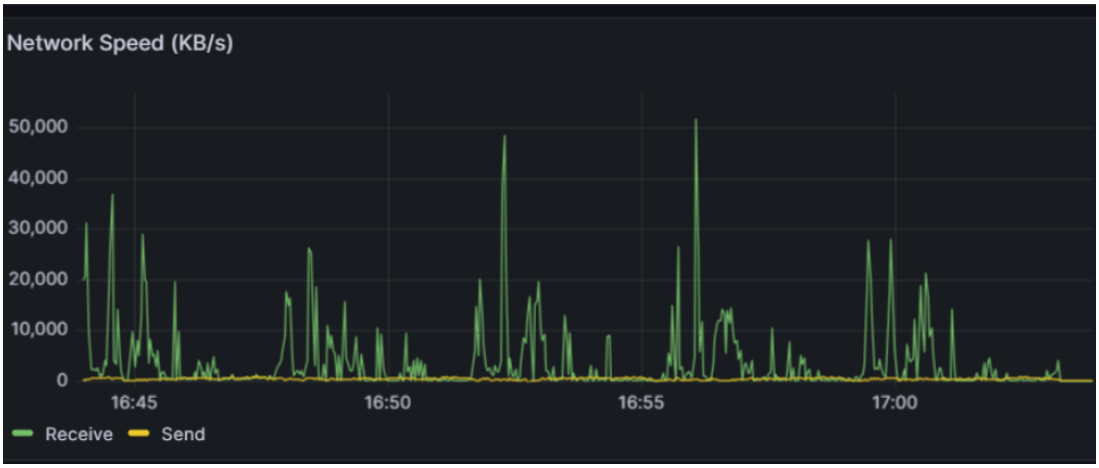


MD Core #2

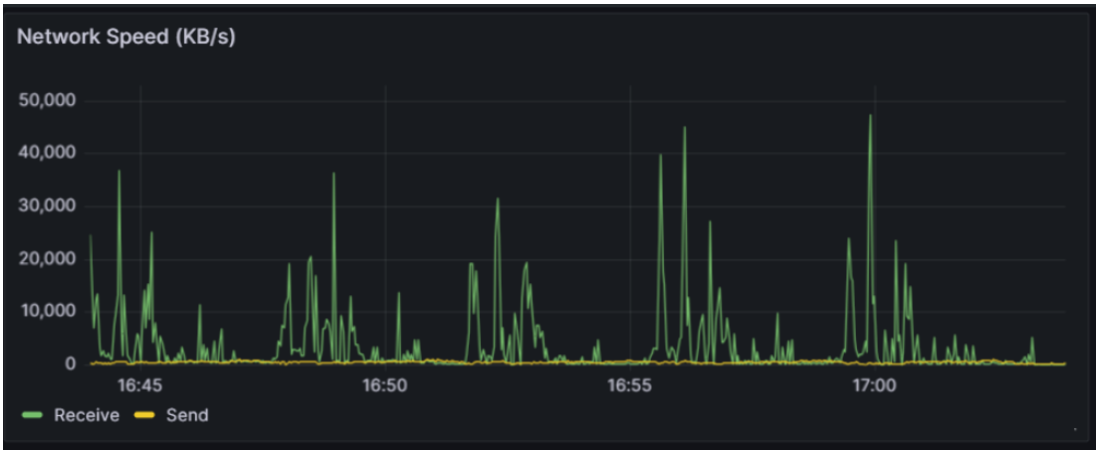


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1

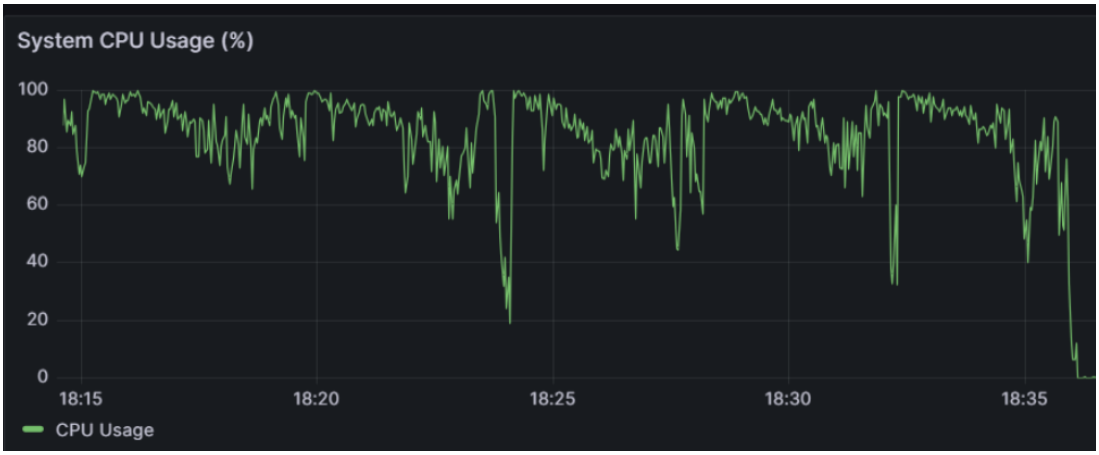


MD Core #2



CPU usage

MD Core #1

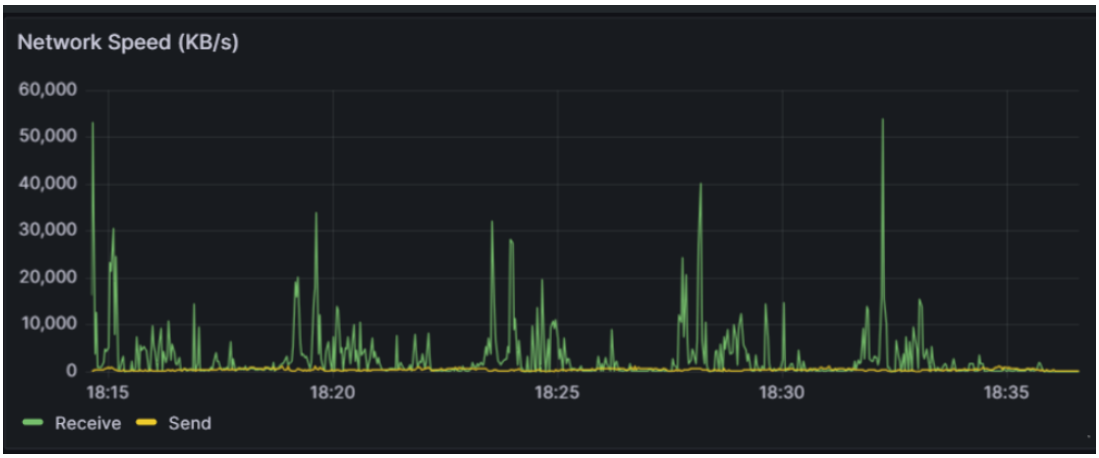


MD Core #2

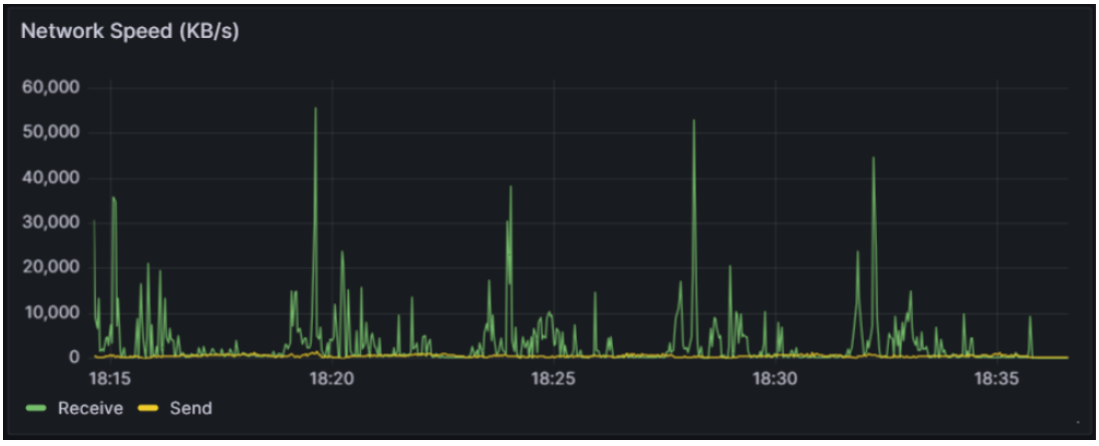


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

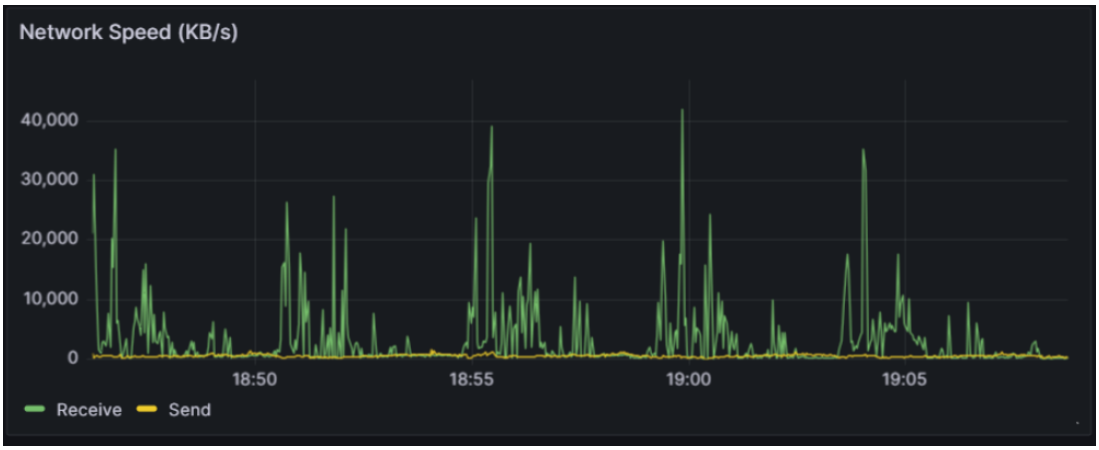


MD Core #2

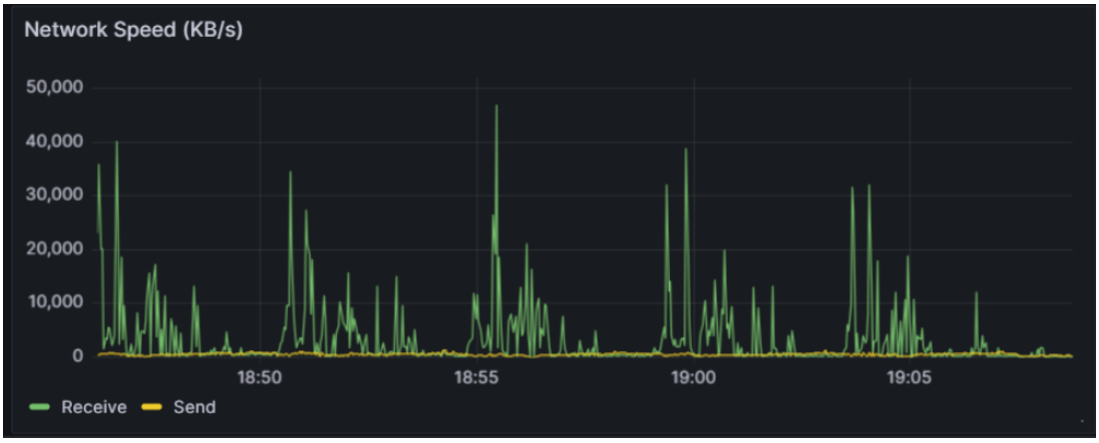


Network usage

MD Core #1



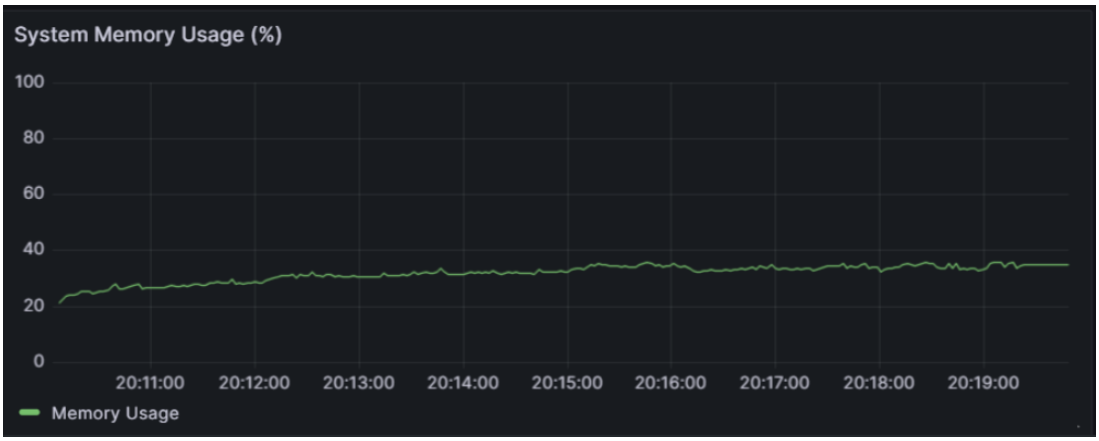
MD Core #2



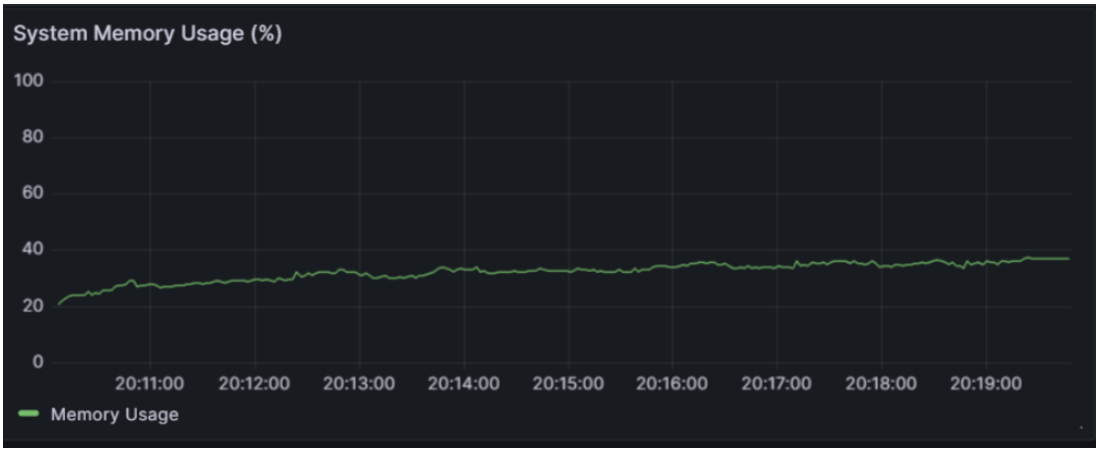
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1

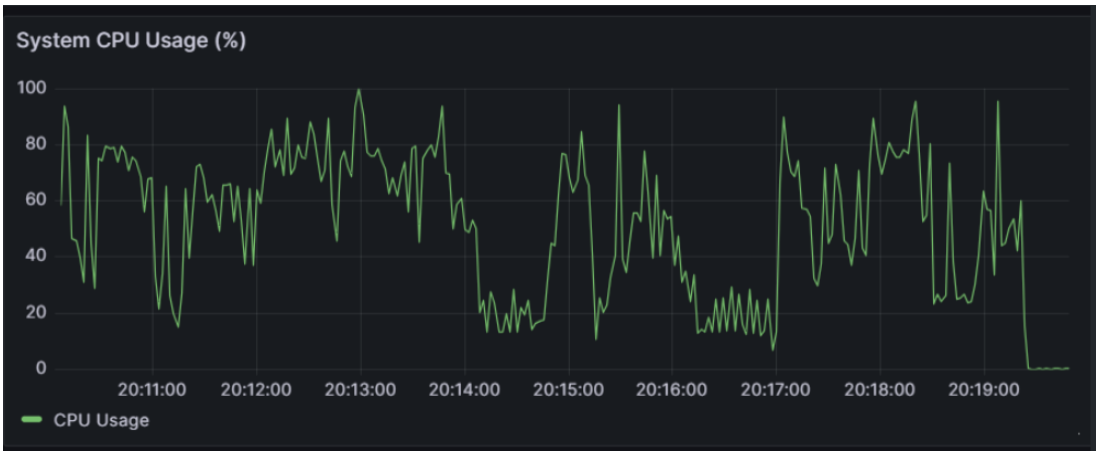


MD Core #2



CPU usage

MD Core #1

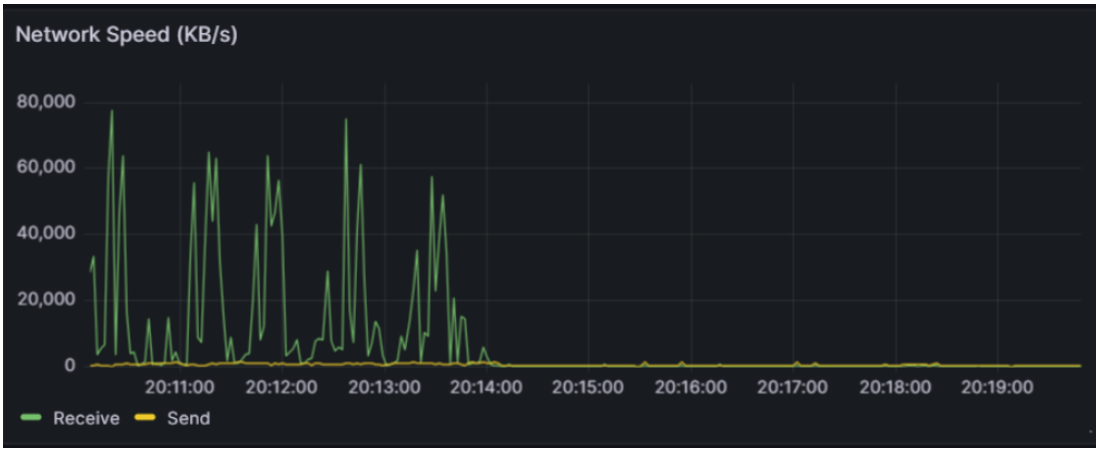


MD Core #2

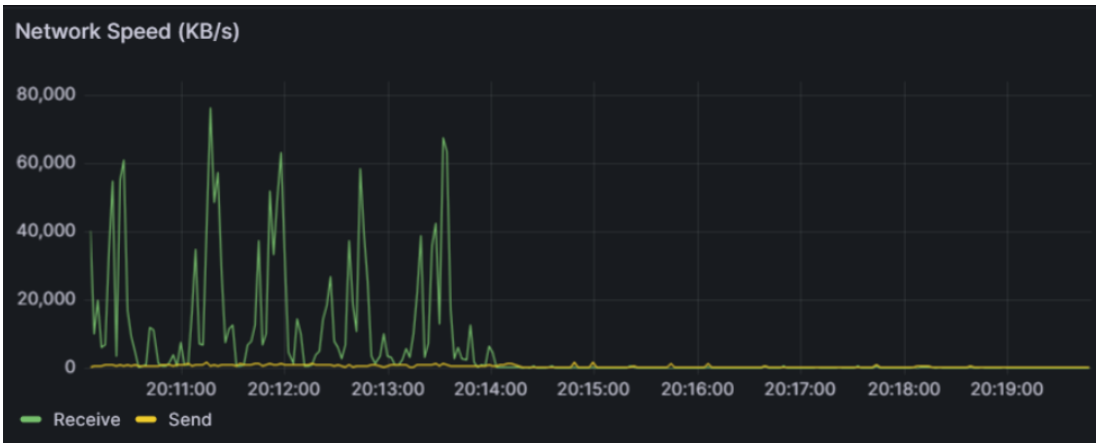


Network usage

MD Core #1



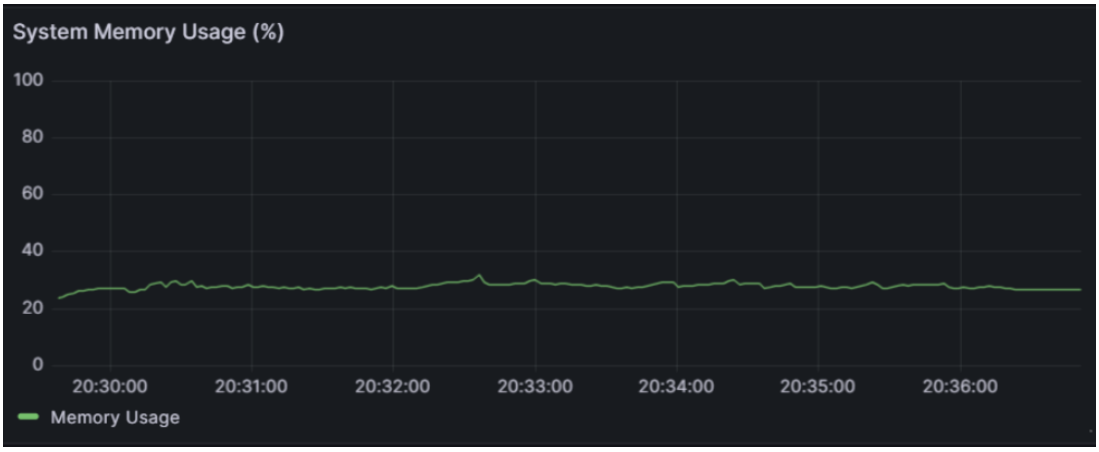
MD Core #2



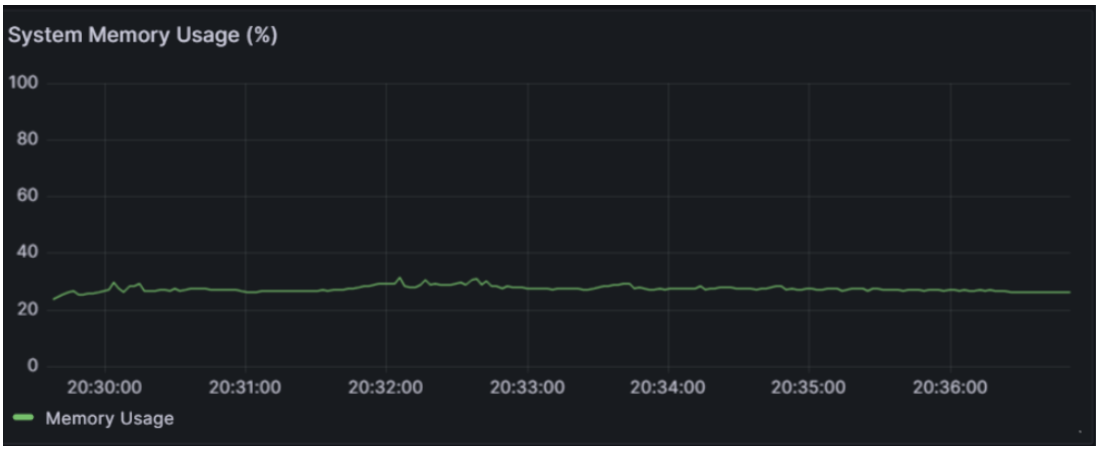
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1

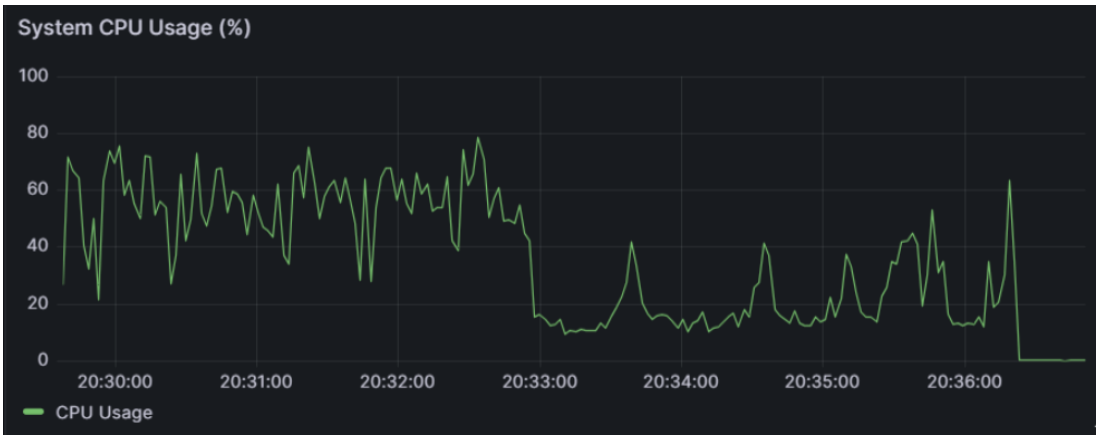


MD Core #2

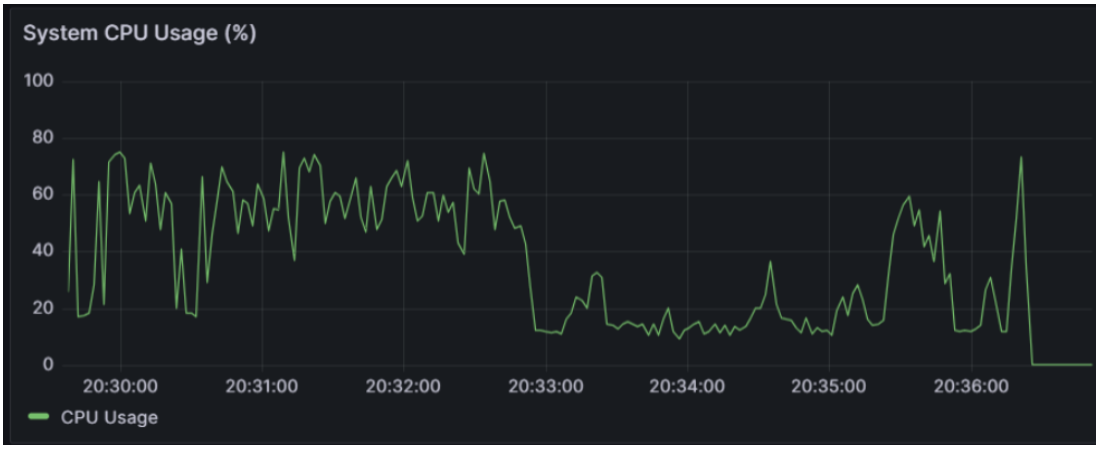


CPU usage

MD Core #1

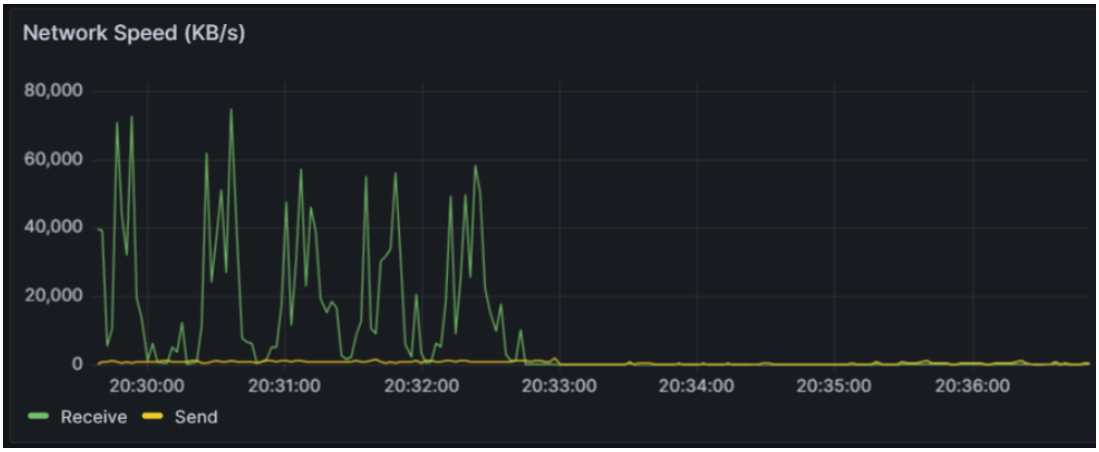


MD Core #2

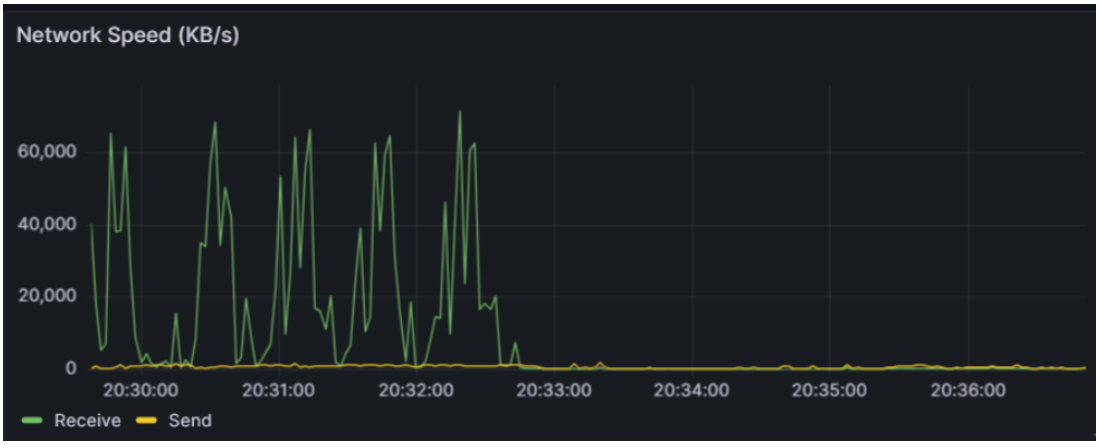


Network usage

MD Core #1



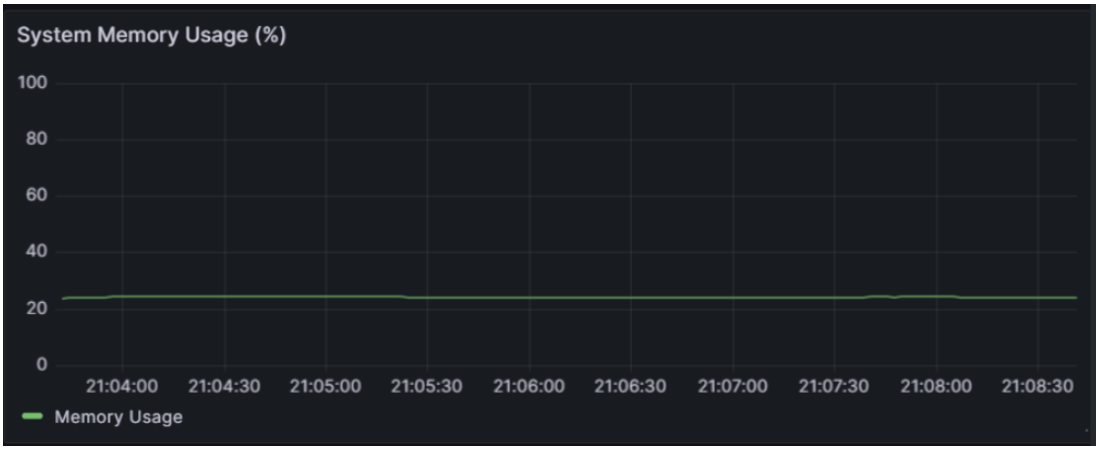
MD Core #2



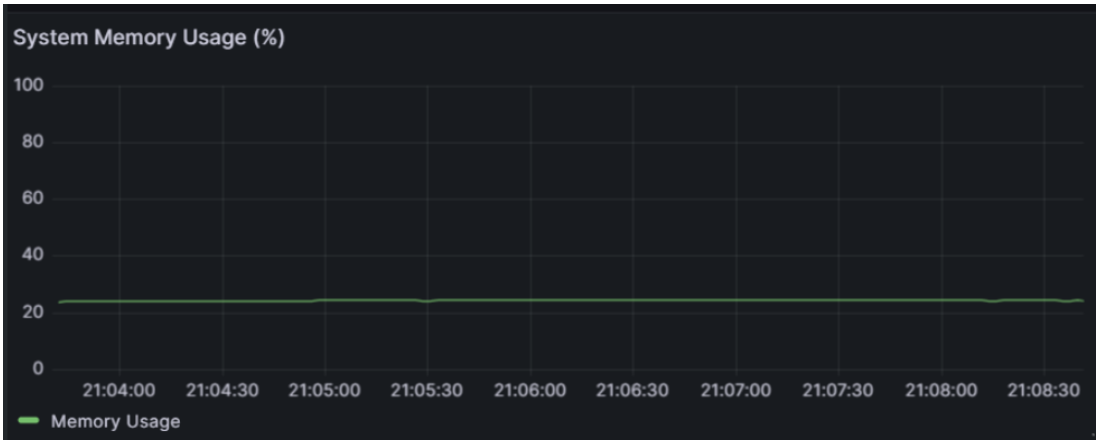
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

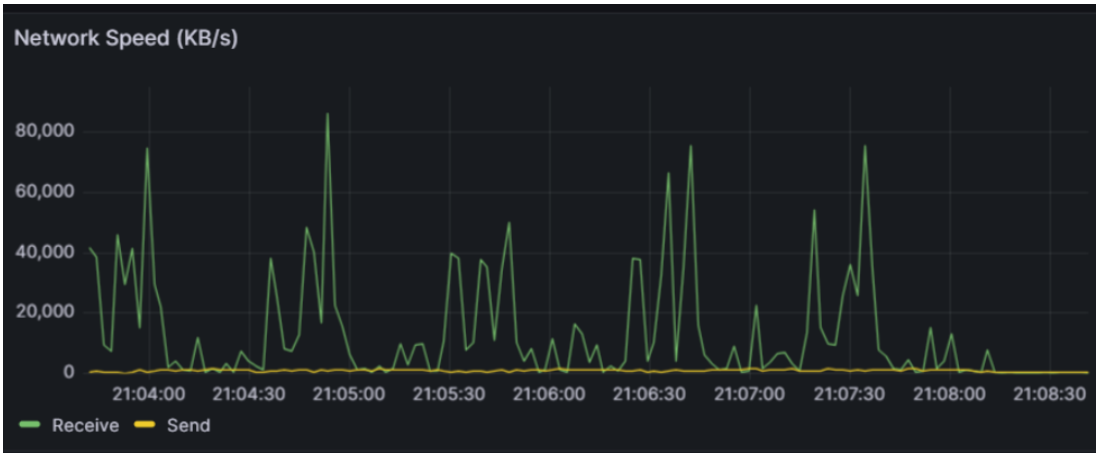


MD Core #2

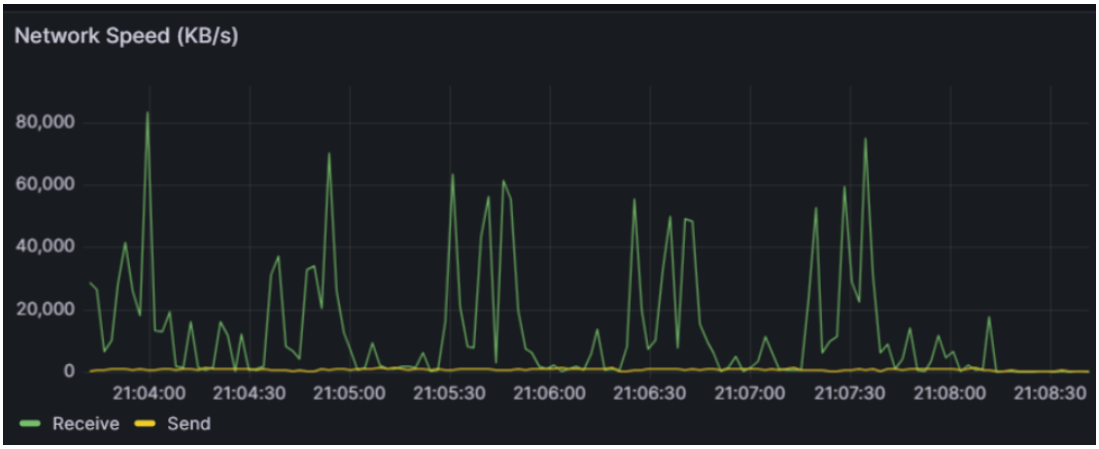


Network usage

MD Core #1



MD Core #2

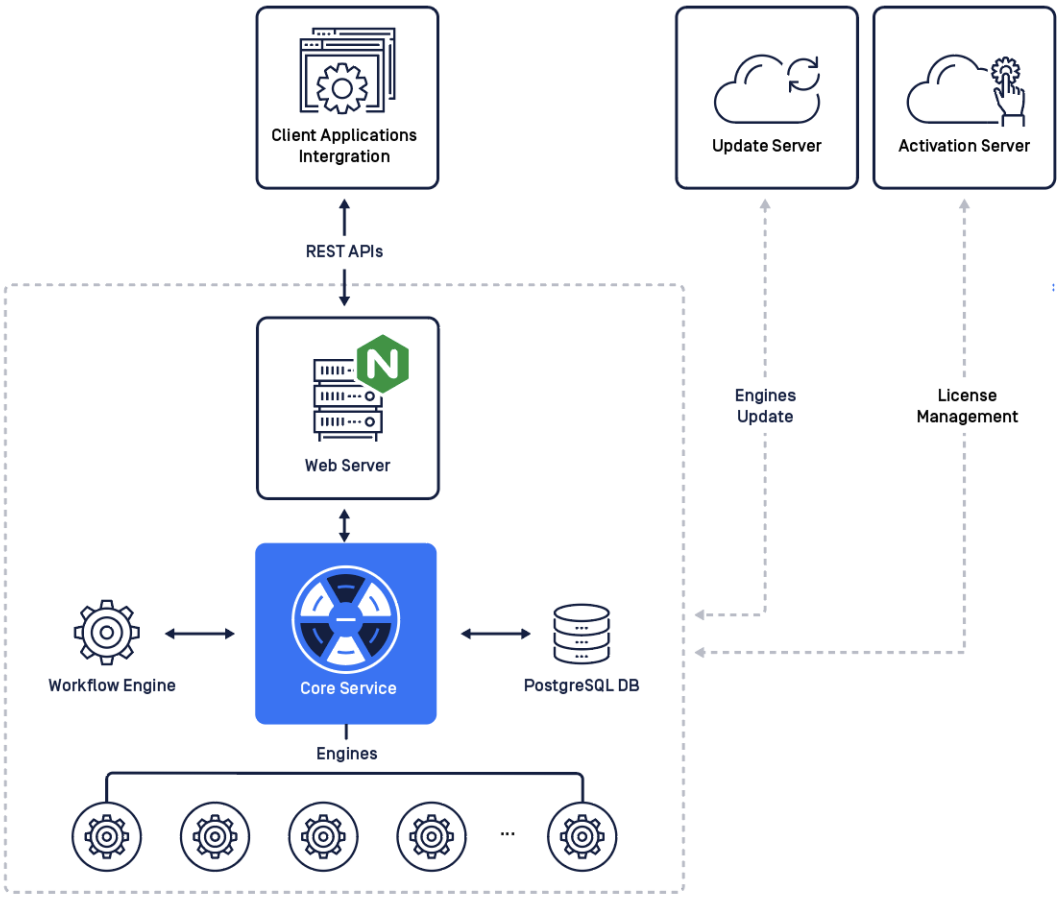

Network Speed (KB/s)
80,000
60,000
40,000
20,000
0
21:04:00
21:04:30
21:05:00
21:05:30
21:06:00
21:06:30
21:07:00
21:07:30
21:08:00
21:08:30
Receive
Send

MetaDefender Core Documentation

The users can consult this web page or, alternatively, they can download the manual in pdf format from the link below:

MetaDefender Core manual (SHA256:).

Recommended System Configuration



MetaDefender Core supports a wide range of deployment options, including air-gapped environments, providing the flexibility to choose the environment that best suits customers' requirements and infrastructure.

- Physical server.
- Virtualization platforms like VMware, Hyper-V and XenServer.
- Cloud Deployment.
- Container Deployment.

> **Note**
> Before installing MetaDefender Core, please refer to the **recommended minimum system configuration** listed below. The requirements are based on a standardized customer deployment and may require more or less resources to achieve the level of performance desired in your environment. Please note that the server specifications are built to allow a high volume daily processing. Please refer to Performance and Load Estimation for more details.

The recommendations below are for MetaDefender Core, API usage only. For any other use cases, please consult the user guide of the licensed products for accurate recommendations.

For certain use cases these might be adjusted and customized on specific needs and SLAs. We highly recommend to engage our ProServ team to assist in fine tuning MetaDefender and get the maximum performance out of your systems.

Microsoft Windows Deployments

[Windows] Supported Operating Systems

Only support 64-bit platforms.

- Windows 10, Windows 11 23H2, 11 IoT (22H2) only for End-User systems (e.g. Kiosk deployments)
  - Based on Microsoft License Terms, you're not allowed to use Windows Desktop as a server, therefore, MetaDefender Core can't be used on these systems for server use cases.
- Windows Server 2016, 2019, 2022

> **Disclaimer**
> End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

> **Plan for legacy OS support**
> OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:
>
> - Ubuntu 20.04 after **April 2025**.
> - Amazon Linux 2 after **June 2025**.
> - Windows 10 after **October 2025**.
>
> If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

[Windows] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 8	8	8 GB	16 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 12	16	16 GB	24 GB + 50 GB + 100 GB
MetaDefender Core 16	16	16 GB	32 GB + 50 GB + 100 GB
MetaDefender Core 20	32	16 GB	40 GB + 50 GB + 100 GB
MetaDefender Core MAX	32	32 GB	120 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core is installed.
- For a better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.
- SBOM processing requires more disk space, please allow at least 15GB free disk for the engine to work properly.

[Windows] Third Party Dependencies

- .NET framework 4.5 or above
- Microsoft Visual C++ Redistributable for Visual Studio 2015-2022 (version 14.38 or higher); x64 version.
- PostgreSQL: Only applicable when using a pre-installed PostgreSQL server running remotely.
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Some engines also have dependencies as described below:

Engine Name	Dependency
Deep CDR	Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 5.8.0 or above)
ESET	MetaDefender Core v4 temporary directory should have more than 200MB free disk space
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
Proactive DLP	• Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 2.0 or above) • .NET 6 dependencies • CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
Scrutiny	Microsoft Visual C++ 2015 - 2019 Redistributable x86 and x64
Systweak	.NET runtime 6.0, 7.0 or 8.0
Bkav Pro	Microsoft Visual C++ 2015 - 2022 Redistributable x64

[Windows] Installation Details

MetaDefender Core on Windows uses "C:\Program Files\OPSWAT" folder for storing resources or the installation directory.

MetaDefender will use its resources folder to store temp files as part of the analysis. **It's recommended to exclude this folder from real-time protection monitoring**. Knowledge base:

- How to set up exclusions on anti-malware software to prevent disruption
- Mitigating scan failure triggered by real-time protection service
- Can local AVs interrupt ongoing scans?

Linux Deployments

[Linux] Supported Operating Systems

Only support 64-bit platforms.

- Red Hat Enterprise Linux 8, 9
- Rocky Linux 9
- Debian 11, 12
- Ubuntu 20.04, 22.04
- Amazon Linux 2 is supported via Amazon Machine Image (AMI)

Plan for legacy OS support

OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:

- Ubuntu 20.04 after **April 2025**.
- Amazon Linux 2 after **June 2025**.
- Windows 10 after **October 2025**.

If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

For Hardended Linux OS

Some AV engines could be failed to initialize and run on MetaDefender Core (e.g. ESET, Kaspersky) due to execution permission required by them on `/var/tmp/ometascan` folder.

Please follow steps at Why have the ESET and Kaspersky scan engines failed to initialize on the hardened Linux OS? for remediation details.

Disclaimer

End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

[Linux] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 5	4	4 GB	10 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 10	8	8 GB	20 GB + 50 GB + 100 GB
MetaDefender Core MAX	16	16 GB	40 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core resides on.
- For better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.

[Linux] Third Party Dependencies

Dependencies list:

- openssl
- grep
- lib32stdc++6 (>= 4.5)
- libc6-i386 (>= 2.10)
- procps
- zlib1g
- libcurl3 (>= 7.19.7)
- libcurl4
- postgresql:
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Not all the above dependencies will need to be installed, it is dependent on different Unix distro and version

Some engines also have dependencies as described below:

Engine name	Dependency
Archive engine	• To extract password protected MS Office files, you need to install .NET 6 dependencies, libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • Rocky Linux 9.x/Red Hat 9.x: libnsl
Deep CDR	• Rocky Linux 9.x: libgomp, ncurses-compat-libs, libnsl • CentOS: libgomp, ncurses-compat-libs • Red Hat 8.x: libgomp, ncurses-compat-libs, libnsl • Red Hat 9.x: libgomp, ncurses-compat-libs, libnsl, compat-openssl11 • Ubuntu/Debian: libgomp1, libncurses5 • libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • .NET 6 dependencies
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
FileType engine	Rocky Linux 9.x/Red Hat 9.x: libnsl
Proactive DLP	libgdiplus, mscorefonts .NET 6 dependencies CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
RocketCyber	libgomp

Note: Some dependencies may need to be installed from external repositories, not from the OS default repositories.

[Linux] Installation Details

MetaDefender Core default installation path is using /var folder for storing resources:

- `/var/lib/ometascan` : installation folder with all its resources (database, DLP processed / CDR sanitized / quarantined file storage, engine package and definition updates).
- `/var/log/ometascan` : application logs.
- `/var/run/ometascan` : runtime folder for Core related processes.
- `/var/lib/ometascan` : all successful deployed engines data files.
- `/usr/lib/ometascan` : all shared libraries required by MetaDefender Core, bundled PostgreSQL server related binaries and libraries.
- `/var/tmp/ometascan/resources` : all temporary processing files for MetaDefender Core to process.

Storage Usage Information

Based on the configuration, MetaDefender Core will need additional disk space to store analysis/result data, quarantined files and temporary files:

	Data Storage
Analysis Reports (Processing History)	Approximately 4.5GB is required for every 1M analysis reports stored. • 100 GB is recommended as a baseline. Database storage requirements are affected by Data Retention settings, engine settings and scanning load.
Quarantined Files	Depends on the customers' dataset and configuration
Sanitized Files	Depends on the customers' dataset and configuration
Temporary processing files	Depends on the customers' dataset and configuration • A 1GB non-archive file requires 1GB free disk space at least. • A 1GB archive file requires (1 * 2)GB free disk space at least. • 50GB is a baseline and should be adjusted based on observation of the solution within the environment, with consideration to peak times and overall file makeup (e.g. complexity, size, archives).
Note	Options are available to control how large and/or complex files are handled to limit the amount of system resources required. See Workflow Configuration Template for more information.

Custom Engines

The recommendations above are specific for MetaDefender pre-packaged bundles.

However for additional Custom Engines, please review the Knowledge base to review additional requirements (if any) for the selected engine.

Browser Requirements for the MetaDefender Core Management Console

One of the following browsers is suggested to view the MetaDefender Core Management Console:

- Microsoft Edge
- Chrome
- Firefox
- Safari

Chrome, Firefox, Safari and Microsoft Edge browsers are tested with the latest available version at the time of release.

Network Bandwidth

In online environment where MetaDefender Core is configured to download engine update packages directly from OPSWAT update server source, then using gigabit network speed (125 MB/sec) is highly recommended to avoid potential network related issue while downloading files, or at least 100 Mbps (12.5 MB/sec) network speed.

Allowlisting URLs

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts':

- https://activation.dl.opswat.com - for license online activation
- https://update.dl.opswat.com - for engines and databases download

Note: *IP address-based allowlisting on your firewall might fail after some time since OPSWAT uses CDN (AWS Cloudfront) to faster delivery updates over the world, and IP address of edge servers might change over time.*

Depending on your location, connections with OPSWAT cloud-based servers be forwarded to specific AWS Cloudfront's edge locations, each is assigned with its own unique IP address ranges. You might also need to allow correct AWS Cloudfront's corresponding IP address ranges by referring to https://ip-ranges.amazonaws.com/ip-ranges.json

Warning

Even the OPSWAT update servers host updates for all of the available engines we support, sometimes custom engines might try to connect to their own cloud for updates, but this can be disabled in firewall and they will be updated just from OPSWAT cloud servers instead.

Differences Between MetaDefender Core v4 and v5

We are very excited to inform you about the release of version 5.0.0 of our MetaDefender Core!

The new version is designed to enhance the product performance, user experience, as well as to provide more flexibility, security, and scalability for our customers.

At the same time, we are also committed to keeping backward compatibility for whom running MetaDefender Core v4, and avoid the need for updating your client integration to move forward with MetaDefender Core v5.

v4 REST API Integration Supported
All public REST API endpoints available on MetaDefender Core v4 will be continously supported on v5.

v4 Product Major Features Supported
All product major features available on MetaDefender Core 4.21.2 (the latest one on v4) will be continously supported on v5.

Seamless Product Upgrade
Upgrading from MetaDefender Core v4 to v5 will be supported and covered with data migration without additional manual steps from users.

Licensing information will be also migrated.

Below are some highlights of the first v5 release, MetaDefender Core 5.0.0:

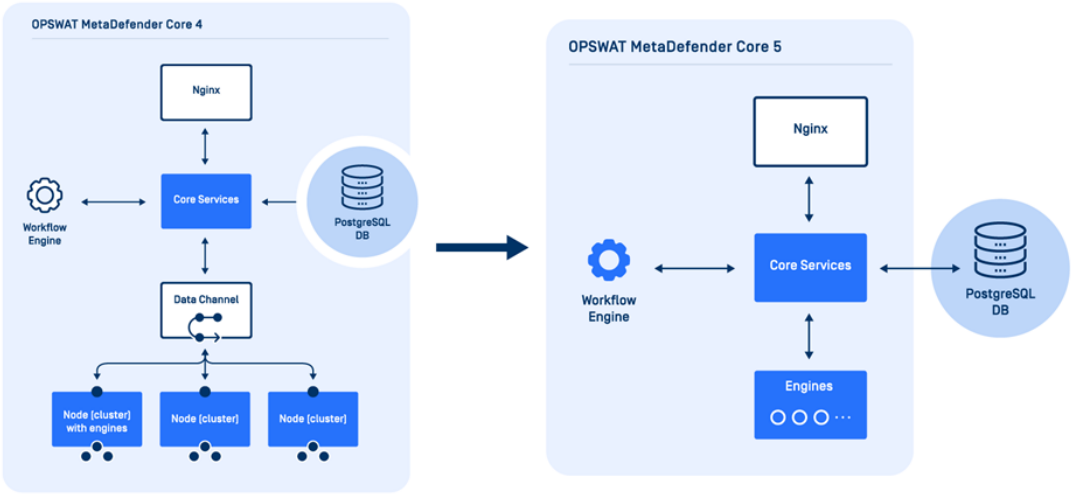
New product architecture

This version introduces a simpler product architecture that speeds up integration and automation, file processing flow, and immensely enhance the product performance and efficiency.

Earlier versions utilized MetaDefender Node concept, designed to support scaling of the scanning infrastructure by distributing scan requests among several different processing Nodes. Product licensing was based on number of Node instances. The benefit of having such a distributed infrastructure is that based on Node loads, Metadefender Core server can always choose the most appropriate Node to assign a new scan task to. However, having nodes in the v4 architecture caused more complexity and possible latency in the processing flow and entailed maintenance cost.

In this release, we retired multi-Node deployment and restructured MetaDefender Core's backend. This alteration significantly improves file processing speed and product efficiency, as well as eliminates extra cost for maintaining node services. Users now can simply install additional MetaDefender Core instances for scalability requirement,

as well as configure their own load balancer to coordinate tasks among multiple MetaDefender Core instances. Accordingly, product licensing is now based on number of the product instances.



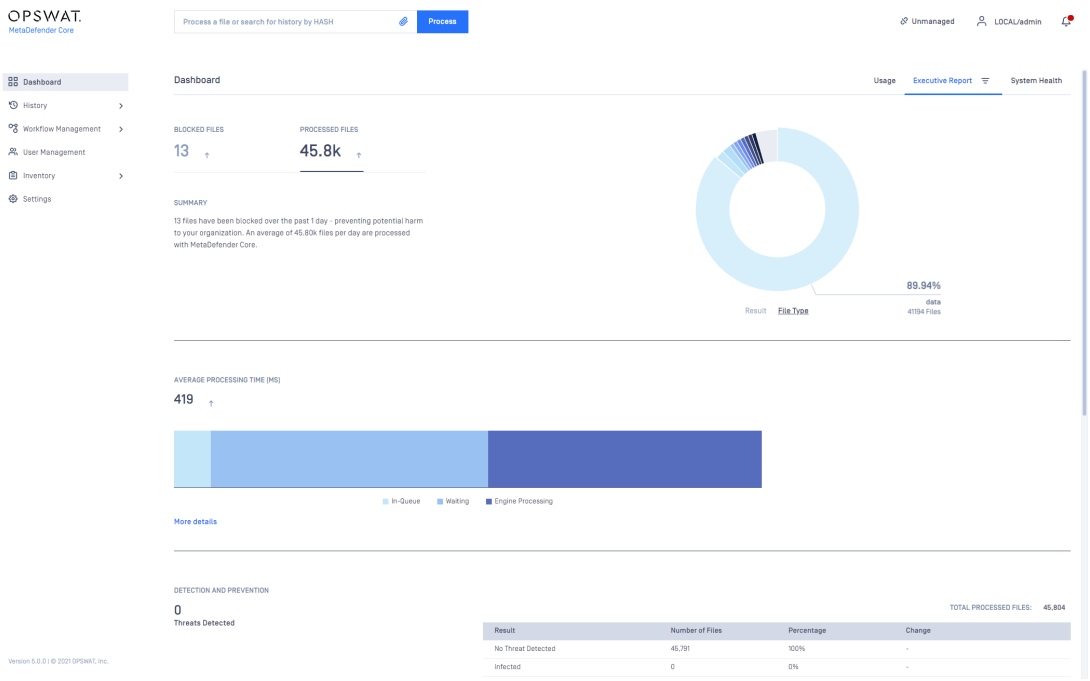
Please be informed that

- All existing administrative APIs related to "Node" on v4 will be kept as it is on v5 to ensure backward compatibility with existing client integrations.
- Applicable settings defined in Windows registry (Windows) or ometascan-node.conf config file (Linux) for Node will be migrated to Core. Details: MetaDefender Configuration

Brand-new product user interface

The second notable update in this version is a brand-new UI for the management console, which was designed with the focus on the quality and thoughtfulness of the user experience. The new administrative console UI provides an advanced set of Core features and observable processing status.

Comprehensive executive dashboard reporting offers dynamic, real-time and actionable threat insights. It also provides visibility of your security posture allowing you to paint a picture of security that aligns with the overall business. The report includes easy-to-read charts, graphs, and statistic explanation that help your security professionals reduce clutter, gather information quicker, spot threat activity early and act swiftly.



We're continuing to improve our product UI to provide users an excellent experience with the application and overall usability. Please expect more powerful UI features coming in next releases.

New compression archive engine for archive sanitization

In previous version, MetaDefender Core's archive engine implemented four functions including decryption, extraction, encryption and compression that with a high volume of files it takes time to process them all. For the purpose of enhancing archive handling performance, this release comes with a separated compression archive engine, which dedicatedly processes archive sanitization (compression and encryption). It is also created to facilitate maintenance and further feature development.

Archive sanitization feature now requires Deep CDR license

As to respect sanitization practice and ensure unknown threats prevention as designed, now starting on version 5, it's mandatory to have Deep CDR licensed for archive sanitization feature enablement on MetaDefender Core workflow configurations.

Archive handing related engines changed

The legacy "Archive" engine on MetaDefender Core v4 will be changed its name to "Archive Extraction" engine, even when you are still on MetaDefender Core v4.

A brand new "compression archive engine" is to dedicatedly handle archive sanitization in v5 (archive compression), that helps relevant archive handling performance improvement, easier for maintainance and further feature development.

Notes when you are still on MetaDefender Core v4 (not upgraded yet)

The legacy "Archive" engine (now name changed to "Archive Extraction" engine) on MetaDefender Core v4.x will still cover both extraction and compression archive tasks as usual.

Even though your MetaDefender Core v4.x might now have a new "Archive Compression" engine auto downloaded, but that engine cannot be deployed (marked as "permanently failed" in relevant engine status check API). This behavior is expected, and not causing any impact to MetaDefender Core processing flow, or its performance though.

- Just in case your client integration has a check for engine status before triggering scan request, then you might want to disable "Archive Compression" engine status on MetaDefender Core v4 using REST API. Detailed instruction: Archive Compression Engine Permanently Failed on MetaDefender Core v4
- When you later upgrade to MetaDefender Core v5 from v4, then you might need to manually re-enable "Archive Compression" engine to make it active and running on MetaDefender Core v5.

Notes for licensing

For offline deployment (MetaDefender Core is not connected to the Internet directly for online update), you will need to re-activate again on OPSWAT portal to retrieve latest engine list in .yml file including "Archive Compression" engine if you are currently licensed with Deep CDR.

Meanwhile, for online deployment, this manual procedure is not required since MetaDefender Core will cover that itself.

Import & export configurations

Not only workflow settings like in previous versions, but also all other general settings (update, security, data retention, server, hashes to skip) and most importantly, user settings, will be able to export and import on MetaDefender Core version 5.

Learn more at Import/Export configuration

Backward compatibility supported

The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

Windows application event logging disabled by default

Applicable only to Windows OS based MetaDefender Core.

Help avoid duplicated logging with separate file option, and improve overall processing performance. Of course, when being still valid to you, you are still supported to enable it back.

CSRF token header renamed

CSRF token originally created for security strengthening and being used by MetaDefender Core management console UI since version 4.21.0 via header `oms_csrf_token`

To get it compliant with some load balancers which do not accept underscore in header name by default, and save you from additional load balancer configuration step, we have now changed the header name to `oms-csrf-token`

This change does not impact to any REST API based client integration.

More comprehensive and accurate error messages on REST APIs

Impacting on following REST API endpoints:

- `PUT /admin/config/session`
- `PUT /admin/config/update`
- `PUT /admin/config/scan`
- `PUT /admin/config/skip`
- `PUT /admin/config/auditlog`
- `PUT /admin/config/webhook`
- `PUT /admin/config/rule`

Upload time and queue time logic updated in JSON scan response

Impacting on following REST API endpoints:

- `GET /file/{data_id}`
- `GET /hash/{md5/sha1/sha256}`
- `GET /archive/{data_id}`
- `POST /file/{async}`

Added new field:

- `receive_data_timestamp`: The timestamp when upload progress started (first byte received) (in milliseconds)
- `upload_time`: Total time elapsed for upload process (in milliseconds)

Modified existing field (new logic):

> **Backward compatibility supported**
> The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

- `upload_timestamp`: The timestamp when upload progress finished (all bytes received) (in milliseconds)
- `queue_time`: Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue time = start_time - upload_ timestamp) (in milliseconds)

Release version 5.0.0 also comes with variety of outstanding features and improvements.

For detailed information about our latest release please view our Release notes

Installation

Starting with MetaDefender Core 4.20.0, the centralized/shared database deployment model is supported (all MetaDefender Core instances share the same database), along with traditional standalone database mode (every MetaDefender Core instance has its own database to manage).

- Standalone mode
- Shared Database Mode

Installation

1. Download the package of your choice from My OPSWAT
2. Install the package on your computer via the Command Line or via the Wizard
3. Open a web browser and point to http://<server name or IP>:<port>
    - The default port is **8008**
    - In case of problem check Inaccessible Management Console page
4. Complete the required steps of the basic configuration wizard
5. You must activate this deployment to use its features

Installation notes

- If the MetaDefender Core package **dependencies are not installed** on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.
- During installation the databases might need to be upgraded. This could take noticeable time depending on database size (eg.: length of scan history).

Standalone mode

Wizard Installation

The Install Wizard is only for the Windows installer (.msi file).

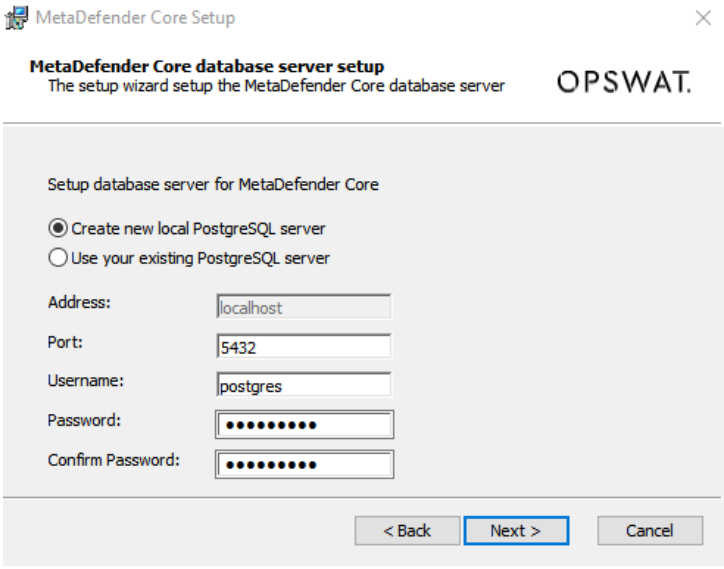
Since MetaDefender Core 4.19.0, we have switched to using PostgreSQL DBMS which needs a dedicated database server. MetaDefender Core supports two modes to setup PostgreSQL server:

Local (Bundle) PostgreSQL server

MetaDefender Core will install a new PostgreSQL server locally in the same box with the product.

> **PostgreSQL user credentials**
> Unicode characters are NOT supported for "Username" and "Password"
>
> "Password" does not accept # (hash) as starting character (e.g. #ABC)

> **The ignition file**
> If the ignition file exist in the designated location, then MetaDefender Core will use the details defined in the ignition file, rather than what is passed via the installation wizard.
>
> More info about the ignition file based installation: Command Line Installation

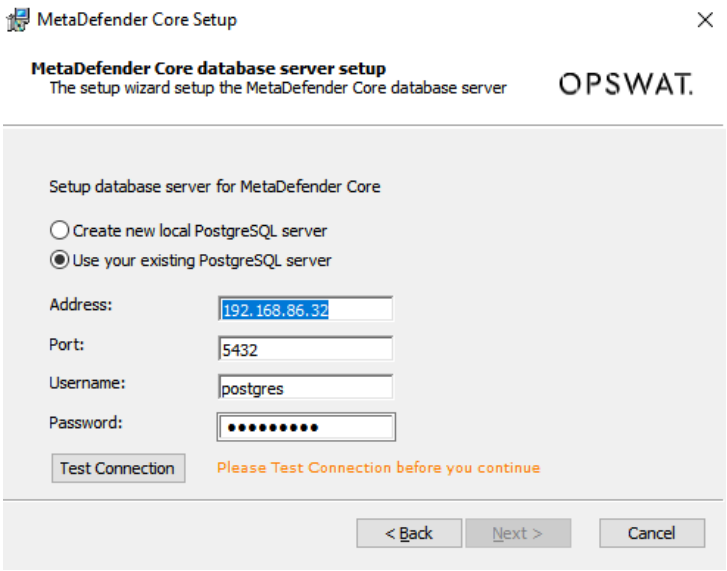


Remote (Existing) PostgreSQL server

MetaDefender Core will connect to remote pre-installed PostgreSQL running remotely.

> **PostgreSQL connection configuration**
> "Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

> **PostgreSQL user privilege requirements**
> MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.
>
> Details: PostgreSQL User Privilege Requirements



Command Line Installation

Ignition File

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Make sure to create the ignition file, if it doesn't exists before you go ahead and install MetaDefender Core via command line.

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

MetaDefender Core supports two modes to setup PostgreSQL server:

1) Local PostgreSQL server

- MetaDefender Core will install a new PostgreSQL server locally, next to the product.
  - A sample iginition file for PostgreSQL server information (***Only non-Unicode characters supported for "user" and "password"__***):

ometascan.conf bash

```
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
```

2) Remote PostgreSQL server

> **Info**
>
> **PostgreSQL version 12 (at least 12.3) or version 14 is required.**
>
> According to PostgreSQL notice, version 12 will stop receiving fixes on November 14, 2024, so we suggest that you use version 14.

MetaDefender Core will connect to a pre-installed PostgreSQL server running remotely and will setup its database.

Make sure to follow steps at Open Connection On PostgreSQL Server first, so that your remote existing PostgreSQL server to accept upcoming connection from MetaDefender Core.

A sample ignition file for PostgreSQL server information:

ometascan.conf bash

```
[dbserver]
type=remote
host=192.168.86.32
port=5432
user=existingserver_admin_user
password=existing_server_pass
```

After installing MetaDefender Core successfully, you may want to remove all credentials info created for [dbserver] section in ignition file for security reason.

> **Warning**
> If the MetaDefender Core package dependencies are not installed on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.

Debian package (.deb)

bash

```
sudo dpkg -i <filename> || sudo apt-get install -f
```

On Red Hat Enterprise 7.x Linux / CentOS package (.rpm)

bash

```
sudo yum install <filename>
```

For systems which enabled GPD check flag:

bash

```
sudo yum install --nogpgcheck <filename>
```

Red Hat Enterprise Linux 8.x package (.rpm)

Install the latest PostgreSQL repository RPM:

> **Note**
> Due to the shorter support cycle on Red Hat Enterprise Linux, all supported versions of PostgreSQL are not available on this platform. The PostgreSQL Yum Repository will integrate with the normal systems and patch management, and provide automatic updates for all supported versions of PostgreSQL throughout the supported lifetime of PostgreSQL. Reference: https://www.postgresql.org/download/linux/redhat/

bash

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

In case installing a remote PostgreSQL on either the same MetaDefender Core server or another server, you need to install the following additional package. PostgreSQL version 12.3 or newer is recommended.

bash

```
sudo dnf install -y postgresql12-contrib
```

Install MetaDefender Core:

bash

```
sudo dnf install -y <filename>
```

Windows package (.msi)

On Windows systems it is possible to install the product by running the corresponding .msi file.

From command line interface it is also possible to install the product by executing

bash

```
msiexec /qn /i <msi file name> <option key>=<option value>
```

where the possible keys and their default values are the following:

Key	Default Value	Description
INSTALLFOLDER	\Program Files\OPSWAT\MetaDefender Core	Customize installation folder for product Example: `INSTALLFOLDER="D:\Products"`
RESTADDRESS	*	REST interface binding IPv4 or IPv6 address ('*' means that service listens on all IPv4 and IPv6 interfaces)
RESTPORT	8008	REST interface binding port
REPORT_ENGINE_ISSUE	true	Enable reporting of engine issue count. (possible values: "true" or "false").

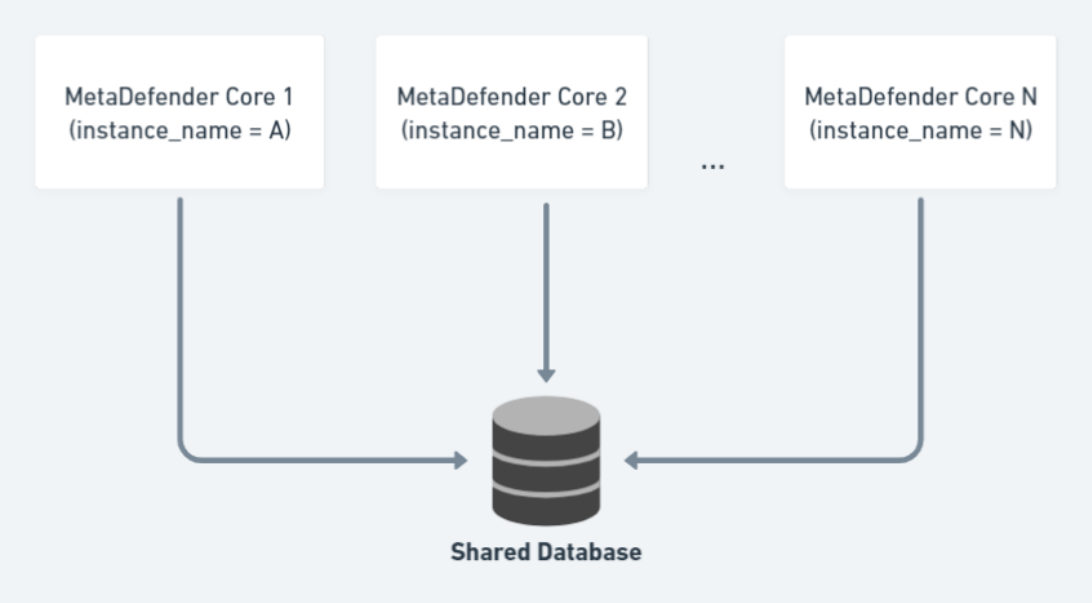
> **Note**
> Reporting of engine issue count
>
> If reporting of engine issue count is enabled, MetaDefender Core server will send only the **number** of initialization errors and **number** of unexpected stops for the specific db/engine version. This information is sent over a HTTPS channel when the product downloads the latest package descriptors. This information is used for early detection of any specific 3rd party engine quality issues.

For details on using msiexec please consult Windows installer documentation.

Shared Database Mode

Starting with MetaDefender 4.20.0, you can use a centralized database deployment model, where all MetaDefender Core instances can share the same external database.

That helps auto-sync all product configurations (**except Single-Sign-On** that requires unique settings for each Core), processing history, live/on-going scan result query, audit, statistics data.



This article demonstrates how to install and setup MetaDefender Core 4.20.0 or newer with shared database mode.

Important notes & current limitations:

Currently not supported

- Does not support Central Management use-case with shared DB mode.
- Does not support to switch from shared database mode back to standalone database mode.
- Does not support to change instance_name of MetaDefender Core in shared database deployment model after installation.
- Does not support upgrading older MetaDefender Core from non-persistent mode to using the shared database mode.
- Does not support upgrading older MetaDefender Core from standalone mode to using the shared database mode.
- Does not support to upgrade multiple MetaDefender Core instances at once in shared database mode
    - Stop all Core instances' scanning service before performing the upgrade.
    - Upgrade each Core server one by one instead.

- For each Core, when the upgrade finished, then you can start using that Core for scanning without waiting for the other Core instances' upgrade to be done.

> **Disclaimer**
> MetaDefender Core does not support mode switching.

Requirements for shared database setup:

- The central remote PostgreSQL server (for shared database) could be utilized with a built-in (native) PostgreSQL database server installed with one of the MetaDefender Core instances in the network. Or using a dedicated existing PostgreSQL database server (see detailed requirements for PostgreSQL version at Recommended System Configuration).
- Before setting up, make sure to follow steps at Open Connection On PostgreSQL Server so that all MetaDefender Core instances could connect to the central remote PostgreSQL server.
- MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core, details: PostgreSQL User Privilege Requirements
- All MetaDefender Core instances must be installed on the same Operating System (Windows / Linux).
- The first MetaDefender Core instance connects to the shared database must be ready first (license activated, users/groups/roles created) before setting up other MetaDefender Core instances to connect and use that shared database.

Engine update:

- If engine updates is configured to be via Folder source, all MetaDefender Core instances must be able to access to the configured update folder, and the option DELETE FILES AFTER IMPORT will be unchecked and prevented for modification.

Authentication:

- Single-Sign-On (SSO) authentication settings (if using) cannot be shared among MetaDefender Core instances because SSO requires an unique login URL for every different Core.
- If an user is locked for login on a certain MetaDefender Core instance(e.g. exceeded the number of failed login times), that user is not prevented for logging in on any other instances.
- If an user is locked for sign-in on a certain MetaDefender Core instance (e.g. exceeded the number of failed login times), if the admin releases lockout on that user on that MetaDefender Core instance, that user is still not able to login on other MetaDefender Core instances.

Sanitized / DLP / Quarantined location changed after MetaDefender Core install:

Windows:

Shared storage path will be stored in 3 strings under key global in registry: quarantinepath, dlppath, sanitizepath as below

(Default)	REG_SZ	(value not set)
dbmode	REG_SZ	4
dlppath	REG_SZ	\\10.40.50.100\test
instance_name	REG_SZ	481rvj
quarantinepath	REG_SZ	\\10.40.50.100\test
restaddress	REG_SZ	*
restport	REG_SZ	8008
sanitizepath	REG_SZ	\\10.40.50.100\test

You can set either 1, 2 or 3 different paths for the 3 settings above (means that quarantined, DLP and sanitized files will be stored in same or different storages corresponding). In case shared storage is not set in installation, values of 3 settings above will be empty.

Restart MetaDefender Core service after changed.

Linux:

Open file `/etc/ometascan/ometascan.conf` and set shared storage path for 3 keys quarantinepath, dlppath, sanitizepath under section [global]

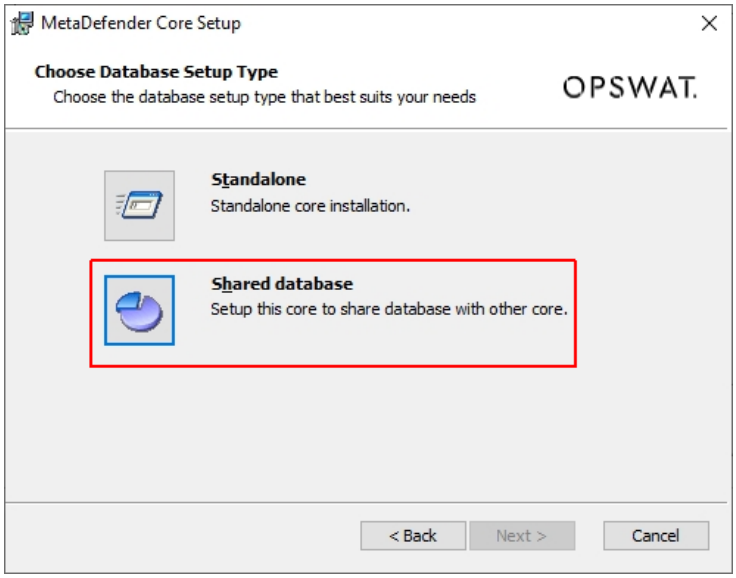
Restart ometascan service after changed.

Wizard Installation (Windows Only)

> **Info**
> Only applicable to MetaDefender Core 4.20.0 or newer using wizard on Windows OS.

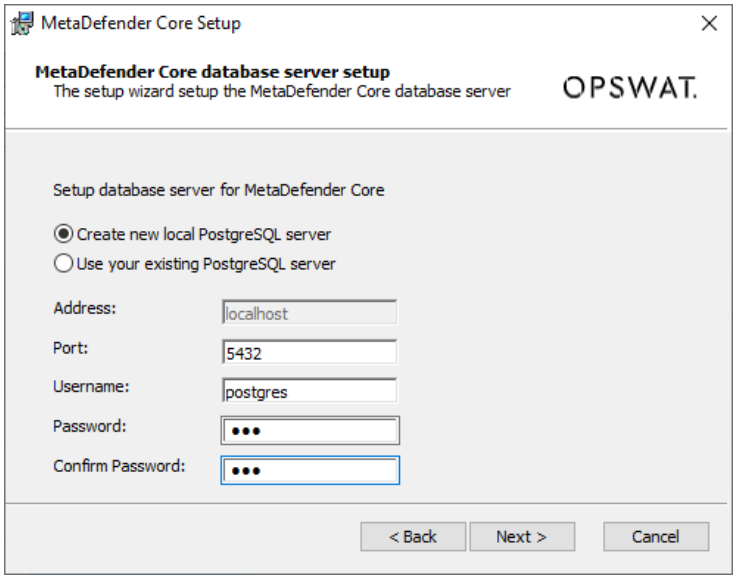
Clean install scenario

1. **Choose Shared database option:**

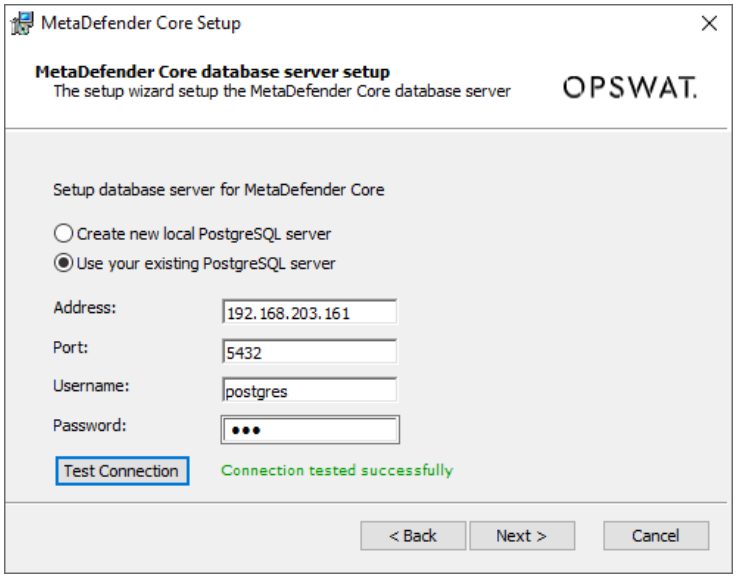


2. **Setup database server for MetaDefender Core:**

You can choose to create a native / built-in PostgreSQL server and setup the shared database on (then other MetaDefender Core instances will want to connect to that PostgreSQL server afterwards)



Or choose to connect to a pre-setup existing PostgreSQL server:



PostgreSQL connection configuration

"Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

PostgreSQL user privilege requirements

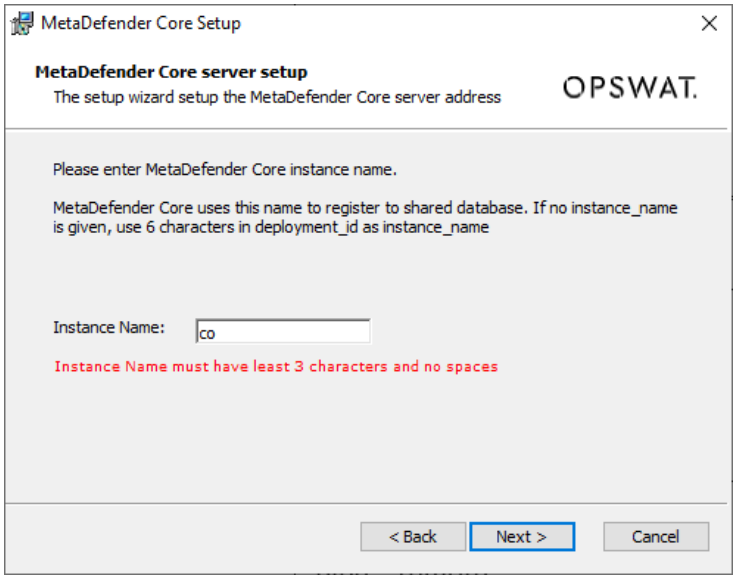
MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.

Details: PostgreSQL User Privilege Requirements

3. **Create an unique instance name for that MetaDefender Core (different from other MetaDefender Core instances):**

Instance name must contain at least 3 characters and no white space.

If you choose to leave it blank, which means it will use the default instance name with 6 characters extracted from that MetaDefender Core deployment ID.

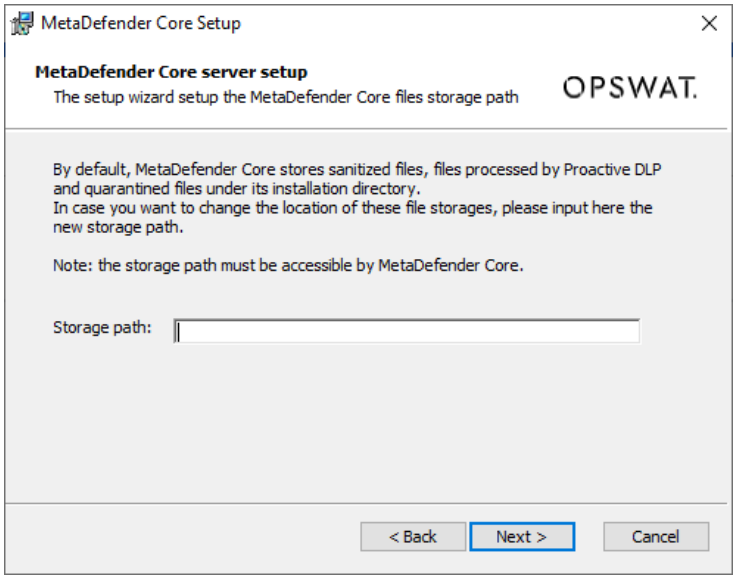


4. **Input the file storage path (sanitized, DLP, quarantined files)**

In the shared database model, all MetaDefender Core instances could use the same folder path to store all their own sanitized, DLP, quarantined files after processed.

The default location is: ***<MetaDefender Core installation folder>\data\ {sanitized|dlp|quarantine}***

If you wish to use that default location, just leave the input box blank and press Next.



Otherwise, you can specify a different location of sanitized/dlp/quarantined files. It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core.

For example:

- C:\new_folder
- \192.168.1.1\new_folder

Command Line Installation

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Clean install scenario

1. Create ignition file if not existed

- Windows: **C:\OPSWAT\ometascan.conf**
- Linux: ***/etc/opswat/ometascan.conf***

An example of ignition file:

ometascan.conf bash

```
eula=true

[global]
dbmode=4
instance_name=MetaDefenderCoreA
storage_path=D:\path\

[user]
name=admin
password=admin
email=admin@local

[dbserver]
type=remote
host=192.168.1.1
port=5432
user=postgres
password=your_own_password
```

Field	Required	Mapping to wizard UI	Possible values
global/dbmode	optional default is 1 (standalone)		1 - Standalone 2 - Central Hub deployment 3 - Non-persistent **4 - Shared**
global/instance_name	optional default is 6 characters of deployment ID		Unicode and number At least 3 characters No whitespace
global/storage_path	optional default is empty		Path to storages (DLP, quarantined, sanitized files) It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core. For example: • C:\folder • \192.168.1.1\folder

2. Run installers using command line to install

Please refer to Command Line Installation for installation command.

Deployment automation support

Note

An Ignition file could be utilized for either / both of following use-cases:

1. Pre-define PostgreSQL server information for MetaDefender Core to connect and setup database: only required on Linux installation, or Windows with command line installation. Learn more details at:
    a. Standalone Mode > Command Line Installation (applicable on version 4.19.0 or newer).
    b. Shared Database Mode > Command Line Installation (only applicable on version 4.20.0 or newer).
2. Automating to accept EULA, create a default local admin user and import MetaDefender Core configurations upon installed: Those steps could be automated using ignition file described in this page.

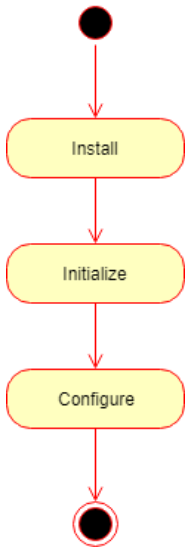
Ignition file location to create:

- **Windows**: `C:\OPSWAT\ometascan.conf`
- **Linux**: `/etc/opswat/ometascan.conf`

The product supports fully automated deployment. It means that it can be installed and configured with no human interaction.

The automated deployment can be split to three steps on a high level:

1. Installation
2. Initialization
3. Configuration



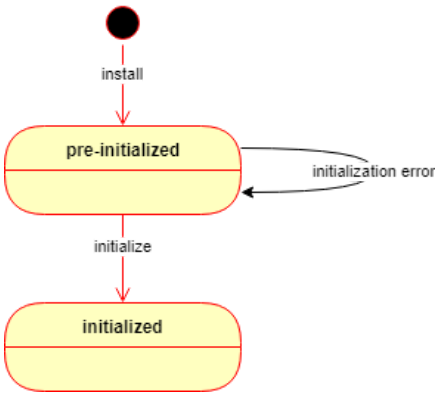
Installation

To automate the installation, install the product from the command line and provide the installation-time options as parameters to the installer. For further details see Command Line Installation

After the installation is complete, the product starts up and waits in a pre-initialized status. The product may be initialized in two ways:

1. Manually using the Wizard Setup or
2. Automatically using an *ignition file* (see below).

> **Warning**
> If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried after fixing the problem (e.g. placing the ignition file in its lookup location) and restarting the OPSWAT MetaDefender Core service.
>
> Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

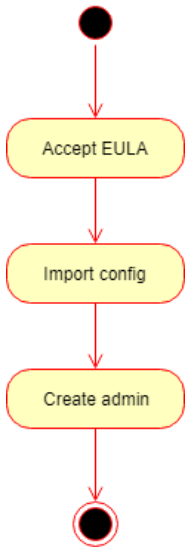


Initialization

Initialization is the process of bringing the product to an operable status.

Basically the initialization consists of the following steps:

1. Accept the End User License Agreement (EULA),
2. Import product configuration and
3. Create the first administrator user account.



Ignition file

The initialization process can be configured in a file called the *ignition file*.

> **Warning**
> In Linux, the ignition file and the folder /etc/opswat should have the permission 755

Ignition file location

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

The ignition file must be in conf format.

ometascan.conf bash

```
eula=true
[user]
name=admin
password=admin
email=admin@local
apikey=44c96cbb4e4e6c0a1d16bcb3c3da24def317
[config]
import=config_package.zip
import_password=123
[security]
session_cookie_samesite=Lax
[internal]
skip_startup_pages=true
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
private_username=my_internal_db_user
private_password=mypassword
[ocm]
ocm_url=https://my.opswat.com
reg_code=XXXXXXXX
```

Ignition file fields

The ignition file accepts the following fields:

Section	Key	Required	Accepted Values	Description
	eula	Optional (mandatory when having [user] section)	true false (default)	Whether to accept the End User License Agreement. This key must be set at the top of ignition file. This key must be set to "true" to accept the EULA. Any other value will cause the initialization to fail. When having [user] section defined, then this key must be set.
[user]		Optional		Initial administrator user account properties. The Administrator role is granted to the account.
	name	Optional	string	User name for the initial administrator user account.
	password	Optional	string	Password for the initial administrator user account. WARNING! Clear text password The password in this configuration file must be stored in its clear-text format and as so it may be visible for unauthorized parties.
	email	Optional	string	E-mail address for the initial administrator user account.
	apikey	Optional	string	API key for the initial administrator user account.
[config]		Optional		
	import	Optional	string	Path to a file in json/zip format that contains a previously exported configuration to be imported. Zip-format file contains a full configuration including user management settings, so it will overwrite [user] section.
	import_password	Optional	string	Password to decrypt .zip configuration package.
[security]		Optional		

Section	Key	Required	Accepted Values	Description
	session_cookie_samesite	Optional	Strict Lax (default) None	Declare if session cookie should be restricted to a first-party or same-site context.
[internal]		Optional		
	skip_startup_pages	Optional	true false (default)	Skip all welcome and other startup pages (e.g. upgrade result) when installing / upgrading MetaDefender, and going straight to the product index default page.
[dbserver]		Optional		
	type	Mandatory when having [dbserver] section	local remote	Where the PostgreSQL database server should physically locate. "remote" means PostgreSQL server and MetaDefender Core are not in the same machine.
	host	Mandatory when having [dbserver] section	string	IP address / domain name of the server where PostgreSQL server locates. "localhost" can be used when applicable.
	port	Mandatory when having [dbserver] section	number	Port of PostgreSQL server is listening for connections from clients (i.e. MetaDefender Core).
	user	Mandatory when having [dbserver] section	string	PostgreSQL server's user. SUPERUSER privilege is required for MetaDefender Core to setup its database and extensions for the first time. **Only non-Unicode characters supported.**

Section	Key	Required	Accepted Values	Description
	password	Mandatory when having [dbserver] section	string	PostgreSQL server's user credentials. **Only non-Unicode characters supported.**
	private_username	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
	private_password	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
[ocm]		Optional		
	ocm_url	Optional	string	URL to My OPSWAT server.
	reg_code	Optional	string	A registration code to enroll to My OPSWAT server.

Detailed initialization process

1) After the product has been started, it looks for the ignition file in the designated location.

Windows: C:\OPSWAT\ometascan.conf ** Linux: /etc/opswat/ometascan.conf

** 2) If an ignition file is found, then

2.1) It gets validated, and if it is valid, then based on the information found in the ignition file:

- The EULA is accepted,
- The configuration is imported,
- The administrator account is created.
- If any of the above steps fails, then the error is logged, and the initialization gets terminated.

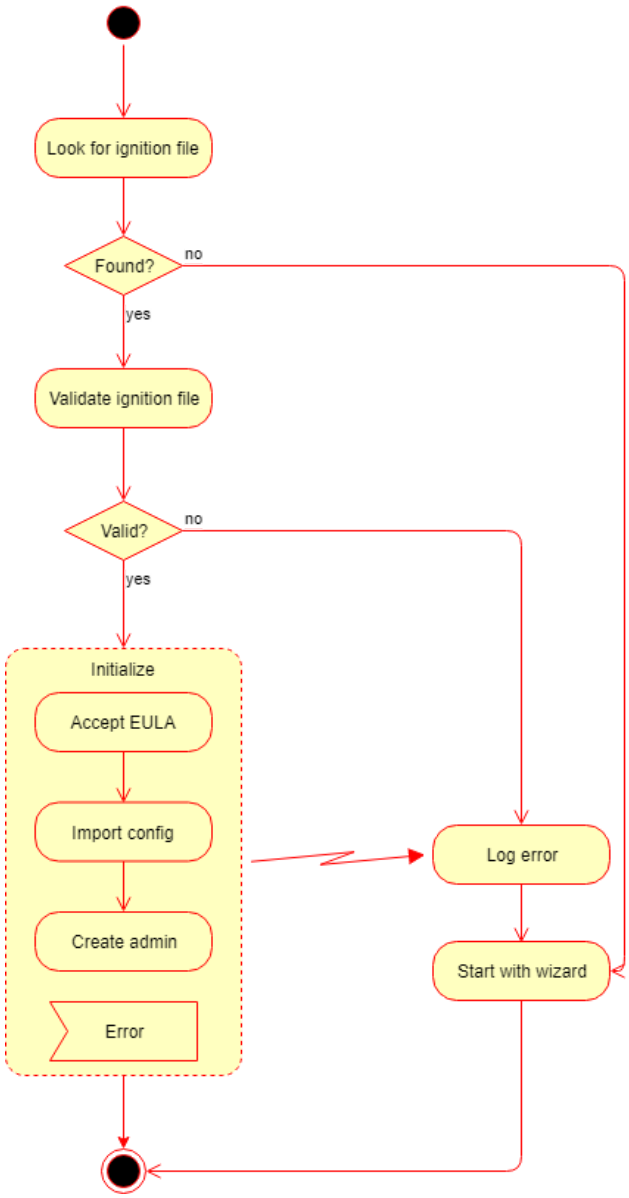
> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

2.2) If it is not valid, then the error is logged, and the initialization gets terminated.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

3) If there is no ignition file, then no initialization is performed.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.



Warning

If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried fixing the problem (e.g. placing the ignition file to its lookup location) and restarting the OPSWAT MetaDefender Core service.

Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

Configuration

After the initialization is complete, the product is ready with the default and the imported configuration.

This configuration can be later changed calling the configuration API functions. For further details about the API see Configuration related APIs API

CIS Level 1 Guidelines

> **Support Red Hat Enterprise Linux 9**
> For more details about Center for Internet Security (CIS) please refer to this document

Instruction steps

I. Install OpenSCAP

bash bash

```
yum install openscap-scanner scap-security-guide
```

II. Generate a result file and a HTML report using OpenSCAP scanner tool

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

III. Remediation of CIS Level 1 issues

Generate a remediation script based on the `ssg-rhel9-ds.xml` file:

bash

```
oscap xccdf generate fix --profile
xccdf_org.ssgproject.content_profile_cis_server_l1 --fix-type bash --output
remediations.sh /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

And execute remediation script.

bash

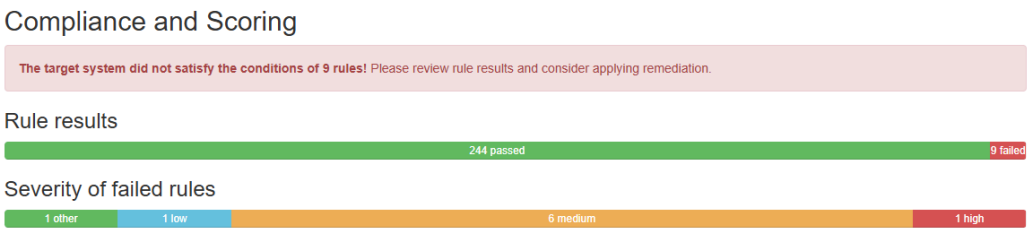
```
./remediations.sh
```

IV. Review the results after remediation

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

Example:



Notes

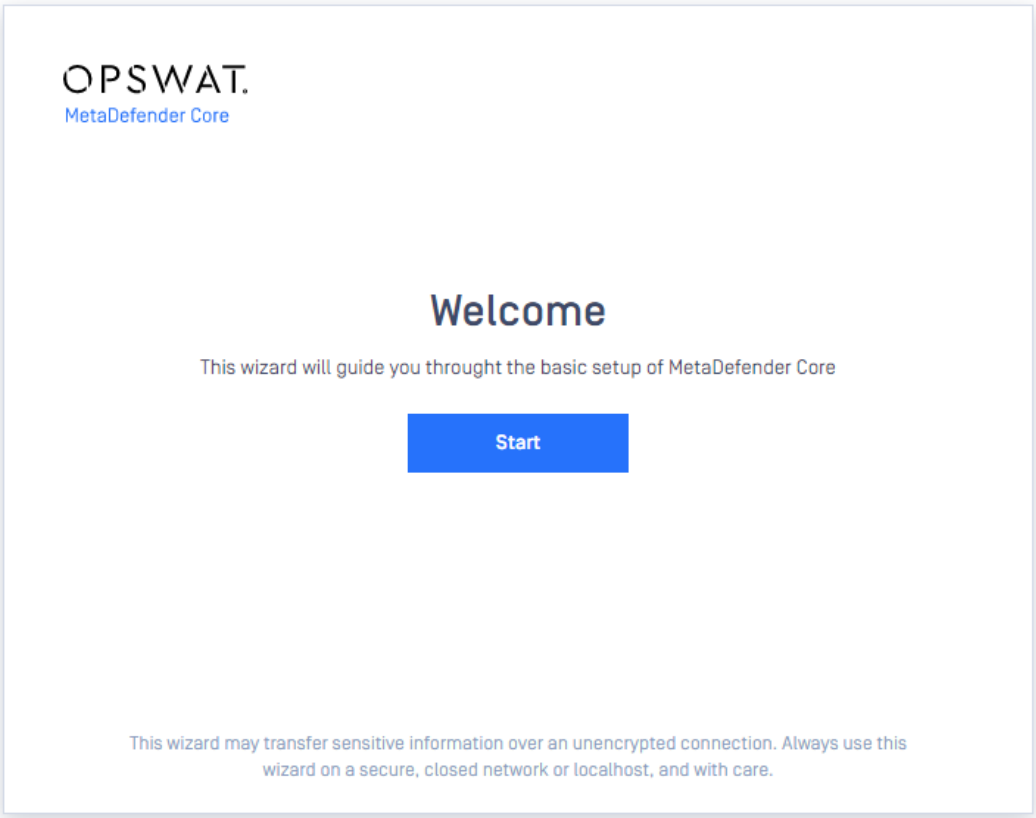
- CIS Level 1 requires `/tmp` folder to be mounted in a separate partition. Please ensure that that new partition have enough disk space for MetaDefender Core to run.
- CIS Level 1 requires that "Ensure No World-Writable Files Exist".
    - For now, when freshly installing MetaDefender Core, all its binary files meet the requirement.
    - When installing/updating engines, some engines might create additional files for its operation, and it might violate this requirement. In this case, you need to again execute the remediation script in the step III.

Wizard Setup

Introduction

Access the MetaDefender Core management console for the first time (i.e. http://localhost:8008), you are to complete a quick walkthrough for wizard setup in order to be able to use the product.

The administrative management console will be available only upon you have successfully finished this wizard stage.

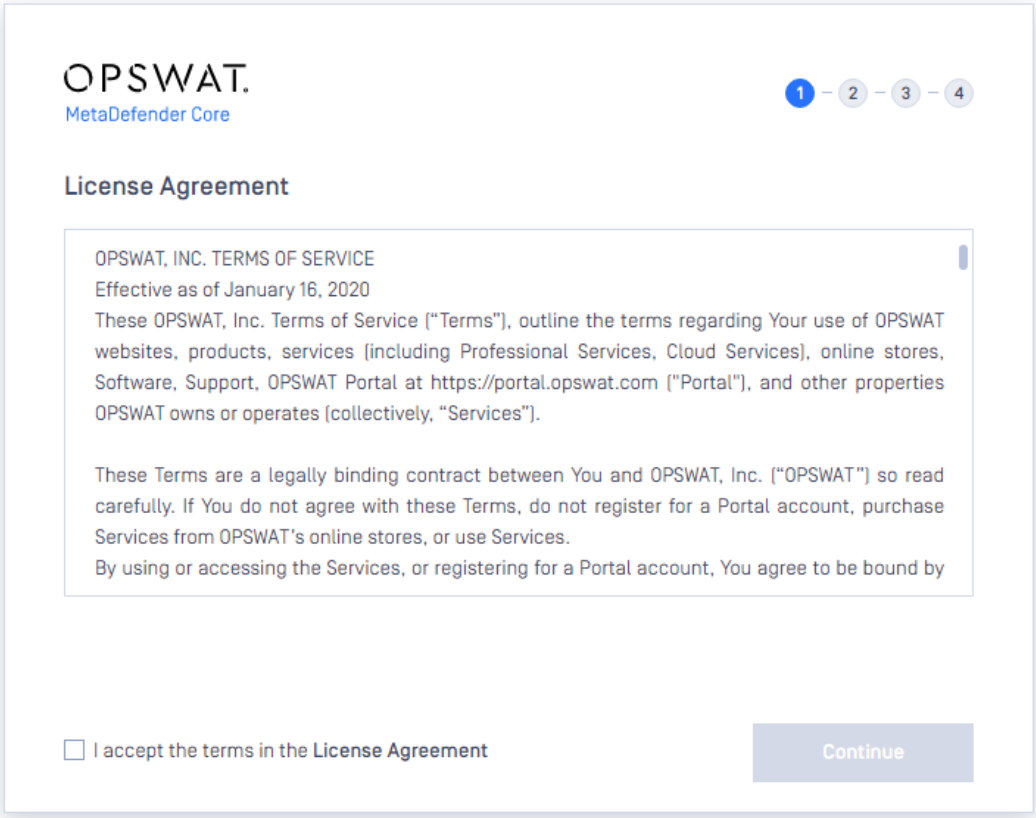


Warning

This wizard may transfer sensitive information over an unencrypted connection. Always use this wizard on a secure, closed network or localhost, and with care.

Basic configuration steps

End-User License Agreement



In the first page you can find the End-User License Agreement. You have to accept the terms before moving on.

Please read through the EULA (https://docs.opswat.com/policies/privacy-policy/portal-privacy-policy) carefully and if you agree with it, check `I accept the terms in the **License Agreement**` and click `Continue` to proceed.

Local Admin User Setup

The next step is to set up a default local administrator account.

This account will be the first one being able to access the administrative management console, and to create accounts for other users. You have to fill out all fields in this page to be able to move forward. When you are done, click NEXT to continue.

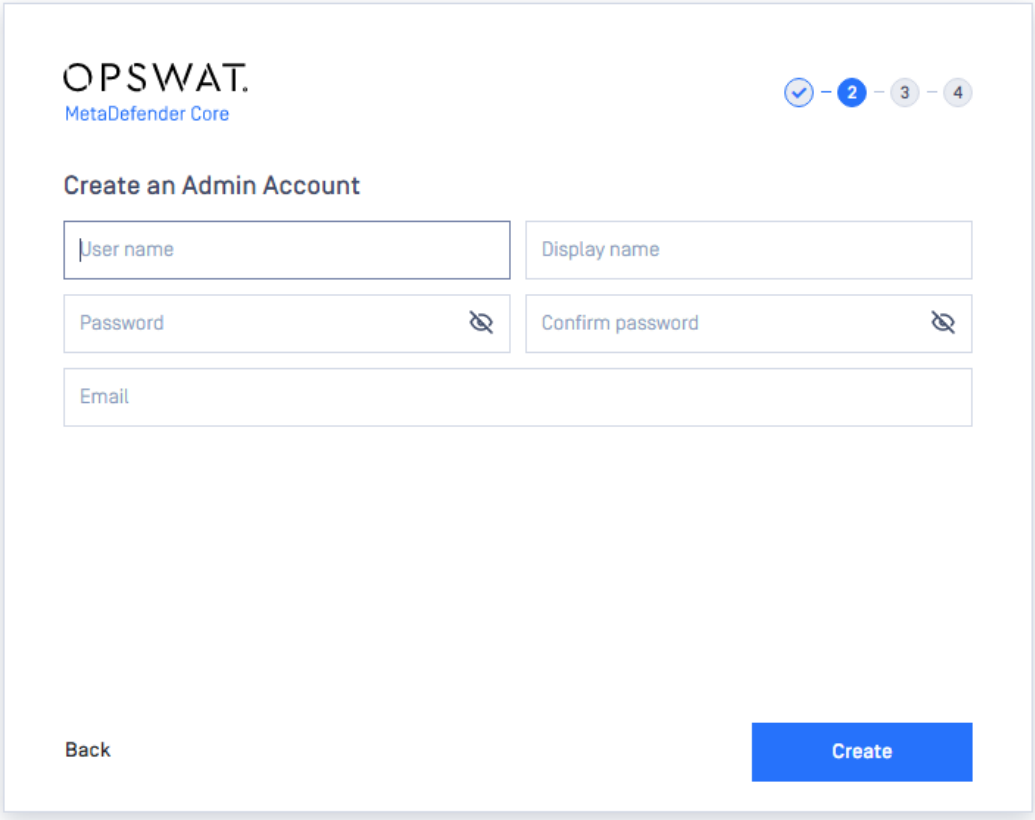
Info - User Directory

The administrator account, that is created via the basic configuration wizard, is always added to the LOCAL user directory as a member.

Learn more about user directory: User directories

The following information is required for the administrator account:

ACCOUNT NAME	The unique name of the account that is used at the time of login and in log messages for accountability.
ACCOUNT DISPLAY NAME	Name of the person bound to this account. This name (appended to the name of the account's user directory) is displayed in the top right corner of the administrative management console.
PASSWORD	Password of the user bound to this account that is used at the time of login. Passwords sent clear-text As long as TLS is not configured for the basic configuration wizard, passwords are sent clear-text over the network and may be disclosed to unauthorized parties. As a mitigation action: 1. Either use the wizard on *localhost* or on a direct network connection, or 2. Enabling HTTPS as soon as possible and change the password immediately if it has already been set.
EMAIL	Email address of the person bound to this account.



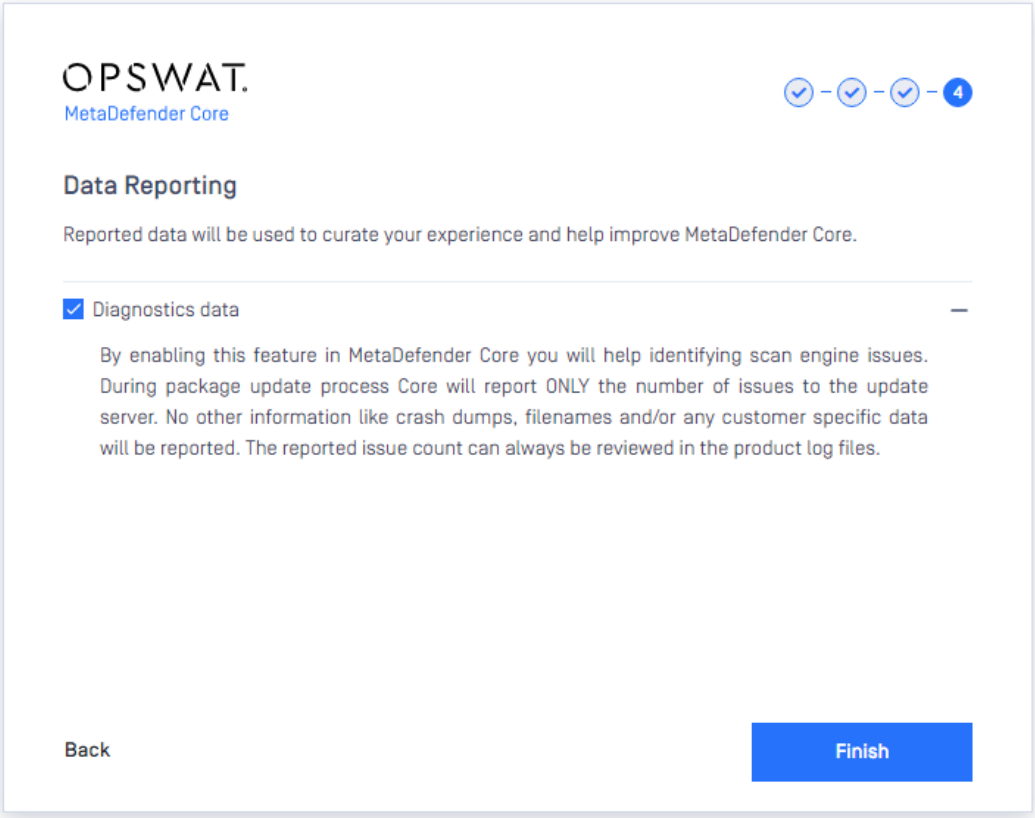
License activation

For license activation details see License Activation

Data Reporting

Enabling this option to allow MetaDefender Core collect its own engine issues and report back to OPSWAT for further analysis.

MetaDefender will not collect any other information.

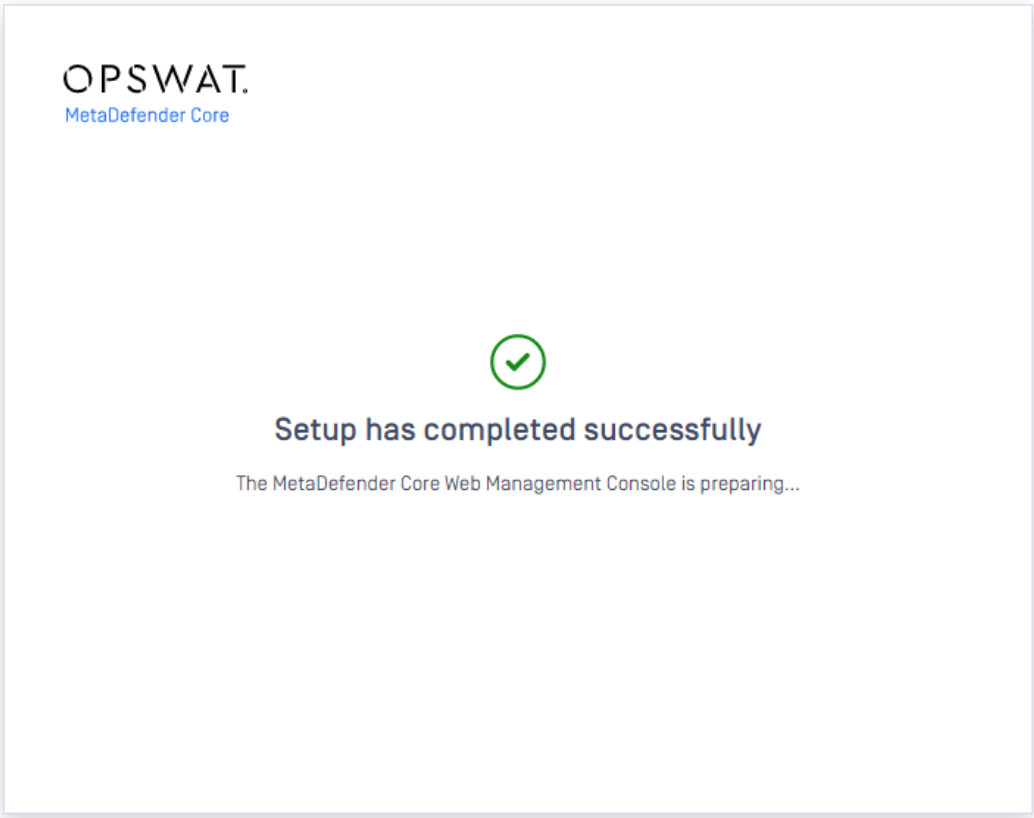


Wizard completion

After you have completed every steps you are ready to finish the wizard and start using the product. Click the `Fishish` button to complete the wizard.

> **Warning**
> The product's service will be restarted and the browser will be redirected to the administrative management console. This could take several seconds.

You can login to the Web Management Console with the administrator user that have just been created in the previous steps.



Setup HTTPS

Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network. Websites, like the Web Management Console, are able to use TLS to secure all communications between their servers and web browsers.

The TLS protocol aims primarily to provide confidentiality (privacy) and data integrity between two communicating computer applications.

Warning

No TLS for the wizard

By default, TLS is not enabled for the basic configuration wizard. As a consequence sessions between the wizard's backend and the browser may be insecure.

Info

Performing the same steps as for the Web Management Console, it is possible to set up TLS for the basic configuration wizard. Remember completing the TLS setup before launching the wizard.

For instructions to set up TLS see Enabling HTTPS

License Activation

##

To activate your installation go to the **Settings > License** menu in the Web Management Console. If you have no valid license, you will only see your installation's **deployment ID**. You will also see a warning in the Web Management Console header.

After successful activation, the product will start downloading the latest available scan engines and malware databases. You can follow the status of the scan engine installation on the **Inventory > Modules** page.

When all the licensed engines are installed, you can start using the MetaDefender Core to process files - File Processing.

Deployment ID

Each MetaDefender Core instance represented by an unique system fingerprint called `Deployment ID`

MetaDefender Core licensing is purely based on deployment ID activated. Therefore, everytime your MetaDefender Core is activated with a different deployment ID, then it will be captured as a different activation record accumulated into your total activation times allowed.

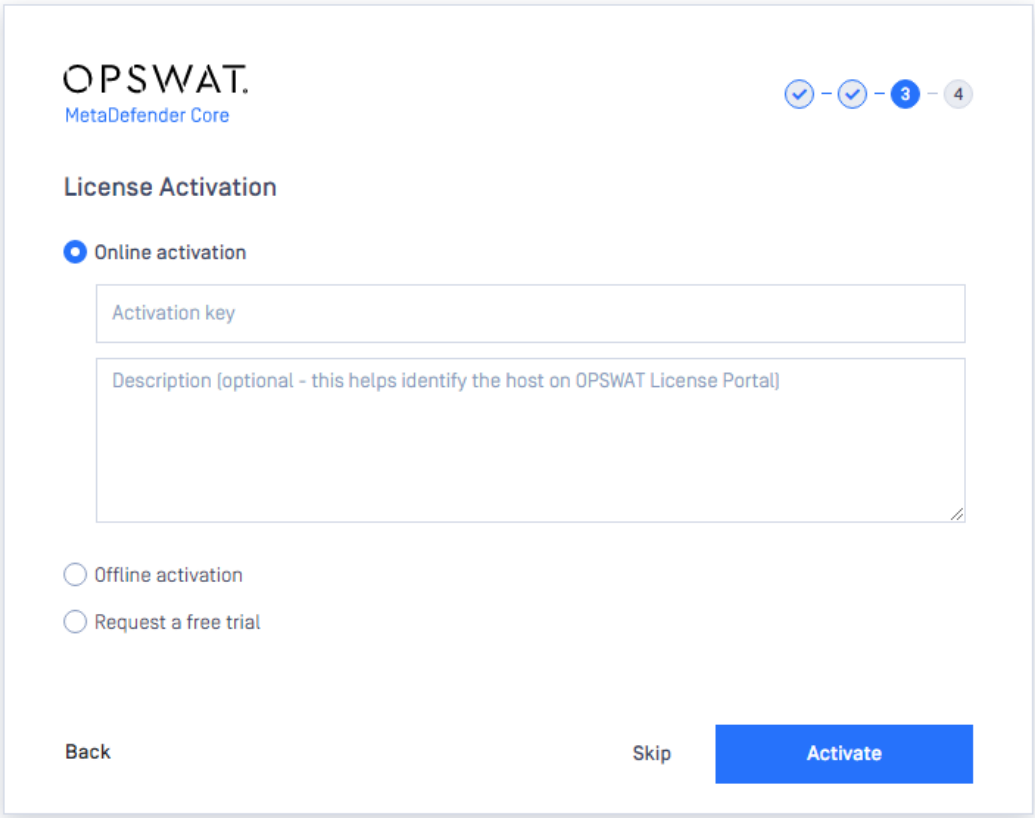
> **Notice**
> Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you alter the system where MetaDefender Core relies on (e.g. network card, OS hostname).
>
> Deployment ID is an UUID version 4 random based string. Learn more at: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you update the system where MetaDefender Core relies on.

Activation Modes

There are three different options for the product activation:

- Online activation: MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.
- Offline activation: Requiring you to upload a license file (.yml). That license file could be retrieved by activating your license via OPSWAT Portal. Follow the displayed instructions for details.
- Request trial key online: For evaluation purpose, you can receive an trial Activation key via email. Follow the displayed instructions for details.



Online activation

MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.

Required to input your Activation License key, and optional for description (but we recommend this for troubleshooting and auditing purpose if needed).

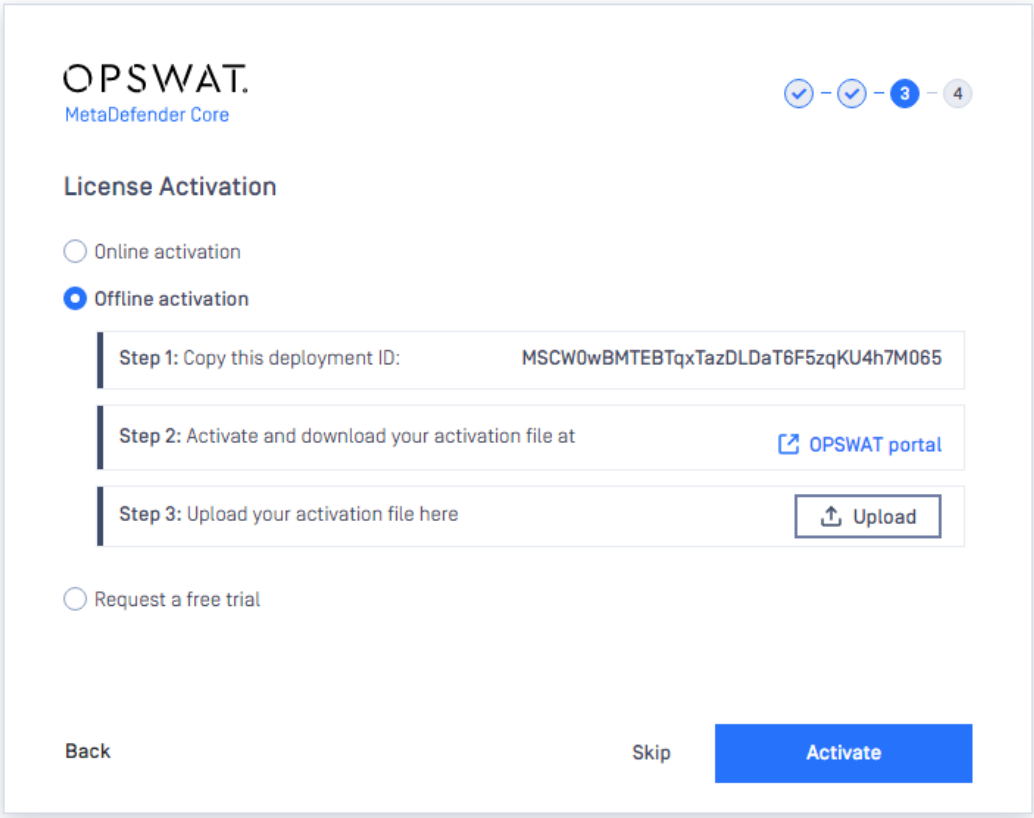
Warning

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts' for license activation:

- https://activation.dl.opswat.com Note: *IP address-based whitelisting on your firewall might fail after some time since OPSWAT uses CDN (Content Delivery Network) to faster delivery updates over the world, and IP address of edge servers might change over time.*

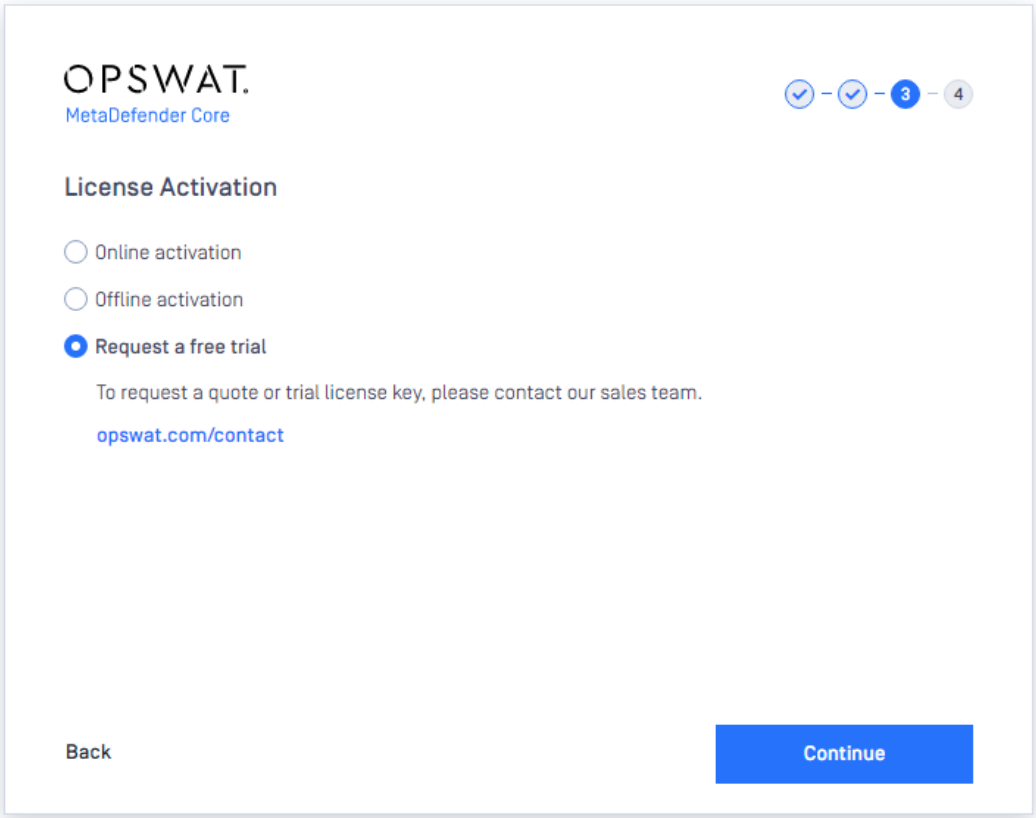
Offline activation

With no internet connection on the server the Metadefender v4 instance may be activated indirectly from a different machine, that has internet connection. The Deployment ID of the Metadefender v4 instance and the the Activation key received at the time of purchasing the product will be required. Follow the steps on the screen to activate the product offline.



Request trial key online

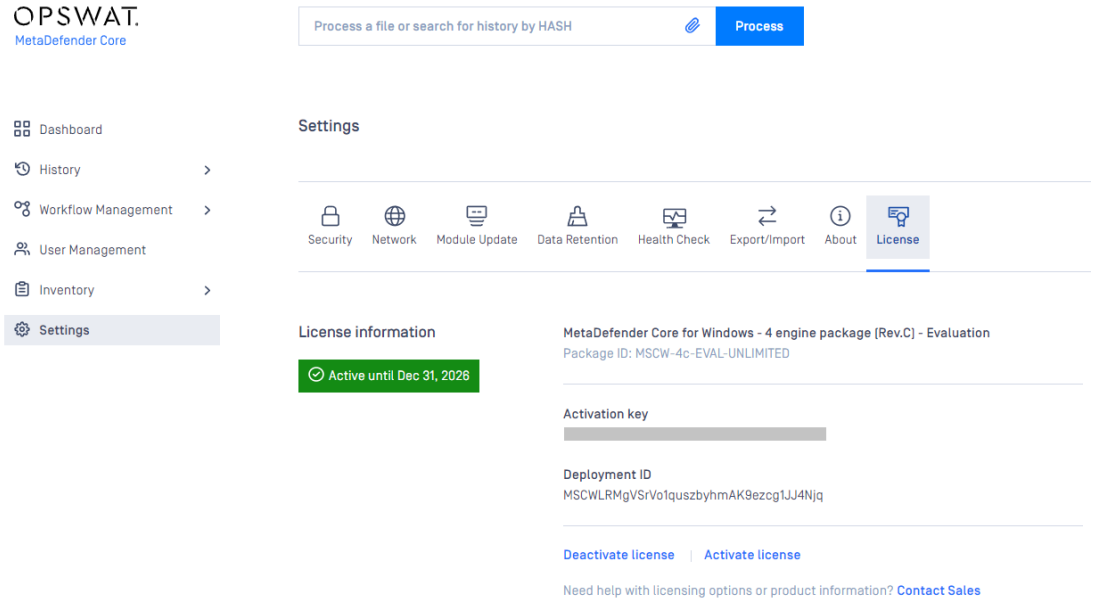
An evaluation license could be acquired by contacting our sales team.



Check Your License Details

For more license details and activating your MetaDefender Core, go to Settings > License menu tab on the Web Management Console:

- Product name: product name as on your order (e.g. MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation)
- Package ID: product identification as on your order (e.g. MSCW-4c-EVAL-UNLIMITED)
- Expiration: last day of license validity
- Activation key: key used for activating product
- Deployment ID: identification of this installation


OPSWAT.
MetaDefender Core
Process a file or search for history by HASH
Process
Dashboard
History
Workflow Management
User Management
Inventory
Settings
Settings
Security
Network
Module Update
Data Retention
Health Check
Export/Import
About
License
License information
Active until Dec 31, 2026
MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation
Package ID: MSCW-4c-EVAL-UNLIMITED
Activation key
Deployment ID
MSCWLRMgVSrVo1quszbyhmAK9ezcg1JJ4Njq
Deactivate license
Activate license
Need help with licensing options or product information? Contact Sales

Scripted license management

Using REST API calls there is an option for scripted activation and deactivation for MetaDefender Core servers with **no Internet connection or if your infrastructure is using automation to create/destroy MetaDefender Core instances**. In this chapter the steps of these two scenarios are described.

Requirements

- an installed MetaDefender Core instance without Internet connection
- another computer that has Internet connection and can run your activation scripts
- a manual or automated way to transfer data between the two computers

Activation steps

1. In order to activate MetaDefender Core, deployment ID and activation key are needed to generate the license file.
    a. Activation key should be purchased from OPSWAT.
    b. MetaDefender Core deployment ID can be queried by using REST API (for details see GET Get Current License Information API page)

> **Info**
> Save this deployment ID in your system, you might need this ID when the instance is unavailable at the time of deactivation.

2. Activate license and get the license file using the following URL, where <quantity> is 1.

activation call bash

```
curl -X GET https://activation.dl.opswat.com/activation?key=<activation-key>&deployment=<deployment unique-ID>&quantity=<quantity>
```

If the activation is successfully, a license file is downloaded. Save this file.

HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	license file	Activation was successfully.
200 Activation failed	error: '<user conform error message>'	Failed activation
200 Invalid parameter	error: 'Could not activate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not activate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
200 Invalid parameter	error: 'Could not activate your product because the quantity you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key, quantity or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Info
In case of any activation issue, contact OPSWAT support for help

3. Upload license file to MetaDefender Core: The license file should be uploaded to the MetaDefender Core v4 to activate the product. For details see page: POST Upload license key API

Deactivation steps

1. For deactivation, following infomation is required:
    a. Activation key
    b. MetaDefender Core deployment ID: can be queried by using REST API (for details see GET Get Current License Information API page)
2. Deactivate license using the following URL:

deactivation call bash

```
curl -X GET https://activation.dl.opswat.com/deactivation?key=<activation-
key>&deployment=<deployment unique-ID>
```

Response HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	result: ok	Successful deactivation
200 No active license found	error: 'Could not found any active license with the given parameters'	The license has not been activated yet or it has been deactivated already.
200 Invalid parameter	error: 'Could not deactivate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not deactivate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Important notes

Warning

Product activation is tied to several hardware, operating system and software parameters. In case of one or more major hardware or software parameter change the product might turn into deactivated status. Operating system updates, other software updates should not affecting the activation status.

If the product is online activated and has live Internet connection then in case of deployment ID change the product reactivates itself. If the product doesn't have Internet connection then the administrator is responsible to reactivate the product.

Because of the above administrator should consider licensing restrictions of the offline deployments if using any virtualization and/or containerization technologies.

File Processing

MetaDefender Core provides REST API interface for file processing business.

There are several ways to process files with MetaDefender Core:

- Analyze Files via Web Interface
- Process Files via REST API

Product Upgrading

> **Info**
> Simply run installer file, details could be found at Installation

> **Info**
> For command line mode, upgrading (e.g. on **Linux**), prior to running the installer file, please make sure to **create the ignition file** with PostgreSQL server information included. Details:
>
> - Standalone DB mode: Command Line Installation (Windows and Linux)
> - Shared DB mode: Command Line Installation

Recommendation for MetaDefender Core upgrade

Our goal is to help you execute the product upgrade continuously, efficiently, and correctly.

PostgreSQL service upgrade

- **Bundled PostgreSQL Service**: If you are using the bundled PostgreSQL server included in the installer, the PostgreSQL service will be automatically upgraded to the newer version (if available). No additional action is required for this process.
- **Remote PostgreSQL Service**: For remote PostgreSQL servers, customers are responsible for managing and upgrading the service independently. Upgrading MetaDefender Core does not update the remote PostgreSQL service.

Before the upgrade

Prepare a proper disaster recovery plan (DRP) including some common practices:

- Backup, or take a snapshot of your current working MetaDefender Core server.
  - This is to help quickly rollback for any upgrade failure might cause later for any reason.
- Optionally backup your PostgreSQL database, especially when using remote shared database mode.
  - Reference: https://www.postgresql.org/docs/current/backup.html
  - Please do it after stopping all linked MetaDefender Core instances to avoid potential data collision.
- Lastly, if backing up entire server is not feasible, then at least export MetaDefender Core configuration package Import/Export configuration just in case you'll need to start off a new server with current MetaDefender Core configuration to keep the scanning service running without any impact.

On the PostgreSQL server, double check server resource to ensure sufficient RAM memory and disk space to be preserved for the upgrade process.

- 8GB RAM available at least during the upgrade.

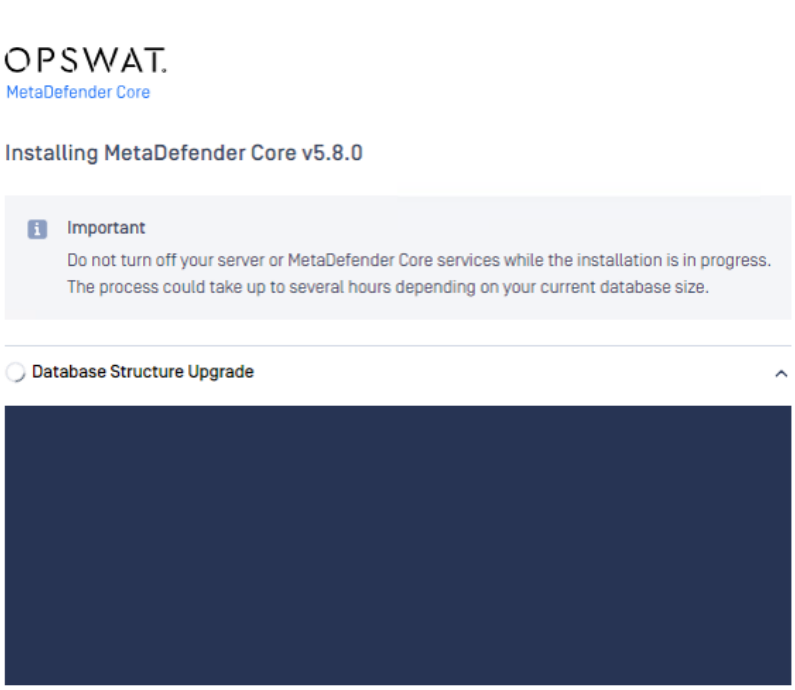
- Disk availability is greater than MetaDefender Core database size.

While doing the upgrade

Applicable for shared database deployment:

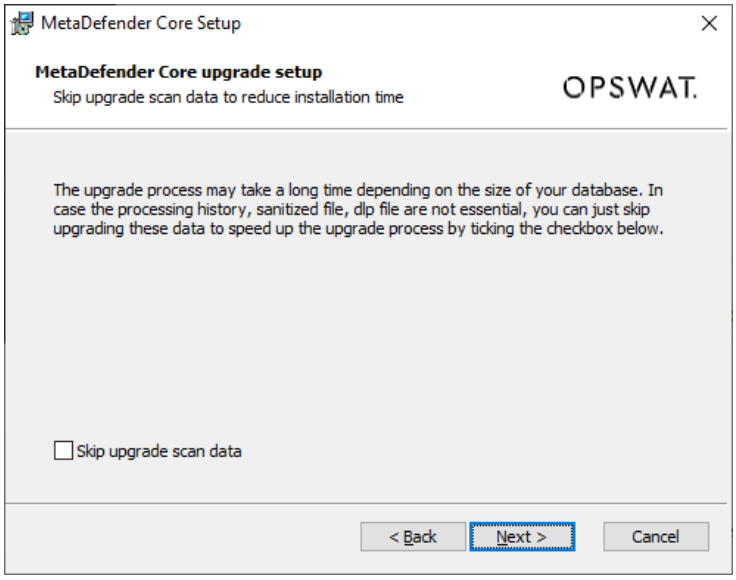
- Stop all MetaDefender Core instances' scanning service before performing the upgrade.
- Upgrade one (1) MetaDefender Core instance at a time.
- Proceed to upgrade MetaDefender Core instance where the bundle local PostgreSQL server is used for shared database setup.
- Otherwise is using remote PostgreSQL then the upgrade order does not matter.

For large size database, mostly for scan history, expecting long time to be spent on the upgrade process.



The product supports an option to skip data migration for scan history to expedite the process drastically and mitigate any failure risk. You can manually backup all data from your PostgreSQL server for offline usage if any.

Check "Skip upgrade scan data" to enable this feature while doing the upgrade.



or add following key in the ignition file:

none

```
[internal]
skip_upgrade_scan_data = 0
```

Troubleshooting

If the upgrade process ever runs into an error:

- Double check your network and services, applicable to remote PostgreSQL server.
- Optionally configure MetaDefender Core to skip migrating scan history.
  - Windows
    - Open Windows registry, and navigate to HKLM\Software\OPSWAT\Metascan
    - Add a new key `internal/skip_upgrade_scan_data` with value = 1
  - Linux
    - Open MetaDefender Core configuration file `/etc/ometascan/ometascan.conf`
    - Add new key into the file
      - [internal]
      - skip_upgrade_scan_data=1
  - Note: This key will be auto removed upon the upgrade finishes.
  - Then restart MetaDefender Core service to retry the upgrade.
    - Might retry this a few times until the upgrade is done.

Just in case the upgrade process still fails, please contact OPSWAT support with the support package - How to Create Support Package? This will help us analyze the root cause and figure out an optimal solution for your system failure.

Using external Load Balancer

Application Load Balancing (Layer 7)

This page provides information about recommended ways to use sessions with a Layer 7 load-balancer.

Most load-balancers have the ability to provide cookies for pinning subsequent traffic from a client to the appropriate server. This method is called sticky session, session persistence or session affinity. Using cookies efficiently requires the client to know when a cookie should be sent, should not be sent or should be deleted.

Sticky session load balancing

Each Metadefender Core instance has its own databases and application sessions that cannot be seen by other instances. Therefore, in order to get the related data/response to our queries we should ask the appropriate Metadefender Core v5 server. On the other hand, to keep the advantages of the used load balancing method, cookies should not be sent if it is not necessary.

Single file scanning

Step	Stage	Task	Cookie usage
1.	Sending file	Initiate processing a file on the client side. Send file through the load-balancer. (See POST File Analysis API)	Cookie should not be sent
2.	Sending file	Save the cookie and the data_id you got from the load-balancer	Save cookie
3.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Send cookie saved in step 2
4.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Send cookie saved in step 2

Batch scanning

Step	Stage	Task	Cookie usage
1.	Open batch	Initiate processing file(s) in batch. Request a batch ID through the load-balancer. (See POST Initiate Batch API)	Cookie should not be sent
2.	Open batch	Save the cookie and the batch_id you got from a Core server through the load-balancer.	Save cookie
3.	Sending files	Send file through the load-balancer. (See POST File Analysis (using batch header) API)	Send cookie saved in step 2
4.	Sending files	Save the data_id you got from the load-balancer.	-
	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Send cookie saved in step 2
5.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
6.	Getting batch status	See GET Batch Status API	Send cookie saved in step 2
7.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Send cookie saved in step 2
8.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API	Send cookie saved in step 2

If it does not matter which upstream server responds, the queries should be sent without cookie.

> **Info**
> It is recommended not to send cookies when it's not necessary to allow load-balancer to use its own method to share the load between Metadefender Core v5 servers.

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration through load-balancer

OPSWAT products that support HTTP load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	4.3.4	-
OPSWAT Client	Windows: 7.6.247.0 Mac: 10.4.243.0	-
MetaDefender Email Security	4.3.0	-
MetaDefender ICAP Server	4.3.0	-
MetaDefender Vault	1.3.0	-
MetaDefender for Secure Storage	2.0.7	-

DNS Load Balancing

Using this method is logically similar to Layer 7 load-balancing.

Briefly how it works

Client uses a domain name to send a query to a server. Client's DNS server has more "A" records for that name with different IPs. When a client resolves the server's domain name DNS server randomly chooses an IP for that name to send back. When a session is used on the application layer, client should know the IP address of the Core server that handles that specific session. In every other case, client should resolve the domain name with DNS query to let requests to be balanced between Metadefender Core servers.

Single file scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Sending file	Initiate processing a file on the client side. (See POST File Analysis API)	Use IP saved in step 2
4.	Sending file	Save the data_id got from the Core server	-
5.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Use IP saved in step 2
6.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Use IP saved in step 2

Batch scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Open batch	Initiate processing file(s) in batch. Request a batch ID. (See POST Initiate Batch API)	Use IP saved in step 2
4.	Sending files	Send file to the specific Core server with the basch ID saved in step 3. (See POST File Analysis (using batch header) API)	Use IP saved in step 2
5.	Sending files	Save the data_id you got from the Core server.	-
6.	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Use IP saved in step 2
7.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
8.	Getting batch status	See GET Batch Status API	Use IP saved in step 2
9.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Use IP saved in step 2
10.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API).	Use IP saved in step 2

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration via DNS load-balancing

OPSWAT products that support DNS load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	does not support yet	-
MetaDefender Client	does not support yet	-
MetaDefender Email Security	does not support yet	-
MetaDefender ICAP Server	does not support yet	-
MetaDefender Vault	does not support yet	-

SSL connection for PostgreSQL communication

> **Note**
> The SSL connection configuration is possible thanks to the support made by PostgreSQL. For reference: https://www.postgresql.org/docs/14/ssl-tcp.html
>
> MetaDefender Core is tested against this mode on particular version 4.21.1

Step 1: Generate SSL certificate for PostgreSQL server

> **Note**
> Following instructions are for self-sign certificate. For production environment, making sure to use the certificate signed by your trusted CA.

1) Generate a private key

bash

```
openssl genrsa -des3 -out server.key 1024
```

Sample output:

bash

```
Generating RSA private key, 1024 bit long modulus ........++++++
...............++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

Type your desired password / pass phrase to encrypt the private key

2) Remove the pass phrase to automatically start up the PostgreSQL server

bash

```
openssl rsa -in server.key -out server.key
```

> **Note**
> On Linux, make sure to set permission on the server.key file

bash

```
chmod og-rwx server.key
```

3) Create a self-signed certificate

bash

```
openssl req -new -key server.key -days 3650 -out server.crt -x509
```

You will be prompted to enter detailed information which is incorporated into your self-signed certificate request.

none

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:CA
Locality Name (eg, city) [Default City]:San Francisco
Organization Name (eg, company) [Default Company Ltd]:OPSWAT
Unit Name (eg, section) []:OPSWAT
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:dummy_email@opswat.com
```

For self-signed certificate, use the server ceriticate as the trusted root certificate:

bash

```
cp server.crt root.crt
```

Copy **server.key**, **server.crt**, **root.crt** to PostgreSQL data folder. For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data**
- Linux: **/var/lib/ometascan/pg_data/**

Step 2: Configure PostgreSQL server for SSL authentication connection

1) Create a custom config file (e.g. ssl.conf) for PostgreSQL.

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\postgres\ssl.conf**
- Linux: **/var/lib/ometascan/postgres/ssl.conf**

2) Content of ssl.conf:

none

```
ssl = on
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
```

3) Modify pg_hba.conf

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data\pg_hba.conf**
- Linux: **/var/lib/ometascan/pg_data/pg_hba.conf**

Add following (modify **::1/128** if needed to match with your environment requirement)

none

```
hostssl all all      ::1/128      scram-sha-256
```

After the modify, the content of pg_hba.conf should be like this (example):

> **Note**
> In a Linux environment, a local connection is required to upgrade from version 5.5.1 or earlier to version 5.6.0 or later.

none

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local         DATABASE  USER  METHOD  [OPTIONS]
# host          DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl     DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostgssenc    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnogssenc  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
local    all all                    scram-sha-256
hostssl all all      ::1/128      scram-sha-256

#More content.......
```

4) Reload the services

- Standalone DB mode: Restart **ometascan** service (MetaDefender Core service)
- Shared DB mode: Restart **ometascan-pg** / **ometascan-postgresql** service (MetaDefender Core PostgreSQL service) and then **ometascan** service (MetaDefender Core service)

Test your SSL database connection:

bash

```
psql.exe "sslmode=require dbname=postgres user=postgres host=localhost port=5432"
Password for user postgres:
psql (12.7)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
Type "help" for help.
```

This message indicate that you are now using SSL connection:

bash

```
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
```

Performance and Load Estimation

> **Disclaimer**
> These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Some factors that affect performance

- MetaDefender Core version
- MetaDefender Core engine package and configuration
  - set of engines (which and how many)
  - product configuration (e.g., thread pool size)
- System environment
  - server profile (CPU, RAM, hard disk)
  - client application location - remote or local
  - system caching and engine level caching
- Dataset
  - encrypted or decrypted
  - file types
    - different file types (e.g., document, image, executable)
    - archive file or compound document format files
  - file size
  - bad or unknown (assume to be clean)
- Performance tool

How test results are calculated

Performance (mainly scanning speed) is measured by throughput rather than unit speed. For example, if it takes 10 seconds to process 1 file, and it also takes 10 seconds to process 10 files, then performance is quantified as 1 second per file, rather than 10 seconds.

- total time / total number of files processed: 10 seconds / 10 files = 1 second / file.

Standalone DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.2.416.0

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

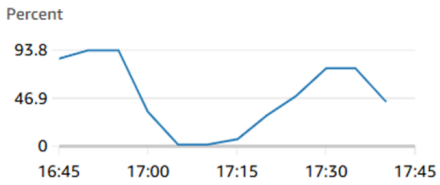
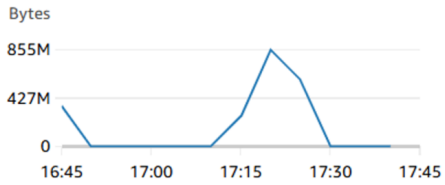
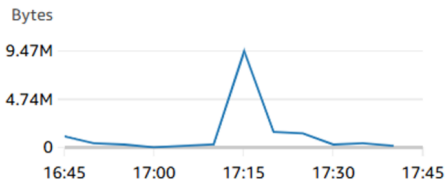
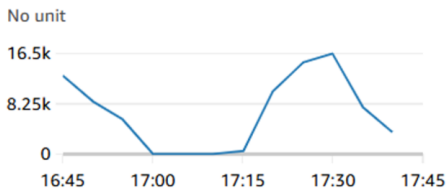
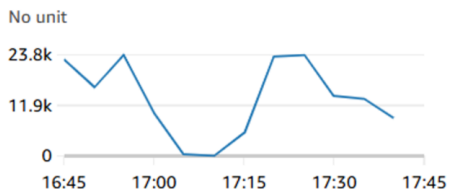
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV)	24.2	47.2 / 80	37.3 / 41.1		315,310	0.0114
Deep CDR	19.9	48.9 / 81.8	32.6 / 37		383,442	0.0094
Proactive DLP	17.1	36.8 / 58.2	30.4 / 33.6		446,228	0.0081
Vulnerability	10.1	59.2 / 85	29.3 / 33.3		755,495	0.0048

MetaDefender Core with common engine packages:

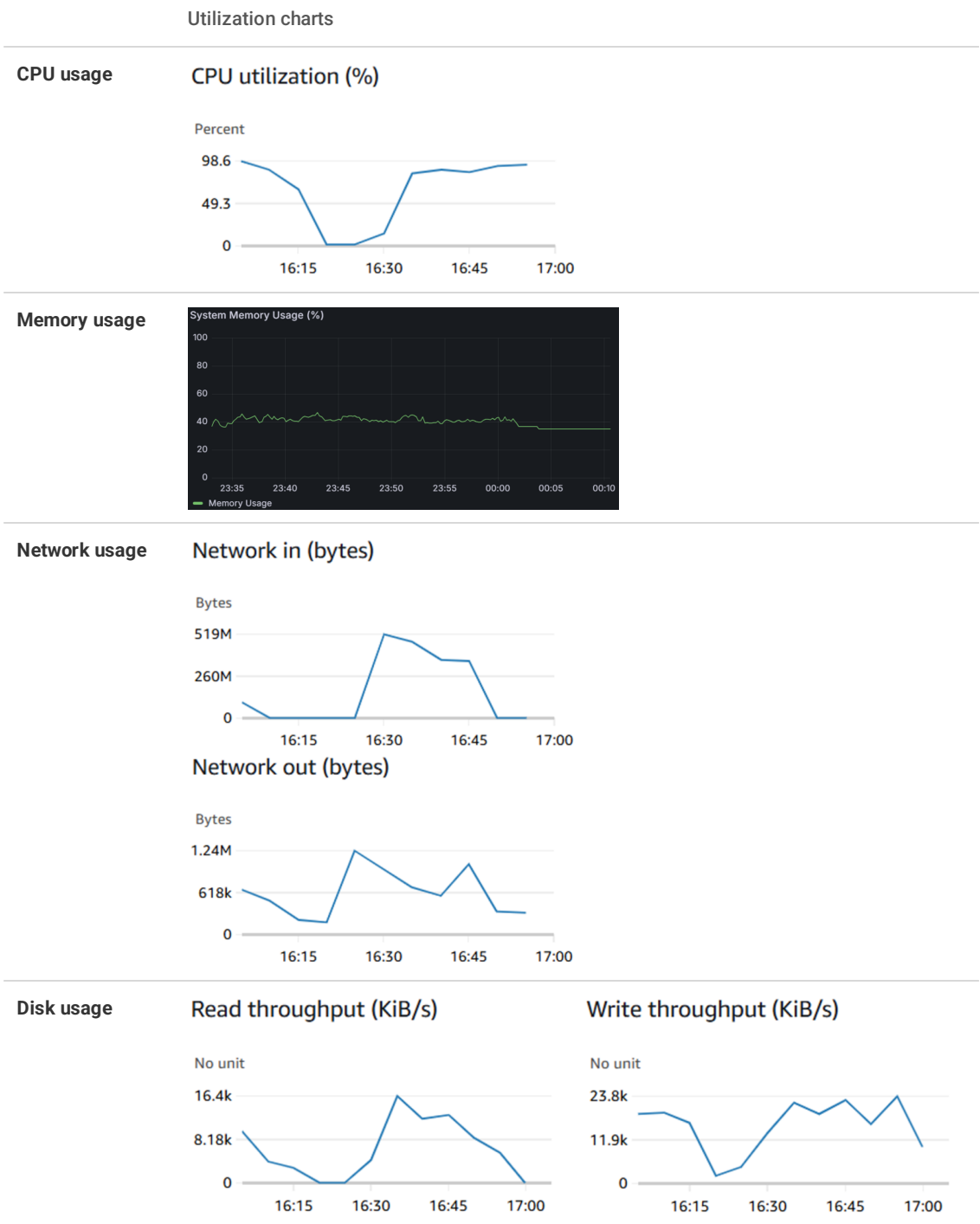
Use case	Scan duration	Avg./Max CPU usage	Avg./Max RAM usage	Avg./Max Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV) + Deep CDR	30.8	68.1 / 88	40.1 / 46.8		247,744	0.0145
Metascan (8 AV) + Deep CDR + Proactive DLP	31.2	74.2 / 98.6	41.5 / 51.1		244,567.3	0.0147
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	31.8	73.5 / 98	40.6 / 51.5		239,953	0.015

System resource utilization charts

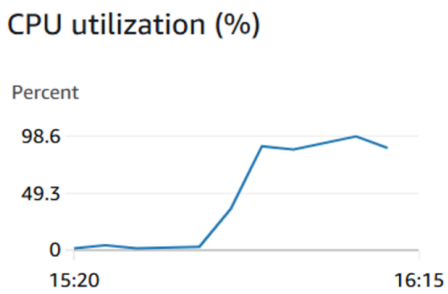
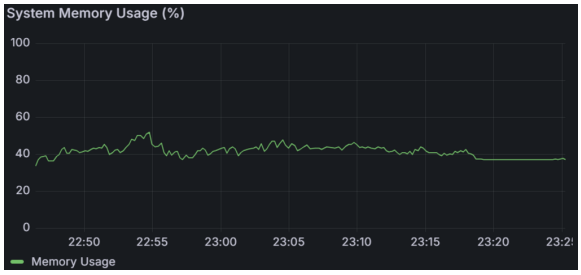
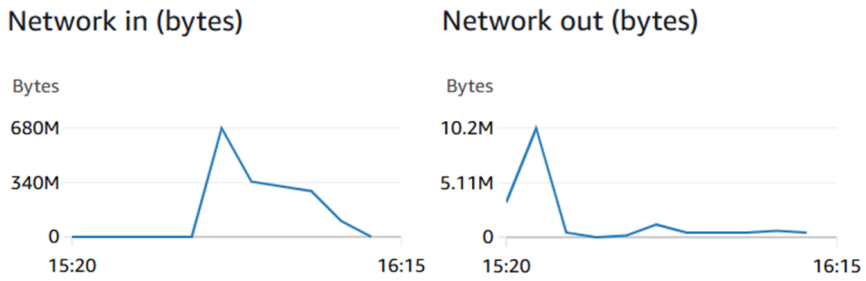
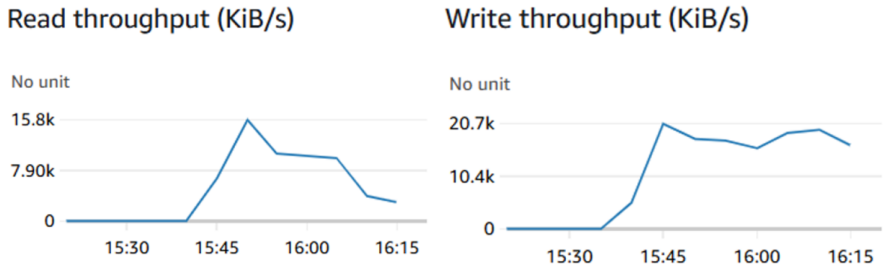
MetaDefender Core with Metascan (8 AV engines):

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

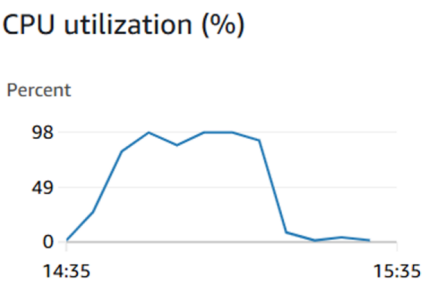
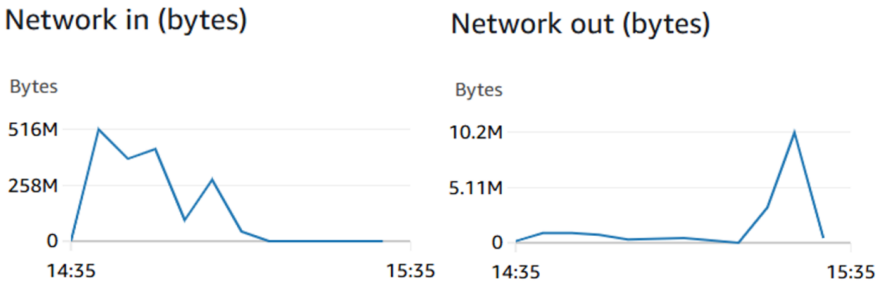
MetaDefender Core with Metascan (8 AV engines) + Deep CDR:



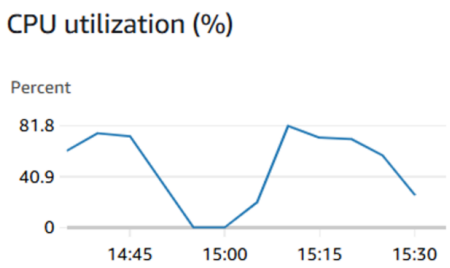
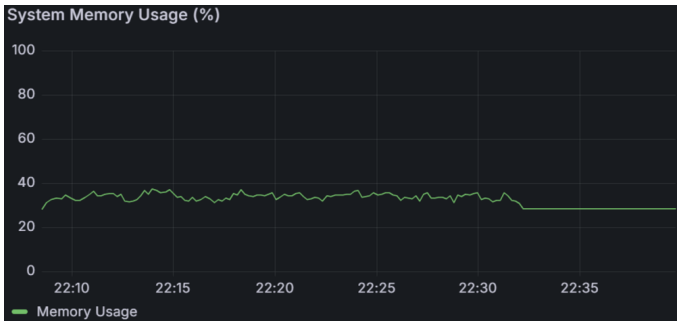
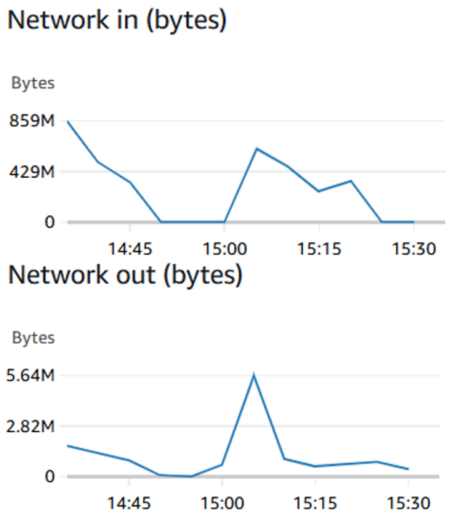
MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

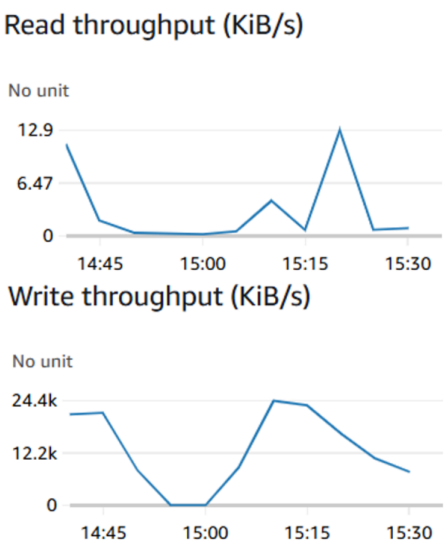
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

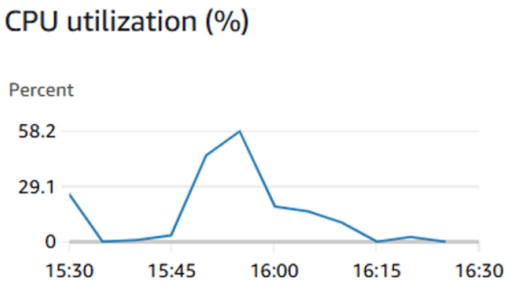
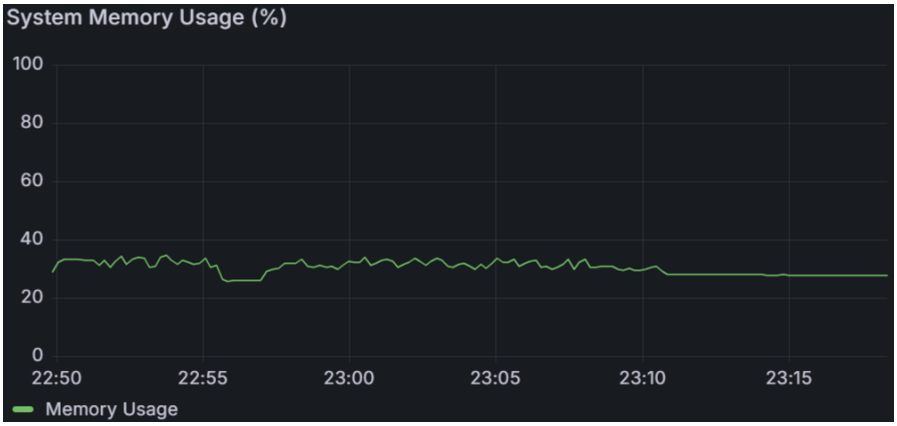
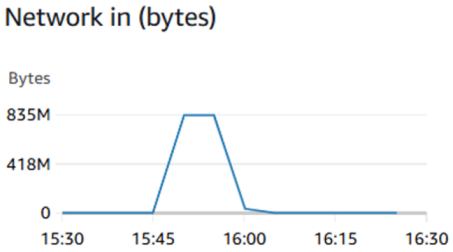
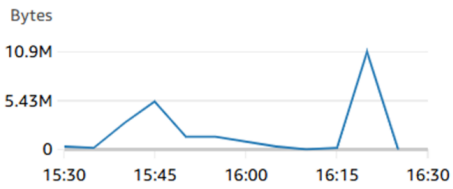
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

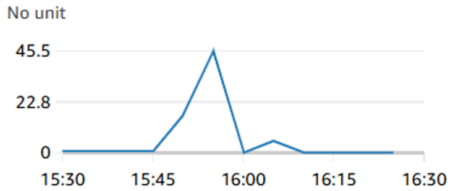
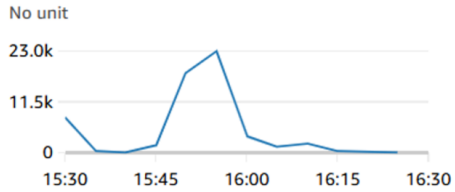
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

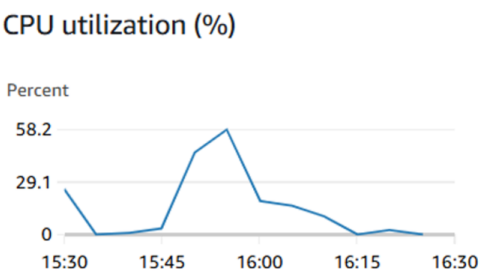
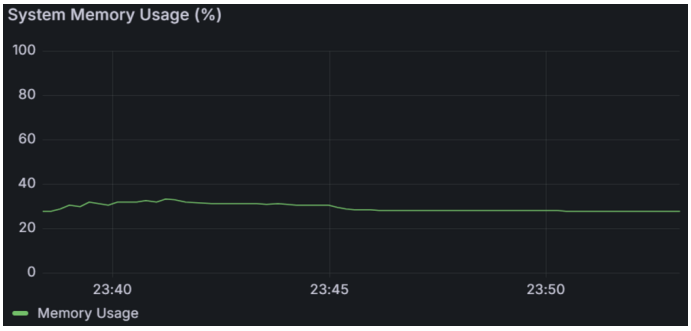
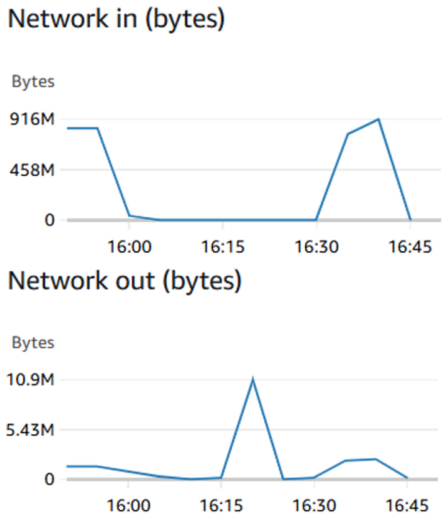
	Utilization charts
Disk usage	

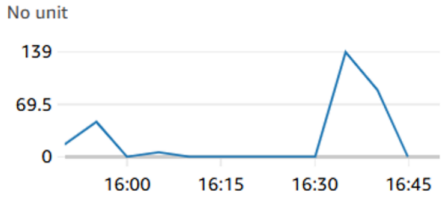
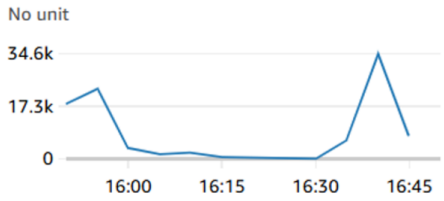
MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	 

	Utilization charts
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

MetaDefender Core with Vulnerability:

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

Standalone DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Ubuntu 22.04	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.57-236

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

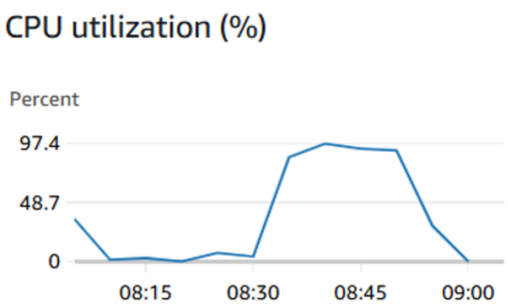
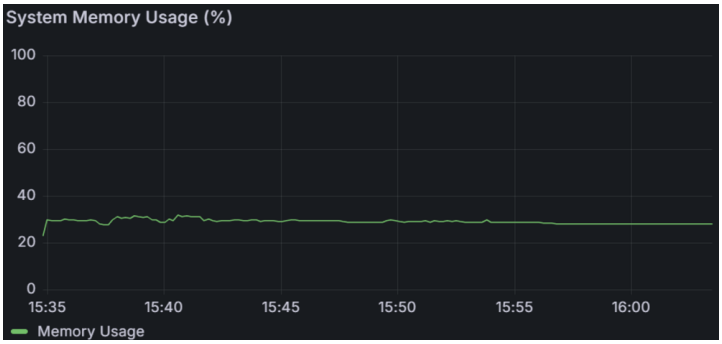
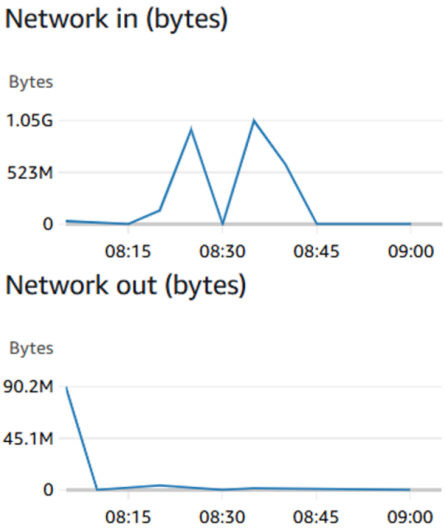
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV)	30.2	69 / 98	29 / 32		252,666	0.0142
Deep CDR	19.4	48 / 62	36 / 49		393,325	0.009
Proactive DLP	16.5	34 / 39	32 / 48		462,455	0.008
Vulnerability	12	50 / 65	46 / 46		635,875	0.006

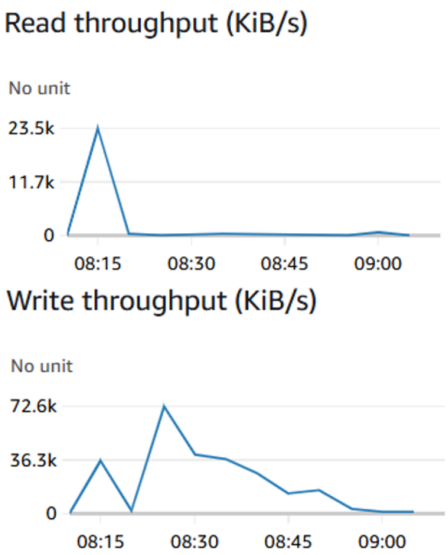
MetaDefender Core with common engine packages:

Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV) + Deep CDR	39.8	78 / 99	62 / 73		191,721	0.0188
Metascan (10 AV) + Deep CDR + Proactive DLP	43.5	61 / 84	47 / 67		175,414	0.0205
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	44.8	62 / 88	69 / 83		170,324	0.0211

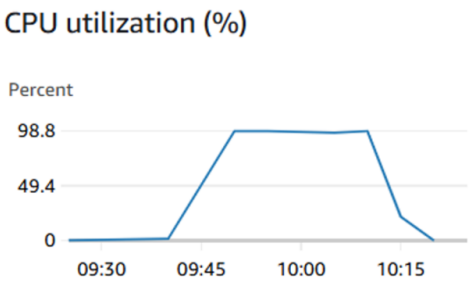
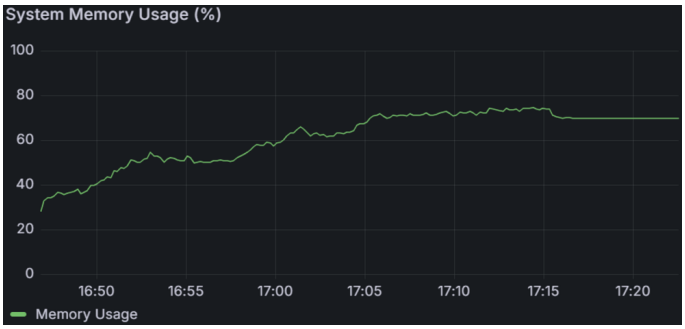
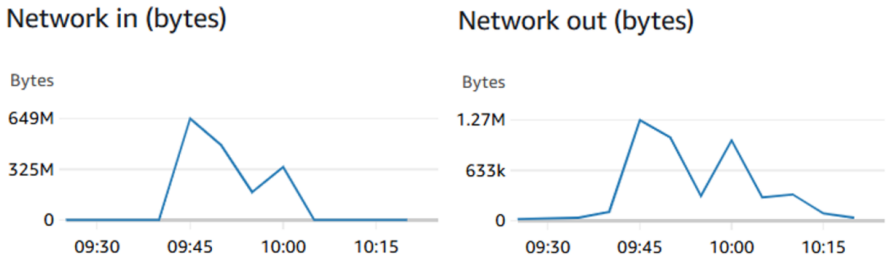
System resource utilization charts

MetaDefender Core with Metascan (10 AV engines) :

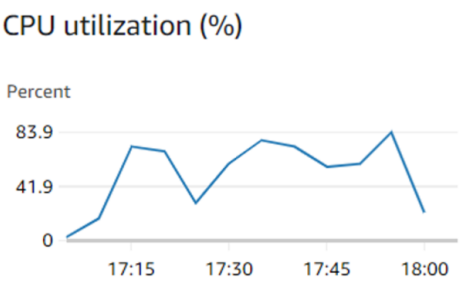
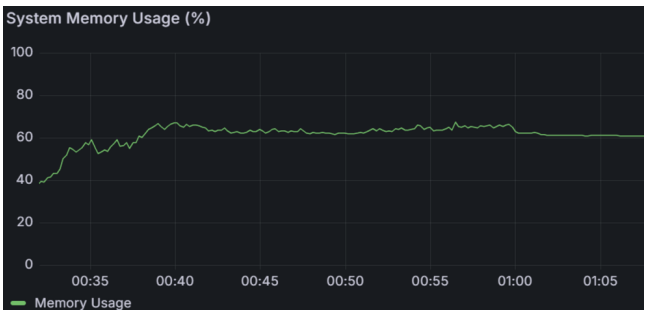
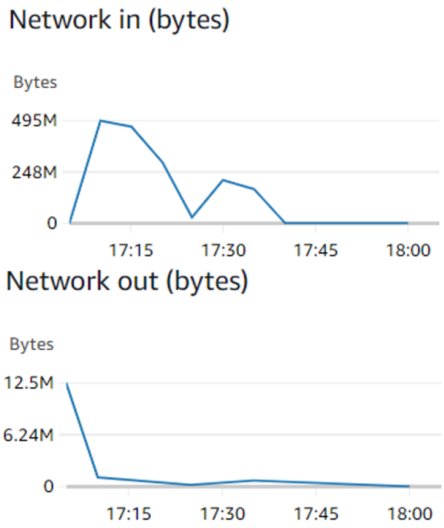
	Utilization charts
CPU usage	
Memory usage	
Network usage	

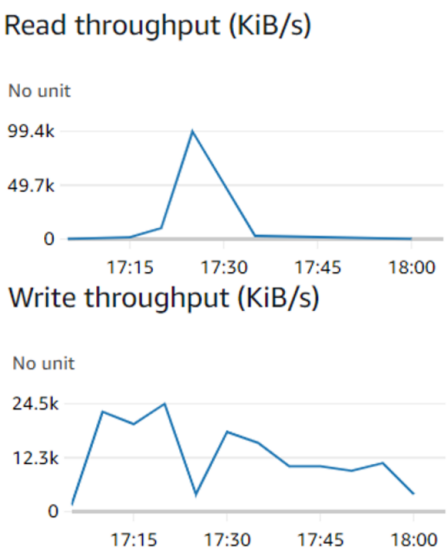
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR :

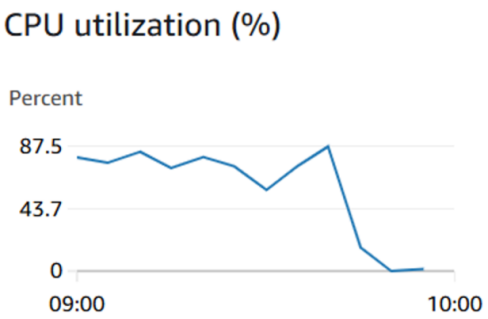
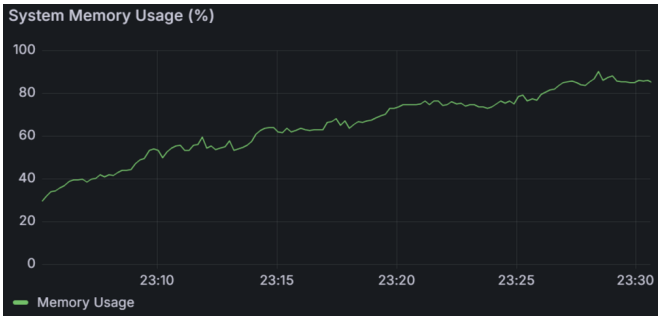
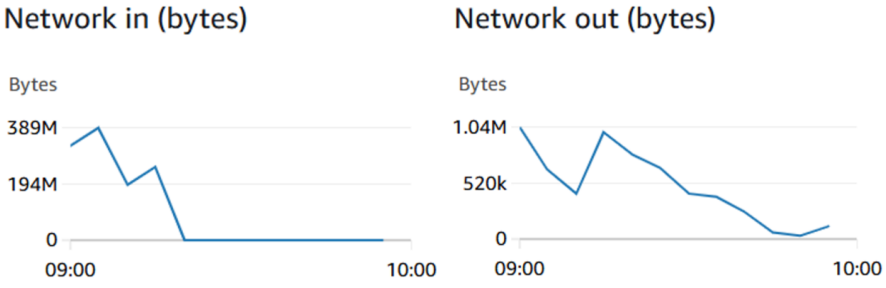
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP :

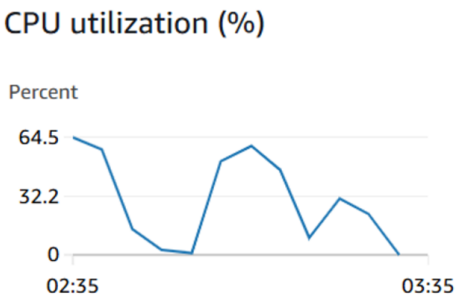
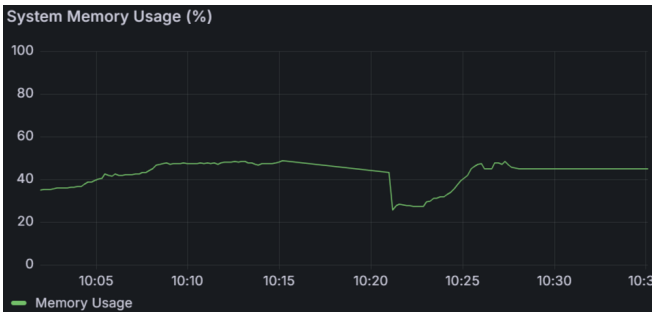
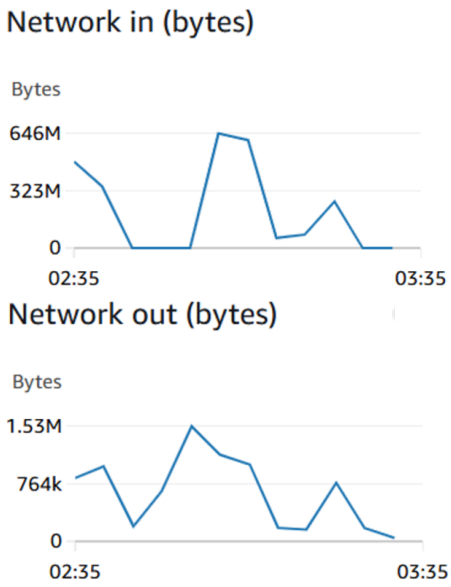
	Utilization charts
CPU usage	
Memory usage	
Network usage	

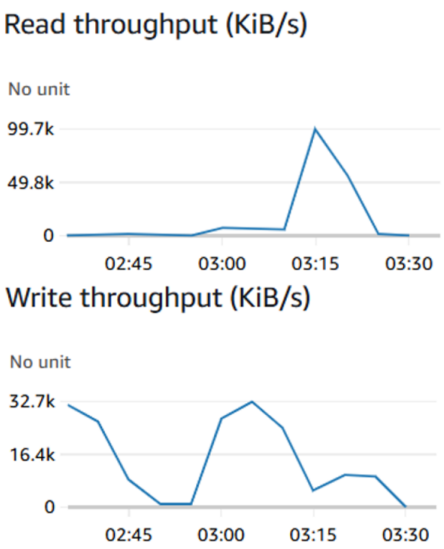
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP + Vulnerability :

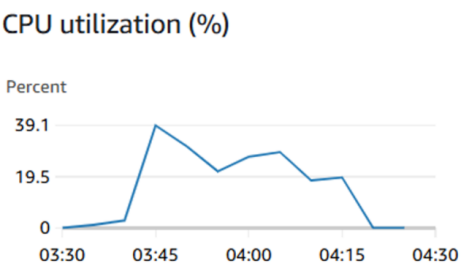
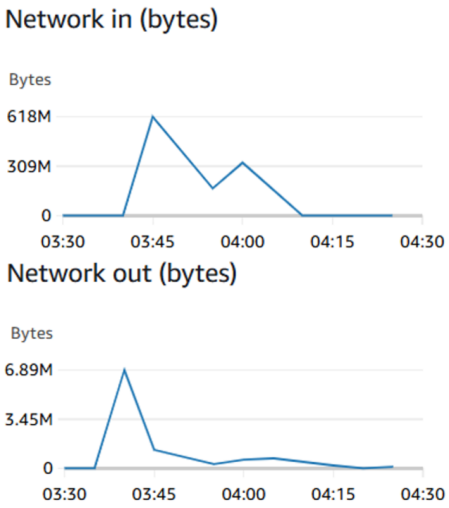
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

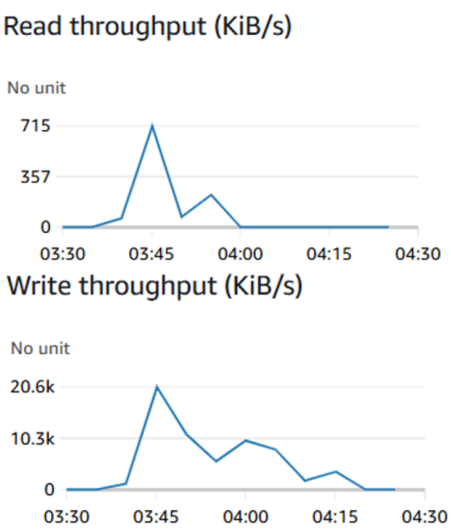
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

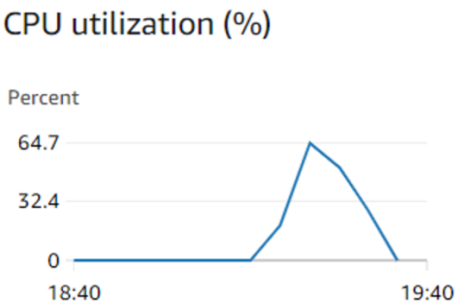
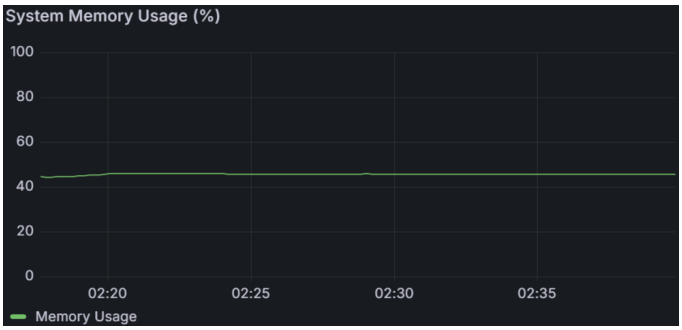
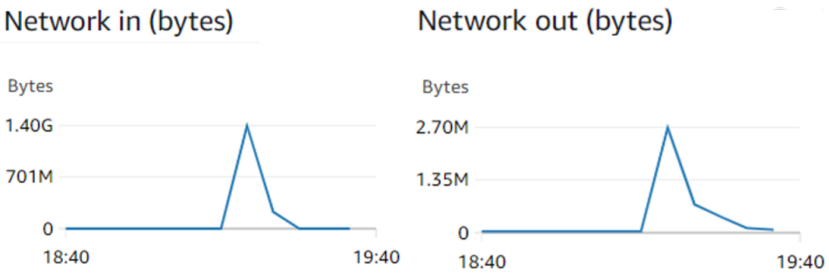
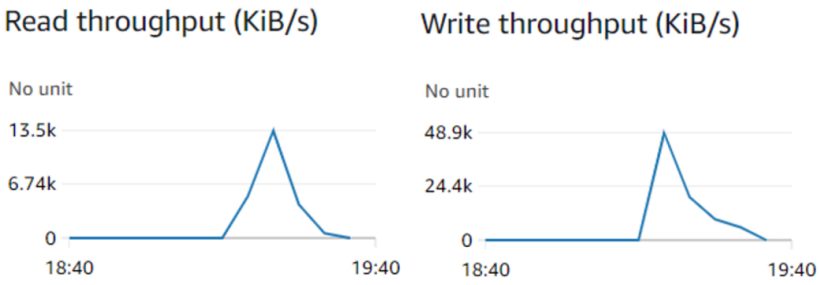
	Utilization charts
CPU usage	
Memory usage	
Network usage	

Utilization charts

Disk usage



MetaDefender Core with Vulnerability :

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

Shared DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

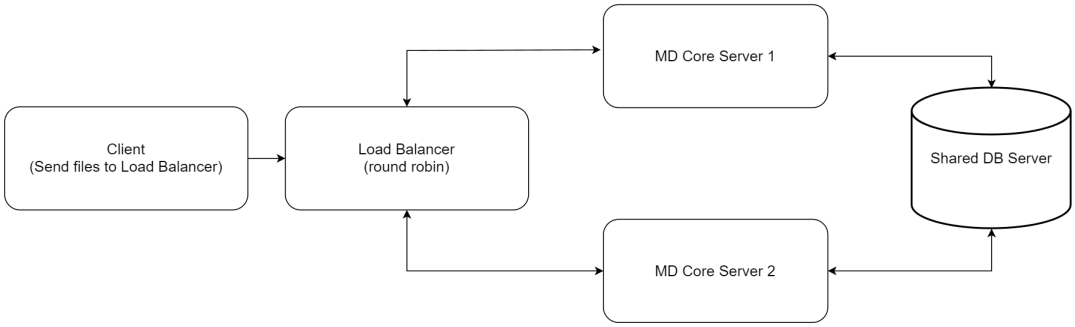
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV)	11.9	37.0	41.9	30.1	30.4	Receive: 6441.1 Send: 769.6	Receive: 6200.2 Send: 693.7	640,821.2	0.0
Deep CDR	10.5	60.1	66.7	30.3	32.2	Receive: 6379.8 Send: 594.8	Receive: 7146.1 Send: 609.6	729,368.5	0.0
Proactive DLP	7.8	38.7	37.5	29.7	31.0	Receive: 8842.6 Send: 791.8	Receive: 8635.6 Send: 729.1	965,455.7	0.0
Vulnerability	4.4	52.2	53.0	26.0	26.9	Receive: 16,215.5 Send: 1006.8	Receive: 15,785.7 Send: 997.2	1714,449.3	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV) + Deep CDR	15.4	72.4	75	34.3	35.4	Receive: 5436.6 Send: 611.3	Receive: 5589.3 Send: 607.2	495,697.4	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP	16.9	77.2	80.2	36.4	37.9	Receive: 5254 Send: 585.4	Receive: 5041.3 Send: 576.1	450,657.6	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	18.0	72.1	80.4	38.6	38.0	Receive: 4504.3 Send: 574.1	Receive: 4649.5 Send: 571.1	423,010	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

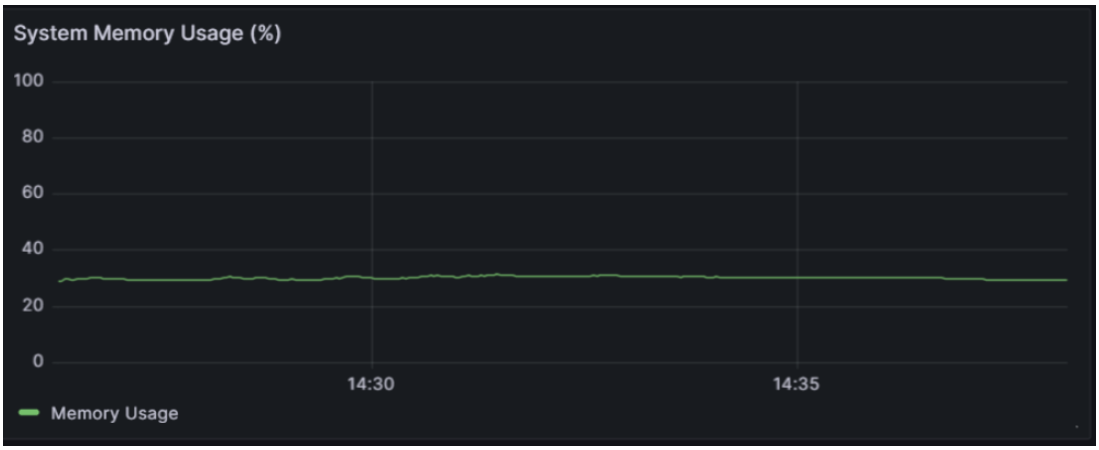
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2

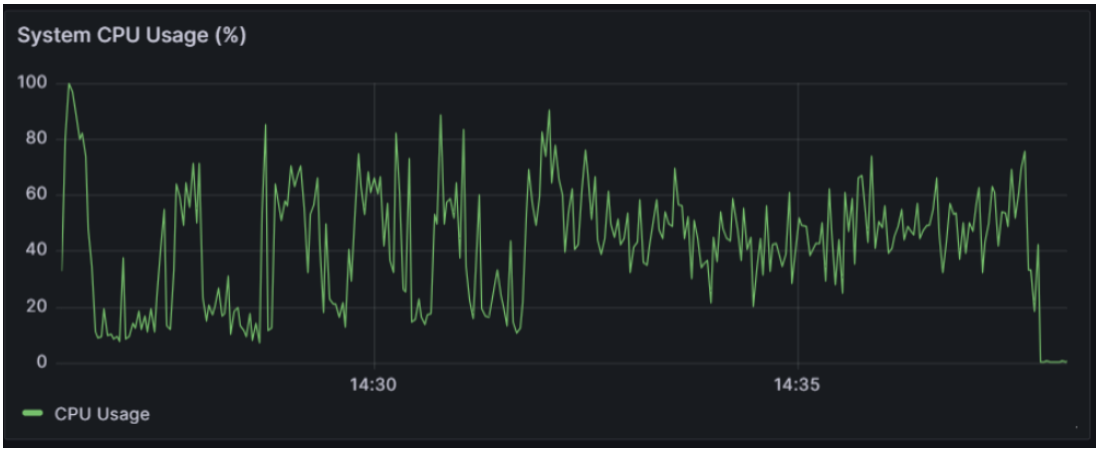


CPU usage

MD Core #1

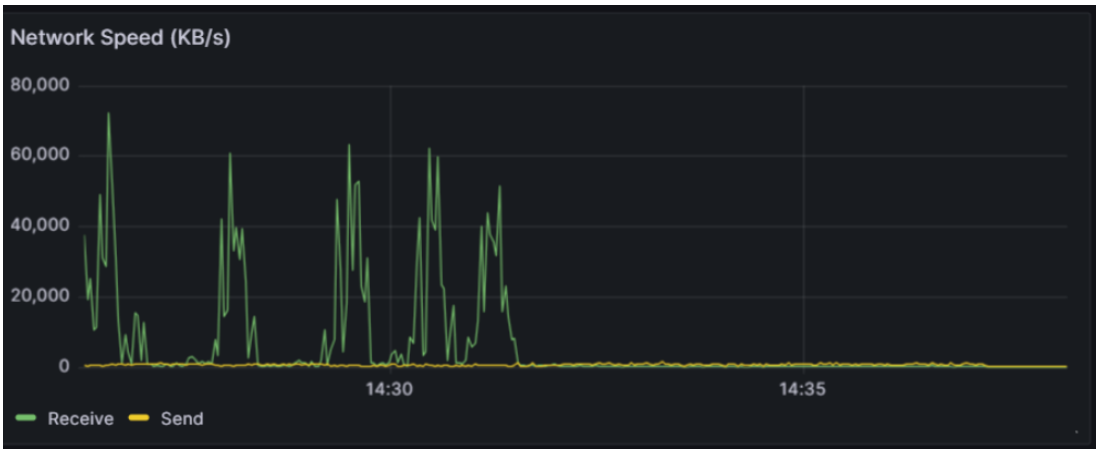


MD Core #2

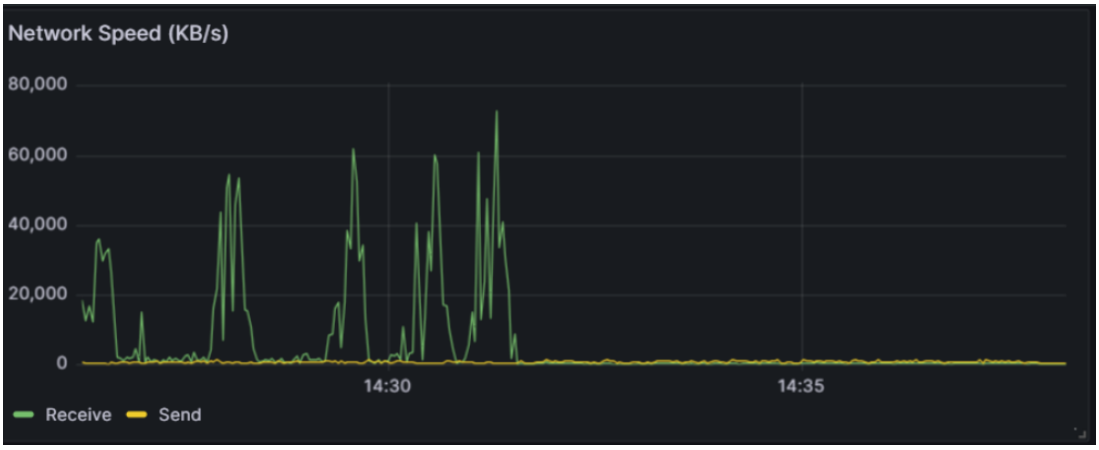


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2

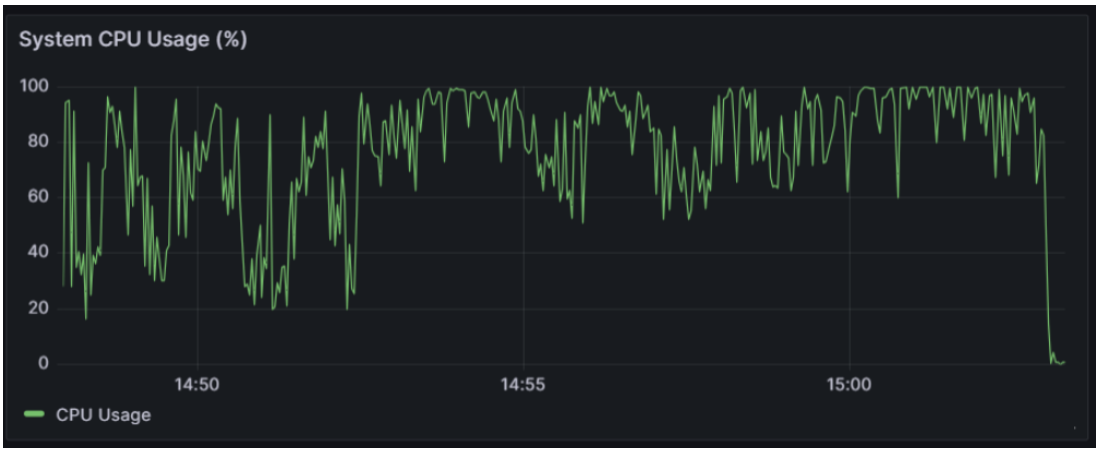


CPU usage

MD Core #1

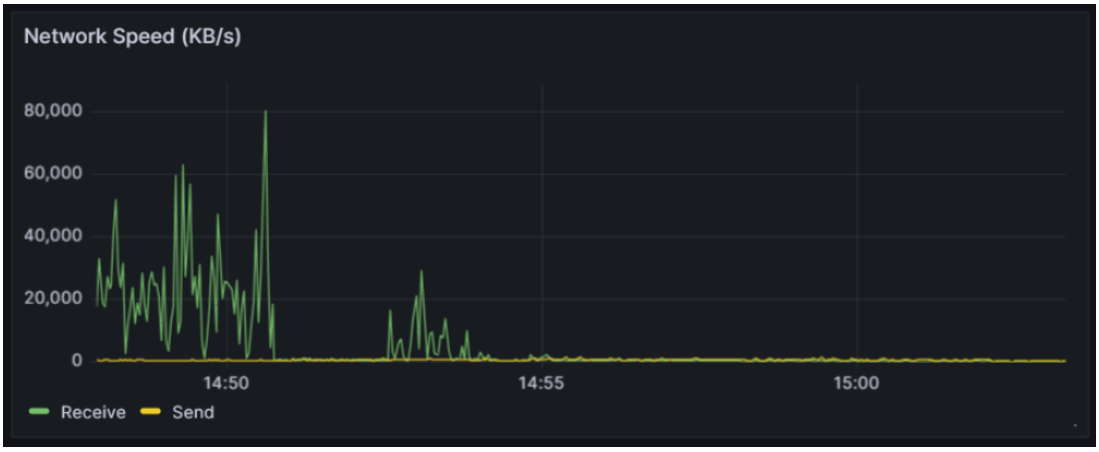


MD Core #2



Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

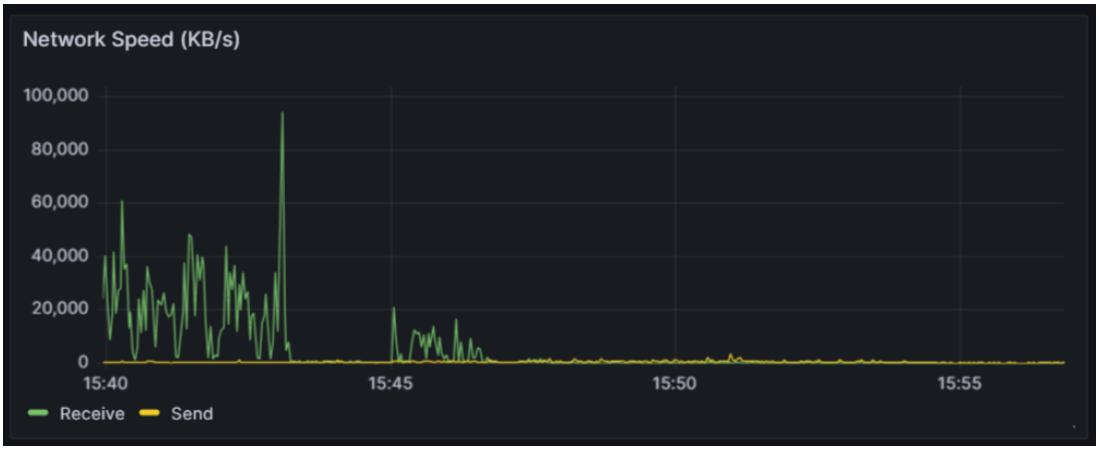


MD Core #2

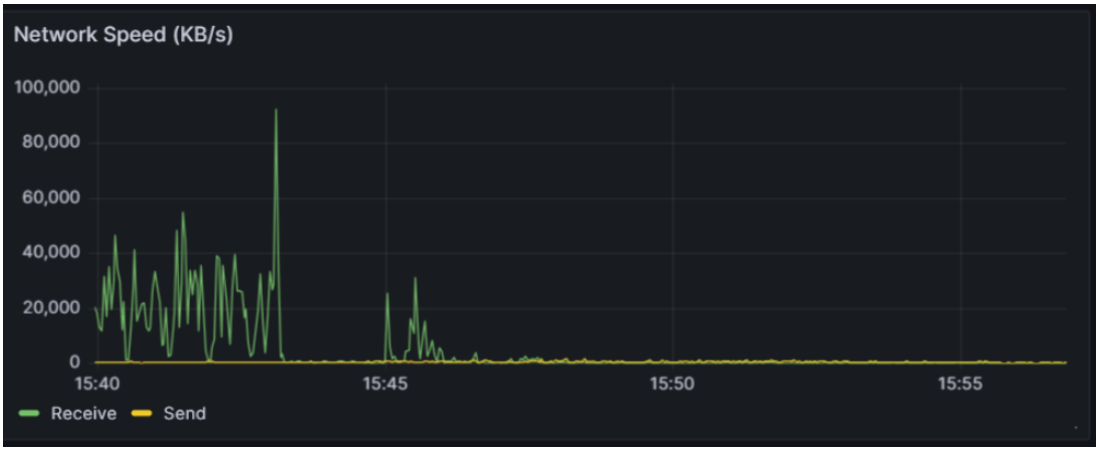


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1

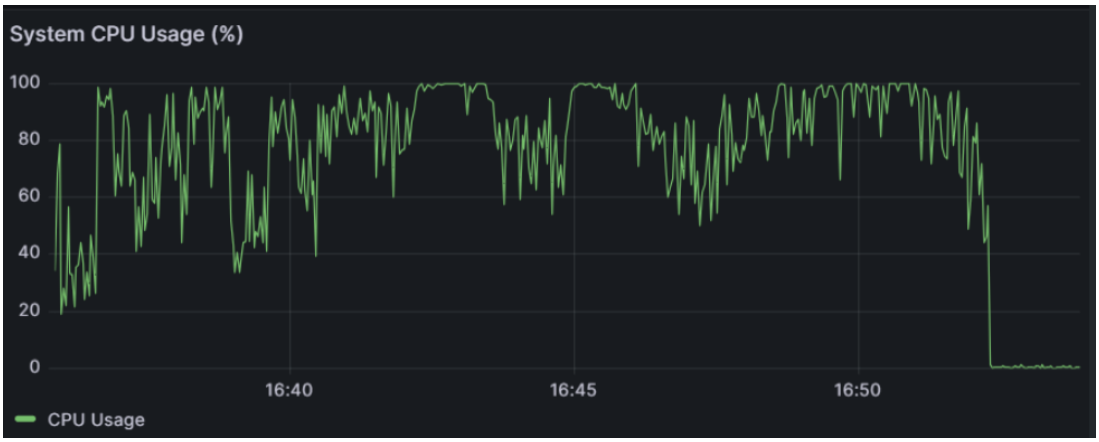


MD Core #2



CPU usage

MD Core #1

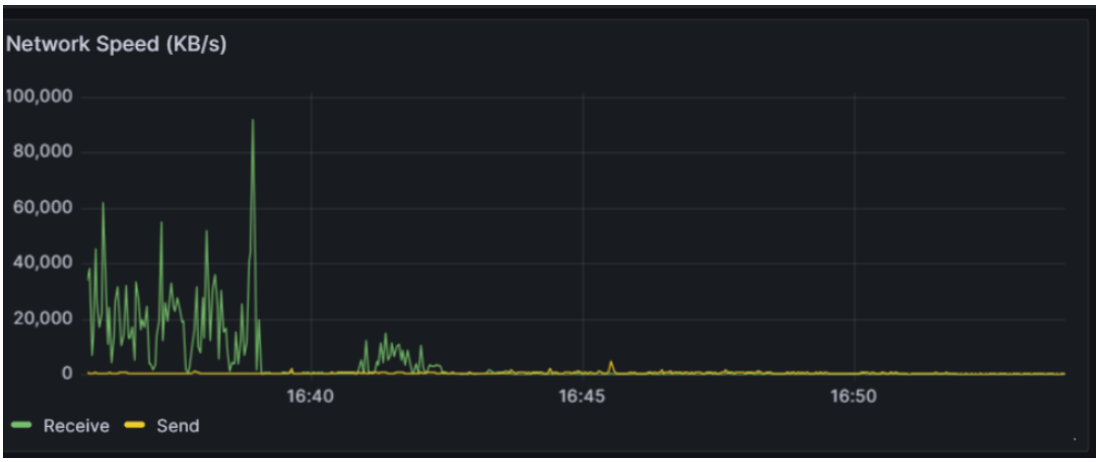


MD Core #2

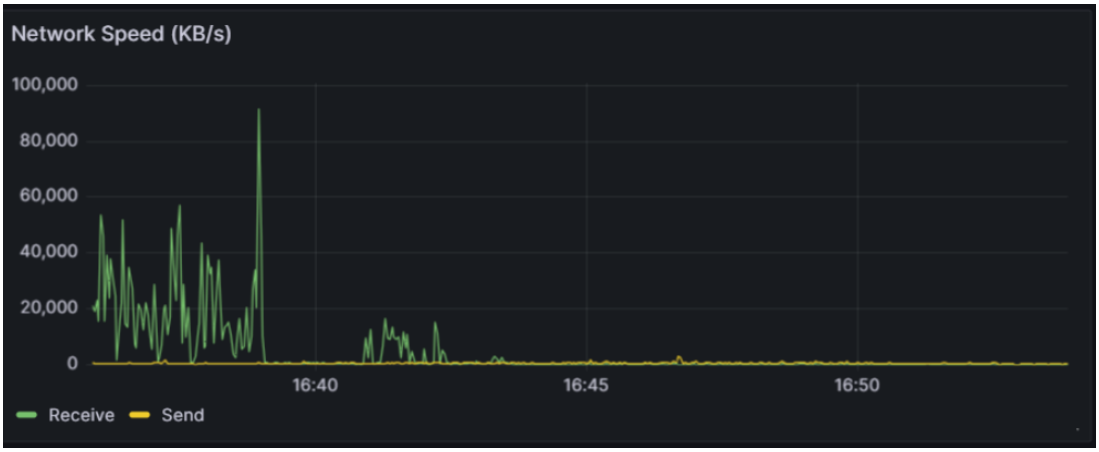


Network usage

MD Core #1



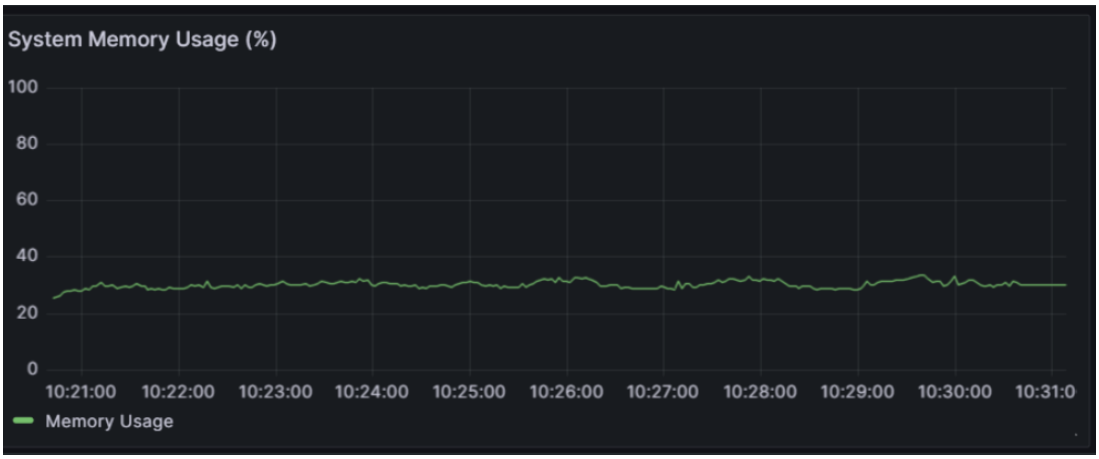
MD Core #2



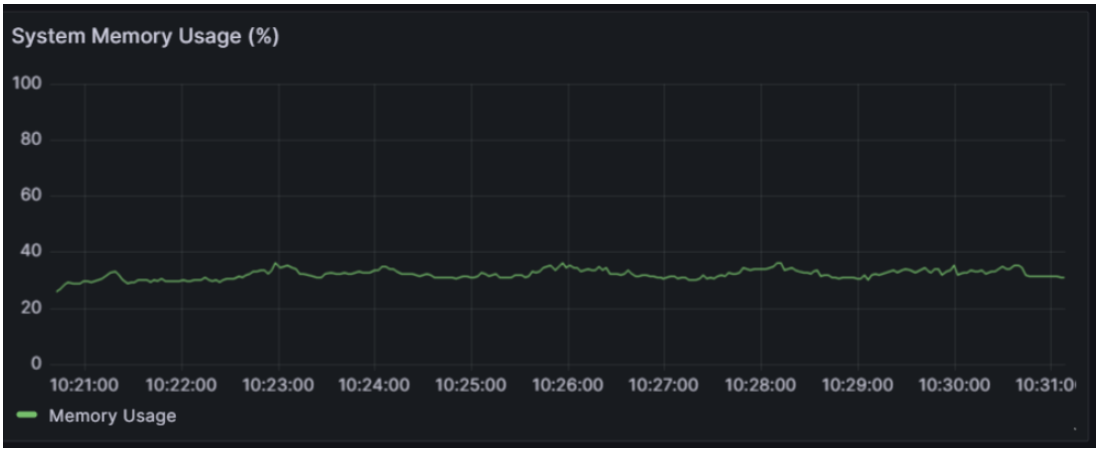
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

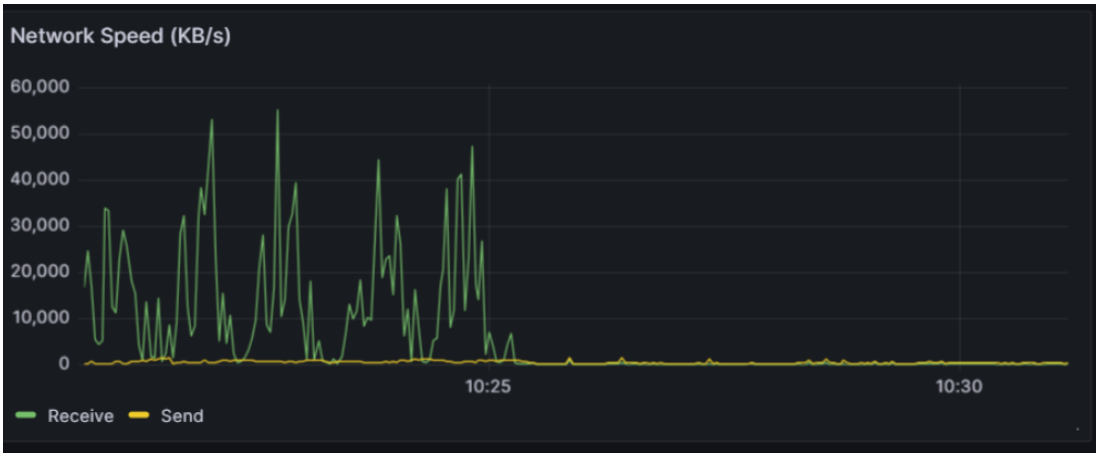


MD Core #2

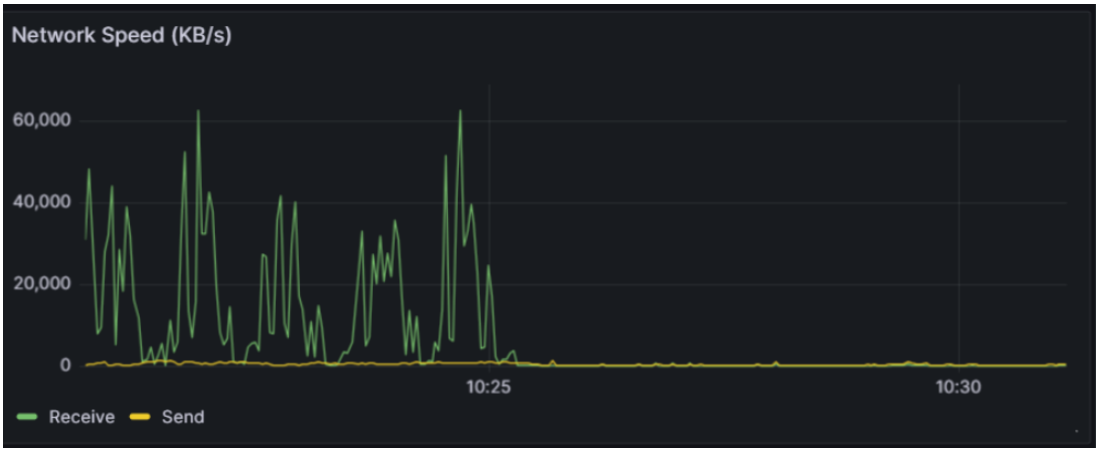


Network usage

MD Core #1



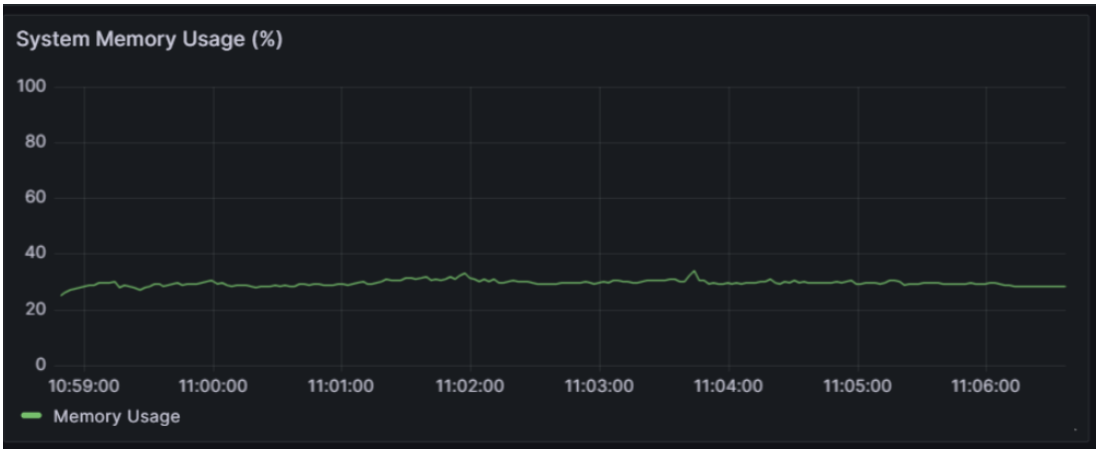
MD Core #2



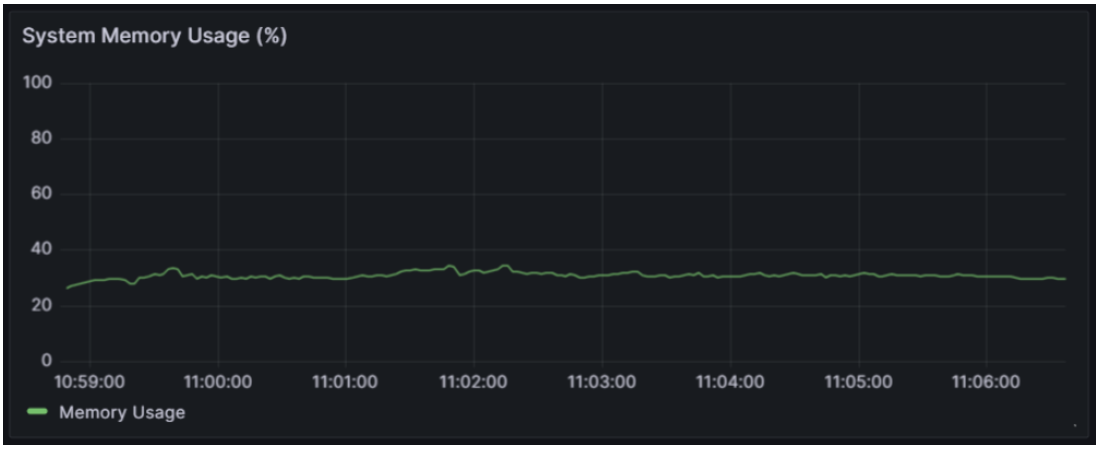
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

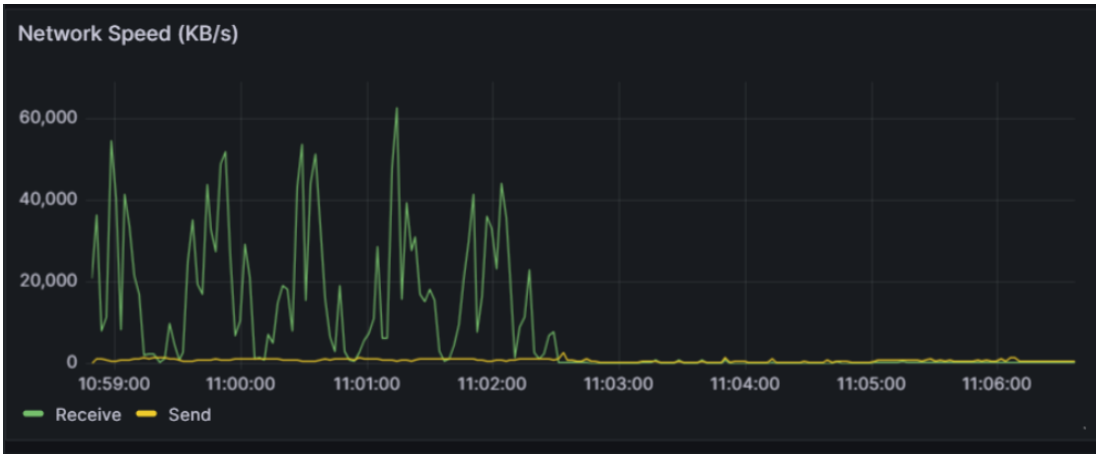


MD Core #2

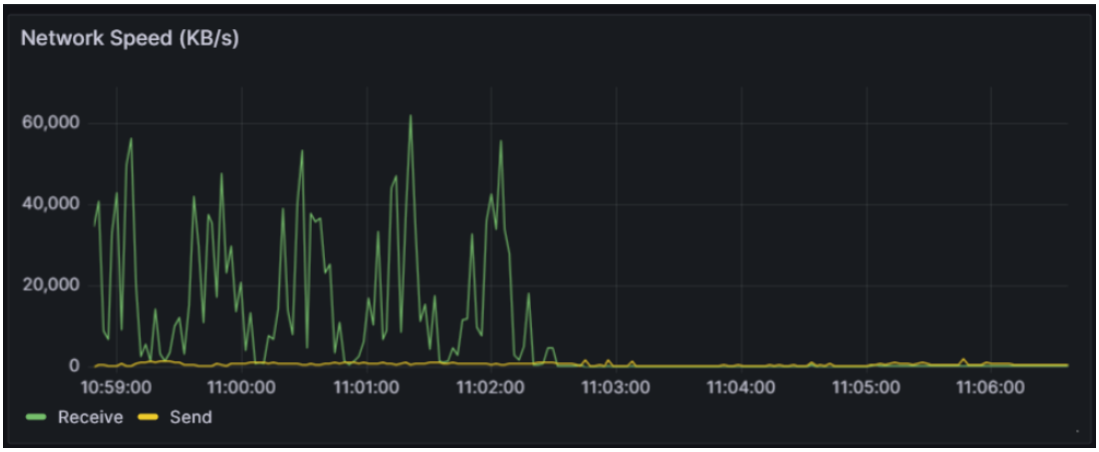


Network usage

MD Core #1



MD Core #2



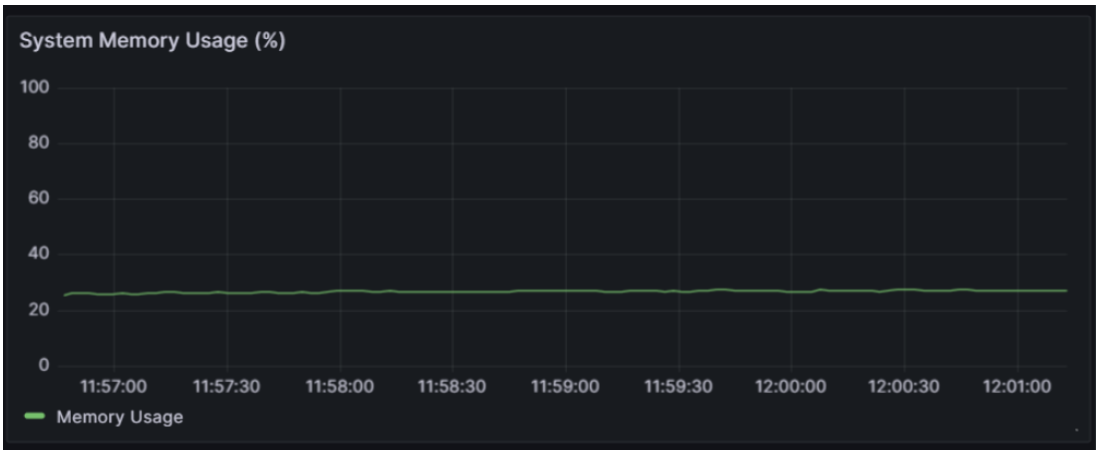
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

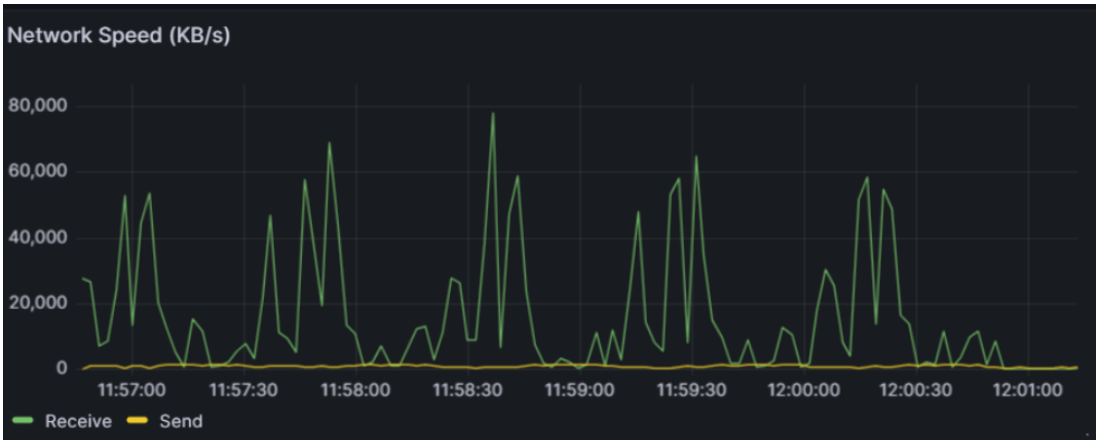


MD Core #2

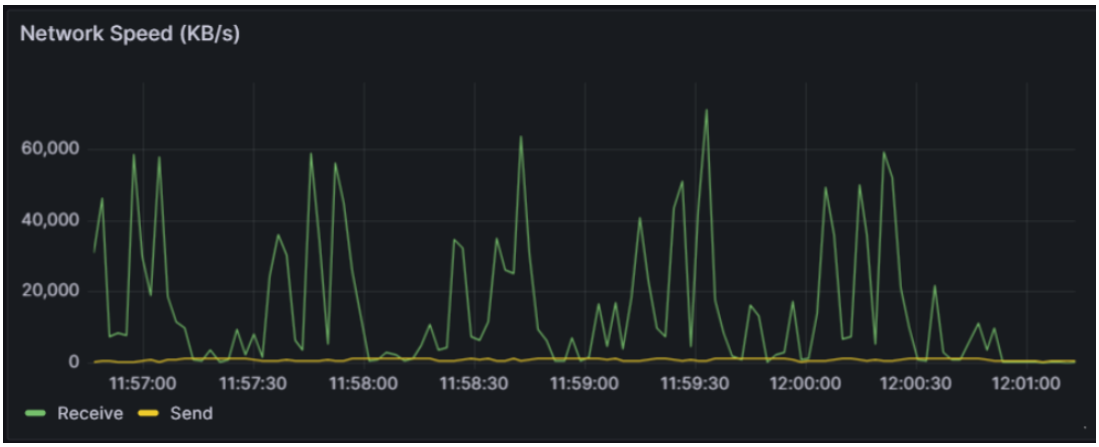


Network usage

MD Core #1



MD Core #2



Shared DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

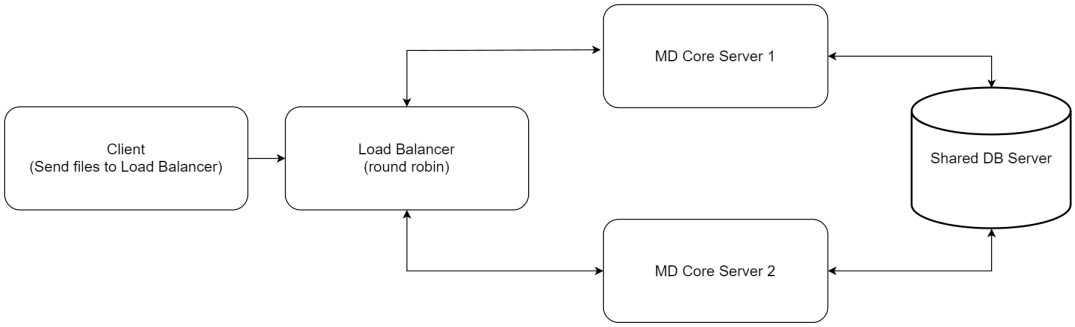
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV)	15.5	63.1	73.3	25.2	25.8	Receive: 4555.9 Send: 694.4	Receive: 4516.2 Send: 692.4	492,620.5	0.0
Deep CDR	9.70	50.7	51.2	31.9	32.5	Receive: 7062.9 Send: 623.2	Receive: 6653 Send: 654.1	786,189.6	0.0
Proactive DLP	7.2	35.2	35.0	27.9	27.5	Receive: 9047.4 Send: 802.4	Receive: 9859.8 Send: 787.5	1,055,110.8	0.0
Vulnerability	5.0	45.7	49.3	24.4	24.5	Receive: 13,585.7 Send: 966.7	Receive: 13,505.8 Send: 987.4	1,519,562.1	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV) + Deep CDR	19.9	79.1	77.6	34.0	34.2	Receive: 3785.9 Send: 573.8	Receive: 3663.1 Send: 577.7	382,716	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP	21.6	80.3	81.5	37.7	44.4	Receive: 3305.6 Send: 543.3	Receive: 3414.3 Send: 556.2	353,694.9	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	22.5	84.2	79.8	37.9	44.7	Receive: 3267.9 Send: 554.9	Receive: 3398.2 Send: 553.8	338,507.6	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

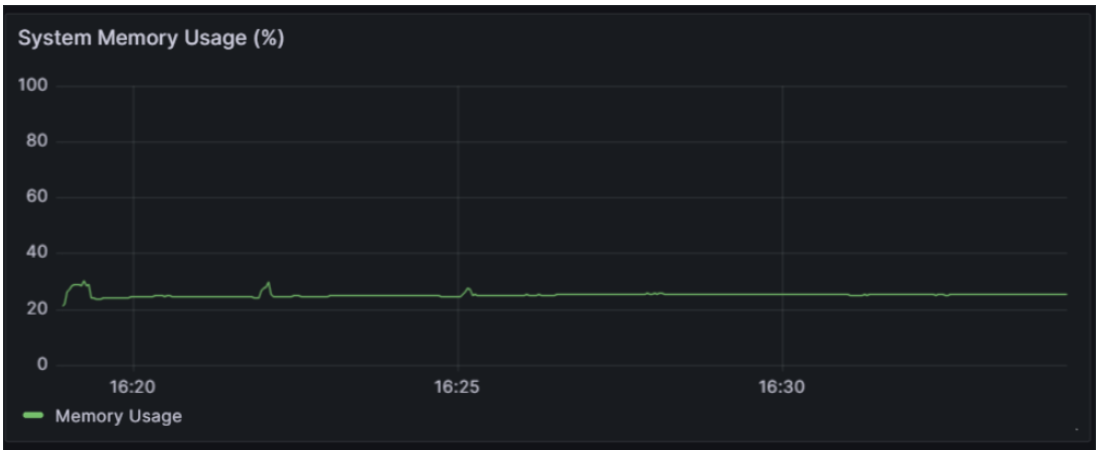
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

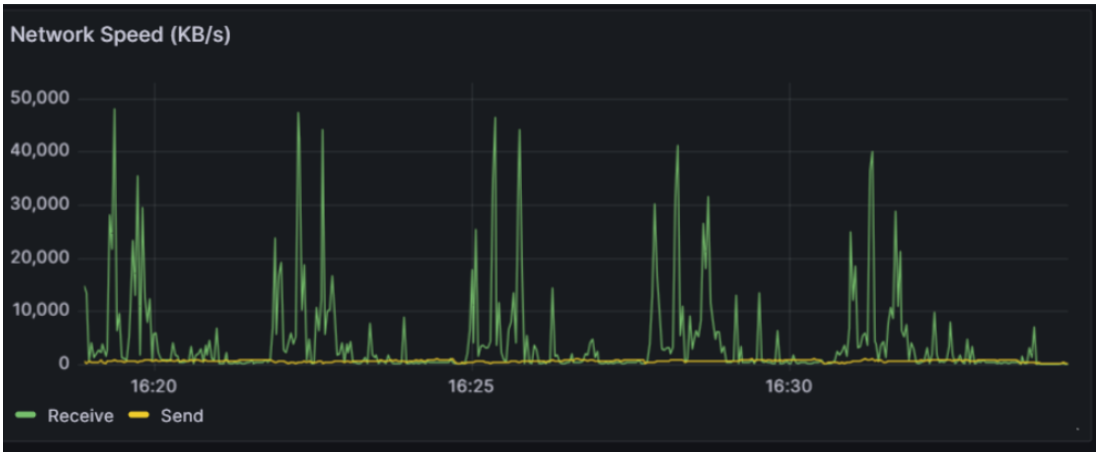


MD Core #2

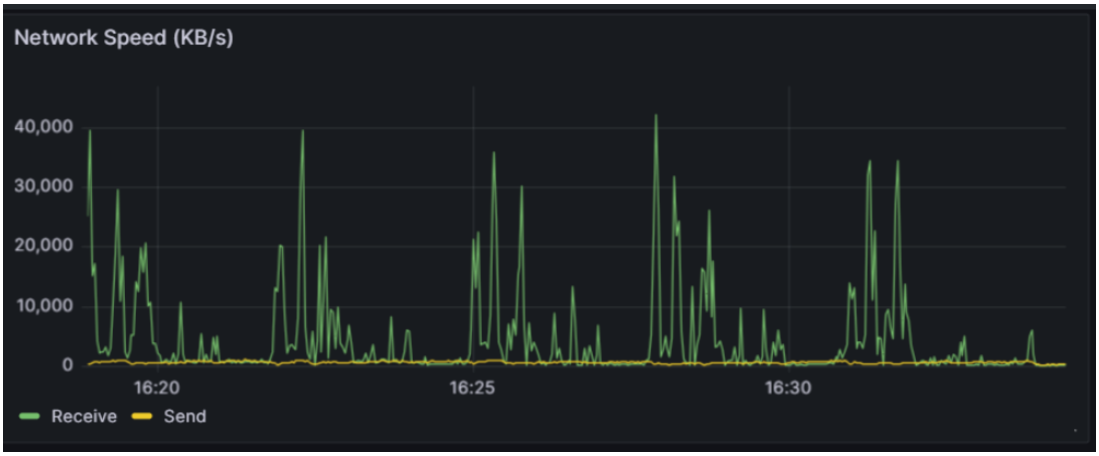


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

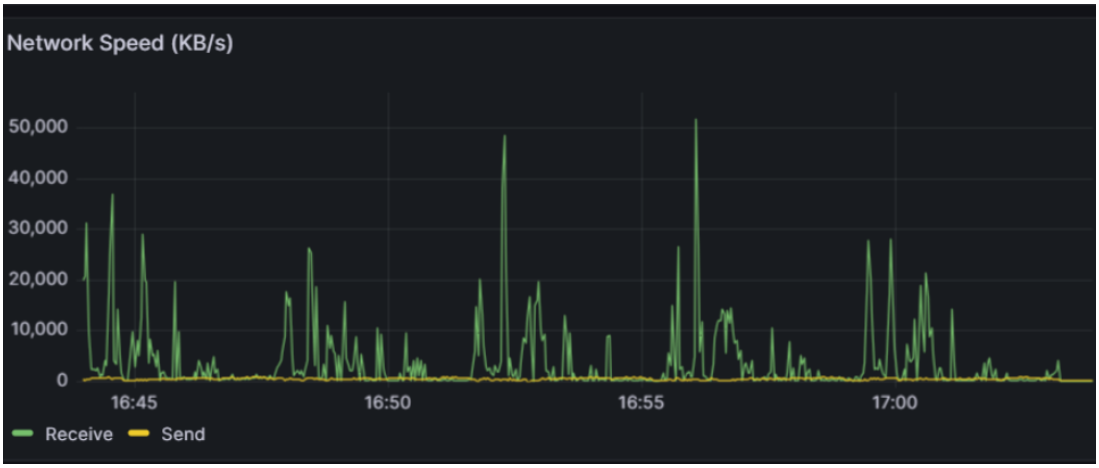


MD Core #2

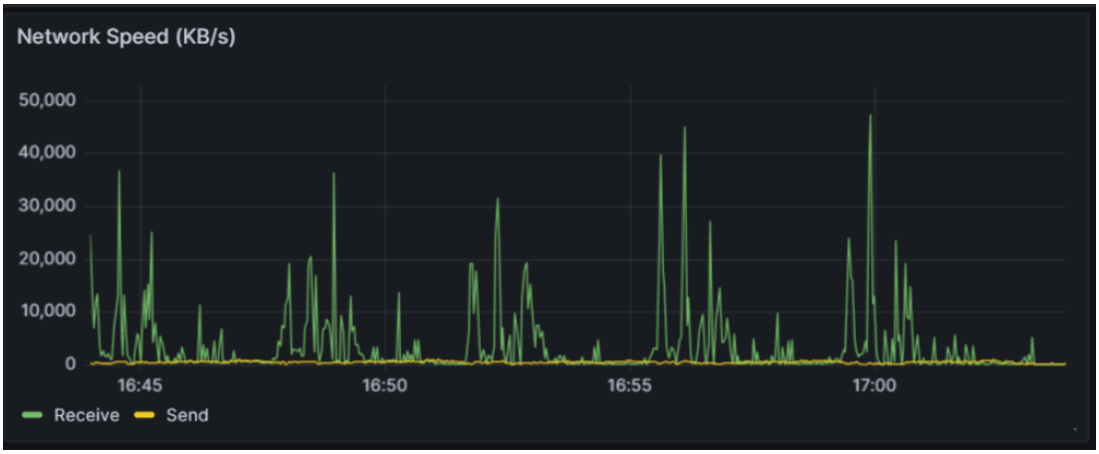


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1

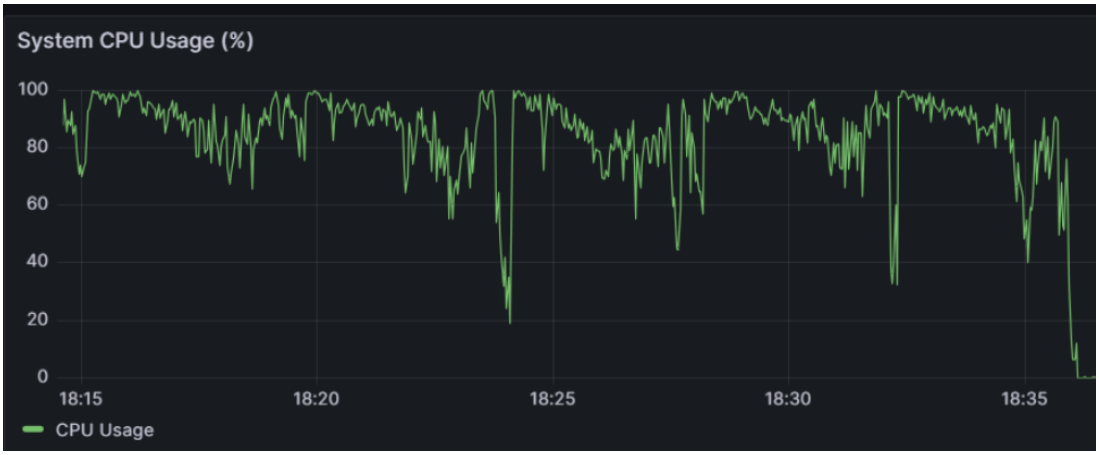


MD Core #2



CPU usage

MD Core #1

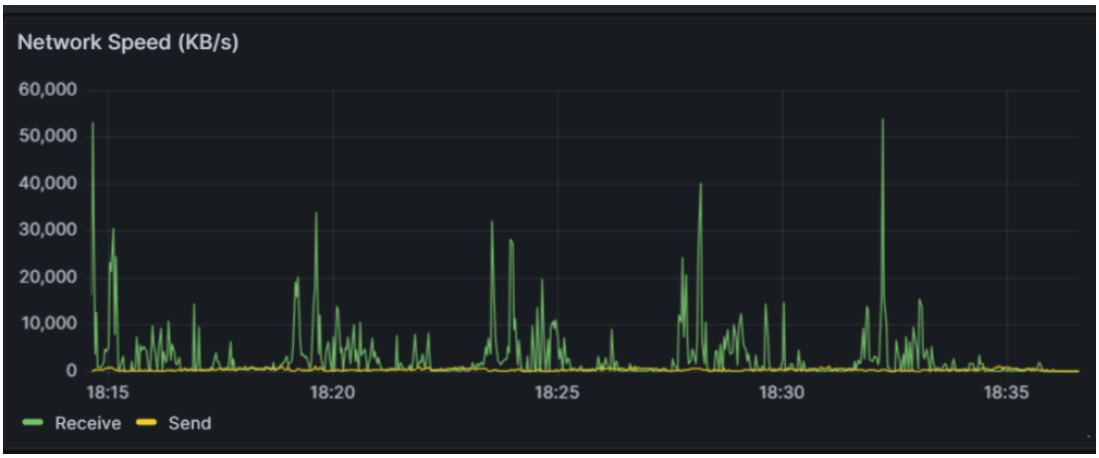


MD Core #2

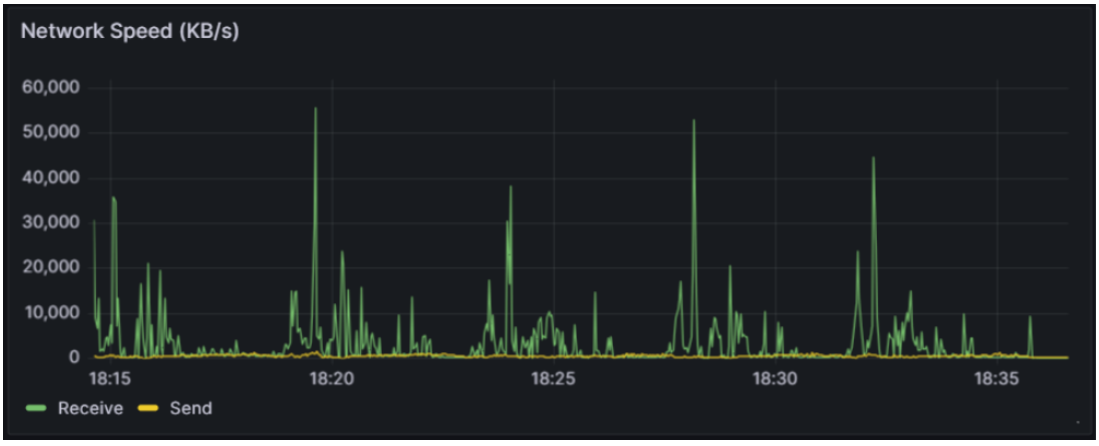


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

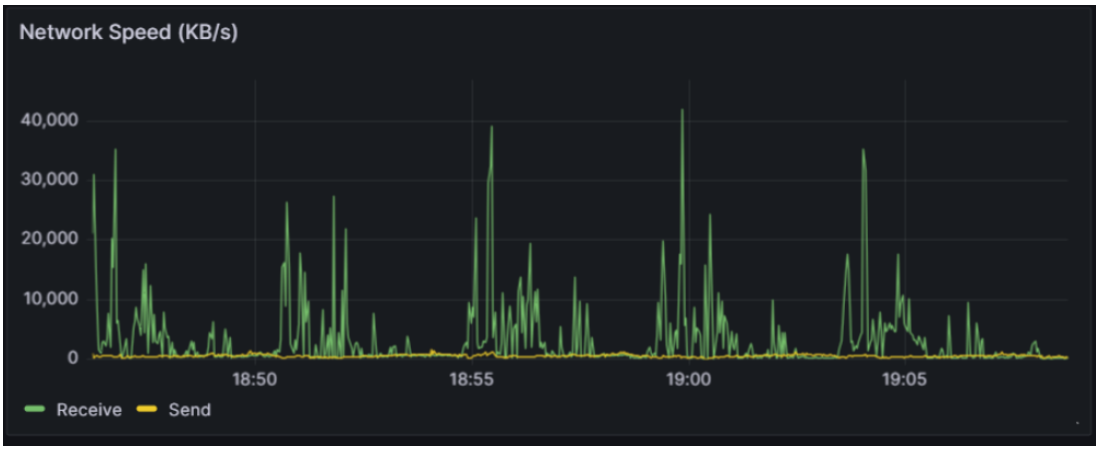


MD Core #2

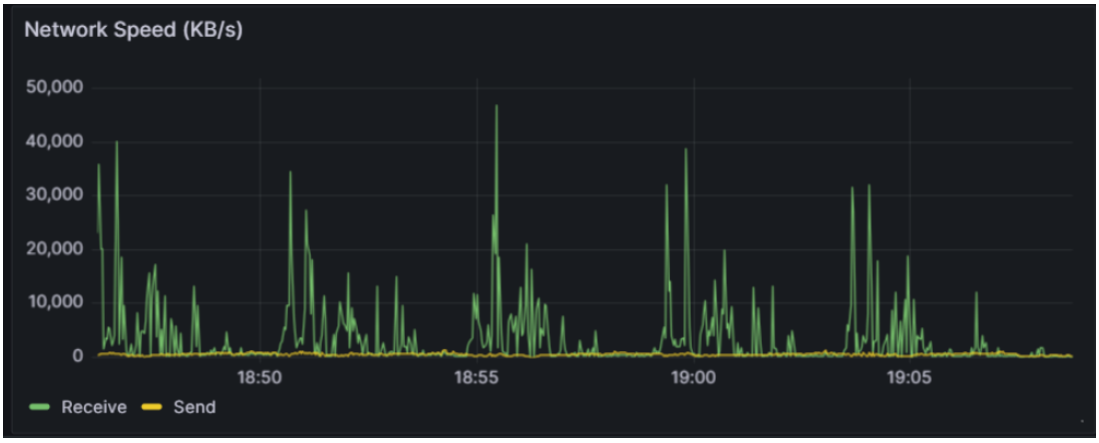


Network usage

MD Core #1



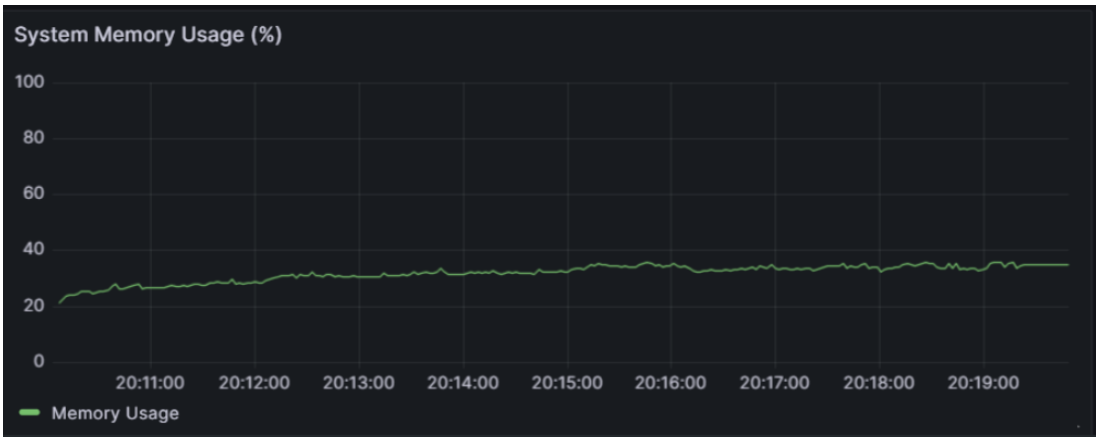
MD Core #2



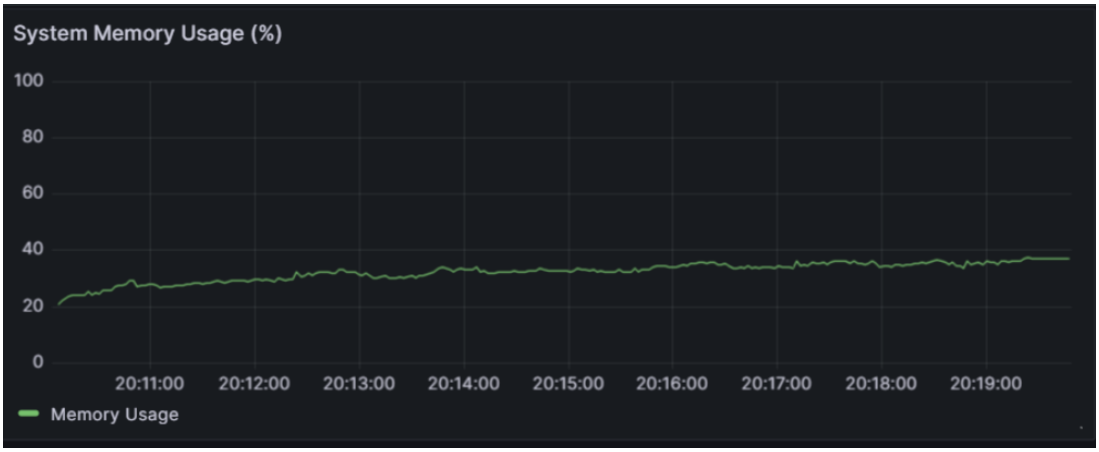
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1

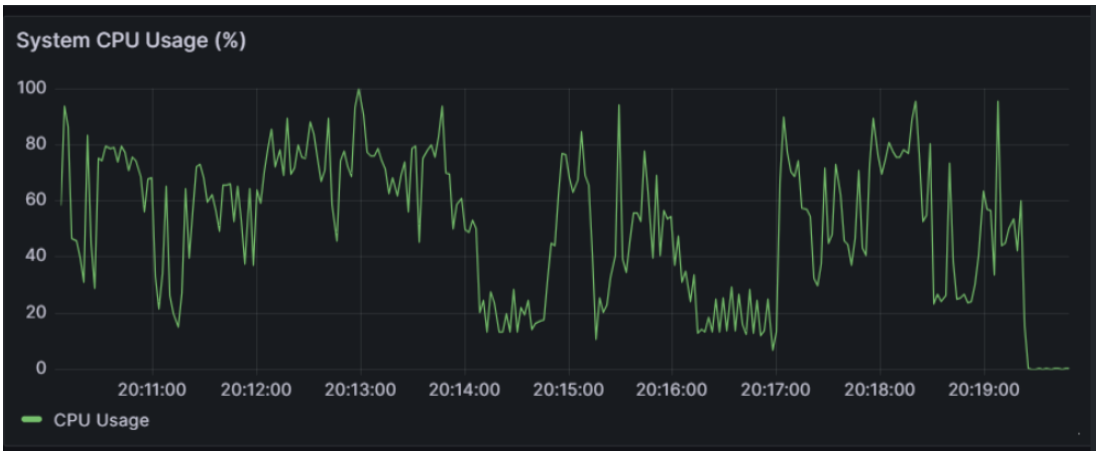


MD Core #2



CPU usage

MD Core #1

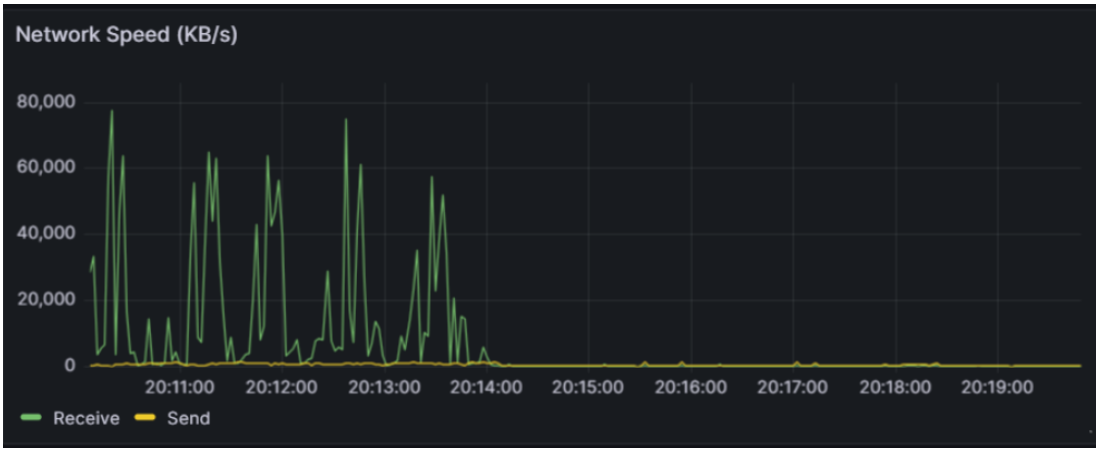


MD Core #2

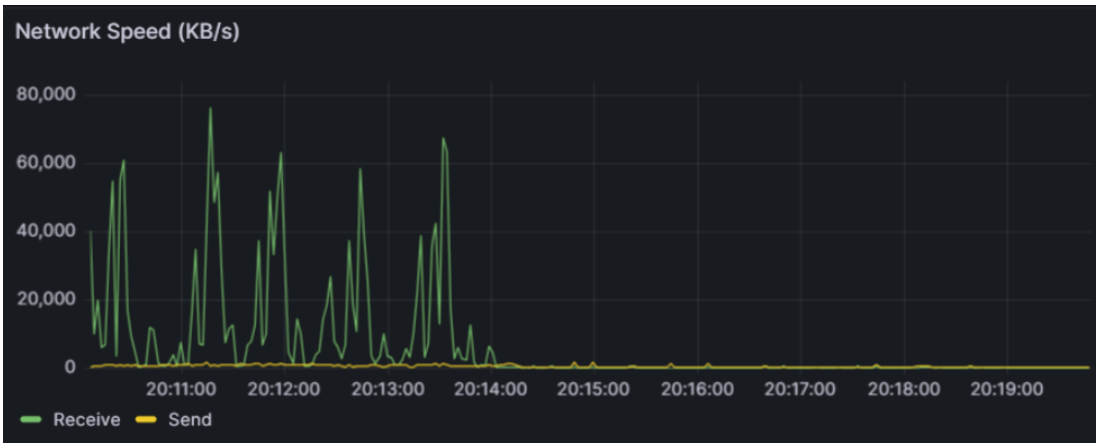


Network usage

MD Core #1



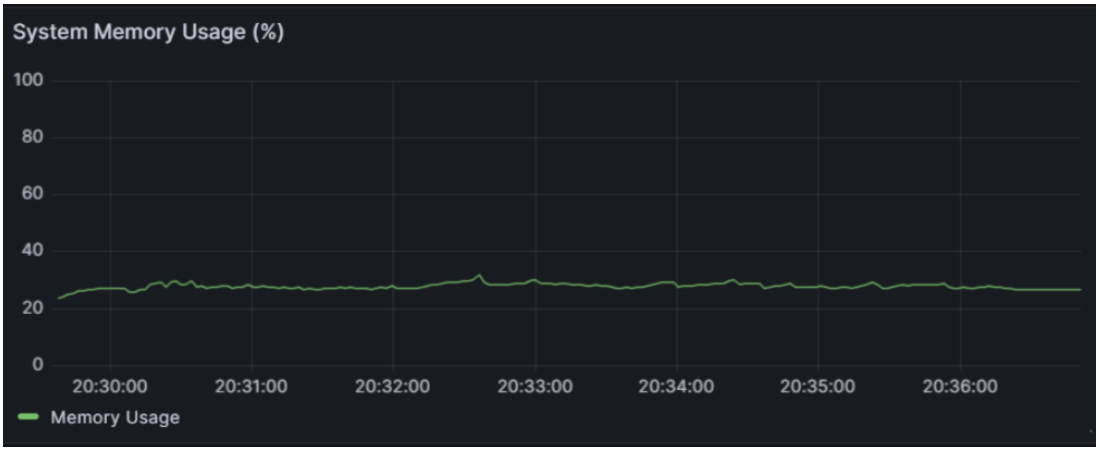
MD Core #2



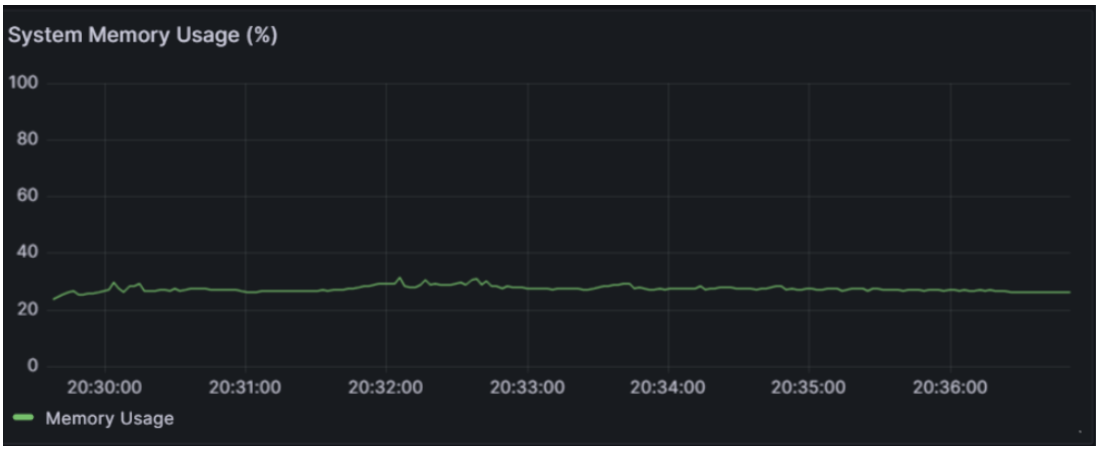
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1

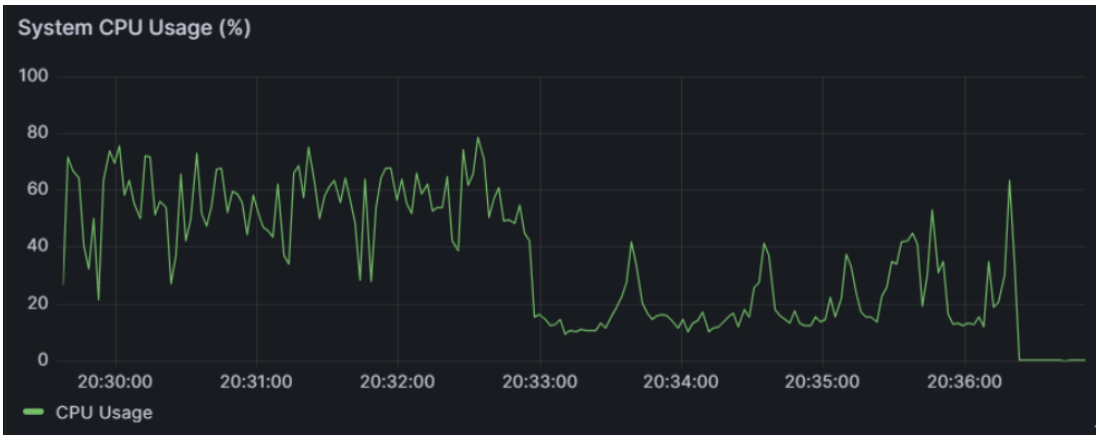


MD Core #2

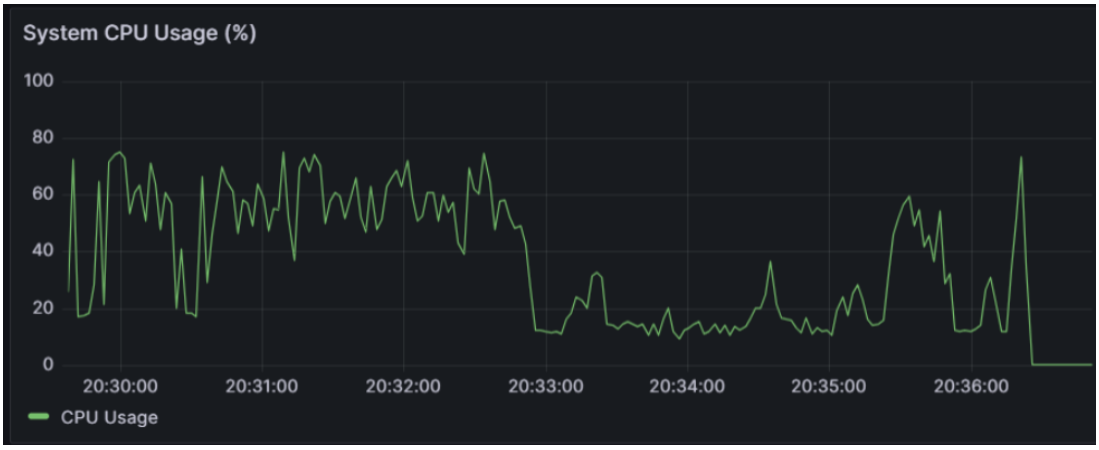


CPU usage

MD Core #1

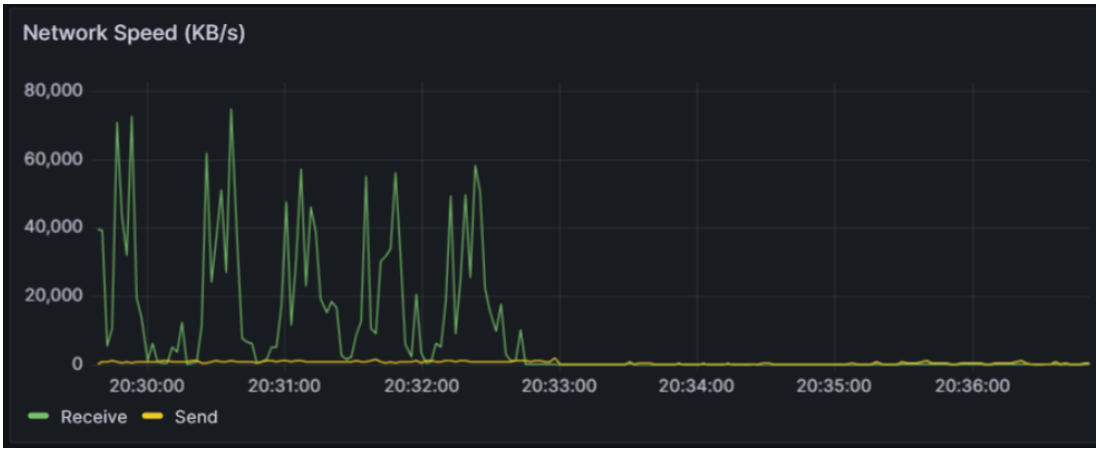


MD Core #2

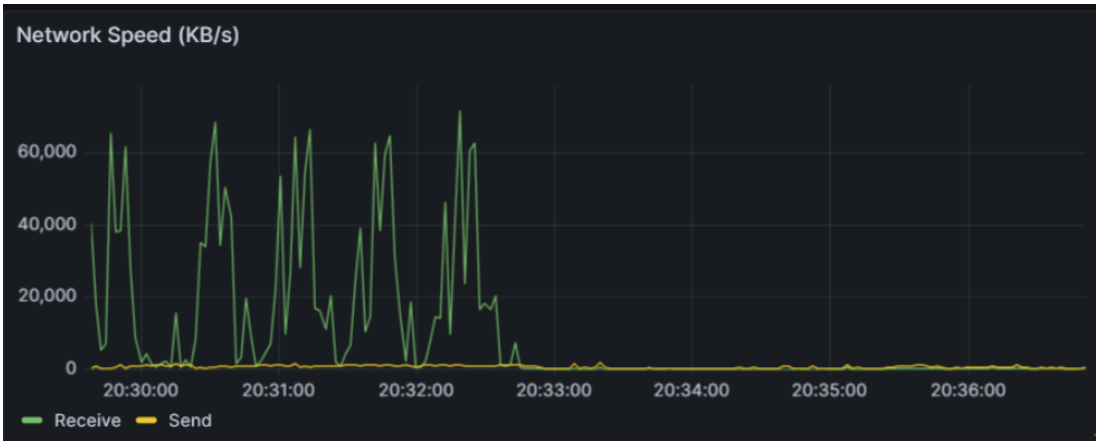


Network usage

MD Core #1



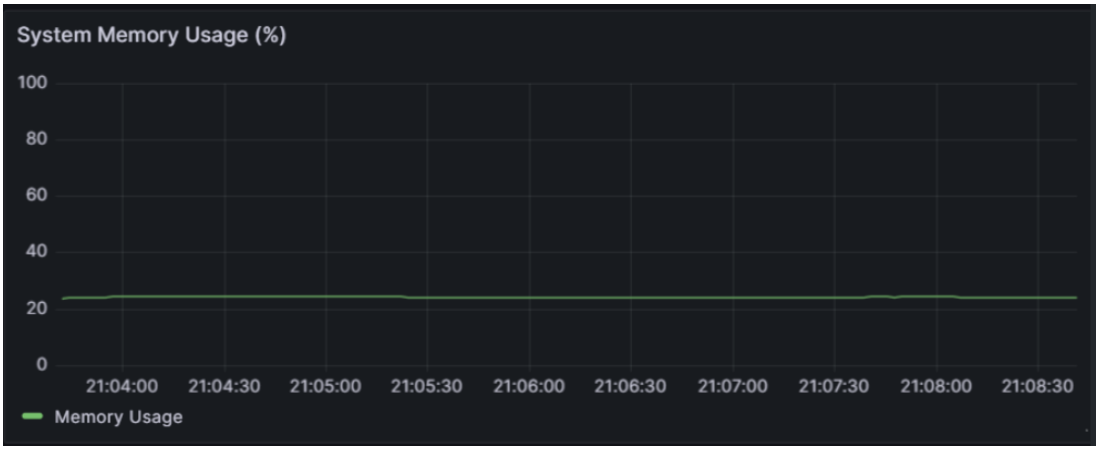
MD Core #2



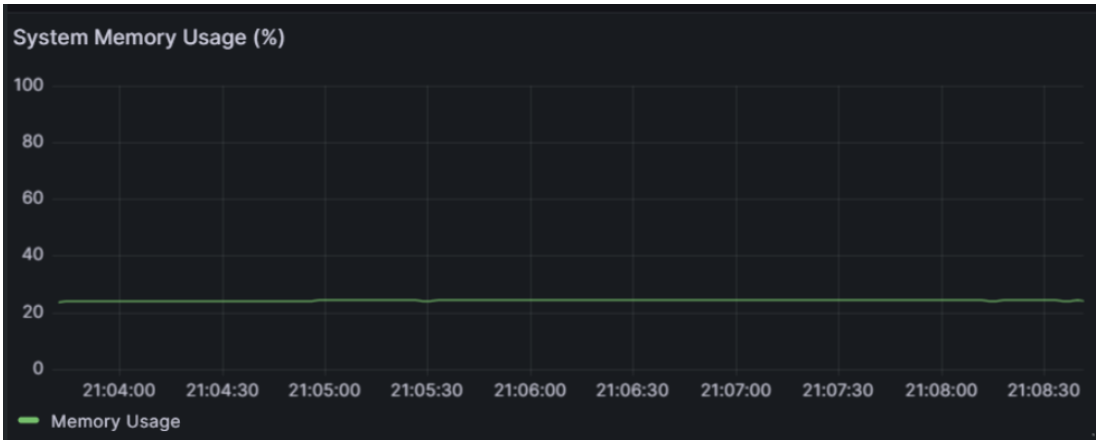
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

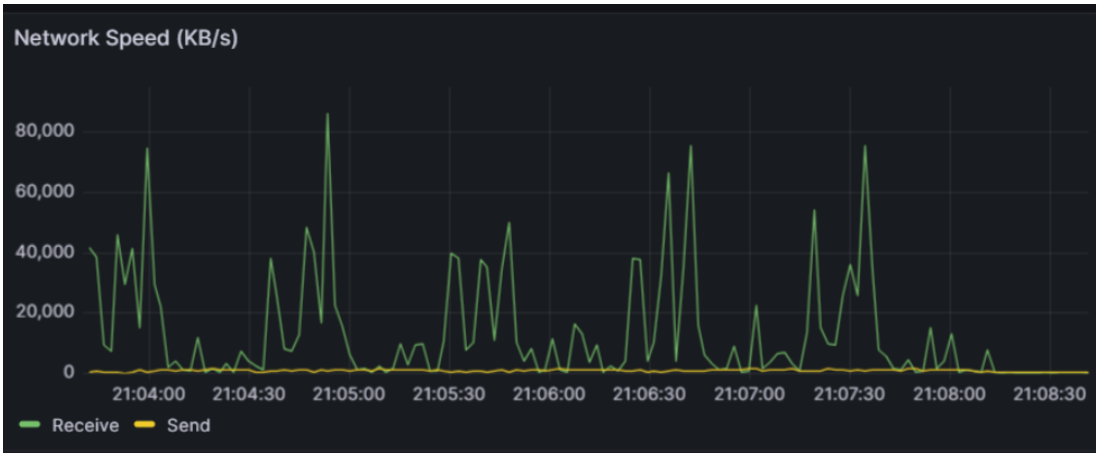


MD Core #2

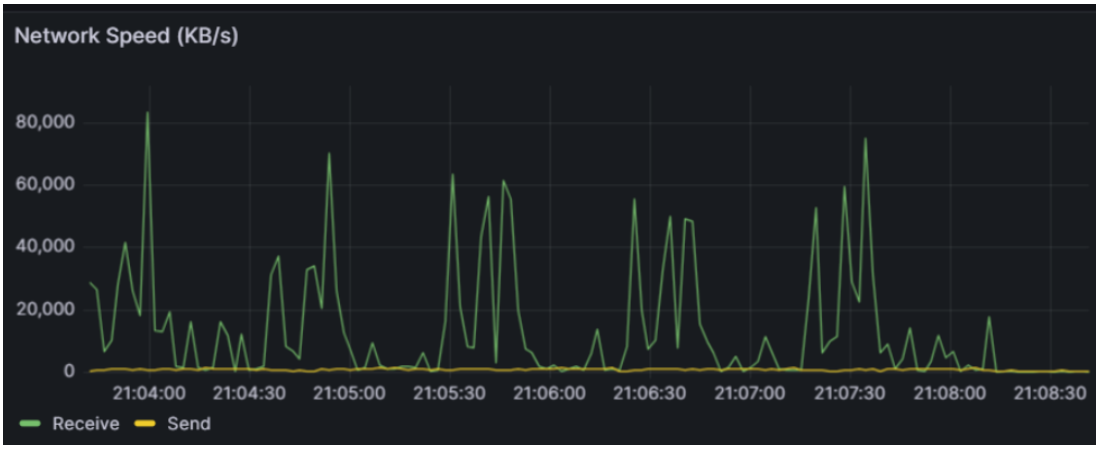


Network usage

MD Core #1



MD Core #2

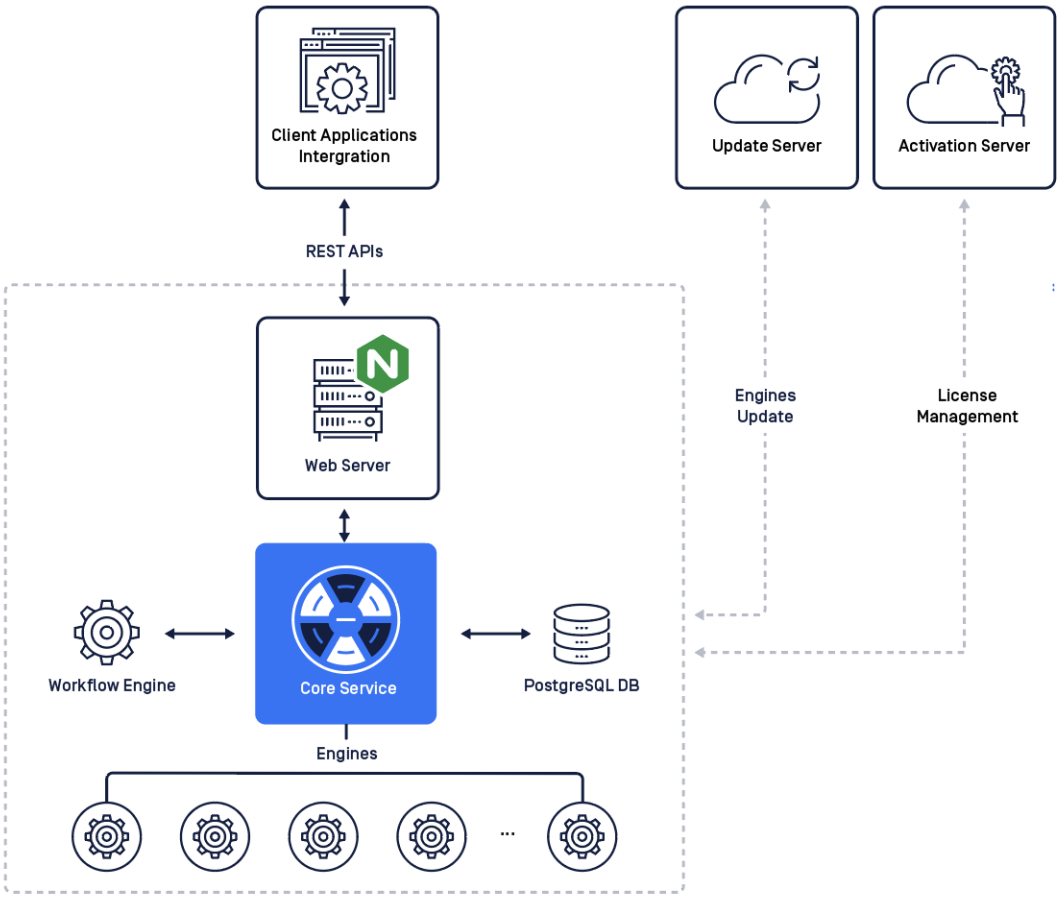

Network Speed (KB/s)
80,000
60,000
40,000
20,000
0
21:04:00
21:04:30
21:05:00
21:05:30
21:06:00
21:06:30
21:07:00
21:07:30
21:08:00
21:08:30
Receive
Send

MetaDefender Core Documentation

The users can consult this web page or, alternatively, they can download the manual in pdf format from the link below:

MetaDefender Core manual (SHA256:).

Recommended System Configuration



MetaDefender Core supports a wide range of deployment options, including air-gapped environments, providing the flexibility to choose the environment that best suits customers' requirements and infrastructure.

- Physical server.
- Virtualization platforms like VMware, Hyper-V and XenServer.
- Cloud Deployment.
- Container Deployment.

> **Note**
> Before installing MetaDefender Core, please refer to the **recommended minimum system configuration** listed below. The requirements are based on a standardized customer deployment and may require more or less resources to achieve the level of performance desired in your environment. Please note that the server specifications are built to allow a high volume daily processing. Please refer to Performance and Load Estimation for more details.

The recommendations below are for MetaDefender Core, API usage only. For any other use cases, please consult the user guide of the licensed products for accurate recommendations.

For certain use cases these might be adjusted and customized on specific needs and SLAs. We highly recommend to engage our ProServ team to assist in fine tuning MetaDefender and get the maximum performance out of your systems.

Microsoft Windows Deployments

[Windows] Supported Operating Systems

Only support 64-bit platforms.

- Windows 10, Windows 11 23H2, 11 IoT (22H2) only for End-User systems (e.g. Kiosk deployments)
  - Based on Microsoft License Terms, you're not allowed to use Windows Desktop as a server, therefore, MetaDefender Core can't be used on these systems for server use cases.
- Windows Server 2016, 2019, 2022

> **Disclaimer**
> End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

> **Plan for legacy OS support**
> OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:
>
> - Ubuntu 20.04 after **April 2025**.
> - Amazon Linux 2 after **June 2025**.
> - Windows 10 after **October 2025**.
>
> If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

[Windows] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 8	8	8 GB	16 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 12	16	16 GB	24 GB + 50 GB + 100 GB
MetaDefender Core 16	16	16 GB	32 GB + 50 GB + 100 GB
MetaDefender Core 20	32	16 GB	40 GB + 50 GB + 100 GB
MetaDefender Core MAX	32	32 GB	120 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core is installed.
- For a better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.
- SBOM processing requires more disk space, please allow at least 15GB free disk for the engine to work properly.

[Windows] Third Party Dependencies

- .NET framework 4.5 or above
- Microsoft Visual C++ Redistributable for Visual Studio 2015-2022 (version 14.38 or higher); x64 version.
- PostgreSQL: Only applicable when using a pre-installed PostgreSQL server running remotely.
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Some engines also have dependencies as described below:

Engine Name	Dependency
Deep CDR	Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 5.8.0 or above)
ESET	MetaDefender Core v4 temporary directory should have more than 200MB free disk space
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
Proactive DLP	• Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 2.0 or above) • .NET 6 dependencies • CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
Scrutiny	Microsoft Visual C++ 2015 - 2019 Redistributable x86 and x64
Systweak	.NET runtime 6.0, 7.0 or 8.0
Bkav Pro	Microsoft Visual C++ 2015 - 2022 Redistributable x64

[Windows] Installation Details

MetaDefender Core on Windows uses "C:\Program Files\OPSWAT" folder for storing resources or the installation directory.

MetaDefender will use its resources folder to store temp files as part of the analysis. **It's recommended to exclude this folder from real-time protection monitoring**. Knowledge base:

- How to set up exclusions on anti-malware software to prevent disruption
- Mitigating scan failure triggered by real-time protection service
- Can local AVs interrupt ongoing scans?

Linux Deployments

[Linux] Supported Operating Systems

Only support 64-bit platforms.

- Red Hat Enterprise Linux 8, 9
- Rocky Linux 9
- Debian 11, 12
- Ubuntu 20.04, 22.04
- Amazon Linux 2 is supported via Amazon Machine Image (AMI)

Plan for legacy OS support

OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:

- Ubuntu 20.04 after **April 2025**.
- Amazon Linux 2 after **June 2025**.
- Windows 10 after **October 2025**.

If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

For Hardended Linux OS

Some AV engines could be failed to initialize and run on MetaDefender Core (e.g. ESET, Kaspersky) due to execution permission required by them on `/var/tmp/ometascan` folder.

Please follow steps at Why have the ESET and Kaspersky scan engines failed to initialize on the hardened Linux OS? for remediation details.

Disclaimer

End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

[Linux] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 5	4	4 GB	10 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 10	8	8 GB	20 GB + 50 GB + 100 GB
MetaDefender Core MAX	16	16 GB	40 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core resides on.
- For better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.

[Linux] Third Party Dependencies

Dependencies list:

- openssl
- grep
- lib32stdc++6 (>= 4.5)
- libc6-i386 (>= 2.10)
- procps
- zlib1g
- libcurl3 (>= 7.19.7)
- libcurl4
- postgresql:
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Not all the above dependencies will need to be installed, it is dependent on different Unix distro and version

Some engines also have dependencies as described below:

Engine name	Dependency
Archive engine	• To extract password protected MS Office files, you need to install .NET 6 dependencies, libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • Rocky Linux 9.x/Red Hat 9.x: libnsl
Deep CDR	• Rocky Linux 9.x: libgomp, ncurses-compat-libs, libnsl • CentOS: libgomp, ncurses-compat-libs • Red Hat 8.x: libgomp, ncurses-compat-libs, libnsl • Red Hat 9.x: libgomp, ncurses-compat-libs, libnsl, compat-openssl11 • Ubuntu/Debian: libgomp1, libncurses5 • libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • .NET 6 dependencies
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
FileType engine	Rocky Linux 9.x/Red Hat 9.x: libnsl
Proactive DLP	libgdiplus, mscorefonts .NET 6 dependencies CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
RocketCyber	libgomp

Note: Some dependencies may need to be installed from external repositories, not from the OS default repositories.

[Linux] Installation Details

MetaDefender Core default installation path is using /var folder for storing resources:

- `/var/lib/ometascan` : installation folder with all its resources (database, DLP processed / CDR sanitized / quarantined file storage, engine package and definition updates).
- `/var/log/ometascan` : application logs.
- `/var/run/ometascan` : runtime folder for Core related processes.
- `/var/lib/ometascan` : all successful deployed engines data files.
- `/usr/lib/ometascan` : all shared libraries required by MetaDefender Core, bundled PostgreSQL server related binaries and libraries.
- `/var/tmp/ometascan/resources` : all temporary processing files for MetaDefender Core to process.

Storage Usage Information

Based on the configuration, MetaDefender Core will need additional disk space to store analysis/result data, quarantined files and temporary files:

	Data Storage
Analysis Reports (Processing History)	Approximately 4.5GB is required for every 1M analysis reports stored. • 100 GB is recommended as a baseline. Database storage requirements are affected by Data Retention settings, engine settings and scanning load.
Quarantined Files	Depends on the customers' dataset and configuration
Sanitized Files	Depends on the customers' dataset and configuration
Temporary processing files	Depends on the customers' dataset and configuration • A 1GB non-archive file requires 1GB free disk space at least. • A 1GB archive file requires (1 * 2)GB free disk space at least. • 50GB is a baseline and should be adjusted based on observation of the solution within the environment, with consideration to peak times and overall file makeup (e.g. complexity, size, archives).
Note	Options are available to control how large and/or complex files are handled to limit the amount of system resources required. See Workflow Configuration Template for more information.

Custom Engines

The recommendations above are specific for MetaDefender pre-packaged bundles.

However for additional Custom Engines, please review the Knowledge base to review additional requirements (if any) for the selected engine.

Browser Requirements for the MetaDefender Core Management Console

One of the following browsers is suggested to view the MetaDefender Core Management Console:

- Microsoft Edge
- Chrome
- Firefox
- Safari

Chrome, Firefox, Safari and Microsoft Edge browsers are tested with the latest available version at the time of release.

Network Bandwidth

In online environment where MetaDefender Core is configured to download engine update packages directly from OPSWAT update server source, then using gigabit network speed (125 MB/sec) is highly recommended to avoid potential network related issue while downloading files, or at least 100 Mbps (12.5 MB/sec) network speed.

Allowlisting URLs

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts':

- https://activation.dl.opswat.com - for license online activation
- https://update.dl.opswat.com - for engines and databases download

Note: *IP address-based allowlisting on your firewall might fail after some time since OPSWAT uses CDN (AWS Cloudfront) to faster delivery updates over the world, and IP address of edge servers might change over time.*

Depending on your location, connections with OPSWAT cloud-based servers be forwarded to specific AWS Cloudfront's edge locations, each is assigned with its own unique IP address ranges. You might also need to allow correct AWS Cloudfront's corresponding IP address ranges by referring to https://ip-ranges.amazonaws.com/ip-ranges.json

Warning

Even the OPSWAT update servers host updates for all of the available engines we support, sometimes custom engines might try to connect to their own cloud for updates, but this can be disabled in firewall and they will be updated just from OPSWAT cloud servers instead.

Differences Between MetaDefender Core v4 and v5

We are very excited to inform you about the release of version 5.0.0 of our MetaDefender Core!

The new version is designed to enhance the product performance, user experience, as well as to provide more flexibility, security, and scalability for our customers.

At the same time, we are also committed to keeping backward compatibility for whom running MetaDefender Core v4, and avoid the need for updating your client integration to move forward with MetaDefender Core v5.

v4 REST API Integration Supported
All public REST API endpoints available on MetaDefender Core v4 will be continously supported on v5.

v4 Product Major Features Supported
All product major features available on MetaDefender Core 4.21.2 (the latest one on v4) will be continously supported on v5.

Seamless Product Upgrade
Upgrading from MetaDefender Core v4 to v5 will be supported and covered with data migration without additional manual steps from users.

Licensing information will be also migrated.

Below are some highlights of the first v5 release, MetaDefender Core 5.0.0:

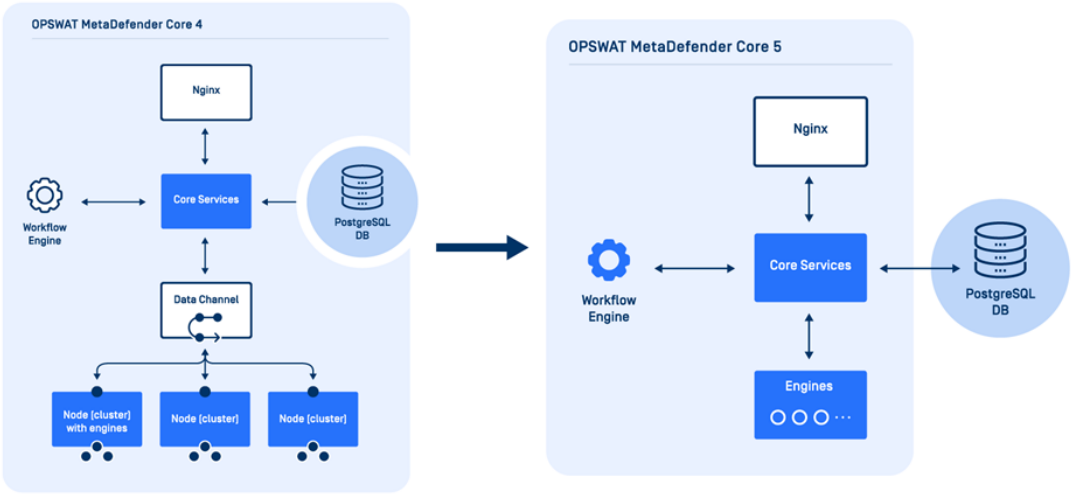
New product architecture

This version introduces a simpler product architecture that speeds up integration and automation, file processing flow, and immensely enhance the product performance and efficiency.

Earlier versions utilized MetaDefender Node concept, designed to support scaling of the scanning infrastructure by distributing scan requests among several different processing Nodes. Product licensing was based on number of Node instances. The benefit of having such a distributed infrastructure is that based on Node loads, Metadefender Core server can always choose the most appropriate Node to assign a new scan task to. However, having nodes in the v4 architecture caused more complexity and possible latency in the processing flow and entailed maintenance cost.

In this release, we retired multi-Node deployment and restructured MetaDefender Core's backend. This alteration significantly improves file processing speed and product efficiency, as well as eliminates extra cost for maintaining node services. Users now can simply install additional MetaDefender Core instances for scalability requirement,

as well as configure their own load balancer to coordinate tasks among multiple MetaDefender Core instances. Accordingly, product licensing is now based on number of the product instances.



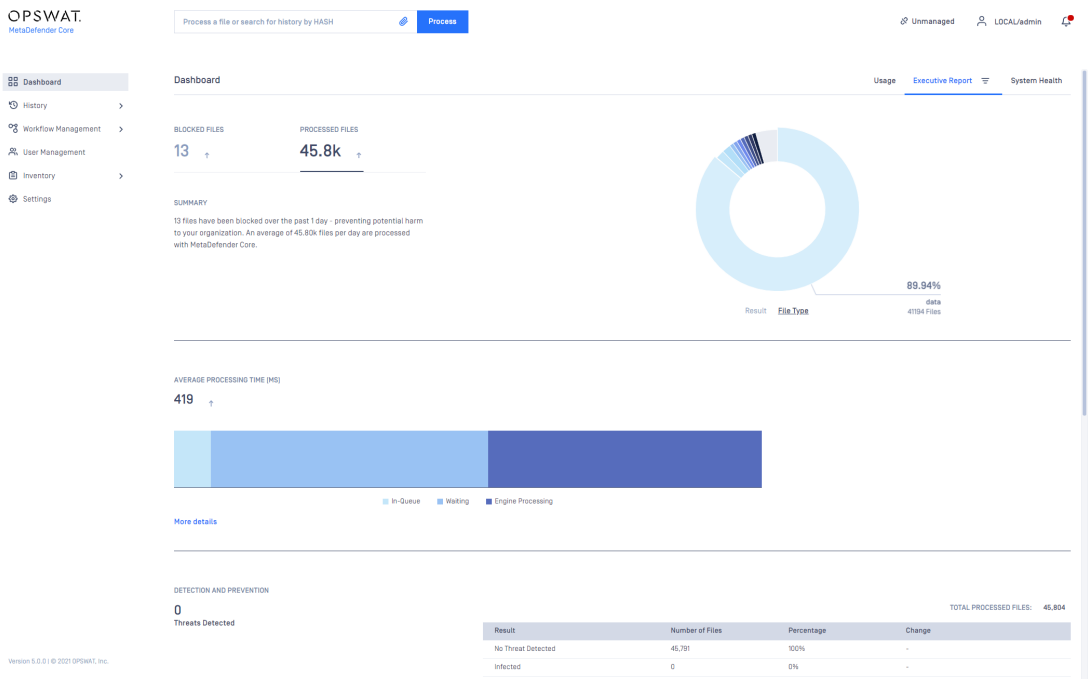
Please be informed that

- All existing administrative APIs related to "Node" on v4 will be kept as it is on v5 to ensure backward compatibility with existing client integrations.
- Applicable settings defined in Windows registry (Windows) or ometascan-node.conf config file (Linux) for Node will be migrated to Core. Details: MetaDefender Configuration

Brand-new product user interface

The second notable update in this version is a brand-new UI for the management console, which was designed with the focus on the quality and thoughtfulness of the user experience. The new administrative console UI provides an advanced set of Core features and observable processing status.

Comprehensive executive dashboard reporting offers dynamic, real-time and actionable threat insights. It also provides visibility of your security posture allowing you to paint a picture of security that aligns with the overall business. The report includes easy-to-read charts, graphs, and statistic explanation that help your security professionals reduce clutter, gather information quicker, spot threat activity early and act swiftly.



We're continuing to improve our product UI to provide users an excellent experience with the application and overall usability. Please expect more powerful UI features coming in next releases.

New compression archive engine for archive sanitization

In previous version, MetaDefender Core's archive engine implemented four functions including decryption, extraction, encryption and compression that with a high volume of files it takes time to process them all. For the purpose of enhancing archive handling performance, this release comes with a separated compression archive engine, which dedicatedly processes archive sanitization (compression and encryption). It is also created to facilitate maintenance and further feature development.

Archive sanitization feature now requires Deep CDR license

As to respect sanitization practice and ensure unknown threats prevention as designed, now starting on version 5, it's mandatory to have Deep CDR licensed for archive sanitization feature enablement on MetaDefender Core workflow configurations.

Archive handing related engines changed

The legacy "Archive" engine on MetaDefender Core v4 will be changed its name to "Archive Extraction" engine, even when you are still on MetaDefender Core v4.

A brand new "compression archive engine" is to dedicatedly handle archive sanitization in v5 (archive compression), that helps relevant archive handling performance improvement, easier for maintainance and further feature development.

Notes when you are still on MetaDefender Core v4 (not upgraded yet)

The legacy "Archive" engine (now name changed to "Archive Extraction" engine) on MetaDefender Core v4.x will still cover both extraction and compression archive tasks as usual.

Even though your MetaDefender Core v4.x might now have a new "Archive Compression" engine auto downloaded, but that engine cannot be deployed (marked as "permanently failed" in relevant engine status check API). This behavior is expected, and not causing any impact to MetaDefender Core processing flow, or its performance though.

- Just in case your client integration has a check for engine status before triggering scan request, then you might want to disable "Archive Compression" engine status on MetaDefender Core v4 using REST API. Detailed instruction: Archive Compression Engine Permanently Failed on MetaDefender Core v4
- When you later upgrade to MetaDefender Core v5 from v4, then you might need to manually re-enable "Archive Compression" engine to make it active and running on MetaDefender Core v5.

Notes for licensing

For offline deployment (MetaDefender Core is not connected to the Internet directly for online update), you will need to re-activate again on OPSWAT portal to retrieve latest engine list in .yml file including "Archive Compression" engine if you are currently licensed with Deep CDR.

Meanwhile, for online deployment, this manual procedure is not required since MetaDefender Core will cover that itself.

Import & export configurations

Not only workflow settings like in previous versions, but also all other general settings (update, security, data retention, server, hashes to skip) and most importantly, user settings, will be able to export and import on MetaDefender Core version 5.

Learn more at Import/Export configuration

Backward compatibility supported

The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

Windows application event logging disabled by default

Applicable only to Windows OS based MetaDefender Core.

Help avoid duplicated logging with separate file option, and improve overall processing performance. Of course, when being still valid to you, you are still supported to enable it back.

CSRF token header renamed

CSRF token originally created for security strengthening and being used by MetaDefender Core management console UI since version 4.21.0 via header `oms_csrf_token`

To get it compliant with some load balancers which do not accept underscore in header name by default, and save you from additional load balancer configuration step, we have now changed the header name to `oms-csrf-token`

This change does not impact to any REST API based client integration.

More comprehensive and accurate error messages on REST APIs

Impacting on following REST API endpoints:

- `PUT /admin/config/session`
- `PUT /admin/config/update`
- `PUT /admin/config/scan`
- `PUT /admin/config/skip`
- `PUT /admin/config/auditlog`
- `PUT /admin/config/webhook`
- `PUT /admin/config/rule`

Upload time and queue time logic updated in JSON scan response

Impacting on following REST API endpoints:

- `GET /file/{data_id}`
- `GET /hash/{md5/sha1/sha256}`
- `GET /archive/{data_id}`
- `POST /file/{async}`

Added new field:

- `receive_data_timestamp`: The timestamp when upload progress started (first byte received) (in milliseconds)
- `upload_time`: Total time elapsed for upload process (in milliseconds)

Modified existing field (new logic):

> **Backward compatibility supported**
> The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

- `upload_timestamp`: The timestamp when upload progress finished (all bytes received) (in milliseconds)
- `queue_time`: Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue time = start_time - upload_ timestamp) (in milliseconds)

Release version 5.0.0 also comes with variety of outstanding features and improvements.

For detailed information about our latest release please view our Release notes

Installation

Starting with MetaDefender Core 4.20.0, the centralized/shared database deployment model is supported (all MetaDefender Core instances share the same database), along with traditional standalone database mode (every MetaDefender Core instance has its own database to manage).

- Standalone mode
- Shared Database Mode

Installation

1. Download the package of your choice from My OPSWAT
2. Install the package on your computer via the Command Line or via the Wizard
3. Open a web browser and point to http://<server name or IP>:<port>
    - The default port is **8008**
    - In case of problem check Inaccessible Management Console page
4. Complete the required steps of the basic configuration wizard
5. You must activate this deployment to use its features

Installation notes

- If the MetaDefender Core package **dependencies are not installed** on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.
- During installation the databases might need to be upgraded. This could take noticeable time depending on database size (eg.: length of scan history).

Standalone mode

Wizard Installation

The Install Wizard is only for the Windows installer (.msi file).

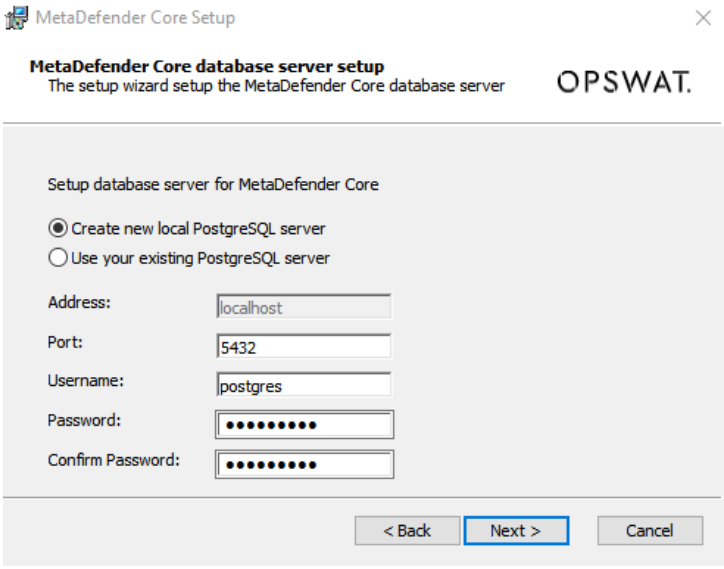
Since MetaDefender Core 4.19.0, we have switched to using PostgreSQL DBMS which needs a dedicated database server. MetaDefender Core supports two modes to setup PostgreSQL server:

Local (Bundle) PostgreSQL server

MetaDefender Core will install a new PostgreSQL server locally in the same box with the product.

> **PostgreSQL user credentials**
> Unicode characters are NOT supported for "Username" and "Password"
>
> "Password" does not accept # (hash) as starting character (e.g. #ABC)

> **The ignition file**
> If the ignition file exist in the designated location, then MetaDefender Core will use the details defined in the ignition file, rather than what is passed via the installation wizard.
>
> More info about the ignition file based installation: Command Line Installation

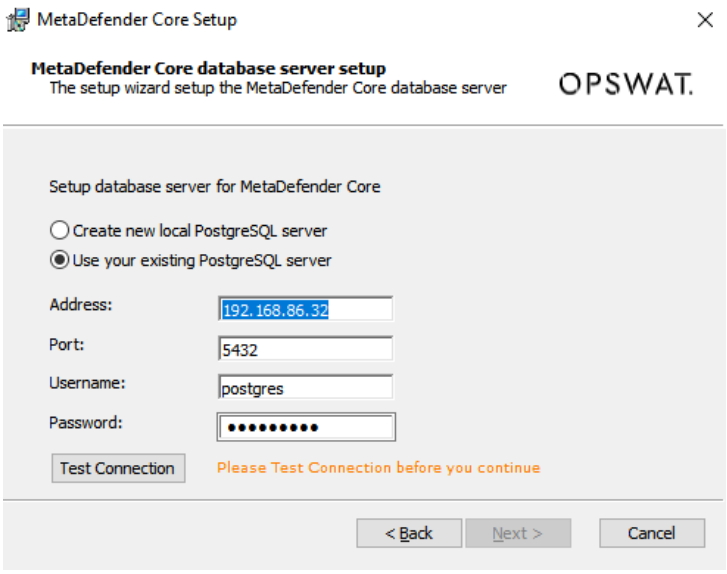


Remote (Existing) PostgreSQL server

MetaDefender Core will connect to remote pre-installed PostgreSQL running remotely.

> **PostgreSQL connection configuration**
> "Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

> **PostgreSQL user privilege requirements**
> MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.
>
> Details: PostgreSQL User Privilege Requirements



Command Line Installation

Ignition File

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Make sure to create the ignition file, if it doesn't exists before you go ahead and install MetaDefender Core via command line.

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

MetaDefender Core supports two modes to setup PostgreSQL server:

1) Local PostgreSQL server

- MetaDefender Core will install a new PostgreSQL server locally, next to the product.
  - A sample iginition file for PostgreSQL server information (***Only non-Unicode characters supported for "user" and "password"__***):

ometascan.conf bash

```
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
```

2) Remote PostgreSQL server

> **Info**
>
> **PostgreSQL version 12 (at least 12.3) or version 14 is required.**
>
> According to PostgreSQL notice, version 12 will stop receiving fixes on November 14, 2024, so we suggest that you use version 14.

MetaDefender Core will connect to a pre-installed PostgreSQL server running remotely and will setup its database.

Make sure to follow steps at Open Connection On PostgreSQL Server first, so that your remote existing PostgreSQL server to accept upcoming connection from MetaDefender Core.

A sample ignition file for PostgreSQL server information:

ometascan.conf bash

```
[dbserver]
type=remote
host=192.168.86.32
port=5432
user=existingserver_admin_user
password=existing_server_pass
```

After installing MetaDefender Core successfully, you may want to remove all credentials info created for [dbserver] section in ignition file for security reason.

> **Warning**
> If the MetaDefender Core package dependencies are not installed on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.

Debian package (.deb)

bash

```
sudo dpkg -i <filename> || sudo apt-get install -f
```

On Red Hat Enterprise 7.x Linux / CentOS package (.rpm)

bash

```
sudo yum install <filename>
```

For systems which enabled GPD check flag:

bash

```
sudo yum install --nogpgcheck <filename>
```

Red Hat Enterprise Linux 8.x package (.rpm)

Install the latest PostgreSQL repository RPM:

> **Note**
> Due to the shorter support cycle on Red Hat Enterprise Linux, all supported versions of PostgreSQL are not available on this platform. The PostgreSQL Yum Repository will integrate with the normal systems and patch management, and provide automatic updates for all supported versions of PostgreSQL throughout the supported lifetime of PostgreSQL. Reference: https://www.postgresql.org/download/linux/redhat/

bash

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

In case installing a remote PostgreSQL on either the same MetaDefender Core server or another server, you need to install the following additional package. PostgreSQL version 12.3 or newer is recommended.

bash

```
sudo dnf install -y postgresql12-contrib
```

Install MetaDefender Core:

bash

```
sudo dnf install -y <filename>
```

Windows package (.msi)

On Windows systems it is possible to install the product by running the corresponding .msi file.

From command line interface it is also possible to install the product by executing

bash

```
msiexec /qn /i <msi file name> <option key>=<option value>
```

where the possible keys and their default values are the following:

Key	Default Value	Description
INSTALLFOLDER	\Program Files\OPSWAT\MetaDefender Core	Customize installation folder for product Example: `INSTALLFOLDER="D:\Products"`
RESTADDRESS	*	REST interface binding IPv4 or IPv6 address ('*' means that service listens on all IPv4 and IPv6 interfaces)
RESTPORT	8008	REST interface binding port
REPORT_ENGINE_ISSUE	true	Enable reporting of engine issue count. (possible values: "true" or "false").

Note

Reporting of engine issue count

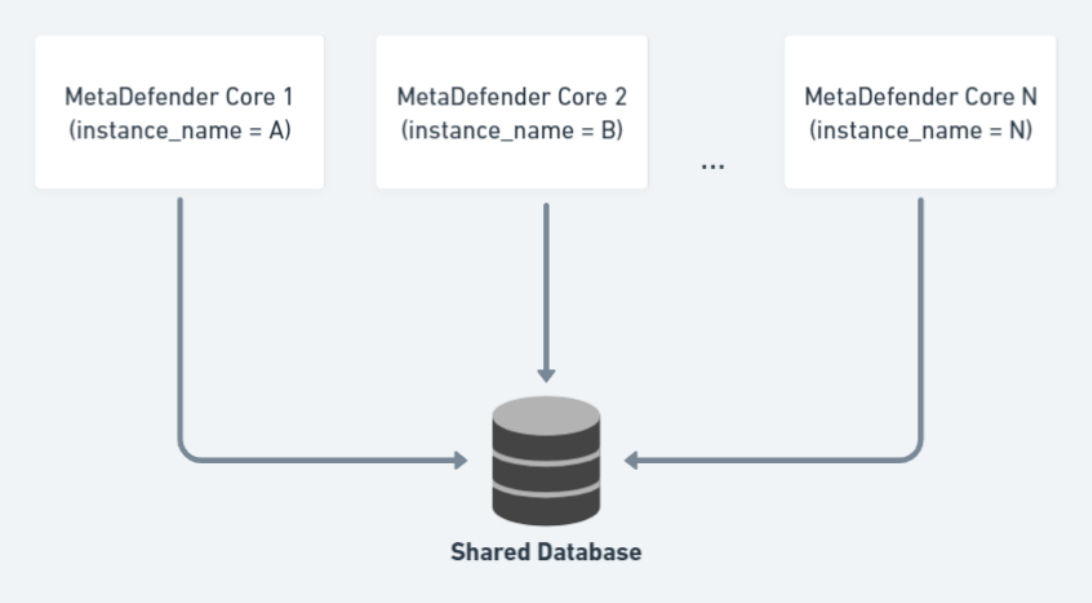
If reporting of engine issue count is enabled, MetaDefender Core server will send only the **number** of initialization errors and **number** of unexpected stops for the specific db/engine version. This information is sent over a HTTPS channel when the product downloads the latest package descriptors. This information is used for early detection of any specific 3rd party engine quality issues.

For details on using msiexec please consult Windows installer documentation.

Shared Database Mode

Starting with MetaDefender 4.20.0, you can use a centralized database deployment model, where all MetaDefender Core instances can share the same external database.

That helps auto-sync all product configurations (**except Single-Sign-On** that requires unique settings for each Core), processing history, live/on-going scan result query, audit, statistics data.



This article demonstrates how to install and setup MetaDefender Core 4.20.0 or newer with shared database mode.

Important notes & current limitations:

Currently not supported

- Does not support Central Management use-case with shared DB mode.
- Does not support to switch from shared database mode back to standalone database mode.
- Does not support to change instance_name of MetaDefender Core in shared database deployment model after installation.
- Does not support upgrading older MetaDefender Core from non-persistent mode to using the shared database mode.
- Does not support upgrading older MetaDefender Core from standalone mode to using the shared database mode.
- Does not support to upgrade multiple MetaDefender Core instances at once in shared database mode
    - Stop all Core instances' scanning service before performing the upgrade.
    - Upgrade each Core server one by one instead.

- For each Core, when the upgrade finished, then you can start using that Core for scanning without waiting for the other Core instances' upgrade to be done.

> **Disclaimer**
> MetaDefender Core does not support mode switching.

Requirements for shared database setup:

- The central remote PostgreSQL server (for shared database) could be utilized with a built-in (native) PostgreSQL database server installed with one of the MetaDefender Core instances in the network. Or using a dedicated existing PostgreSQL database server (see detailed requirements for PostgreSQL version at Recommended System Configuration).
- Before setting up, make sure to follow steps at Open Connection On PostgreSQL Server so that all MetaDefender Core instances could connect to the central remote PostgreSQL server.
- MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core, details: PostgreSQL User Privilege Requirements
- All MetaDefender Core instances must be installed on the same Operating System (Windows / Linux).
- The first MetaDefender Core instance connects to the shared database must be ready first (license activated, users/groups/roles created) before setting up other MetaDefender Core instances to connect and use that shared database.

Engine update:

- If engine updates is configured to be via Folder source, all MetaDefender Core instances must be able to access to the configured update folder, and the option DELETE FILES AFTER IMPORT will be unchecked and prevented for modification.

Authentication:

- Single-Sign-On (SSO) authentication settings (if using) cannot be shared among MetaDefender Core instances because SSO requires an unique login URL for every different Core.
- If an user is locked for login on a certain MetaDefender Core instance(e.g. exceeded the number of failed login times), that user is not prevented for logging in on any other instances.
- If an user is locked for sign-in on a certain MetaDefender Core instance (e.g. exceeded the number of failed login times), if the admin releases lockout on that user on that MetaDefender Core instance, that user is still not able to login on other MetaDefender Core instances.

Sanitized / DLP / Quarantined location changed after MetaDefender Core install:

Windows:

Shared storage path will be stored in 3 strings under key global in registry: quarantinepath, dlppath, sanitizepath as below

(Default)	REG_SZ	(value not set)
dbmode	REG_SZ	4
dlppath	REG_SZ	\\10.40.50.100\test
instance_name	REG_SZ	481rvj
quarantinepath	REG_SZ	\\10.40.50.100\test
restaddress	REG_SZ	*
restport	REG_SZ	8008
sanitizepath	REG_SZ	\\10.40.50.100\test

You can set either 1, 2 or 3 different paths for the 3 settings above (means that quarantined, DLP and sanitized files will be stored in same or different storages corresponding). In case shared storage is not set in installation, values of 3 settings above will be empty.

Restart MetaDefender Core service after changed.

Linux:

Open file `/etc/ometascan/ometascan.conf` and set shared storage path for 3 keys quarantinepath, dlppath, sanitizepath under section [global]

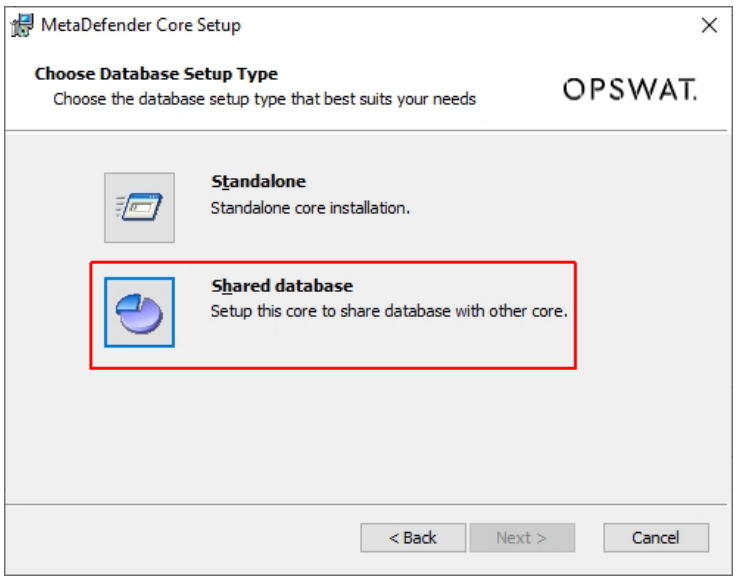
Restart ometascan service after changed.

Wizard Installation (Windows Only)

> **Info**
> Only applicable to MetaDefender Core 4.20.0 or newer using wizard on Windows OS.

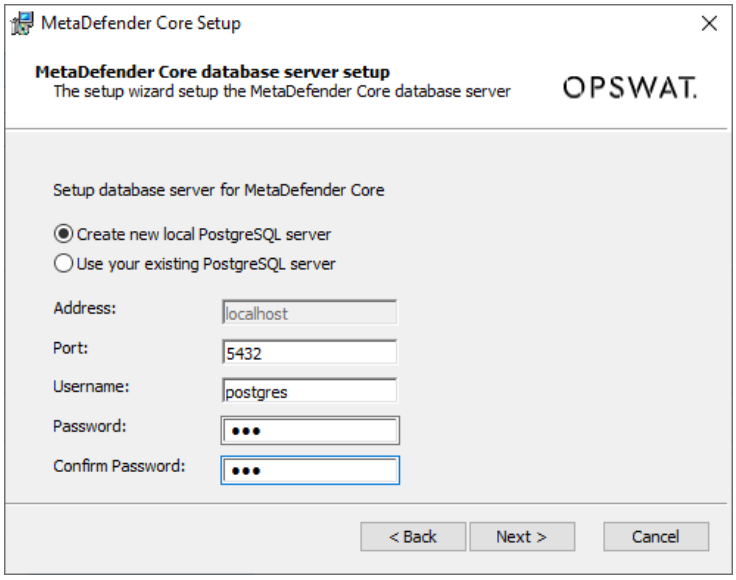
Clean install scenario

1. **Choose Shared database option:**

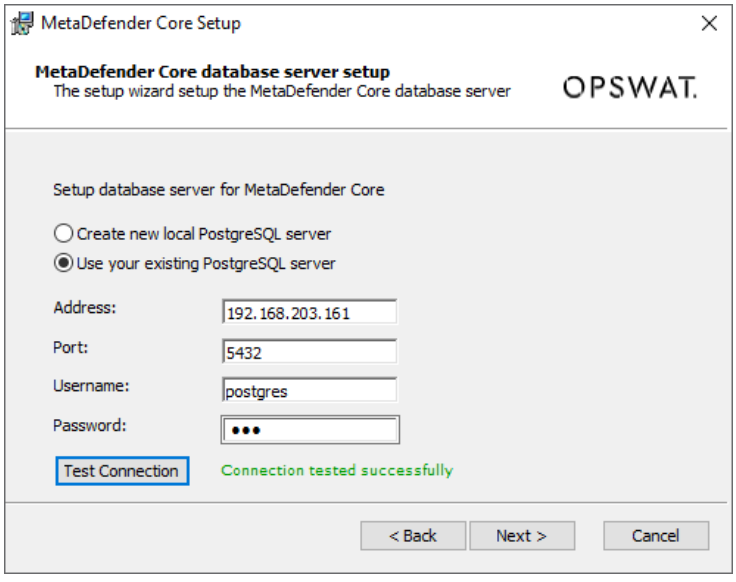


2. **Setup database server for MetaDefender Core:**

You can choose to create a native / built-in PostgreSQL server and setup the shared database on (then other MetaDefender Core instances will want to connect to that PostgreSQL server afterwards)



Or choose to connect to a pre-setup existing PostgreSQL server:



PostgreSQL connection configuration

"Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

PostgreSQL user privilege requirements

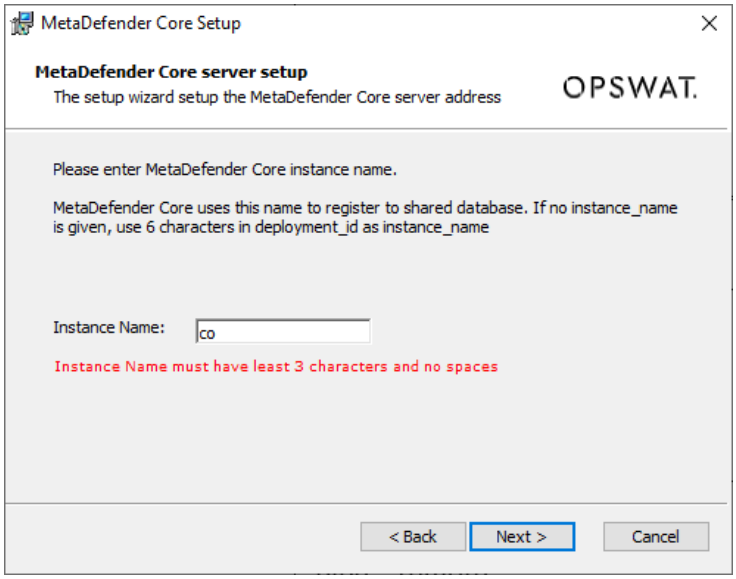
MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.

Details: PostgreSQL User Privilege Requirements

3. **Create an unique instance name for that MetaDefender Core (different from other MetaDefender Core instances):**

Instance name must contain at least 3 characters and no white space.

If you choose to leave it blank, which means it will use the default instance name with 6 characters extracted from that MetaDefender Core deployment ID.

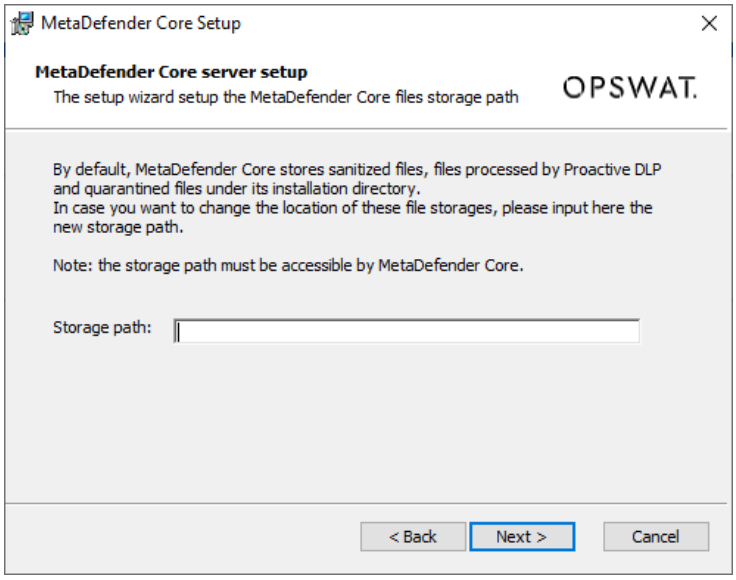


4. **Input the file storage path (sanitized, DLP, quarantined files)**

In the shared database model, all MetaDefender Core instances could use the same folder path to store all their own sanitized, DLP, quarantined files after processed.

The default location is: ***<MetaDefender Core installation folder>\data\ {sanitized|dlp|quarantine}***

If you wish to use that default location, just leave the input box blank and press Next.



Otherwise, you can specify a different location of sanitized/dlp/quarantined files. It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core.

For example:

- C:\new_folder
- \192.168.1.1\new_folder

Command Line Installation

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Clean install scenario

1. Create ignition file if not existed

- Windows: **C:\OPSWAT\ometascan.conf**
- Linux: ***/etc/opswat/ometascan.conf***

An example of ignition file:

ometascan.conf bash

```
eula=true

[global]
dbmode=4
instance_name=MetaDefenderCoreA
storage_path=D:\path\

[user]
name=admin
password=admin
email=admin@local

[dbserver]
type=remote
host=192.168.1.1
port=5432
user=postgres
password=your_own_password
```

Field	Required	Mapping to wizard UI	Possible values
global/dbmode	optional default is 1 (standalone)		1 - Standalone 2 - Central Hub deployment 3 - Non-persistent **4 - Shared**
global/instance_name	optional default is 6 characters of deployment ID		Unicode and number At least 3 characters No whitespace
global/storage_path	optional default is empty		Path to storages (DLP, quarantined, sanitized files) It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core. For example: • C:\folder • \192.168.1.1\folder

2. Run installers using command line to install

Please refer to Command Line Installation for installation command.

Deployment automation support

Note

An Ignition file could be utilized for either / both of following use-cases:

1. Pre-define PostgreSQL server information for MetaDefender Core to connect and setup database: only required on Linux installation, or Windows with command line installation. Learn more details at:
    a. Standalone Mode > Command Line Installation (applicable on version 4.19.0 or newer).
    b. Shared Database Mode > Command Line Installation (only applicable on version 4.20.0 or newer).
2. Automating to accept EULA, create a default local admin user and import MetaDefender Core configurations upon installed: Those steps could be automated using ignition file described in this page.

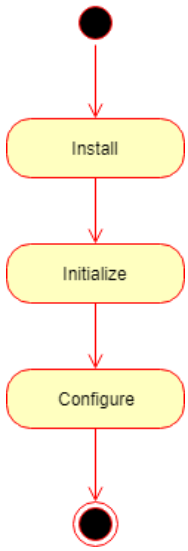
Ignition file location to create:

- **Windows**: `C:\OPSWAT\ometascan.conf`
- **Linux**: `/etc/opswat/ometascan.conf`

The product supports fully automated deployment. It means that it can be installed and configured with no human interaction.

The automated deployment can be split to three steps on a high level:

1. Installation
2. Initialization
3. Configuration



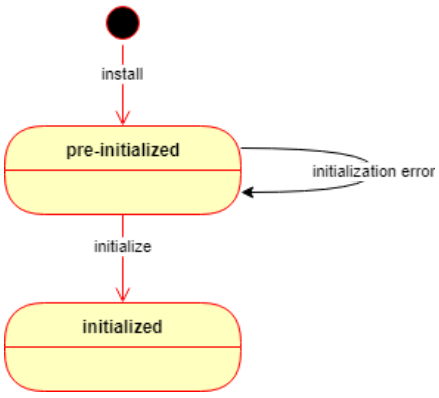
Installation

To automate the installation, install the product from the command line and provide the installation-time options as parameters to the installer. For further details see Command Line Installation

After the installation is complete, the product starts up and waits in a pre-initialized status. The product may be initialized in two ways:

1. Manually using the Wizard Setup or
2. Automatically using an *ignition file* (see below).

> **Warning**
> If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried after fixing the problem (e.g. placing the ignition file in its lookup location) and restarting the OPSWAT MetaDefender Core service.
>
> Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

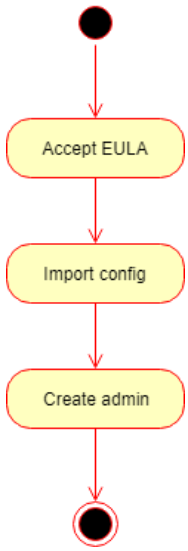


Initialization

Initialization is the process of bringing the product to an operable status.

Basically the initialization consists of the following steps:

1. Accept the End User License Agreement (EULA),
2. Import product configuration and
3. Create the first administrator user account.



Ignition file

The initialization process can be configured in a file called the *ignition file*.

> **Warning**
> In Linux, the ignition file and the folder /etc/opswat should have the permission 755

Ignition file location

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

The ignition file must be in conf format.

ometascan.conf bash

```
eula=true
[user]
name=admin
password=admin
email=admin@local
apikey=44c96cbb4e4e6c0a1d16bcb3c3da24def317
[config]
import=config_package.zip
import_password=123
[security]
session_cookie_samesite=Lax
[internal]
skip_startup_pages=true
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
private_username=my_internal_db_user
private_password=mypassword
[ocm]
ocm_url=https://my.opswat.com
reg_code=XXXXXXXX
```

Ignition file fields

The ignition file accepts the following fields:

Section	Key	Required	Accepted Values	Description
	eula	Optional (mandatory when having [user] section)	true false (default)	Whether to accept the End User License Agreement. This key must be set at the top of ignition file. This key must be set to "true" to accept the EULA. Any other value will cause the initialization to fail. When having [user] section defined, then this key must be set.
[user]		Optional		Initial administrator user account properties. The Administrator role is granted to the account.
	name	Optional	string	User name for the initial administrator user account.
	password	Optional	string	Password for the initial administrator user account. WARNING! Clear text password The password in this configuration file must be stored in its clear-text format and as so it may be visible for unauthorized parties.
	email	Optional	string	E-mail address for the initial administrator user account.
	apikey	Optional	string	API key for the initial administrator user account.
[config]		Optional		
	import	Optional	string	Path to a file in json/zip format that contains a previously exported configuration to be imported. Zip-format file contains a full configuration including user management settings, so it will overwrite [user] section.
	import_password	Optional	string	Password to decrypt .zip configuration package.
[security]		Optional		

Section	Key	Required	Accepted Values	Description
	session_cookie_samesite	Optional	Strict Lax (default) None	Declare if session cookie should be restricted to a first-party or same-site context.
[internal]		Optional		
	skip_startup_pages	Optional	true false (default)	Skip all welcome and other startup pages (e.g. upgrade result) when installing / upgrading MetaDefender, and going straight to the product index default page.
[dbserver]		Optional		
	type	Mandatory when having [dbserver] section	local remote	Where the PostgreSQL database server should physically locate. "remote" means PostgreSQL server and MetaDefender Core are not in the same machine.
	host	Mandatory when having [dbserver] section	string	IP address / domain name of the server where PostgreSQL server locates. "localhost" can be used when applicable.
	port	Mandatory when having [dbserver] section	number	Port of PostgreSQL server is listening for connections from clients (i.e. MetaDefender Core).
	user	Mandatory when having [dbserver] section	string	PostgreSQL server's user. SUPERUSER privilege is required for MetaDefender Core to setup its database and extensions for the first time. **Only non-Unicode characters supported.**

Section	Key	Required	Accepted Values	Description
	password	Mandatory when having [dbserver] section	string	PostgreSQL server's user credentials. **Only non-Unicode characters supported.**
	private_username	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
	private_password	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
[ocm]		Optional		
	ocm_url	Optional	string	URL to My OPSWAT server.
	reg_code	Optional	string	A registration code to enroll to My OPSWAT server.

Detailed initialization process

1) After the product has been started, it looks for the ignition file in the designated location.

Windows: C:\OPSWAT\ometascan.conf ** Linux: /etc/opswat/ometascan.conf

** 2) If an ignition file is found, then

2.1) It gets validated, and if it is valid, then based on the information found in the ignition file:

- The EULA is accepted,
- The configuration is imported,
- The administrator account is created.
- If any of the above steps fails, then the error is logged, and the initialization gets terminated.

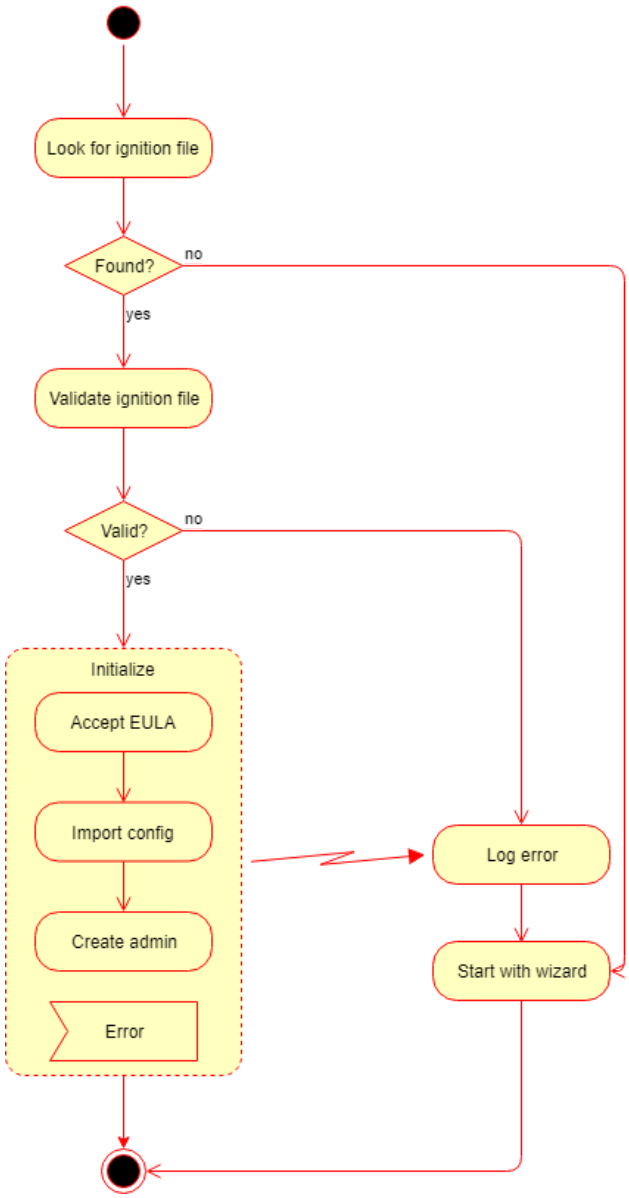
> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

2.2) If it is not valid, then the error is logged, and the initialization gets terminated.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

3) If there is no ignition file, then no initialization is performed.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.



Warning

If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried fixing the problem (e.g. placing the ignition file to its lookup location) and restarting the OPSWAT MetaDefender Core service.

Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

Configuration

After the initialization is complete, the product is ready with the default and the imported configuration.

This configuration can be later changed calling the configuration API functions. For further details about the API see Configuration related APIs API

CIS Level 1 Guidelines

Support Red Hat Enterprise Linux 9
For more details about Center for Internet Security (CIS) please refer to this document

Instruction steps

I. Install OpenSCAP

bash bash

```
yum install openscap-scanner scap-security-guide
```

II. Generate a result file and a HTML report using OpenSCAP scanner tool

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

III. Remediation of CIS Level 1 issues

Generate a remediation script based on the `ssg-rhel9-ds.xml` file:

bash

```
oscap xccdf generate fix --profile
xccdf_org.ssgproject.content_profile_cis_server_l1 --fix-type bash --output
remediations.sh /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

And execute remediation script.

bash

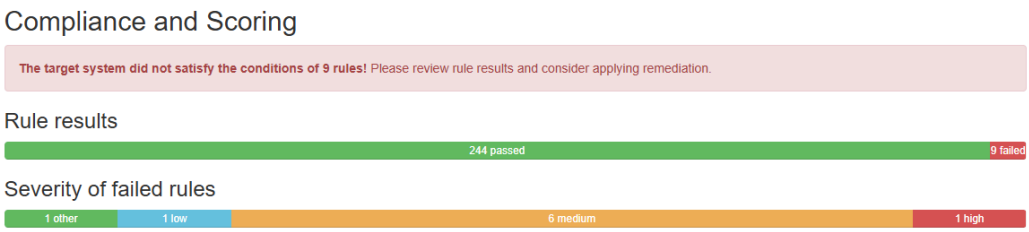
```
./remediations.sh
```

IV. Review the results after remediation

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

Example:



Notes

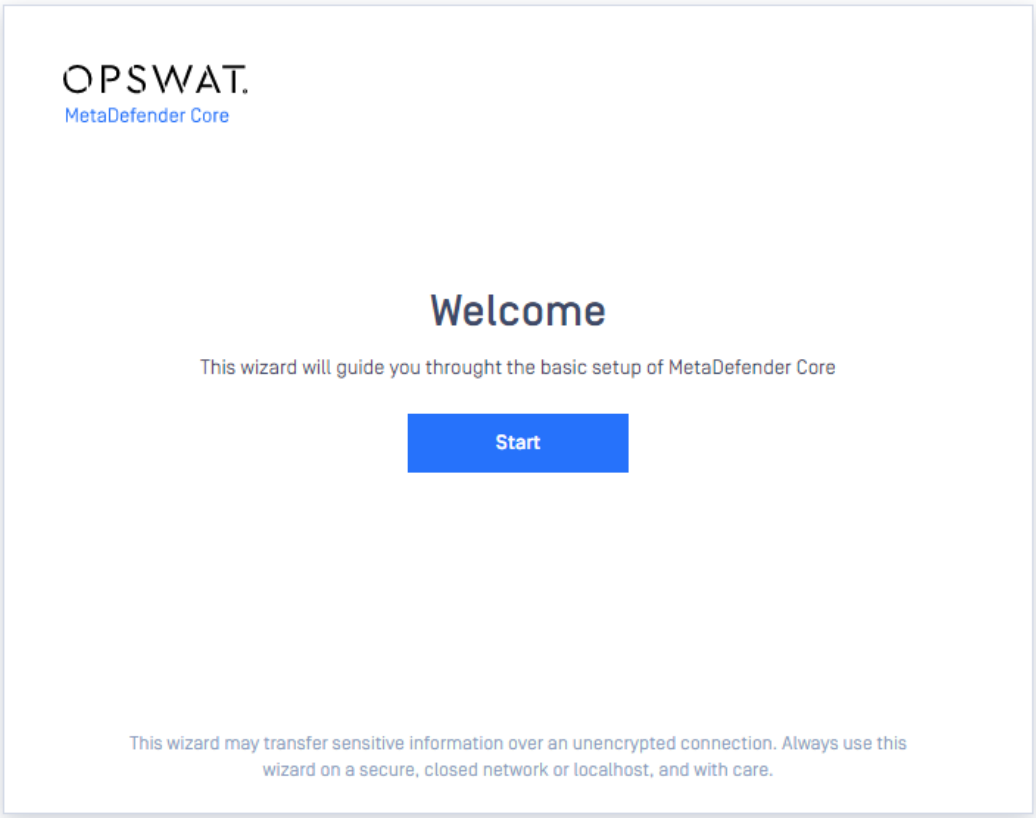
- CIS Level 1 requires `/tmp` folder to be mounted in a separate partition. Please ensure that that new partition have enough disk space for MetaDefender Core to run.
- CIS Level 1 requires that "Ensure No World-Writable Files Exist".
    - For now, when freshly installing MetaDefender Core, all its binary files meet the requirement.
    - When installing/updating engines, some engines might create additional files for its operation, and it might violate this requirement. In this case, you need to again execute the remediation script in the step III.

Wizard Setup

Introduction

Access the MetaDefender Core management console for the first time (i.e. http://localhost:8008), you are to complete a quick walkthrough for wizard setup in order to be able to use the product.

The administrative management console will be available only upon you have successfully finished this wizard stage.

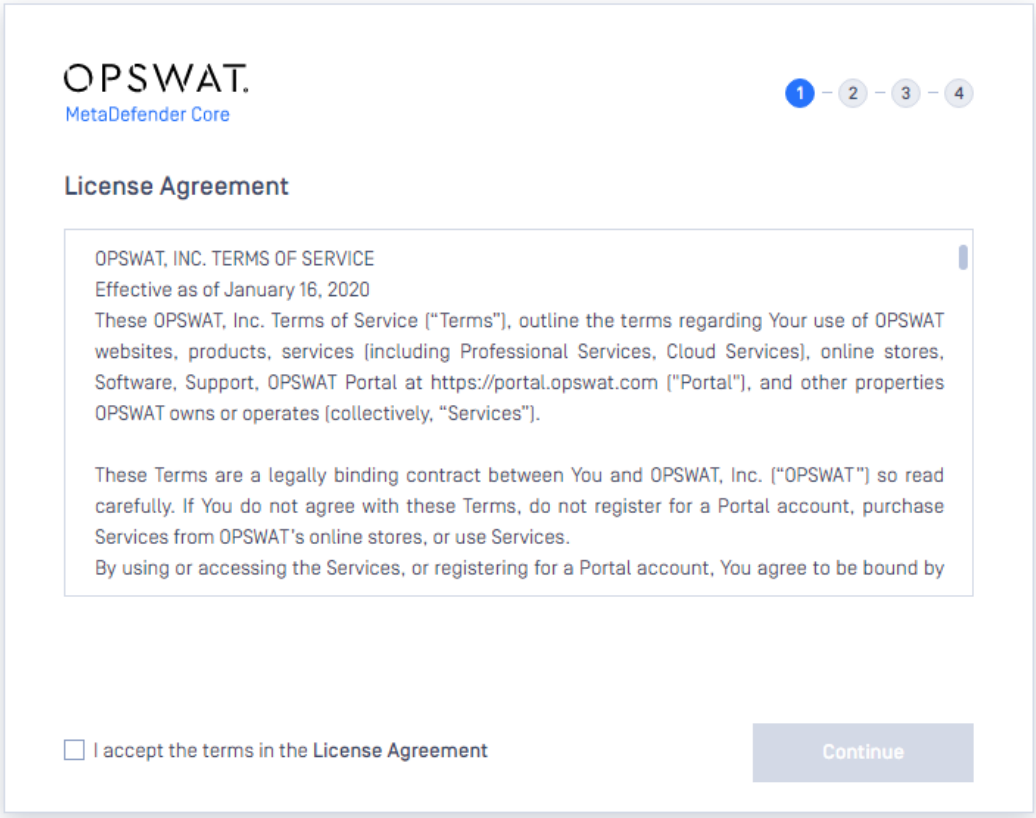


Warning

This wizard may transfer sensitive information over an unencrypted connection. Always use this wizard on a secure, closed network or localhost, and with care.

Basic configuration steps

End-User License Agreement



In the first page you can find the End-User License Agreement. You have to accept the terms before moving on.

Please read through the EULA (https://docs.opswat.com/policies/privacy-policy/portal-privacy-policy) carefully and if you agree with it, check `I accept the terms in the **License Agreement**` and click `Continue` to proceed.

Local Admin User Setup

The next step is to set up a default local administrator account.

This account will be the first one being able to access the administrative management console, and to create accounts for other users. You have to fill out all fields in this page to be able to move forward. When you are done, click NEXT to continue.

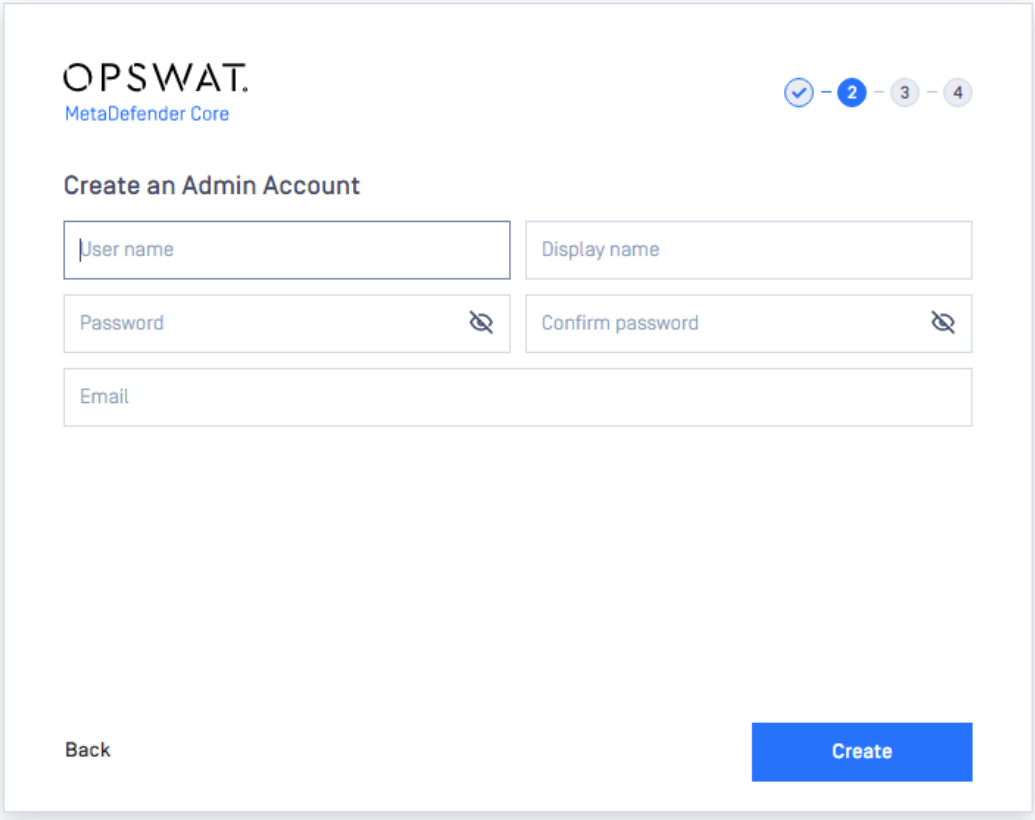
Info - User Directory

The administrator account, that is created via the basic configuration wizard, is always added to the LOCAL user directory as a member.

Learn more about user directory: User directories

The following information is required for the administrator account:

ACCOUNT NAME	The unique name of the account that is used at the time of login and in log messages for accountability.
ACCOUNT DISPLAY NAME	Name of the person bound to this account. This name (appended to the name of the account's user directory) is displayed in the top right corner of the administrative management console.
PASSWORD	Password of the user bound to this account that is used at the time of login. Passwords sent clear-text As long as TLS is not configured for the basic configuration wizard, passwords are sent clear-text over the network and may be disclosed to unauthorized parties. As a mitigation action: 1. Either use the wizard on *localhost* or on a direct network connection, or 2. Enabling HTTPS as soon as possible and change the password immediately if it has already been set.
EMAIL	Email address of the person bound to this account.



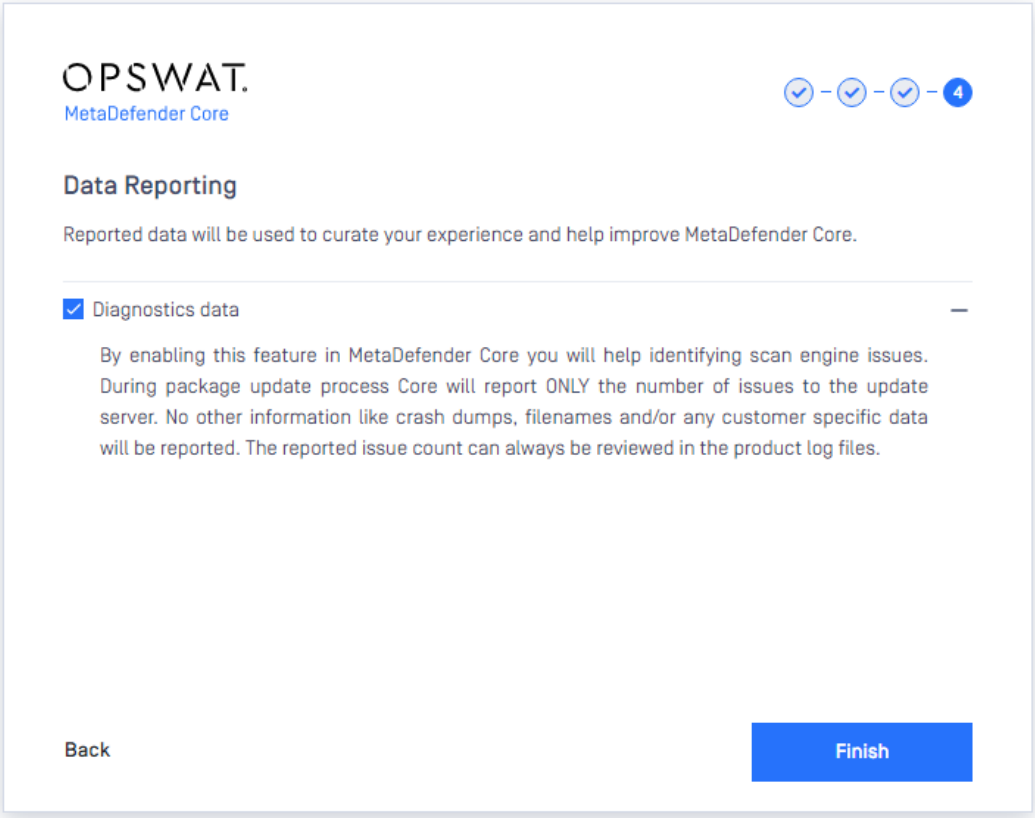
License activation

For license activation details see License Activation

Data Reporting

Enabling this option to allow MetaDefender Core collect its own engine issues and report back to OPSWAT for further analysis.

MetaDefender will not collect any other information.

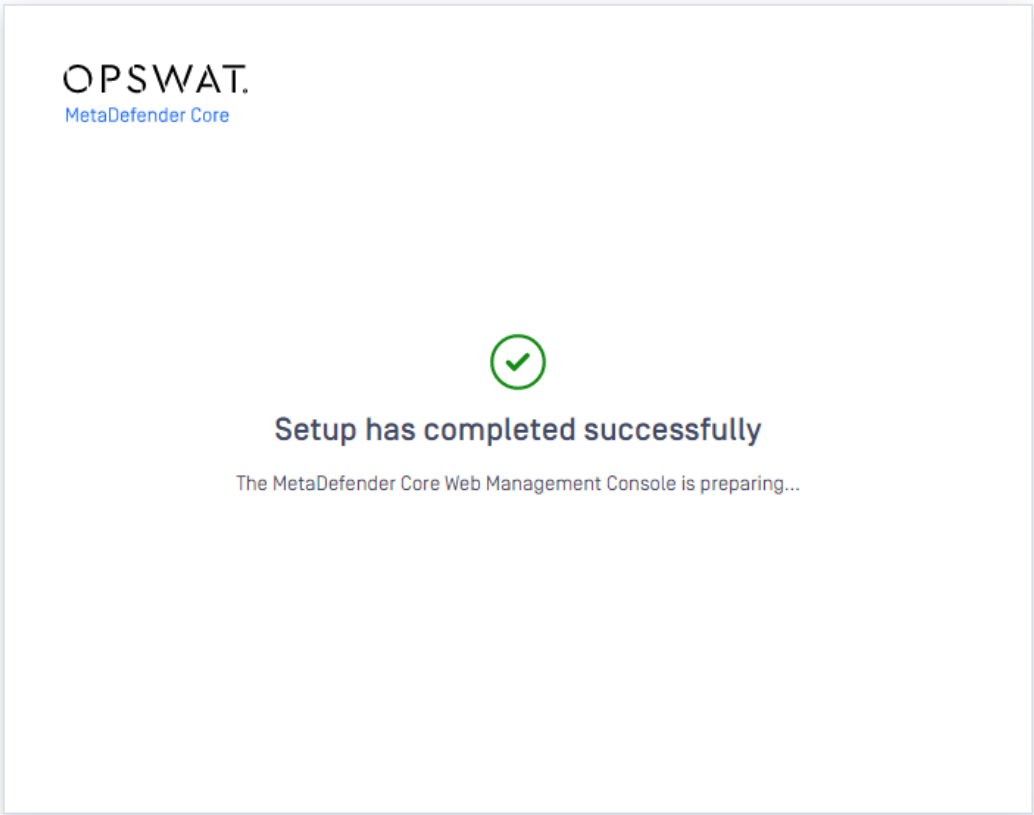


Wizard completion

After you have completed every steps you are ready to finish the wizard and start using the product. Click the `Fishish` button to complete the wizard.

> **Warning**
> The product's service will be restarted and the browser will be redirected to the administrative management console. This could take several seconds.

You can login to the Web Management Console with the administrator user that have just been created in the previous steps.



Setup HTTPS

Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network. Websites, like the Web Management Console, are able to use TLS to secure all communications between their servers and web browsers.

The TLS protocol aims primarily to provide confidentiality (privacy) and data integrity between two communicating computer applications.

Warning

No TLS for the wizard

By default, TLS is not enabled for the basic configuration wizard. As a consequence sessions between the wizard's backend and the browser may be insecure.

Info

Performing the same steps as for the Web Management Console, it is possible to set up TLS for the basic configuration wizard. Remember completing the TLS setup before launching the wizard.

For instructions to set up TLS see Enabling HTTPS

License Activation

##

To activate your installation go to the **Settings > License** menu in the Web Management Console. If you have no valid license, you will only see your installation's **deployment ID**. You will also see a warning in the Web Management Console header.

After successful activation, the product will start downloading the latest available scan engines and malware databases. You can follow the status of the scan engine installation on the **Inventory > Modules** page.

When all the licensed engines are installed, you can start using the MetaDefender Core to process files - File Processing.

Deployment ID

Each MetaDefender Core instance represented by an unique system fingerprint called `Deployment ID`

MetaDefender Core licensing is purely based on deployment ID activated. Therefore, everytime your MetaDefender Core is activated with a different deployment ID, then it will be captured as a different activation record accumulated into your total activation times allowed.

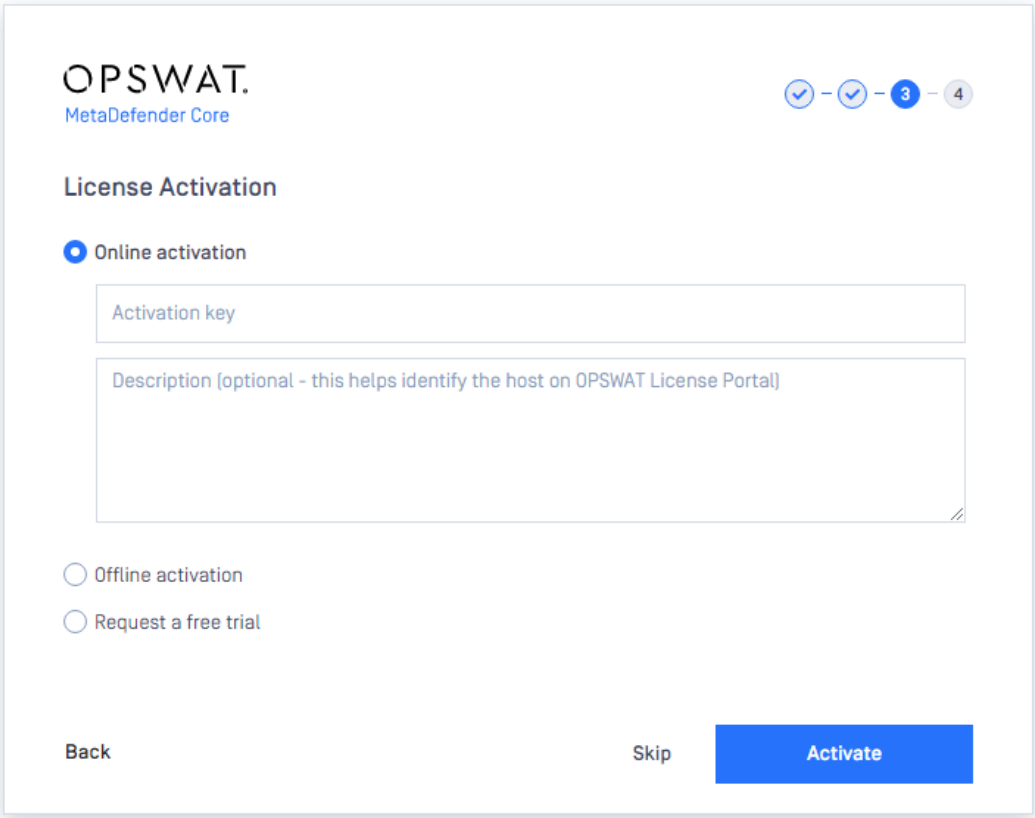
> **Notice**
> Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you alter the system where MetaDefender Core relies on (e.g. network card, OS hostname).
>
> Deployment ID is an UUID version 4 random based string. Learn more at: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you update the system where MetaDefender Core relies on.

Activation Modes

There are three different options for the product activation:

- Online activation: MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.
- Offline activation: Requiring you to upload a license file (.yml). That license file could be retrieved by activating your license via OPSWAT Portal. Follow the displayed instructions for details.
- Request trial key online: For evaluation purpose, you can receive an trial Activation key via email. Follow the displayed instructions for details.



Online activation

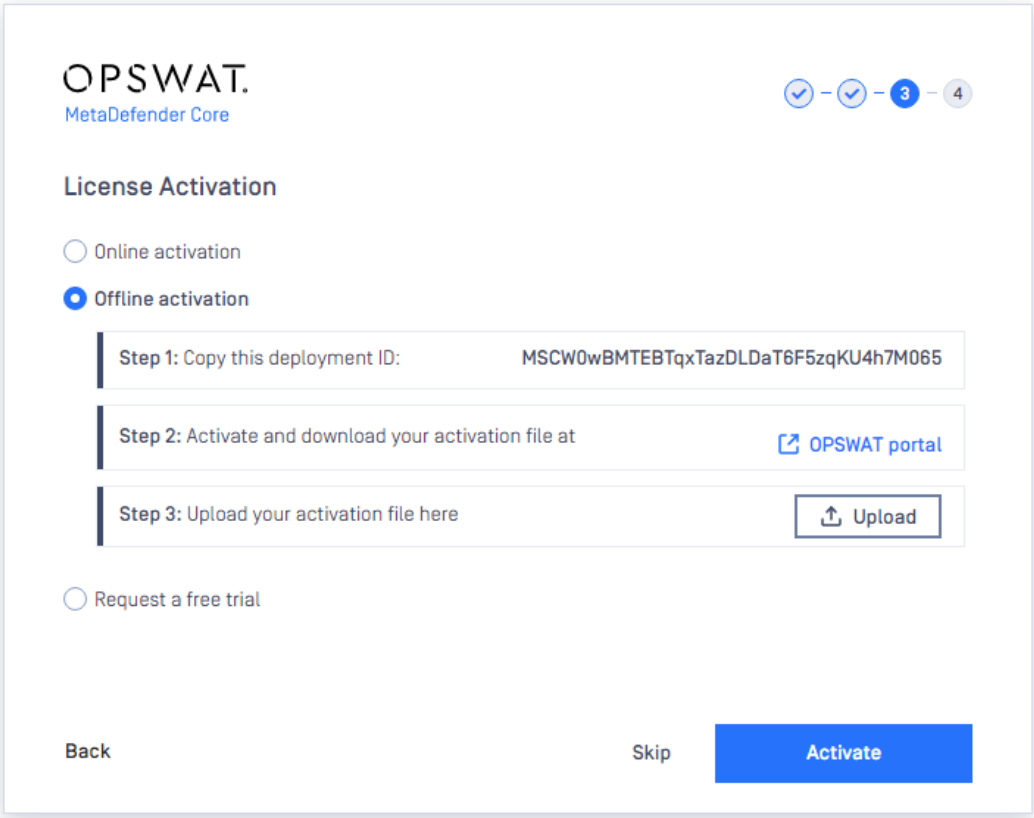
MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.

Required to input your Activation License key, and optional for description (but we recommend this for troubleshooting and auditing purpose if needed).

> **Warning**
> If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts' for license activation:
>
> - https://activation.dl.opswat.com Note: *IP address-based whitelisting on your firewall might fail after some time since OPSWAT uses CDN (Content Delivery Network) to faster delivery updates over the world, and IP address of edge servers might change over time.*

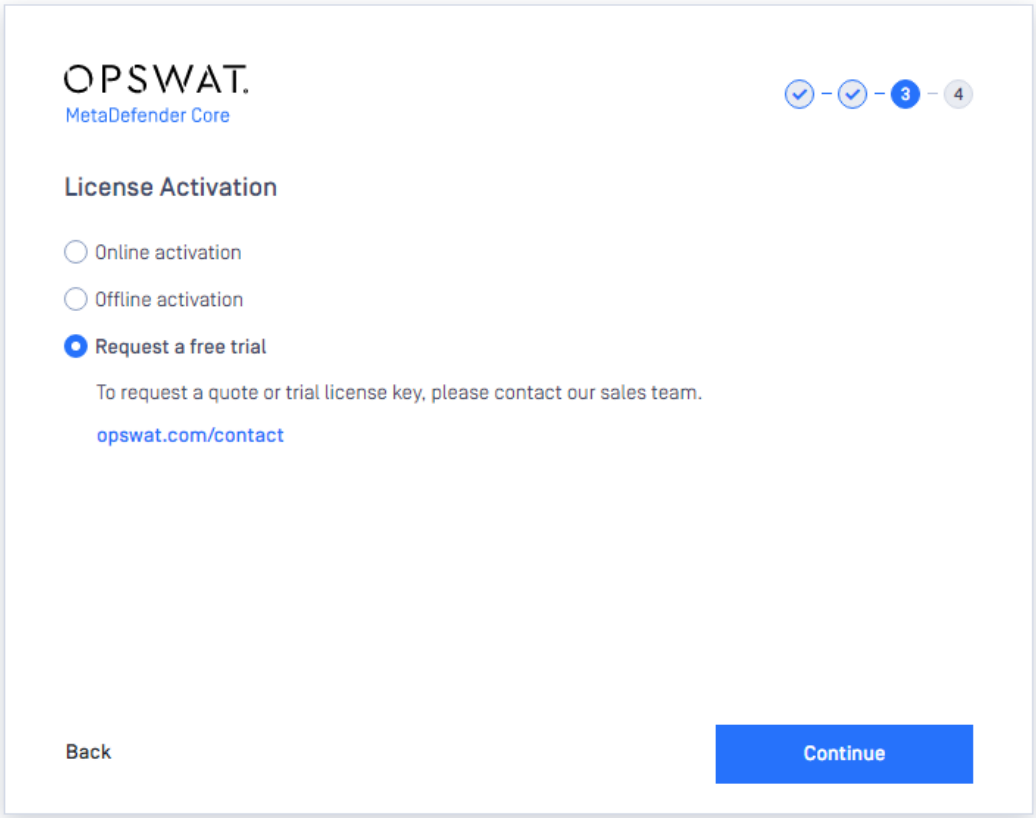
Offline activation

With no internet connection on the server the Metadefender v4 instance may be activated indirectly from a different machine, that has internet connection. The Deployment ID of the Metadefender v4 instance and the the Activation key received at the time of purchasing the product will be required. Follow the steps on the screen to activate the product offline.



Request trial key online

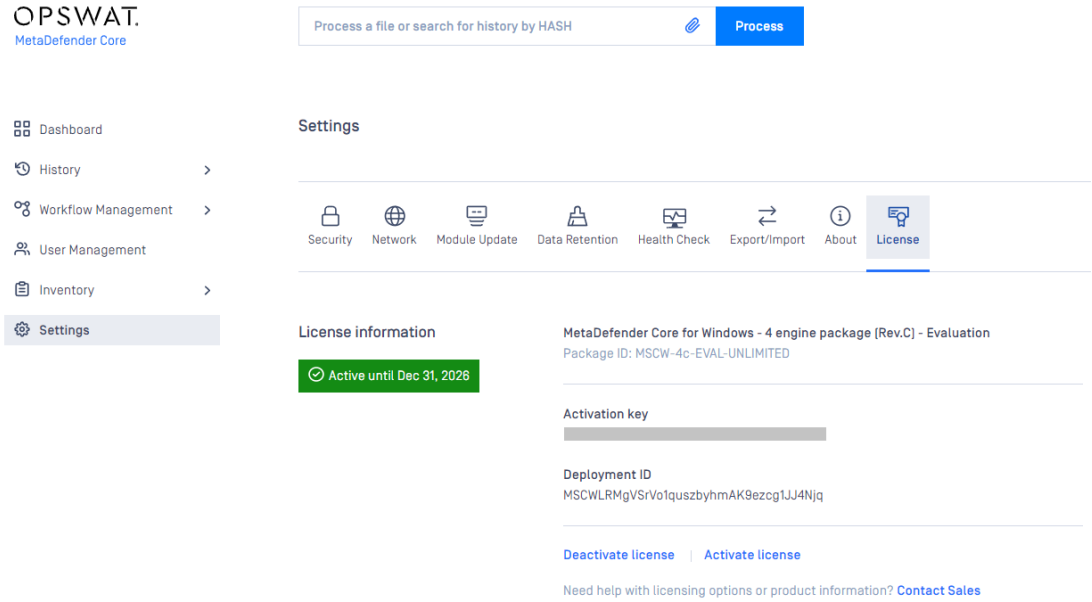
An evaluation license could be acquired by contacting our sales team.



Check Your License Details

For more license details and activating your MetaDefender Core, go to Settings > License menu tab on the Web Management Console:

- Product name: product name as on your order (e.g. MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation)
- Package ID: product identification as on your order (e.g. MSCW-4c-EVAL-UNLIMITED)
- Expiration: last day of license validity
- Activation key: key used for activating product
- Deployment ID: identification of this installation


OPSWAT.
MetaDefender Core
Process a file or search for history by HASH
Process
Dashboard
History
Workflow Management
User Management
Inventory
Settings
Settings
Security
Network
Module Update
Data Retention
Health Check
Export/Import
About
License
License information
Active until Dec 31, 2026
MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation
Package ID: MSCW-4c-EVAL-UNLIMITED
Activation key
Deployment ID
MSCWLRMgVSrVo1quszbyhmAK9ezcg1JJ4Njq
Deactivate license
Activate license
Need help with licensing options or product information? Contact Sales

Scripted license management

Using REST API calls there is an option for scripted activation and deactivation for MetaDefender Core servers with **no Internet connection or if your infrastructure is using automation to create/destroy MetaDefender Core instances**. In this chapter the steps of these two scenarios are described.

Requirements

- an installed MetaDefender Core instance without Internet connection
- another computer that has Internet connection and can run your activation scripts
- a manual or automated way to transfer data between the two computers

Activation steps

1. In order to activate MetaDefender Core, deployment ID and activation key are needed to generate the license file.
    a. Activation key should be purchased from OPSWAT.
    b. MetaDefender Core deployment ID can be queried by using REST API (for details see GET Get Current License Information API page)

> **Info**
> Save this deployment ID in your system, you might need this ID when the instance is unavailable at the time of deactivation.

2. Activate license and get the license file using the following URL, where <quantity> is 1.

activation call bash

```
curl -X GET https://activation.dl.opswat.com/activation?key=<activation-key>&deployment=<deployment unique-ID>&quantity=<quantity>
```

If the activation is successfully, a license file is downloaded. Save this file.

HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	license file	Activation was successfully.
200 Activation failed	error: '<user conform error message>'	Failed activation
200 Invalid parameter	error: 'Could not activate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not activate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
200 Invalid parameter	error: 'Could not activate your product because the quantity you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key, quantity or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Info
In case of any activation issue, contact OPSWAT support for help

3. Upload license file to MetaDefender Core: The license file should be uploaded to the MetaDefender Core v4 to activate the product. For details see page: POST Upload license key API

Deactivation steps

1. For deactivation, following infomation is required:
    a. Activation key
    b. MetaDefender Core deployment ID: can be queried by using REST API (for details see GET Get Current License Information API page)
2. Deactivate license using the following URL:

deactivation call bash

```
curl -X GET https://activation.dl.opswat.com/deactivation?key=<activation-
key>&deployment=<deployment unique-ID>
```

Response HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	result: ok	Successful deactivation
200 No active license found	error: 'Could not found any active license with the given parameters'	The license has not been activated yet or it has been deactivated already.
200 Invalid parameter	error: 'Could not deactivate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not deactivate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Important notes

Warning

Product activation is tied to several hardware, operating system and software parameters. In case of one or more major hardware or software parameter change the product might turn into deactivated status. Operating system updates, other software updates should not affecting the activation status.

If the product is online activated and has live Internet connection then in case of deployment ID change the product reactivates itself. If the product doesn't have Internet connection then the administrator is responsible to reactivate the product.

Because of the above administrator should consider licensing restrictions of the offline deployments if using any virtualization and/or containerization technologies.

File Processing

MetaDefender Core provides REST API interface for file processing business.

There are several ways to process files with MetaDefender Core:

- Analyze Files via Web Interface
- Process Files via REST API

Product Upgrading

Info
Simply run installer file, details could be found at Installation

Info
For command line mode, upgrading (e.g. on **Linux**), prior to running the installer file, please make sure to **create the ignition file** with PostgreSQL server information included. Details:

- Standalone DB mode: Command Line Installation (Windows and Linux)
- Shared DB mode: Command Line Installation

Recommendation for MetaDefender Core upgrade

Our goal is to help you execute the product upgrade continuously, efficiently, and correctly.

PostgreSQL service upgrade

- **Bundled PostgreSQL Service**: If you are using the bundled PostgreSQL server included in the installer, the PostgreSQL service will be automatically upgraded to the newer version (if available). No additional action is required for this process.
- **Remote PostgreSQL Service**: For remote PostgreSQL servers, customers are responsible for managing and upgrading the service independently. Upgrading MetaDefender Core does not update the remote PostgreSQL service.

Before the upgrade

Prepare a proper disaster recovery plan (DRP) including some common practices:

- Backup, or take a snapshot of your current working MetaDefender Core server.
  - This is to help quickly rollback for any upgrade failure might cause later for any reason.
- Optionally backup your PostgreSQL database, especially when using remote shared database mode.
  - Reference: https://www.postgresql.org/docs/current/backup.html
  - Please do it after stopping all linked MetaDefender Core instances to avoid potential data collision.
- Lastly, if backing up entire server is not feasible, then at least export MetaDefender Core configuration package Import/Export configuration just in case you'll need to start off a new server with current MetaDefender Core configuration to keep the scanning service running without any impact.

On the PostgreSQL server, double check server resource to ensure sufficient RAM memory and disk space to be preserved for the upgrade process.

- 8GB RAM available at least during the upgrade.

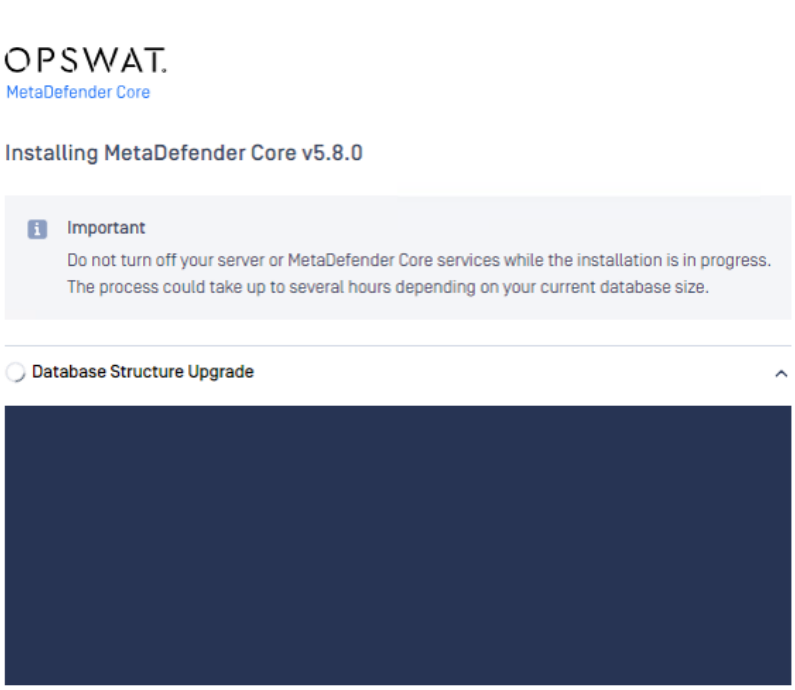
- Disk availability is greater than MetaDefender Core database size.

While doing the upgrade

Applicable for shared database deployment:

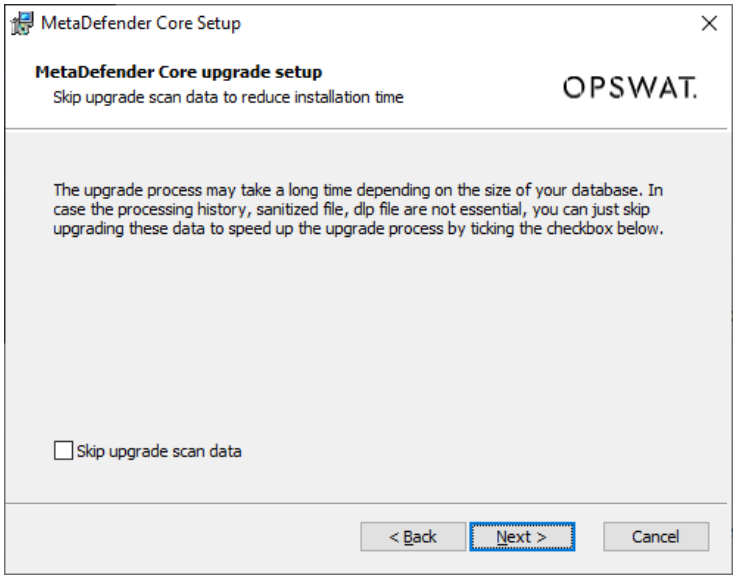
- Stop all MetaDefender Core instances' scanning service before performing the upgrade.
- Upgrade one (1) MetaDefender Core instance at a time.
- Proceed to upgrade MetaDefender Core instance where the bundle local PostgreSQL server is used for shared database setup.
- Otherwise is using remote PostgreSQL then the upgrade order does not matter.

For large size database, mostly for scan history, expecting long time to be spent on the upgrade process.



The product supports an option to skip data migration for scan history to expedite the process drastically and mitigate any failure risk. You can manually backup all data from your PostgreSQL server for offline usage if any.

Check "Skip upgrade scan data" to enable this feature while doing the upgrade.



or add following key in the ignition file:

none

```
[internal]
skip_upgrade_scan_data = 0
```

Troubleshooting

If the upgrade process ever runs into an error:

- Double check your network and services, applicable to remote PostgreSQL server.
- Optionally configure MetaDefender Core to skip migrating scan history.
  - Windows
    - Open Windows registry, and navigate to HKLM\Software\OPSWAT\Metascan
    - Add a new key `internal/skip_upgrade_scan_data` with value = 1
  - Linux
    - Open MetaDefender Core configuration file `/etc/ometascan/ometascan.conf`
    - Add new key into the file
      - [internal]
      - skip_upgrade_scan_data=1
  - Note: This key will be auto removed upon the upgrade finishes.
  - Then restart MetaDefender Core service to retry the upgrade.
    - Might retry this a few times until the upgrade is done.

Just in case the upgrade process still fails, please contact OPSWAT support with the support package - How to Create Support Package? This will help us analyze the root cause and figure out an optimal solution for your system failure.

Using external Load Balancer

Application Load Balancing (Layer 7)

This page provides information about recommended ways to use sessions with a Layer 7 load-balancer.

Most load-balancers have the ability to provide cookies for pinning subsequent traffic from a client to the appropriate server. This method is called sticky session, session persistence or session affinity. Using cookies efficiently requires the client to know when a cookie should be sent, should not be sent or should be deleted.

Sticky session load balancing

Each Metadefender Core instance has its own databases and application sessions that cannot be seen by other instances. Therefore, in order to get the related data/response to our queries we should ask the appropriate Metadefender Core v5 server. On the other hand, to keep the advantages of the used load balancing method, cookies should not be sent if it is not necessary.

Single file scanning

Step	Stage	Task	Cookie usage
1.	Sending file	Initiate processing a file on the client side. Send file through the load-balancer. (See POST File Analysis API)	Cookie should not be sent
2.	Sending file	Save the cookie and the data_id you got from the load-balancer	Save cookie
3.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Send cookie saved in step 2
4.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Send cookie saved in step 2

Batch scanning

Step	Stage	Task	Cookie usage
1.	Open batch	Initiate processing file(s) in batch. Request a batch ID through the load-balancer. (See POST Initiate Batch API)	Cookie should not be sent
2.	Open batch	Save the cookie and the batch_id you got from a Core server through the load-balancer.	Save cookie
3.	Sending files	Send file through the load-balancer. (See POST File Analysis (using batch header) API)	Send cookie saved in step 2
4.	Sending files	Save the data_id you got from the load-balancer.	-
	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Send cookie saved in step 2
5.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
6.	Getting batch status	See GET Batch Status API	Send cookie saved in step 2
7.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Send cookie saved in step 2
8.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API	Send cookie saved in step 2

If it does not matter which upstream server responds, the queries should be sent without cookie.

> **Info**
> It is recommended not to send cookies when it's not necessary to allow load-balancer to use its own method to share the load between Metadefender Core v5 servers.

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration through load-balancer

OPSWAT products that support HTTP load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	4.3.4	-
OPSWAT Client	Windows: 7.6.247.0 Mac: 10.4.243.0	-
MetaDefender Email Security	4.3.0	-
MetaDefender ICAP Server	4.3.0	-
MetaDefender Vault	1.3.0	-
MetaDefender for Secure Storage	2.0.7	-

DNS Load Balancing

Using this method is logically similar to Layer 7 load-balancing.

Briefly how it works

Client uses a domain name to send a query to a server. Client's DNS server has more "A" records for that name with different IPs. When a client resolves the server's domain name DNS server randomly chooses an IP for that name to send back. When a session is used on the application layer, client should know the IP address of the Core server that handles that specific session. In every other case, client should resolve the domain name with DNS query to let requests to be balanced between Metadefender Core servers.

Single file scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Sending file	Initiate processing a file on the client side. (See POST File Analysis API)	Use IP saved in step 2
4.	Sending file	Save the data_id got from the Core server	-
5.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Use IP saved in step 2
6.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Use IP saved in step 2

Batch scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Open batch	Initiate processing file(s) in batch. Request a batch ID. (See POST Initiate Batch API)	Use IP saved in step 2
4.	Sending files	Send file to the specific Core server with the basch ID saved in step 3. (See POST File Analysis (using batch header) API)	Use IP saved in step 2
5.	Sending files	Save the data_id you got from the Core server.	-
6.	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Use IP saved in step 2
7.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
8.	Getting batch status	See GET Batch Status API	Use IP saved in step 2
9.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Use IP saved in step 2
10.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API).	Use IP saved in step 2

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration via DNS load-balancing

OPSWAT products that support DNS load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	does not support yet	-
MetaDefender Client	does not support yet	-
MetaDefender Email Security	does not support yet	-
MetaDefender ICAP Server	does not support yet	-
MetaDefender Vault	does not support yet	-

SSL connection for PostgreSQL communication

> **Note**
> The SSL connection configuration is possible thanks to the support made by PostgreSQL. For reference: https://www.postgresql.org/docs/14/ssl-tcp.html
>
> MetaDefender Core is tested against this mode on particular version 4.21.1

Step 1: Generate SSL certificate for PostgreSQL server

> **Note**
> Following instructions are for self-sign certificate. For production environment, making sure to use the certificate signed by your trusted CA.

1) Generate a private key

bash

```
openssl genrsa -des3 -out server.key 1024
```

Sample output:

bash

```
Generating RSA private key, 1024 bit long modulus ........++++++
...............++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

Type your desired password / pass phrase to encrypt the private key

2) Remove the pass phrase to automatically start up the PostgreSQL server

bash

```
openssl rsa -in server.key -out server.key
```

> **Note**
> On Linux, make sure to set permission on the server.key file

bash

```
chmod og-rwx server.key
```

3) Create a self-signed certificate

bash

```
openssl req -new -key server.key -days 3650 -out server.crt -x509
```

You will be prompted to enter detailed information which is incorporated into your self-signed certificate request.

none

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:CA
Locality Name (eg, city) [Default City]:San Francisco
Organization Name (eg, company) [Default Company Ltd]:OPSWAT
Unit Name (eg, section) []:OPSWAT
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:dummy_email@opswat.com
```

For self-signed certificate, use the server ceriticate as the trusted root certificate:

bash

```
cp server.crt root.crt
```

Copy **server.key**, **server.crt**, **root.crt** to PostgreSQL data folder. For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data**
- Linux: **/var/lib/ometascan/pg_data/**

Step 2: Configure PostgreSQL server for SSL authentication connection

1) Create a custom config file (e.g. ssl.conf) for PostgreSQL.

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\postgres\ssl.conf**
- Linux: **/var/lib/ometascan/postgres/ssl.conf**

2) Content of ssl.conf:

none

```
ssl = on
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
```

3) Modify pg_hba.conf

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data\pg_hba.conf**
- Linux: **/var/lib/ometascan/pg_data/pg_hba.conf**

Add following (modify **::1/128** if needed to match with your environment requirement)

none

```
hostssl all all      ::1/128      scram-sha-256
```

After the modify, the content of pg_hba.conf should be like this (example):

> **Note**
> In a Linux environment, a local connection is required to upgrade from version 5.5.1 or earlier to version 5.6.0 or later.

none

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local         DATABASE  USER  METHOD  [OPTIONS]
# host          DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl     DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostgssenc    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnogssenc  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
local    all all                    scram-sha-256
hostssl all all      ::1/128      scram-sha-256

#More content.......
```

4) Reload the services

- Standalone DB mode: Restart **ometascan** service (MetaDefender Core service)
- Shared DB mode: Restart **ometascan-pg** / **ometascan-postgresql** service (MetaDefender Core PostgreSQL service) and then **ometascan** service (MetaDefender Core service)

Test your SSL database connection:

bash

```
psql.exe "sslmode=require dbname=postgres user=postgres host=localhost port=5432"
Password for user postgres:
psql (12.7)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
Type "help" for help.
```

This message indicate that you are now using SSL connection:

bash

```
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
```

Performance and Load Estimation

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Some factors that affect performance

- MetaDefender Core version
- MetaDefender Core engine package and configuration
  - set of engines (which and how many)
  - product configuration (e.g., thread pool size)
- System environment
  - server profile (CPU, RAM, hard disk)
  - client application location - remote or local
  - system caching and engine level caching
- Dataset
  - encrypted or decrypted
  - file types
    - different file types (e.g., document, image, executable)
    - archive file or compound document format files
  - file size
  - bad or unknown (assume to be clean)
- Performance tool

How test results are calculated

Performance (mainly scanning speed) is measured by throughput rather than unit speed. For example, if it takes 10 seconds to process 1 file, and it also takes 10 seconds to process 10 files, then performance is quantified as 1 second per file, rather than 10 seconds.

- total time / total number of files processed: 10 seconds / 10 files = 1 second / file.

Standalone DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.2.416.0

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

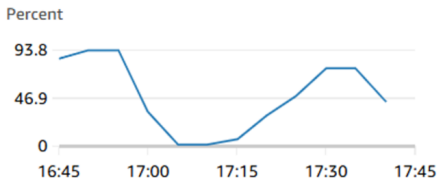
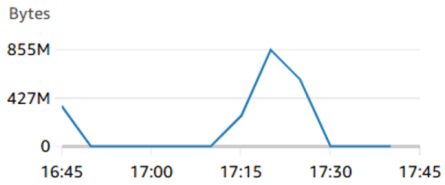
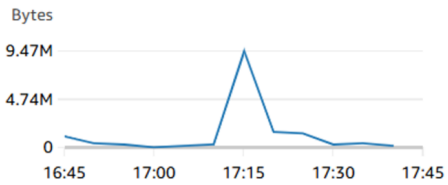
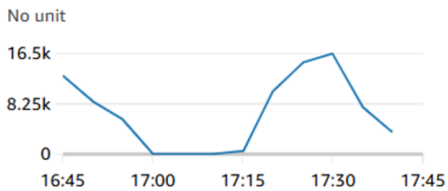
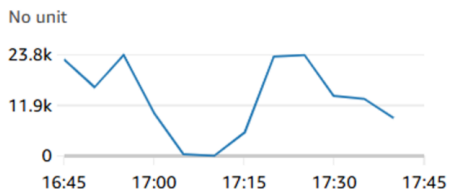
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV)	24.2	47.2 / 80	37.3 / 41.1		315,310	0.0114
Deep CDR	19.9	48.9 / 81.8	32.6 / 37		383,442	0.0094
Proactive DLP	17.1	36.8 / 58.2	30.4 / 33.6		446,228	0.0081
Vulnerability	10.1	59.2 / 85	29.3 / 33.3		755,495	0.0048

MetaDefender Core with common engine packages:

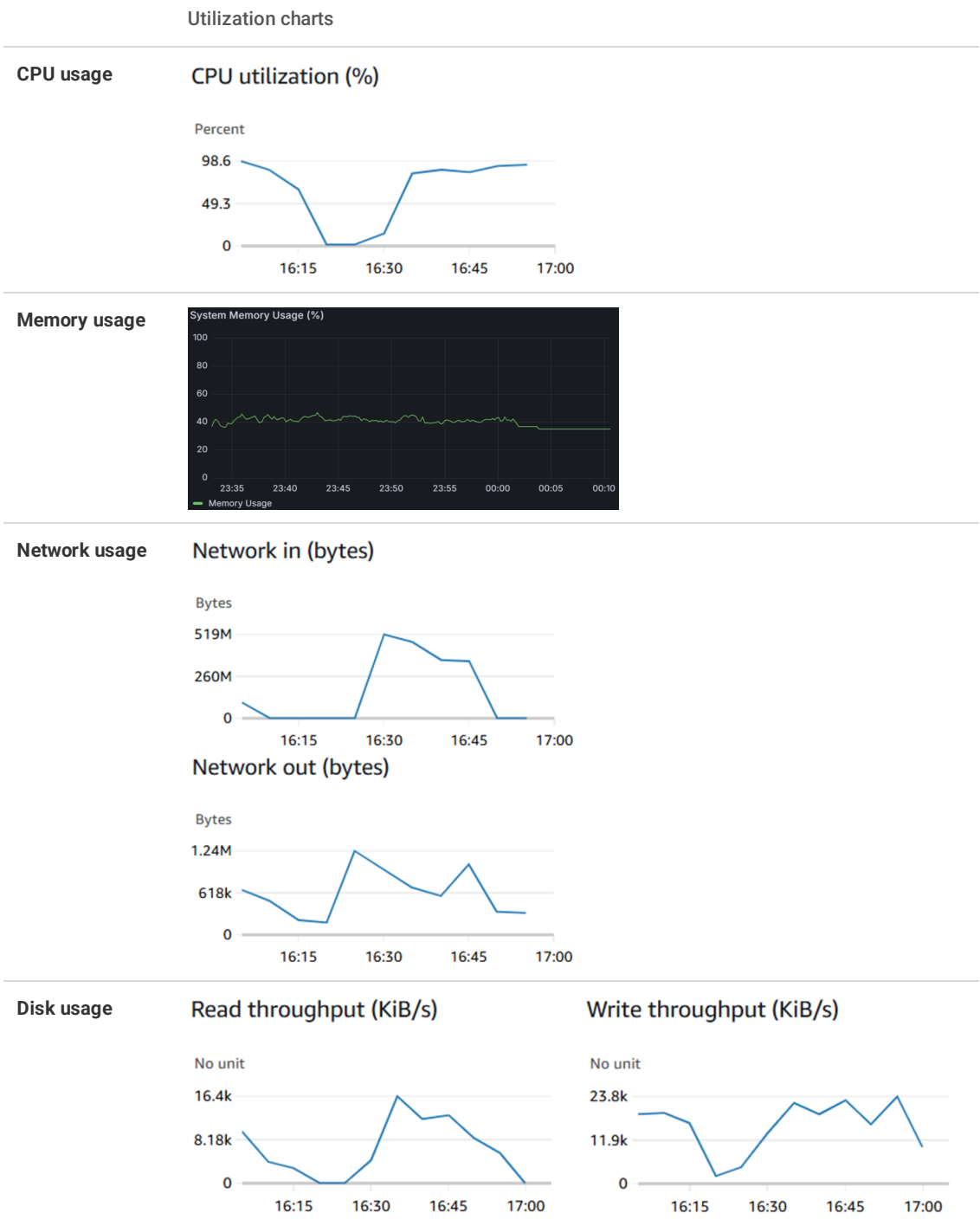
Use case	Scan duration	Avg./Max CPU usage	Avg./Max RAM usage	Avg./Max Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV) + Deep CDR	30.8	68.1 / 88	40.1 / 46.8		247,744	0.0145
Metascan (8 AV) + Deep CDR + Proactive DLP	31.2	74.2 / 98.6	41.5 / 51.1		244,567.3	0.0147
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	31.8	73.5 / 98	40.6 / 51.5		239,953	0.015

System resource utilization charts

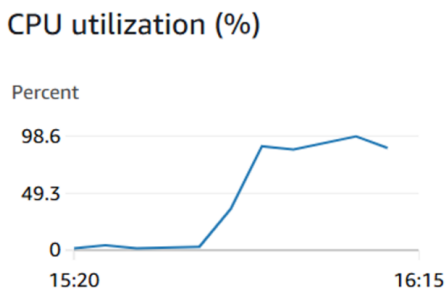
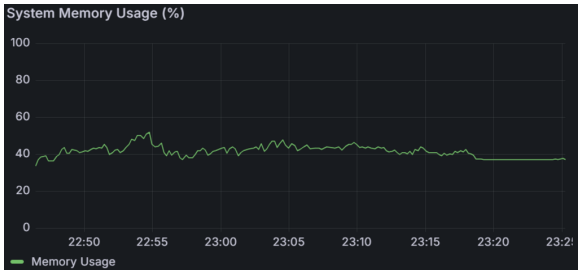
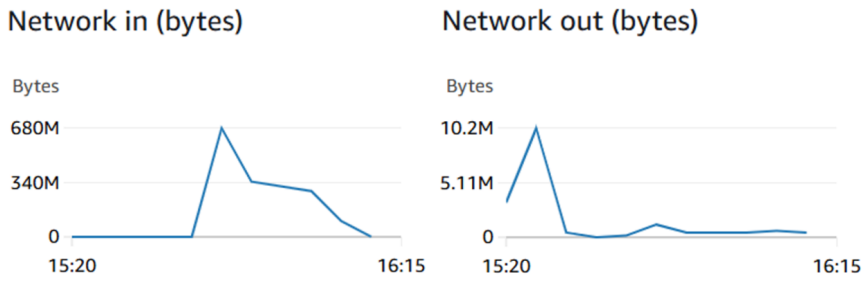
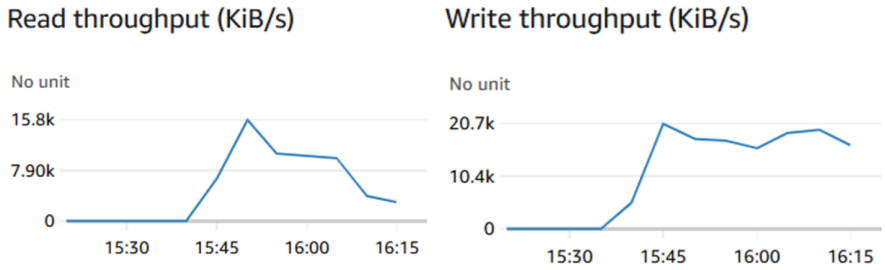
MetaDefender Core with Metascan (8 AV engines):

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

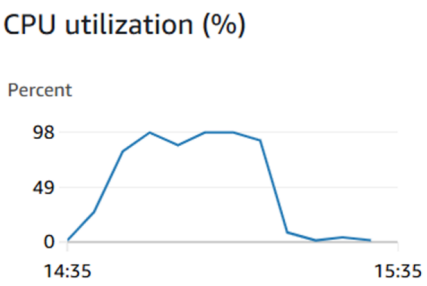
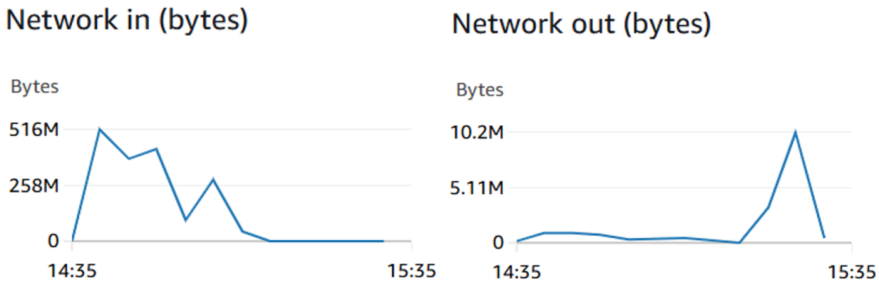
MetaDefender Core with Metascan (8 AV engines) + Deep CDR:



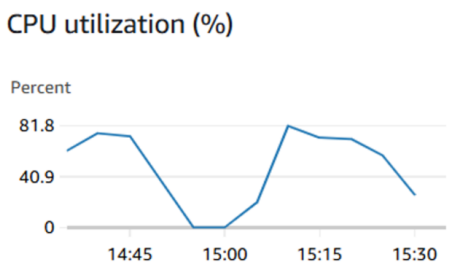
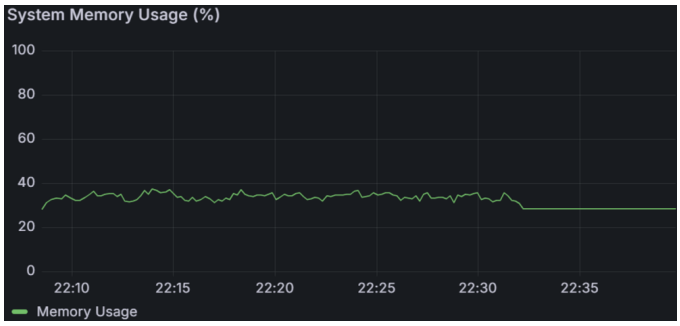
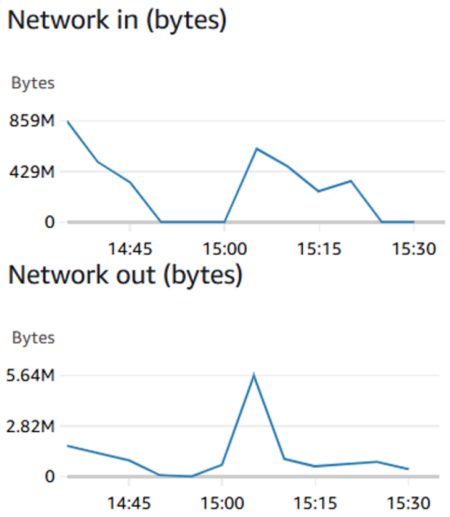
MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

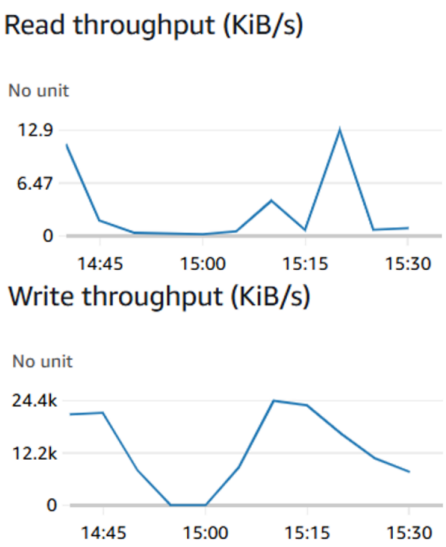
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

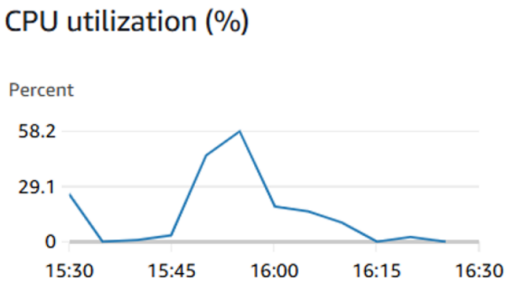
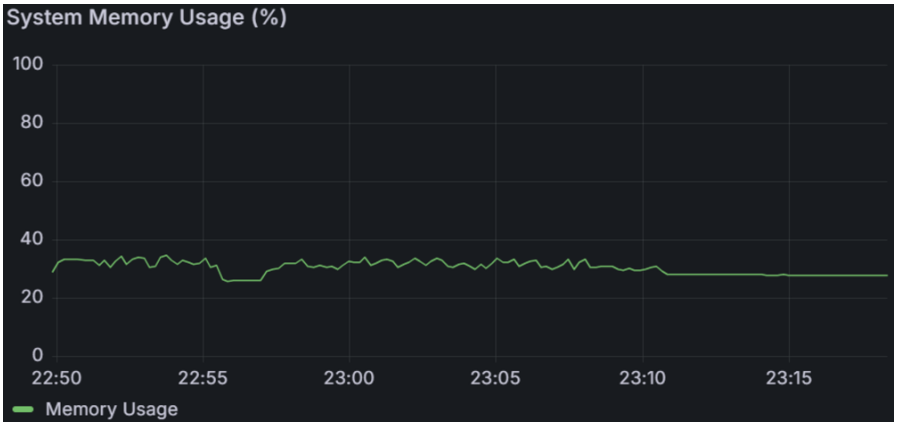
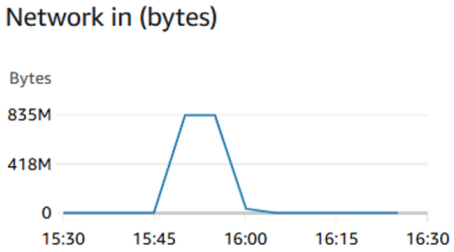
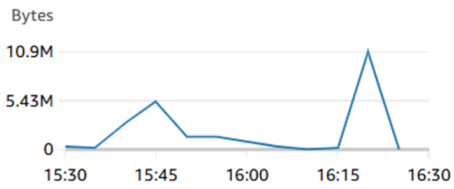
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

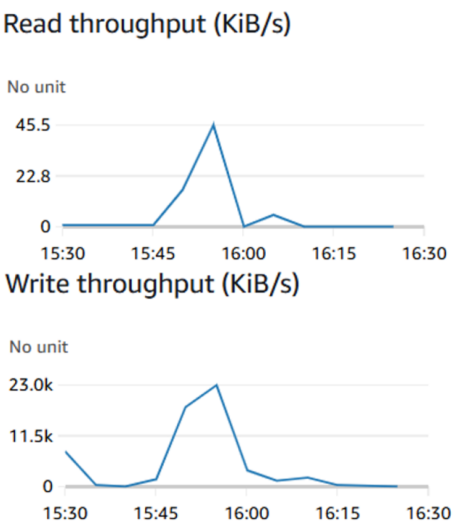
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

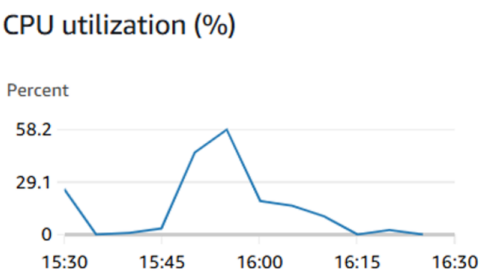
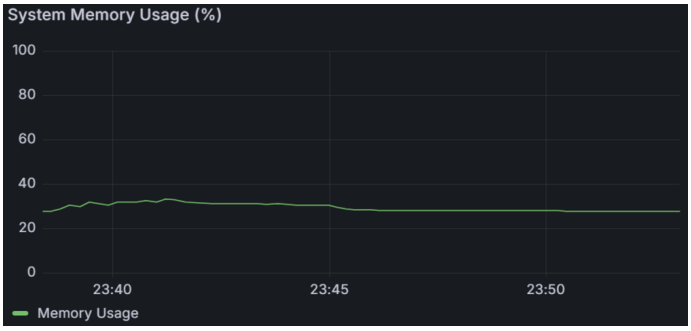
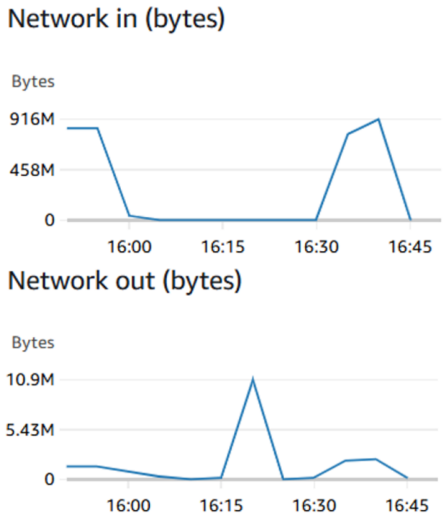
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	 

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability:

	Utilization charts
CPU usage	
Memory usage	
Network usage	

Utilization charts

Disk usage



Standalone DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Ubuntu 22.04	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.57-236

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

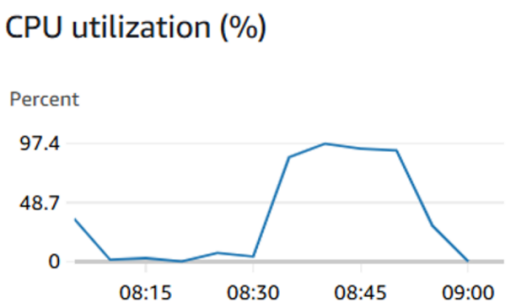
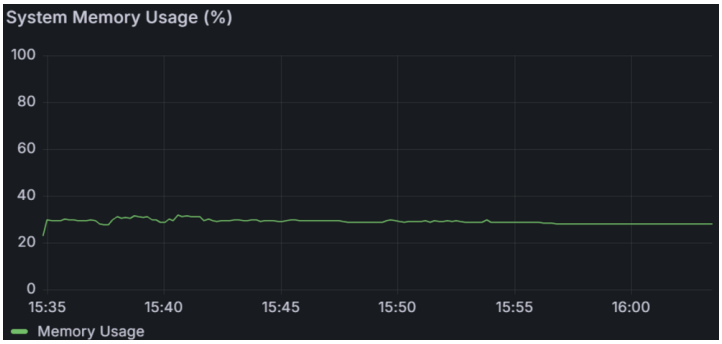
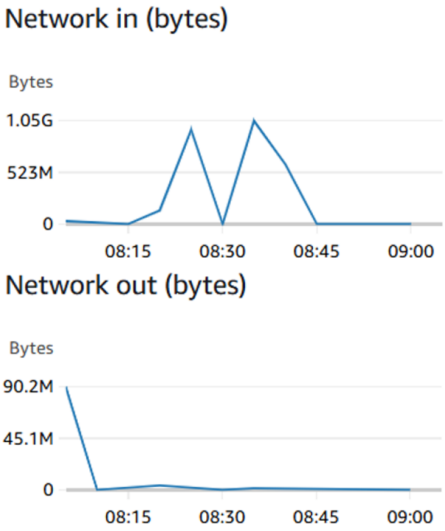
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV)	30.2	69 / 98	29 / 32		252,666	0.0142
Deep CDR	19.4	48 / 62	36 / 49		393,325	0.009
Proactive DLP	16.5	34 / 39	32 / 48		462,455	0.008
Vulnerability	12	50 / 65	46 / 46		635,875	0.006

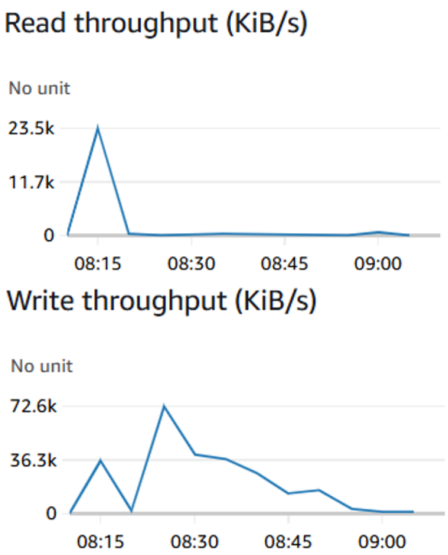
MetaDefender Core with common engine packages:

Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV) + Deep CDR	39.8	78 / 99	62 / 73		191,721	0.0188
Metascan (10 AV) + Deep CDR + Proactive DLP	43.5	61 / 84	47 / 67		175,414	0.0205
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	44.8	62 / 88	69 / 83		170,324	0.0211

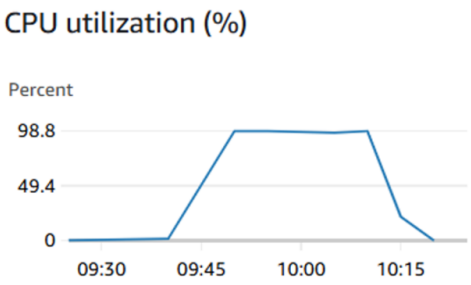
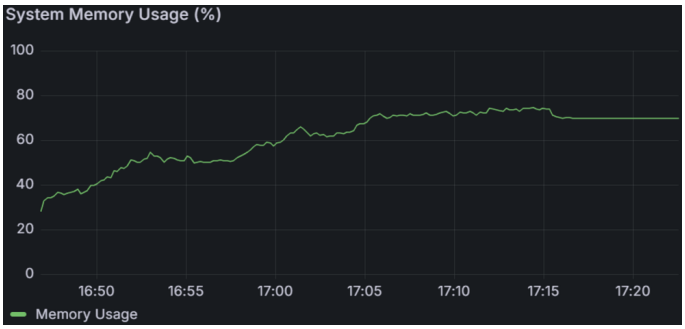
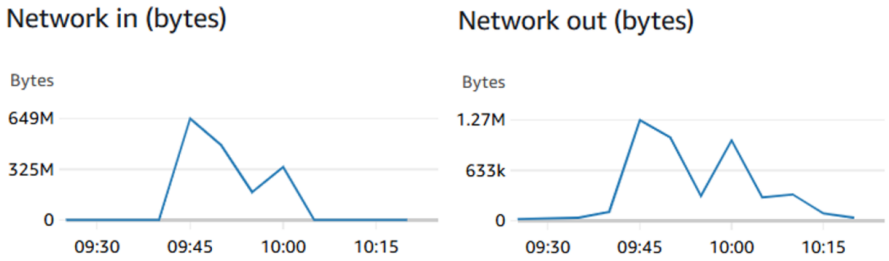
System resource utilization charts

MetaDefender Core with Metascan (10 AV engines) :

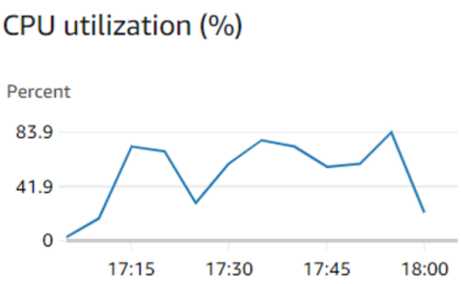
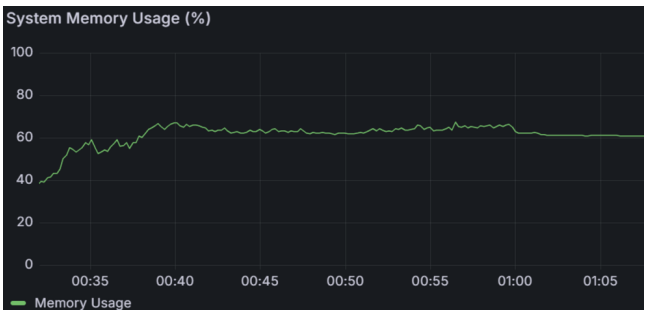
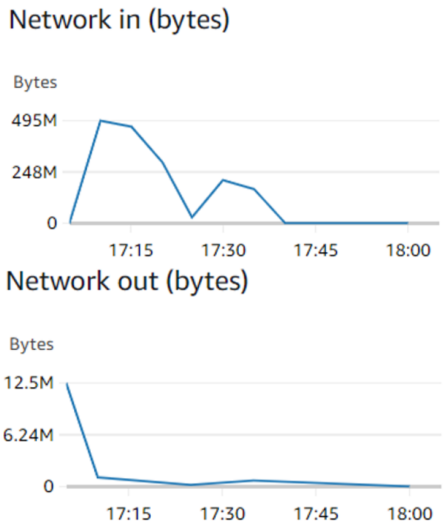
	Utilization charts
CPU usage	
Memory usage	
Network usage	

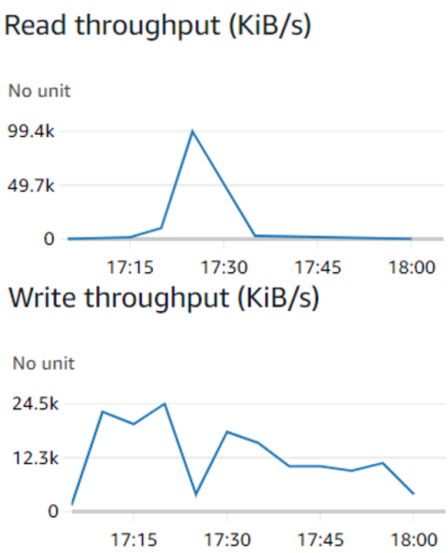
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR :

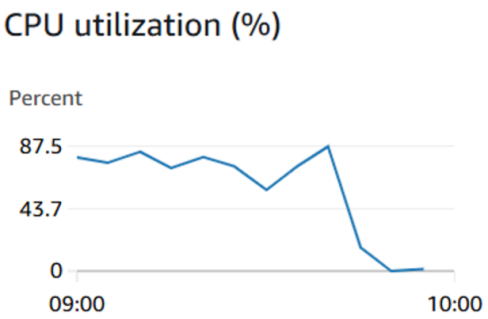
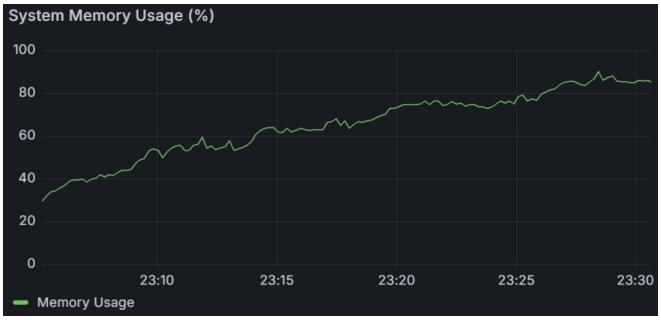
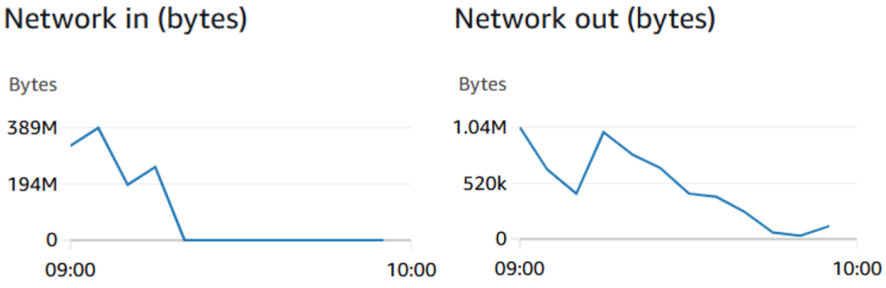
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP :

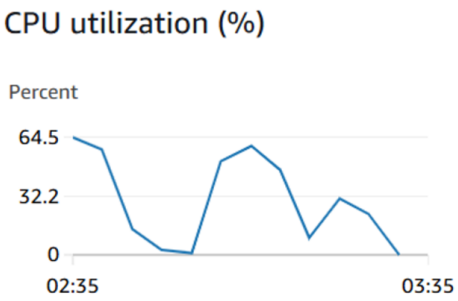
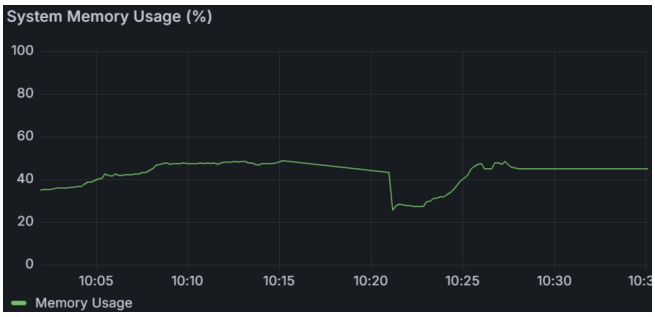
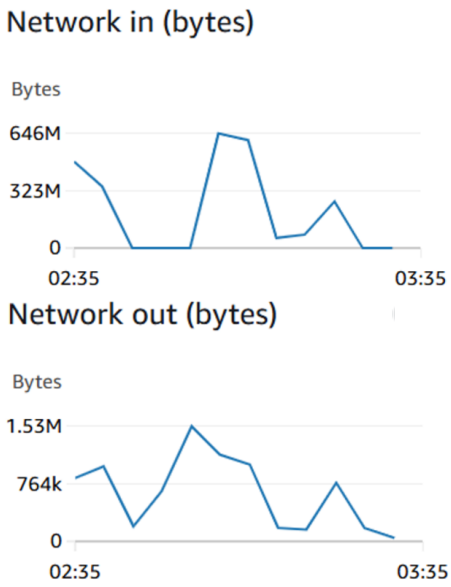
	Utilization charts
CPU usage	
Memory usage	
Network usage	

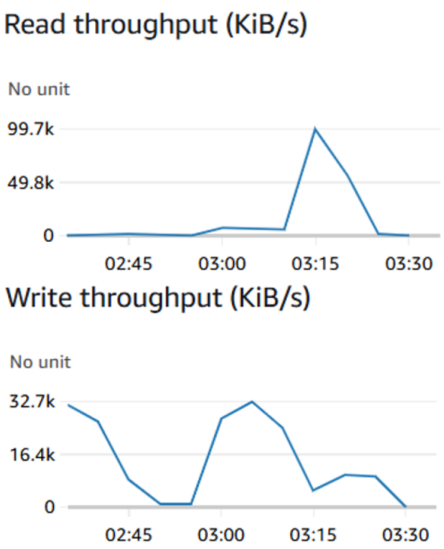
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP + Vulnerability :

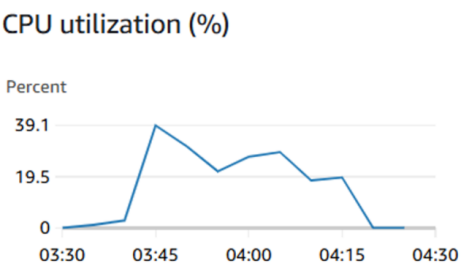
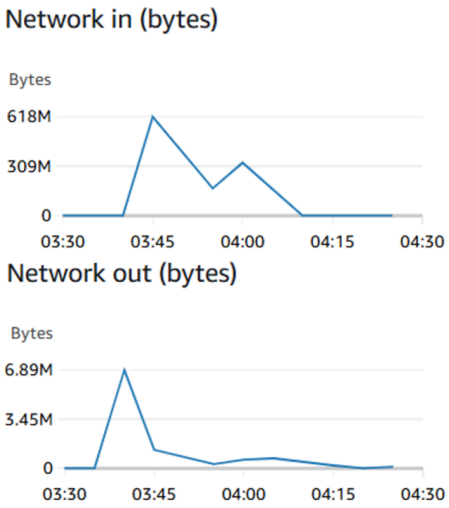
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

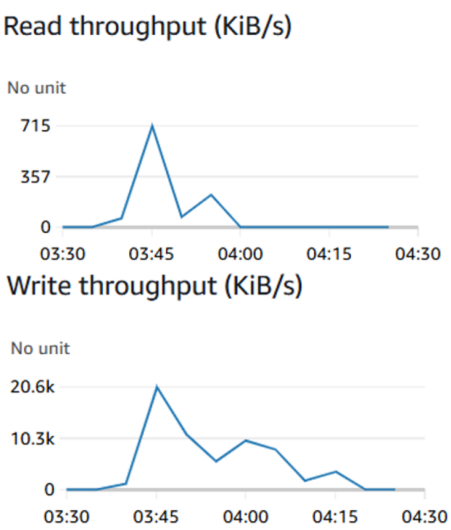
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

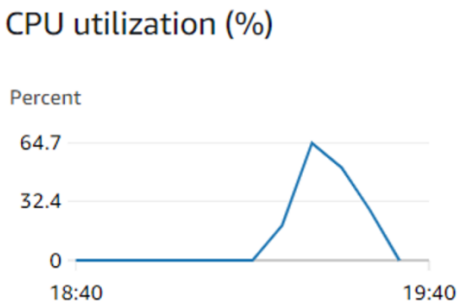
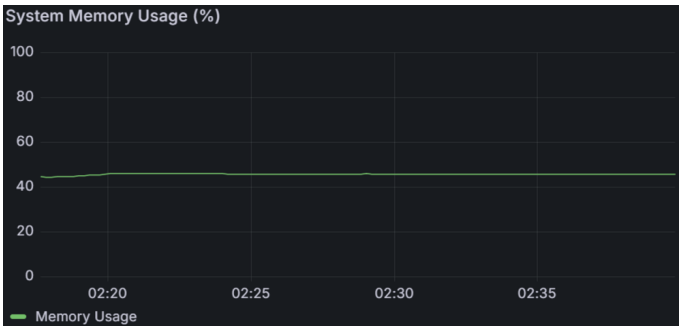
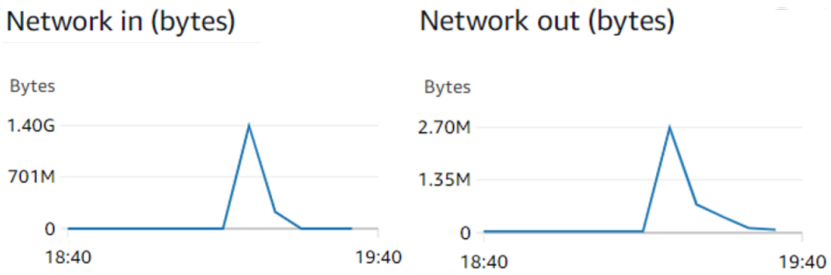
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability :

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

Shared DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

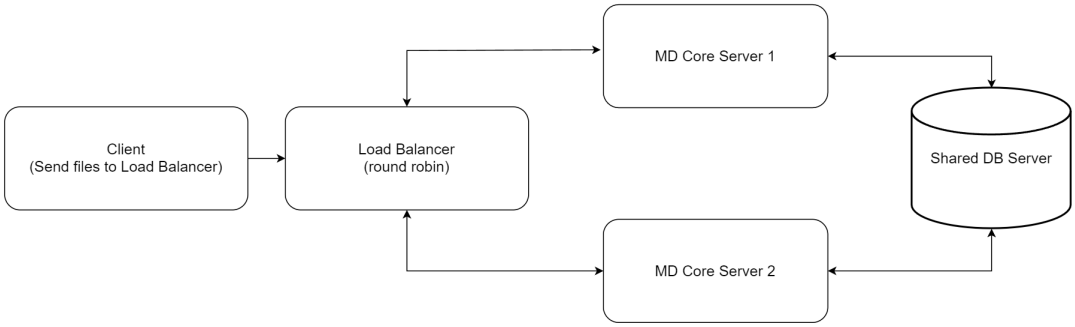
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV)	11.9	37.0	41.9	30.1	30.4	Receive: 6441.1 Send: 769.6	Receive: 6200.2 Send: 693.7	640,821.2	0.0
Deep CDR	10.5	60.1	66.7	30.3	32.2	Receive: 6379.8 Send: 594.8	Receive: 7146.1 Send: 609.6	729,368.5	0.0
Proactive DLP	7.8	38.7	37.5	29.7	31.0	Receive: 8842.6 Send: 791.8	Receive: 8635.6 Send: 729.1	965,455.7	0.0
Vulnerability	4.4	52.2	53.0	26.0	26.9	Receive: 16,215.5 Send: 1006.8	Receive: 15,785.7 Send: 997.2	1714,449.3	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV) + Deep CDR	15.4	72.4	75	34.3	35.4	Receive: 5436.6 Send: 611.3	Receive: 5589.3 Send: 607.2	495,697.4	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP	16.9	77.2	80.2	36.4	37.9	Receive: 5254 Send: 585.4	Receive: 5041.3 Send: 576.1	450,657.6	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	18.0	72.1	80.4	38.6	38.0	Receive: 4504.3 Send: 574.1	Receive: 4649.5 Send: 571.1	423,010	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

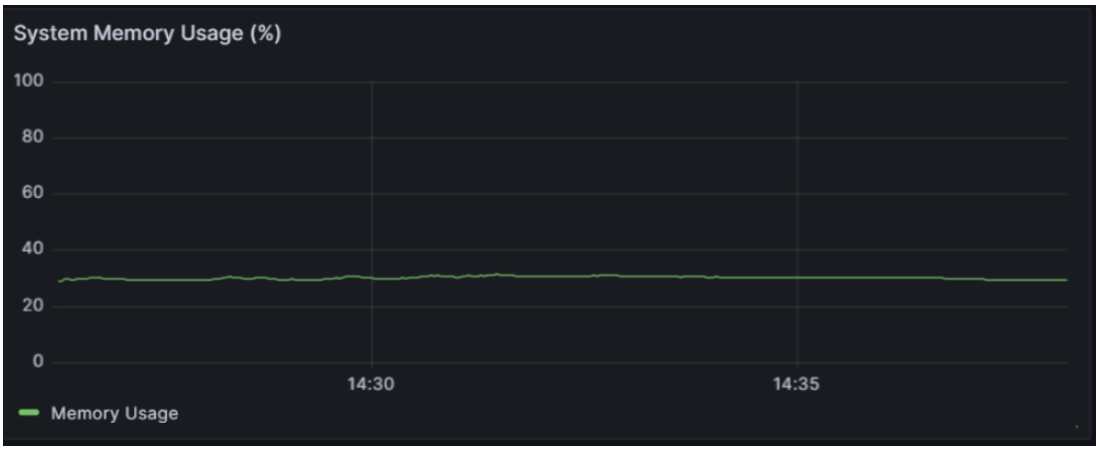
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2

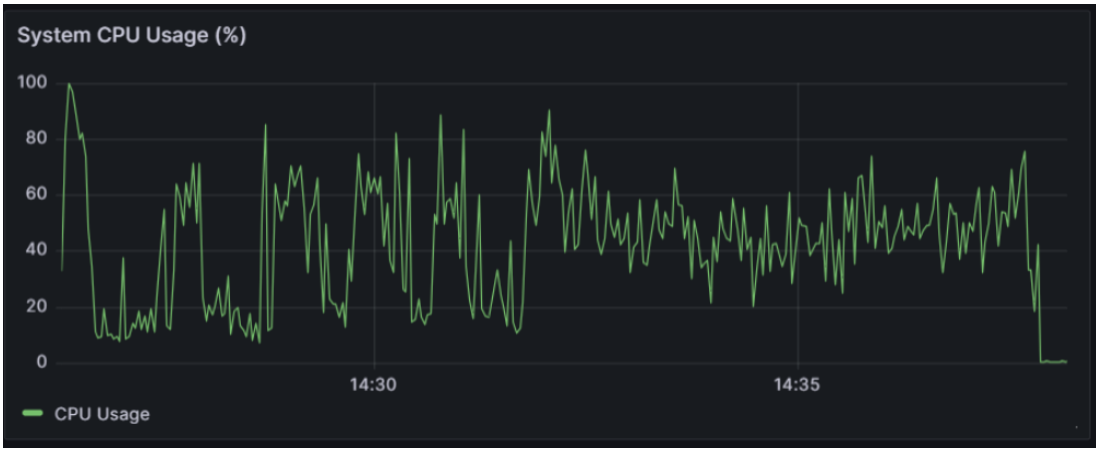


CPU usage

MD Core #1

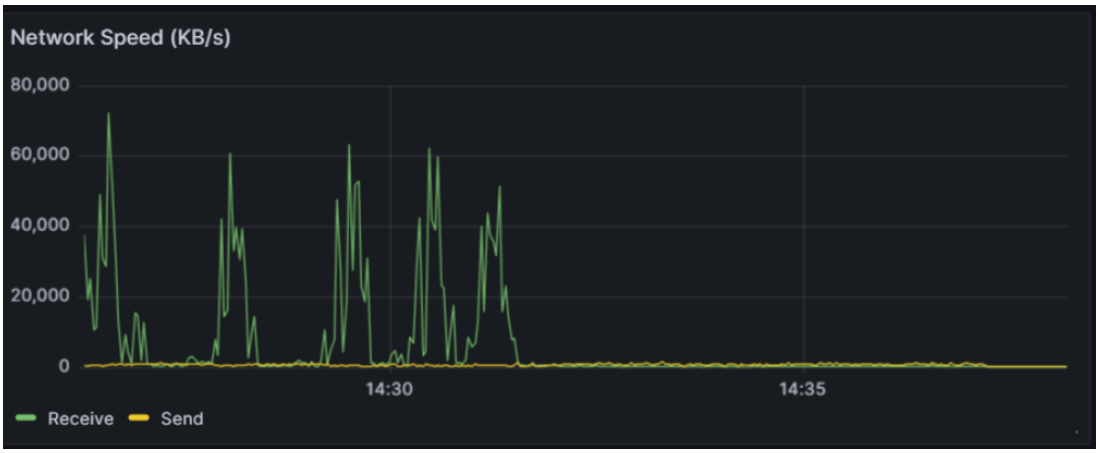


MD Core #2

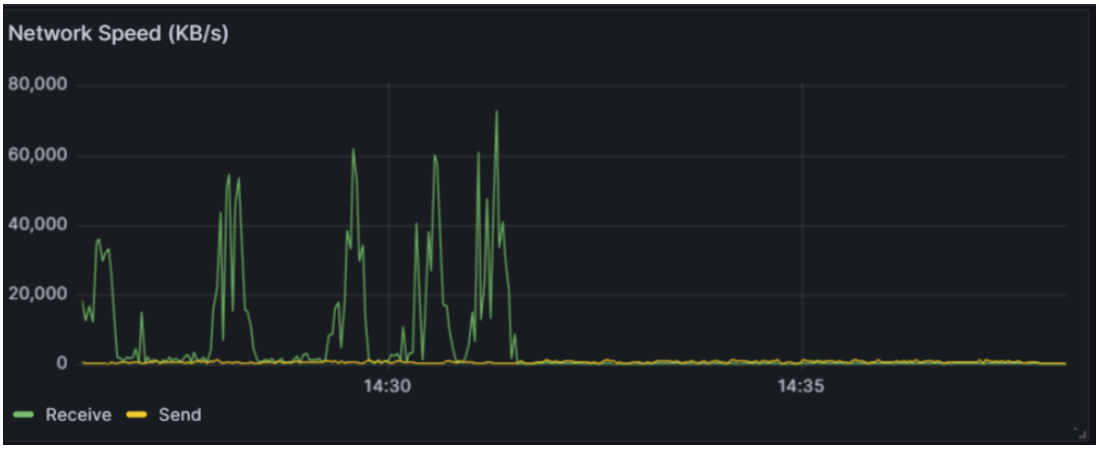


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2

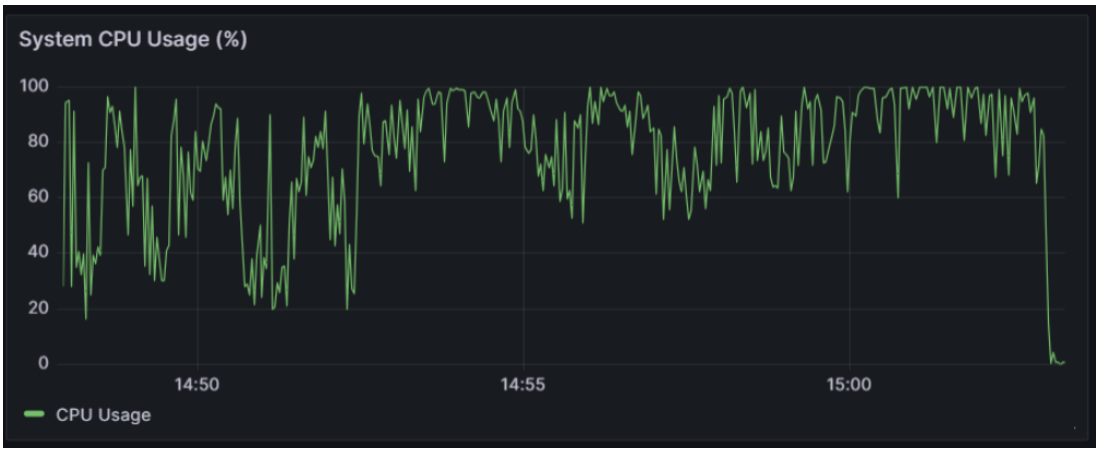


CPU usage

MD Core #1

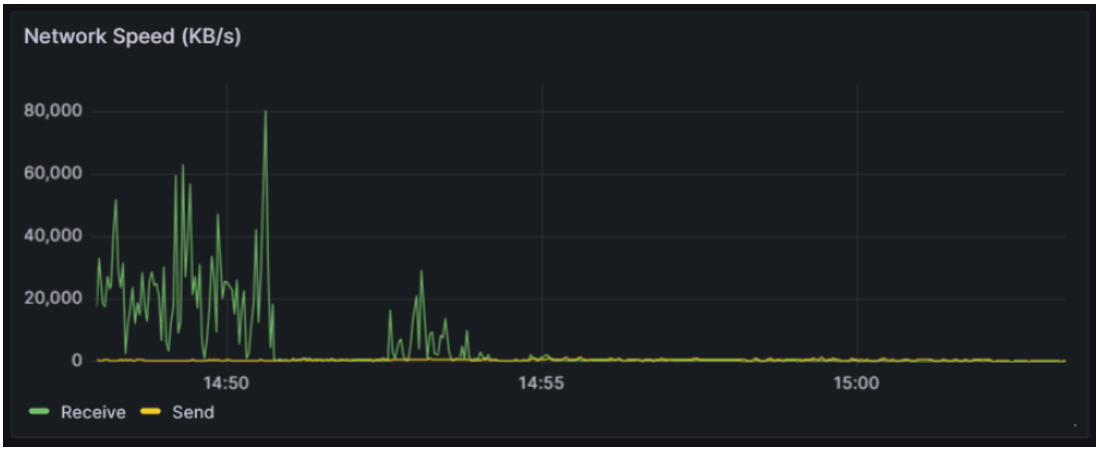


MD Core #2



Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

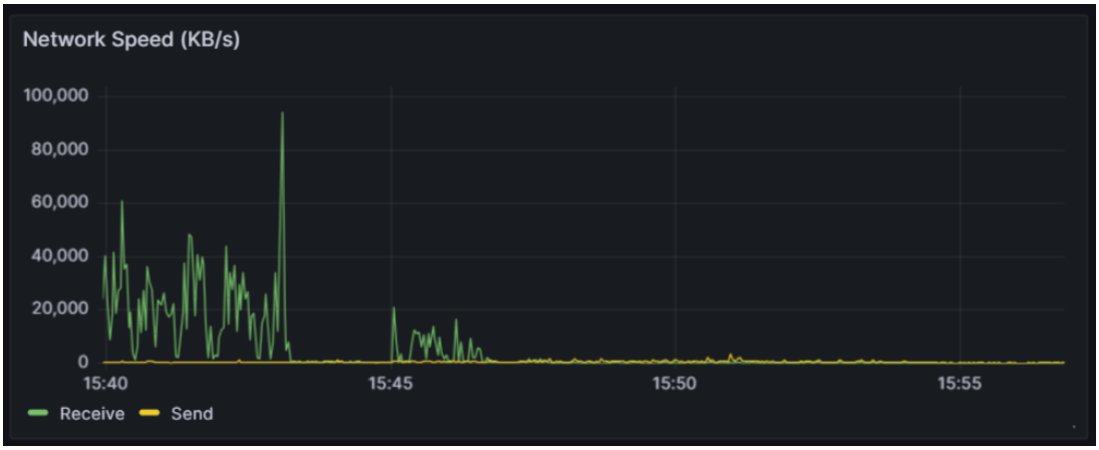


MD Core #2

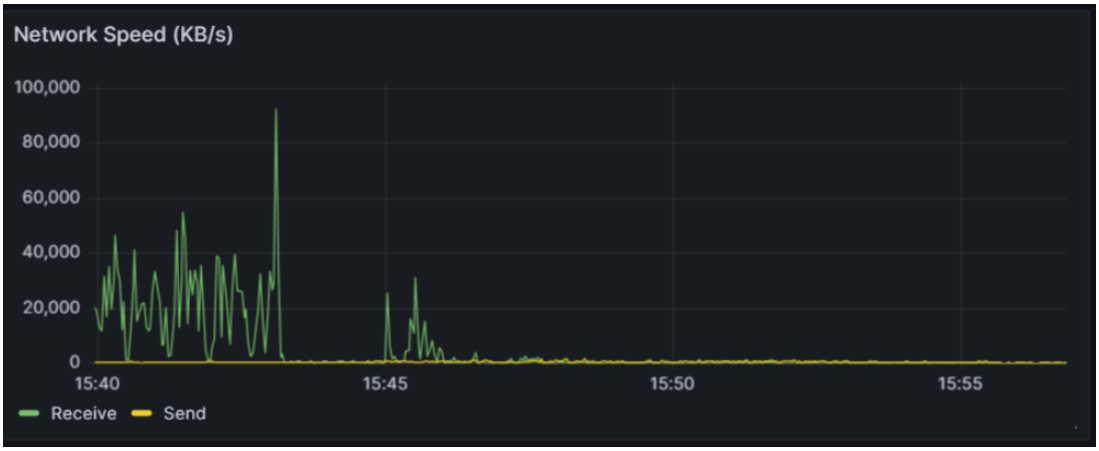


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1

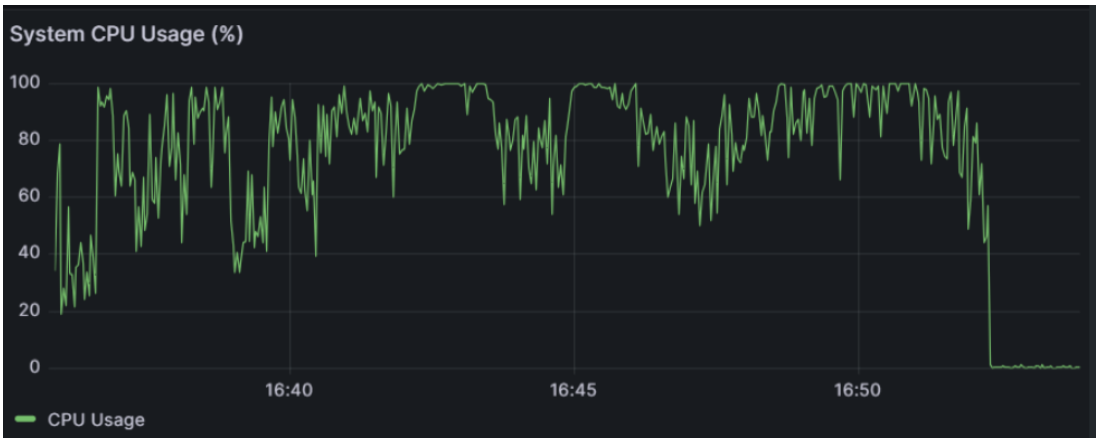


MD Core #2



CPU usage

MD Core #1

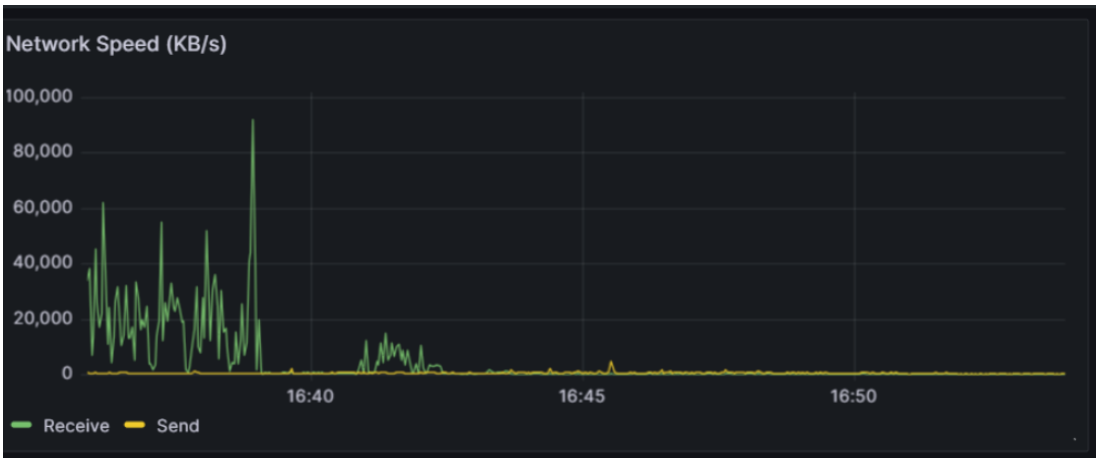


MD Core #2

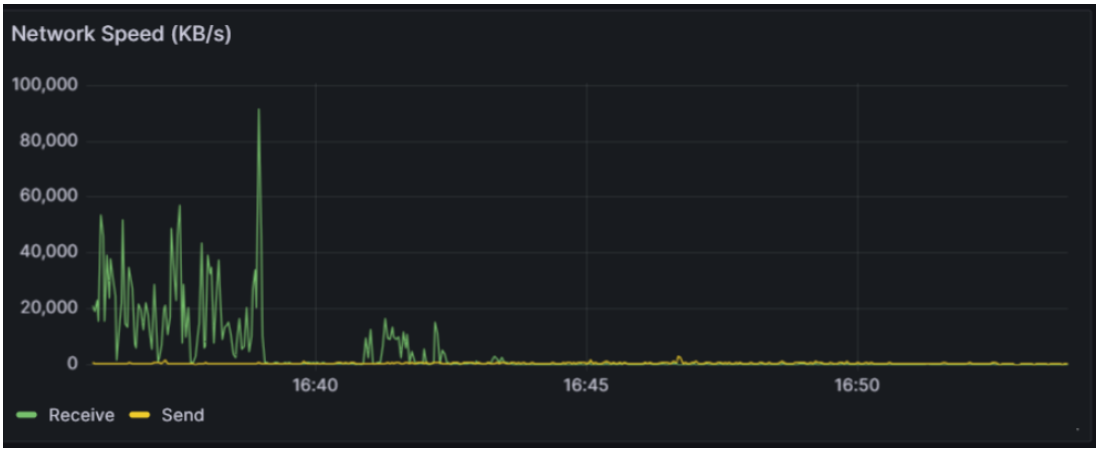


Network usage

MD Core #1



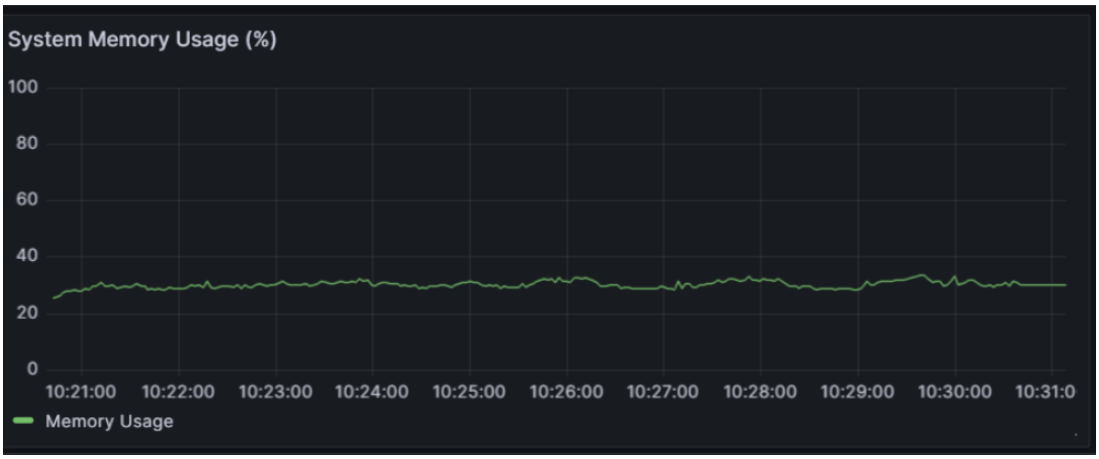
MD Core #2



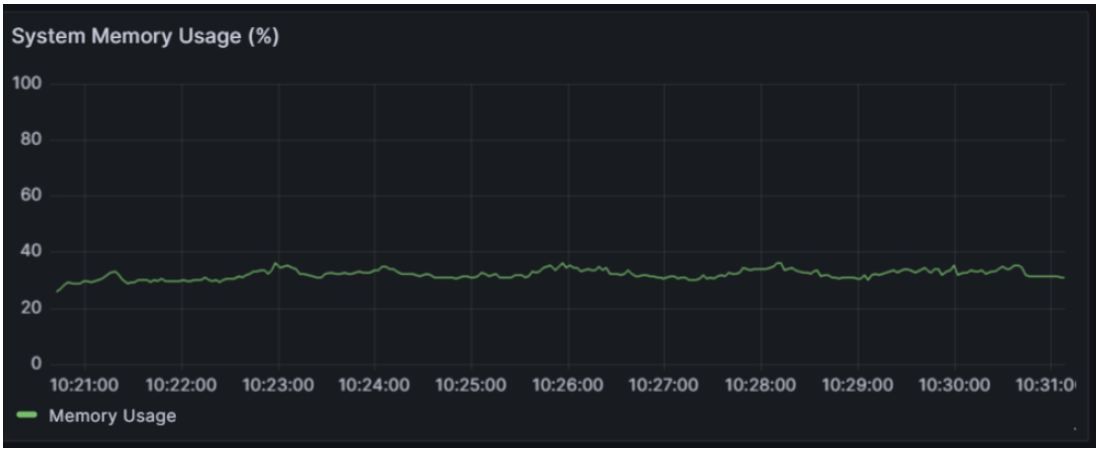
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

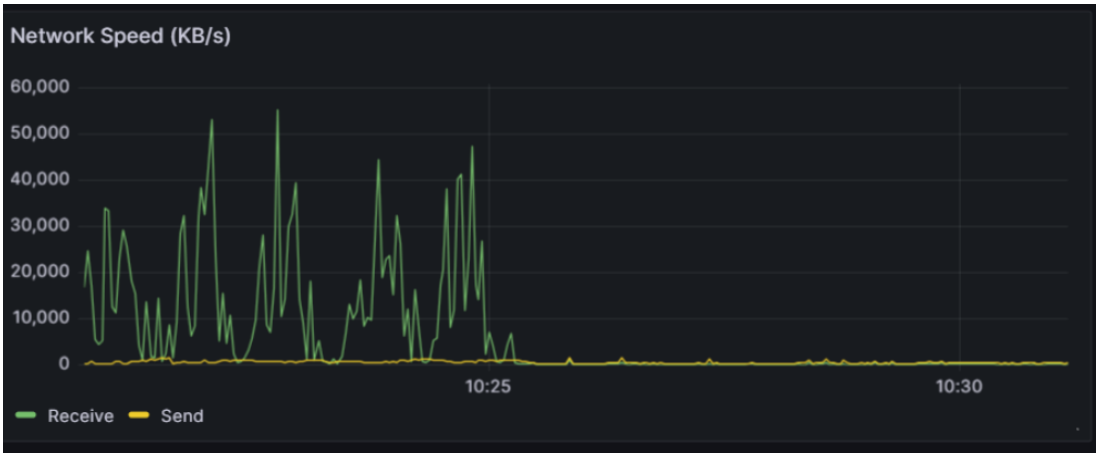


MD Core #2

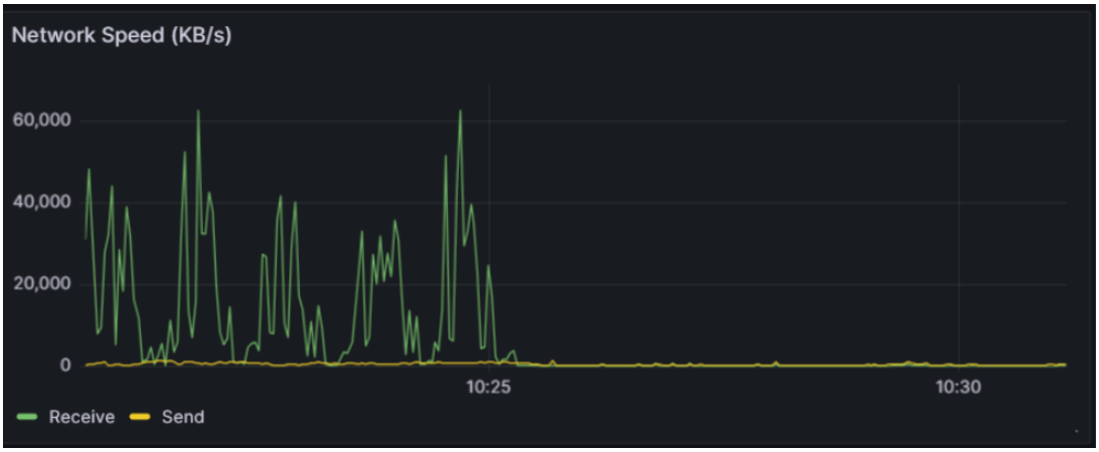


Network usage

MD Core #1



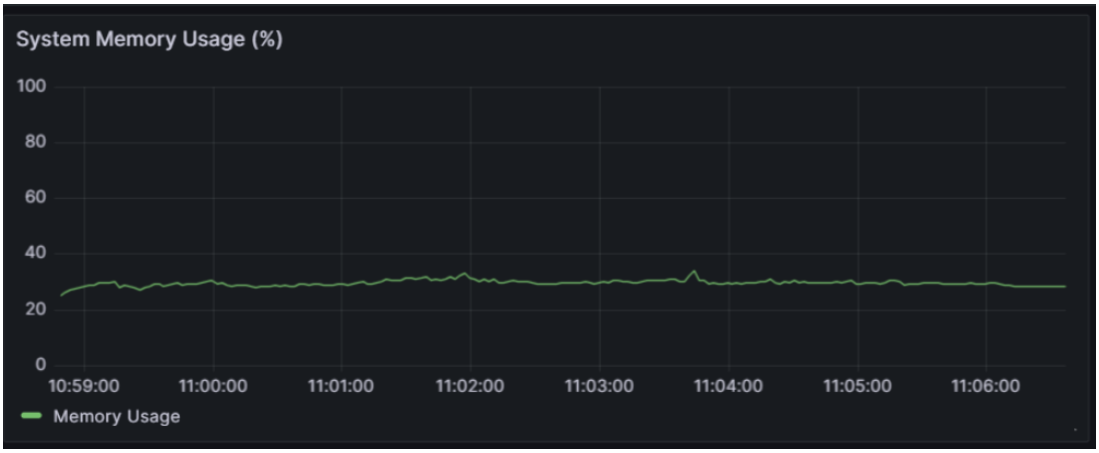
MD Core #2



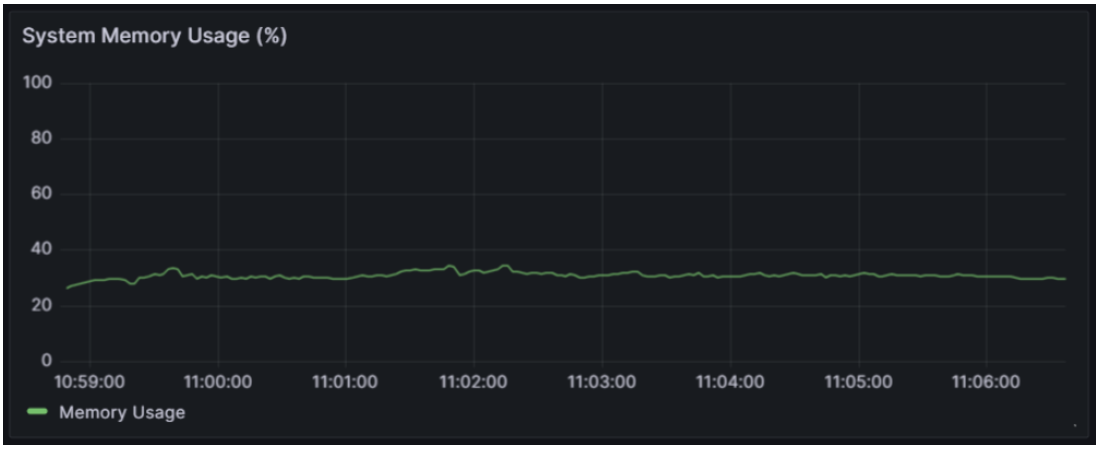
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

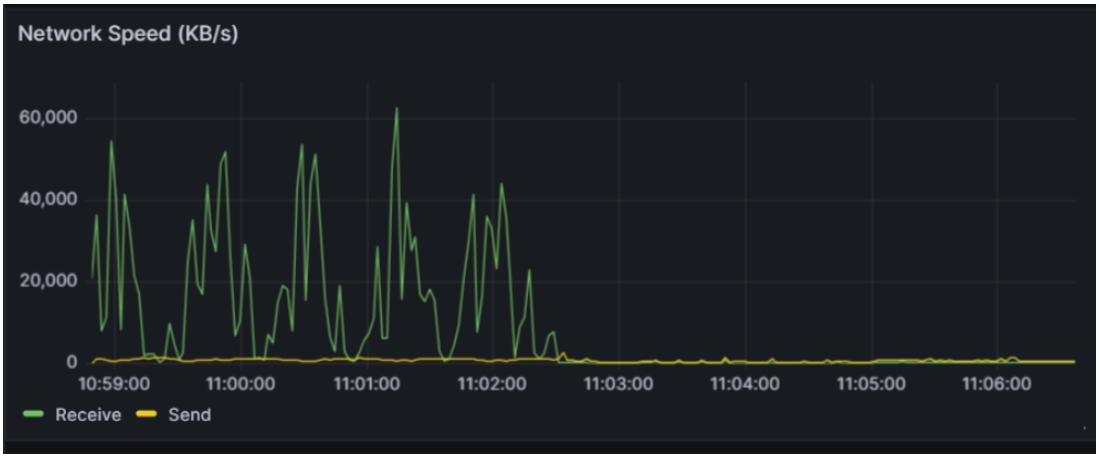


MD Core #2

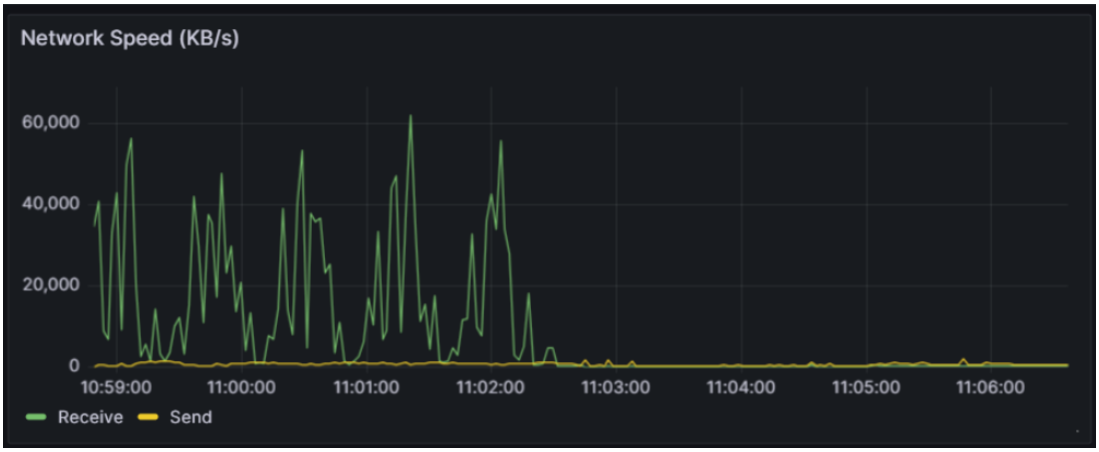


Network usage

MD Core #1



MD Core #2



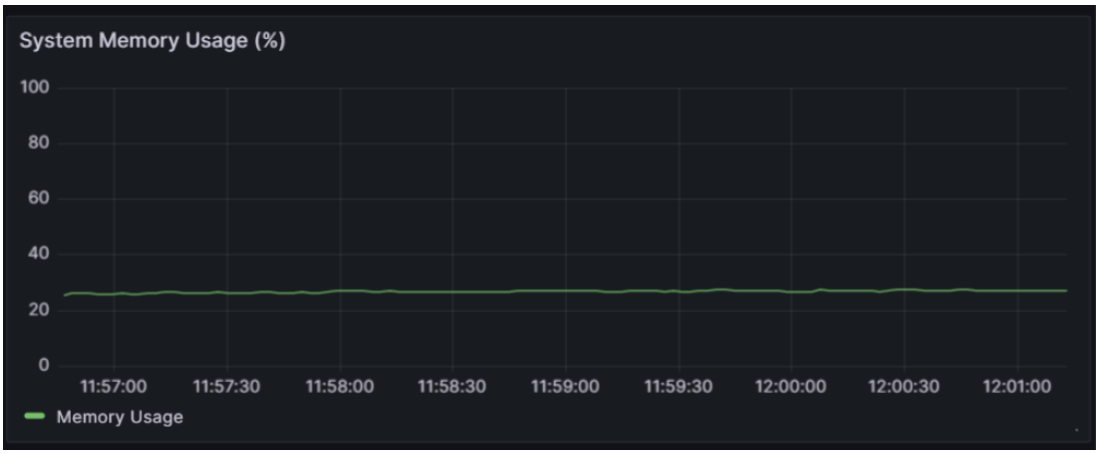
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

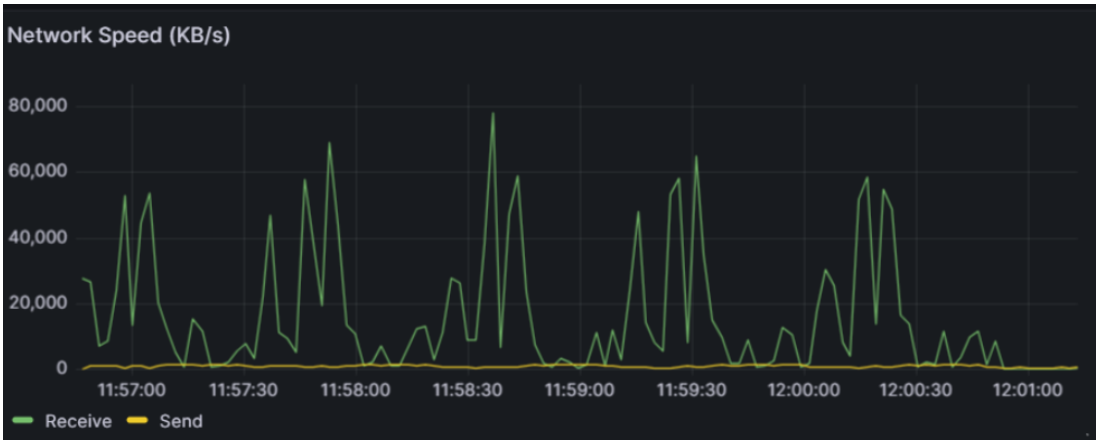


MD Core #2

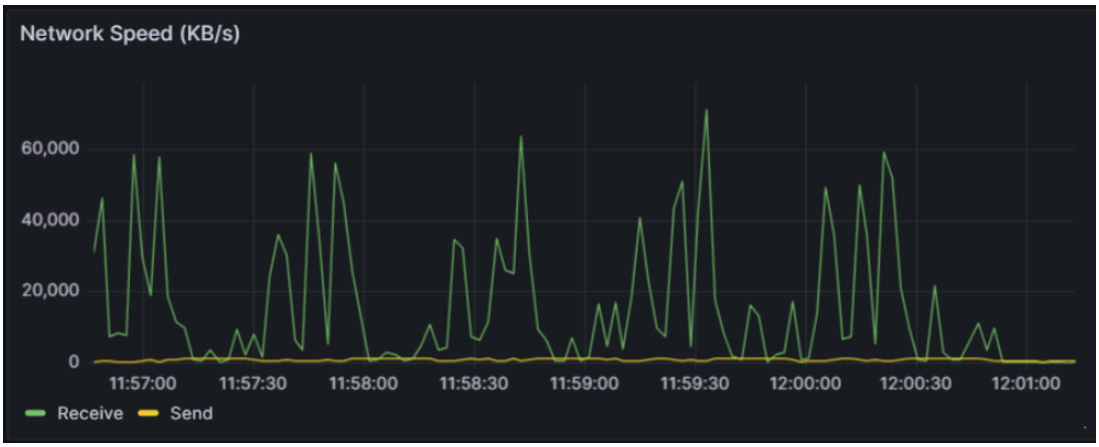


Network usage

MD Core #1



MD Core #2



Shared DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

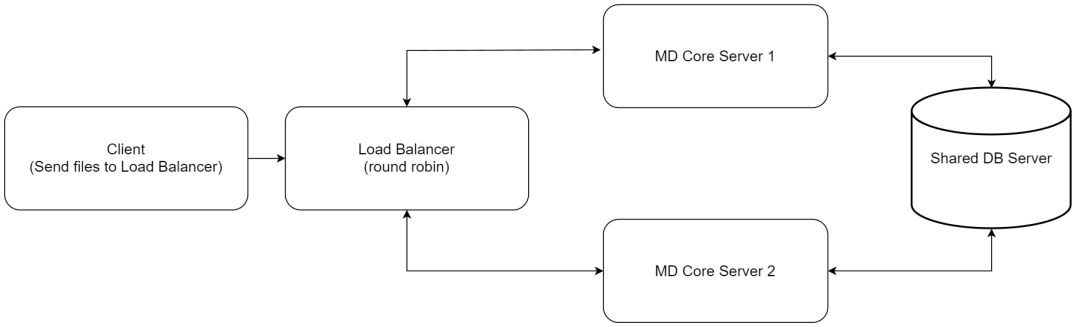
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV)	15.5	63.1	73.3	25.2	25.8	Receive: 4555.9 Send: 694.4	Receive: 4516.2 Send: 692.4	492,620.5	0.0
Deep CDR	9.70	50.7	51.2	31.9	32.5	Receive: 7062.9 Send: 623.2	Receive: 6653 Send: 654.1	786,189.6	0.0
Proactive DLP	7.2	35.2	35.0	27.9	27.5	Receive: 9047.4 Send: 802.4	Receive: 9859.8 Send: 787.5	1,055,110.8	0.0
Vulnerability	5.0	45.7	49.3	24.4	24.5	Receive: 13,585.7 Send: 966.7	Receive: 13,505.8 Send: 987.4	1,519,562.1	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV) + Deep CDR	19.9	79.1	77.6	34.0	34.2	Receive: 3785.9 Send: 573.8	Receive: 3663.1 Send: 577.7	382,716	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP	21.6	80.3	81.5	37.7	44.4	Receive: 3305.6 Send: 543.3	Receive: 3414.3 Send: 556.2	353,694.9	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	22.5	84.2	79.8	37.9	44.7	Receive: 3267.9 Send: 554.9	Receive: 3398.2 Send: 553.8	338,507.6	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

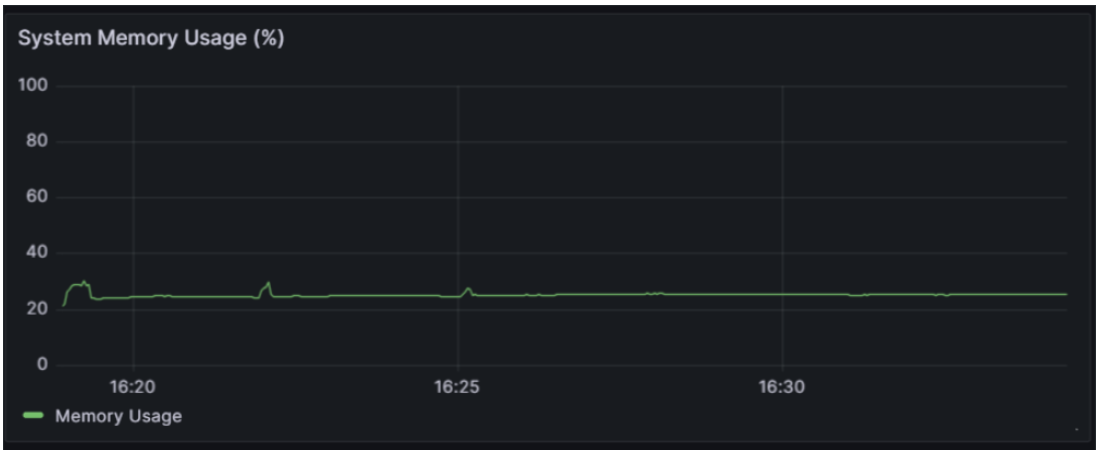
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

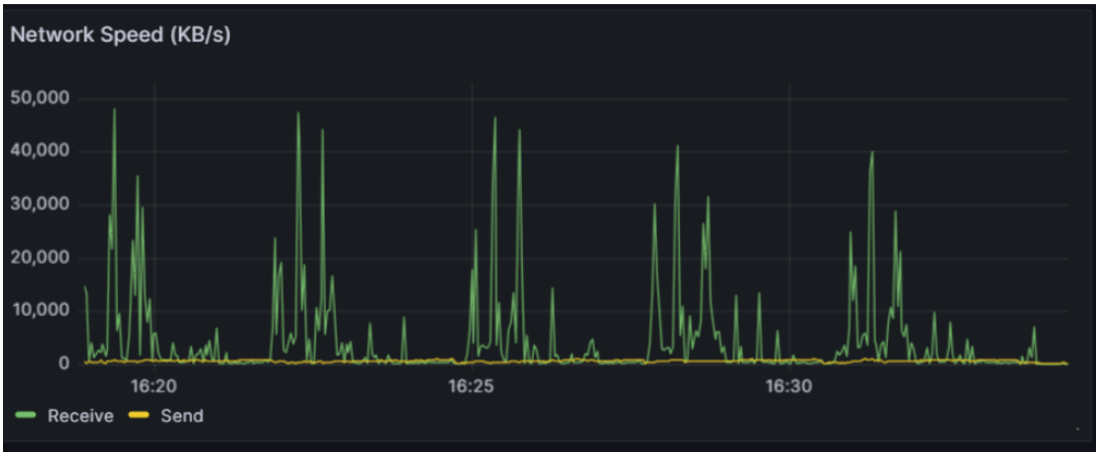


MD Core #2

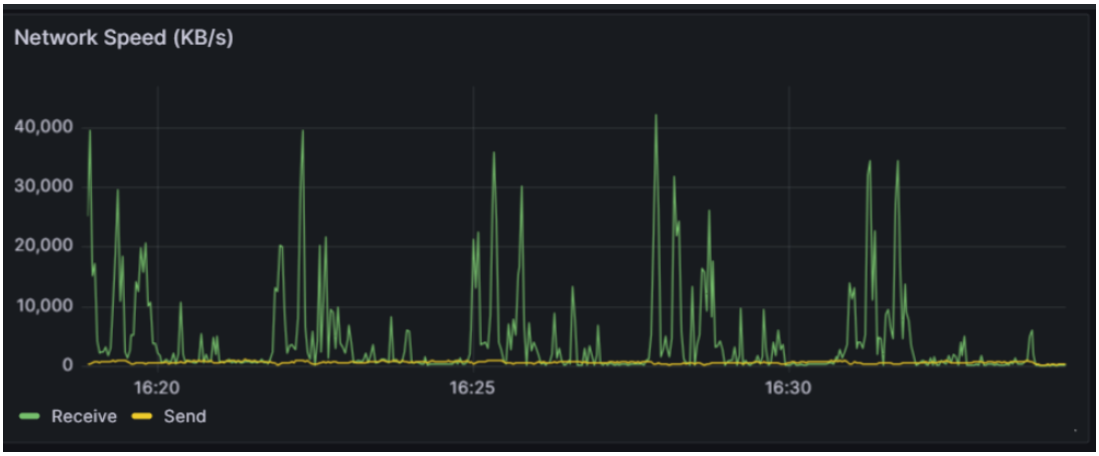


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

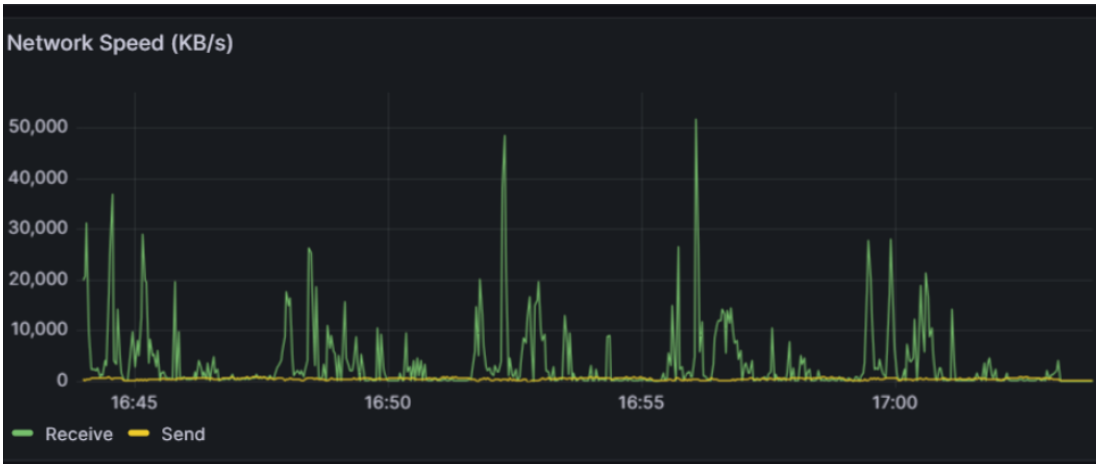


MD Core #2

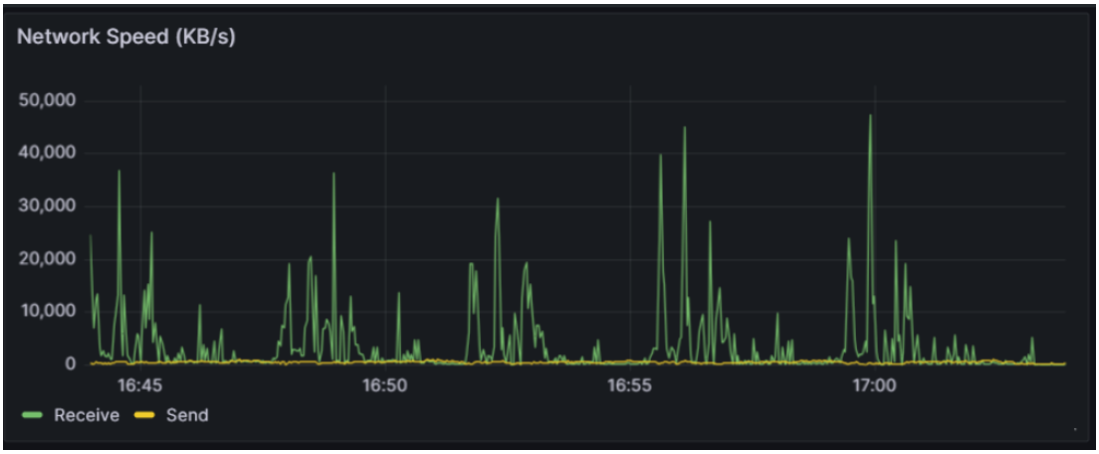


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1

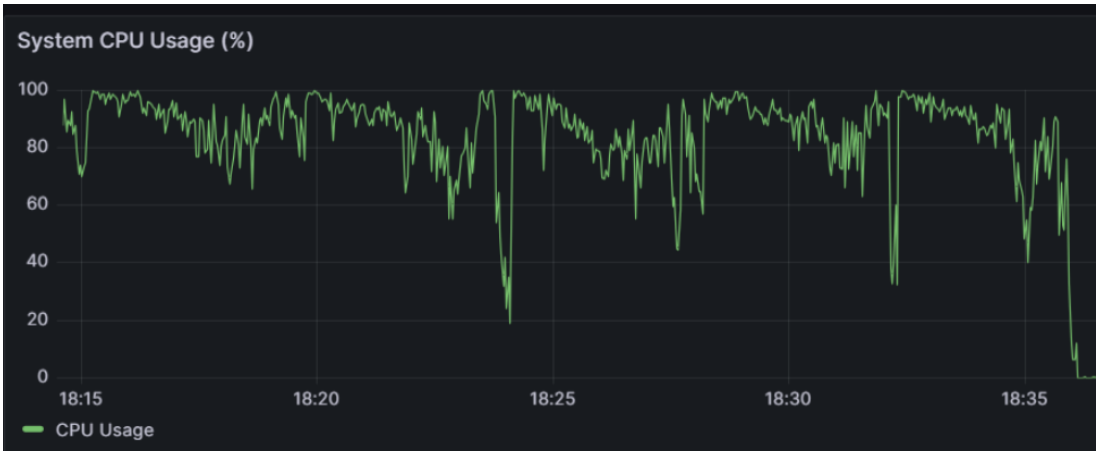


MD Core #2



CPU usage

MD Core #1

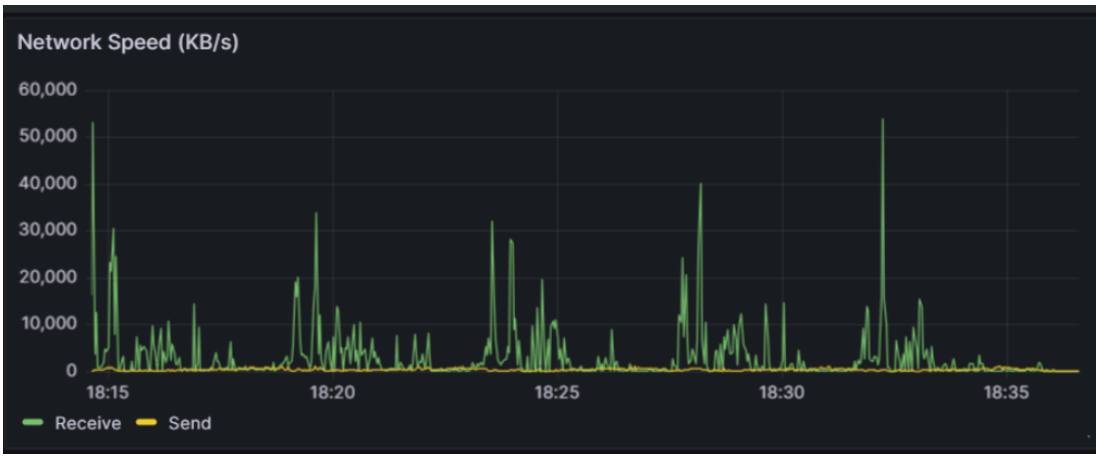


MD Core #2

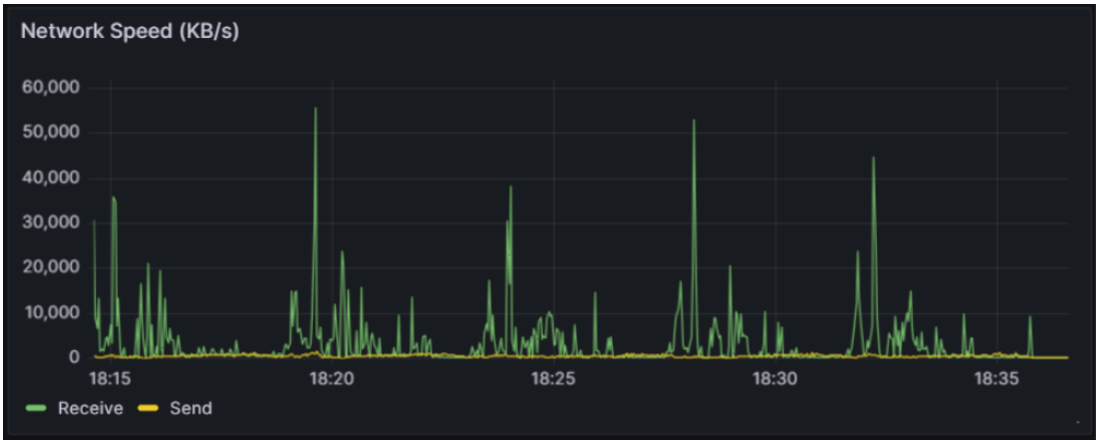


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

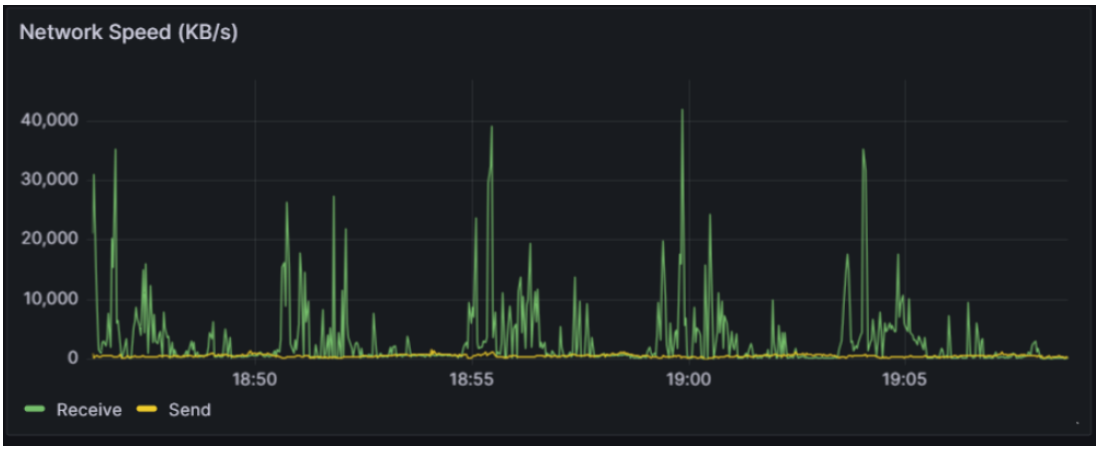


MD Core #2

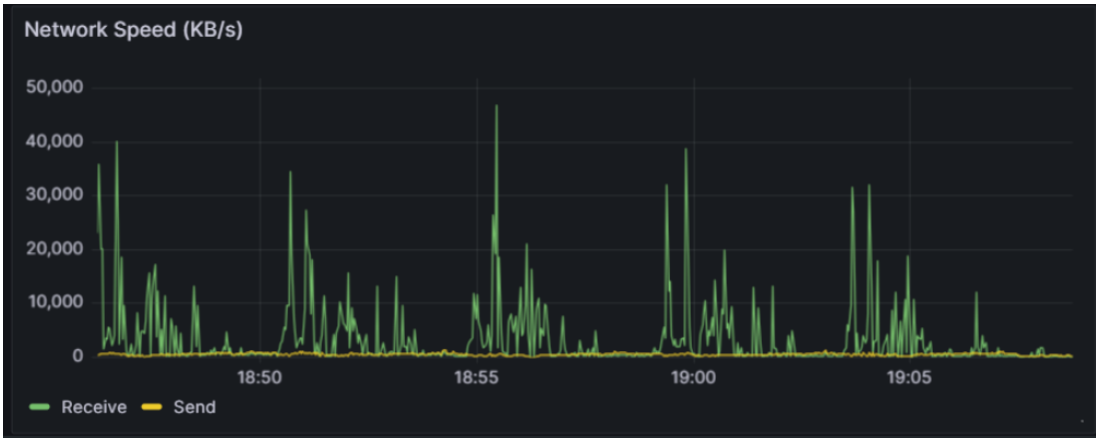


Network usage

MD Core #1



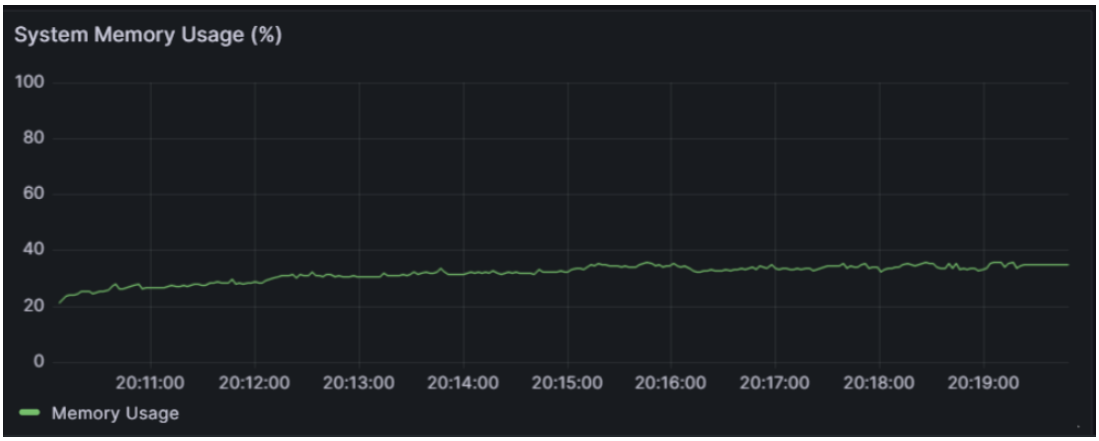
MD Core #2



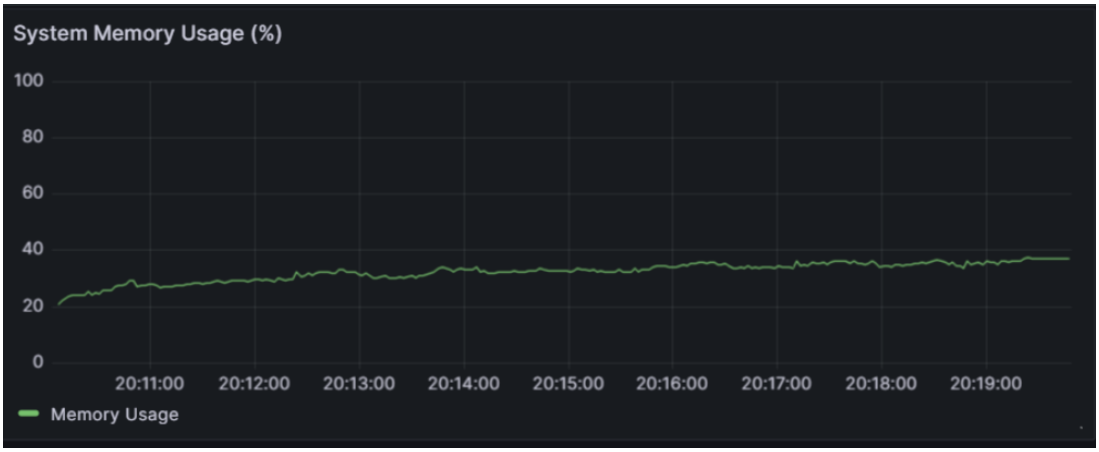
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1

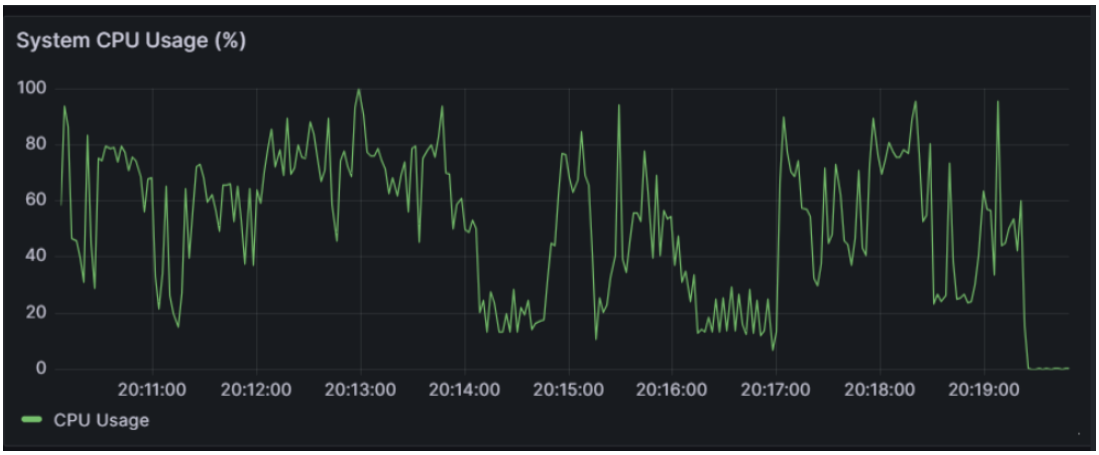


MD Core #2



CPU usage

MD Core #1

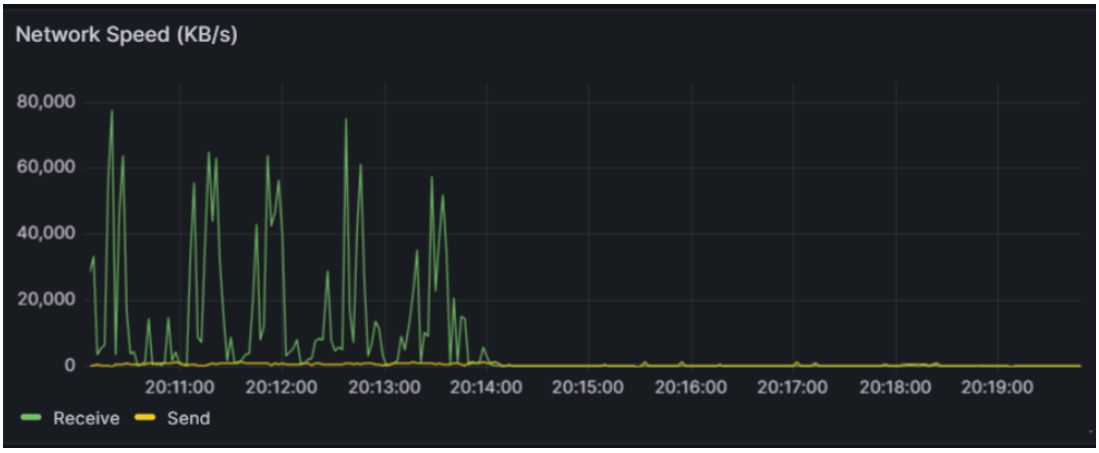


MD Core #2

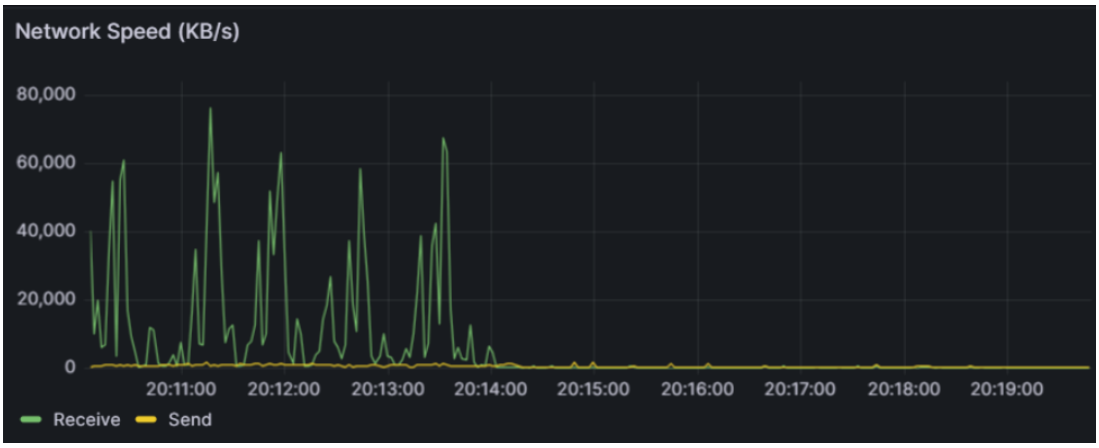


Network usage

MD Core #1



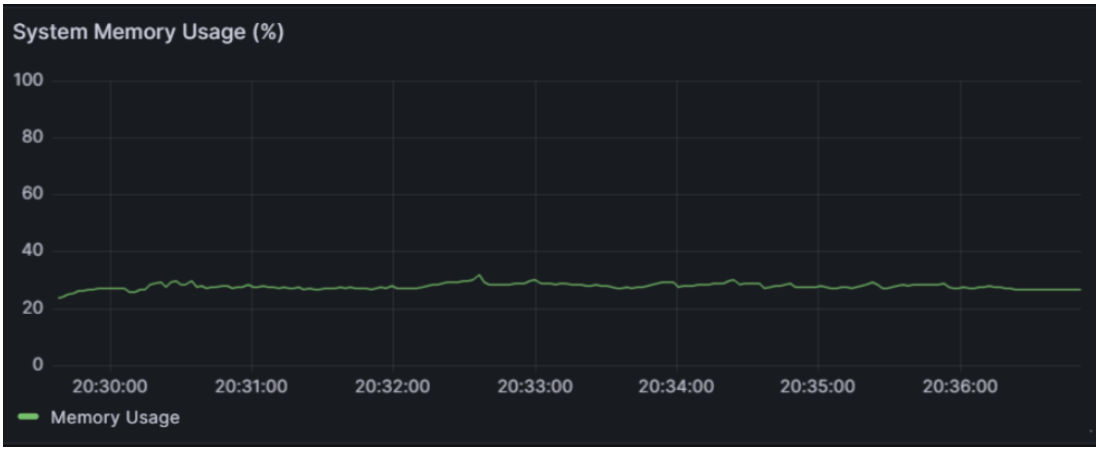
MD Core #2



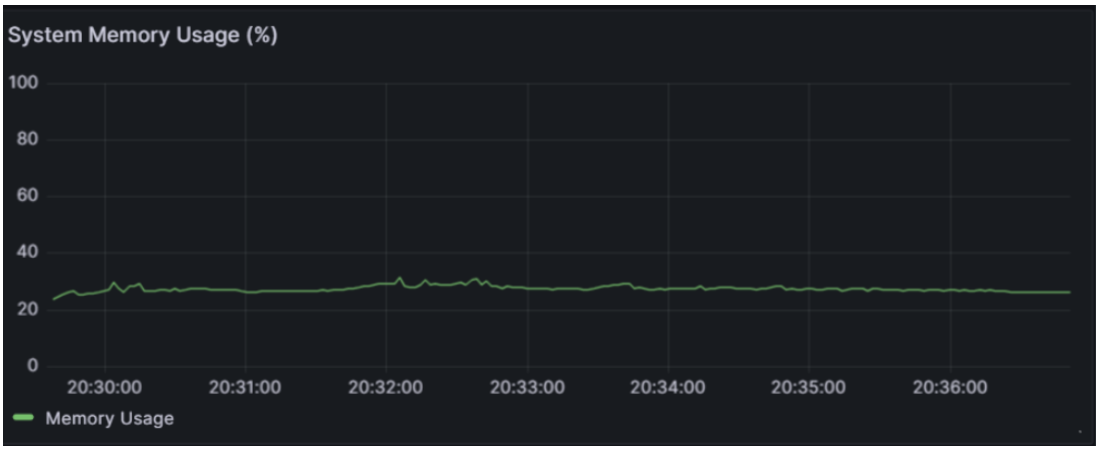
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1

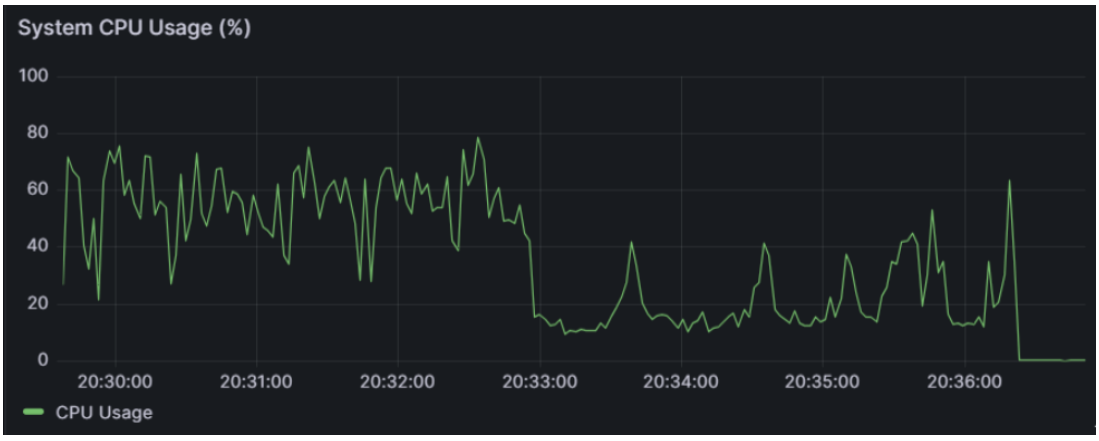


MD Core #2

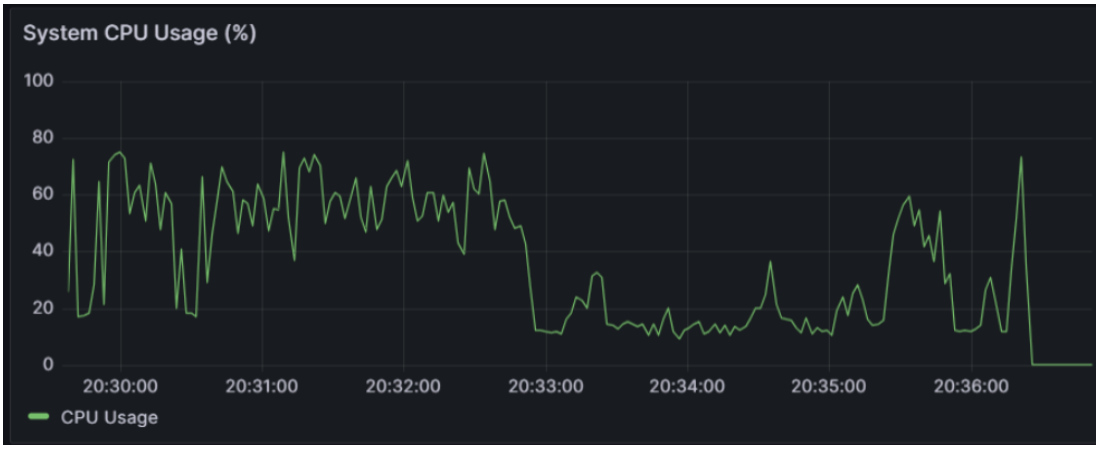


CPU usage

MD Core #1

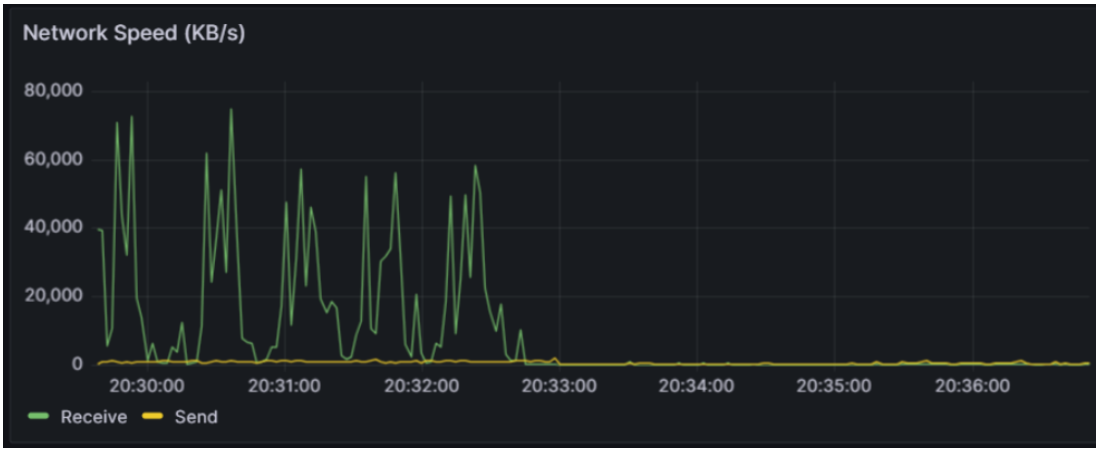


MD Core #2

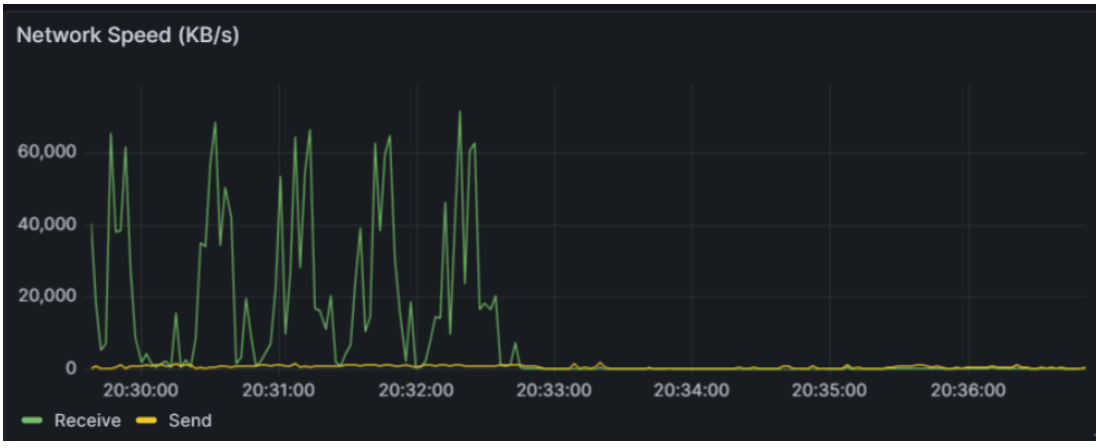


Network usage

MD Core #1



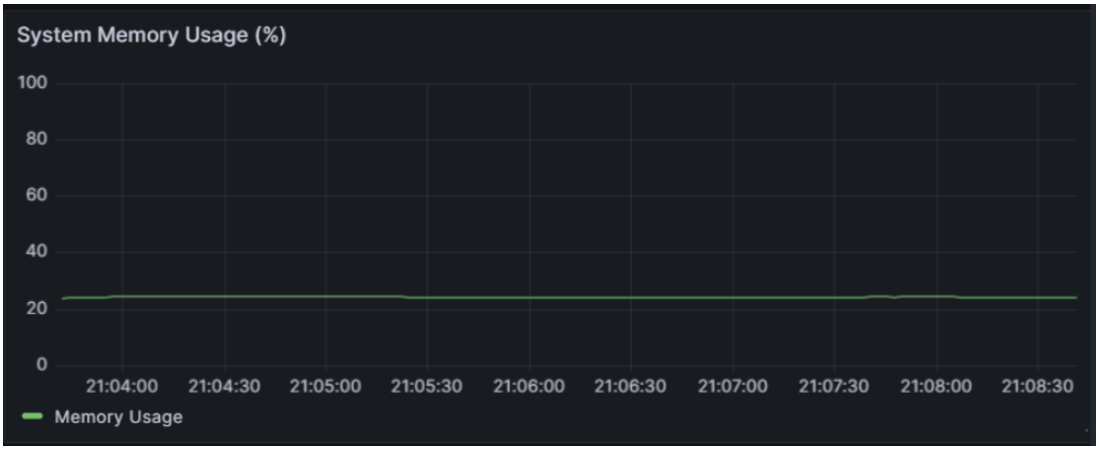
MD Core #2



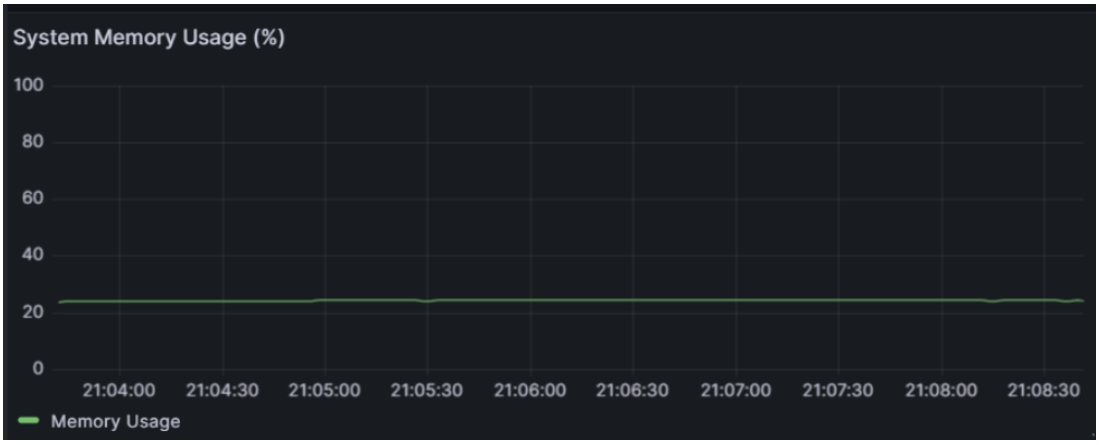
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

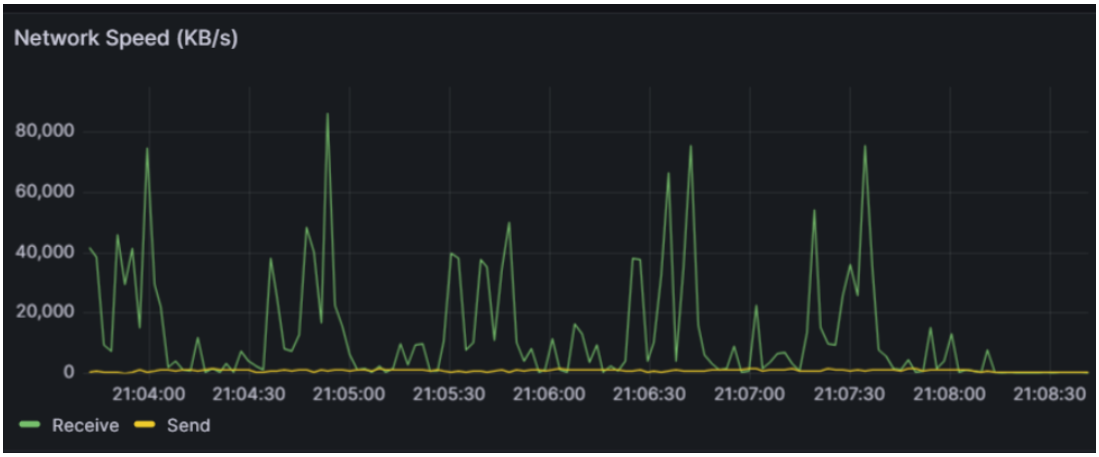


MD Core #2

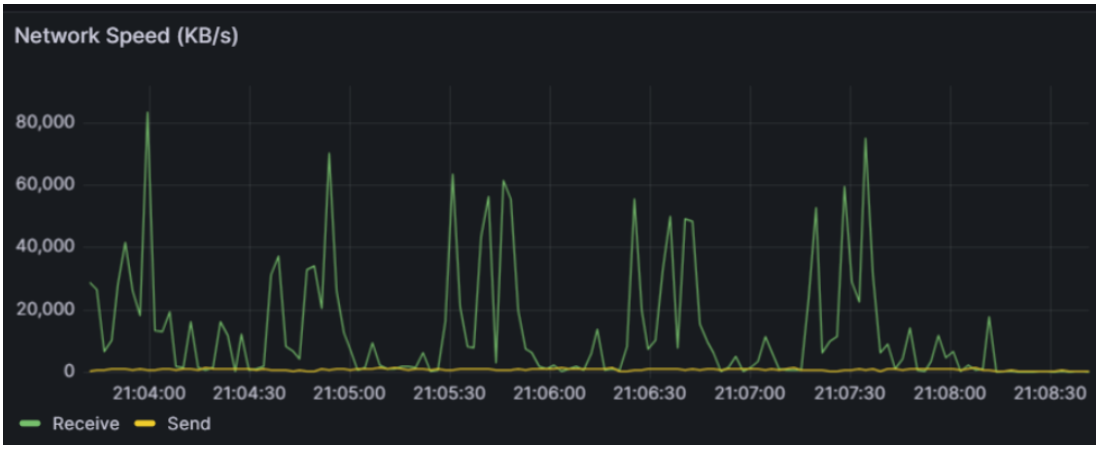


Network usage

MD Core #1



MD Core #2

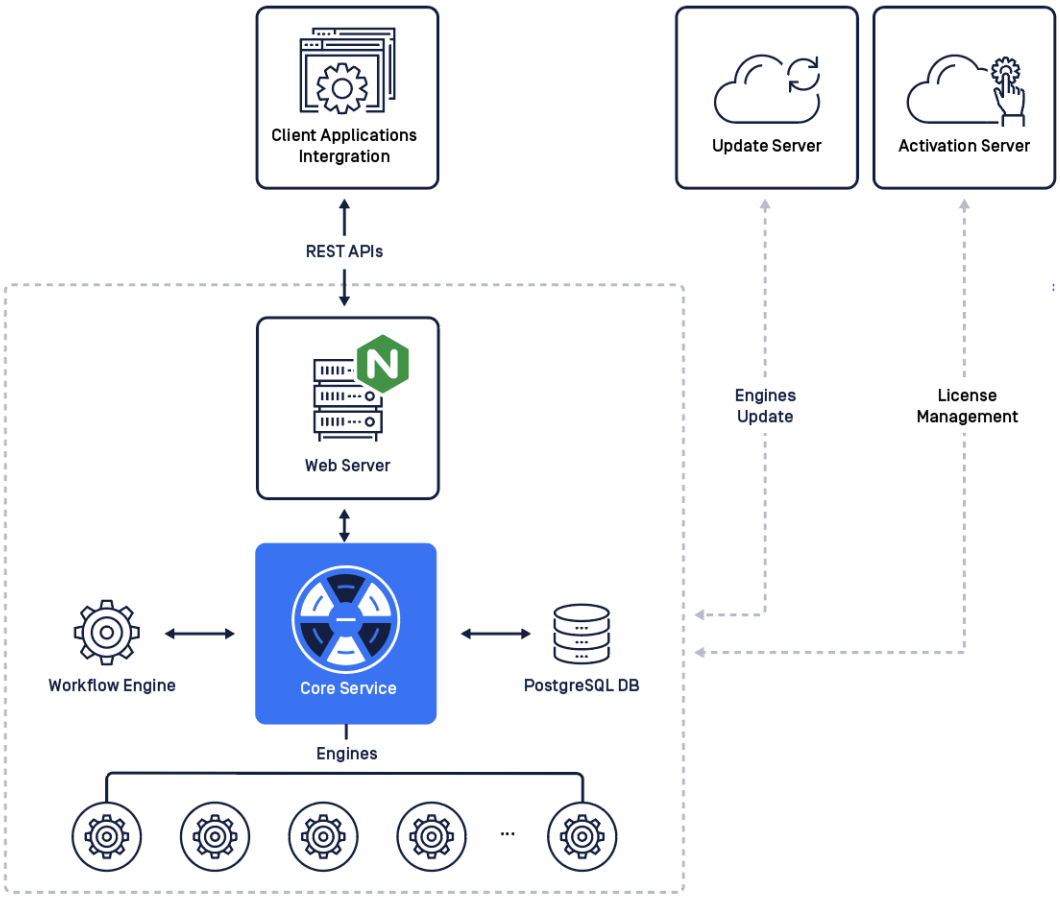

Network Speed (KB/s)
80,000
60,000
40,000
20,000
0
21:04:00
21:04:30
21:05:00
21:05:30
21:06:00
21:06:30
21:07:00
21:07:30
21:08:00
21:08:30
Receive
Send

MetaDefender Core Documentation

The users can consult this web page or, alternatively, they can download the manual in pdf format from the link below:

MetaDefender Core manual (SHA256:).

Recommended System Configuration



MetaDefender Core supports a wide range of deployment options, including air-gapped environments, providing the flexibility to choose the environment that best suits customers' requirements and infrastructure.

- Physical server.
- Virtualization platforms like VMware, Hyper-V and XenServer.
- Cloud Deployment.
- Container Deployment.

> **Note**
> Before installing MetaDefender Core, please refer to the **recommended minimum system configuration** listed below. The requirements are based on a standardized customer deployment and may require more or less resources to achieve the level of performance desired in your environment. Please note that the server specifications are built to allow a high volume daily processing. Please refer to Performance and Load Estimation for more details.

The recommendations below are for MetaDefender Core, API usage only. For any other use cases, please consult the user guide of the licensed products for accurate recommendations.

For certain use cases these might be adjusted and customized on specific needs and SLAs. We highly recommend to engage our ProServ team to assist in fine tuning MetaDefender and get the maximum performance out of your systems.

Microsoft Windows Deployments

[Windows] Supported Operating Systems

Only support 64-bit platforms.

- Windows 10, Windows 11 23H2, 11 IoT (22H2) only for End-User systems (e.g. Kiosk deployments)
  - Based on Microsoft License Terms, you're not allowed to use Windows Desktop as a server, therefore, MetaDefender Core can't be used on these systems for server use cases.
- Windows Server 2016, 2019, 2022

> **Disclaimer**
> End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

> **Plan for legacy OS support**
> OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:
>
> - Ubuntu 20.04 after **April 2025**.
> - Amazon Linux 2 after **June 2025**.
> - Windows 10 after **October 2025**.
>
> If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

[Windows] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 8	8	8 GB	16 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 12	16	16 GB	24 GB + 50 GB + 100 GB
MetaDefender Core 16	16	16 GB	32 GB + 50 GB + 100 GB
MetaDefender Core 20	32	16 GB	40 GB + 50 GB + 100 GB
MetaDefender Core MAX	32	32 GB	120 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core is installed.
- For a better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.
- SBOM processing requires more disk space, please allow at least 15GB free disk for the engine to work properly.

[Windows] Third Party Dependencies

- .NET framework 4.5 or above
- Microsoft Visual C++ Redistributable for Visual Studio 2015-2022 (version 14.38 or higher); x64 version.
- PostgreSQL: Only applicable when using a pre-installed PostgreSQL server running remotely.
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Some engines also have dependencies as described below:

Engine Name	Dependency
Deep CDR	Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 5.8.0 or above)
ESET	MetaDefender Core v4 temporary directory should have more than 200MB free disk space
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
Proactive DLP	• Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 2.0 or above) • .NET 6 dependencies • CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
Scrutiny	Microsoft Visual C++ 2015 - 2019 Redistributable x86 and x64
Systweak	.NET runtime 6.0, 7.0 or 8.0
Bkav Pro	Microsoft Visual C++ 2015 - 2022 Redistributable x64

[Windows] Installation Details

MetaDefender Core on Windows uses "C:\Program Files\OPSWAT" folder for storing resources or the installation directory.

MetaDefender will use its resources folder to store temp files as part of the analysis. **It's recommended to exclude this folder from real-time protection monitoring**. Knowledge base:

- How to set up exclusions on anti-malware software to prevent disruption
- Mitigating scan failure triggered by real-time protection service
- Can local AVs interrupt ongoing scans?

Linux Deployments

[Linux] Supported Operating Systems

Only support 64-bit platforms.

- Red Hat Enterprise Linux 8, 9
- Rocky Linux 9
- Debian 11, 12
- Ubuntu 20.04, 22.04
- Amazon Linux 2 is supported via Amazon Machine Image (AMI)

Plan for legacy OS support

OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:

- Ubuntu 20.04 after **April 2025**.
- Amazon Linux 2 after **June 2025**.
- Windows 10 after **October 2025**.

If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

For Hardended Linux OS

Some AV engines could be failed to initialize and run on MetaDefender Core (e.g. ESET, Kaspersky) due to execution permission required by them on `/var/tmp/ometascan` folder.

Please follow steps at Why have the ESET and Kaspersky scan engines failed to initialize on the hardened Linux OS? for remediation details.

Disclaimer

End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

[Linux] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 5	4	4 GB	10 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 10	8	8 GB	20 GB + 50 GB + 100 GB
MetaDefender Core MAX	16	16 GB	40 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core resides on.
- For better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.

[Linux] Third Party Dependencies

Dependencies list:

- openssl
- grep
- lib32stdc++6 (>= 4.5)
- libc6-i386 (>= 2.10)
- procps
- zlib1g
- libcurl3 (>= 7.19.7)
- libcurl4
- postgresql:
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Not all the above dependencies will need to be installed, it is dependent on different Unix distro and version

Some engines also have dependencies as described below:

Engine name	Dependency
Archive engine	• To extract password protected MS Office files, you need to install .NET 6 dependencies, libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • Rocky Linux 9.x/Red Hat 9.x: libnsl
Deep CDR	• Rocky Linux 9.x: libgomp, ncurses-compat-libs, libnsl • CentOS: libgomp, ncurses-compat-libs • Red Hat 8.x: libgomp, ncurses-compat-libs, libnsl • Red Hat 9.x: libgomp, ncurses-compat-libs, libnsl, compat-openssl11 • Ubuntu/Debian: libgomp1, libncurses5 • libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • .NET 6 dependencies
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
FileType engine	Rocky Linux 9.x/Red Hat 9.x: libnsl
Proactive DLP	libgdiplus, mscorefonts .NET 6 dependencies CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
RocketCyber	libgomp

Note: Some dependencies may need to be installed from external repositories, not from the OS default repositories.

[Linux] Installation Details

MetaDefender Core default installation path is using /var folder for storing resources:

- `/var/lib/ometascan` : installation folder with all its resources (database, DLP processed / CDR sanitized / quarantined file storage, engine package and definition updates).
- `/var/log/ometascan` : application logs.
- `/var/run/ometascan` : runtime folder for Core related processes.
- `/var/lib/ometascan` : all successful deployed engines data files.
- `/usr/lib/ometascan` : all shared libraries required by MetaDefender Core, bundled PostgreSQL server related binaries and libraries.
- `/var/tmp/ometascan/resources` : all temporary processing files for MetaDefender Core to process.

Storage Usage Information

Based on the configuration, MetaDefender Core will need additional disk space to store analysis/result data, quarantined files and temporary files:

	Data Storage
Analysis Reports (Processing History)	Approximately 4.5GB is required for every 1M analysis reports stored. • 100 GB is recommended as a baseline. Database storage requirements are affected by Data Retention settings, engine settings and scanning load.
Quarantined Files	Depends on the customers' dataset and configuration
Sanitized Files	Depends on the customers' dataset and configuration
Temporary processing files	Depends on the customers' dataset and configuration • A 1GB non-archive file requires 1GB free disk space at least. • A 1GB archive file requires (1 * 2)GB free disk space at least. • 50GB is a baseline and should be adjusted based on observation of the solution within the environment, with consideration to peak times and overall file makeup (e.g. complexity, size, archives).
Note	Options are available to control how large and/or complex files are handled to limit the amount of system resources required. See Workflow Configuration Template for more information.

Custom Engines

The recommendations above are specific for MetaDefender pre-packaged bundles.

However for additional Custom Engines, please review the Knowledge base to review additional requirements (if any) for the selected engine.

Browser Requirements for the MetaDefender Core Management Console

One of the following browsers is suggested to view the MetaDefender Core Management Console:

- Microsoft Edge
- Chrome
- Firefox
- Safari

Chrome, Firefox, Safari and Microsoft Edge browsers are tested with the latest available version at the time of release.

Network Bandwidth

In online environment where MetaDefender Core is configured to download engine update packages directly from OPSWAT update server source, then using gigabit network speed (125 MB/sec) is highly recommended to avoid potential network related issue while downloading files, or at least 100 Mbps (12.5 MB/sec) network speed.

Allowlisting URLs

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts':

- https://activation.dl.opswat.com - for license online activation
- https://update.dl.opswat.com - for engines and databases download

Note: *IP address-based allowlisting on your firewall might fail after some time since OPSWAT uses CDN (AWS Cloudfront) to faster delivery updates over the world, and IP address of edge servers might change over time.*

Depending on your location, connections with OPSWAT cloud-based servers be forwarded to specific AWS Cloudfront's edge locations, each is assigned with its own unique IP address ranges. You might also need to allow correct AWS Cloudfront's corresponding IP address ranges by referring to https://ip-ranges.amazonaws.com/ip-ranges.json

Warning

Even the OPSWAT update servers host updates for all of the available engines we support, sometimes custom engines might try to connect to their own cloud for updates, but this can be disabled in firewall and they will be updated just from OPSWAT cloud servers instead.

Differences Between MetaDefender Core v4 and v5

We are very excited to inform you about the release of version 5.0.0 of our MetaDefender Core!

The new version is designed to enhance the product performance, user experience, as well as to provide more flexibility, security, and scalability for our customers.

At the same time, we are also committed to keeping backward compatibility for whom running MetaDefender Core v4, and avoid the need for updating your client integration to move forward with MetaDefender Core v5.

v4 REST API Integration Supported
All public REST API endpoints available on MetaDefender Core v4 will be continously supported on v5.

v4 Product Major Features Supported
All product major features available on MetaDefender Core 4.21.2 (the latest one on v4) will be continously supported on v5.

Seamless Product Upgrade
Upgrading from MetaDefender Core v4 to v5 will be supported and covered with data migration without additional manual steps from users.

Licensing information will be also migrated.

Below are some highlights of the first v5 release, MetaDefender Core 5.0.0:

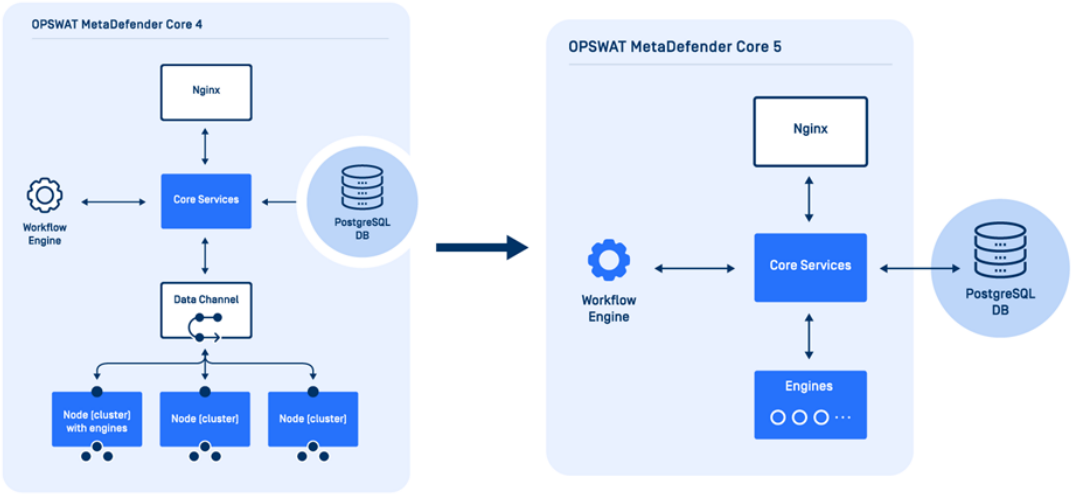
New product architecture

This version introduces a simpler product architecture that speeds up integration and automation, file processing flow, and immensely enhance the product performance and efficiency.

Earlier versions utilized MetaDefender Node concept, designed to support scaling of the scanning infrastructure by distributing scan requests among several different processing Nodes. Product licensing was based on number of Node instances. The benefit of having such a distributed infrastructure is that based on Node loads, Metadefender Core server can always choose the most appropriate Node to assign a new scan task to. However, having nodes in the v4 architecture caused more complexity and possible latency in the processing flow and entailed maintenance cost.

In this release, we retired multi-Node deployment and restructured MetaDefender Core's backend. This alteration significantly improves file processing speed and product efficiency, as well as eliminates extra cost for maintaining node services. Users now can simply install additional MetaDefender Core instances for scalability requirement,

as well as configure their own load balancer to coordinate tasks among multiple MetaDefender Core instances. Accordingly, product licensing is now based on number of the product instances.



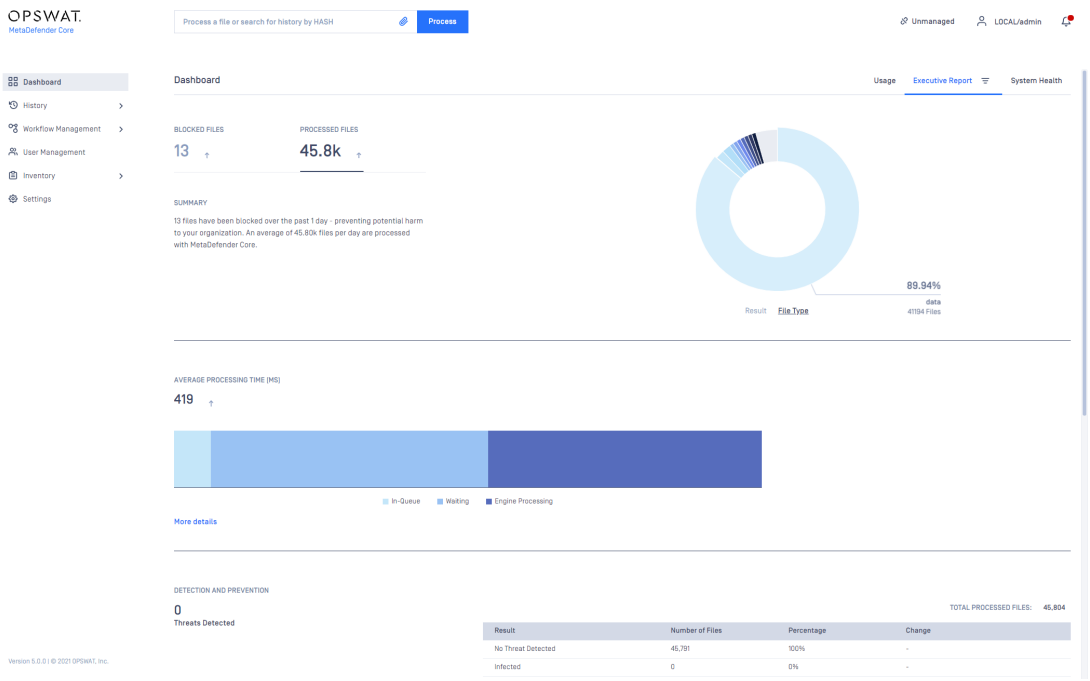
Please be informed that

- All existing administrative APIs related to "Node" on v4 will be kept as it is on v5 to ensure backward compatibility with existing client integrations.
- Applicable settings defined in Windows registry (Windows) or ometascan-node.conf config file (Linux) for Node will be migrated to Core. Details: MetaDefender Configuration

Brand-new product user interface

The second notable update in this version is a brand-new UI for the management console, which was designed with the focus on the quality and thoughtfulness of the user experience. The new administrative console UI provides an advanced set of Core features and observable processing status.

Comprehensive executive dashboard reporting offers dynamic, real-time and actionable threat insights. It also provides visibility of your security posture allowing you to paint a picture of security that aligns with the overall business. The report includes easy-to-read charts, graphs, and statistic explanation that help your security professionals reduce clutter, gather information quicker, spot threat activity early and act swiftly.



We're continuing to improve our product UI to provide users an excellent experience with the application and overall usability. Please expect more powerful UI features coming in next releases.

New compression archive engine for archive sanitization

In previous version, MetaDefender Core's archive engine implemented four functions including decryption, extraction, encryption and compression that with a high volume of files it takes time to process them all. For the purpose of enhancing archive handling performance, this release comes with a separated compression archive engine, which dedicatedly processes archive sanitization (compression and encryption). It is also created to facilitate maintenance and further feature development.

Archive sanitization feature now requires Deep CDR license

As to respect sanitization practice and ensure unknown threats prevention as designed, now starting on version 5, it's mandatory to have Deep CDR licensed for archive sanitization feature enablement on MetaDefender Core workflow configurations.

Archive handing related engines changed

The legacy "Archive" engine on MetaDefender Core v4 will be changed its name to "Archive Extraction" engine, even when you are still on MetaDefender Core v4.

A brand new "compression archive engine" is to dedicatedly handle archive sanitization in v5 (archive compression), that helps relevant archive handling performance improvement, easier for maintainance and further feature development.

Notes when you are still on MetaDefender Core v4 (not upgraded yet)

The legacy "Archive" engine (now name changed to "Archive Extraction" engine) on MetaDefender Core v4.x will still cover both extraction and compression archive tasks as usual.

Even though your MetaDefender Core v4.x might now have a new "Archive Compression" engine auto downloaded, but that engine cannot be deployed (marked as "permanently failed" in relevant engine status check API). This behavior is expected, and not causing any impact to MetaDefender Core processing flow, or its performance though.

- Just in case your client integration has a check for engine status before triggering scan request, then you might want to disable "Archive Compression" engine status on MetaDefender Core v4 using REST API. Detailed instruction: Archive Compression Engine Permanently Failed on MetaDefender Core v4
- When you later upgrade to MetaDefender Core v5 from v4, then you might need to manually re-enable "Archive Compression" engine to make it active and running on MetaDefender Core v5.

Notes for licensing

For offline deployment (MetaDefender Core is not connected to the Internet directly for online update), you will need to re-activate again on OPSWAT portal to retrieve latest engine list in .yml file including "Archive Compression" engine if you are currently licensed with Deep CDR.

Meanwhile, for online deployment, this manual procedure is not required since MetaDefender Core will cover that itself.

Import & export configurations

Not only workflow settings like in previous versions, but also all other general settings (update, security, data retention, server, hashes to skip) and most importantly, user settings, will be able to export and import on MetaDefender Core version 5.

Learn more at Import/Export configuration

Backward compatibility supported

The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

Windows application event logging disabled by default

Applicable only to Windows OS based MetaDefender Core.

Help avoid duplicated logging with separate file option, and improve overall processing performance. Of course, when being still valid to you, you are still supported to enable it back.

CSRF token header renamed

CSRF token originally created for security strengthening and being used by MetaDefender Core management console UI since version 4.21.0 via header `oms_csrf_token`

To get it compliant with some load balancers which do not accept underscore in header name by default, and save you from additional load balancer configuration step, we have now changed the header name to `oms-csrf-token`

This change does not impact to any REST API based client integration.

More comprehensive and accurate error messages on REST APIs

Impacting on following REST API endpoints:

- `PUT /admin/config/session`
- `PUT /admin/config/update`
- `PUT /admin/config/scan`
- `PUT /admin/config/skip`
- `PUT /admin/config/auditlog`
- `PUT /admin/config/webhook`
- `PUT /admin/config/rule`

Upload time and queue time logic updated in JSON scan response

Impacting on following REST API endpoints:

- `GET /file/{data_id}`
- `GET /hash/{md5/sha1/sha256}`
- `GET /archive/{data_id}`
- `POST /file/{async}`

Added new field:

- `receive_data_timestamp`: The timestamp when upload progress started (first byte received) (in milliseconds)
- `upload_time`: Total time elapsed for upload process (in milliseconds)

Modified existing field (new logic):

> **Backward compatibility supported**
> The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

- `upload_timestamp`: The timestamp when upload progress finished (all bytes received) (in milliseconds)
- `queue_time`: Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue time = start_time - upload_ timestamp) (in milliseconds)

Release version 5.0.0 also comes with variety of outstanding features and improvements.

For detailed information about our latest release please view our Release notes

Installation

Starting with MetaDefender Core 4.20.0, the centralized/shared database deployment model is supported (all MetaDefender Core instances share the same database), along with traditional standalone database mode (every MetaDefender Core instance has its own database to manage).

- Standalone mode
- Shared Database Mode

Installation

1. Download the package of your choice from My OPSWAT
2. Install the package on your computer via the Command Line or via the Wizard
3. Open a web browser and point to http://<server name or IP>:<port>
    - The default port is **8008**
    - In case of problem check Inaccessible Management Console page
4. Complete the required steps of the basic configuration wizard
5. You must activate this deployment to use its features

Installation notes

- If the MetaDefender Core package **dependencies are not installed** on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.
- During installation the databases might need to be upgraded. This could take noticeable time depending on database size (eg.: length of scan history).

Standalone mode

Wizard Installation

The Install Wizard is only for the Windows installer (.msi file).

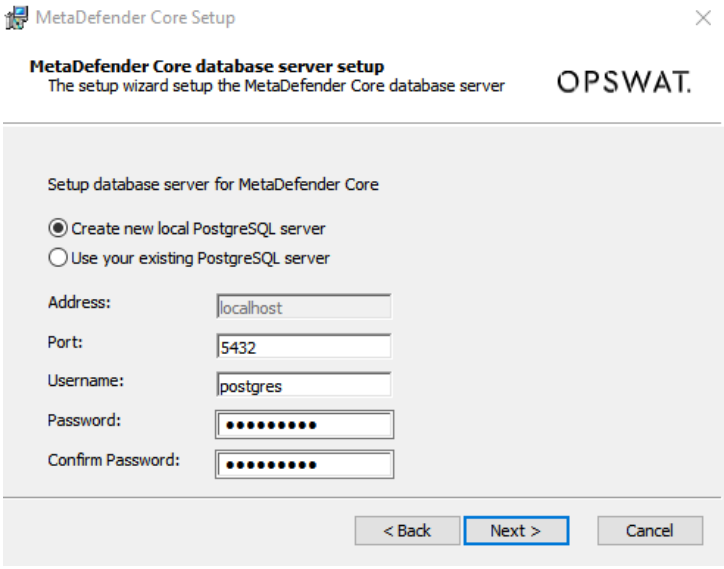
Since MetaDefender Core 4.19.0, we have switched to using PostgreSQL DBMS which needs a dedicated database server. MetaDefender Core supports two modes to setup PostgreSQL server:

Local (Bundle) PostgreSQL server

MetaDefender Core will install a new PostgreSQL server locally in the same box with the product.

> **PostgreSQL user credentials**
> Unicode characters are NOT supported for "Username" and "Password"
>
> "Password" does not accept # (hash) as starting character (e.g. #ABC)

> **The ignition file**
> If the ignition file exist in the designated location, then MetaDefender Core will use the details defined in the ignition file, rather than what is passed via the installation wizard.
>
> More info about the ignition file based installation: Command Line Installation

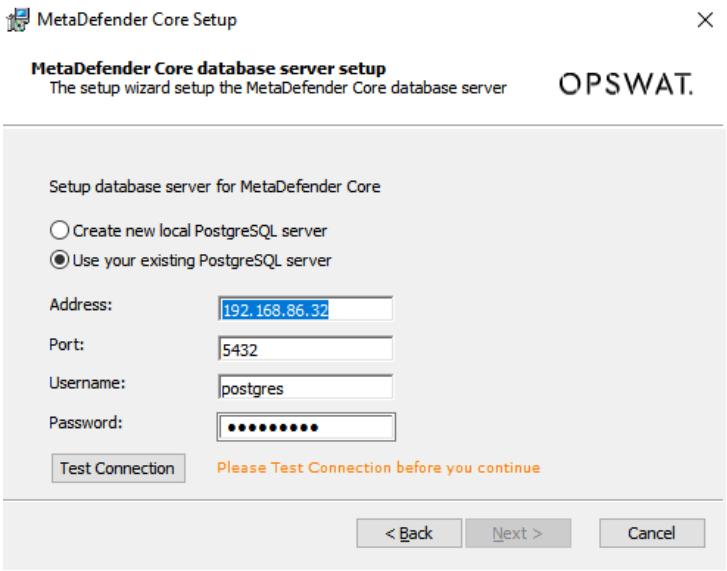


Remote (Existing) PostgreSQL server

MetaDefender Core will connect to remote pre-installed PostgreSQL running remotely.

> **PostgreSQL connection configuration**
> "Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

> **PostgreSQL user privilege requirements**
> MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.
>
> Details: PostgreSQL User Privilege Requirements



Command Line Installation

Ignition File

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Make sure to create the ignition file, if it doesn't exists before you go ahead and install MetaDefender Core via command line.

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

MetaDefender Core supports two modes to setup PostgreSQL server:

1) Local PostgreSQL server

- MetaDefender Core will install a new PostgreSQL server locally, next to the product.
  - A sample iginition file for PostgreSQL server information (***Only non-Unicode characters supported for "user" and "password"__***):

ometascan.conf bash

```
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
```

2) Remote PostgreSQL server

> **Info**
>
> **PostgreSQL version 12 (at least 12.3) or version 14 is required.**
>
> According to PostgreSQL notice, version 12 will stop receiving fixes on November 14, 2024, so we suggest that you use version 14.

MetaDefender Core will connect to a pre-installed PostgreSQL server running remotely and will setup its database.

Make sure to follow steps at Open Connection On PostgreSQL Server first, so that your remote existing PostgreSQL server to accept upcoming connection from MetaDefender Core.

A sample ignition file for PostgreSQL server information:

ometascan.conf bash

```
[dbserver]
type=remote
host=192.168.86.32
port=5432
user=existingserver_admin_user
password=existing_server_pass
```

After installing MetaDefender Core successfully, you may want to remove all credentials info created for [dbserver] section in ignition file for security reason.

> **Warning**
> If the MetaDefender Core package dependencies are not installed on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.

Debian package (.deb)

bash

```
sudo dpkg -i <filename> || sudo apt-get install -f
```

On Red Hat Enterprise 7.x Linux / CentOS package (.rpm)

bash

```
sudo yum install <filename>
```

For systems which enabled GPD check flag:

bash

```
sudo yum install --nogpgcheck <filename>
```

Red Hat Enterprise Linux 8.x package (.rpm)

Install the latest PostgreSQL repository RPM:

> **Note**
> Due to the shorter support cycle on Red Hat Enterprise Linux, all supported versions of PostgreSQL are not available on this platform. The PostgreSQL Yum Repository will integrate with the normal systems and patch management, and provide automatic updates for all supported versions of PostgreSQL throughout the supported lifetime of PostgreSQL. Reference: https://www.postgresql.org/download/linux/redhat/

bash

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

In case installing a remote PostgreSQL on either the same MetaDefender Core server or another server, you need to install the following additional package. PostgreSQL version 12.3 or newer is recommended.

bash

```
sudo dnf install -y postgresql12-contrib
```

Install MetaDefender Core:

bash

```
sudo dnf install -y <filename>
```

Windows package (.msi)

On Windows systems it is possible to install the product by running the corresponding .msi file.

From command line interface it is also possible to install the product by executing

bash

```
msiexec /qn /i <msi file name> <option key>=<option value>
```

where the possible keys and their default values are the following:

Key	Default Value	Description
INSTALLFOLDER	\Program Files\OPSWAT\MetaDefender Core	Customize installation folder for product Example: `INSTALLFOLDER="D:\Products"`
RESTADDRESS	*	REST interface binding IPv4 or IPv6 address ('*' means that service listens on all IPv4 and IPv6 interfaces)
RESTPORT	8008	REST interface binding port
REPORT_ENGINE_ISSUE	true	Enable reporting of engine issue count. (possible values: "true" or "false").

Note

Reporting of engine issue count

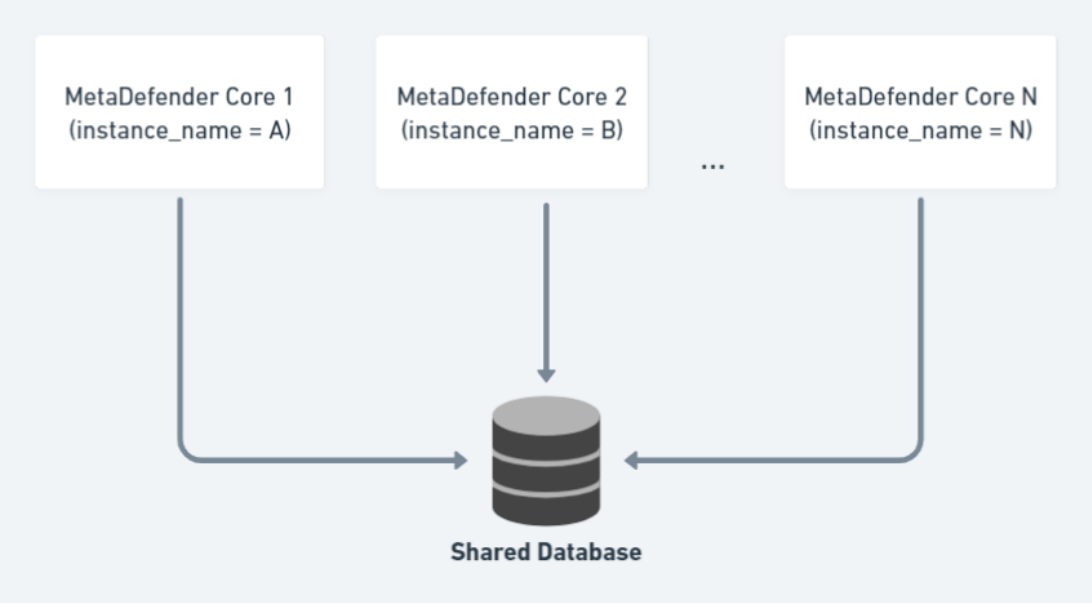
If reporting of engine issue count is enabled, MetaDefender Core server will send only the **number** of initialization errors and **number** of unexpected stops for the specific db/engine version. This information is sent over a HTTPS channel when the product downloads the latest package descriptors. This information is used for early detection of any specific 3rd party engine quality issues.

For details on using msiexec please consult Windows installer documentation.

Shared Database Mode

Starting with MetaDefender 4.20.0, you can use a centralized database deployment model, where all MetaDefender Core instances can share the same external database.

That helps auto-sync all product configurations (**except Single-Sign-On** that requires unique settings for each Core), processing history, live/on-going scan result query, audit, statistics data.



This article demonstrates how to install and setup MetaDefender Core 4.20.0 or newer with shared database mode.

Important notes & current limitations:

Currently not supported

- Does not support Central Management use-case with shared DB mode.
- Does not support to switch from shared database mode back to standalone database mode.
- Does not support to change instance_name of MetaDefender Core in shared database deployment model after installation.
- Does not support upgrading older MetaDefender Core from non-persistent mode to using the shared database mode.
- Does not support upgrading older MetaDefender Core from standalone mode to using the shared database mode.
- Does not support to upgrade multiple MetaDefender Core instances at once in shared database mode
    - Stop all Core instances' scanning service before performing the upgrade.
    - Upgrade each Core server one by one instead.

- For each Core, when the upgrade finished, then you can start using that Core for scanning without waiting for the other Core instances' upgrade to be done.

> **Disclaimer**
> MetaDefender Core does not support mode switching.

Requirements for shared database setup:

- The central remote PostgreSQL server (for shared database) could be utilized with a built-in (native) PostgreSQL database server installed with one of the MetaDefender Core instances in the network. Or using a dedicated existing PostgreSQL database server (see detailed requirements for PostgreSQL version at Recommended System Configuration).
- Before setting up, make sure to follow steps at Open Connection On PostgreSQL Server so that all MetaDefender Core instances could connect to the central remote PostgreSQL server.
- MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core, details: PostgreSQL User Privilege Requirements
- All MetaDefender Core instances must be installed on the same Operating System (Windows / Linux).
- The first MetaDefender Core instance connects to the shared database must be ready first (license activated, users/groups/roles created) before setting up other MetaDefender Core instances to connect and use that shared database.

Engine update:

- If engine updates is configured to be via Folder source, all MetaDefender Core instances must be able to access to the configured update folder, and the option DELETE FILES AFTER IMPORT will be unchecked and prevented for modification.

Authentication:

- Single-Sign-On (SSO) authentication settings (if using) cannot be shared among MetaDefender Core instances because SSO requires an unique login URL for every different Core.
- If an user is locked for login on a certain MetaDefender Core instance(e.g. exceeded the number of failed login times), that user is not prevented for logging in on any other instances.
- If an user is locked for sign-in on a certain MetaDefender Core instance (e.g. exceeded the number of failed login times), if the admin releases lockout on that user on that MetaDefender Core instance, that user is still not able to login on other MetaDefender Core instances.

Sanitized / DLP / Quarantined location changed after MetaDefender Core install:

Windows:

Shared storage path will be stored in 3 strings under key global in registry: quarantinepath, dlppath, sanitizepath as below

(Default)	REG_SZ	(value not set)
dbmode	REG_SZ	4
dlppath	REG_SZ	\\10.40.50.100\test
instance_name	REG_SZ	481rvj
quarantinepath	REG_SZ	\\10.40.50.100\test
restaddress	REG_SZ	*
restport	REG_SZ	8008
sanitizepath	REG_SZ	\\10.40.50.100\test

You can set either 1, 2 or 3 different paths for the 3 settings above (means that quarantined, DLP and sanitized files will be stored in same or different storages corresponding). In case shared storage is not set in installation, values of 3 settings above will be empty.

Restart MetaDefender Core service after changed.

Linux:

Open file `/etc/ometascan/ometascan.conf` and set shared storage path for 3 keys quarantinepath, dlppath, sanitizepath under section [global]

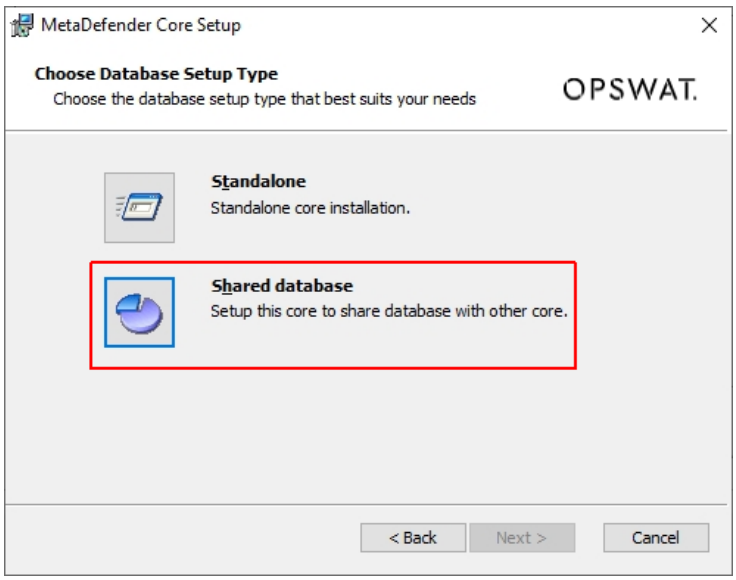
Restart ometascan service after changed.

Wizard Installation (Windows Only)

> **Info**
> Only applicable to MetaDefender Core 4.20.0 or newer using wizard on Windows OS.

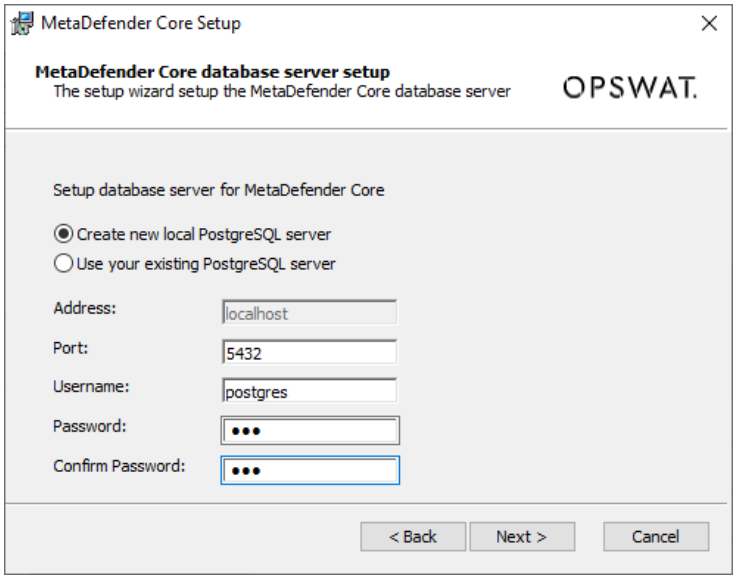
Clean install scenario

1. **Choose Shared database option:**

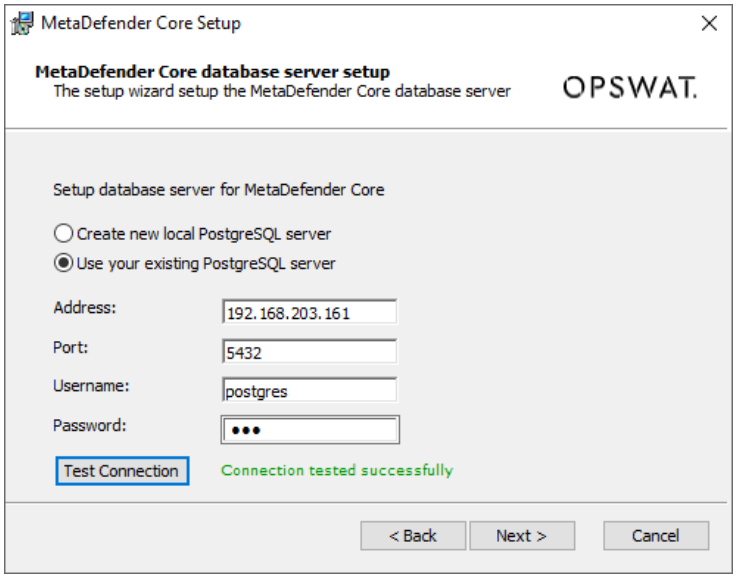


2. **Setup database server for MetaDefender Core:**

You can choose to create a native / built-in PostgreSQL server and setup the shared database on (then other MetaDefender Core instances will want to connect to that PostgreSQL server afterwards)



Or choose to connect to a pre-setup existing PostgreSQL server:



PostgreSQL connection configuration

"Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

PostgreSQL user privilege requirements

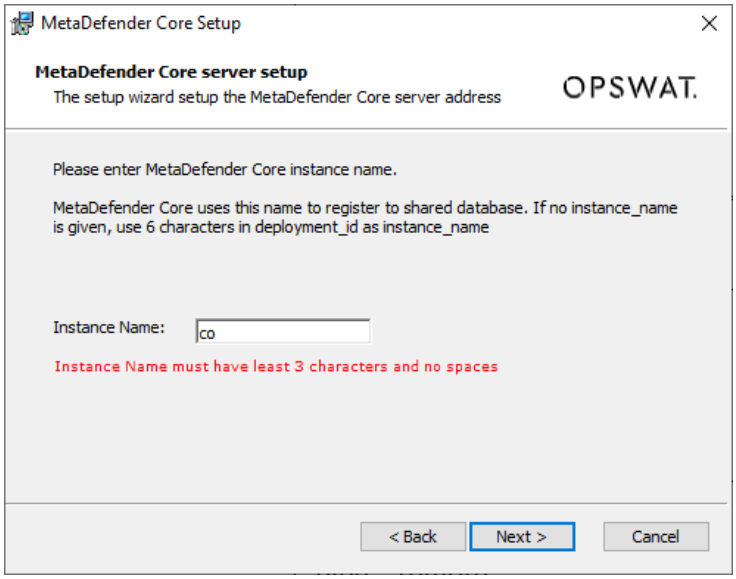
MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.

Details: PostgreSQL User Privilege Requirements

3. **Create an unique instance name for that MetaDefender Core (different from other MetaDefender Core instances):**

Instance name must contain at least 3 characters and no white space.

If you choose to leave it blank, which means it will use the default instance name with 6 characters extracted from that MetaDefender Core deployment ID.

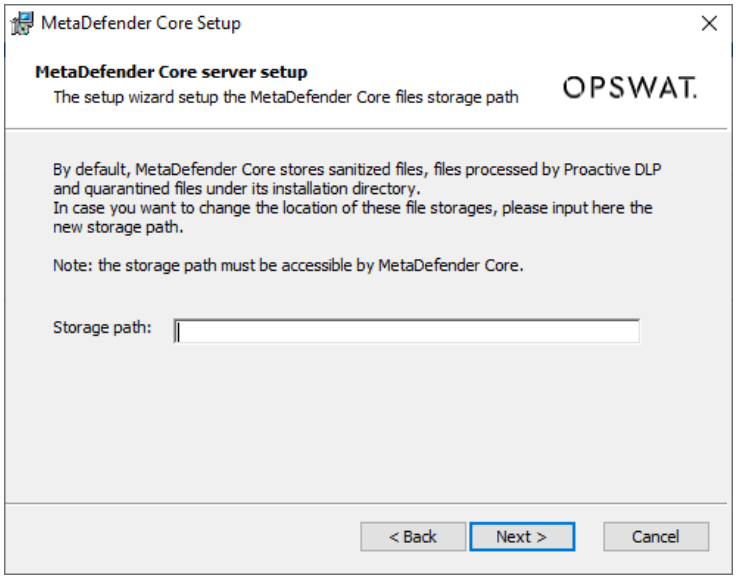


4. **Input the file storage path (sanitized, DLP, quarantined files)**

In the shared database model, all MetaDefender Core instances could use the same folder path to store all their own sanitized, DLP, quarantined files after processed.

The default location is: ***<MetaDefender Core installation folder>\data\ {sanitized|dlp|quarantine}***

If you wish to use that default location, just leave the input box blank and press Next.



Otherwise, you can specify a different location of sanitized/dlp/quarantined files. It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core.

For example:

- C:\new_folder
- \192.168.1.1\new_folder

Command Line Installation

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Clean install scenario

1. Create ignition file if not existed

- Windows: **C:\OPSWAT\ometascan.conf**
- Linux: ***/etc/opswat/ometascan.conf***

An example of ignition file:

ometascan.conf bash

```
eula=true

[global]
dbmode=4
instance_name=MetaDefenderCoreA
storage_path=D:\path\

[user]
name=admin
password=admin
email=admin@local

[dbserver]
type=remote
host=192.168.1.1
port=5432
user=postgres
password=your_own_password
```

Field	Required	Mapping to wizard UI	Possible values
global/dbmode	optional default is 1 (standalone)		1 - Standalone 2 - Central Hub deployment 3 - Non-persistent **4 - Shared**
global/instance_name	optional default is 6 characters of deployment ID		Unicode and number At least 3 characters No whitespace
global/storage_path	optional default is empty		Path to storages (DLP, quarantined, sanitized files) It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core. For example: • C:\folder • \192.168.1.1\folder

2. Run installers using command line to install

Please refer to Command Line Installation for installation command.

Deployment automation support

Note

An Ignition file could be utilized for either / both of following use-cases:

1. Pre-define PostgreSQL server information for MetaDefender Core to connect and setup database: only required on Linux installation, or Windows with command line installation. Learn more details at:
    a. Standalone Mode > Command Line Installation (applicable on version 4.19.0 or newer).
    b. Shared Database Mode > Command Line Installation (only applicable on version 4.20.0 or newer).
2. Automating to accept EULA, create a default local admin user and import MetaDefender Core configurations upon installed: Those steps could be automated using ignition file described in this page.

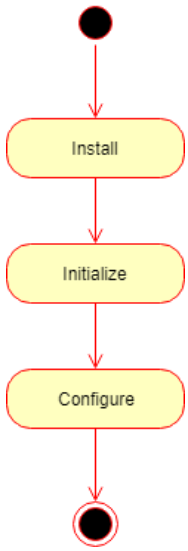
Ignition file location to create:

- **Windows**: `C:\OPSWAT\ometascan.conf`
- **Linux**: `/etc/opswat/ometascan.conf`

The product supports fully automated deployment. It means that it can be installed and configured with no human interaction.

The automated deployment can be split to three steps on a high level:

1. Installation
2. Initialization
3. Configuration



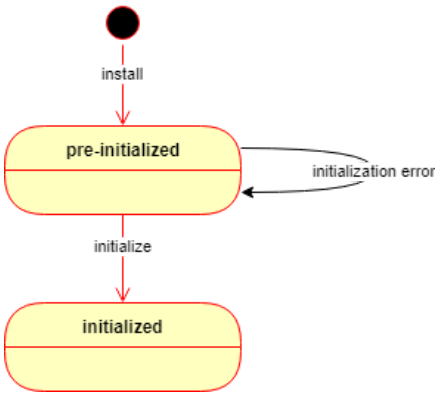
Installation

To automate the installation, install the product from the command line and provide the installation-time options as parameters to the installer. For further details see Command Line Installation

After the installation is complete, the product starts up and waits in a pre-initialized status. The product may be initialized in two ways:

1. Manually using the Wizard Setup or
2. Automatically using an *ignition file* (see below).

> **Warning**
> If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried after fixing the problem (e.g. placing the ignition file in its lookup location) and restarting the OPSWAT MetaDefender Core service.
>
> Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

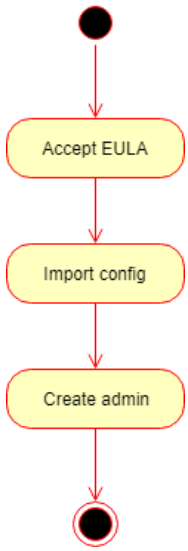


Initialization

Initialization is the process of bringing the product to an operable status.

Basically the initialization consists of the following steps:

1. Accept the End User License Agreement (EULA),
2. Import product configuration and
3. Create the first administrator user account.



Ignition file

The initialization process can be configured in a file called the *ignition file*.

> **Warning**
> In Linux, the ignition file and the folder /etc/opswat should have the permission 755

Ignition file location

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

The ignition file must be in conf format.

ometascan.conf bash

```
eula=true
[user]
name=admin
password=admin
email=admin@local
apikey=44c96cbb4e4e6c0a1d16bcb3c3da24def317
[config]
import=config_package.zip
import_password=123
[security]
session_cookie_samesite=Lax
[internal]
skip_startup_pages=true
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
private_username=my_internal_db_user
private_password=mypassword
[ocm]
ocm_url=https://my.opswat.com
reg_code=XXXXXXXX
```

Ignition file fields

The ignition file accepts the following fields:

Section	Key	Required	Accepted Values	Description
	eula	Optional (mandatory when having [user] section)	true false (default)	Whether to accept the End User License Agreement. This key must be set at the top of ignition file. This key must be set to "true" to accept the EULA. Any other value will cause the initialization to fail. When having [user] section defined, then this key must be set.
[user]		Optional		Initial administrator user account properties. The Administrator role is granted to the account.
	name	Optional	string	User name for the initial administrator user account.
	password	Optional	string	Password for the initial administrator user account. WARNING! Clear text password The password in this configuration file must be stored in its clear-text format and as so it may be visible for unauthorized parties.
	email	Optional	string	E-mail address for the initial administrator user account.
	apikey	Optional	string	API key for the initial administrator user account.
[config]		Optional		
	import	Optional	string	Path to a file in json/zip format that contains a previously exported configuration to be imported. Zip-format file contains a full configuration including user management settings, so it will overwrite [user] section.
	import_password	Optional	string	Password to decrypt .zip configuration package.
[security]		Optional		

Section	Key	Required	Accepted Values	Description
	session_cookie_samesite	Optional	Strict Lax (default) None	Declare if session cookie should be restricted to a first-party or same-site context.
[internal]		Optional		
	skip_startup_pages	Optional	true false (default)	Skip all welcome and other startup pages (e.g. upgrade result) when installing / upgrading MetaDefender, and going straight to the product index default page.
[dbserver]		Optional		
	type	Mandatory when having [dbserver] section	local remote	Where the PostgreSQL database server should physically locate. "remote" means PostgreSQL server and MetaDefender Core are not in the same machine.
	host	Mandatory when having [dbserver] section	string	IP address / domain name of the server where PostgreSQL server locates. "localhost" can be used when applicable.
	port	Mandatory when having [dbserver] section	number	Port of PostgreSQL server is listening for connections from clients (i.e. MetaDefender Core).
	user	Mandatory when having [dbserver] section	string	PostgreSQL server's user. SUPERUSER privilege is required for MetaDefender Core to setup its database and extensions for the first time. **Only non-Unicode characters supported.**

Section	Key	Required	Accepted Values	Description
	password	Mandatory when having [dbserver] section	string	PostgreSQL server's user credentials. **Only non-Unicode characters supported.**
	private_username	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
	private_password	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
[ocm]		Optional		
	ocm_url	Optional	string	URL to My OPSWAT server.
	reg_code	Optional	string	A registration code to enroll to My OPSWAT server.

Detailed initialization process

1) After the product has been started, it looks for the ignition file in the designated location.

Windows: C:\OPSWAT\ometascan.conf ** Linux: /etc/opswat/ometascan.conf

** 2) If an ignition file is found, then

2.1) It gets validated, and if it is valid, then based on the information found in the ignition file:

- The EULA is accepted,
- The configuration is imported,
- The administrator account is created.
- If any of the above steps fails, then the error is logged, and the initialization gets terminated.

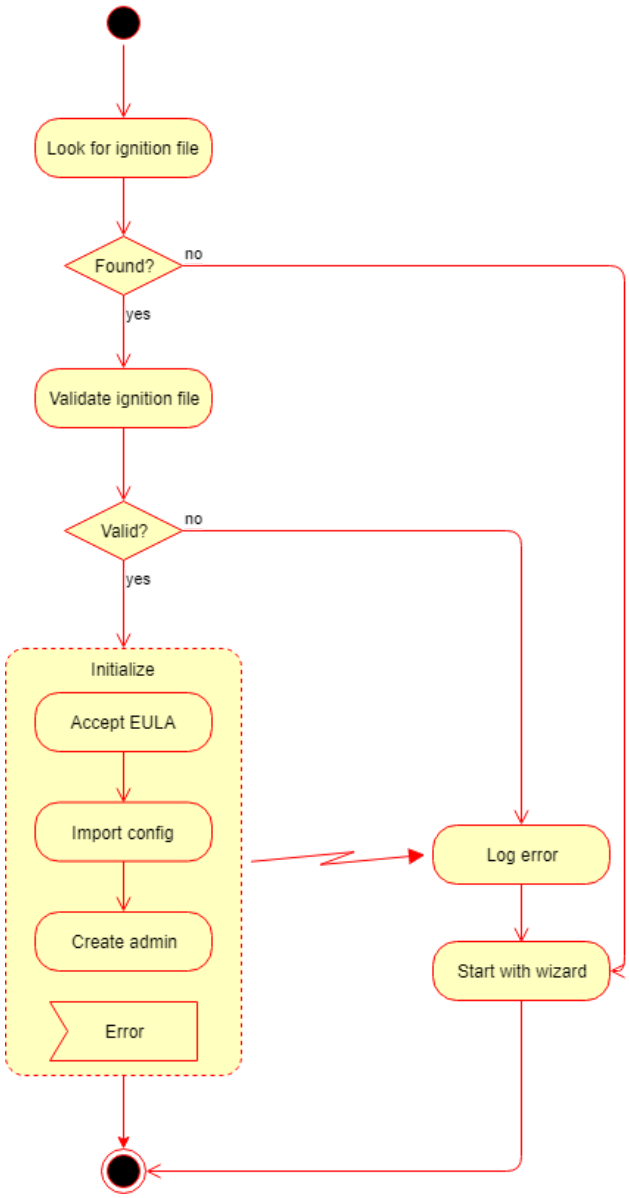
> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

2.2) If it is not valid, then the error is logged, and the initialization gets terminated.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

3) If there is no ignition file, then no initialization is performed.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.



Warning

If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried fixing the problem (e.g. placing the ignition file to its lookup location) and restarting the OPSWAT MetaDefender Core service.

Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

Configuration

After the initialization is complete, the product is ready with the default and the imported configuration.

This configuration can be later changed calling the configuration API functions. For further details about the API see Configuration related APIs API

CIS Level 1 Guidelines

Support Red Hat Enterprise Linux 9
For more details about Center for Internet Security (CIS) please refer to this document

Instruction steps

I. Install OpenSCAP

bash bash

```
yum install openscap-scanner scap-security-guide
```

II. Generate a result file and a HTML report using OpenSCAP scanner tool

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

III. Remediation of CIS Level 1 issues

Generate a remediation script based on the `ssg-rhel9-ds.xml` file:

bash

```
oscap xccdf generate fix --profile
xccdf_org.ssgproject.content_profile_cis_server_l1 --fix-type bash --output
remediations.sh /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

And execute remediation script.

bash

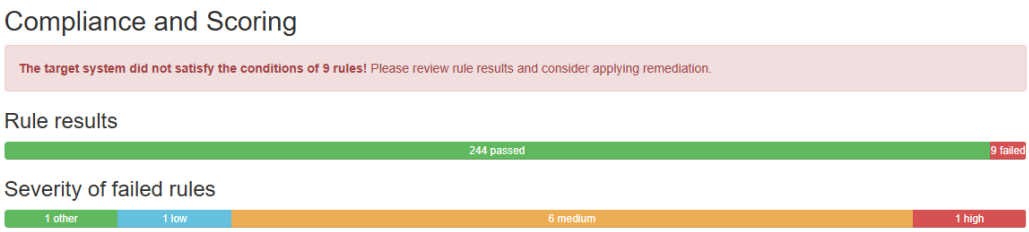
```
./remediations.sh
```

IV. Review the results after remediation

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

Example:



Notes

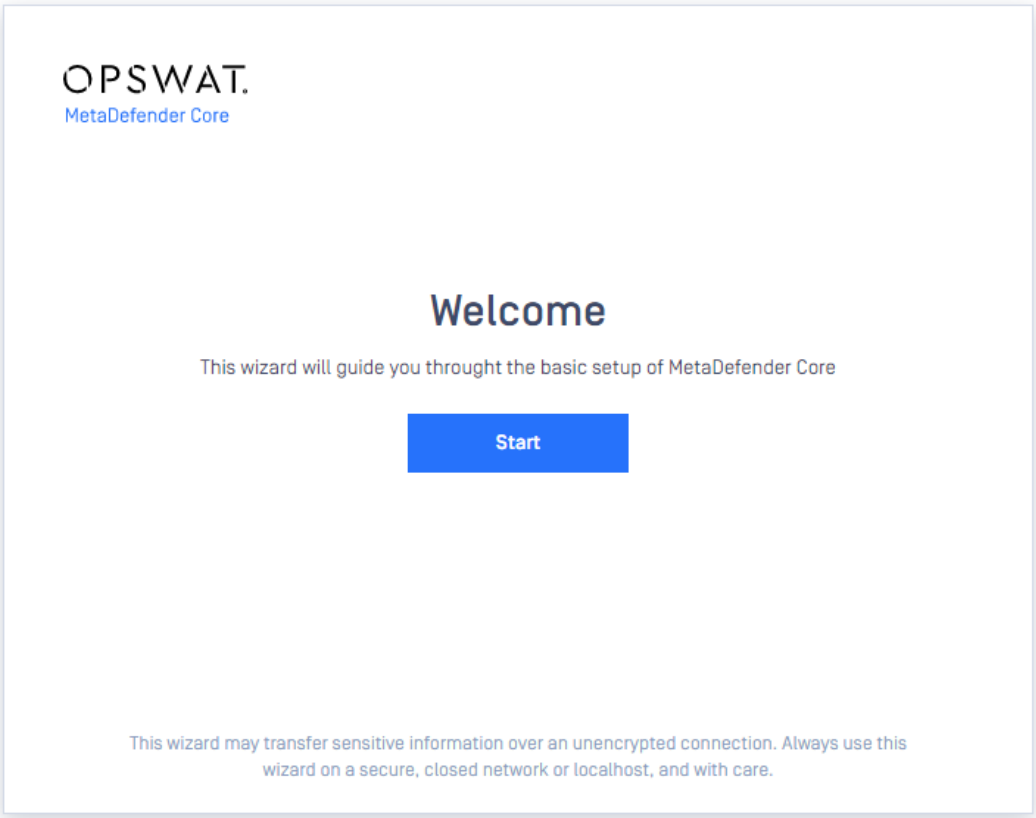
- CIS Level 1 requires `/tmp` folder to be mounted in a separate partition. Please ensure that that new partition have enough disk space for MetaDefender Core to run.
- CIS Level 1 requires that "Ensure No World-Writable Files Exist".
    - For now, when freshly installing MetaDefender Core, all its binary files meet the requirement.
    - When installing/updating engines, some engines might create additional files for its operation, and it might violate this requirement. In this case, you need to again execute the remediation script in the step III.

Wizard Setup

Introduction

Access the MetaDefender Core management console for the first time (i.e. http://localhost:8008), you are to complete a quick walkthrough for wizard setup in order to be able to use the product.

The administrative management console will be available only upon you have successfully finished this wizard stage.

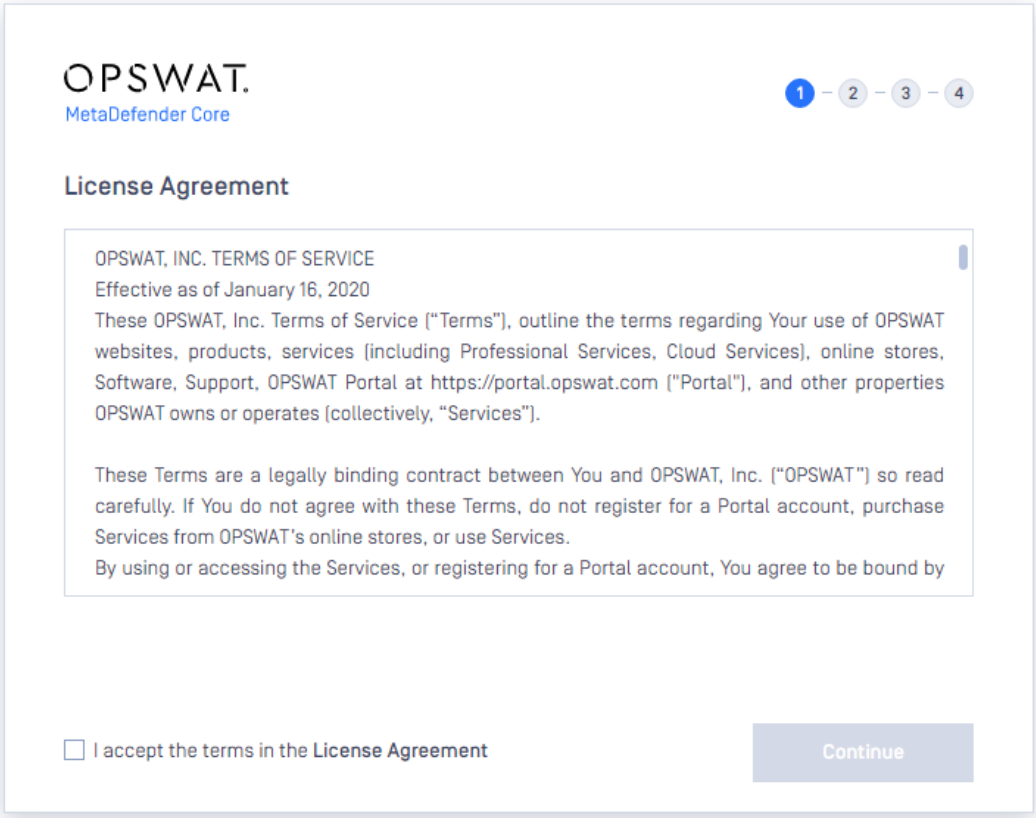


Warning

This wizard may transfer sensitive information over an unencrypted connection. Always use this wizard on a secure, closed network or localhost, and with care.

Basic configuration steps

End-User License Agreement



In the first page you can find the End-User License Agreement. You have to accept the terms before moving on.

Please read through the EULA (https://docs.opswat.com/policies/privacy-policy/portal-privacy-policy) carefully and if you agree with it, check `I accept the terms in the **License Agreement**` and click `Continue` to proceed.

Local Admin User Setup

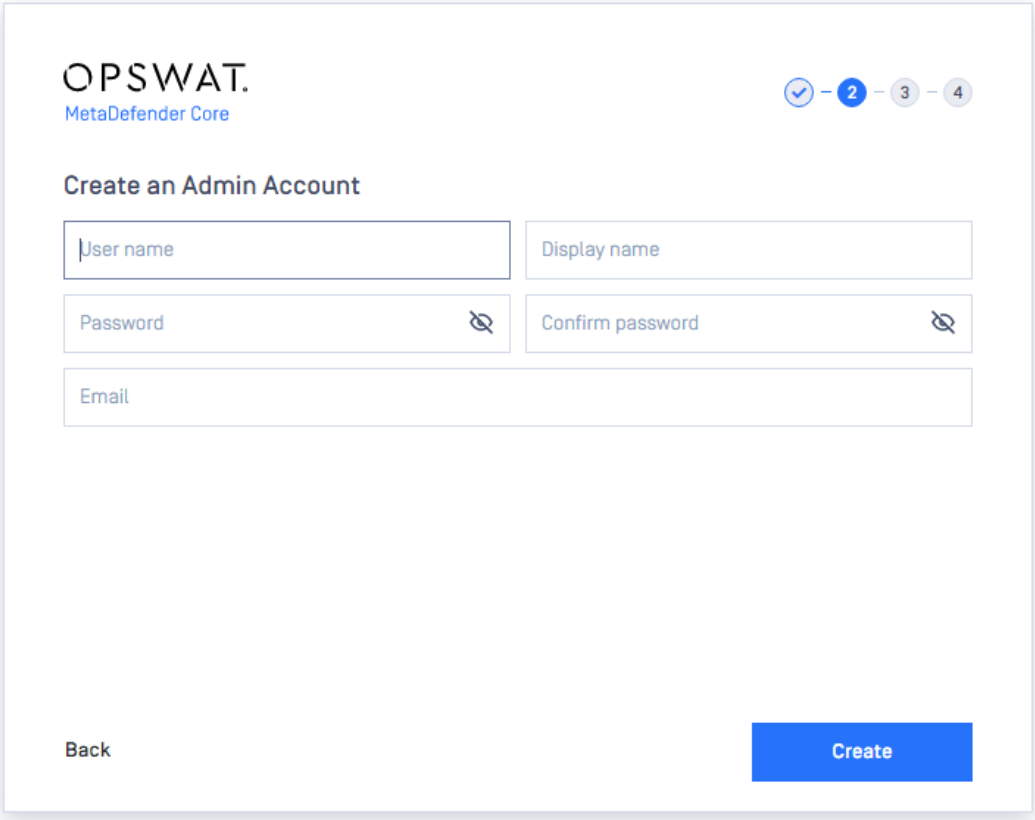
The next step is to set up a default local administrator account.

This account will be the first one being able to access the administrative management console, and to create accounts for other users. You have to fill out all fields in this page to be able to move forward. When you are done, click NEXT to continue.

> **Info - User Directory**
> The administrator account, that is created via the basic configuration wizard, is always added to the LOCAL user directory as a member.
>
> Learn more about user directory: User directories

The following information is required for the administrator account:

ACCOUNT NAME	The unique name of the account that is used at the time of login and in log messages for accountability.
ACCOUNT DISPLAY NAME	Name of the person bound to this account. This name (appended to the name of the account's user directory) is displayed in the top right corner of the administrative management console.
PASSWORD	Password of the user bound to this account that is used at the time of login. Passwords sent clear-text As long as TLS is not configured for the basic configuration wizard, passwords are sent clear-text over the network and may be disclosed to unauthorized parties. As a mitigation action: 1. Either use the wizard on *localhost* or on a direct network connection, or 2. Enabling HTTPS as soon as possible and change the password immediately if it has already been set.
EMAIL	Email address of the person bound to this account.



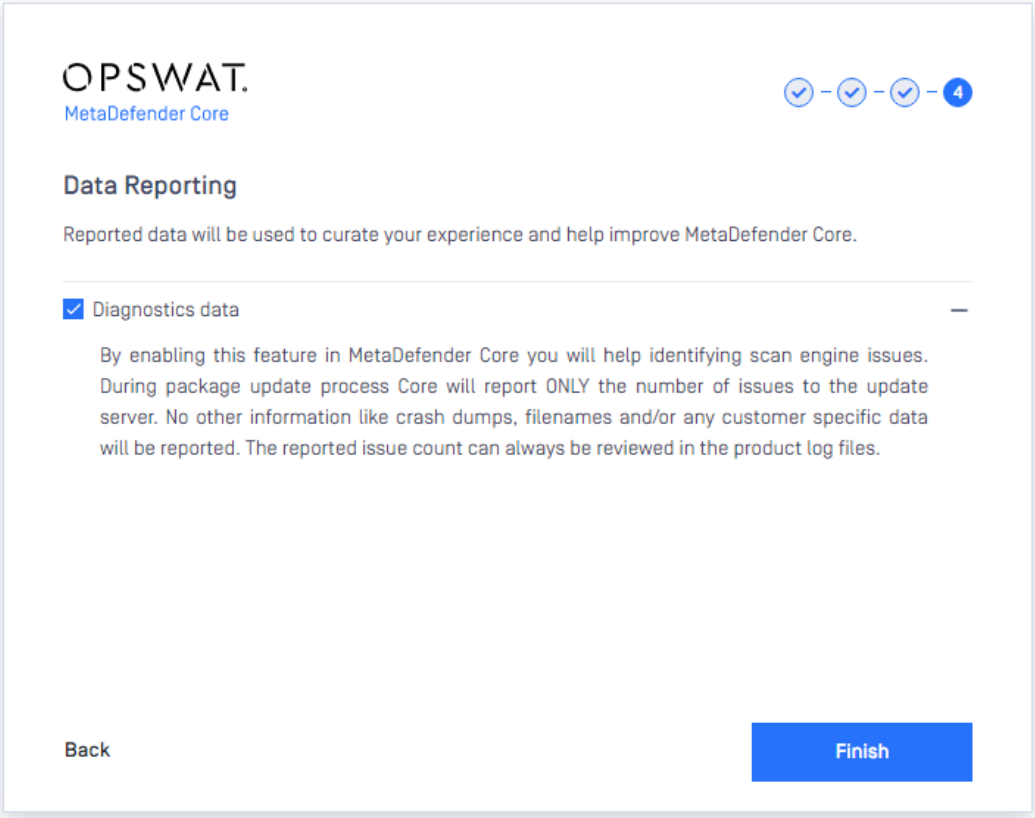
License activation

For license activation details see License Activation

Data Reporting

Enabling this option to allow MetaDefender Core collect its own engine issues and report back to OPSWAT for further analysis.

MetaDefender will not collect any other information.

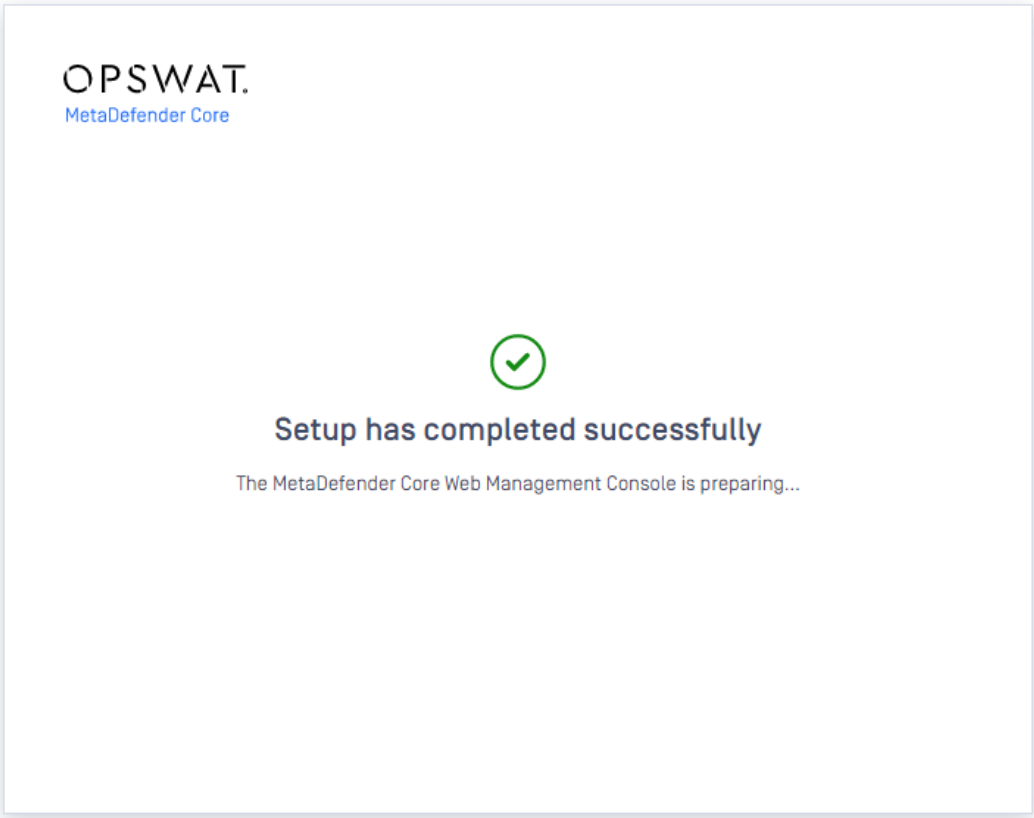


Wizard completion

After you have completed every steps you are ready to finish the wizard and start using the product. Click the `Fishish` button to complete the wizard.

> **Warning**
> The product's service will be restarted and the browser will be redirected to the administrative management console. This could take several seconds.

You can login to the Web Management Console with the administrator user that have just been created in the previous steps.



Setup HTTPS

Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network. Websites, like the Web Management Console, are able to use TLS to secure all communications between their servers and web browsers.

The TLS protocol aims primarily to provide confidentiality (privacy) and data integrity between two communicating computer applications.

Warning

No TLS for the wizard

By default, TLS is not enabled for the basic configuration wizard. As a consequence sessions between the wizard's backend and the browser may be insecure.

Info

Performing the same steps as for the Web Management Console, it is possible to set up TLS for the basic configuration wizard. Remember completing the TLS setup before launching the wizard.

For instructions to set up TLS see Enabling HTTPS

License Activation

##

To activate your installation go to the **Settings > License** menu in the Web Management Console. If you have no valid license, you will only see your installation's **deployment ID**. You will also see a warning in the Web Management Console header.

After successful activation, the product will start downloading the latest available scan engines and malware databases. You can follow the status of the scan engine installation on the **Inventory > Modules** page.

When all the licensed engines are installed, you can start using the MetaDefender Core to process files - File Processing.

Deployment ID

Each MetaDefender Core instance represented by an unique system fingerprint called `Deployment ID`

MetaDefender Core licensing is purely based on deployment ID activated. Therefore, everytime your MetaDefender Core is activated with a different deployment ID, then it will be captured as a different activation record accumulated into your total activation times allowed.

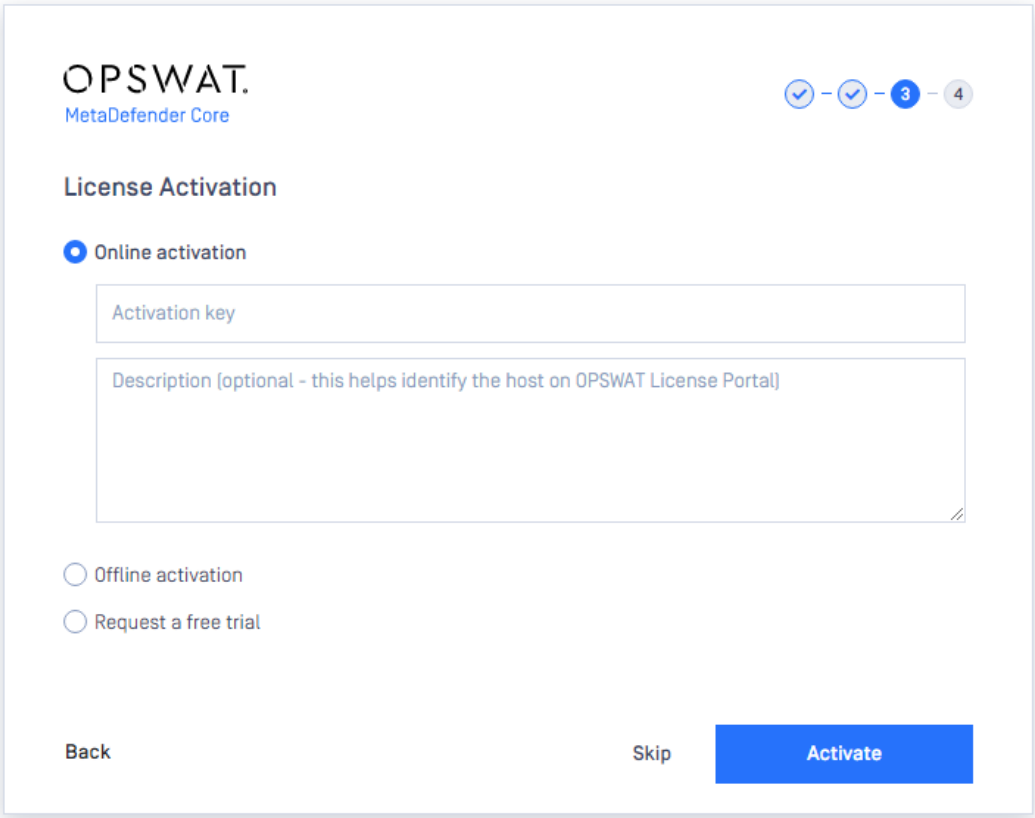
> **Notice**
> Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you alter the system where MetaDefender Core relies on (e.g. network card, OS hostname).
>
> Deployment ID is an UUID version 4 random based string. Learn more at: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you update the system where MetaDefender Core relies on.

Activation Modes

There are three different options for the product activation:

- Online activation: MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.
- Offline activation: Requiring you to upload a license file (.yml). That license file could be retrieved by activating your license via OPSWAT Portal. Follow the displayed instructions for details.
- Request trial key online: For evaluation purpose, you can receive an trial Activation key via email. Follow the displayed instructions for details.



Online activation

MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.

Required to input your Activation License key, and optional for description (but we recommend this for troubleshooting and auditing purpose if needed).

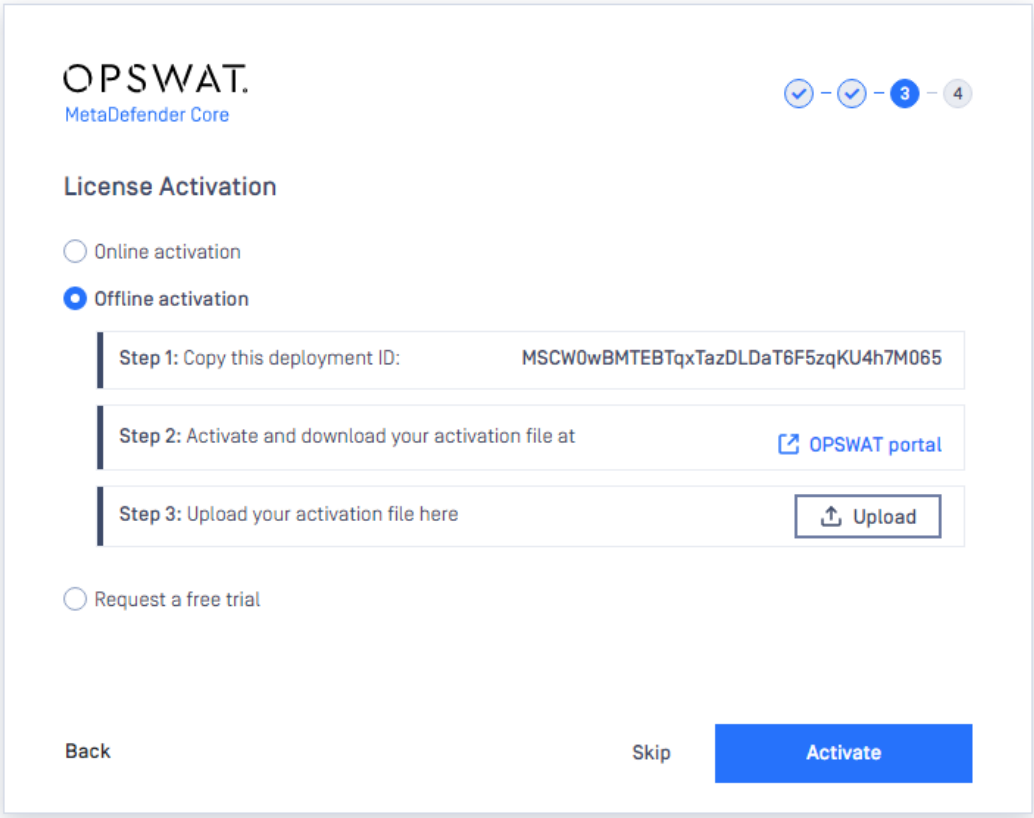
Warning

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts' for license activation:

- https://activation.dl.opswat.com Note: *IP address-based whitelisting on your firewall might fail after some time since OPSWAT uses CDN (Content Delivery Network) to faster delivery updates over the world, and IP address of edge servers might change over time.*

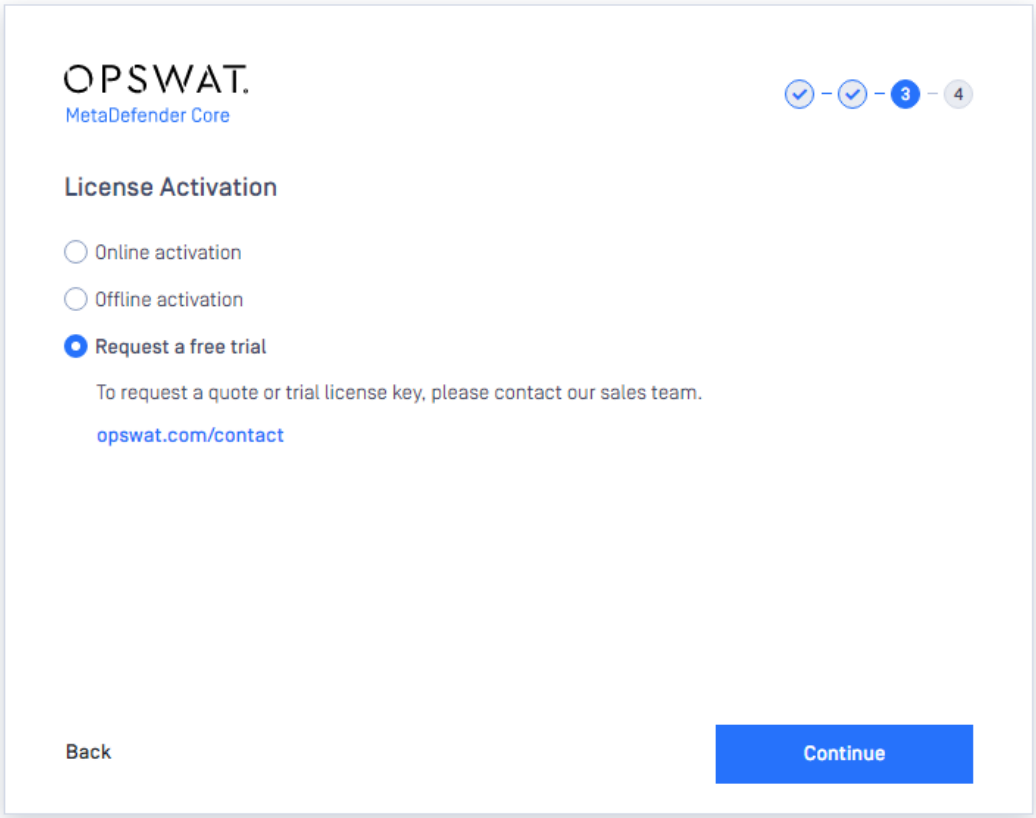
Offline activation

With no internet connection on the server the Metadefender v4 instance may be activated indirectly from a different machine, that has internet connection. The Deployment ID of the Metadefender v4 instance and the the Activation key received at the time of purchasing the product will be required. Follow the steps on the screen to activate the product offline.



Request trial key online

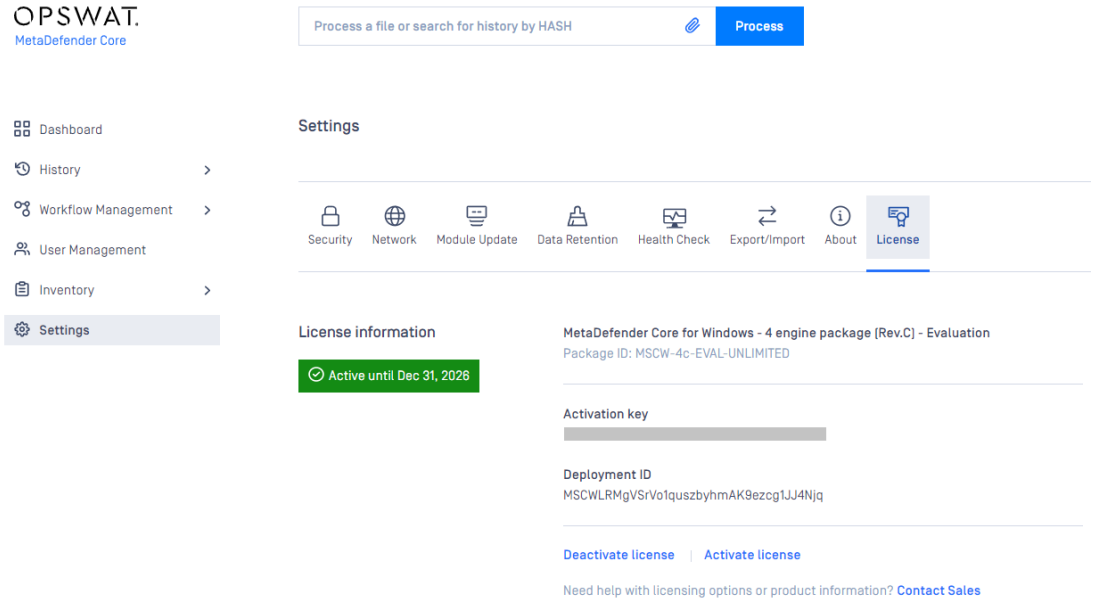
An evaluation license could be acquired by contacting our sales team.



Check Your License Details

For more license details and activating your MetaDefender Core, go to Settings > License menu tab on the Web Management Console:

- Product name: product name as on your order (e.g. MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation)
- Package ID: product identification as on your order (e.g. MSCW-4c-EVAL-UNLIMITED)
- Expiration: last day of license validity
- Activation key: key used for activating product
- Deployment ID: identification of this installation


OPSWAT.
MetaDefender Core
Process a file or search for history by HASH
Process
Dashboard
History
Workflow Management
User Management
Inventory
Settings
Settings
Security
Network
Module Update
Data Retention
Health Check
Export/Import
About
License
License information
Active until Dec 31, 2026
MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation
Package ID: MSCW-4c-EVAL-UNLIMITED
Activation key
Deployment ID
MSCWLRMgVSrVo1quszbyhmAK9ezcg1JJ4Njq
Deactivate license
Activate license
Need help with licensing options or product information? Contact Sales

Scripted license management

Using REST API calls there is an option for scripted activation and deactivation for MetaDefender Core servers with **no Internet connection or if your infrastructure is using automation to create/destroy MetaDefender Core instances**. In this chapter the steps of these two scenarios are described.

Requirements

- an installed MetaDefender Core instance without Internet connection
- another computer that has Internet connection and can run your activation scripts
- a manual or automated way to transfer data between the two computers

Activation steps

1. In order to activate MetaDefender Core, deployment ID and activation key are needed to generate the license file.
    a. Activation key should be purchased from OPSWAT.
    b. MetaDefender Core deployment ID can be queried by using REST API (for details see GET Get Current License Information API page)

> **Info**
> Save this deployment ID in your system, you might need this ID when the instance is unavailable at the time of deactivation.

2. Activate license and get the license file using the following URL, where <quantity> is 1.

activation call bash

```
curl -X GET https://activation.dl.opswat.com/activation?key=<activation-key>&deployment=<deployment unique-ID>&quantity=<quantity>
```

If the activation is successfully, a license file is downloaded. Save this file.

HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	license file	Activation was successfully.
200 Activation failed	error: '<user conform error message>'	Failed activation
200 Invalid parameter	error: 'Could not activate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not activate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
200 Invalid parameter	error: 'Could not activate your product because the quantity you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key, quantity or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Info
In case of any activation issue, contact OPSWAT support for help

3. Upload license file to MetaDefender Core: The license file should be uploaded to the MetaDefender Core v4 to activate the product. For details see page: POST Upload license key API

Deactivation steps

1. For deactivation, following infomation is required:
    a. Activation key
    b. MetaDefender Core deployment ID: can be queried by using REST API (for details see GET Get Current License Information API page)
2. Deactivate license using the following URL:

deactivation call bash

```
curl -X GET https://activation.dl.opswat.com/deactivation?key=<activation-
key>&deployment=<deployment unique-ID>
```

Response HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	result: ok	Successful deactivation
200 No active license found	error: 'Could not found any active license with the given parameters'	The license has not been activated yet or it has been deactivated already.
200 Invalid parameter	error: 'Could not deactivate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not deactivate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Important notes

Warning

Product activation is tied to several hardware, operating system and software parameters. In case of one or more major hardware or software parameter change the product might turn into deactivated status. Operating system updates, other software updates should not affecting the activation status.

If the product is online activated and has live Internet connection then in case of deployment ID change the product reactivates itself. If the product doesn't have Internet connection then the administrator is responsible to reactivate the product.

Because of the above administrator should consider licensing restrictions of the offline deployments if using any virtualization and/or containerization technologies.

File Processing

MetaDefender Core provides REST API interface for file processing business.

There are several ways to process files with MetaDefender Core:

- Analyze Files via Web Interface
- Process Files via REST API

Product Upgrading

> **Info**
> Simply run installer file, details could be found at Installation

> **Info**
> For command line mode, upgrading (e.g. on **Linux**), prior to running the installer file, please make sure to **create the ignition file** with PostgreSQL server information included. Details:
>
> - Standalone DB mode: Command Line Installation (Windows and Linux)
> - Shared DB mode: Command Line Installation

Recommendation for MetaDefender Core upgrade

Our goal is to help you execute the product upgrade continuously, efficiently, and correctly.

PostgreSQL service upgrade

- **Bundled PostgreSQL Service**: If you are using the bundled PostgreSQL server included in the installer, the PostgreSQL service will be automatically upgraded to the newer version (if available). No additional action is required for this process.
- **Remote PostgreSQL Service**: For remote PostgreSQL servers, customers are responsible for managing and upgrading the service independently. Upgrading MetaDefender Core does not update the remote PostgreSQL service.

Before the upgrade

Prepare a proper disaster recovery plan (DRP) including some common practices:

- Backup, or take a snapshot of your current working MetaDefender Core server.
  - This is to help quickly rollback for any upgrade failure might cause later for any reason.
- Optionally backup your PostgreSQL database, especially when using remote shared database mode.
  - Reference: https://www.postgresql.org/docs/current/backup.html
  - Please do it after stopping all linked MetaDefender Core instances to avoid potential data collision.
- Lastly, if backing up entire server is not feasible, then at least export MetaDefender Core configuration package Import/Export configuration just in case you'll need to start off a new server with current MetaDefender Core configuration to keep the scanning service running without any impact.

On the PostgreSQL server, double check server resource to ensure sufficient RAM memory and disk space to be preserved for the upgrade process.

- 8GB RAM available at least during the upgrade.

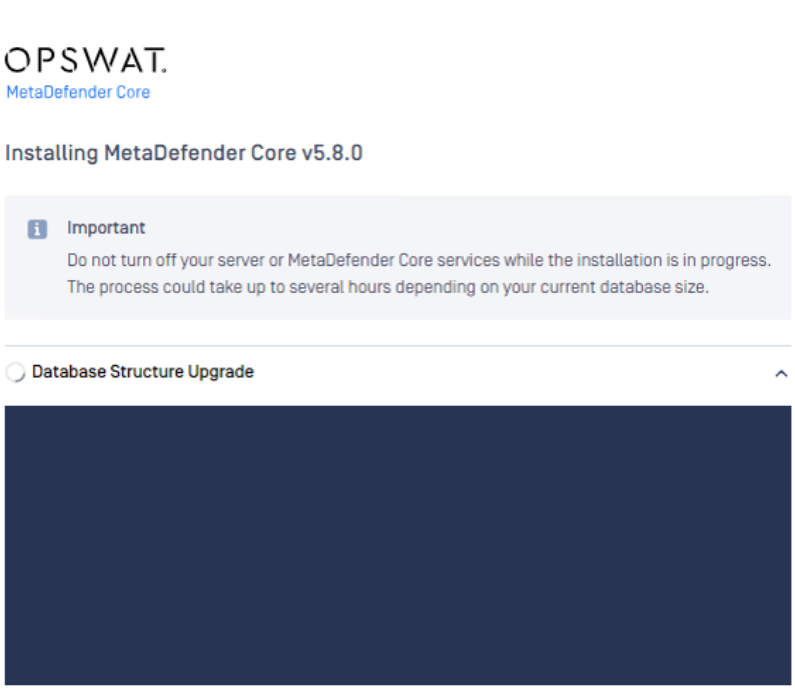
- Disk availability is greater than MetaDefender Core database size.

While doing the upgrade

Applicable for shared database deployment:

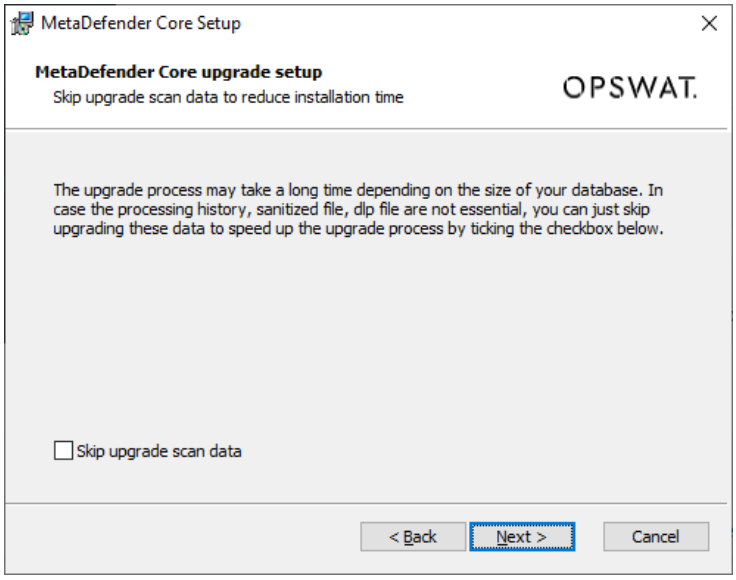
- Stop all MetaDefender Core instances' scanning service before performing the upgrade.
- Upgrade one (1) MetaDefender Core instance at a time.
- Proceed to upgrade MetaDefender Core instance where the bundle local PostgreSQL server is used for shared database setup.
- Otherwise is using remote PostgreSQL then the upgrade order does not matter.

For large size database, mostly for scan history, expecting long time to be spent on the upgrade process.



The product supports an option to skip data migration for scan history to expedite the process drastically and mitigate any failure risk. You can manually backup all data from your PostgreSQL server for offline usage if any.

Check "Skip upgrade scan data" to enable this feature while doing the upgrade.



or add following key in the ignition file:

none

```
[internal]
skip_upgrade_scan_data = 0
```

Troubleshooting

If the upgrade process ever runs into an error:

- Double check your network and services, applicable to remote PostgreSQL server.
- Optionally configure MetaDefender Core to skip migrating scan history.
  - Windows
    - Open Windows registry, and navigate to HKLM\Software\OPSWAT\Metascan
    - Add a new key `internal/skip_upgrade_scan_data` with value = 1
  - Linux
    - Open MetaDefender Core configuration file `/etc/ometascan/ometascan.conf`
    - Add new key into the file
      - [internal]
      - skip_upgrade_scan_data=1
  - Note: This key will be auto removed upon the upgrade finishes.
  - Then restart MetaDefender Core service to retry the upgrade.
    - Might retry this a few times until the upgrade is done.

Just in case the upgrade process still fails, please contact OPSWAT support with the support package - How to Create Support Package? This will help us analyze the root cause and figure out an optimal solution for your system failure.

Using external Load Balancer

Application Load Balancing (Layer 7)

This page provides information about recommended ways to use sessions with a Layer 7 load-balancer.

Most load-balancers have the ability to provide cookies for pinning subsequent traffic from a client to the appropriate server. This method is called sticky session, session persistence or session affinity. Using cookies efficiently requires the client to know when a cookie should be sent, should not be sent or should be deleted.

Sticky session load balancing

Each Metadefender Core instance has its own databases and application sessions that cannot be seen by other instances. Therefore, in order to get the related data/response to our queries we should ask the appropriate Metadefender Core v5 server. On the other hand, to keep the advantages of the used load balancing method, cookies should not be sent if it is not necessary.

Single file scanning

Step	Stage	Task	Cookie usage
1.	Sending file	Initiate processing a file on the client side. Send file through the load-balancer. (See POST File Analysis API)	Cookie should not be sent
2.	Sending file	Save the cookie and the data_id you got from the load-balancer	Save cookie
3.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Send cookie saved in step 2
4.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Send cookie saved in step 2

Batch scanning

Step	Stage	Task	Cookie usage
1.	Open batch	Initiate processing file(s) in batch. Request a batch ID through the load-balancer. (See POST Initiate Batch API)	Cookie should not be sent
2.	Open batch	Save the cookie and the batch_id you got from a Core server through the load-balancer.	Save cookie
3.	Sending files	Send file through the load-balancer. (See POST File Analysis (using batch header) API)	Send cookie saved in step 2
4.	Sending files	Save the data_id you got from the load-balancer.	-
	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Send cookie saved in step 2
5.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
6.	Getting batch status	See GET Batch Status API	Send cookie saved in step 2
7.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Send cookie saved in step 2
8.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API	Send cookie saved in step 2

If it does not matter which upstream server responds, the queries should be sent without cookie.

> **Info**
> It is recommended not to send cookies when it's not necessary to allow load-balancer to use its own method to share the load between Metadefender Core v5 servers.

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration through load-balancer

OPSWAT products that support HTTP load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	4.3.4	-
OPSWAT Client	Windows: 7.6.247.0 Mac: 10.4.243.0	-
MetaDefender Email Security	4.3.0	-
MetaDefender ICAP Server	4.3.0	-
MetaDefender Vault	1.3.0	-
MetaDefender for Secure Storage	2.0.7	-

DNS Load Balancing

Using this method is logically similar to Layer 7 load-balancing.

Briefly how it works

Client uses a domain name to send a query to a server. Client's DNS server has more "A" records for that name with different IPs. When a client resolves the server's domain name DNS server randomly chooses an IP for that name to send back. When a session is used on the application layer, client should know the IP address of the Core server that handles that specific session. In every other case, client should resolve the domain name with DNS query to let requests to be balanced between Metadefender Core servers.

Single file scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Sending file	Initiate processing a file on the client side. (See POST File Analysis API)	Use IP saved in step 2
4.	Sending file	Save the data_id got from the Core server	-
5.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Use IP saved in step 2
6.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Use IP saved in step 2

Batch scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Open batch	Initiate processing file(s) in batch. Request a batch ID. (See POST Initiate Batch API)	Use IP saved in step 2
4.	Sending files	Send file to the specific Core server with the basch ID saved in step 3. (See POST File Analysis (using batch header) API)	Use IP saved in step 2
5.	Sending files	Save the data_id you got from the Core server.	-
6.	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Use IP saved in step 2
7.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
8.	Getting batch status	See GET Batch Status API	Use IP saved in step 2
9.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Use IP saved in step 2
10.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API).	Use IP saved in step 2

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration via DNS load-balancing

OPSWAT products that support DNS load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	does not support yet	-
MetaDefender Client	does not support yet	-
MetaDefender Email Security	does not support yet	-
MetaDefender ICAP Server	does not support yet	-
MetaDefender Vault	does not support yet	-

SSL connection for PostgreSQL communication

> **Note**
> The SSL connection configuration is possible thanks to the support made by PostgreSQL. For reference: https://www.postgresql.org/docs/14/ssl-tcp.html
>
> MetaDefender Core is tested against this mode on particular version 4.21.1

Step 1: Generate SSL certificate for PostgreSQL server

> **Note**
> Following instructions are for self-sign certificate. For production environment, making sure to use the certificate signed by your trusted CA.

1) Generate a private key

bash

```
openssl genrsa -des3 -out server.key 1024
```

Sample output:

bash

```
Generating RSA private key, 1024 bit long modulus ........++++++
...............++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

Type your desired password / pass phrase to encrypt the private key

2) Remove the pass phrase to automatically start up the PostgreSQL server

bash

```
openssl rsa -in server.key -out server.key
```

> **Note**
> On Linux, make sure to set permission on the server.key file

bash

```
chmod og-rwx server.key
```

3) Create a self-signed certificate

bash

```
openssl req -new -key server.key -days 3650 -out server.crt -x509
```

You will be prompted to enter detailed information which is incorporated into your self-signed certificate request.

none

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:CA
Locality Name (eg, city) [Default City]:San Francisco
Organization Name (eg, company) [Default Company Ltd]:OPSWAT
Unit Name (eg, section) []:OPSWAT
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:dummy_email@opswat.com
```

For self-signed certificate, use the server ceriticate as the trusted root certificate:

bash

```
cp server.crt root.crt
```

Copy **server.key**, **server.crt**, **root.crt** to PostgreSQL data folder. For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data**
- Linux: **/var/lib/ometascan/pg_data/**

Step 2: Configure PostgreSQL server for SSL authentication connection

1) Create a custom config file (e.g. ssl.conf) for PostgreSQL.

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\postgres\ssl.conf**
- Linux: **/var/lib/ometascan/postgres/ssl.conf**

2) Content of ssl.conf:

none

```
ssl = on
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
```

3) Modify pg_hba.conf

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data\pg_hba.conf**
- Linux: **/var/lib/ometascan/pg_data/pg_hba.conf**

Add following (modify **::1/128** if needed to match with your environment requirement)

none

```
hostssl all all      ::1/128      scram-sha-256
```

After the modify, the content of pg_hba.conf should be like this (example):

> **Note**
> In a Linux environment, a local connection is required to upgrade from version 5.5.1 or earlier to version 5.6.0 or later.

none

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local         DATABASE  USER  METHOD  [OPTIONS]
# host          DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl     DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostgssenc    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnogssenc  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
local    all all                    scram-sha-256
hostssl all all      ::1/128      scram-sha-256

#More content.......
```

4) Reload the services

- Standalone DB mode: Restart **ometascan** service (MetaDefender Core service)
- Shared DB mode: Restart **ometascan-pg** / **ometascan-postgresql** service (MetaDefender Core PostgreSQL service) and then **ometascan** service (MetaDefender Core service)

Test your SSL database connection:

bash

```
psql.exe "sslmode=require dbname=postgres user=postgres host=localhost port=5432"
Password for user postgres:
psql (12.7)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
Type "help" for help.
```

This message indicate that you are now using SSL connection:

bash

```
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
```

Performance and Load Estimation

> **Disclaimer**
> These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Some factors that affect performance

- MetaDefender Core version
- MetaDefender Core engine package and configuration
  - set of engines (which and how many)
  - product configuration (e.g., thread pool size)
- System environment
  - server profile (CPU, RAM, hard disk)
  - client application location - remote or local
  - system caching and engine level caching
- Dataset
  - encrypted or decrypted
  - file types
    - different file types (e.g., document, image, executable)
    - archive file or compound document format files
  - file size
  - bad or unknown (assume to be clean)
- Performance tool

How test results are calculated

Performance (mainly scanning speed) is measured by throughput rather than unit speed. For example, if it takes 10 seconds to process 1 file, and it also takes 10 seconds to process 10 files, then performance is quantified as 1 second per file, rather than 10 seconds.

- total time / total number of files processed: 10 seconds / 10 files = 1 second / file.

Standalone DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.2.416.0

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

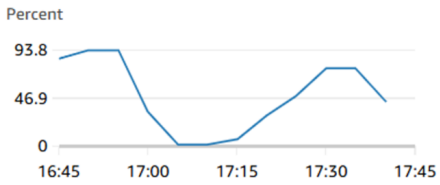
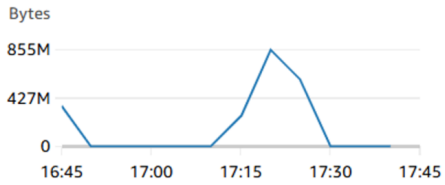
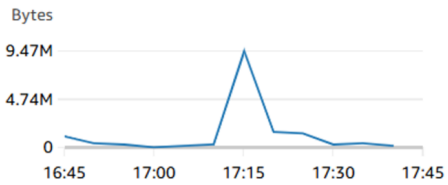
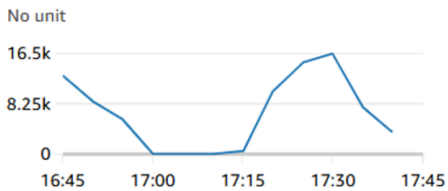
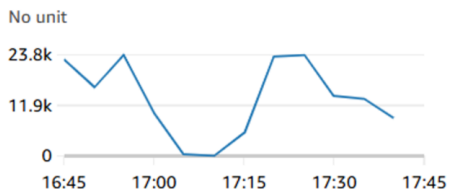
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV)	24.2	47.2 / 80	37.3 / 41.1		315,310	0.0114
Deep CDR	19.9	48.9 / 81.8	32.6 / 37		383,442	0.0094
Proactive DLP	17.1	36.8 / 58.2	30.4 / 33.6		446,228	0.0081
Vulnerability	10.1	59.2 / 85	29.3 / 33.3		755,495	0.0048

MetaDefender Core with common engine packages:

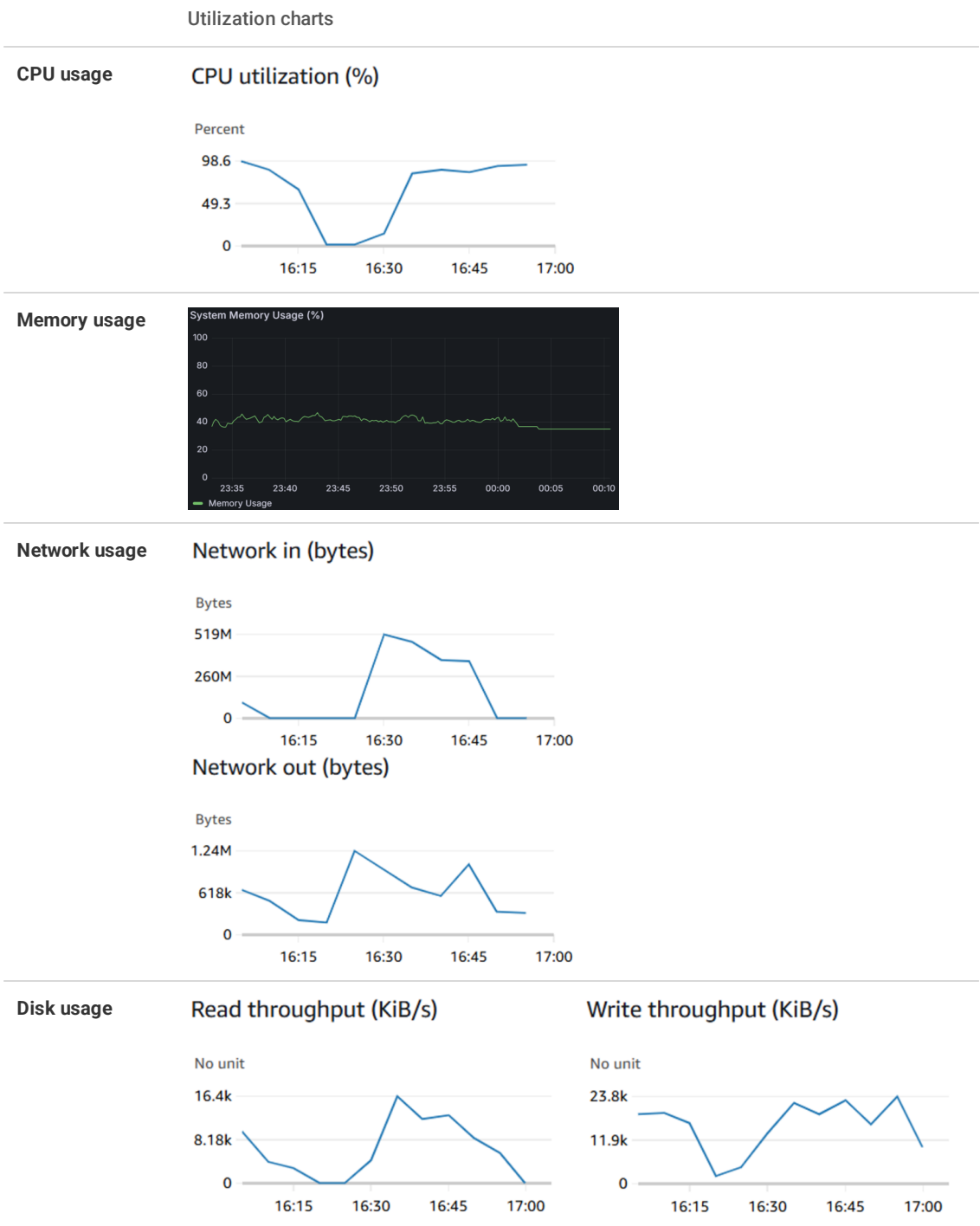
Use case	Scan duration	Avg./Max CPU usage	Avg./Max RAM usage	Avg./Max Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV) + Deep CDR	30.8	68.1 / 88	40.1 / 46.8		247,744	0.0145
Metascan (8 AV) + Deep CDR + Proactive DLP	31.2	74.2 / 98.6	41.5 / 51.1		244,567.3	0.0147
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	31.8	73.5 / 98	40.6 / 51.5		239,953	0.015

System resource utilization charts

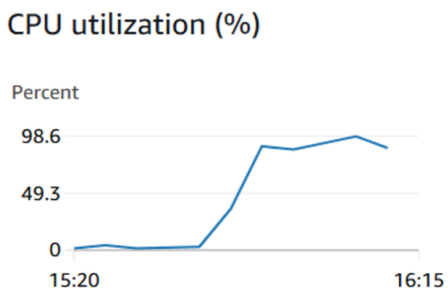
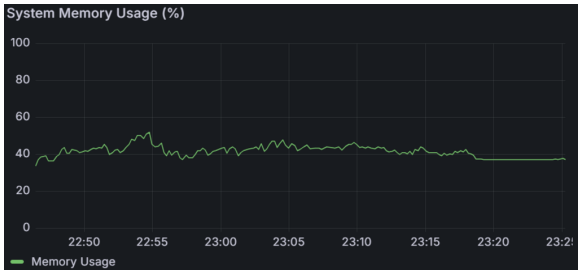
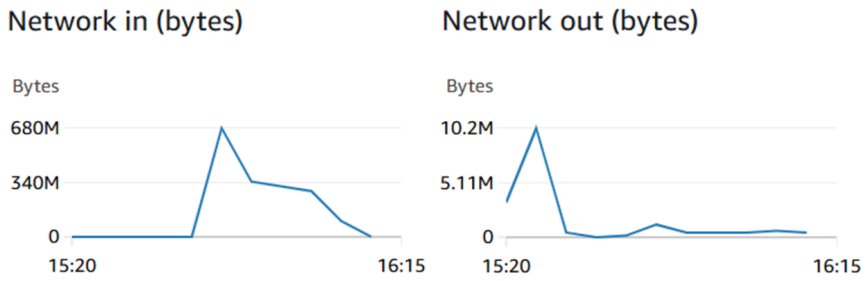
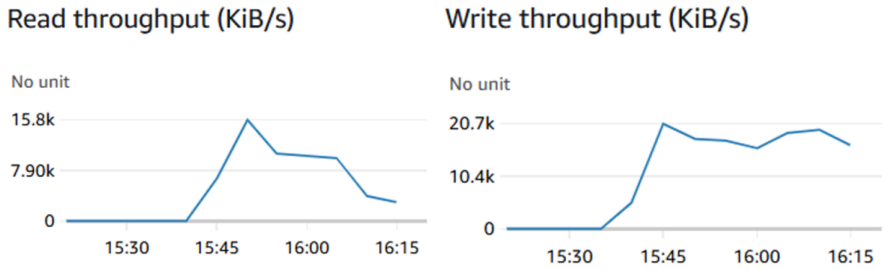
MetaDefender Core with Metascan (8 AV engines):

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

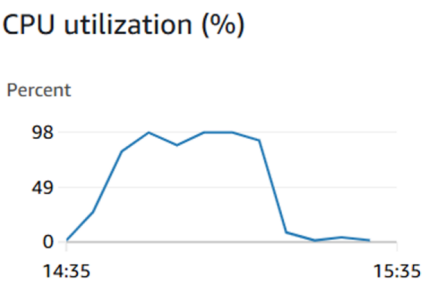
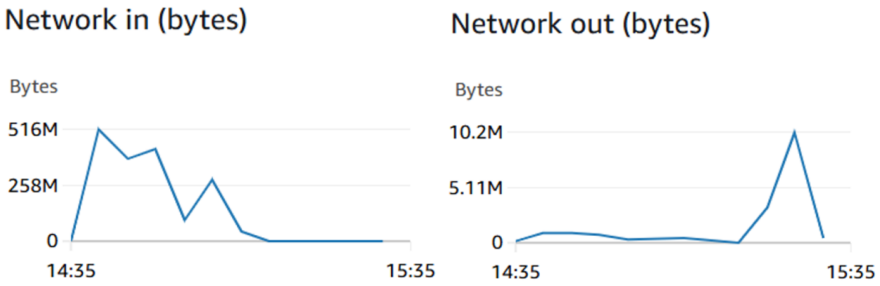
MetaDefender Core with Metascan (8 AV engines) + Deep CDR:



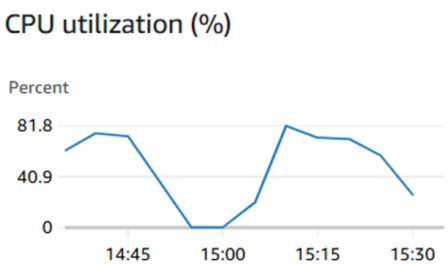
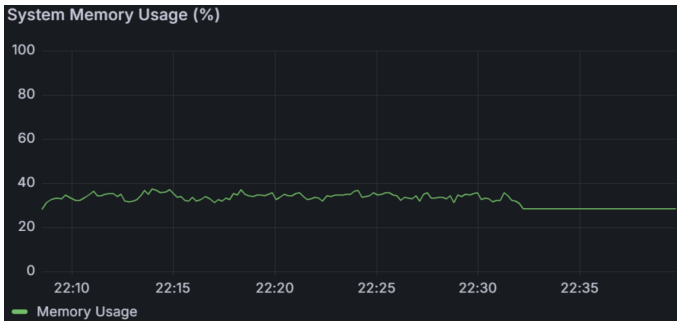
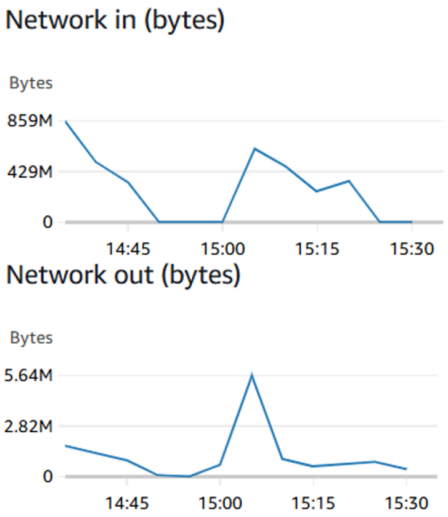
MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

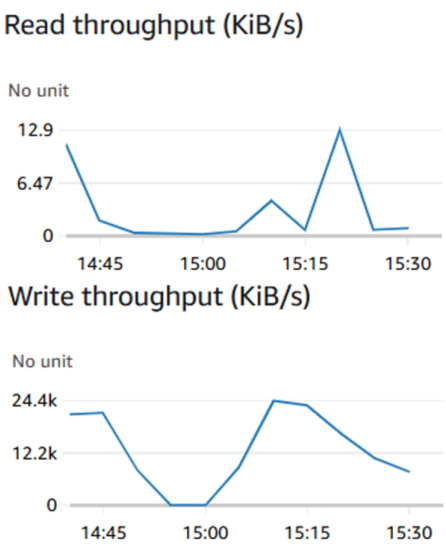
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

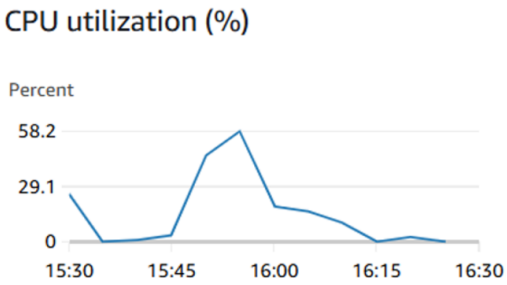
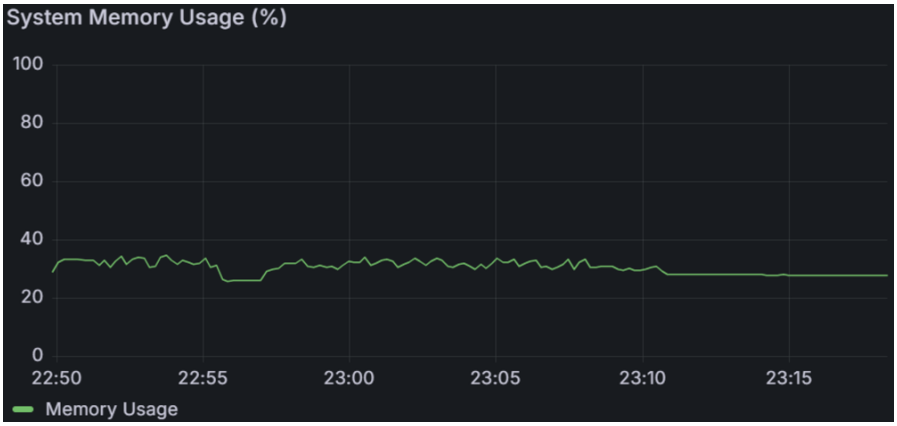
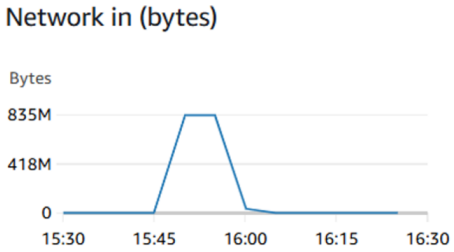
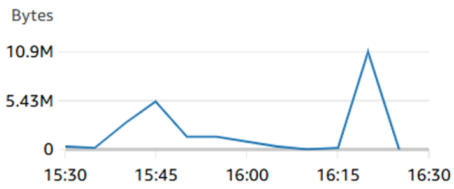
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

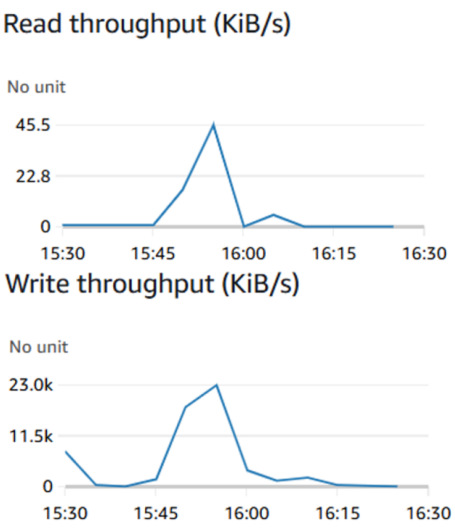
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

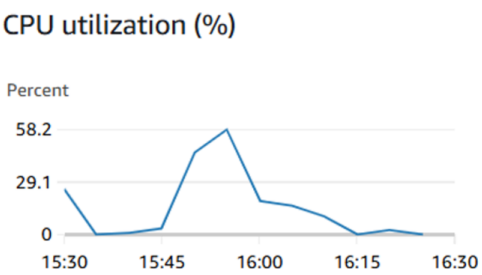
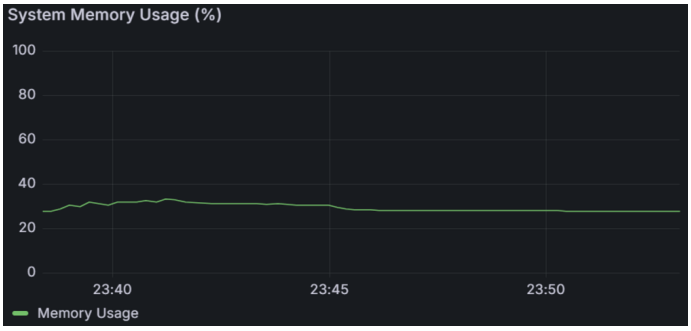
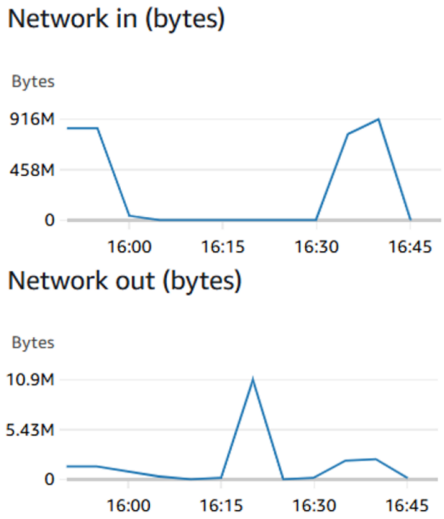
	Utilization charts
Disk usage	

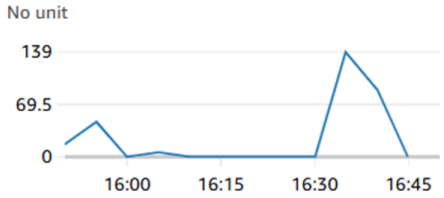
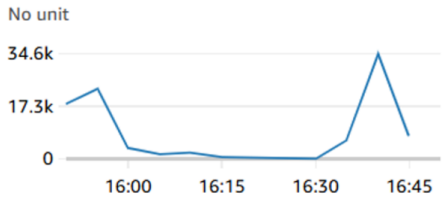
MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	 

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability:

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

Standalone DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Ubuntu 22.04	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.57-236

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

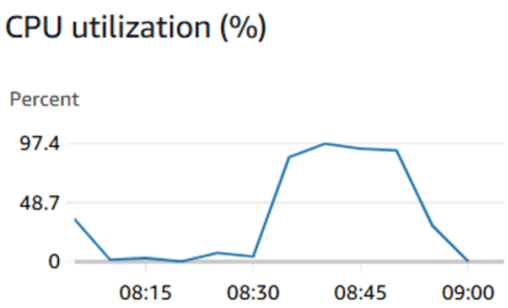
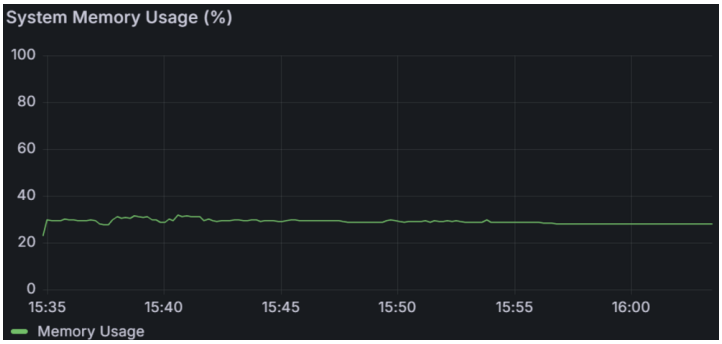
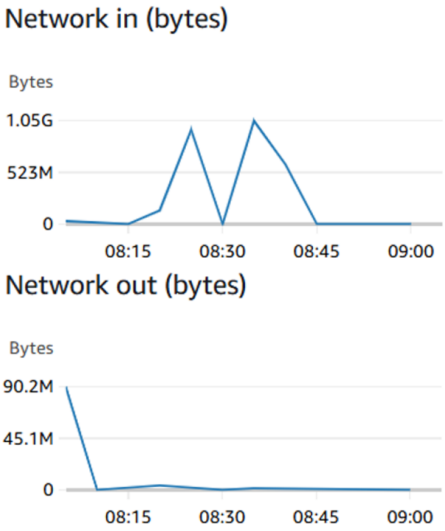
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV)	30.2	69 / 98	29 / 32		252,666	0.0142
Deep CDR	19.4	48 / 62	36 / 49		393,325	0.009
Proactive DLP	16.5	34 / 39	32 / 48		462,455	0.008
Vulnerability	12	50 / 65	46 / 46		635,875	0.006

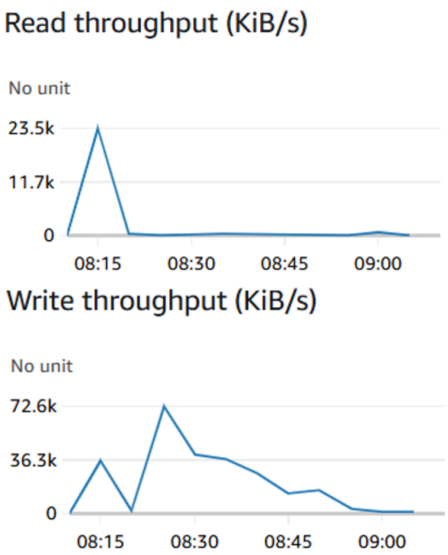
MetaDefender Core with common engine packages:

Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV) + Deep CDR	39.8	78 / 99	62 / 73		191,721	0.0188
Metascan (10 AV) + Deep CDR + Proactive DLP	43.5	61 / 84	47 / 67		175,414	0.0205
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	44.8	62 / 88	69 / 83		170,324	0.0211

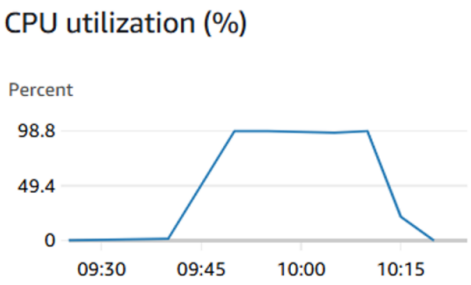
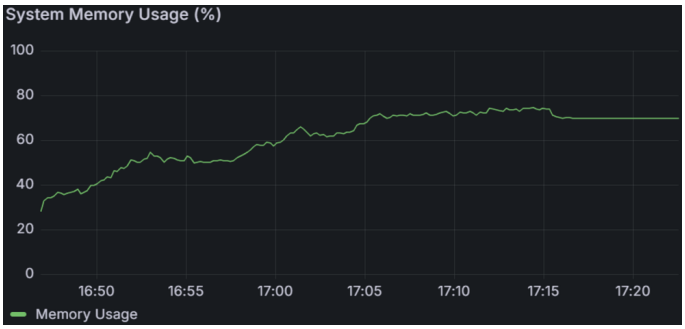
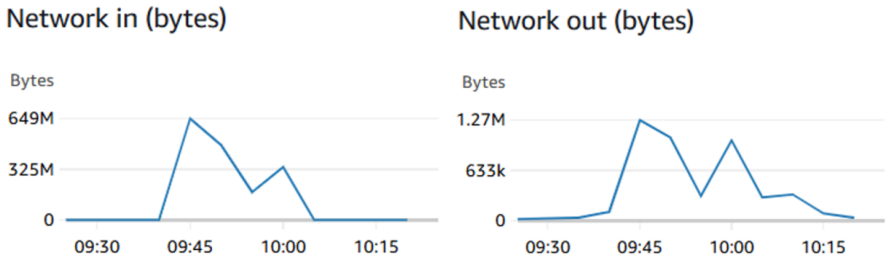
System resource utilization charts

MetaDefender Core with Metascan (10 AV engines) :

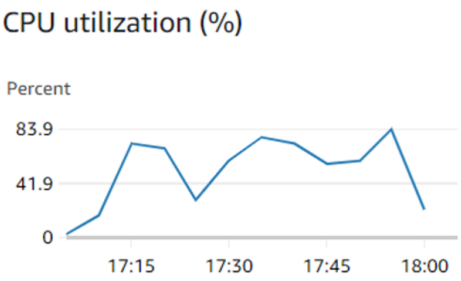
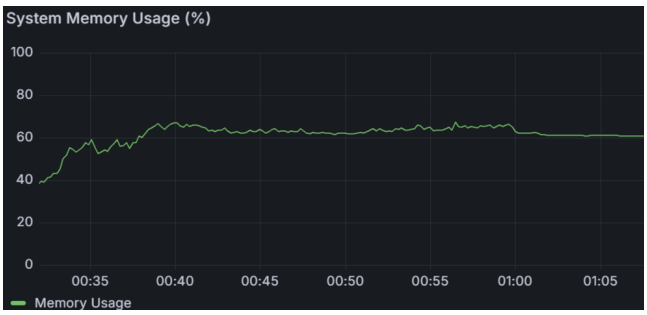
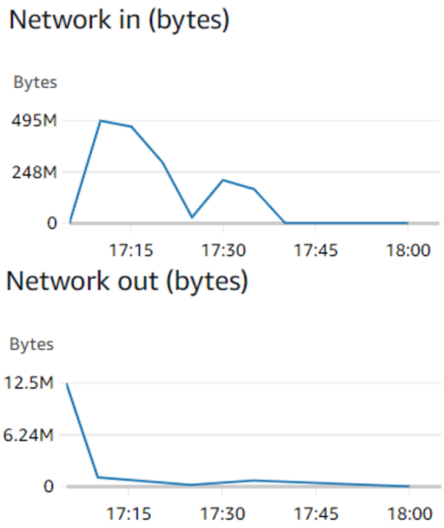
	Utilization charts
CPU usage	
Memory usage	
Network usage	

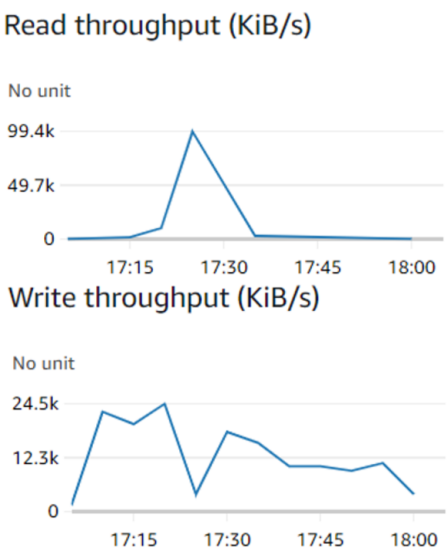
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR :

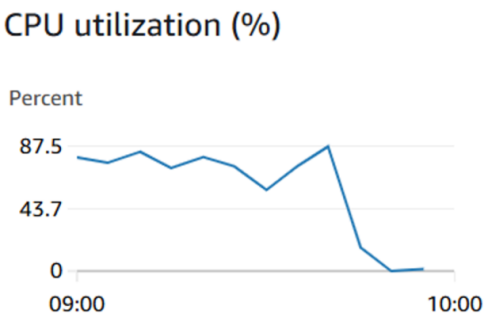
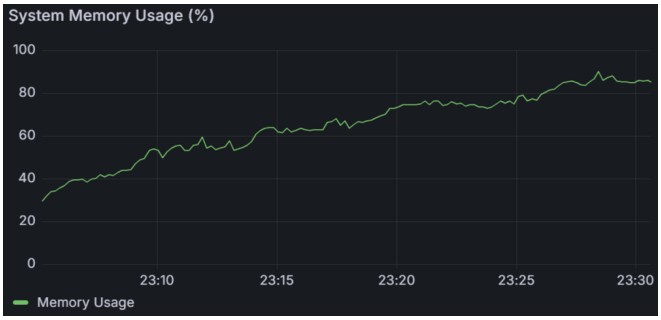
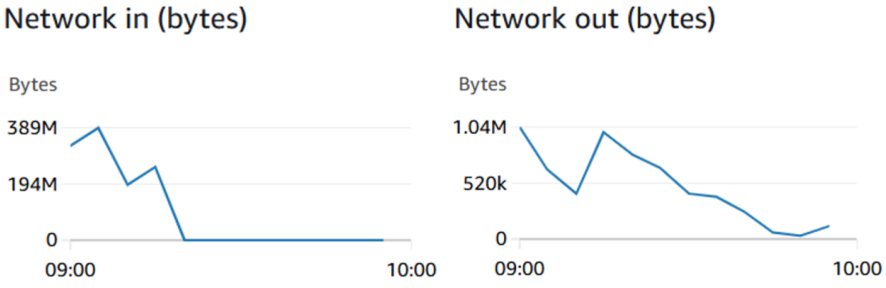
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP :

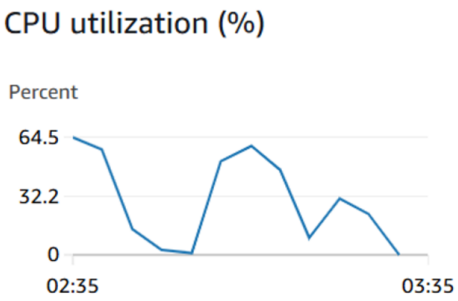
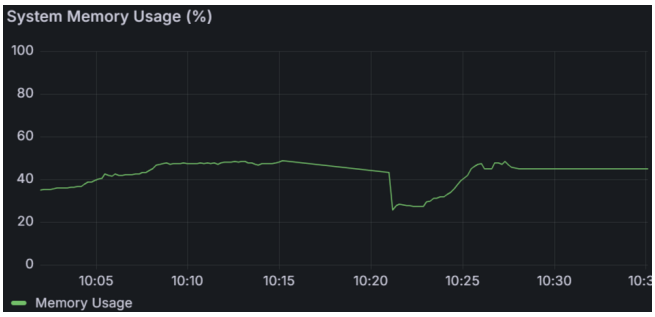
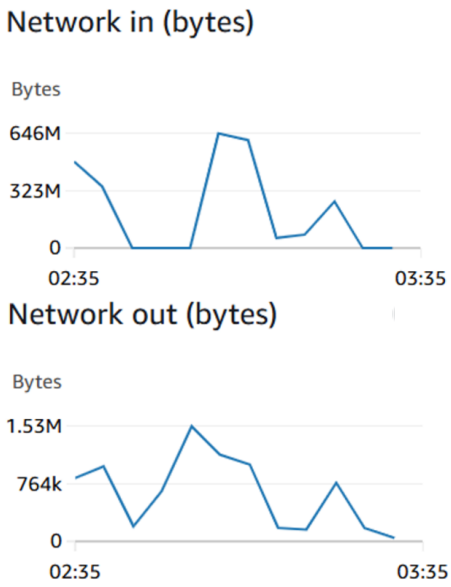
	Utilization charts
CPU usage	
Memory usage	
Network usage	

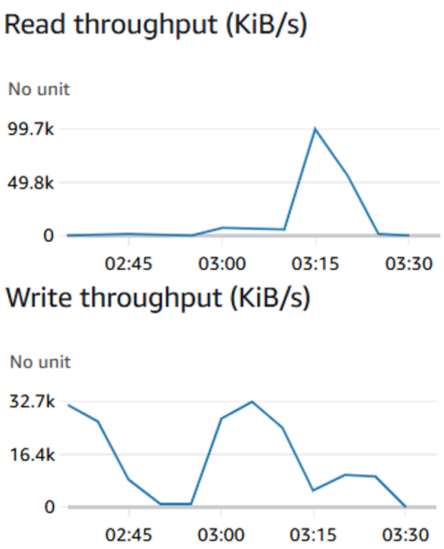
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP + Vulnerability :

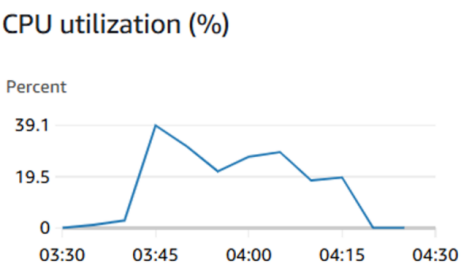
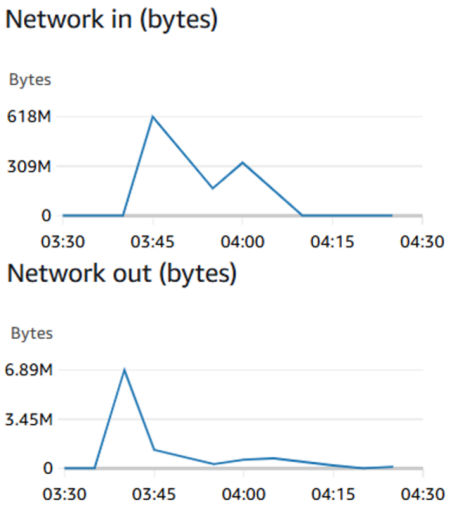
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

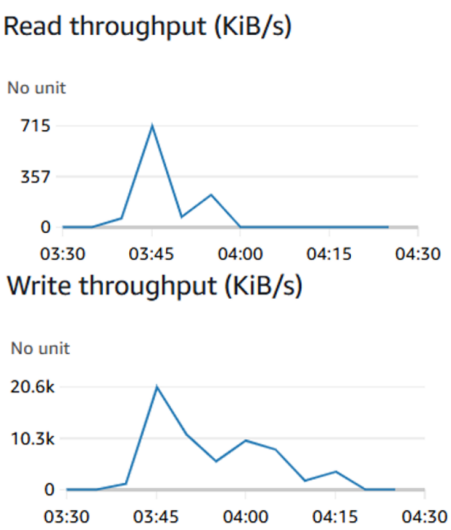
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

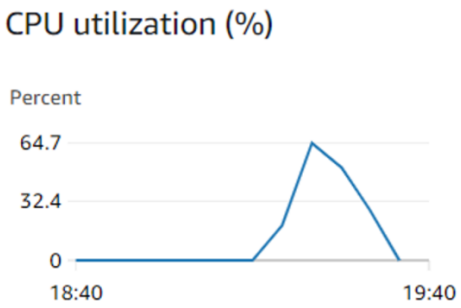
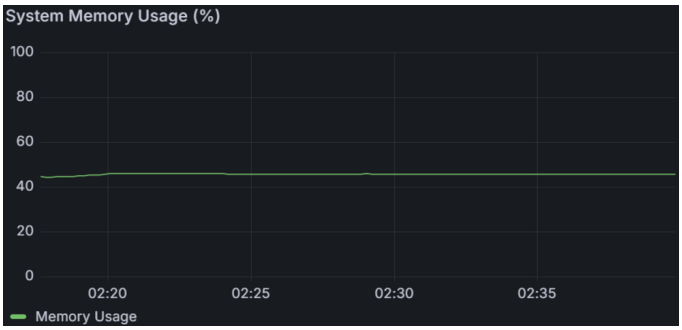
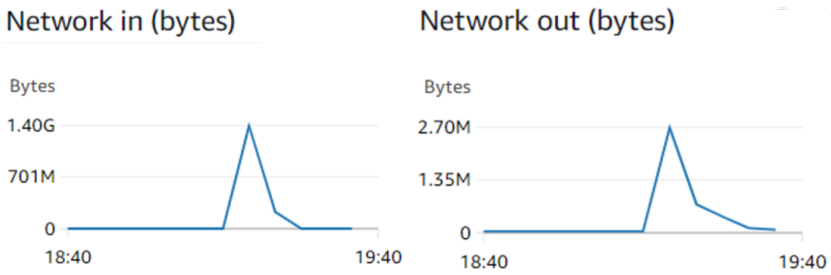
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability :

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

Shared DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

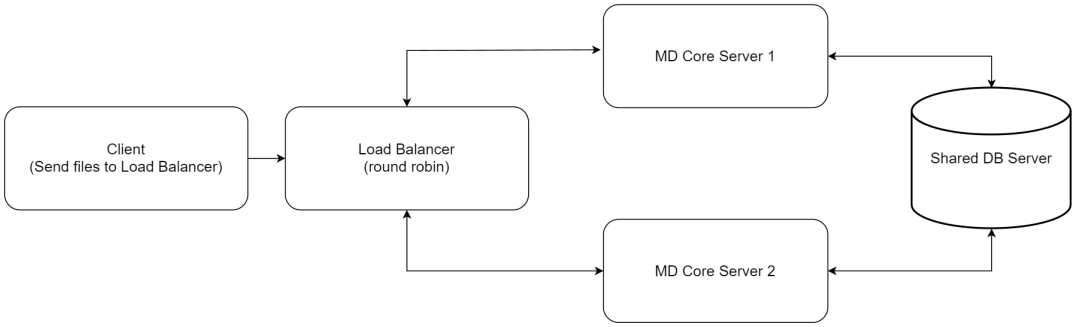
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV)	11.9	37.0	41.9	30.1	30.4	Receive: 6441.1 Send: 769.6	Receive: 6200.2 Send: 693.7	640,821.2	0.0
Deep CDR	10.5	60.1	66.7	30.3	32.2	Receive: 6379.8 Send: 594.8	Receive: 7146.1 Send: 609.6	729,368.5	0.0
Proactive DLP	7.8	38.7	37.5	29.7	31.0	Receive: 8842.6 Send: 791.8	Receive: 8635.6 Send: 729.1	965,455.7	0.0
Vulnerability	4.4	52.2	53.0	26.0	26.9	Receive: 16,215.5 Send: 1006.8	Receive: 15,785.7 Send: 997.2	1714,449.3	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV) + Deep CDR	15.4	72.4	75	34.3	35.4	Receive: 5436.6 Send: 611.3	Receive: 5589.3 Send: 607.2	495,697.4	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP	16.9	77.2	80.2	36.4	37.9	Receive: 5254 Send: 585.4	Receive: 5041.3 Send: 576.1	450,657.6	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	18.0	72.1	80.4	38.6	38.0	Receive: 4504.3 Send: 574.1	Receive: 4649.5 Send: 571.1	423,010	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

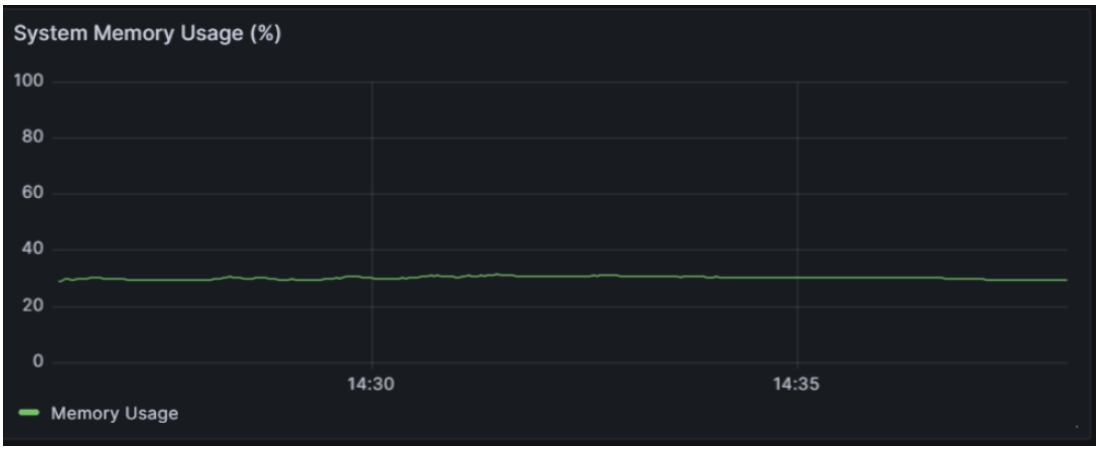
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2

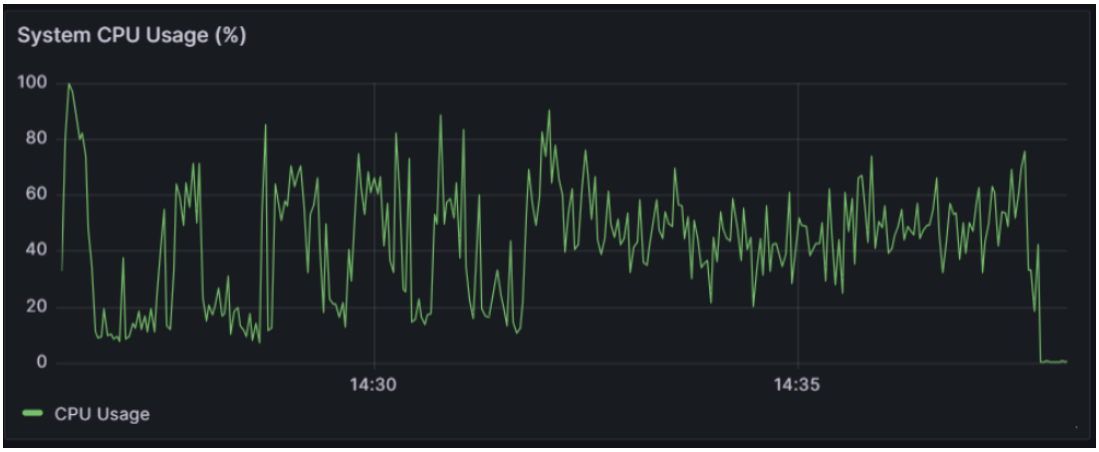


CPU usage

MD Core #1

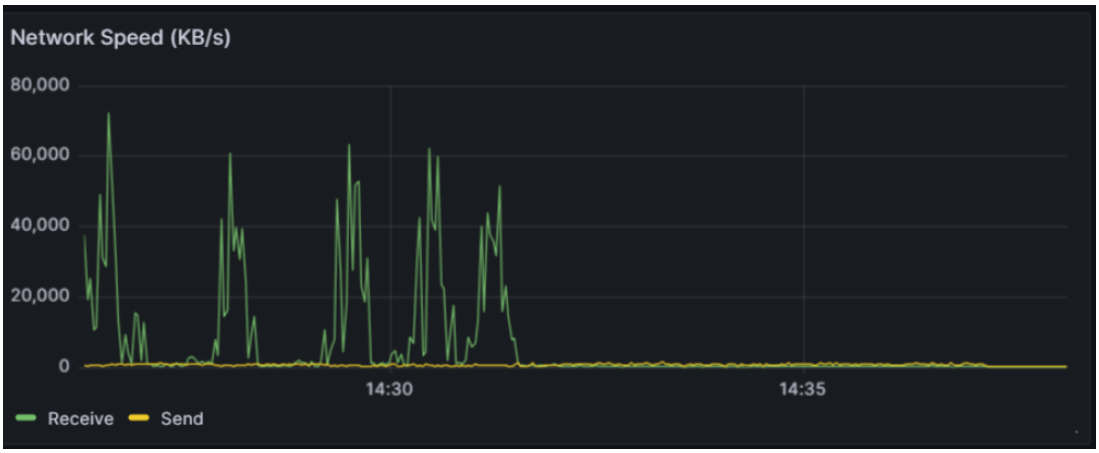


MD Core #2

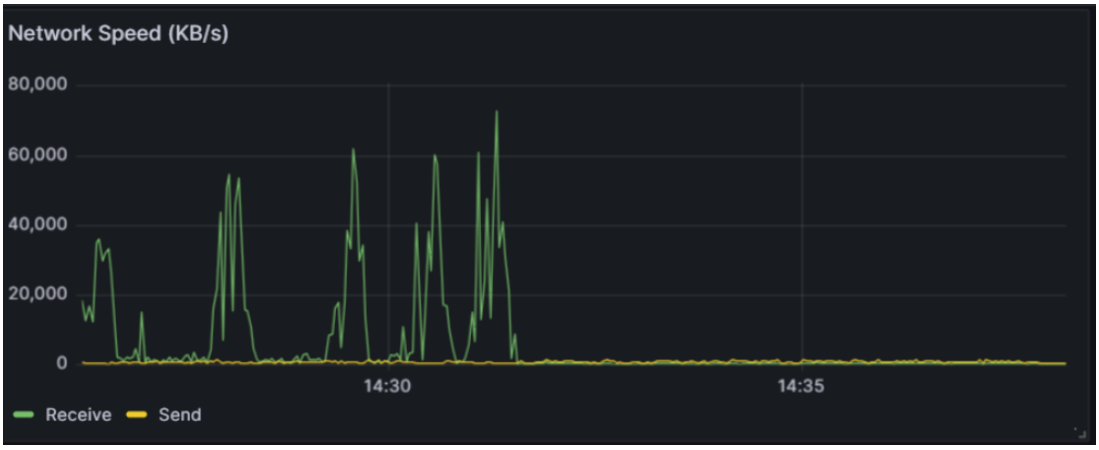


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2

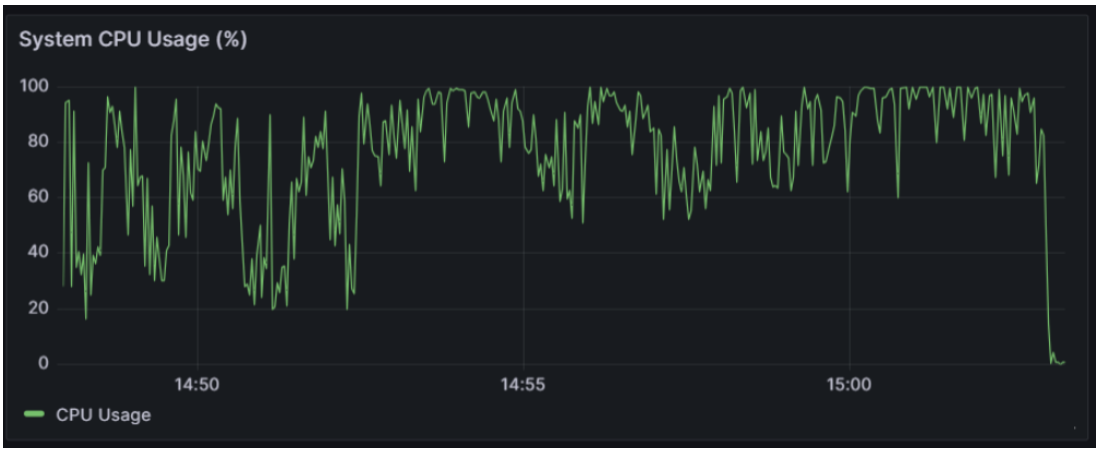


CPU usage

MD Core #1

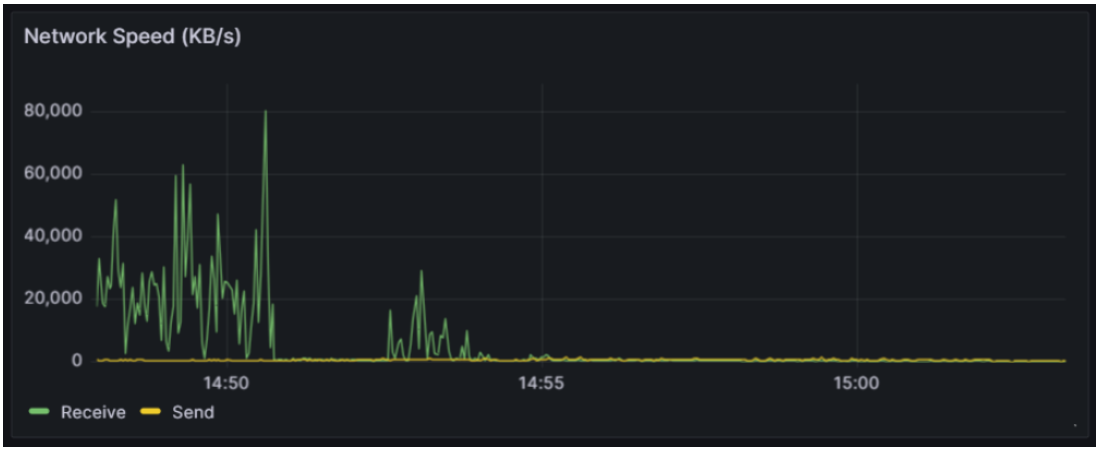


MD Core #2



Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

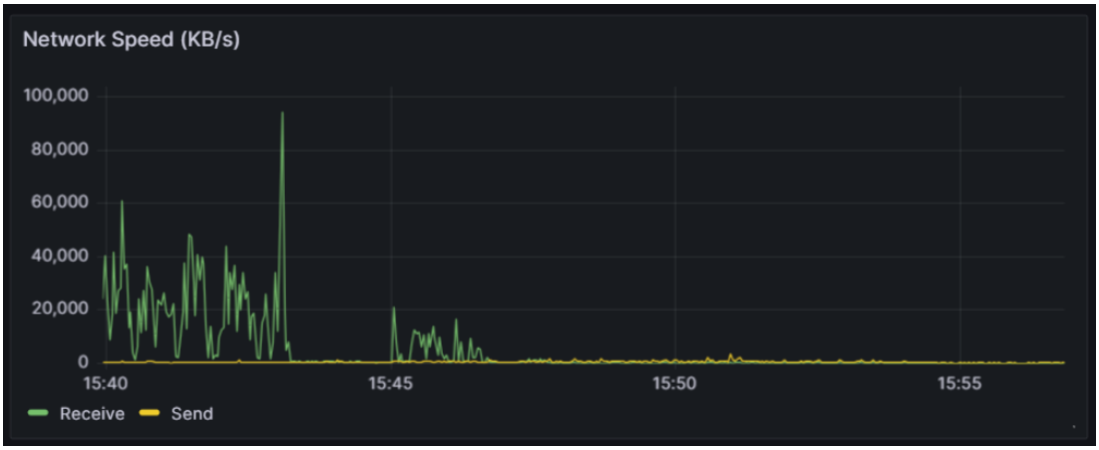


MD Core #2

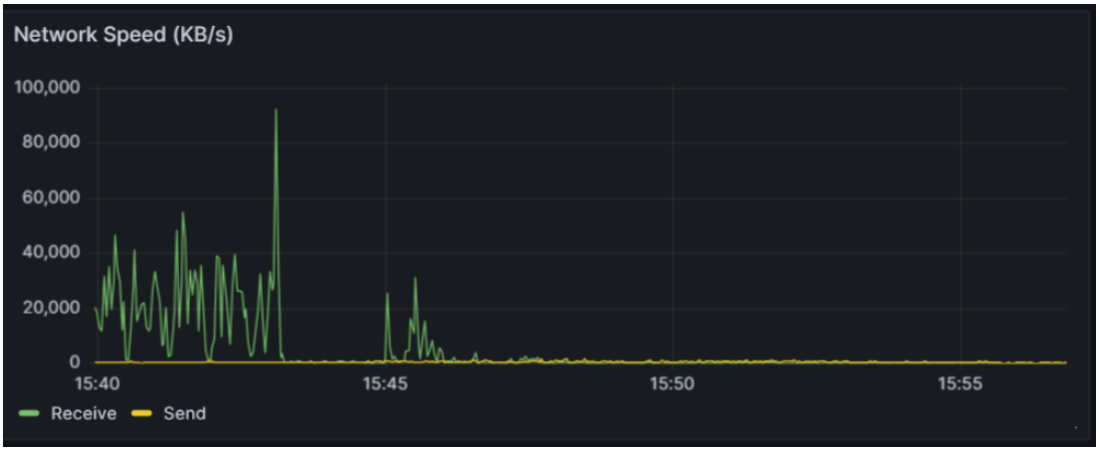


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1

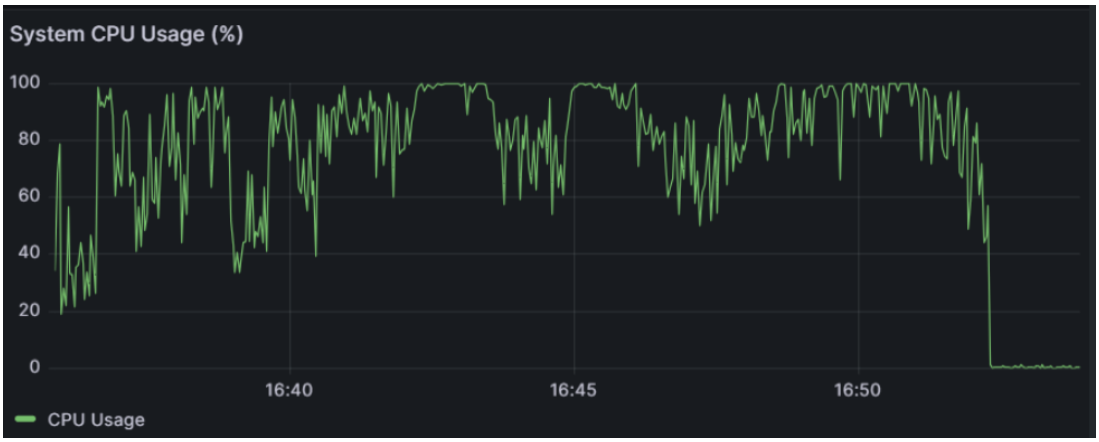


MD Core #2



CPU usage

MD Core #1

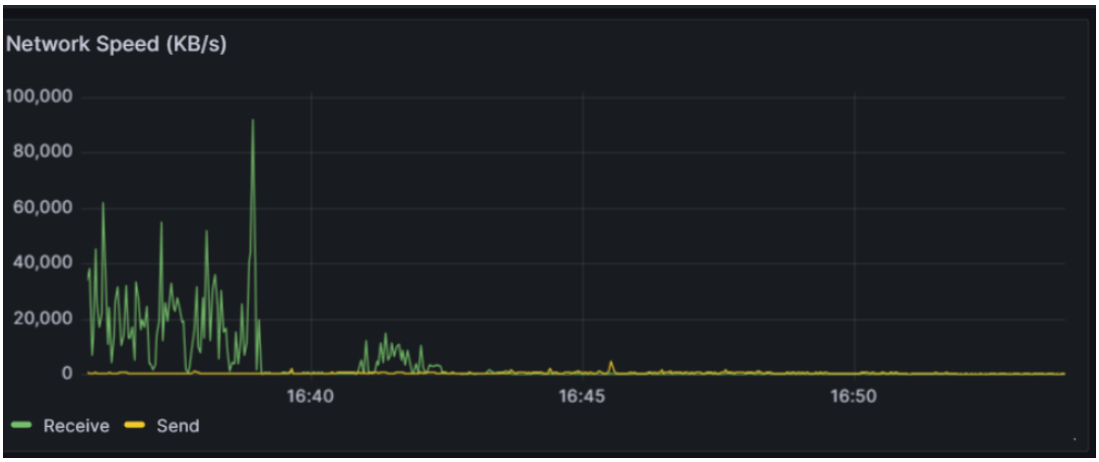


MD Core #2

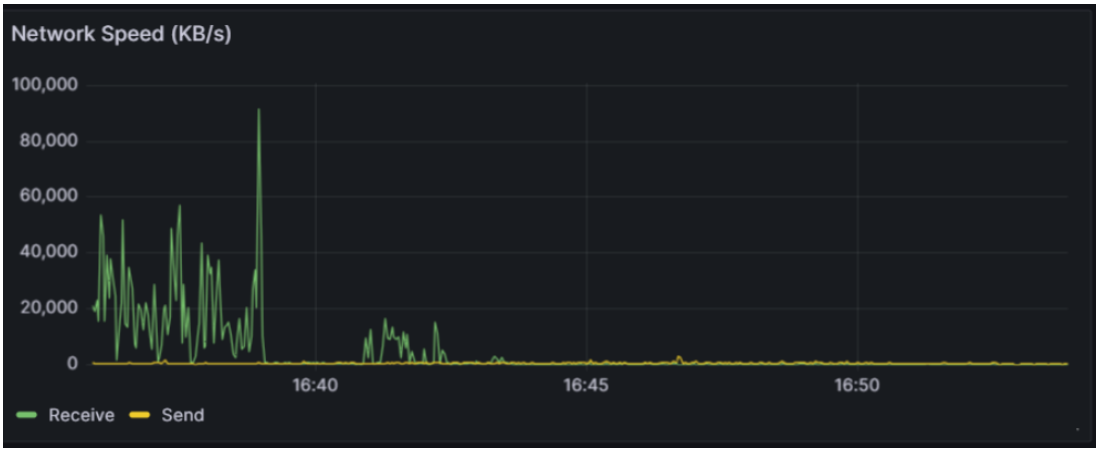


Network usage

MD Core #1



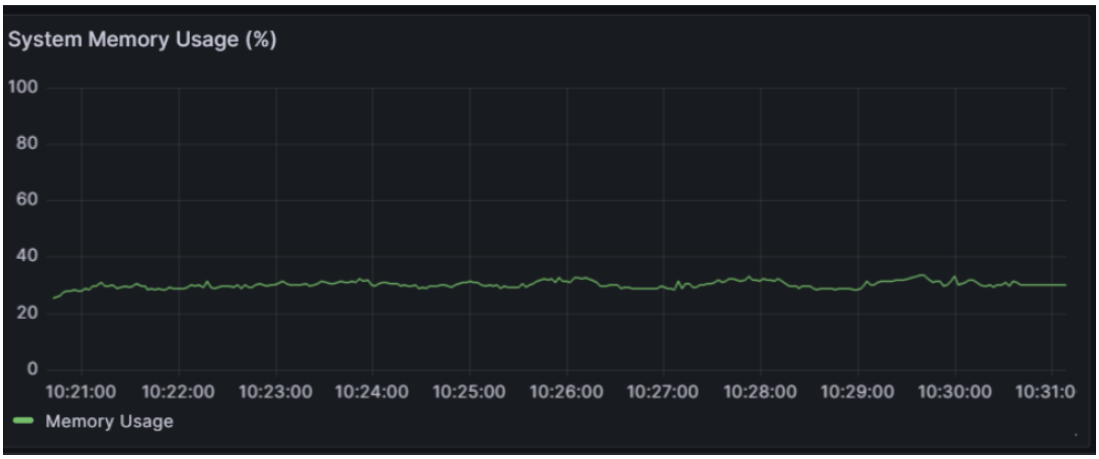
MD Core #2



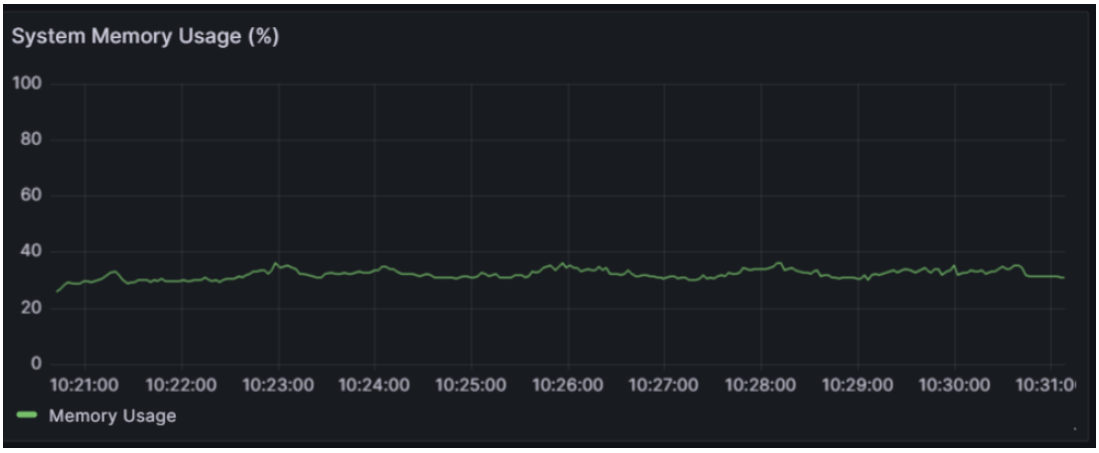
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

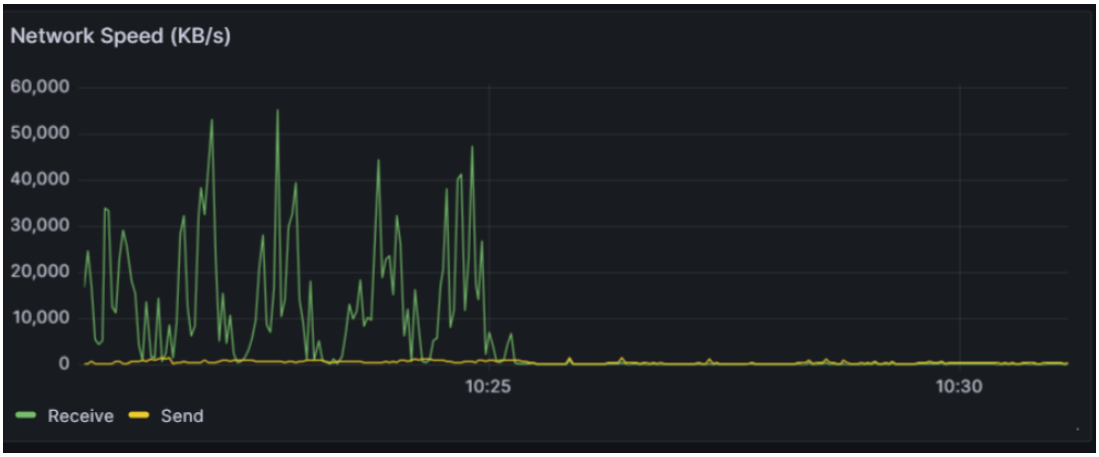


MD Core #2

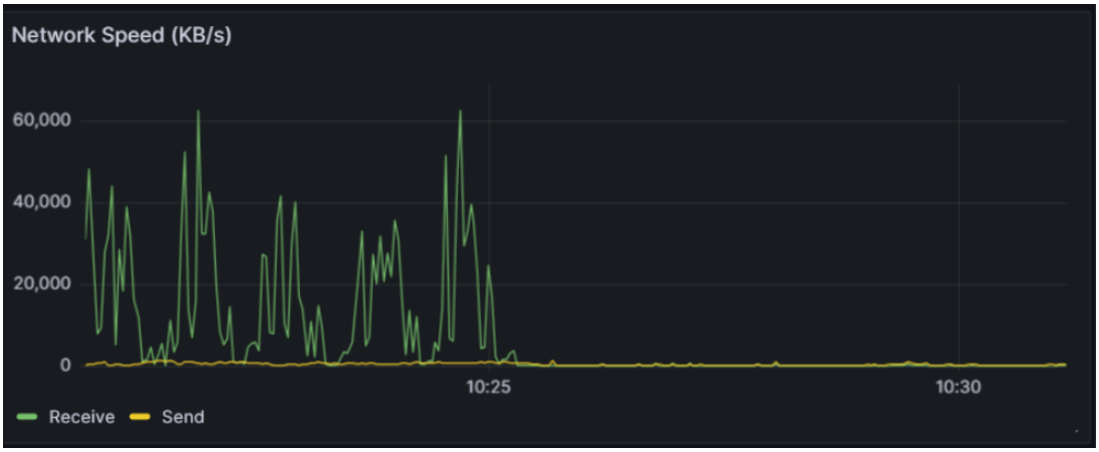


Network usage

MD Core #1



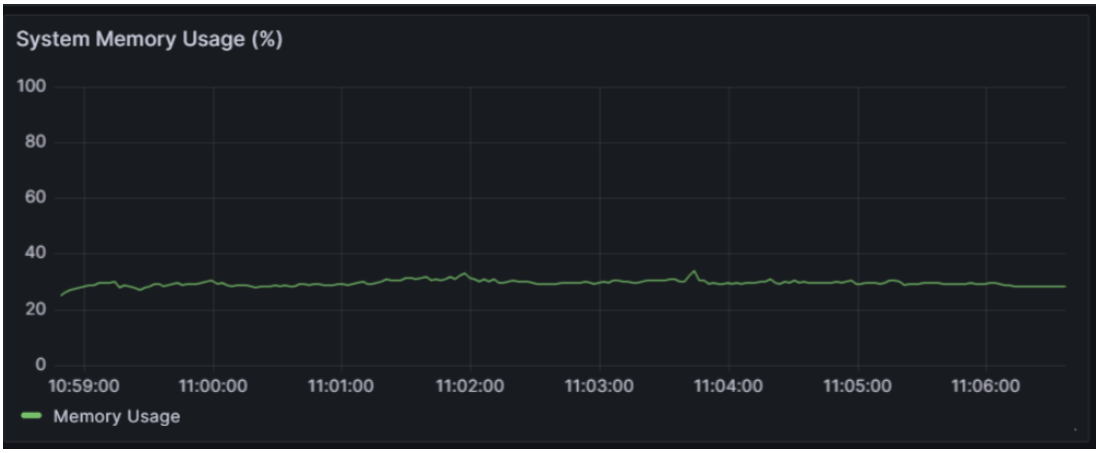
MD Core #2



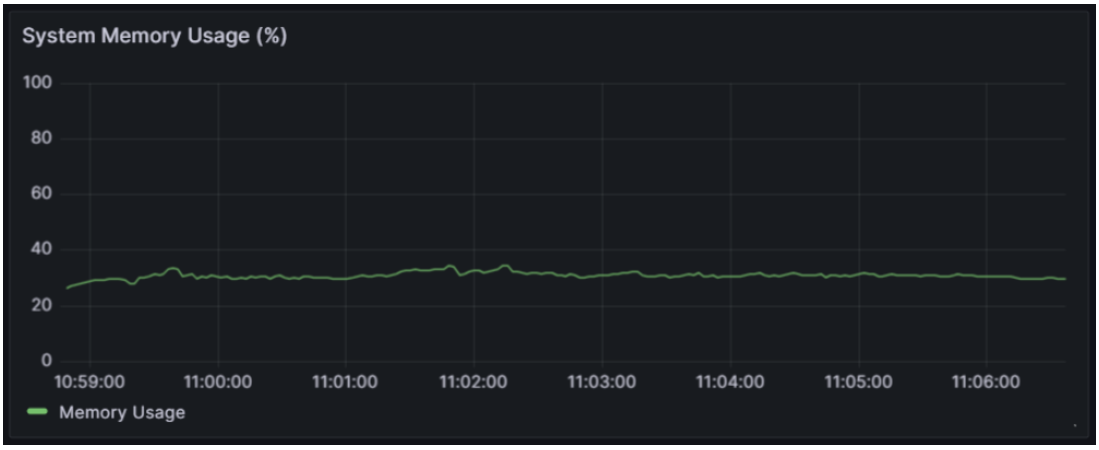
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

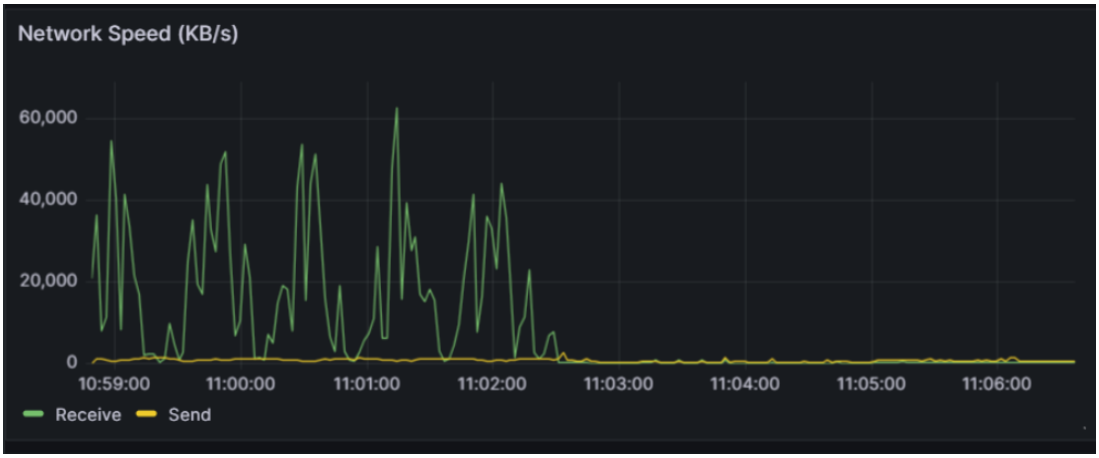


MD Core #2

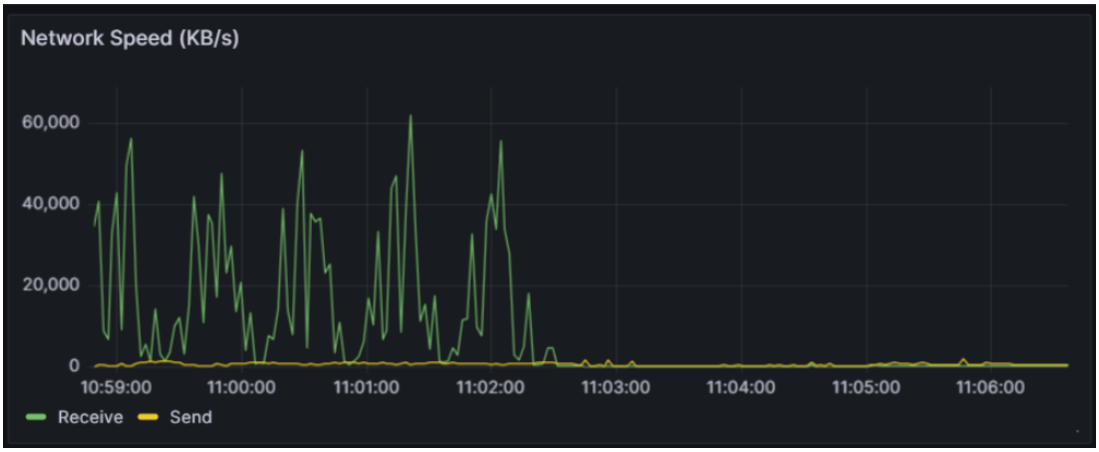


Network usage

MD Core #1



MD Core #2



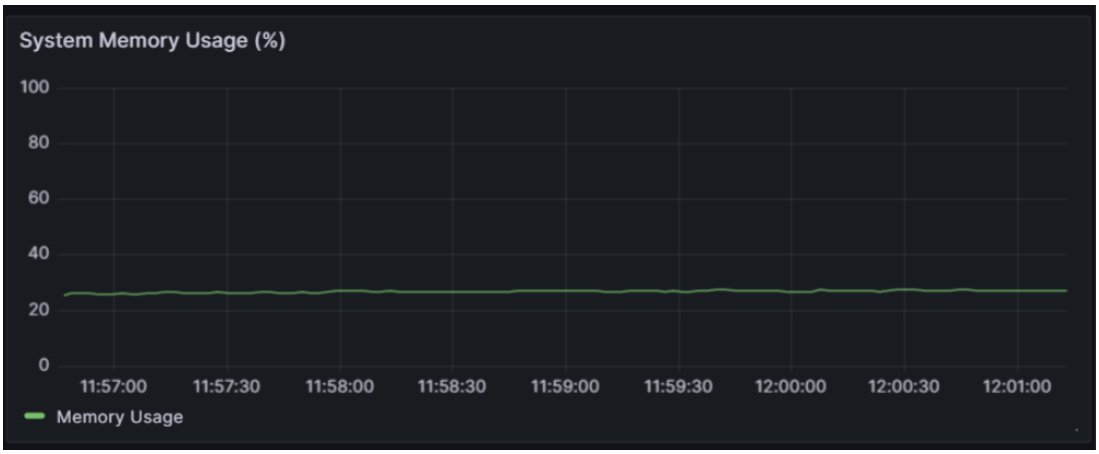
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

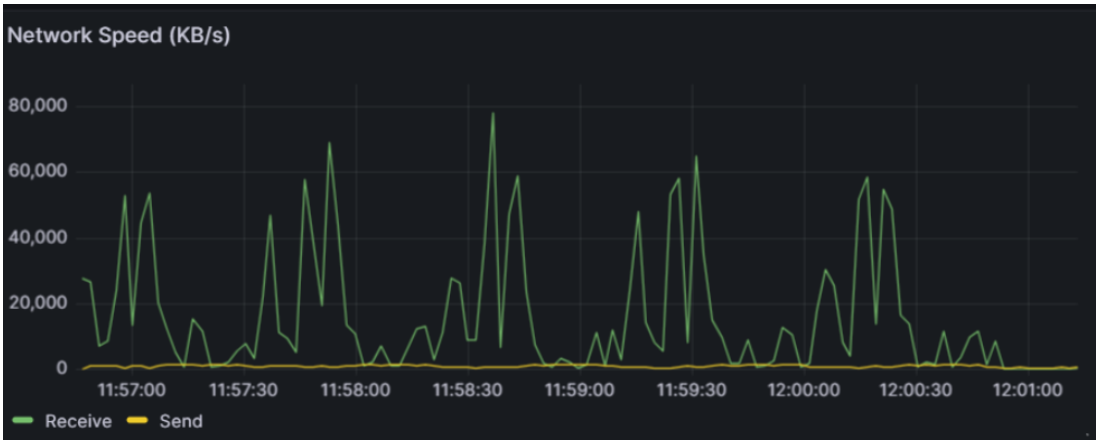


MD Core #2

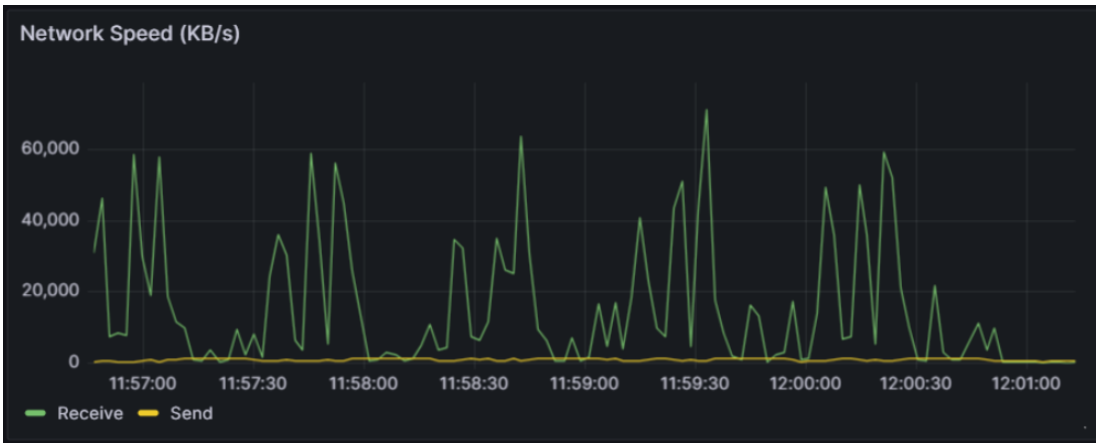


Network usage

MD Core #1



MD Core #2



Shared DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

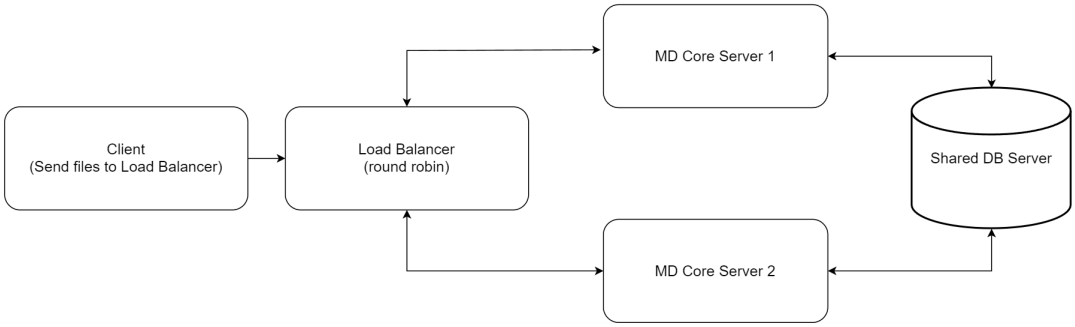
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV)	15.5	63.1	73.3	25.2	25.8	Receive: 4555.9 Send: 694.4	Receive: 4516.2 Send: 692.4	492,620.5	0.0
Deep CDR	9.70	50.7	51.2	31.9	32.5	Receive: 7062.9 Send: 623.2	Receive: 6653 Send: 654.1	786,189.6	0.0
Proactive DLP	7.2	35.2	35.0	27.9	27.5	Receive: 9047.4 Send: 802.4	Receive: 9859.8 Send: 787.5	1,055,110.8	0.0
Vulnerability	5.0	45.7	49.3	24.4	24.5	Receive: 13,585.7 Send: 966.7	Receive: 13,505.8 Send: 987.4	1,519,562.1	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV) + Deep CDR	19.9	79.1	77.6	34.0	34.2	Receive: 3785.9 Send: 573.8	Receive: 3663.1 Send: 577.7	382,716	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP	21.6	80.3	81.5	37.7	44.4	Receive: 3305.6 Send: 543.3	Receive: 3414.3 Send: 556.2	353,694.9	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	22.5	84.2	79.8	37.9	44.7	Receive: 3267.9 Send: 554.9	Receive: 3398.2 Send: 553.8	338,507.6	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

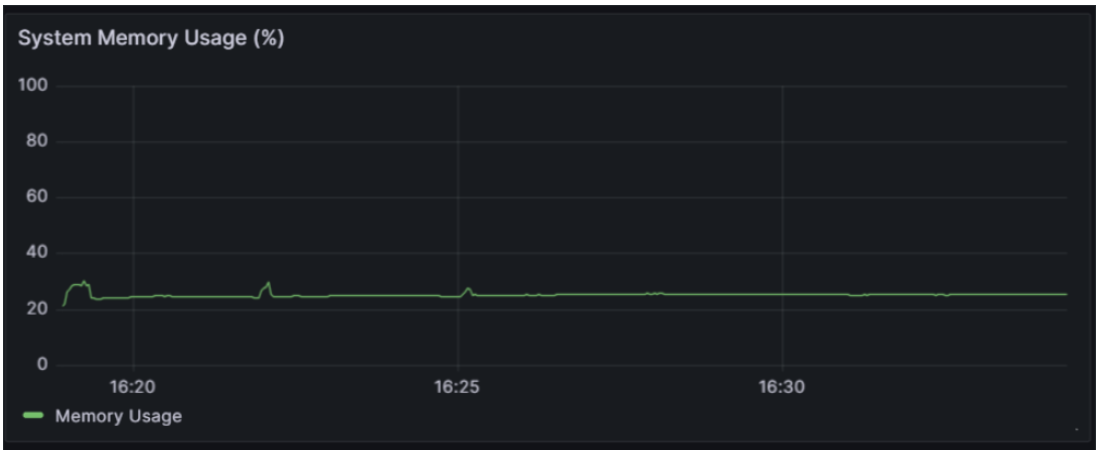
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

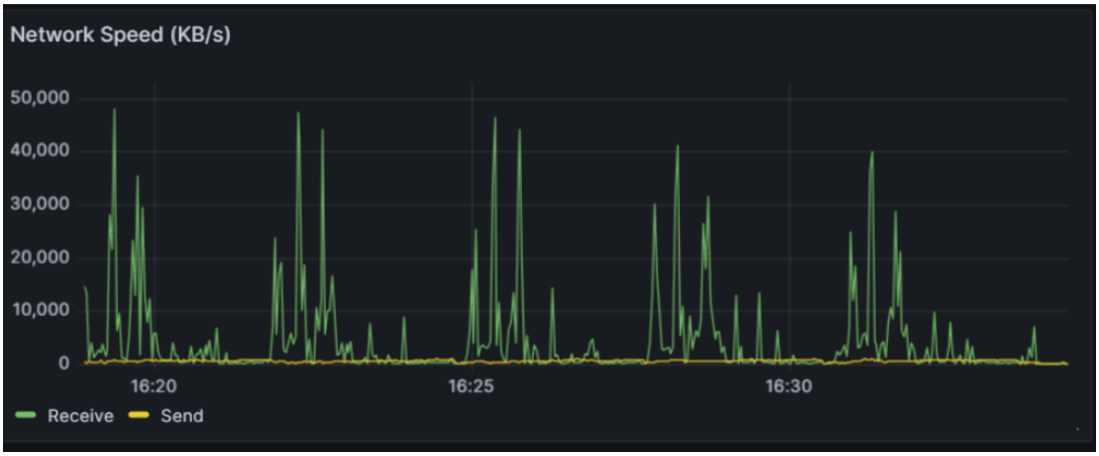


MD Core #2

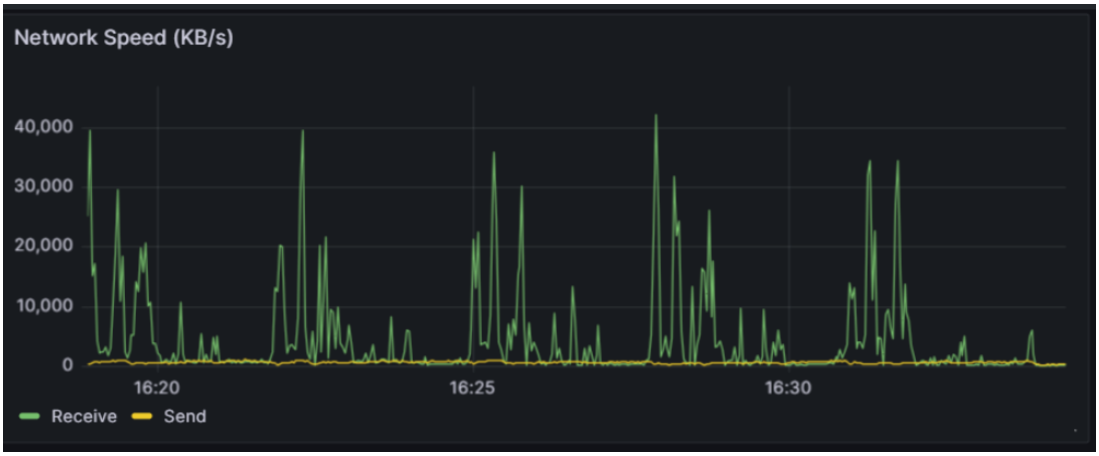


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

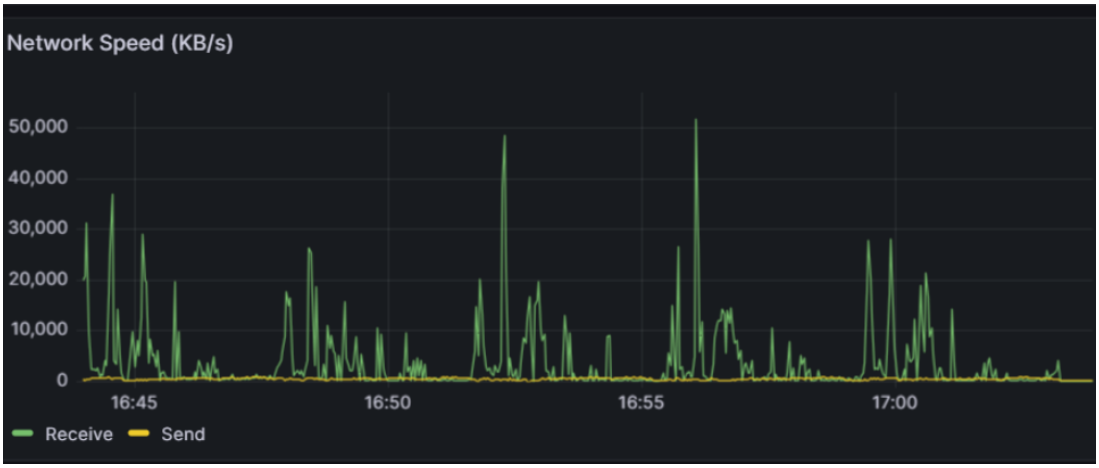


MD Core #2

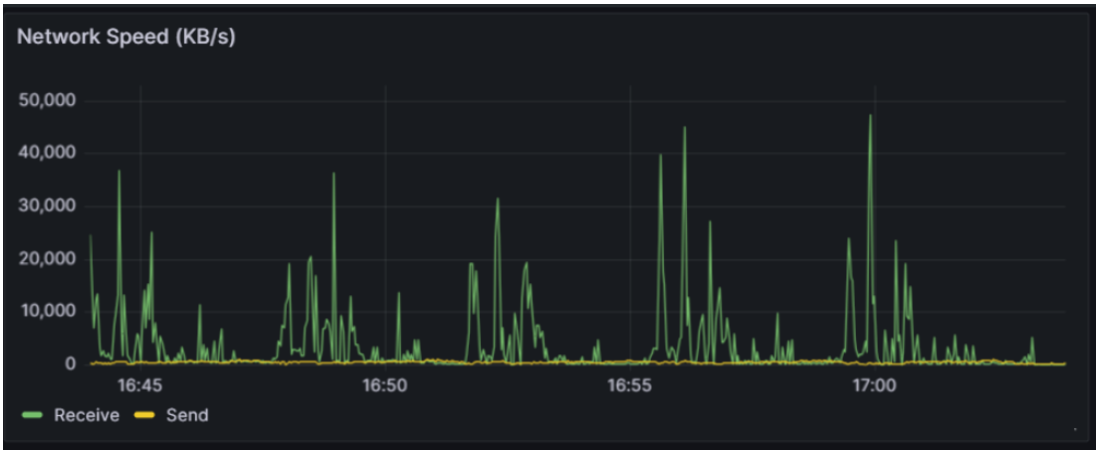


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1

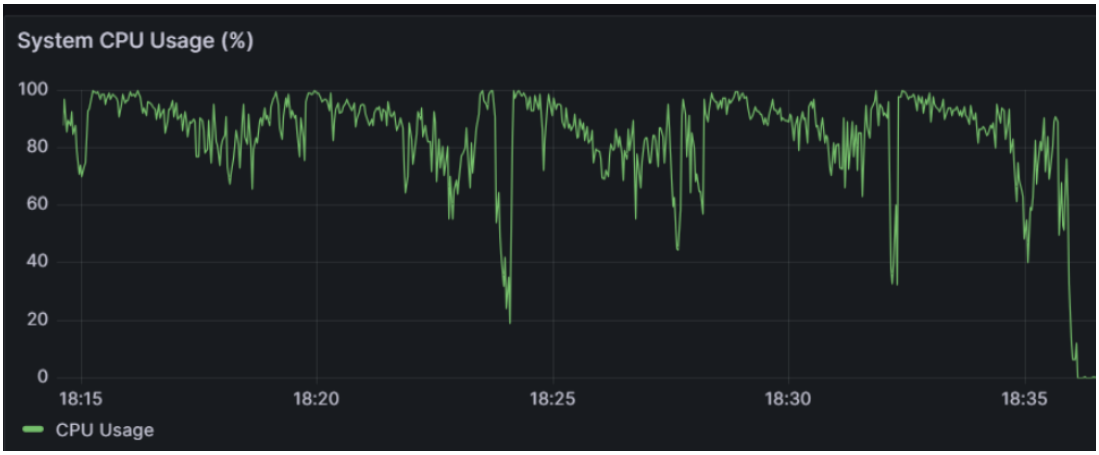


MD Core #2



CPU usage

MD Core #1

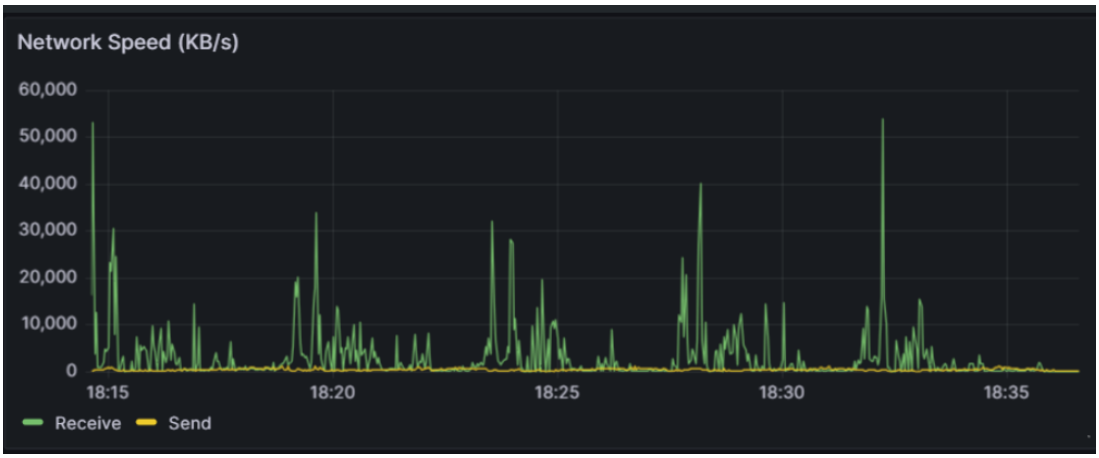


MD Core #2

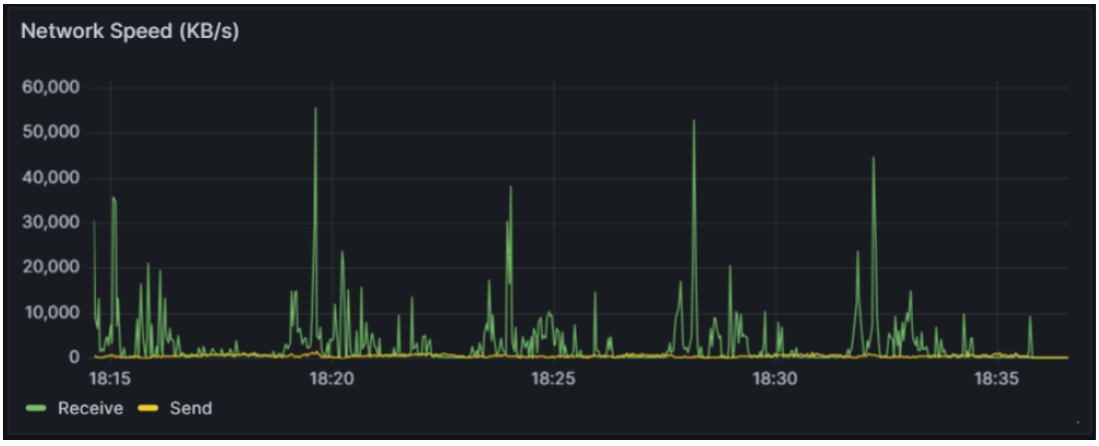


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

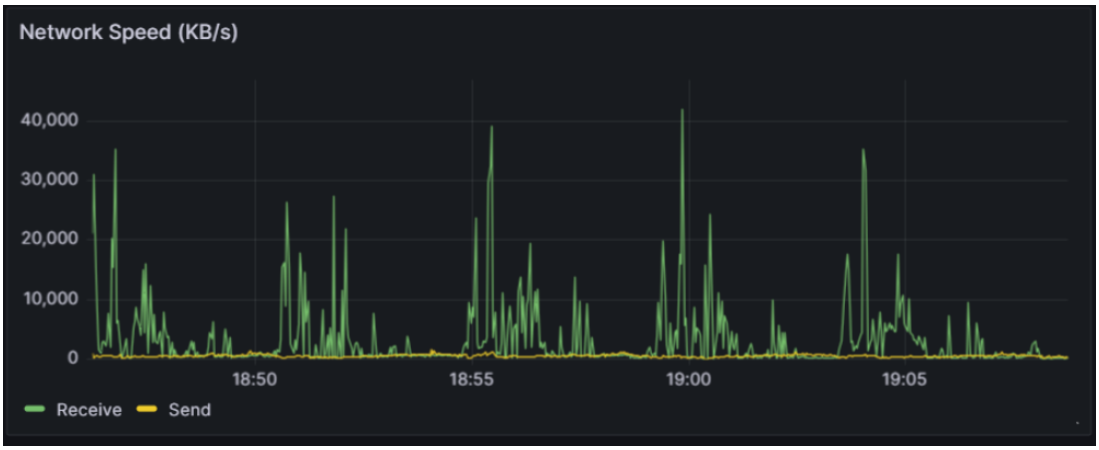


MD Core #2

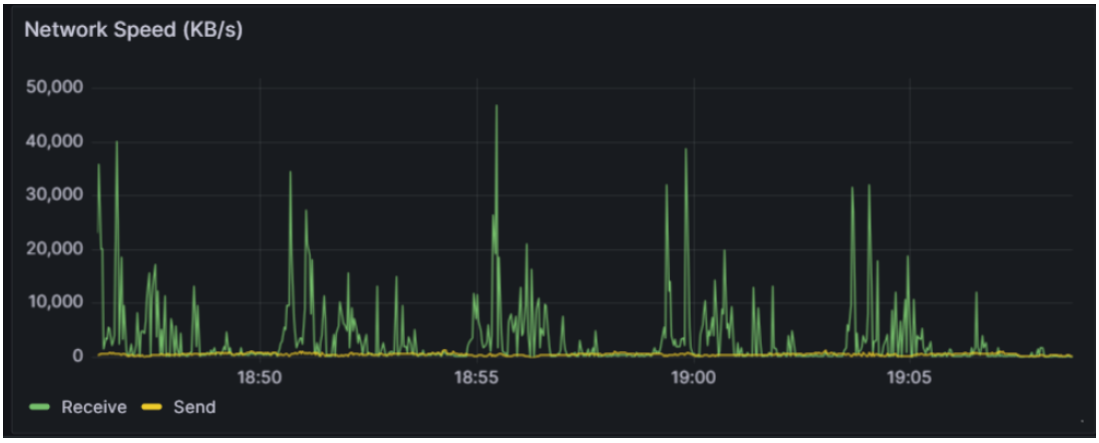


Network usage

MD Core #1



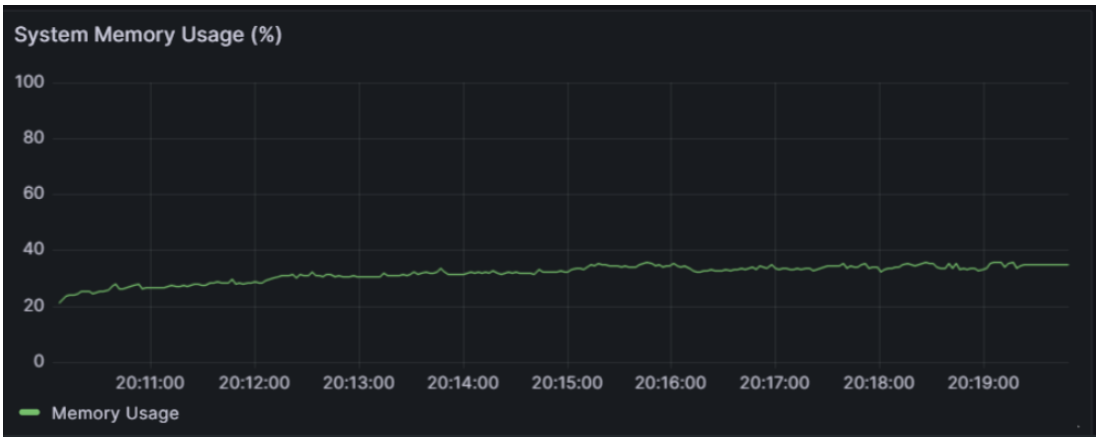
MD Core #2



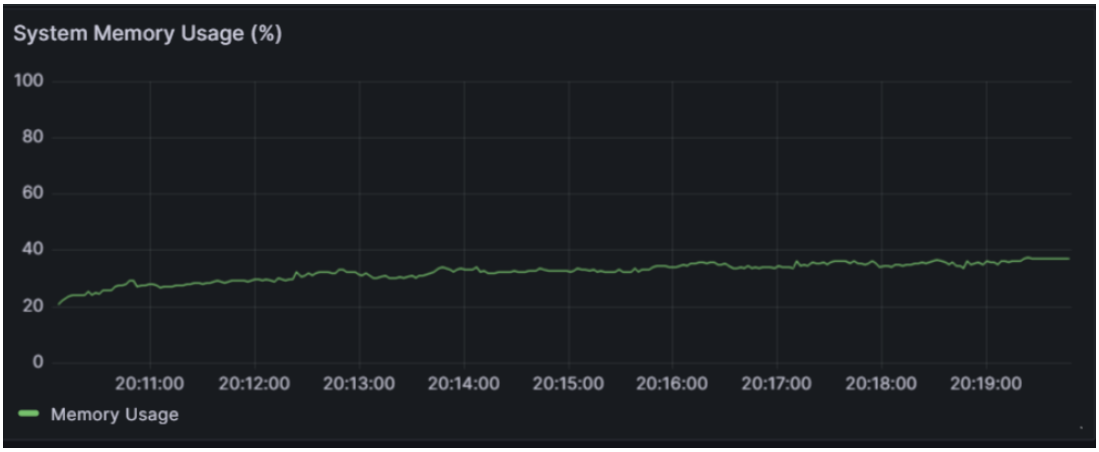
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1

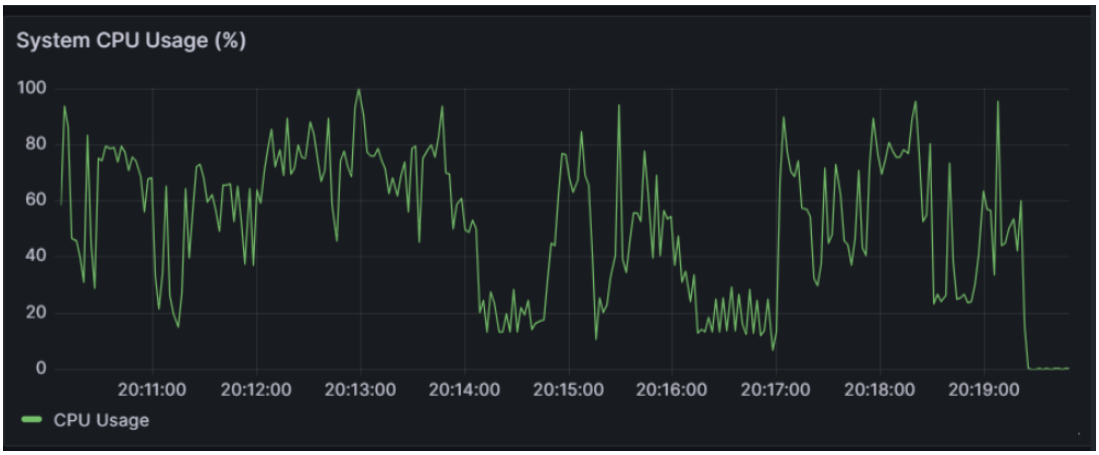


MD Core #2



CPU usage

MD Core #1

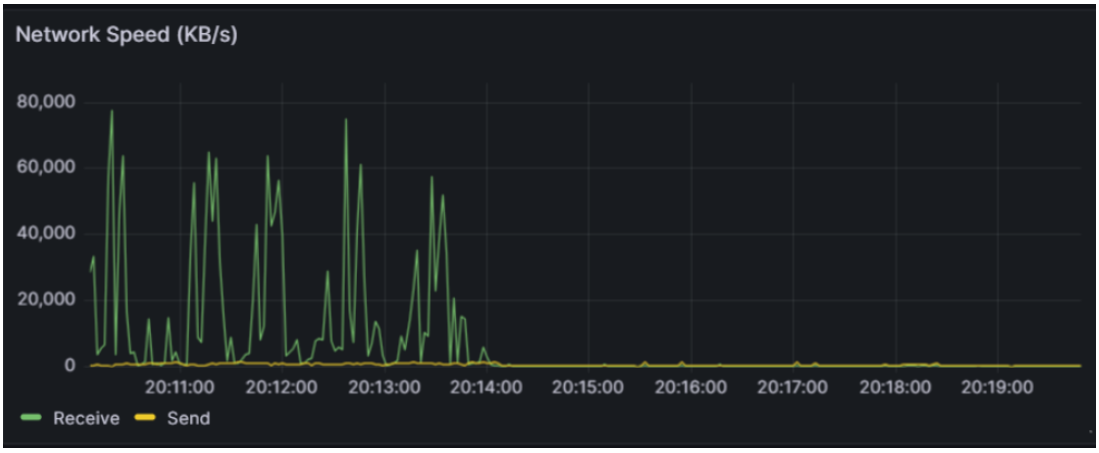


MD Core #2

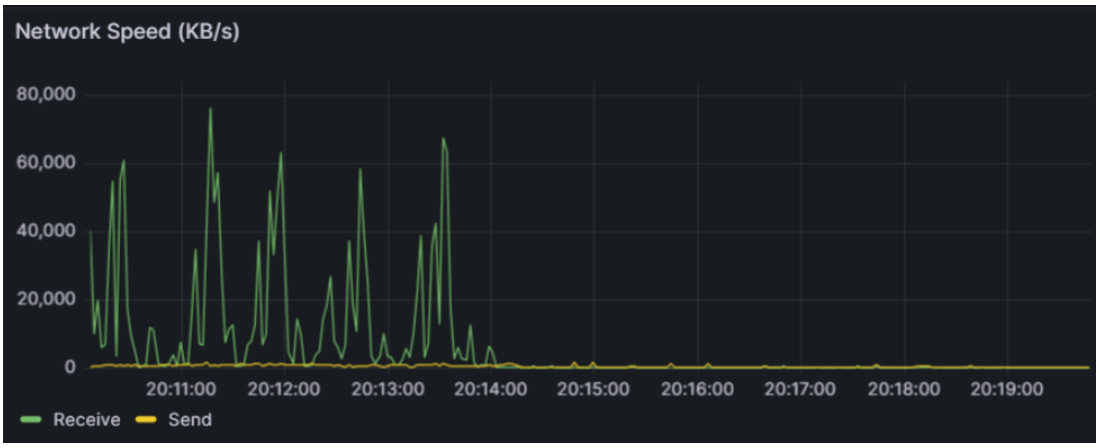


Network usage

MD Core #1



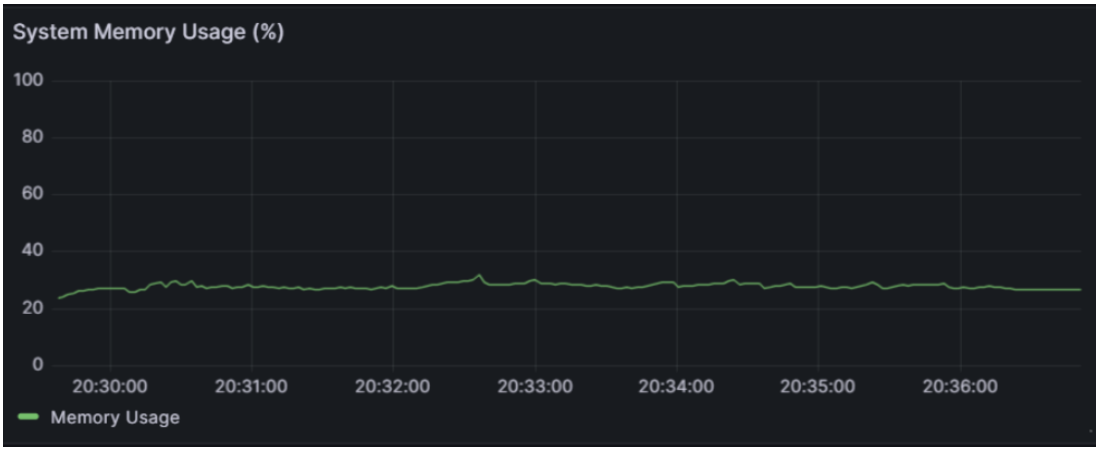
MD Core #2



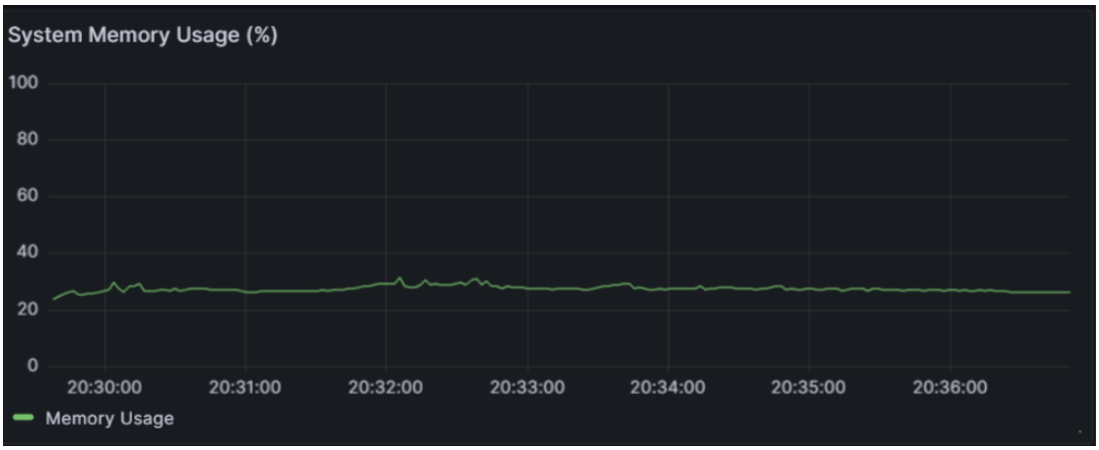
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1

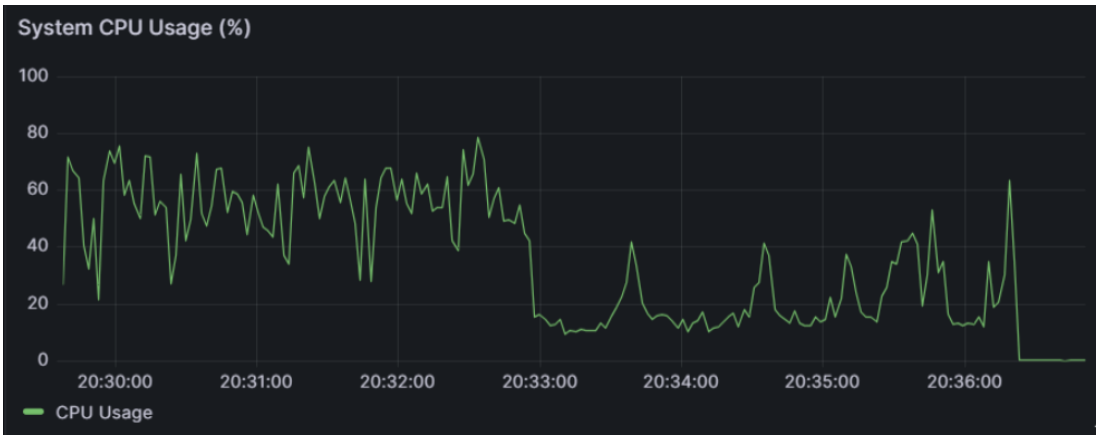


MD Core #2

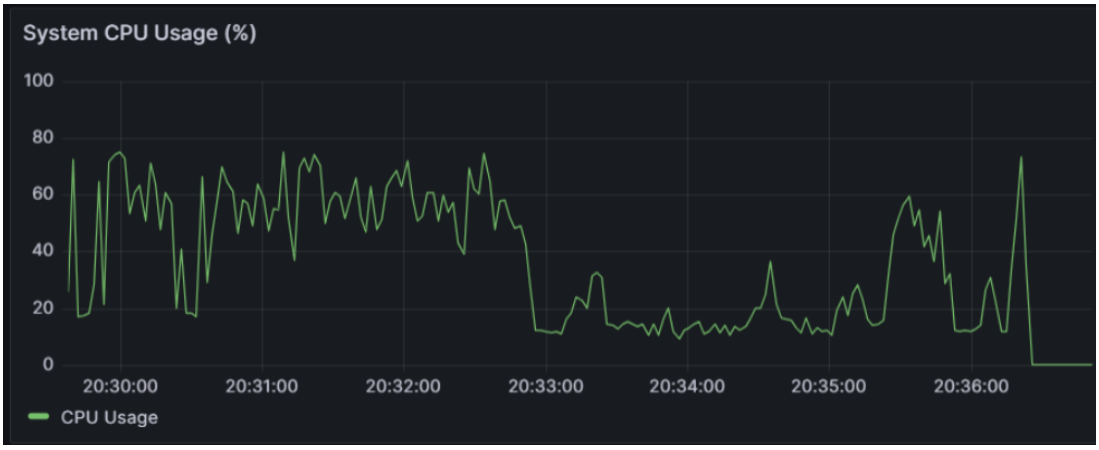


CPU usage

MD Core #1

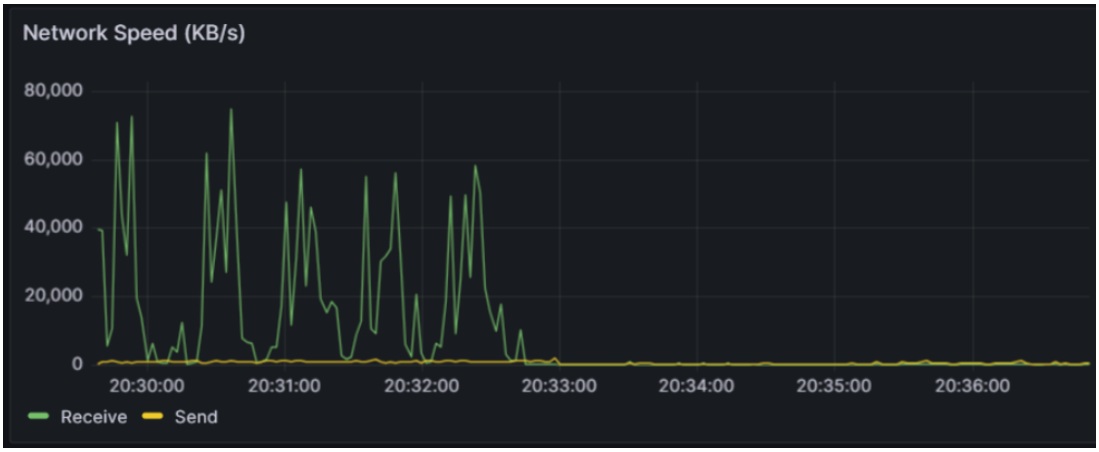


MD Core #2

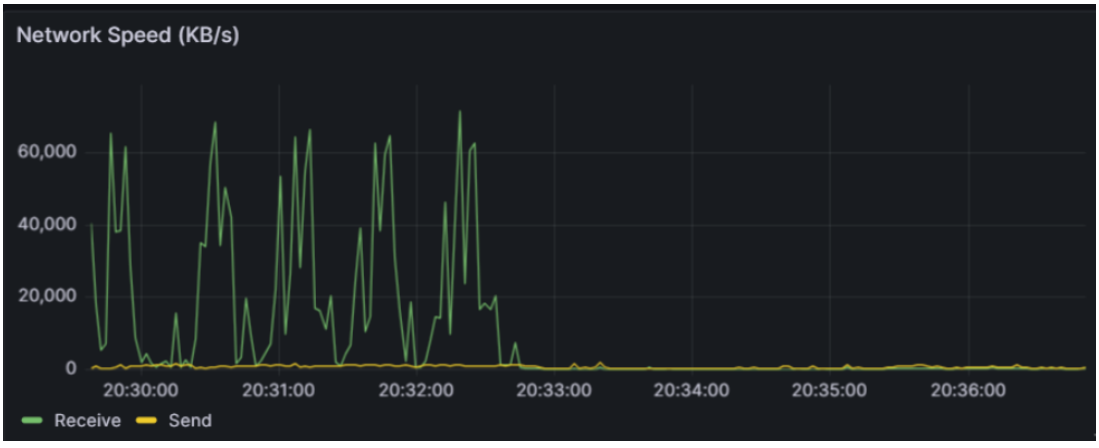


Network usage

MD Core #1



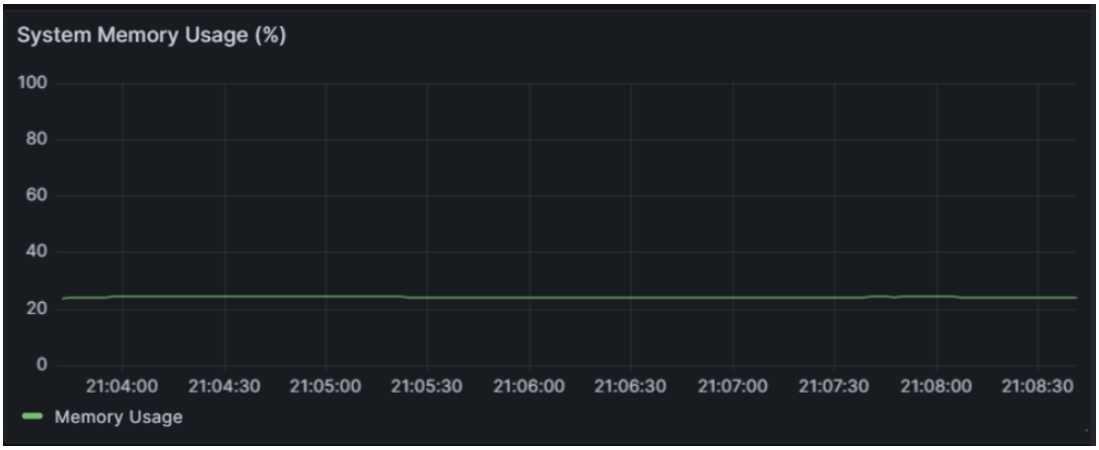
MD Core #2



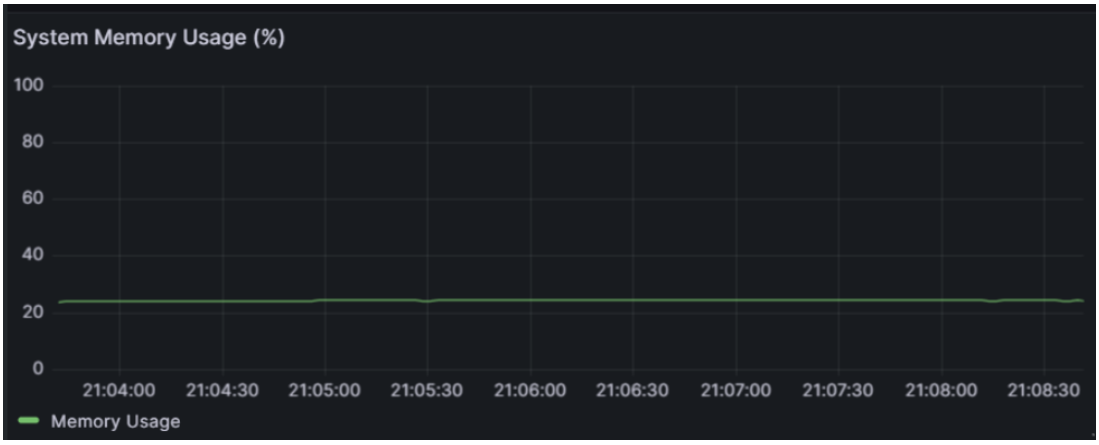
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

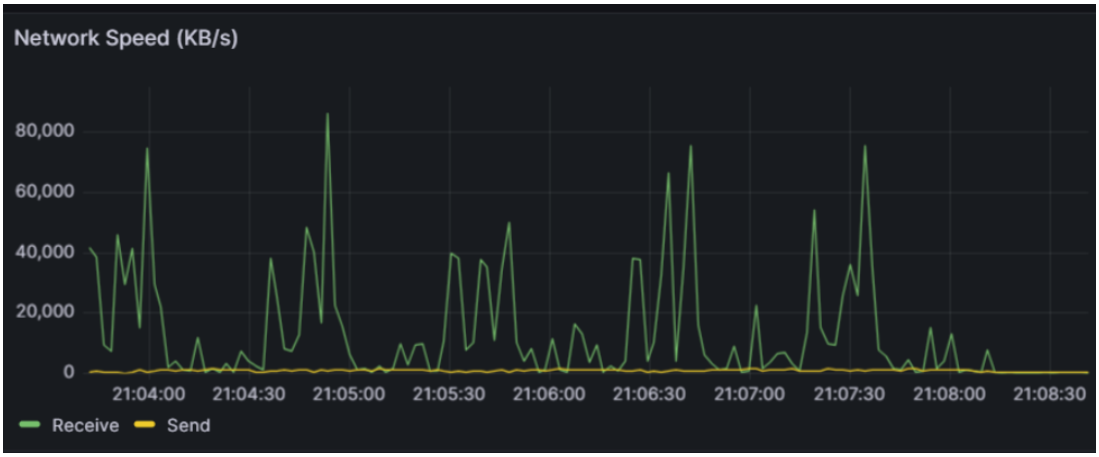


MD Core #2

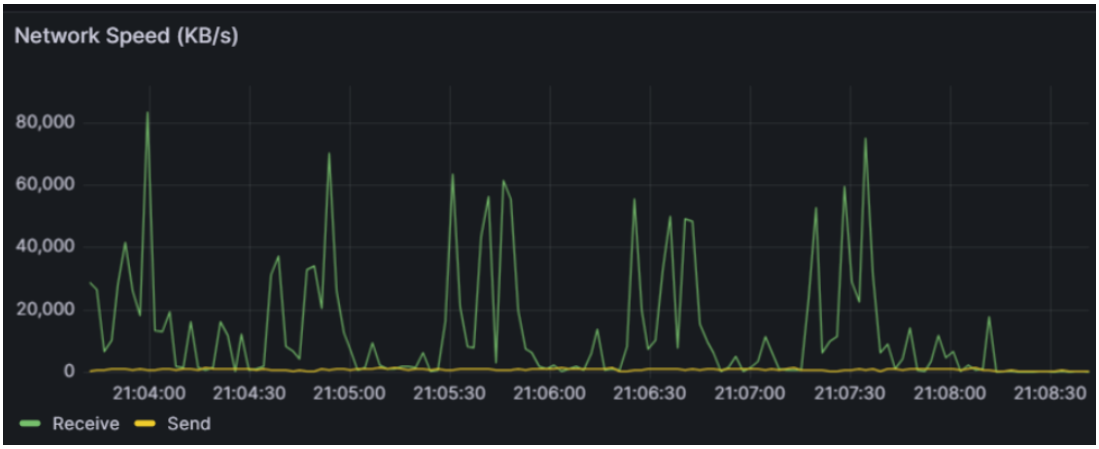


Network usage

MD Core #1



MD Core #2

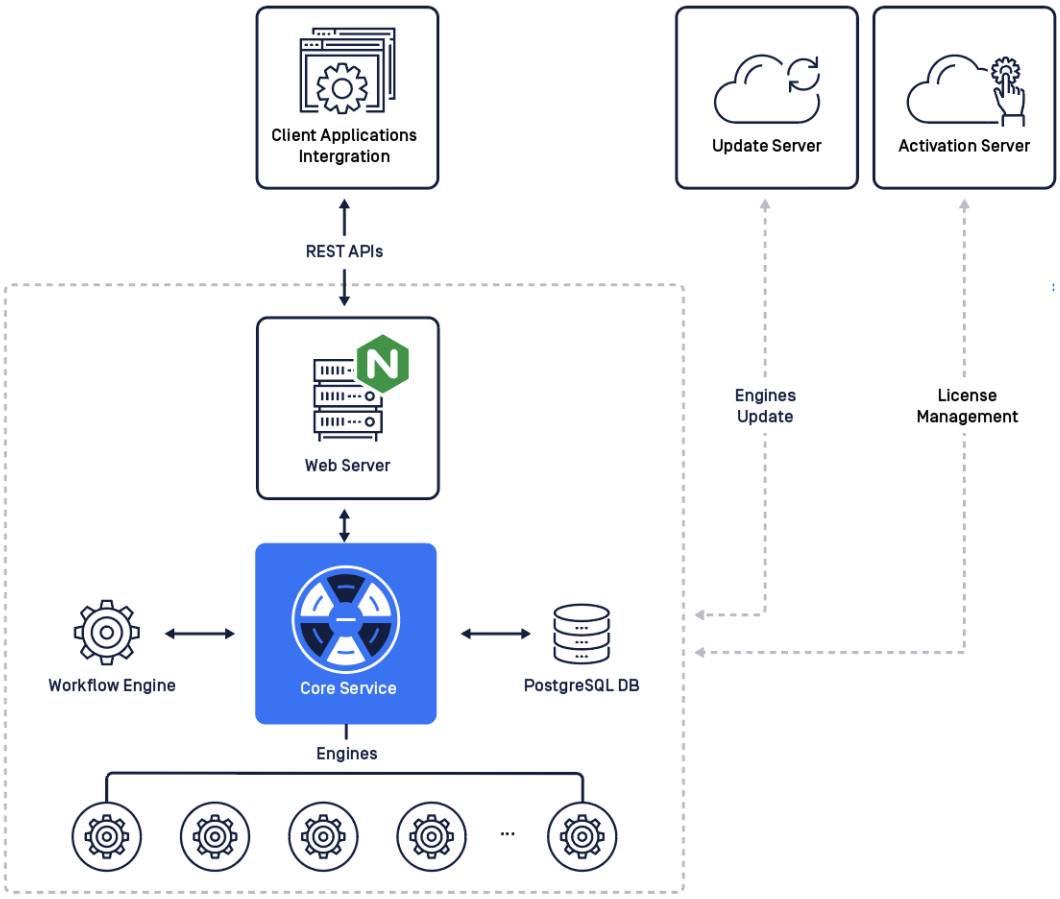

Network Speed (KB/s)
80,000
60,000
40,000
20,000
0
21:04:00
21:04:30
21:05:00
21:05:30
21:06:00
21:06:30
21:07:00
21:07:30
21:08:00
21:08:30
Receive
Send

MetaDefender Core Documentation

The users can consult this web page or, alternatively, they can download the manual in pdf format from the link below:

MetaDefender Core manual (SHA256:).

Recommended System Configuration



MetaDefender Core supports a wide range of deployment options, including air-gapped environments, providing the flexibility to choose the environment that best suits customers' requirements and infrastructure.

- Physical server.
- Virtualization platforms like VMware, Hyper-V and XenServer.
- Cloud Deployment.
- Container Deployment.

> **Note**
> Before installing MetaDefender Core, please refer to the **recommended minimum system configuration** listed below. The requirements are based on a standardized customer deployment and may require more or less resources to achieve the level of performance desired in your environment. Please note that the server specifications are built to allow a high volume daily processing. Please refer to Performance and Load Estimation for more details.

The recommendations below are for MetaDefender Core, API usage only. For any other use cases, please consult the user guide of the licensed products for accurate recommendations.

For certain use cases these might be adjusted and customized on specific needs and SLAs. We highly recommend to engage our ProServ team to assist in fine tuning MetaDefender and get the maximum performance out of your systems.

Microsoft Windows Deployments

[Windows] Supported Operating Systems

Only support 64-bit platforms.

- Windows 10, Windows 11 23H2, 11 IoT (22H2) only for End-User systems (e.g. Kiosk deployments)
  - Based on Microsoft License Terms, you're not allowed to use Windows Desktop as a server, therefore, MetaDefender Core can't be used on these systems for server use cases.
- Windows Server 2016, 2019, 2022

> **Disclaimer**
> End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

> **Plan for legacy OS support**
> OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:
>
> - Ubuntu 20.04 after **April 2025**.
> - Amazon Linux 2 after **June 2025**.
> - Windows 10 after **October 2025**.
>
> If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

[Windows] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 8	8	8 GB	16 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 12	16	16 GB	24 GB + 50 GB + 100 GB
MetaDefender Core 16	16	16 GB	32 GB + 50 GB + 100 GB
MetaDefender Core 20	32	16 GB	40 GB + 50 GB + 100 GB
MetaDefender Core MAX	32	32 GB	120 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core is installed.
- For a better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.
- SBOM processing requires more disk space, please allow at least 15GB free disk for the engine to work properly.

[Windows] Third Party Dependencies

- .NET framework 4.5 or above
- Microsoft Visual C++ Redistributable for Visual Studio 2015-2022 (version 14.38 or higher); x64 version.
- PostgreSQL: Only applicable when using a pre-installed PostgreSQL server running remotely.
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Some engines also have dependencies as described below:

Engine Name	Dependency
Deep CDR	Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 5.8.0 or above)
ESET	MetaDefender Core v4 temporary directory should have more than 200MB free disk space
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
Proactive DLP	• Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 2.0 or above) • .NET 6 dependencies • CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
Scrutiny	Microsoft Visual C++ 2015 - 2019 Redistributable x86 and x64
Systweak	.NET runtime 6.0, 7.0 or 8.0
Bkav Pro	Microsoft Visual C++ 2015 - 2022 Redistributable x64

[Windows] Installation Details

MetaDefender Core on Windows uses "C:\Program Files\OPSWAT" folder for storing resources or the installation directory.

MetaDefender will use its resources folder to store temp files as part of the analysis. **It's recommended to exclude this folder from real-time protection monitoring**. Knowledge base:

- How to set up exclusions on anti-malware software to prevent disruption
- Mitigating scan failure triggered by real-time protection service
- Can local AVs interrupt ongoing scans?

Linux Deployments

[Linux] Supported Operating Systems

Only support 64-bit platforms.

- Red Hat Enterprise Linux 8, 9
- Rocky Linux 9
- Debian 11, 12
- Ubuntu 20.04, 22.04
- Amazon Linux 2 is supported via Amazon Machine Image (AMI)

Plan for legacy OS support

OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:

- Ubuntu 20.04 after **April 2025**.
- Amazon Linux 2 after **June 2025**.
- Windows 10 after **October 2025**.

If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

For Hardended Linux OS

Some AV engines could be failed to initialize and run on MetaDefender Core (e.g. ESET, Kaspersky) due to execution permission required by them on `/var/tmp/ometascan` folder.

Please follow steps at Why have the ESET and Kaspersky scan engines failed to initialize on the hardened Linux OS? for remediation details.

Disclaimer

End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

[Linux] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 5	4	4 GB	10 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 10	8	8 GB	20 GB + 50 GB + 100 GB
MetaDefender Core MAX	16	16 GB	40 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core resides on.
- For better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.

[Linux] Third Party Dependencies

Dependencies list:

- openssl
- grep
- lib32stdc++6 (>= 4.5)
- libc6-i386 (>= 2.10)
- procps
- zlib1g
- libcurl3 (>= 7.19.7)
- libcurl4
- postgresql:
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Not all the above dependencies will need to be installed, it is dependent on different Unix distro and version

Some engines also have dependencies as described below:

Engine name	Dependency
Archive engine	• To extract password protected MS Office files, you need to install .NET 6 dependencies, libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • Rocky Linux 9.x/Red Hat 9.x: libnsl
Deep CDR	• Rocky Linux 9.x: libgomp, ncurses-compat-libs, libnsl • CentOS: libgomp, ncurses-compat-libs • Red Hat 8.x: libgomp, ncurses-compat-libs, libnsl • Red Hat 9.x: libgomp, ncurses-compat-libs, libnsl, compat-openssl11 • Ubuntu/Debian: libgomp1, libncurses5 • libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • .NET 6 dependencies
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
FileType engine	Rocky Linux 9.x/Red Hat 9.x: libnsl
Proactive DLP	libgdiplus, mscorefonts .NET 6 dependencies CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
RocketCyber	libgomp

Note: Some dependencies may need to be installed from external repositories, not from the OS default repositories.

[Linux] Installation Details

MetaDefender Core default installation path is using /var folder for storing resources:

- `/var/lib/ometascan` : installation folder with all its resources (database, DLP processed / CDR sanitized / quarantined file storage, engine package and definition updates).
- `/var/log/ometascan` : application logs.
- `/var/run/ometascan` : runtime folder for Core related processes.
- `/var/lib/ometascan` : all successful deployed engines data files.
- `/usr/lib/ometascan` : all shared libraries required by MetaDefender Core, bundled PostgreSQL server related binaries and libraries.
- `/var/tmp/ometascan/resources` : all temporary processing files for MetaDefender Core to process.

Storage Usage Information

Based on the configuration, MetaDefender Core will need additional disk space to store analysis/result data, quarantined files and temporary files:

	Data Storage
Analysis Reports (Processing History)	Approximately 4.5GB is required for every 1M analysis reports stored. • 100 GB is recommended as a baseline. Database storage requirements are affected by Data Retention settings, engine settings and scanning load.
Quarantined Files	Depends on the customers' dataset and configuration
Sanitized Files	Depends on the customers' dataset and configuration
Temporary processing files	Depends on the customers' dataset and configuration • A 1GB non-archive file requires 1GB free disk space at least. • A 1GB archive file requires (1 * 2)GB free disk space at least. • 50GB is a baseline and should be adjusted based on observation of the solution within the environment, with consideration to peak times and overall file makeup (e.g. complexity, size, archives).
Note	Options are available to control how large and/or complex files are handled to limit the amount of system resources required. See Workflow Configuration Template for more information.

Custom Engines

The recommendations above are specific for MetaDefender pre-packaged bundles.

However for additional Custom Engines, please review the Knowledge base to review additional requirements (if any) for the selected engine.

Browser Requirements for the MetaDefender Core Management Console

One of the following browsers is suggested to view the MetaDefender Core Management Console:

- Microsoft Edge
- Chrome
- Firefox
- Safari

Chrome, Firefox, Safari and Microsoft Edge browsers are tested with the latest available version at the time of release.

Network Bandwidth

In online environment where MetaDefender Core is configured to download engine update packages directly from OPSWAT update server source, then using gigabit network speed (125 MB/sec) is highly recommended to avoid potential network related issue while downloading files, or at least 100 Mbps (12.5 MB/sec) network speed.

Allowlisting URLs

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts':

- https://activation.dl.opswat.com - for license online activation
- https://update.dl.opswat.com - for engines and databases download

Note: *IP address-based allowlisting on your firewall might fail after some time since OPSWAT uses CDN (AWS Cloudfront) to faster delivery updates over the world, and IP address of edge servers might change over time.*

Depending on your location, connections with OPSWAT cloud-based servers be forwarded to specific AWS Cloudfront's edge locations, each is assigned with its own unique IP address ranges. You might also need to allow correct AWS Cloudfront's corresponding IP address ranges by referring to https://ip-ranges.amazonaws.com/ip-ranges.json

Warning

Even the OPSWAT update servers host updates for all of the available engines we support, sometimes custom engines might try to connect to their own cloud for updates, but this can be disabled in firewall and they will be updated just from OPSWAT cloud servers instead.

Differences Between MetaDefender Core v4 and v5

We are very excited to inform you about the release of version 5.0.0 of our MetaDefender Core!

The new version is designed to enhance the product performance, user experience, as well as to provide more flexibility, security, and scalability for our customers.

At the same time, we are also committed to keeping backward compatibility for whom running MetaDefender Core v4, and avoid the need for updating your client integration to move forward with MetaDefender Core v5.

v4 REST API Integration Supported
All public REST API endpoints available on MetaDefender Core v4 will be continously supported on v5.

v4 Product Major Features Supported
All product major features available on MetaDefender Core 4.21.2 (the latest one on v4) will be continously supported on v5.

Seamless Product Upgrade
Upgrading from MetaDefender Core v4 to v5 will be supported and covered with data migration without additional manual steps from users.

Licensing information will be also migrated.

Below are some highlights of the first v5 release, MetaDefender Core 5.0.0:

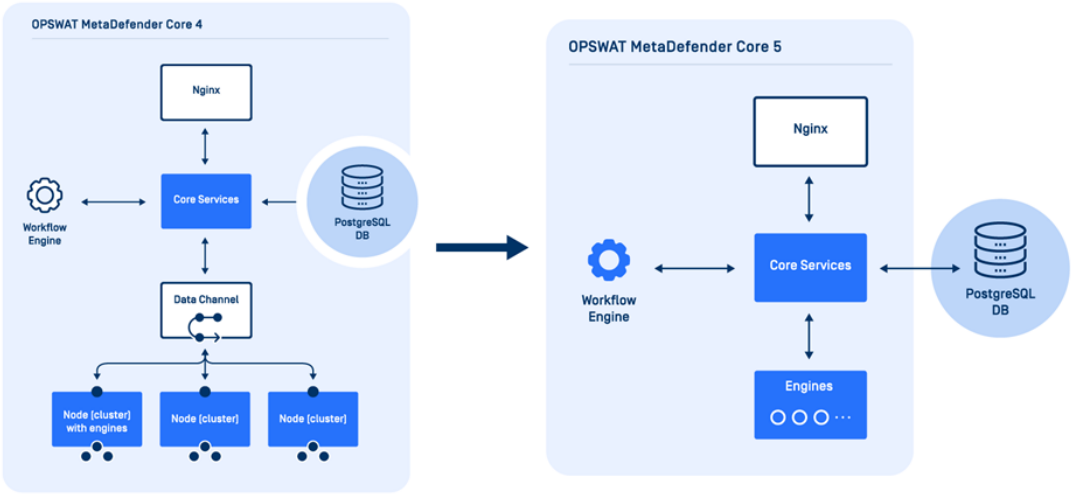
New product architecture

This version introduces a simpler product architecture that speeds up integration and automation, file processing flow, and immensely enhance the product performance and efficiency.

Earlier versions utilized MetaDefender Node concept, designed to support scaling of the scanning infrastructure by distributing scan requests among several different processing Nodes. Product licensing was based on number of Node instances. The benefit of having such a distributed infrastructure is that based on Node loads, Metadefender Core server can always choose the most appropriate Node to assign a new scan task to. However, having nodes in the v4 architecture caused more complexity and possible latency in the processing flow and entailed maintenance cost.

In this release, we retired multi-Node deployment and restructured MetaDefender Core's backend. This alteration significantly improves file processing speed and product efficiency, as well as eliminates extra cost for maintaining node services. Users now can simply install additional MetaDefender Core instances for scalability requirement,

as well as configure their own load balancer to coordinate tasks among multiple MetaDefender Core instances. Accordingly, product licensing is now based on number of the product instances.



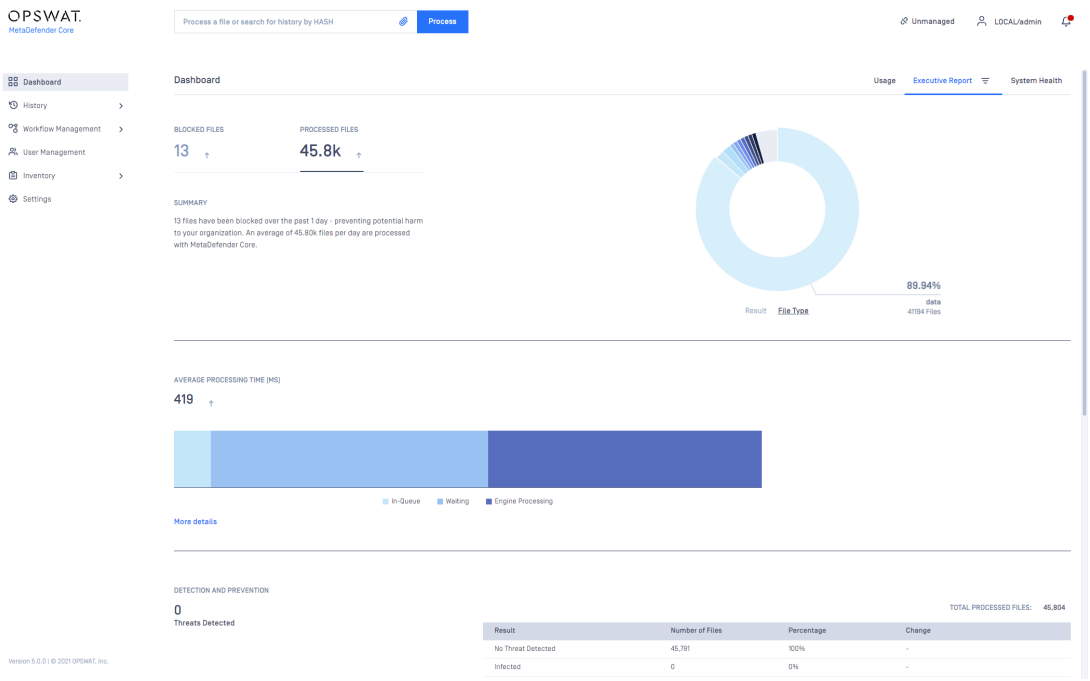
Please be informed that

- All existing administrative APIs related to "Node" on v4 will be kept as it is on v5 to ensure backward compatibility with existing client integrations.
- Applicable settings defined in Windows registry (Windows) or ometascan-node.conf config file (Linux) for Node will be migrated to Core. Details: MetaDefender Configuration

Brand-new product user interface

The second notable update in this version is a brand-new UI for the management console, which was designed with the focus on the quality and thoughtfulness of the user experience. The new administrative console UI provides an advanced set of Core features and observable processing status.

Comprehensive executive dashboard reporting offers dynamic, real-time and actionable threat insights. It also provides visibility of your security posture allowing you to paint a picture of security that aligns with the overall business. The report includes easy-to-read charts, graphs, and statistic explanation that help your security professionals reduce clutter, gather information quicker, spot threat activity early and act swiftly.



We're continuing to improve our product UI to provide users an excellent experience with the application and overall usability. Please expect more powerful UI features coming in next releases.

New compression archive engine for archive sanitization

In previous version, MetaDefender Core's archive engine implemented four functions including decryption, extraction, encryption and compression that with a high volume of files it takes time to process them all. For the purpose of enhancing archive handling performance, this release comes with a separated compression archive engine, which dedicatedly processes archive sanitization (compression and encryption). It is also created to facilitate maintenance and further feature development.

Archive sanitization feature now requires Deep CDR license

As to respect sanitization practice and ensure unknown threats prevention as designed, now starting on version 5, it's mandatory to have Deep CDR licensed for archive sanitization feature enablement on MetaDefender Core workflow configurations.

Archive handing related engines changed

The legacy "Archive" engine on MetaDefender Core v4 will be changed its name to "Archive Extraction" engine, even when you are still on MetaDefender Core v4.

A brand new "compression archive engine" is to dedicatedly handle archive sanitization in v5 (archive compression), that helps relevant archive handling performance improvement, easier for maintainance and further feature development.

Notes when you are still on MetaDefender Core v4 (not upgraded yet)

The legacy "Archive" engine (now name changed to "Archive Extraction" engine) on MetaDefender Core v4.x will still cover both extraction and compression archive tasks as usual.

Even though your MetaDefender Core v4.x might now have a new "Archive Compression" engine auto downloaded, but that engine cannot be deployed (marked as "permanently failed" in relevant engine status check API). This behavior is expected, and not causing any impact to MetaDefender Core processing flow, or its performance though.

- Just in case your client integration has a check for engine status before triggering scan request, then you might want to disable "Archive Compression" engine status on MetaDefender Core v4 using REST API. Detailed instruction: Archive Compression Engine Permanently Failed on MetaDefender Core v4
- When you later upgrade to MetaDefender Core v5 from v4, then you might need to manually re-enable "Archive Compression" engine to make it active and running on MetaDefender Core v5.

Notes for licensing

For offline deployment (MetaDefender Core is not connected to the Internet directly for online update), you will need to re-activate again on OPSWAT portal to retrieve latest engine list in .yml file including "Archive Compression" engine if you are currently licensed with Deep CDR.

Meanwhile, for online deployment, this manual procedure is not required since MetaDefender Core will cover that itself.

Import & export configurations

Not only workflow settings like in previous versions, but also all other general settings (update, security, data retention, server, hashes to skip) and most importantly, user settings, will be able to export and import on MetaDefender Core version 5.

Learn more at Import/Export configuration

Backward compatibility supported

The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

Windows application event logging disabled by default

Applicable only to Windows OS based MetaDefender Core.

Help avoid duplicated logging with separate file option, and improve overall processing performance. Of course, when being still valid to you, you are still supported to enable it back.

CSRF token header renamed

CSRF token originally created for security strengthening and being used by MetaDefender Core management console UI since version 4.21.0 via header `oms_csrf_token`

To get it compliant with some load balancers which do not accept underscore in header name by default, and save you from additional load balancer configuration step, we have now changed the header name to `oms-csrf-token`

This change does not impact to any REST API based client integration.

More comprehensive and accurate error messages on REST APIs

Impacting on following REST API endpoints:

- `PUT /admin/config/session`
- `PUT /admin/config/update`
- `PUT /admin/config/scan`
- `PUT /admin/config/skip`
- `PUT /admin/config/auditlog`
- `PUT /admin/config/webhook`
- `PUT /admin/config/rule`

Upload time and queue time logic updated in JSON scan response

Impacting on following REST API endpoints:

- `GET /file/{data_id}`
- `GET /hash/{md5/sha1/sha256}`
- `GET /archive/{data_id}`
- `POST /file/{async}`

Added new field:

- `receive_data_timestamp`: The timestamp when upload progress started (first byte received) (in milliseconds)
- `upload_time`: Total time elapsed for upload process (in milliseconds)

Modified existing field (new logic):

> **Backward compatibility supported**
> The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

- `upload_timestamp`: The timestamp when upload progress finished (all bytes received) (in milliseconds)
- `queue_time`: Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue time = start_time - upload_ timestamp) (in milliseconds)

Release version 5.0.0 also comes with variety of outstanding features and improvements.

For detailed information about our latest release please view our Release notes

Installation

Starting with MetaDefender Core 4.20.0, the centralized/shared database deployment model is supported (all MetaDefender Core instances share the same database), along with traditional standalone database mode (every MetaDefender Core instance has its own database to manage).

- Standalone mode
- Shared Database Mode

Installation

1. Download the package of your choice from My OPSWAT
2. Install the package on your computer via the Command Line or via the Wizard
3. Open a web browser and point to http://<server name or IP>:<port>
    - The default port is **8008**
    - In case of problem check Inaccessible Management Console page
4. Complete the required steps of the basic configuration wizard
5. You must activate this deployment to use its features

Installation notes

- If the MetaDefender Core package **dependencies are not installed** on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.
- During installation the databases might need to be upgraded. This could take noticeable time depending on database size (eg.: length of scan history).

Standalone mode

Wizard Installation

The Install Wizard is only for the Windows installer (.msi file).

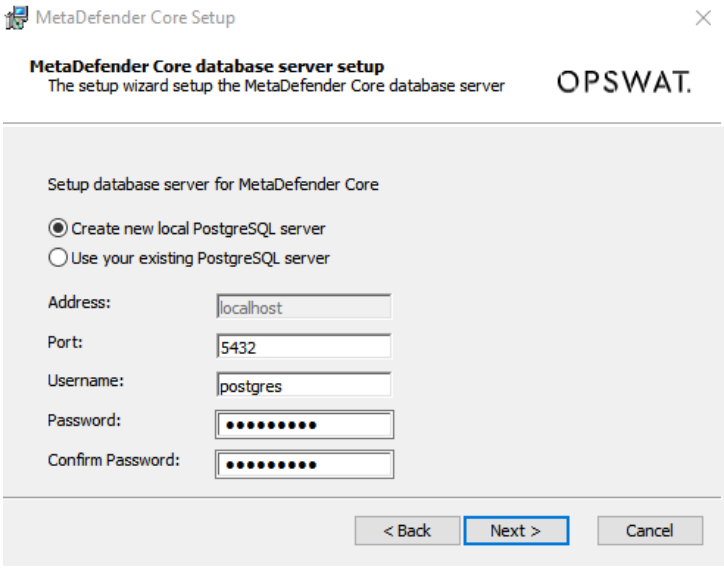
Since MetaDefender Core 4.19.0, we have switched to using PostgreSQL DBMS which needs a dedicated database server. MetaDefender Core supports two modes to setup PostgreSQL server:

Local (Bundle) PostgreSQL server

MetaDefender Core will install a new PostgreSQL server locally in the same box with the product.

> **PostgreSQL user credentials**
> Unicode characters are NOT supported for "Username" and "Password"
>
> "Password" does not accept # (hash) as starting character (e.g. #ABC)

> **The ignition file**
> If the ignition file exist in the designated location, then MetaDefender Core will use the details defined in the ignition file, rather than what is passed via the installation wizard.
>
> More info about the ignition file based installation: Command Line Installation

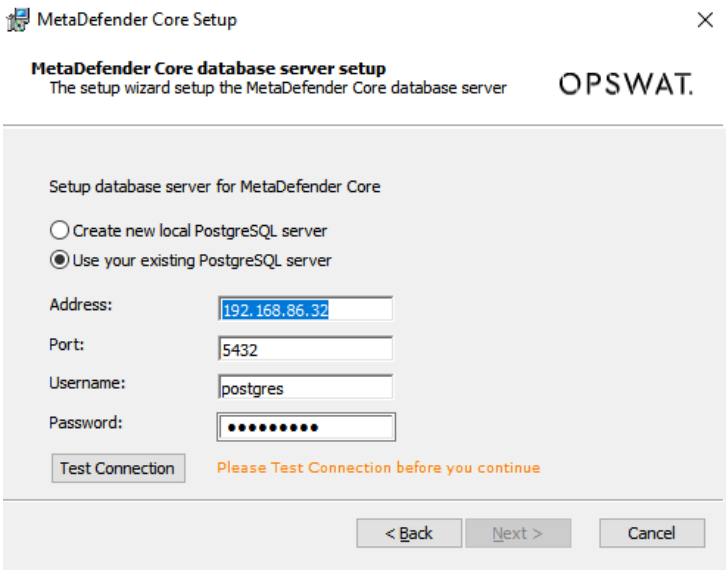


Remote (Existing) PostgreSQL server

MetaDefender Core will connect to remote pre-installed PostgreSQL running remotely.

> **PostgreSQL connection configuration**
> "Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

> **PostgreSQL user privilege requirements**
> MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.
>
> Details: PostgreSQL User Privilege Requirements



Command Line Installation

Ignition File

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Make sure to create the ignition file, if it doesn't exists before you go ahead and install MetaDefender Core via command line.

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

MetaDefender Core supports two modes to setup PostgreSQL server:

1) Local PostgreSQL server

- MetaDefender Core will install a new PostgreSQL server locally, next to the product.
  - A sample iginition file for PostgreSQL server information (***Only non-Unicode characters supported for "user" and "password"__***):

ometascan.conf bash

```
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
```

2) Remote PostgreSQL server

> **Info**
>
> **PostgreSQL version 12 (at least 12.3) or version 14 is required.**
>
> According to PostgreSQL notice, version 12 will stop receiving fixes on November 14, 2024, so we suggest that you use version 14.

MetaDefender Core will connect to a pre-installed PostgreSQL server running remotely and will setup its database.

Make sure to follow steps at Open Connection On PostgreSQL Server first, so that your remote existing PostgreSQL server to accept upcoming connection from MetaDefender Core.

A sample ignition file for PostgreSQL server information:

ometascan.conf bash

```
[dbserver]
type=remote
host=192.168.86.32
port=5432
user=existingserver_admin_user
password=existing_server_pass
```

After installing MetaDefender Core successfully, you may want to remove all credentials info created for [dbserver] section in ignition file for security reason.

> **Warning**
> If the MetaDefender Core package dependencies are not installed on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.

Debian package (.deb)

bash

```
sudo dpkg -i <filename> || sudo apt-get install -f
```

On Red Hat Enterprise 7.x Linux / CentOS package (.rpm)

bash

```
sudo yum install <filename>
```

For systems which enabled GPD check flag:

bash

```
sudo yum install --nogpgcheck <filename>
```

Red Hat Enterprise Linux 8.x package (.rpm)

Install the latest PostgreSQL repository RPM:

> **Note**
> Due to the shorter support cycle on Red Hat Enterprise Linux, all supported versions of PostgreSQL are not available on this platform. The PostgreSQL Yum Repository will integrate with the normal systems and patch management, and provide automatic updates for all supported versions of PostgreSQL throughout the supported lifetime of PostgreSQL. Reference: https://www.postgresql.org/download/linux/redhat/

bash

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

In case installing a remote PostgreSQL on either the same MetaDefender Core server or another server, you need to install the following additional package. PostgreSQL version 12.3 or newer is recommended.

bash

```
sudo dnf install -y postgresql12-contrib
```

Install MetaDefender Core:

bash

```
sudo dnf install -y <filename>
```

Windows package (.msi)

On Windows systems it is possible to install the product by running the corresponding .msi file.

From command line interface it is also possible to install the product by executing

bash

```
msiexec /qn /i <msi file name> <option key>=<option value>
```

where the possible keys and their default values are the following:

Key	Default Value	Description
INSTALLFOLDER	\Program Files\OPSWAT\MetaDefender Core	Customize installation folder for product Example: `INSTALLFOLDER="D:\Products"`
RESTADDRESS	*	REST interface binding IPv4 or IPv6 address ('*' means that service listens on all IPv4 and IPv6 interfaces)
RESTPORT	8008	REST interface binding port
REPORT_ENGINE_ISSUE	true	Enable reporting of engine issue count. (possible values: "true" or "false").

Note

Reporting of engine issue count

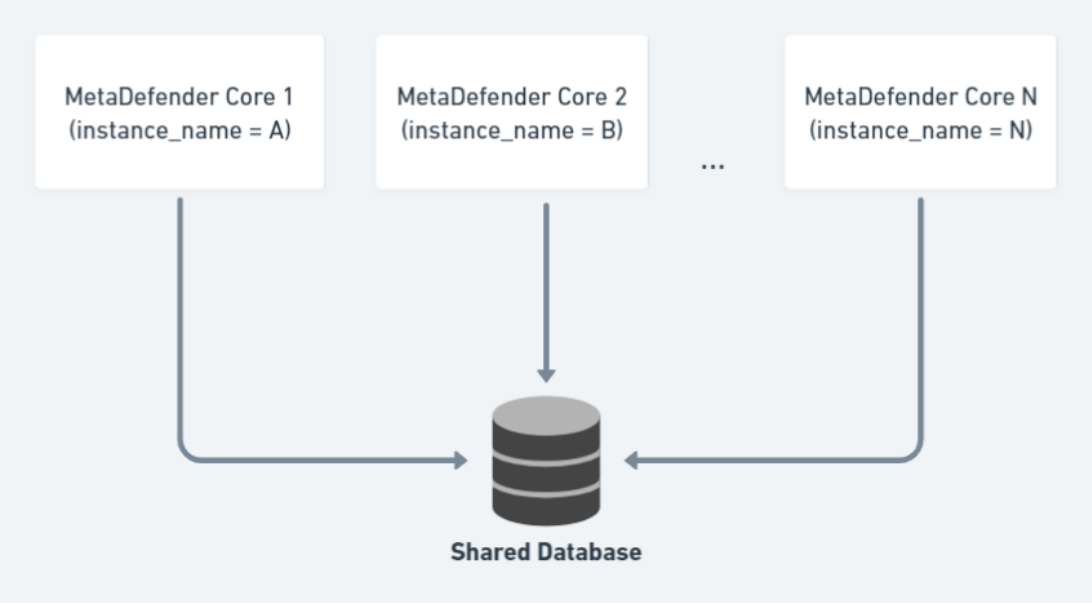
If reporting of engine issue count is enabled, MetaDefender Core server will send only the **number** of initialization errors and **number** of unexpected stops for the specific db/engine version. This information is sent over a HTTPS channel when the product downloads the latest package descriptors. This information is used for early detection of any specific 3rd party engine quality issues.

For details on using msiexec please consult Windows installer documentation.

Shared Database Mode

Starting with MetaDefender 4.20.0, you can use a centralized database deployment model, where all MetaDefender Core instances can share the same external database.

That helps auto-sync all product configurations (**except Single-Sign-On** that requires unique settings for each Core), processing history, live/on-going scan result query, audit, statistics data.



This article demonstrates how to install and setup MetaDefender Core 4.20.0 or newer with shared database mode.

Important notes & current limitations:

Currently not supported

- Does not support Central Management use-case with shared DB mode.
- Does not support to switch from shared database mode back to standalone database mode.
- Does not support to change instance_name of MetaDefender Core in shared database deployment model after installation.
- Does not support upgrading older MetaDefender Core from non-persistent mode to using the shared database mode.
- Does not support upgrading older MetaDefender Core from standalone mode to using the shared database mode.
- Does not support to upgrade multiple MetaDefender Core instances at once in shared database mode
    - Stop all Core instances' scanning service before performing the upgrade.
    - Upgrade each Core server one by one instead.

- For each Core, when the upgrade finished, then you can start using that Core for scanning without waiting for the other Core instances' upgrade to be done.

> **Disclaimer**
> MetaDefender Core does not support mode switching.

Requirements for shared database setup:

- The central remote PostgreSQL server (for shared database) could be utilized with a built-in (native) PostgreSQL database server installed with one of the MetaDefender Core instances in the network. Or using a dedicated existing PostgreSQL database server (see detailed requirements for PostgreSQL version at Recommended System Configuration).
- Before setting up, make sure to follow steps at Open Connection On PostgreSQL Server so that all MetaDefender Core instances could connect to the central remote PostgreSQL server.
- MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core, details: PostgreSQL User Privilege Requirements
- All MetaDefender Core instances must be installed on the same Operating System (Windows / Linux).
- The first MetaDefender Core instance connects to the shared database must be ready first (license activated, users/groups/roles created) before setting up other MetaDefender Core instances to connect and use that shared database.

Engine update:

- If engine updates is configured to be via Folder source, all MetaDefender Core instances must be able to access to the configured update folder, and the option DELETE FILES AFTER IMPORT will be unchecked and prevented for modification.

Authentication:

- Single-Sign-On (SSO) authentication settings (if using) cannot be shared among MetaDefender Core instances because SSO requires an unique login URL for every different Core.
- If an user is locked for login on a certain MetaDefender Core instance(e.g. exceeded the number of failed login times), that user is not prevented for logging in on any other instances.
- If an user is locked for sign-in on a certain MetaDefender Core instance (e.g. exceeded the number of failed login times), if the admin releases lockout on that user on that MetaDefender Core instance, that user is still not able to login on other MetaDefender Core instances.

Sanitized / DLP / Quarantined location changed after MetaDefender Core install:

Windows:

Shared storage path will be stored in 3 strings under key global in registry: quarantinepath, dlppath, sanitizepath as below

(Default)	REG_SZ	(value not set)
dbmode	REG_SZ	4
dlppath	REG_SZ	\\10.40.50.100\test
instance_name	REG_SZ	481rvj
quarantinepath	REG_SZ	\\10.40.50.100\test
restaddress	REG_SZ	*
restport	REG_SZ	8008
sanitizepath	REG_SZ	\\10.40.50.100\test

You can set either 1, 2 or 3 different paths for the 3 settings above (means that quarantined, DLP and sanitized files will be stored in same or different storages corresponding). In case shared storage is not set in installation, values of 3 settings above will be empty.

Restart MetaDefender Core service after changed.

Linux:

Open file `/etc/ometascan/ometascan.conf` and set shared storage path for 3 keys quarantinepath, dlppath, sanitizepath under section [global]

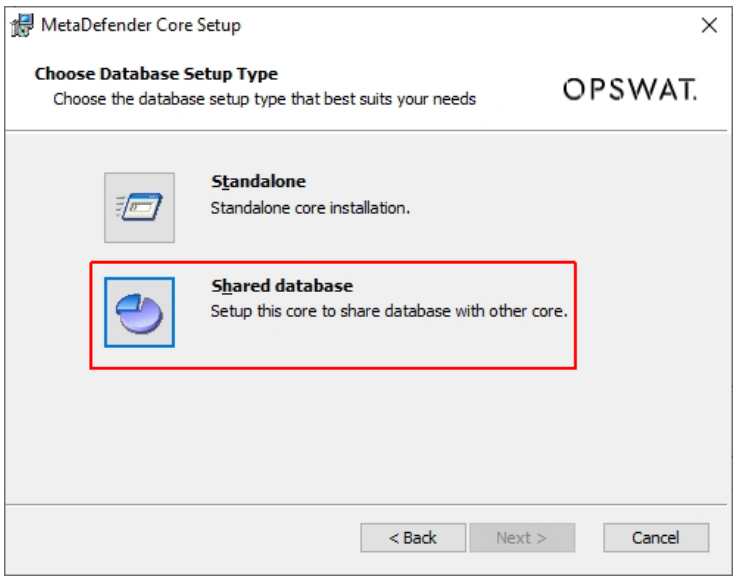
Restart ometascan service after changed.

Wizard Installation (Windows Only)

> **Info**
> Only applicable to MetaDefender Core 4.20.0 or newer using wizard on Windows OS.

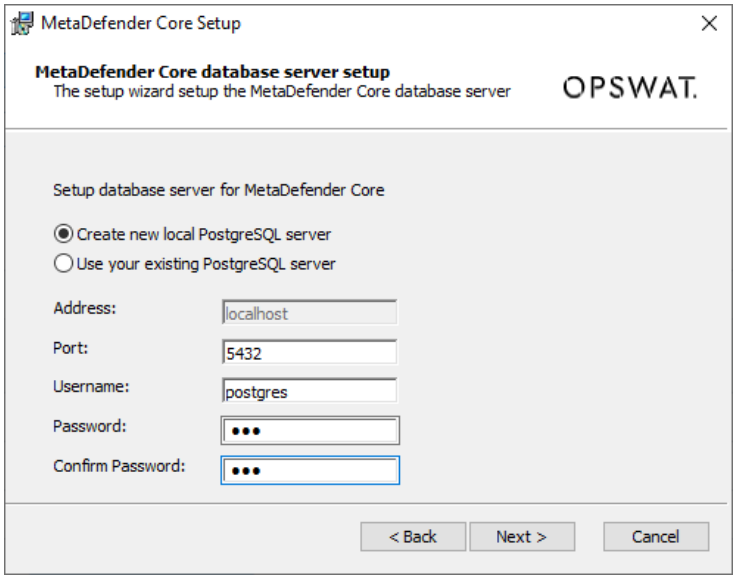
Clean install scenario

1. **Choose Shared database option:**

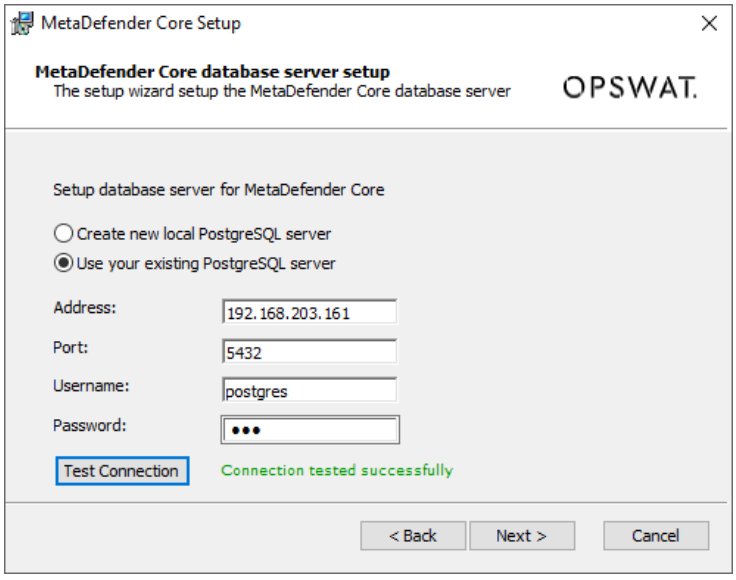


2. **Setup database server for MetaDefender Core:**

You can choose to create a native / built-in PostgreSQL server and setup the shared database on (then other MetaDefender Core instances will want to connect to that PostgreSQL server afterwards)



Or choose to connect to a pre-setup existing PostgreSQL server:



PostgreSQL connection configuration

"Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

PostgreSQL user privilege requirements

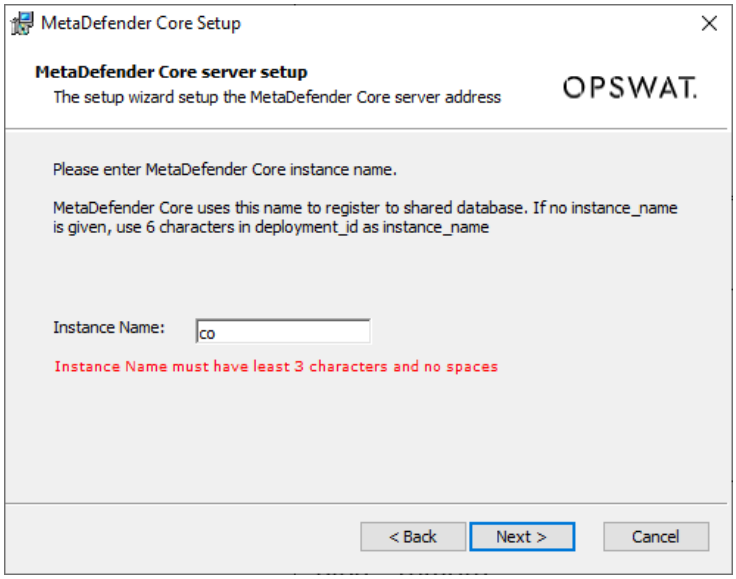
MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.

Details: PostgreSQL User Privilege Requirements

3. **Create an unique instance name for that MetaDefender Core (different from other MetaDefender Core instances):**

Instance name must contain at least 3 characters and no white space.

If you choose to leave it blank, which means it will use the default instance name with 6 characters extracted from that MetaDefender Core deployment ID.

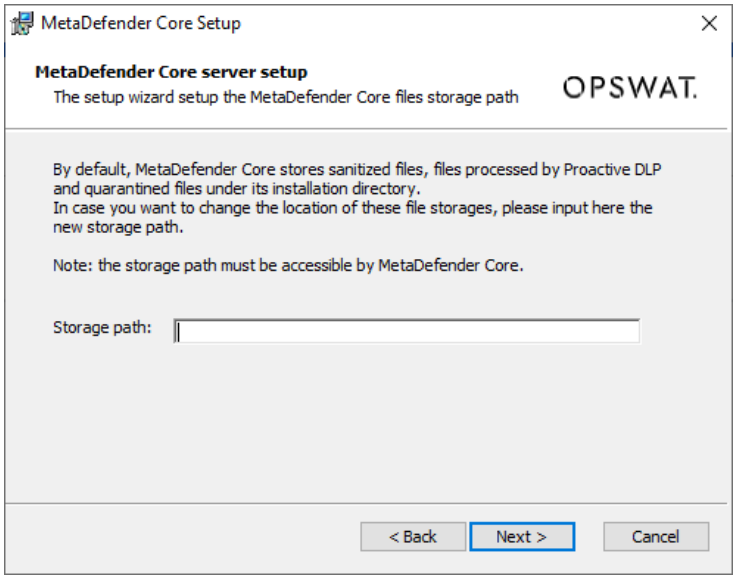


4. **Input the file storage path (sanitized, DLP, quarantined files)**

In the shared database model, all MetaDefender Core instances could use the same folder path to store all their own sanitized, DLP, quarantined files after processed.

The default location is: ***<MetaDefender Core installation folder>\data\ {sanitized|dlp|quarantine}***

If you wish to use that default location, just leave the input box blank and press Next.



Otherwise, you can specify a different location of sanitized/dlp/quarantined files. It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core.

For example:

- C:\new_folder
- \192.168.1.1\new_folder

Command Line Installation

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Clean install scenario

1. Create ignition file if not existed

- Windows: **C:\OPSWAT\ometascan.conf**
- Linux: ***/etc/opswat/ometascan.conf***

An example of ignition file:

ometascan.conf bash

```
eula=true

[global]
dbmode=4
instance_name=MetaDefenderCoreA
storage_path=D:\path\

[user]
name=admin
password=admin
email=admin@local

[dbserver]
type=remote
host=192.168.1.1
port=5432
user=postgres
password=your_own_password
```

Field	Required	Mapping to wizard UI	Possible values
global/dbmode	optional default is 1 (standalone)		1 - Standalone 2 - Central Hub deployment 3 - Non-persistent **4 - Shared**
global/instance_name	optional default is 6 characters of deployment ID		Unicode and number At least 3 characters No whitespace
global/storage_path	optional default is empty		Path to storages (DLP, quarantined, sanitized files) It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core. For example: • C:\folder • \192.168.1.1\folder

2. Run installers using command line to install

Please refer to Command Line Installation for installation command.

Deployment automation support

Note

An Ignition file could be utilized for either / both of following use-cases:

1. Pre-define PostgreSQL server information for MetaDefender Core to connect and setup database: only required on Linux installation, or Windows with command line installation. Learn more details at:
    a. Standalone Mode > Command Line Installation (applicable on version 4.19.0 or newer).
    b. Shared Database Mode > Command Line Installation (only applicable on version 4.20.0 or newer).
2. Automating to accept EULA, create a default local admin user and import MetaDefender Core configurations upon installed: Those steps could be automated using ignition file described in this page.

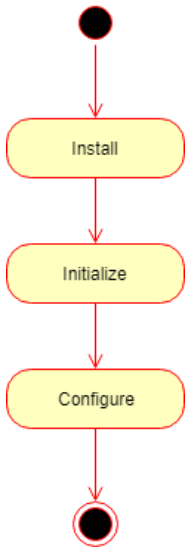
Ignition file location to create:

- **Windows**: `C:\OPSWAT\ometascan.conf`
- **Linux**: `/etc/opswat/ometascan.conf`

The product supports fully automated deployment. It means that it can be installed and configured with no human interaction.

The automated deployment can be split to three steps on a high level:

1. Installation
2. Initialization
3. Configuration



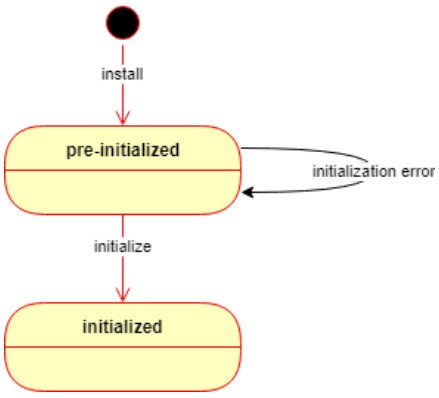
Installation

To automate the installation, install the product from the command line and provide the installation-time options as parameters to the installer. For further details see Command Line Installation

After the installation is complete, the product starts up and waits in a pre-initialized status. The product may be initialized in two ways:

1. Manually using the Wizard Setup or
2. Automatically using an *ignition file* (see below).

> **Warning**
> If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried after fixing the problem (e.g. placing the ignition file in its lookup location) and restarting the OPSWAT MetaDefender Core service.
>
> Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

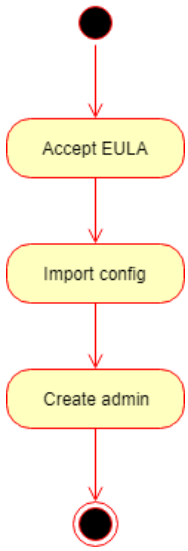


Initialization

Initialization is the process of bringing the product to an operable status.

Basically the initialization consists of the following steps:

1. Accept the End User License Agreement (EULA),
2. Import product configuration and
3. Create the first administrator user account.



Ignition file

The initialization process can be configured in a file called the *ignition file*.

> **Warning**
> In Linux, the ignition file and the folder /etc/opswat should have the permission 755

Ignition file location

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

The ignition file must be in conf format.

ometascan.conf bash

```
eula=true
[user]
name=admin
password=admin
email=admin@local
apikey=44c96cbb4e4e6c0a1d16bcb3c3da24def317
[config]
import=config_package.zip
import_password=123
[security]
session_cookie_samesite=Lax
[internal]
skip_startup_pages=true
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
private_username=my_internal_db_user
private_password=mypassword
[ocm]
ocm_url=https://my.opswat.com
reg_code=XXXXXXXX
```

Ignition file fields

The ignition file accepts the following fields:

Section	Key	Required	Accepted Values	Description
	eula	Optional (mandatory when having [user] section)	true false (default)	Whether to accept the End User License Agreement. This key must be set at the top of ignition file. This key must be set to "true" to accept the EULA. Any other value will cause the initialization to fail. When having [user] section defined, then this key must be set.
[user]		Optional		Initial administrator user account properties. The Administrator role is granted to the account.
	name	Optional	string	User name for the initial administrator user account.
	password	Optional	string	Password for the initial administrator user account. WARNING! Clear text password The password in this configuration file must be stored in its clear-text format and as so it may be visible for unauthorized parties.
	email	Optional	string	E-mail address for the initial administrator user account.
	apikey	Optional	string	API key for the initial administrator user account.
[config]		Optional		
	import	Optional	string	Path to a file in json/zip format that contains a previously exported configuration to be imported. Zip-format file contains a full configuration including user management settings, so it will overwrite [user] section.
	import_password	Optional	string	Password to decrypt .zip configuration package.
[security]		Optional		

Section	Key	Required	Accepted Values	Description
	session_cookie_samesite	Optional	Strict Lax (default) None	Declare if session cookie should be restricted to a first-party or same-site context.
[internal]		Optional		
	skip_startup_pages	Optional	true false (default)	Skip all welcome and other startup pages (e.g. upgrade result) when installing / upgrading MetaDefender, and going straight to the product index default page.
[dbserver]		Optional		
	type	Mandatory when having [dbserver] section	local remote	Where the PostgreSQL database server should physically locate. "remote" means PostgreSQL server and MetaDefender Core are not in the same machine.
	host	Mandatory when having [dbserver] section	string	IP address / domain name of the server where PostgreSQL server locates. "localhost" can be used when applicable.
	port	Mandatory when having [dbserver] section	number	Port of PostgreSQL server is listening for connections from clients (i.e. MetaDefender Core).
	user	Mandatory when having [dbserver] section	string	PostgreSQL server's user. SUPERUSER privilege is required for MetaDefender Core to setup its database and extensions for the first time. **Only non-Unicode characters supported.**

Section	Key	Required	Accepted Values	Description
	password	Mandatory when having [dbserver] section	string	PostgreSQL server's user credentials. **Only non-Unicode characters supported.**
	private_username	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
	private_password	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
[ocm]		Optional		
	ocm_url	Optional	string	URL to My OPSWAT server.
	reg_code	Optional	string	A registration code to enroll to My OPSWAT server.

Detailed initialization process

1) After the product has been started, it looks for the ignition file in the designated location.

Windows: C:\OPSWAT\ometascan.conf ** Linux: /etc/opswat/ometascan.conf

** 2) If an ignition file is found, then

2.1) It gets validated, and if it is valid, then based on the information found in the ignition file:

- The EULA is accepted,
- The configuration is imported,
- The administrator account is created.
- If any of the above steps fails, then the error is logged, and the initialization gets terminated.

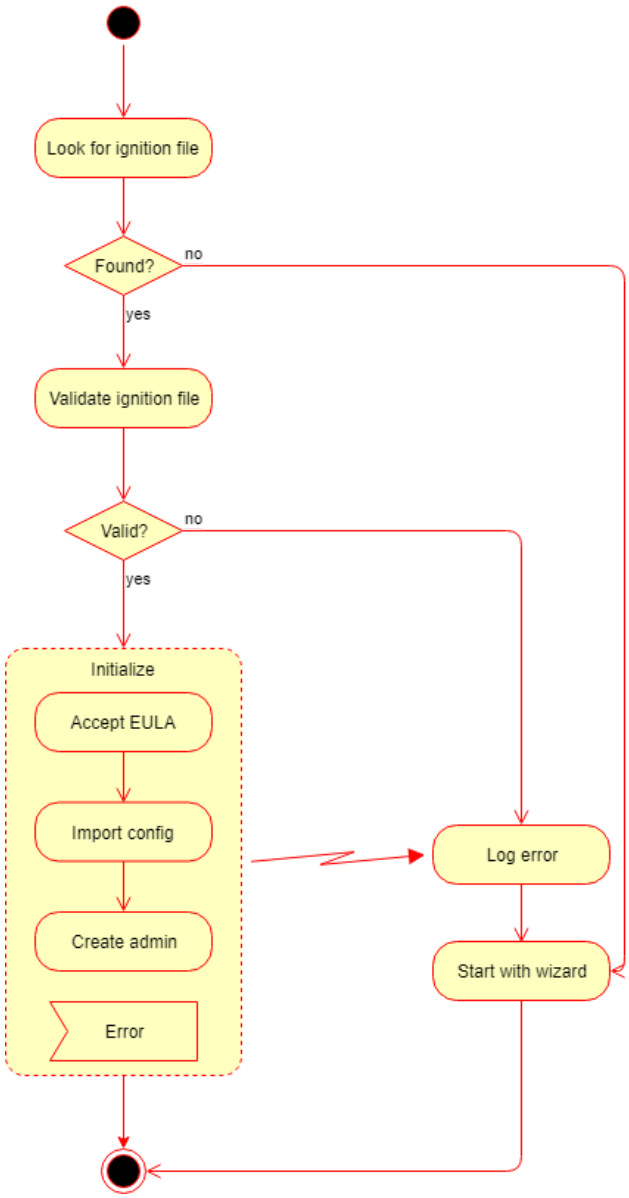
> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

2.2) If it is not valid, then the error is logged, and the initialization gets terminated.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

3) If there is no ignition file, then no initialization is performed.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.



Warning

If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried fixing the problem (e.g. placing the ignition file to its lookup location) and restarting the OPSWAT MetaDefender Core service.

Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

Configuration

After the initialization is complete, the product is ready with the default and the imported configuration.

This configuration can be later changed calling the configuration API functions. For further details about the API see Configuration related APIs API

CIS Level 1 Guidelines

Support Red Hat Enterprise Linux 9
For more details about Center for Internet Security (CIS) please refer to this document

Instruction steps

I. Install OpenSCAP

bash bash

```
yum install openscap-scanner scap-security-guide
```

II. Generate a result file and a HTML report using OpenSCAP scanner tool

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

III. Remediation of CIS Level 1 issues

Generate a remediation script based on the `ssg-rhel9-ds.xml` file:

bash

```
oscap xccdf generate fix --profile
xccdf_org.ssgproject.content_profile_cis_server_l1 --fix-type bash --output
remediations.sh /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

And execute remediation script.

bash

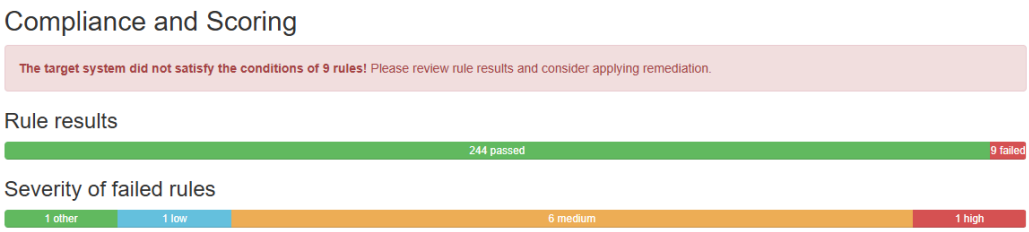
```
./remediations.sh
```

IV. Review the results after remediation

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

Example:



Notes

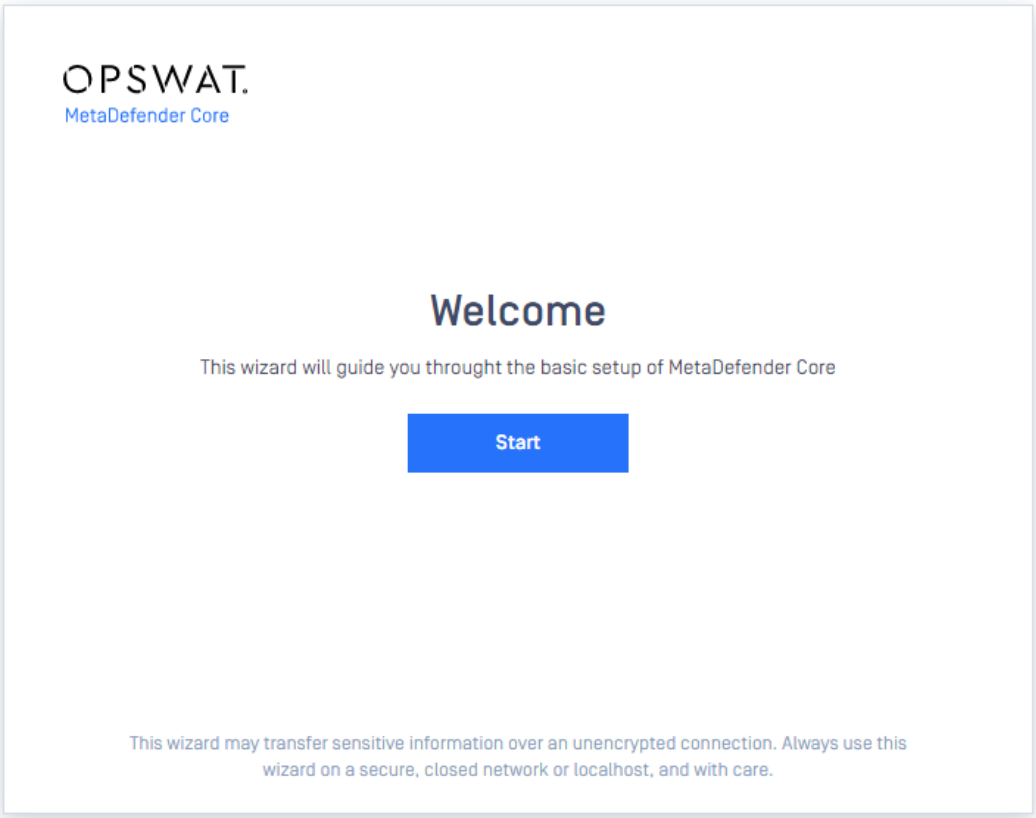
- CIS Level 1 requires `/tmp` folder to be mounted in a separate partition. Please ensure that that new partition have enough disk space for MetaDefender Core to run.
- CIS Level 1 requires that "Ensure No World-Writable Files Exist".
    - For now, when freshly installing MetaDefender Core, all its binary files meet the requirement.
    - When installing/updating engines, some engines might create additional files for its operation, and it might violate this requirement. In this case, you need to again execute the remediation script in the step III.

Wizard Setup

Introduction

Access the MetaDefender Core management console for the first time (i.e. http://localhost:8008), you are to complete a quick walkthrough for wizard setup in order to be able to use the product.

The administrative management console will be available only upon you have successfully finished this wizard stage.

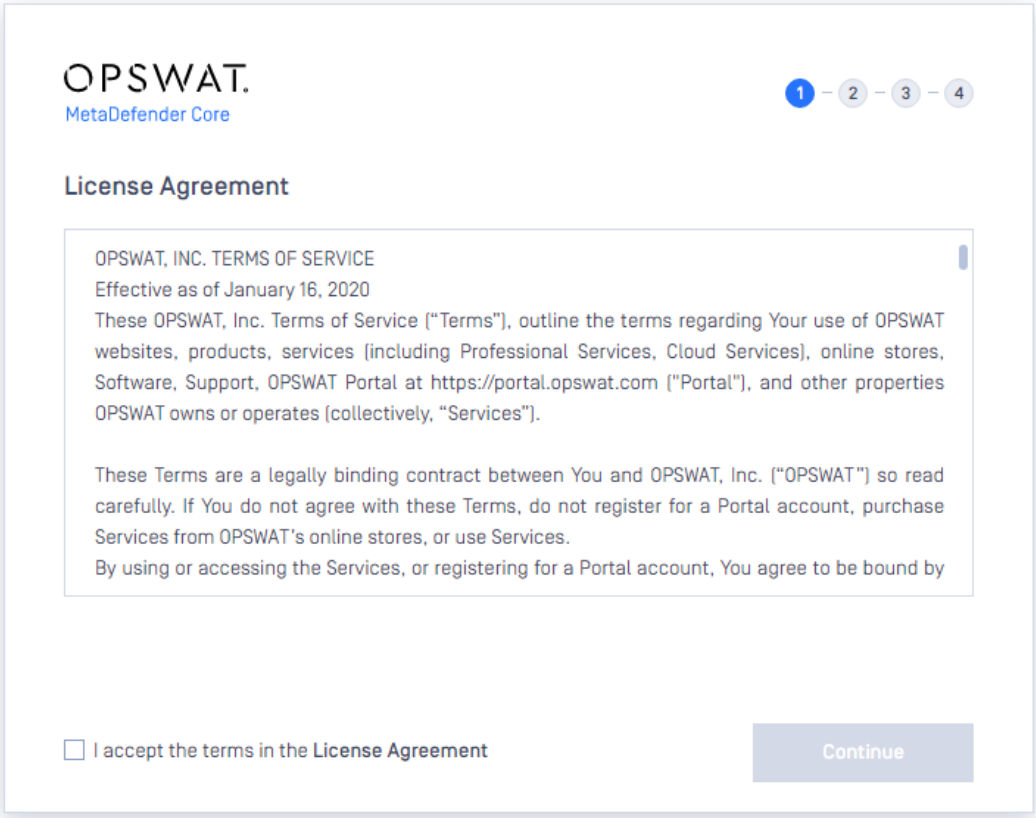


Warning

This wizard may transfer sensitive information over an unencrypted connection. Always use this wizard on a secure, closed network or localhost, and with care.

Basic configuration steps

End-User License Agreement



In the first page you can find the End-User License Agreement. You have to accept the terms before moving on.

Please read through the EULA (https://docs.opswat.com/policies/privacy-policy/portal-privacy-policy) carefully and if you agree with it, check `I accept the terms in the **License Agreement**` and click `Continue` to proceed.

Local Admin User Setup

The next step is to set up a default local administrator account.

This account will be the first one being able to access the administrative management console, and to create accounts for other users. You have to fill out all fields in this page to be able to move forward. When you are done, click NEXT to continue.

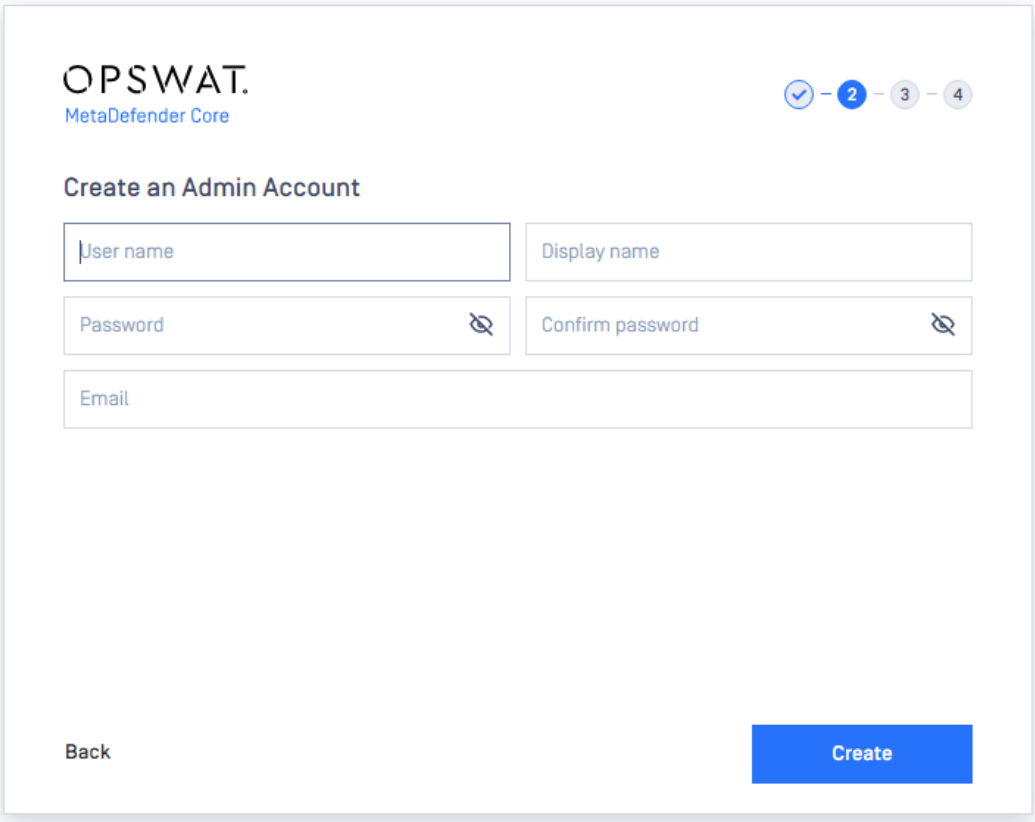
Info - User Directory

The administrator account, that is created via the basic configuration wizard, is always added to the LOCAL user directory as a member.

Learn more about user directory: User directories

The following information is required for the administrator account:

ACCOUNT NAME	The unique name of the account that is used at the time of login and in log messages for accountability.
ACCOUNT DISPLAY NAME	Name of the person bound to this account. This name (appended to the name of the account's user directory) is displayed in the top right corner of the administrative management console.
PASSWORD	Password of the user bound to this account that is used at the time of login. Passwords sent clear-text As long as TLS is not configured for the basic configuration wizard, passwords are sent clear-text over the network and may be disclosed to unauthorized parties. As a mitigation action: 1. Either use the wizard on *localhost* or on a direct network connection, or 2. Enabling HTTPS as soon as possible and change the password immediately if it has already been set.
EMAIL	Email address of the person bound to this account.



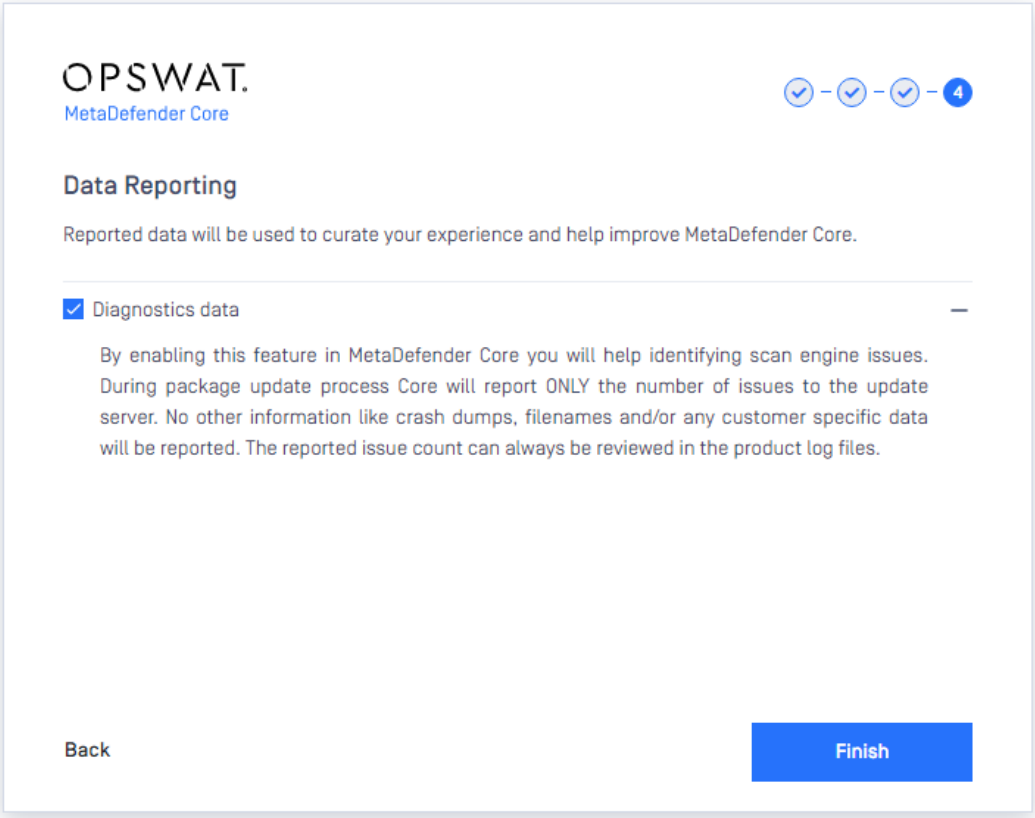
License activation

For license activation details see License Activation

Data Reporting

Enabling this option to allow MetaDefender Core collect its own engine issues and report back to OPSWAT for further analysis.

MetaDefender will not collect any other information.

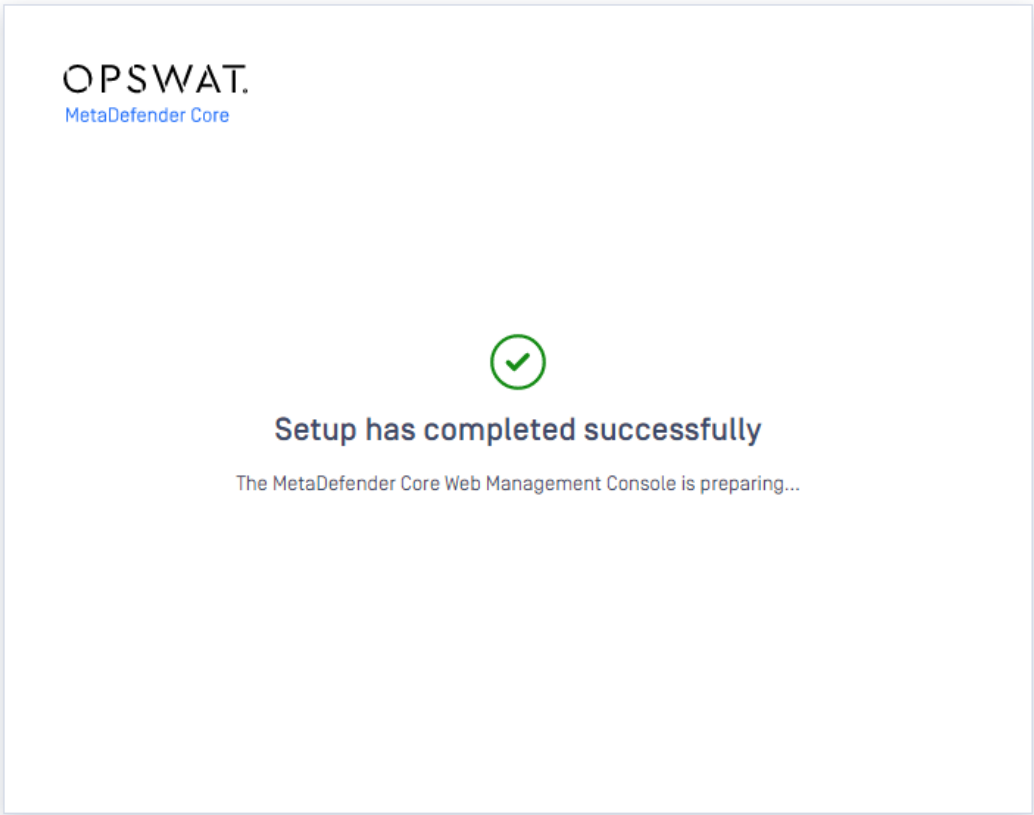


Wizard completion

After you have completed every steps you are ready to finish the wizard and start using the product. Click the `Fishish` button to complete the wizard.

> **Warning**
> The product's service will be restarted and the browser will be redirected to the administrative management console. This could take several seconds.

You can login to the Web Management Console with the administrator user that have just been created in the previous steps.



Setup HTTPS

Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network. Websites, like the Web Management Console, are able to use TLS to secure all communications between their servers and web browsers.

The TLS protocol aims primarily to provide confidentiality (privacy) and data integrity between two communicating computer applications.

Warning

No TLS for the wizard

By default, TLS is not enabled for the basic configuration wizard. As a consequence sessions between the wizard's backend and the browser may be insecure.

Info

Performing the same steps as for the Web Management Console, it is possible to set up TLS for the basic configuration wizard. Remember completing the TLS setup before launching the wizard.

For instructions to set up TLS see Enabling HTTPS

License Activation

##

To activate your installation go to the **Settings > License** menu in the Web Management Console. If you have no valid license, you will only see your installation's **deployment ID**. You will also see a warning in the Web Management Console header.

After successful activation, the product will start downloading the latest available scan engines and malware databases. You can follow the status of the scan engine installation on the **Inventory > Modules** page.

When all the licensed engines are installed, you can start using the MetaDefender Core to process files - File Processing.

Deployment ID

Each MetaDefender Core instance represented by an unique system fingerprint called `Deployment ID`

MetaDefender Core licensing is purely based on deployment ID activated. Therefore, everytime your MetaDefender Core is activated with a different deployment ID, then it will be captured as a different activation record accumulated into your total activation times allowed.

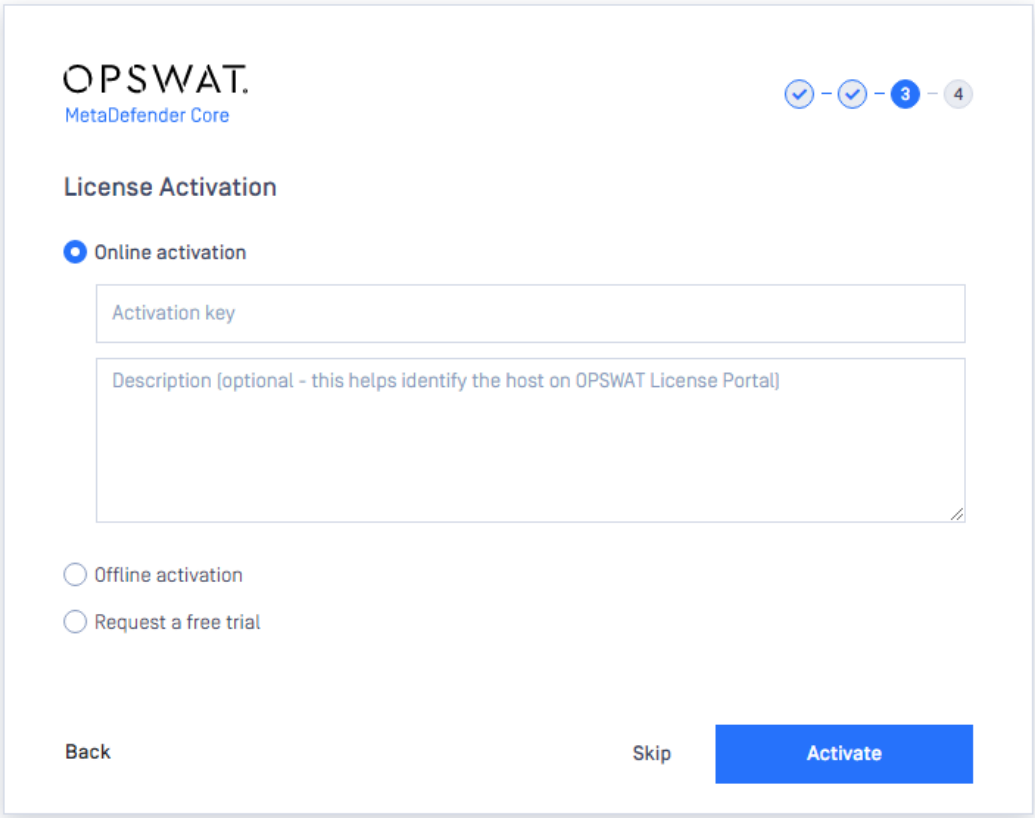
> **Notice**
> Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you alter the system where MetaDefender Core relies on (e.g. network card, OS hostname).
>
> Deployment ID is an UUID version 4 random based string. Learn more at: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you update the system where MetaDefender Core relies on.

Activation Modes

There are three different options for the product activation:

- Online activation: MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.
- Offline activation: Requiring you to upload a license file (.yml). That license file could be retrieved by activating your license via OPSWAT Portal. Follow the displayed instructions for details.
- Request trial key online: For evaluation purpose, you can receive an trial Activation key via email. Follow the displayed instructions for details.



Online activation

MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.

Required to input your Activation License key, and optional for description (but we recommend this for troubleshooting and auditing purpose if needed).

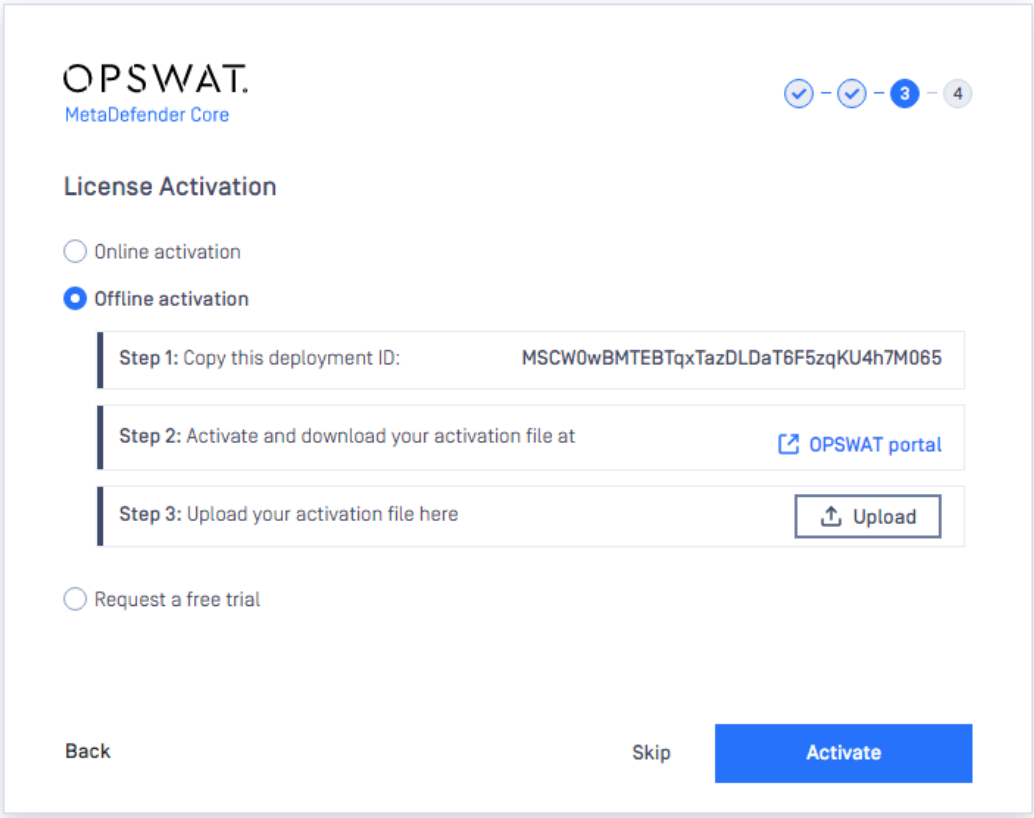
Warning

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts' for license activation:

- https://activation.dl.opswat.com Note: *IP address-based whitelisting on your firewall might fail after some time since OPSWAT uses CDN (Content Delivery Network) to faster delivery updates over the world, and IP address of edge servers might change over time.*

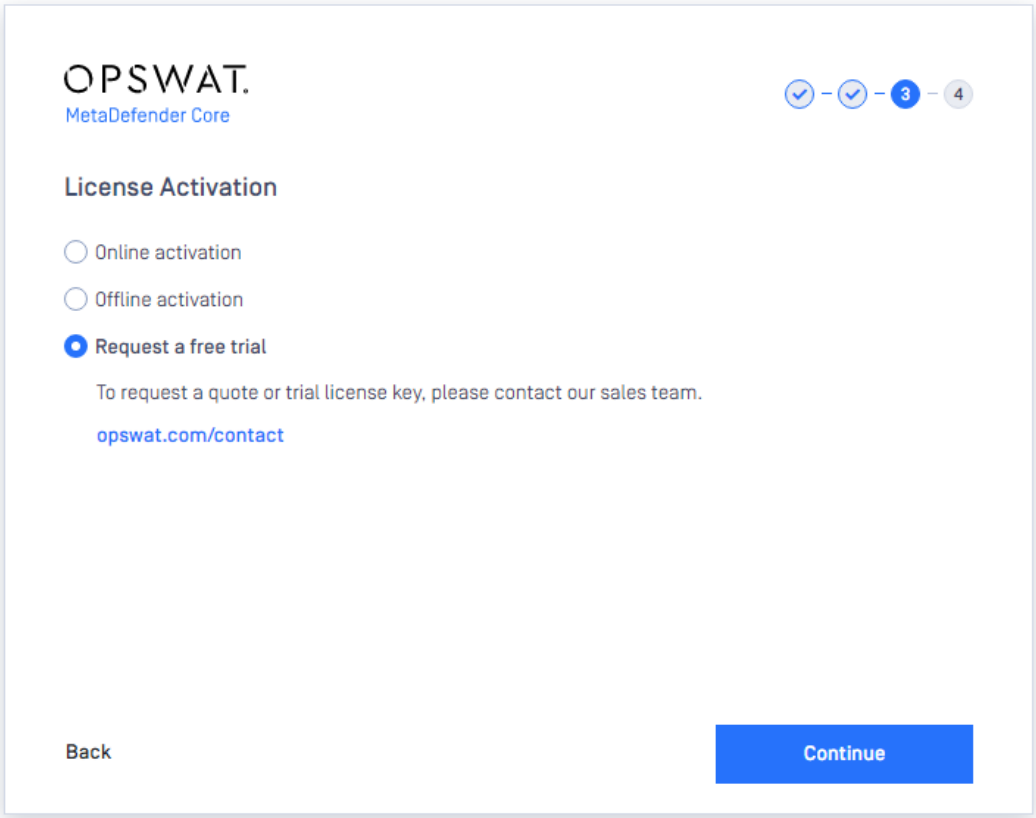
Offline activation

With no internet connection on the server the Metadefender v4 instance may be activated indirectly from a different machine, that has internet connection. The Deployment ID of the Metadefender v4 instance and the the Activation key received at the time of purchasing the product will be required. Follow the steps on the screen to activate the product offline.



Request trial key online

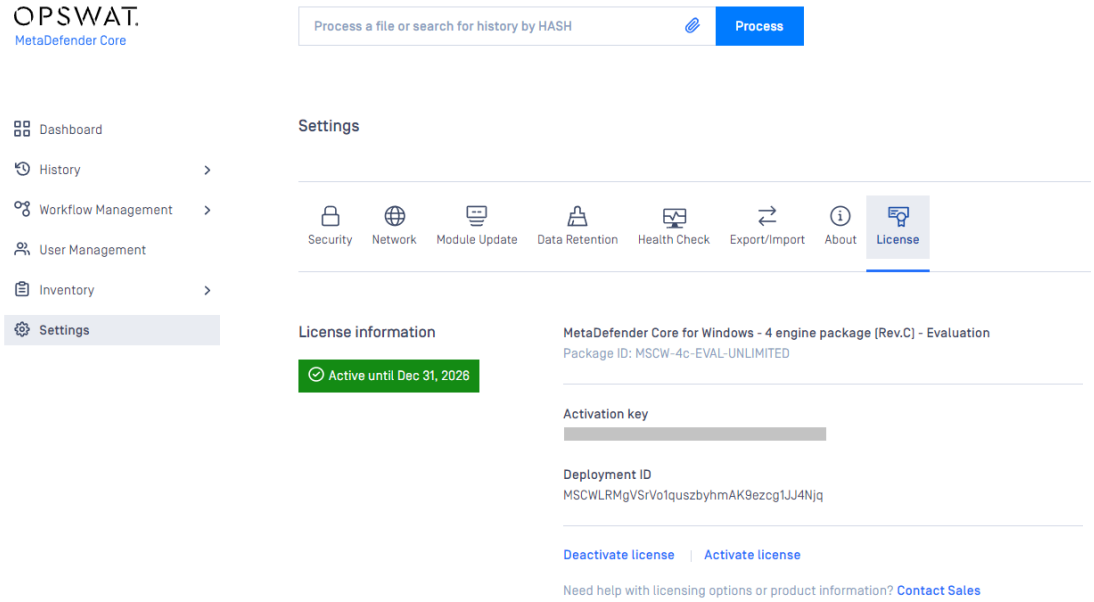
An evaluation license could be acquired by contacting our sales team.



Check Your License Details

For more license details and activating your MetaDefender Core, go to Settings > License menu tab on the Web Management Console:

- Product name: product name as on your order (e.g. MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation)
- Package ID: product identification as on your order (e.g. MSCW-4c-EVAL-UNLIMITED)
- Expiration: last day of license validity
- Activation key: key used for activating product
- Deployment ID: identification of this installation


OPSWAT.
MetaDefender Core
Process a file or search for history by HASH
Process
Dashboard
History
Workflow Management
User Management
Inventory
Settings
Settings
Security
Network
Module Update
Data Retention
Health Check
Export/Import
About
License
License information
Active until Dec 31, 2026
MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation
Package ID: MSCW-4c-EVAL-UNLIMITED
Activation key
Deployment ID
MSCWLRMgVSrVo1quszbyhmAK9ezcg1JJ4Njq
Deactivate license
Activate license
Need help with licensing options or product information? Contact Sales

Scripted license management

Using REST API calls there is an option for scripted activation and deactivation for MetaDefender Core servers with **no Internet connection or if your infrastructure is using automation to create/destroy MetaDefender Core instances**. In this chapter the steps of these two scenarios are described.

Requirements

- an installed MetaDefender Core instance without Internet connection
- another computer that has Internet connection and can run your activation scripts
- a manual or automated way to transfer data between the two computers

Activation steps

1. In order to activate MetaDefender Core, deployment ID and activation key are needed to generate the license file.
    a. Activation key should be purchased from OPSWAT.
    b. MetaDefender Core deployment ID can be queried by using REST API (for details see GET Get Current License Information API page)

> **Info**
> Save this deployment ID in your system, you might need this ID when the instance is unavailable at the time of deactivation.

2. Activate license and get the license file using the following URL, where <quantity> is 1.

activation call bash

```
curl -X GET https://activation.dl.opswat.com/activation?key=<activation-key>&deployment=<deployment unique-ID>&quantity=<quantity>
```

If the activation is successfully, a license file is downloaded. Save this file.

HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	license file	Activation was successfully.
200 Activation failed	error: '<user conform error message>'	Failed activation
200 Invalid parameter	error: 'Could not activate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not activate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
200 Invalid parameter	error: 'Could not activate your product because the quantity you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key, quantity or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Info
In case of any activation issue, contact OPSWAT support for help

3. Upload license file to MetaDefender Core: The license file should be uploaded to the MetaDefender Core v4 to activate the product. For details see page: POST Upload license key API

Deactivation steps

1. For deactivation, following infomation is required:
    a. Activation key
    b. MetaDefender Core deployment ID: can be queried by using REST API (for details see GET Get Current License Information API page)
2. Deactivate license using the following URL:

deactivation call bash

```
curl -X GET https://activation.dl.opswat.com/deactivation?key=<activation-
key>&deployment=<deployment unique-ID>
```

Response HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	result: ok	Successful deactivation
200 No active license found	error: 'Could not found any active license with the given parameters'	The license has not been activated yet or it has been deactivated already.
200 Invalid parameter	error: 'Could not deactivate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not deactivate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Important notes

Warning

Product activation is tied to several hardware, operating system and software parameters. In case of one or more major hardware or software parameter change the product might turn into deactivated status. Operating system updates, other software updates should not affecting the activation status.

If the product is online activated and has live Internet connection then in case of deployment ID change the product reactivates itself. If the product doesn't have Internet connection then the administrator is responsible to reactivate the product.

Because of the above administrator should consider licensing restrictions of the offline deployments if using any virtualization and/or containerization technologies.

File Processing

MetaDefender Core provides REST API interface for file processing business.

There are several ways to process files with MetaDefender Core:

- Analyze Files via Web Interface
- Process Files via REST API

Product Upgrading

> **Info**
> Simply run installer file, details could be found at Installation

> **Info**
> For command line mode, upgrading (e.g. on **Linux**), prior to running the installer file, please make sure to **create the ignition file** with PostgreSQL server information included. Details:
>
> - Standalone DB mode: Command Line Installation (Windows and Linux)
> - Shared DB mode: Command Line Installation

Recommendation for MetaDefender Core upgrade

Our goal is to help you execute the product upgrade continuously, efficiently, and correctly.

PostgreSQL service upgrade

- **Bundled PostgreSQL Service**: If you are using the bundled PostgreSQL server included in the installer, the PostgreSQL service will be automatically upgraded to the newer version (if available). No additional action is required for this process.
- **Remote PostgreSQL Service**: For remote PostgreSQL servers, customers are responsible for managing and upgrading the service independently. Upgrading MetaDefender Core does not update the remote PostgreSQL service.

Before the upgrade

Prepare a proper disaster recovery plan (DRP) including some common practices:

- Backup, or take a snapshot of your current working MetaDefender Core server.
  - This is to help quickly rollback for any upgrade failure might cause later for any reason.
- Optionally backup your PostgreSQL database, especially when using remote shared database mode.
  - Reference: https://www.postgresql.org/docs/current/backup.html
  - Please do it after stopping all linked MetaDefender Core instances to avoid potential data collision.
- Lastly, if backing up entire server is not feasible, then at least export MetaDefender Core configuration package Import/Export configuration just in case you'll need to start off a new server with current MetaDefender Core configuration to keep the scanning service running without any impact.

On the PostgreSQL server, double check server resource to ensure sufficient RAM memory and disk space to be preserved for the upgrade process.

- 8GB RAM available at least during the upgrade.

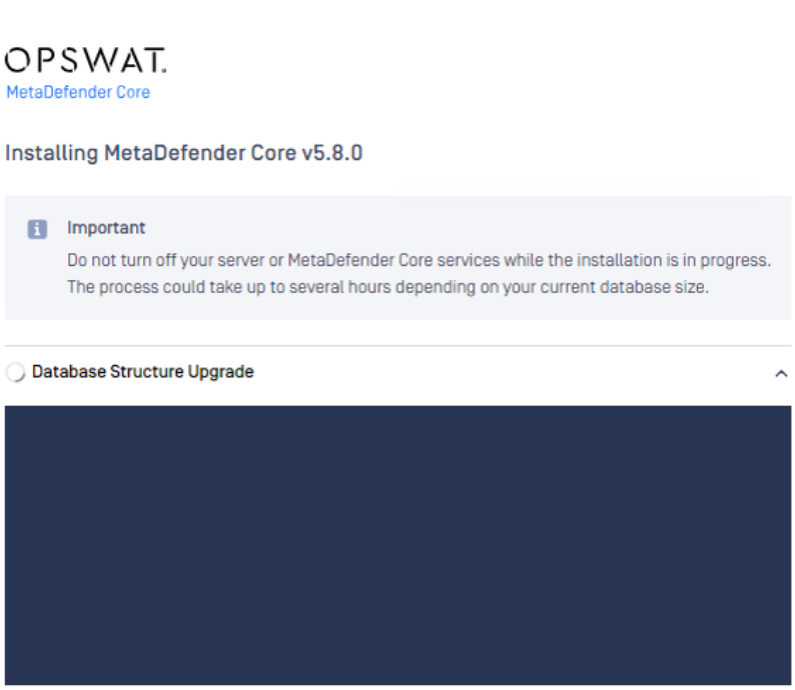
- Disk availability is greater than MetaDefender Core database size.

While doing the upgrade

Applicable for shared database deployment:

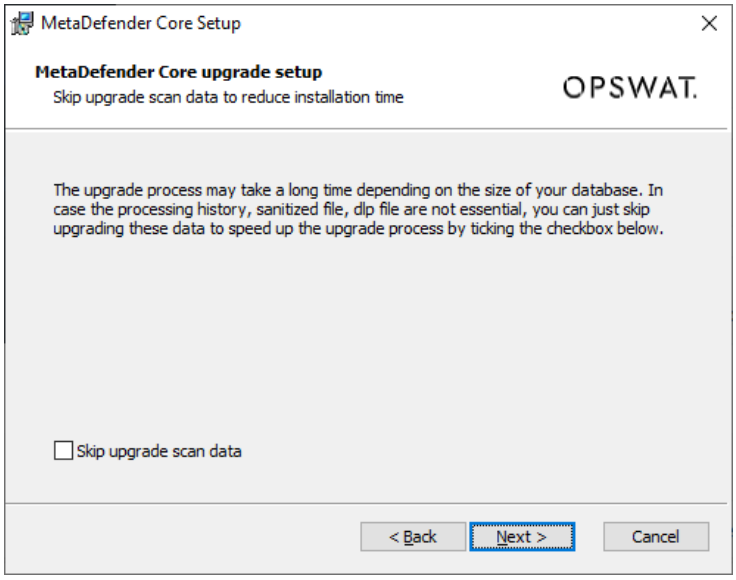
- Stop all MetaDefender Core instances' scanning service before performing the upgrade.
- Upgrade one (1) MetaDefender Core instance at a time.
- Proceed to upgrade MetaDefender Core instance where the bundle local PostgreSQL server is used for shared database setup.
- Otherwise is using remote PostgreSQL then the upgrade order does not matter.

For large size database, mostly for scan history, expecting long time to be spent on the upgrade process.



The product supports an option to skip data migration for scan history to expedite the process drastically and mitigate any failure risk. You can manually backup all data from your PostgreSQL server for offline usage if any.

Check "Skip upgrade scan data" to enable this feature while doing the upgrade.



or add following key in the ignition file:

none

```
[internal]
skip_upgrade_scan_data = 0
```

Troubleshooting

If the upgrade process ever runs into an error:

- Double check your network and services, applicable to remote PostgreSQL server.
- Optionally configure MetaDefender Core to skip migrating scan history.
  - Windows
    - Open Windows registry, and navigate to HKLM\Software\OPSWAT\Metascan
    - Add a new key `internal/skip_upgrade_scan_data` with value = 1
  - Linux
    - Open MetaDefender Core configuration file `/etc/ometascan/ometascan.conf`
    - Add new key into the file
      - [internal]
      - skip_upgrade_scan_data=1
  - Note: This key will be auto removed upon the upgrade finishes.
  - Then restart MetaDefender Core service to retry the upgrade.
    - Might retry this a few times until the upgrade is done.

Just in case the upgrade process still fails, please contact OPSWAT support with the support package - How to Create Support Package? This will help us analyze the root cause and figure out an optimal solution for your system failure.

Using external Load Balancer

Application Load Balancing (Layer 7)

This page provides information about recommended ways to use sessions with a Layer 7 load-balancer.

Most load-balancers have the ability to provide cookies for pinning subsequent traffic from a client to the appropriate server. This method is called sticky session, session persistence or session affinity. Using cookies efficiently requires the client to know when a cookie should be sent, should not be sent or should be deleted.

Sticky session load balancing

Each Metadefender Core instance has its own databases and application sessions that cannot be seen by other instances. Therefore, in order to get the related data/response to our queries we should ask the appropriate Metadefender Core v5 server. On the other hand, to keep the advantages of the used load balancing method, cookies should not be sent if it is not necessary.

Single file scanning

Step	Stage	Task	Cookie usage
1.	Sending file	Initiate processing a file on the client side. Send file through the load-balancer. (See POST File Analysis API)	Cookie should not be sent
2.	Sending file	Save the cookie and the data_id you got from the load-balancer	Save cookie
3.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Send cookie saved in step 2
4.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Send cookie saved in step 2

Batch scanning

Step	Stage	Task	Cookie usage
1.	Open batch	Initiate processing file(s) in batch. Request a batch ID through the load-balancer. (See POST Initiate Batch API)	Cookie should not be sent
2.	Open batch	Save the cookie and the batch_id you got from a Core server through the load-balancer.	Save cookie
3.	Sending files	Send file through the load-balancer. (See POST File Analysis (using batch header) API)	Send cookie saved in step 2
4.	Sending files	Save the data_id you got from the load-balancer.	-
	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Send cookie saved in step 2
5.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
6.	Getting batch status	See GET Batch Status API	Send cookie saved in step 2
7.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Send cookie saved in step 2
8.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API	Send cookie saved in step 2

If it does not matter which upstream server responds, the queries should be sent without cookie.

> **Info**
> It is recommended not to send cookies when it's not necessary to allow load-balancer to use its own method to share the load between Metadefender Core v5 servers.

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration through load-balancer

OPSWAT products that support HTTP load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	4.3.4	-
OPSWAT Client	Windows: 7.6.247.0 Mac: 10.4.243.0	-
MetaDefender Email Security	4.3.0	-
MetaDefender ICAP Server	4.3.0	-
MetaDefender Vault	1.3.0	-
MetaDefender for Secure Storage	2.0.7	-

DNS Load Balancing

Using this method is logically similar to Layer 7 load-balancing.

Briefly how it works

Client uses a domain name to send a query to a server. Client's DNS server has more "A" records for that name with different IPs. When a client resolves the server's domain name DNS server randomly chooses an IP for that name to send back. When a session is used on the application layer, client should know the IP address of the Core server that handles that specific session. In every other case, client should resolve the domain name with DNS query to let requests to be balanced between Metadefender Core servers.

Single file scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Sending file	Initiate processing a file on the client side. (See POST File Analysis API)	Use IP saved in step 2
4.	Sending file	Save the data_id got from the Core server	-
5.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Use IP saved in step 2
6.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Use IP saved in step 2

Batch scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Open batch	Initiate processing file(s) in batch. Request a batch ID. (See POST Initiate Batch API)	Use IP saved in step 2
4.	Sending files	Send file to the specific Core server with the basch ID saved in step 3. (See POST File Analysis (using batch header) API)	Use IP saved in step 2
5.	Sending files	Save the data_id you got from the Core server.	-
6.	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Use IP saved in step 2
7.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
8.	Getting batch status	See GET Batch Status API	Use IP saved in step 2
9.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Use IP saved in step 2
10.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API).	Use IP saved in step 2

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration via DNS load-balancing

OPSWAT products that support DNS load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	does not support yet	-
MetaDefender Client	does not support yet	-
MetaDefender Email Security	does not support yet	-
MetaDefender ICAP Server	does not support yet	-
MetaDefender Vault	does not support yet	-

SSL connection for PostgreSQL communication

> **Note**
> The SSL connection configuration is possible thanks to the support made by PostgreSQL. For reference: https://www.postgresql.org/docs/14/ssl-tcp.html
>
> MetaDefender Core is tested against this mode on particular version 4.21.1

Step 1: Generate SSL certificate for PostgreSQL server

> **Note**
> Following instructions are for self-sign certificate. For production environment, making sure to use the certificate signed by your trusted CA.

1) Generate a private key

bash

```
openssl genrsa -des3 -out server.key 1024
```

Sample output:

bash

```
Generating RSA private key, 1024 bit long modulus ........++++++
...............++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

Type your desired password / pass phrase to encrypt the private key

2) Remove the pass phrase to automatically start up the PostgreSQL server

bash

```
openssl rsa -in server.key -out server.key
```

> **Note**
> On Linux, make sure to set permission on the server.key file

bash

```
chmod og-rwx server.key
```

3) Create a self-signed certificate

bash

```
openssl req -new -key server.key -days 3650 -out server.crt -x509
```

You will be prompted to enter detailed information which is incorporated into your self-signed certificate request.

none

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:CA
Locality Name (eg, city) [Default City]:San Francisco
Organization Name (eg, company) [Default Company Ltd]:OPSWAT
Unit Name (eg, section) []:OPSWAT
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:dummy_email@opswat.com
```

For self-signed certificate, use the server ceriticate as the trusted root certificate:

bash

```
cp server.crt root.crt
```

Copy **server.key**, **server.crt**, **root.crt** to PostgreSQL data folder. For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data**
- Linux: **/var/lib/ometascan/pg_data/**

Step 2: Configure PostgreSQL server for SSL authentication connection

1) Create a custom config file (e.g. ssl.conf) for PostgreSQL.

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\postgres\ssl.conf**
- Linux: **/var/lib/ometascan/postgres/ssl.conf**

2) Content of ssl.conf:

none

```
ssl = on
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
```

3) Modify pg_hba.conf

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data\pg_hba.conf**
- Linux: **/var/lib/ometascan/pg_data/pg_hba.conf**

Add following (modify **::1/128** if needed to match with your environment requirement)

none

```
hostssl all all      ::1/128      scram-sha-256
```

After the modify, the content of pg_hba.conf should be like this (example):

> **Note**
> In a Linux environment, a local connection is required to upgrade from version 5.5.1 or earlier to version 5.6.0 or later.

none

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local         DATABASE  USER  METHOD  [OPTIONS]
# host          DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl     DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostgssenc    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnogssenc  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
local    all all                    scram-sha-256
hostssl all all      ::1/128      scram-sha-256

#More content.......
```

4) Reload the services

- Standalone DB mode: Restart **ometascan** service (MetaDefender Core service)
- Shared DB mode: Restart **ometascan-pg** / **ometascan-postgresql** service (MetaDefender Core PostgreSQL service) and then **ometascan** service (MetaDefender Core service)

Test your SSL database connection:

bash

```
psql.exe "sslmode=require dbname=postgres user=postgres host=localhost port=5432"
Password for user postgres:
psql (12.7)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
Type "help" for help.
```

This message indicate that you are now using SSL connection:

bash

```
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
```

Performance and Load Estimation

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Some factors that affect performance

- MetaDefender Core version
- MetaDefender Core engine package and configuration
  - set of engines (which and how many)
  - product configuration (e.g., thread pool size)
- System environment
  - server profile (CPU, RAM, hard disk)
  - client application location - remote or local
  - system caching and engine level caching
- Dataset
  - encrypted or decrypted
  - file types
    - different file types (e.g., document, image, executable)
    - archive file or compound document format files
  - file size
  - bad or unknown (assume to be clean)
- Performance tool

How test results are calculated

Performance (mainly scanning speed) is measured by throughput rather than unit speed. For example, if it takes 10 seconds to process 1 file, and it also takes 10 seconds to process 10 files, then performance is quantified as 1 second per file, rather than 10 seconds.

- total time / total number of files processed: 10 seconds / 10 files = 1 second / file.

Standalone DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.2.416.0

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

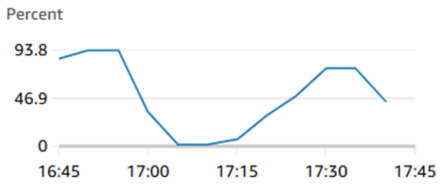
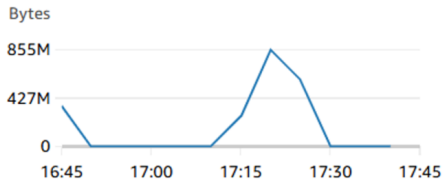
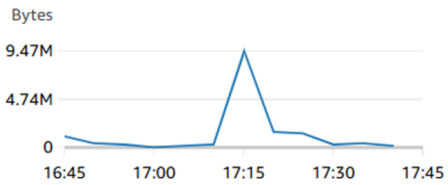
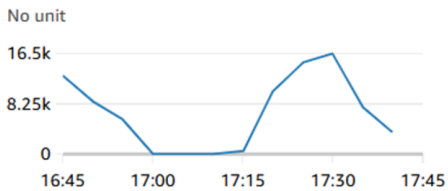
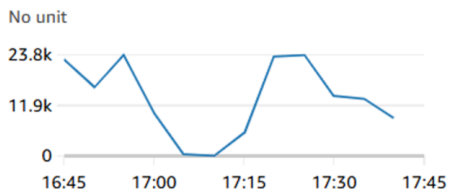
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV)	24.2	47.2 / 80	37.3 / 41.1		315,310	0.0114
Deep CDR	19.9	48.9 / 81.8	32.6 / 37		383,442	0.0094
Proactive DLP	17.1	36.8 / 58.2	30.4 / 33.6		446,228	0.0081
Vulnerability	10.1	59.2 / 85	29.3 / 33.3		755,495	0.0048

MetaDefender Core with common engine packages:

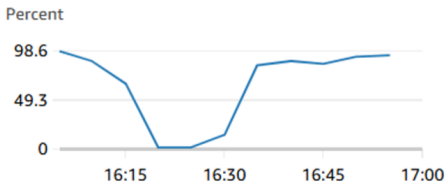
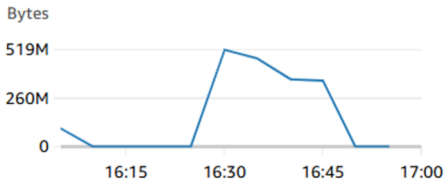
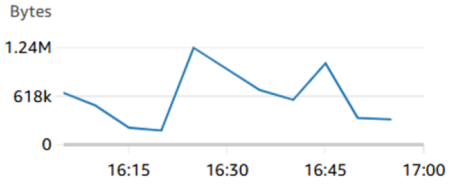
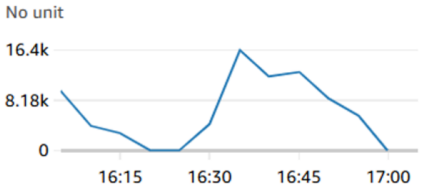
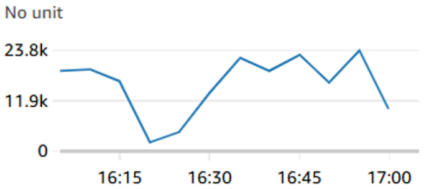
Use case	Scan duration	Avg./Max CPU usage	Avg./Max RAM usage	Avg./Max Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV) + Deep CDR	30.8	68.1 / 88	40.1 / 46.8		247,744	0.0145
Metascan (8 AV) + Deep CDR + Proactive DLP	31.2	74.2 / 98.6	41.5 / 51.1		244,567.3	0.0147
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	31.8	73.5 / 98	40.6 / 51.5		239,953	0.015

System resource utilization charts

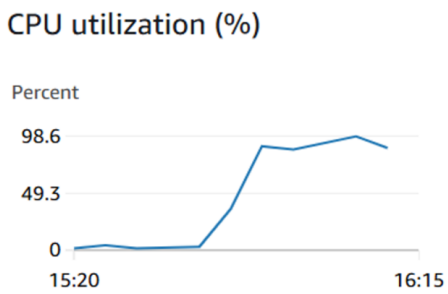
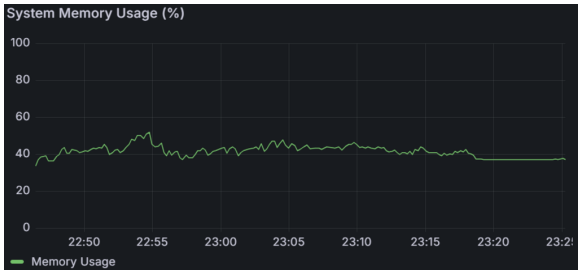
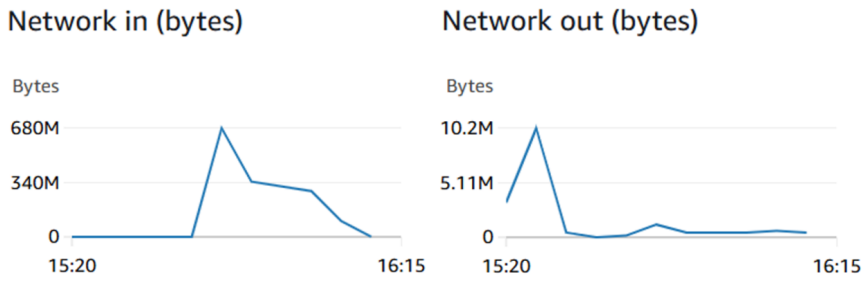
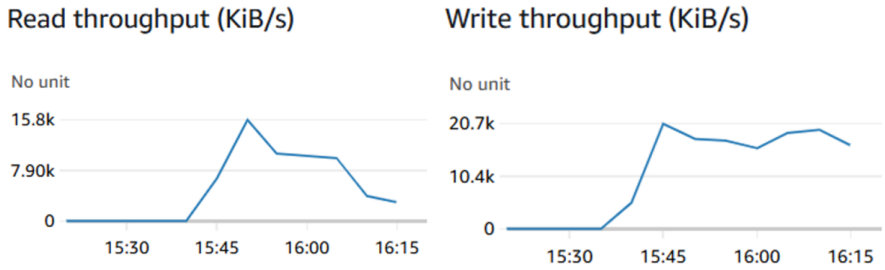
MetaDefender Core with Metascan (8 AV engines):

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

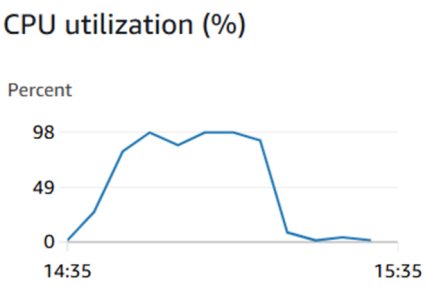
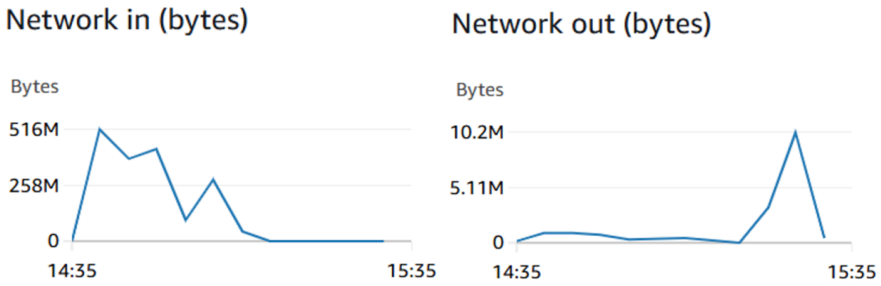
MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

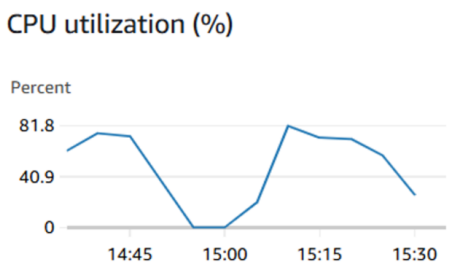
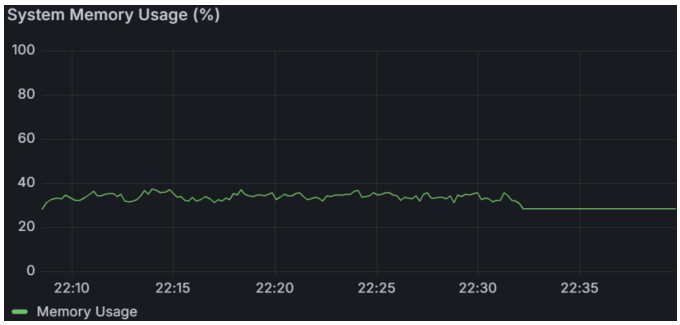
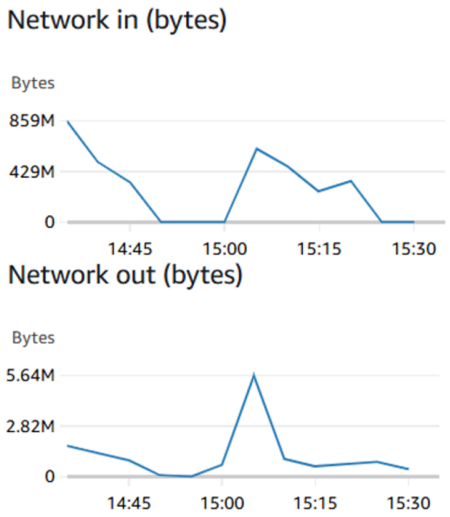
MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

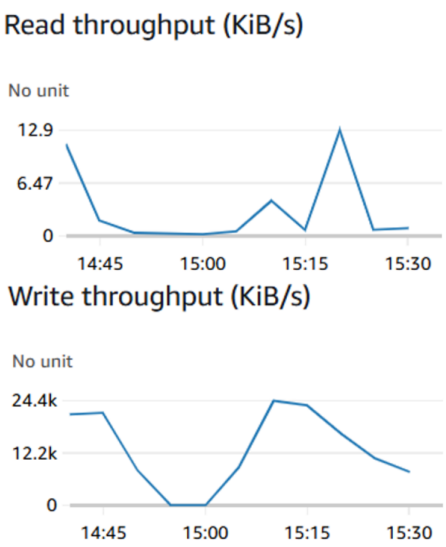
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

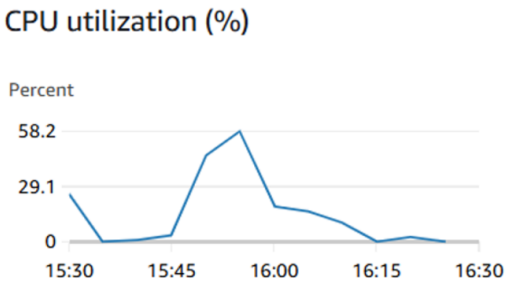
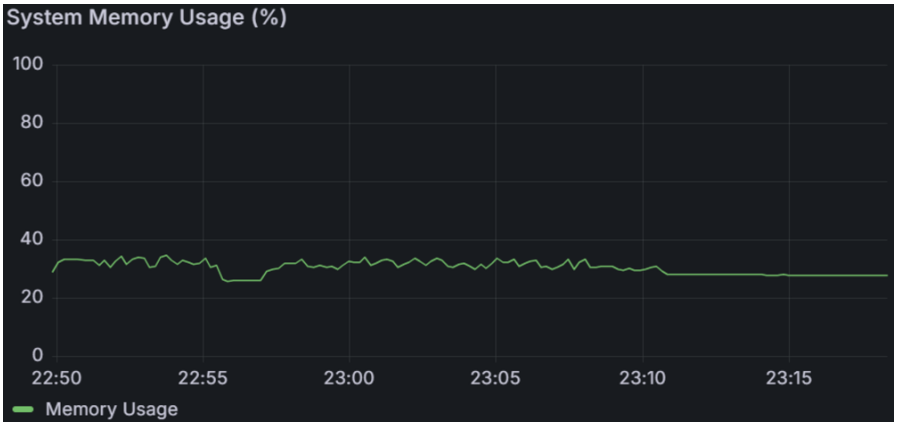
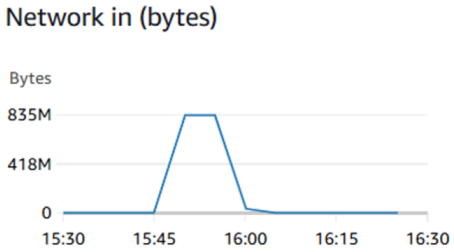
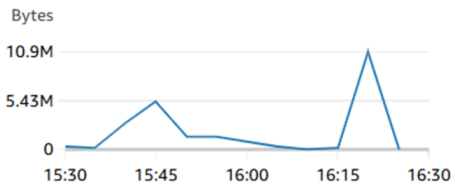
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

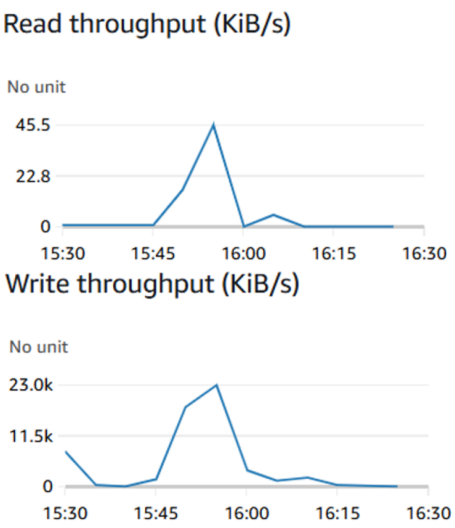
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

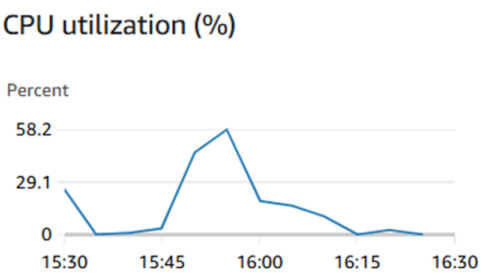
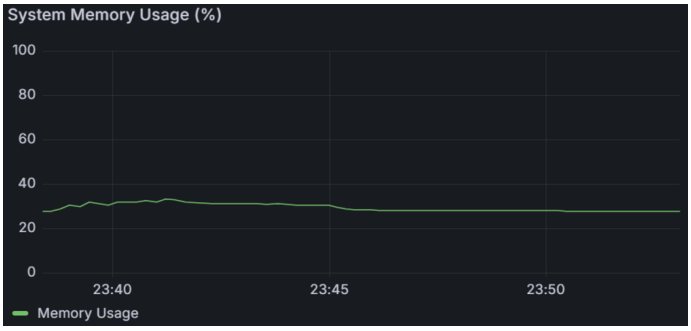
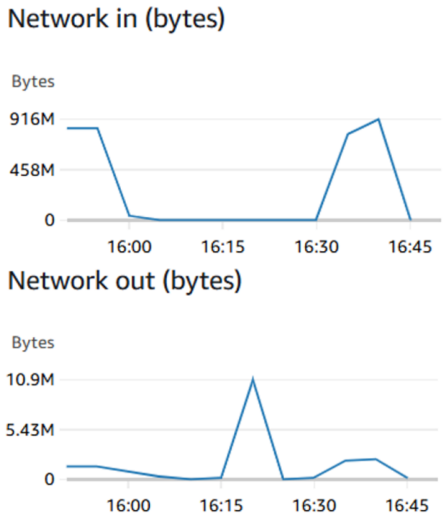
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	 

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability:

	Utilization charts
CPU usage	
Memory usage	
Network usage	

Utilization charts

Disk usage



Standalone DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Ubuntu 22.04	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.57-236

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

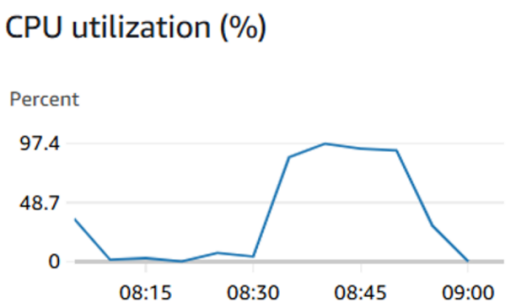
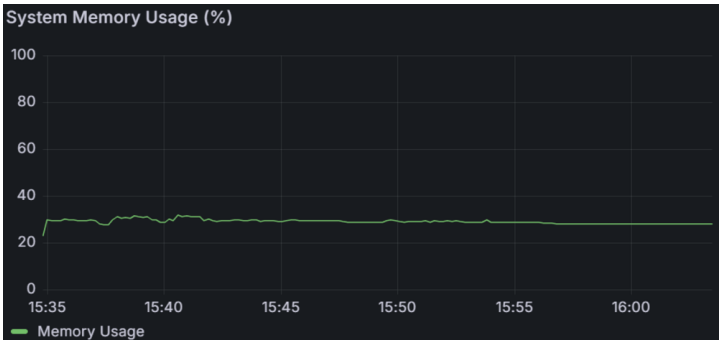
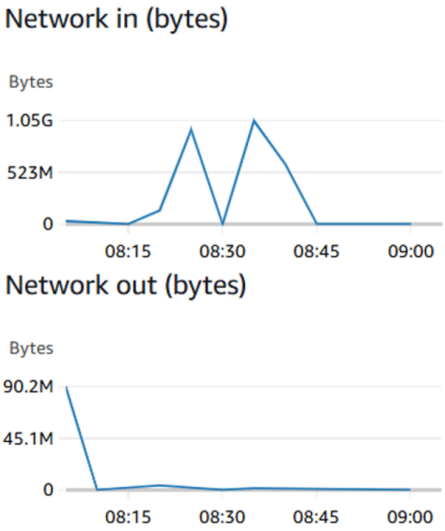
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV)	30.2	69 / 98	29 / 32		252,666	0.0142
Deep CDR	19.4	48 / 62	36 / 49		393,325	0.009
Proactive DLP	16.5	34 / 39	32 / 48		462,455	0.008
Vulnerability	12	50 / 65	46 / 46		635,875	0.006

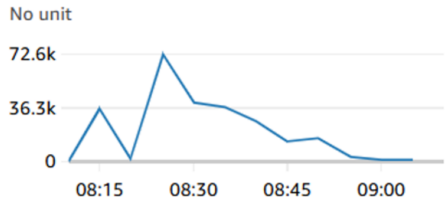
MetaDefender Core with common engine packages:

Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV) + Deep CDR	39.8	78 / 99	62 / 73		191,721	0.0188
Metascan (10 AV) + Deep CDR + Proactive DLP	43.5	61 / 84	47 / 67		175,414	0.0205
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	44.8	62 / 88	69 / 83		170,324	0.0211

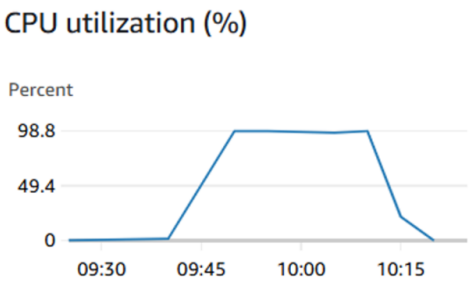
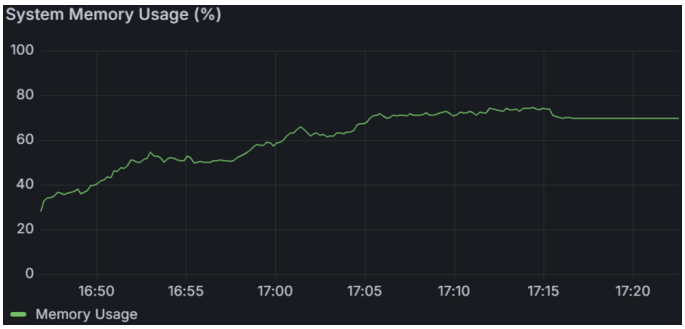
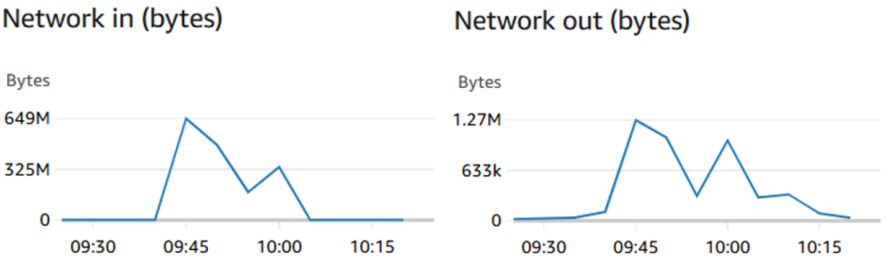
System resource utilization charts

MetaDefender Core with Metascan (10 AV engines) :

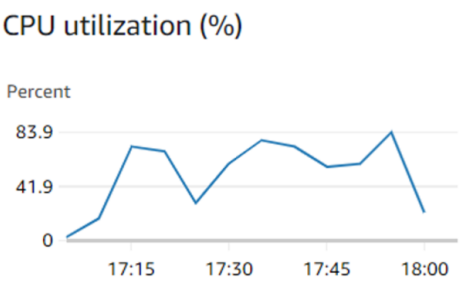
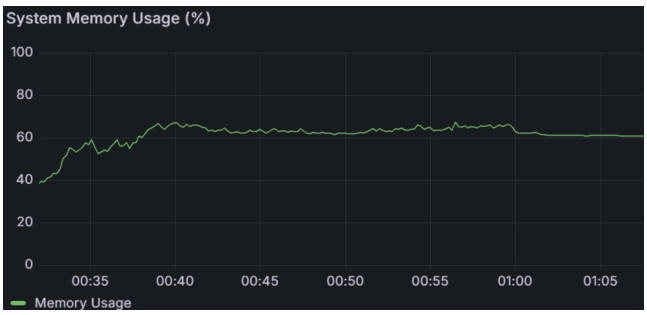
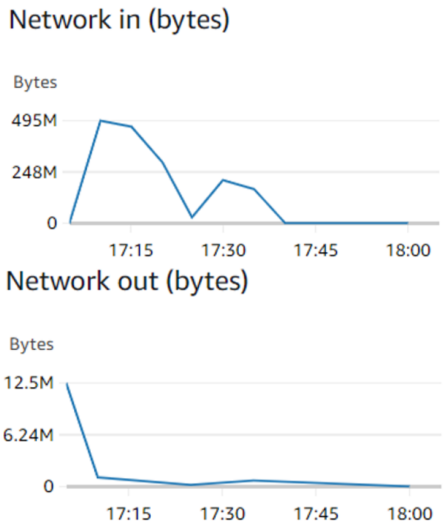
	Utilization charts
CPU usage	
Memory usage	
Network usage	

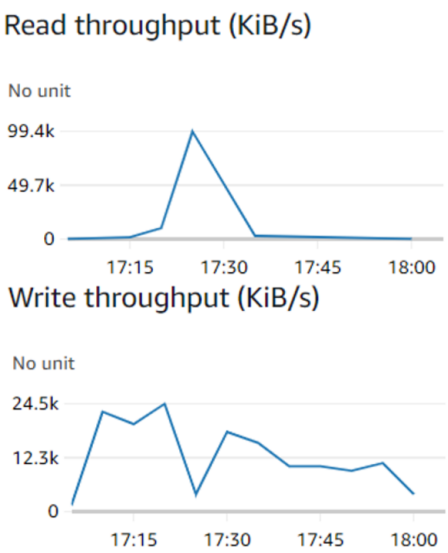
	Utilization charts
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

MetaDefender Core with Metascan (10 AV engines) + Deep CDR :

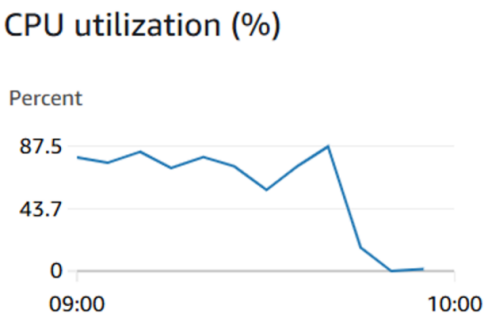
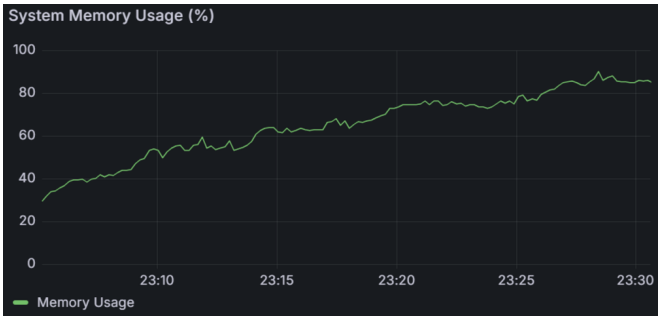
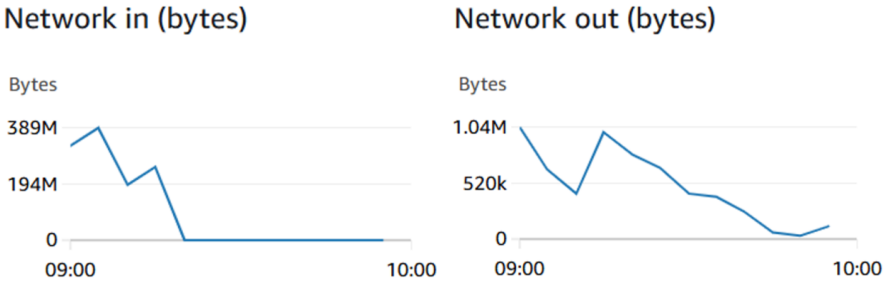
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP :

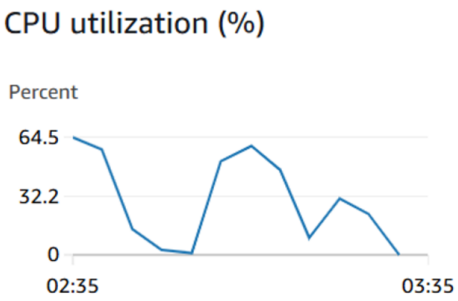
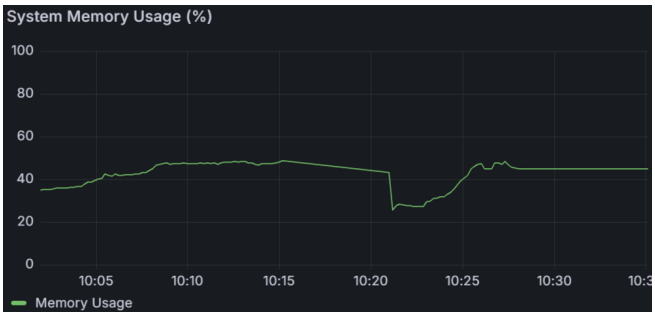
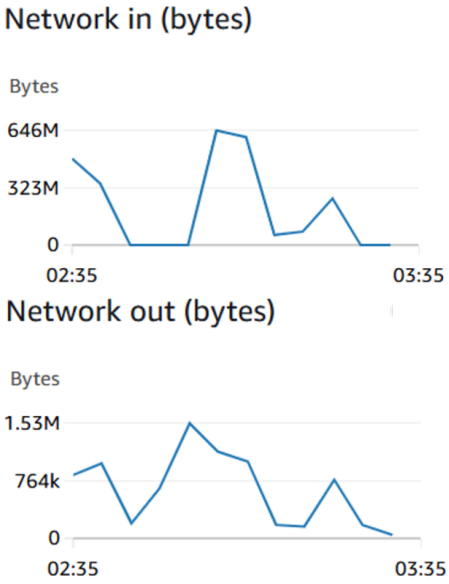
	Utilization charts
CPU usage	
Memory usage	
Network usage	

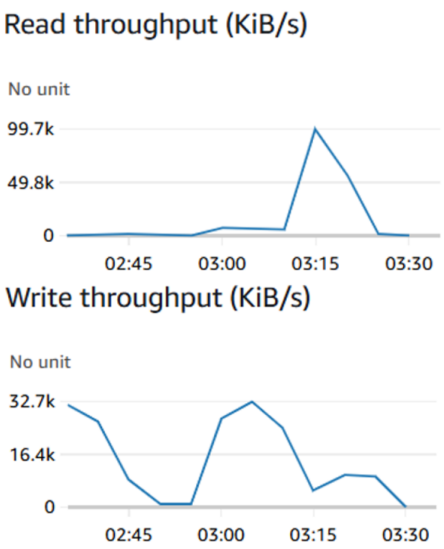
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP + Vulnerability :

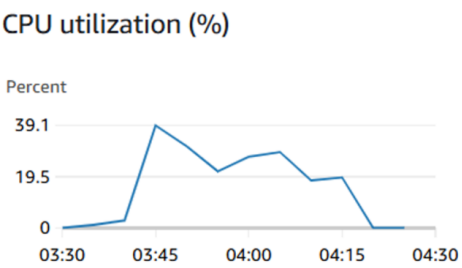
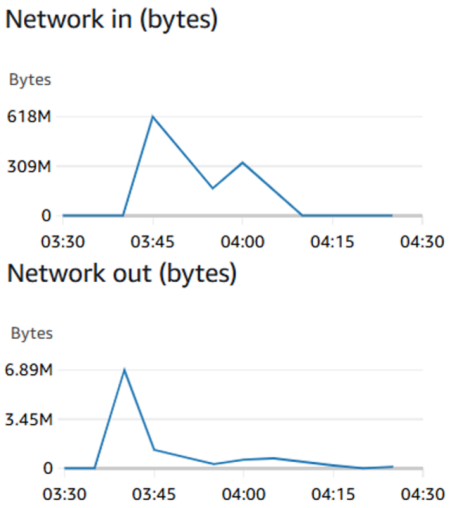
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

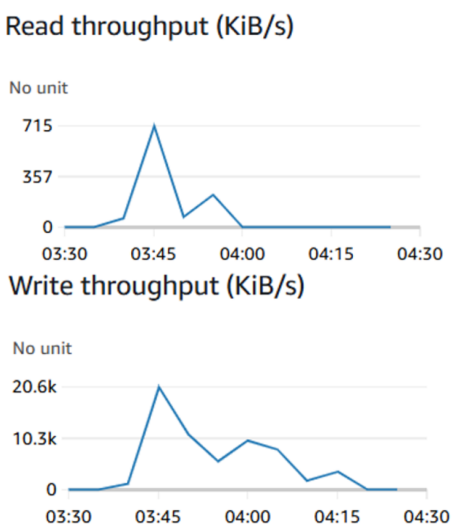
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

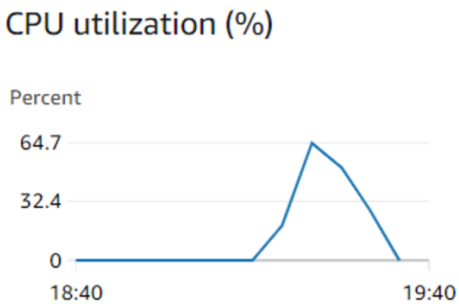
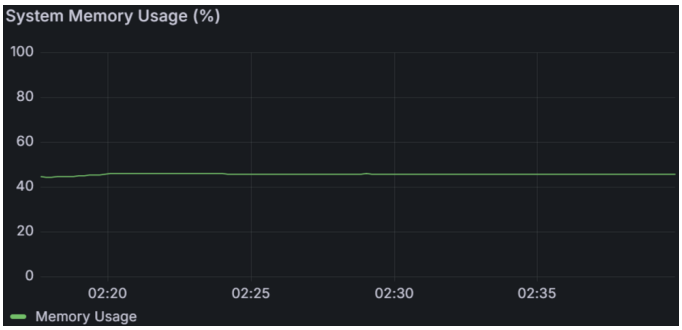
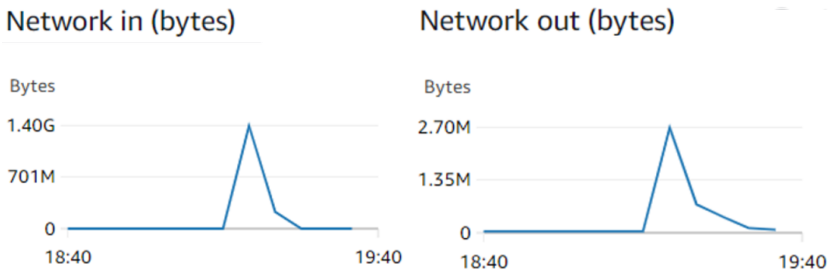
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability :

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

Shared DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

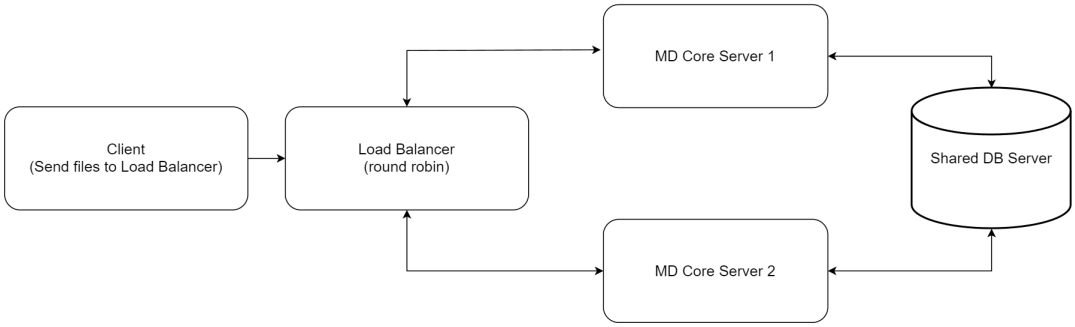
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV)	11.9	37.0	41.9	30.1	30.4	Receive: 6441.1 Send: 769.6	Receive: 6200.2 Send: 693.7	640,821.2	0.0
Deep CDR	10.5	60.1	66.7	30.3	32.2	Receive: 6379.8 Send: 594.8	Receive: 7146.1 Send: 609.6	729,368.5	0.0
Proactive DLP	7.8	38.7	37.5	29.7	31.0	Receive: 8842.6 Send: 791.8	Receive: 8635.6 Send: 729.1	965,455.7	0.0
Vulnerability	4.4	52.2	53.0	26.0	26.9	Receive: 16,215.5 Send: 1006.8	Receive: 15,785.7 Send: 997.2	1714,449.3	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV) + Deep CDR	15.4	72.4	75	34.3	35.4	Receive: 5436.6 Send: 611.3	Receive: 5589.3 Send: 607.2	495,697.4	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP	16.9	77.2	80.2	36.4	37.9	Receive: 5254 Send: 585.4	Receive: 5041.3 Send: 576.1	450,657.6	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	18.0	72.1	80.4	38.6	38.0	Receive: 4504.3 Send: 574.1	Receive: 4649.5 Send: 571.1	423,010	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

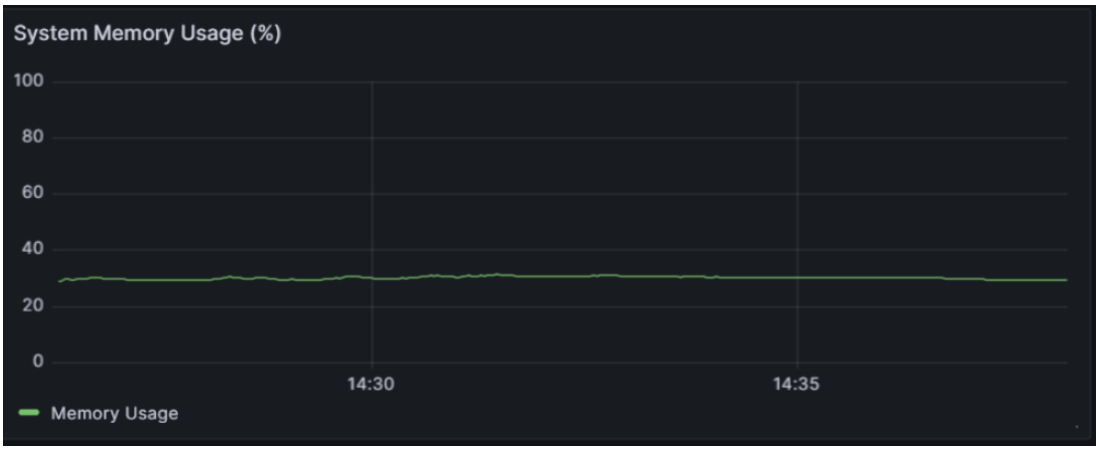
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2

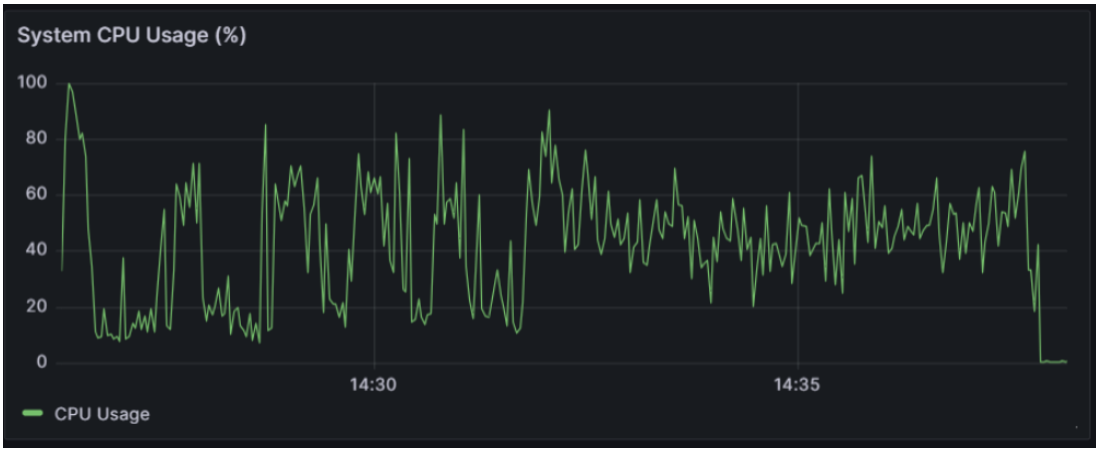


CPU usage

MD Core #1

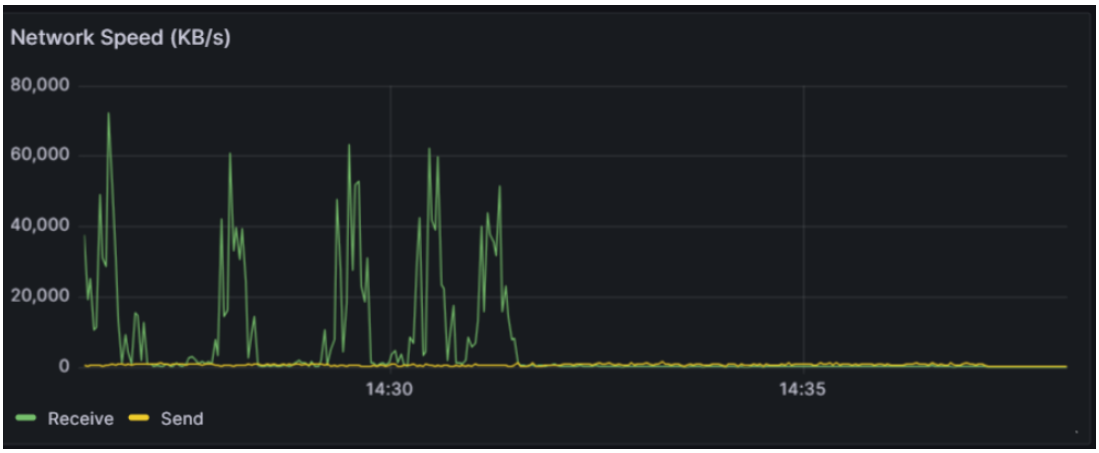


MD Core #2

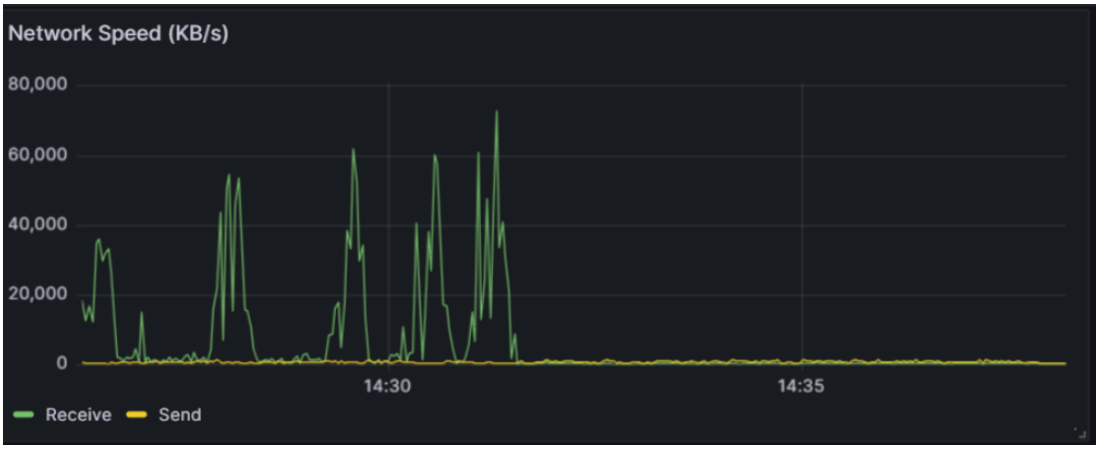


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2

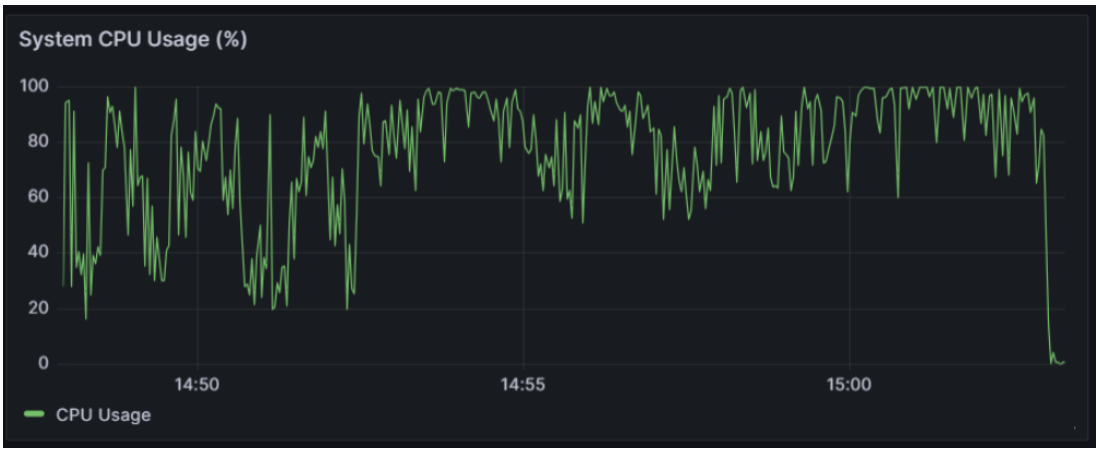


CPU usage

MD Core #1

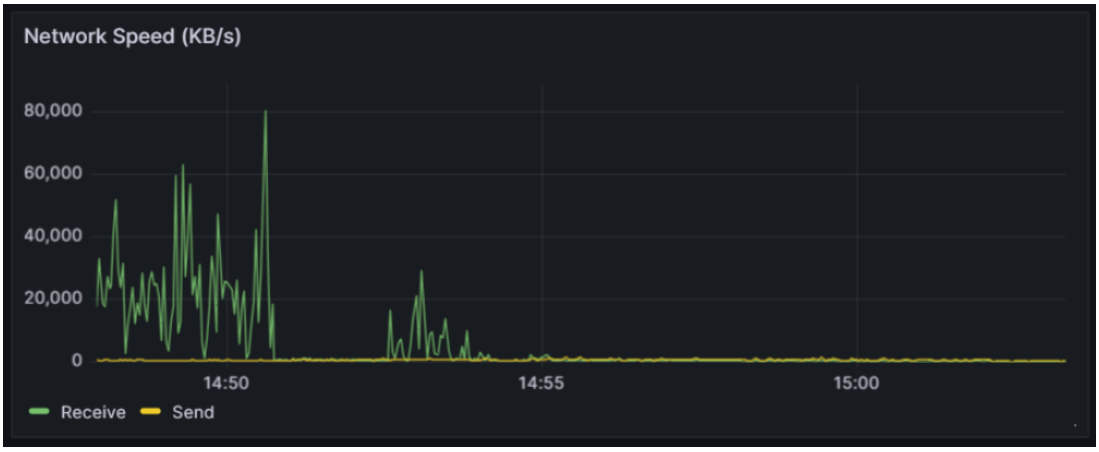


MD Core #2



Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

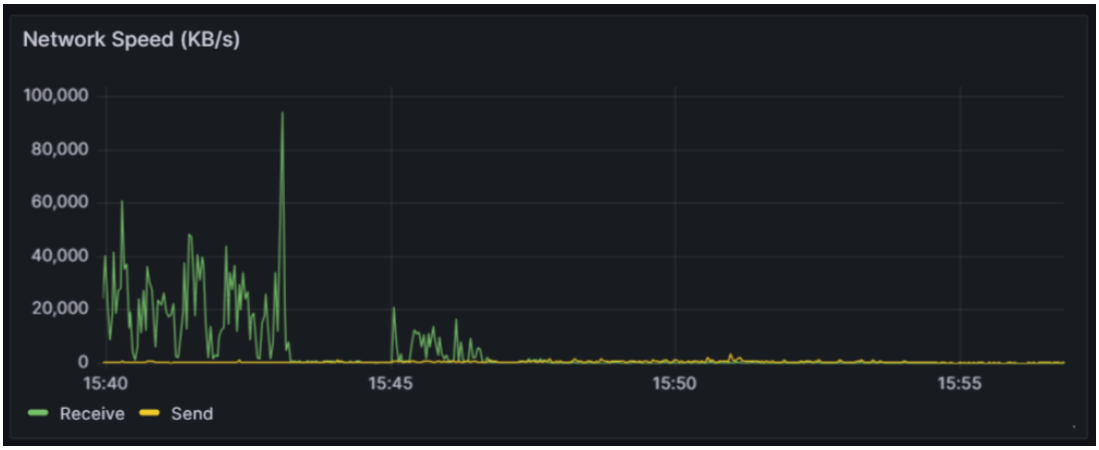


MD Core #2

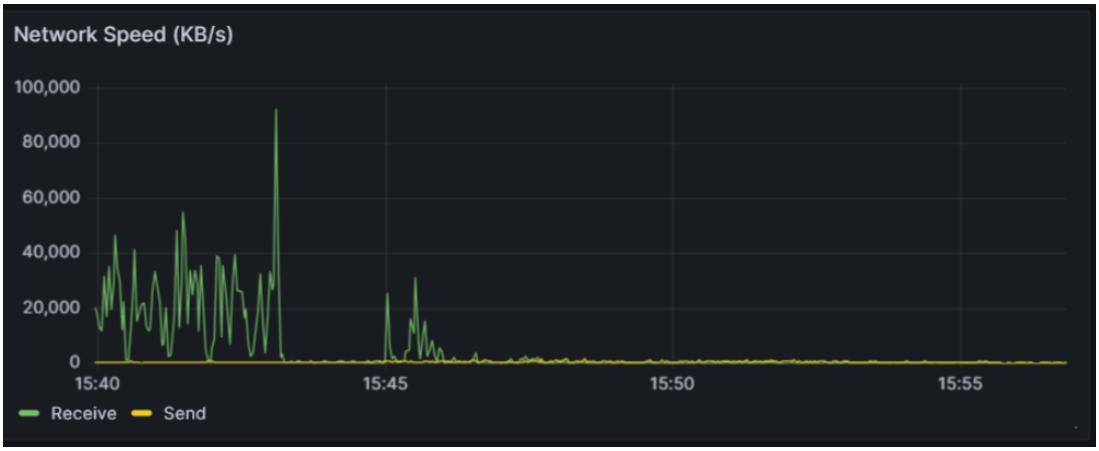


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1

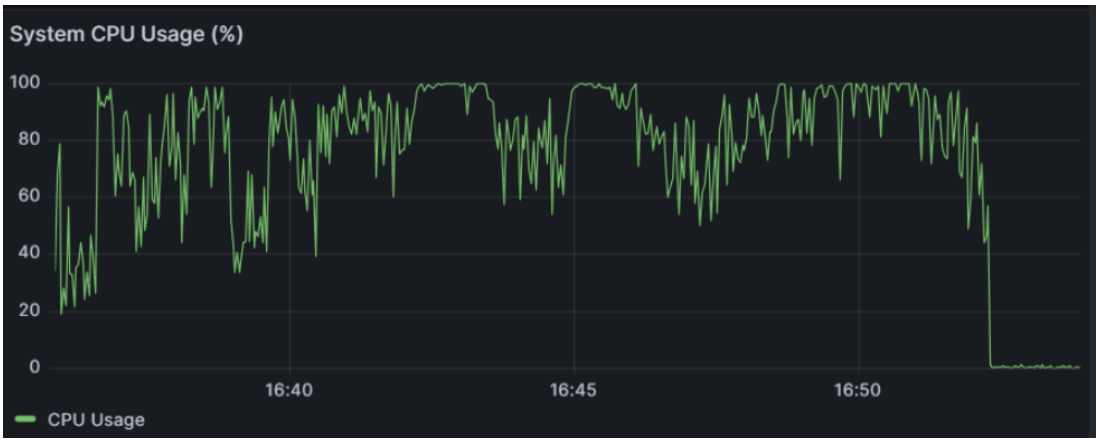


MD Core #2



CPU usage

MD Core #1

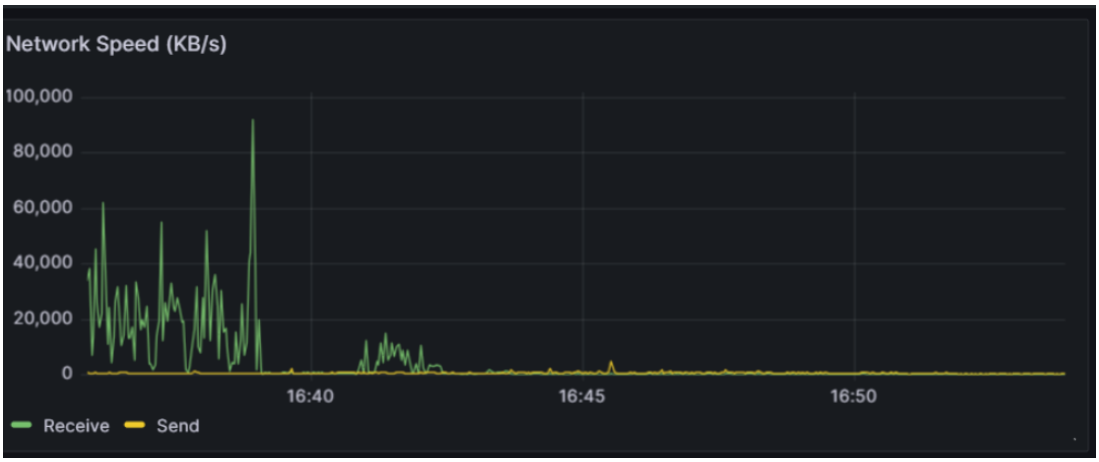


MD Core #2

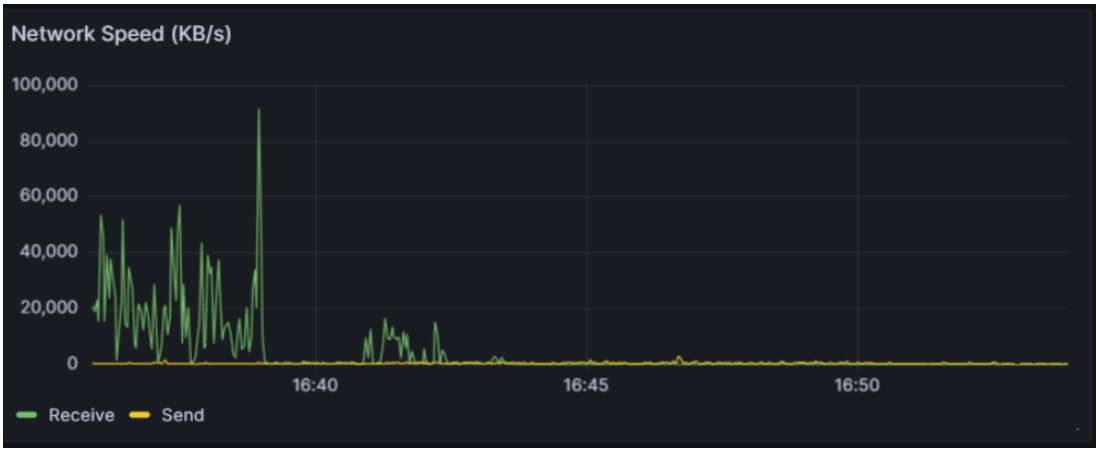


Network usage

MD Core #1



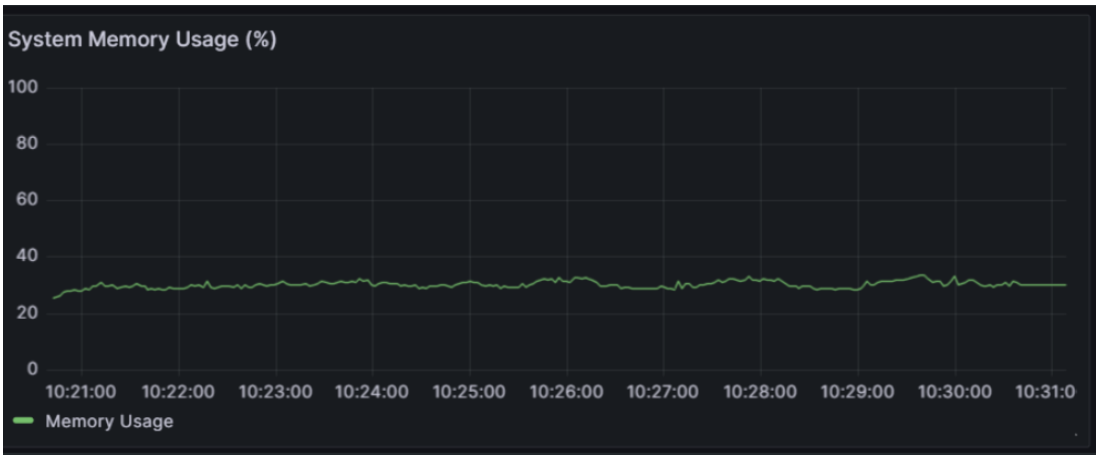
MD Core #2



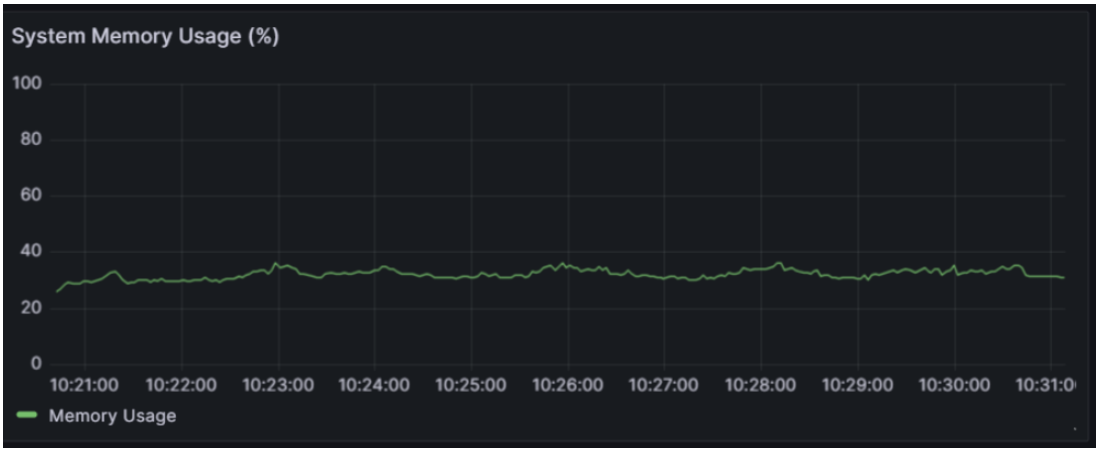
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

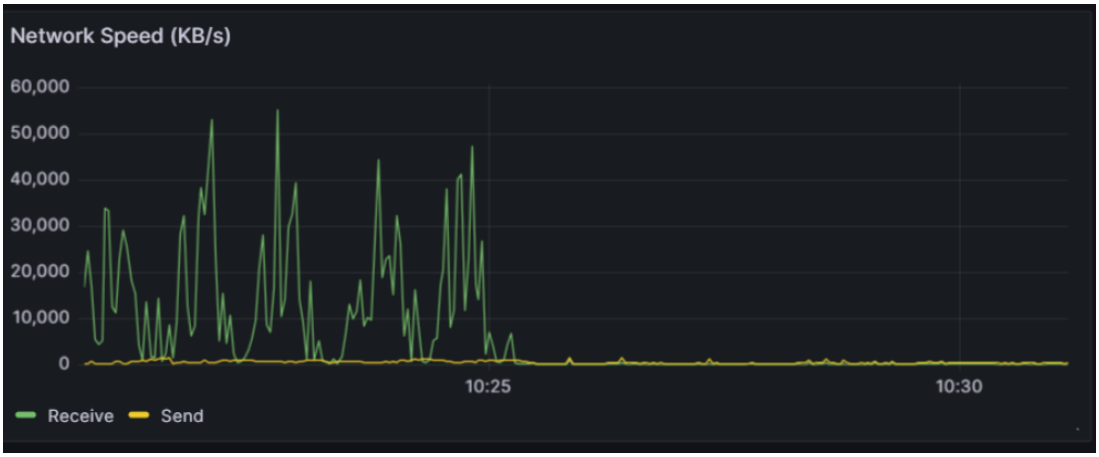


MD Core #2

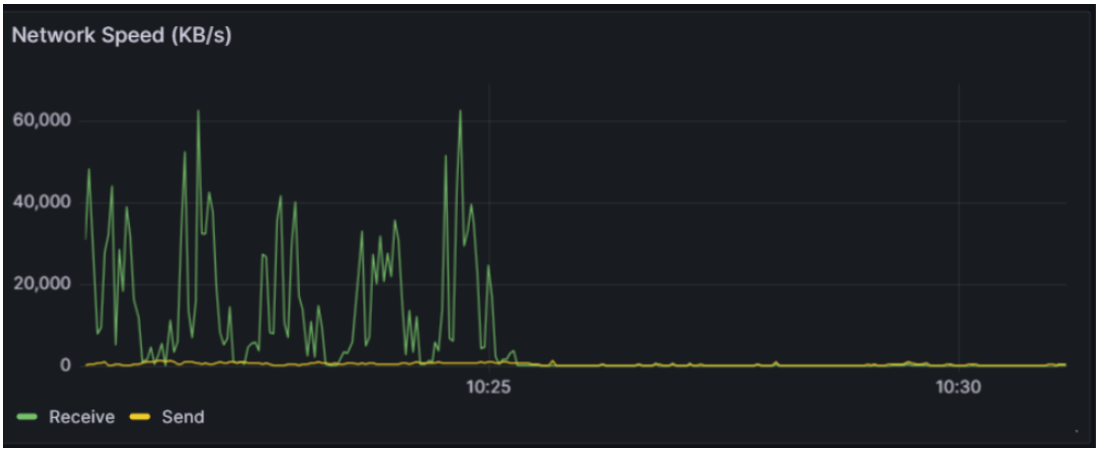


Network usage

MD Core #1



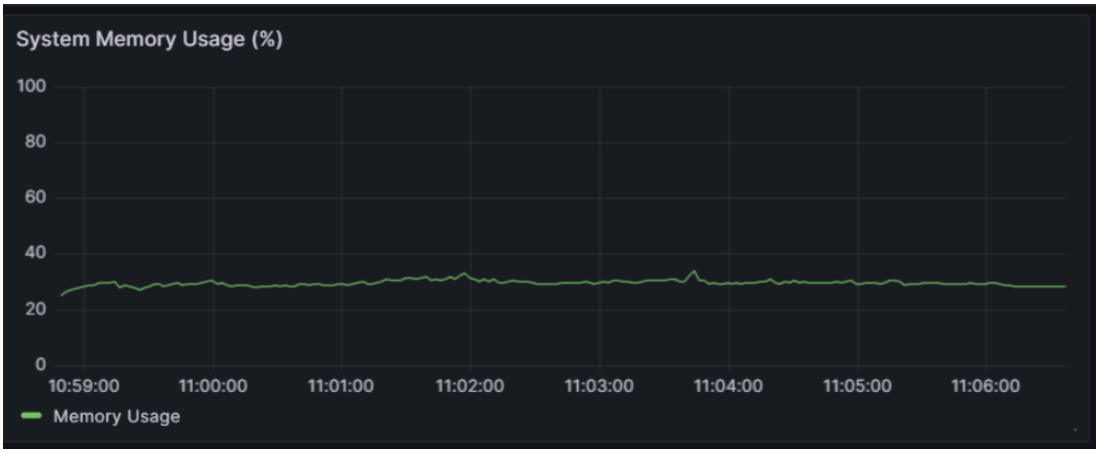
MD Core #2



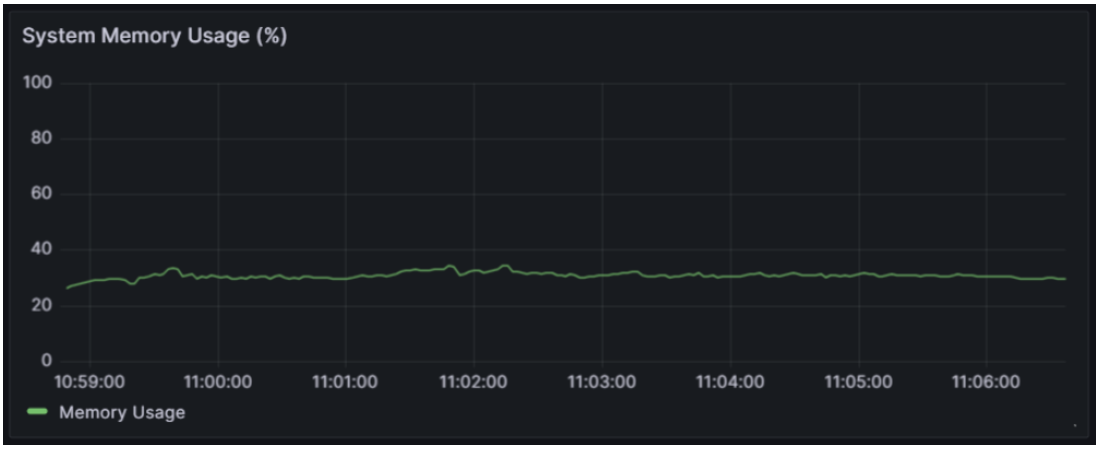
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

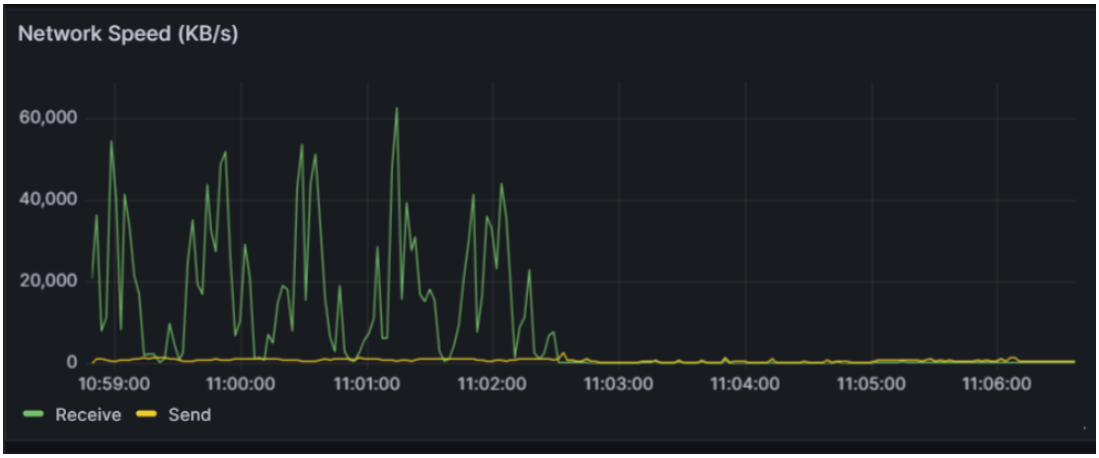


MD Core #2

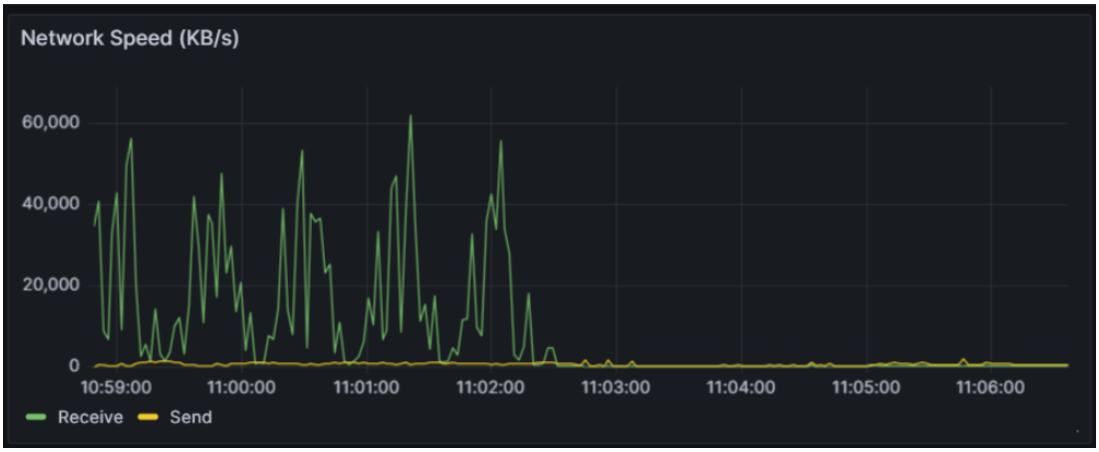


Network usage

MD Core #1



MD Core #2



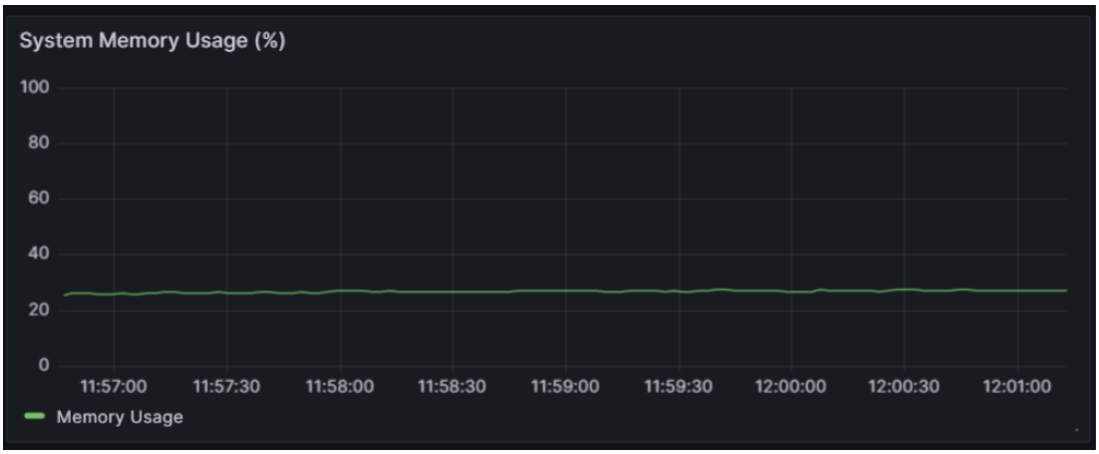
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

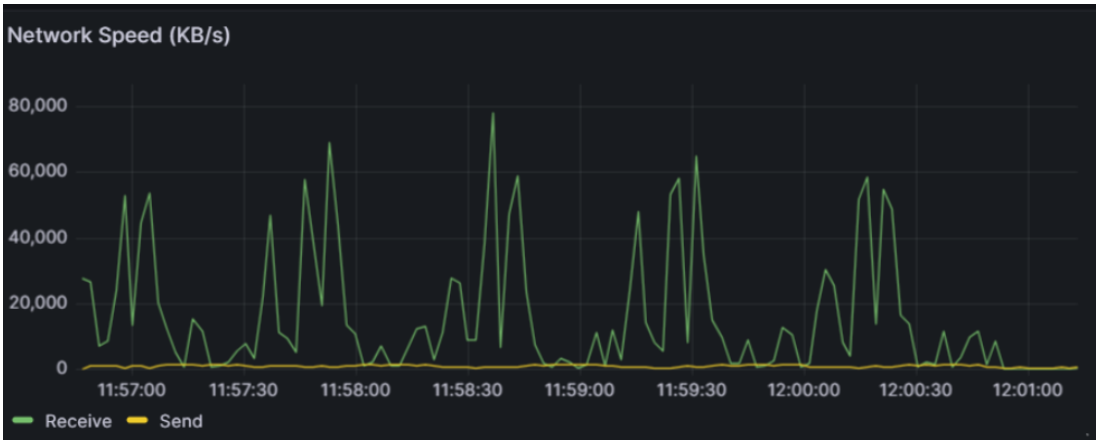


MD Core #2

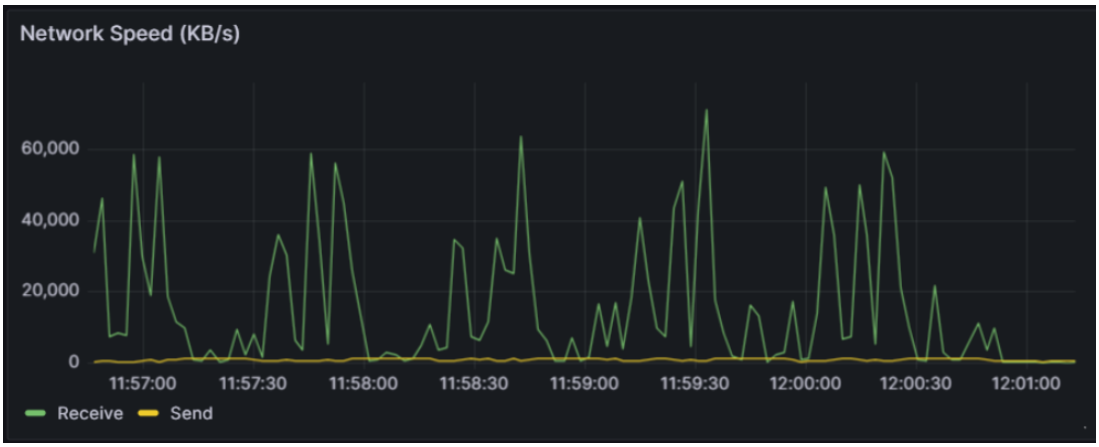


Network usage

MD Core #1



MD Core #2



Shared DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

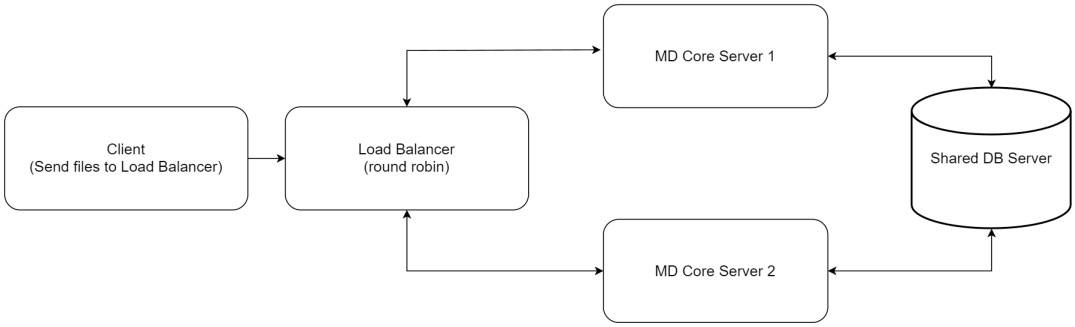
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV)	15.5	63.1	73.3	25.2	25.8	Receive: 4555.9 Send: 694.4	Receive: 4516.2 Send: 692.4	492,620.5	0.0
Deep CDR	9.70	50.7	51.2	31.9	32.5	Receive: 7062.9 Send: 623.2	Receive: 6653 Send: 654.1	786,189.6	0.0
Proactive DLP	7.2	35.2	35.0	27.9	27.5	Receive: 9047.4 Send: 802.4	Receive: 9859.8 Send: 787.5	1,055,110.8	0.0
Vulnerability	5.0	45.7	49.3	24.4	24.5	Receive: 13,585.7 Send: 966.7	Receive: 13,505.8 Send: 987.4	1,519,562.1	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV) + Deep CDR	19.9	79.1	77.6	34.0	34.2	Receive: 3785.9 Send: 573.8	Receive: 3663.1 Send: 577.7	382,716	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP	21.6	80.3	81.5	37.7	44.4	Receive: 3305.6 Send: 543.3	Receive: 3414.3 Send: 556.2	353,694.9	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	22.5	84.2	79.8	37.9	44.7	Receive: 3267.9 Send: 554.9	Receive: 3398.2 Send: 553.8	338,507.6	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

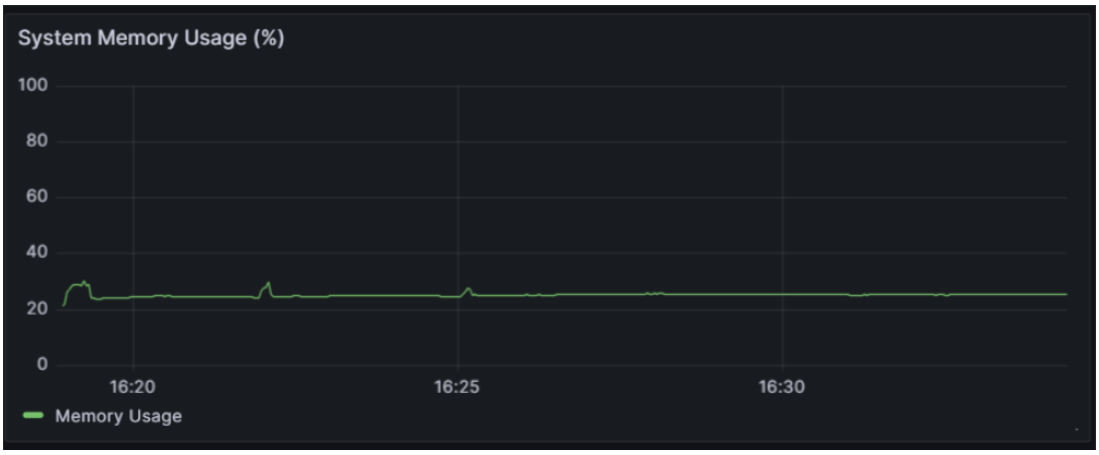
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

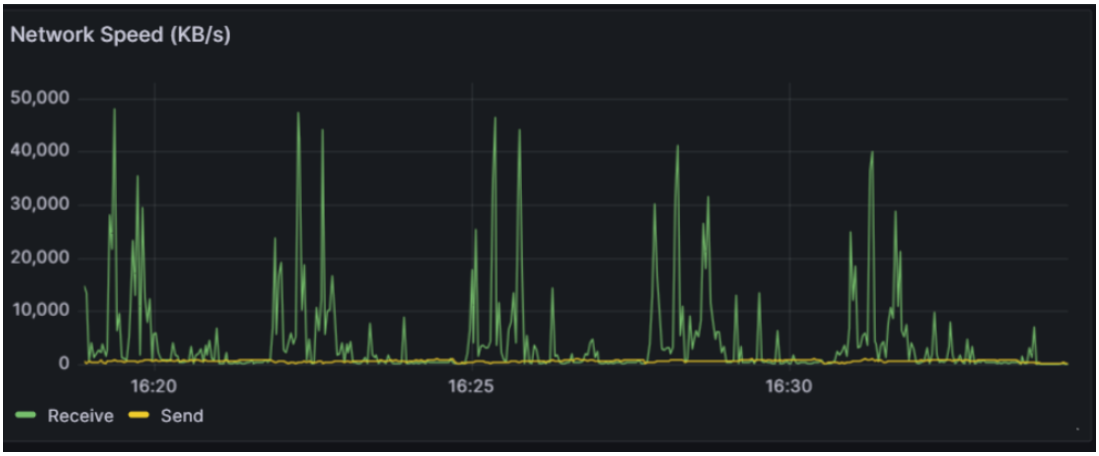


MD Core #2

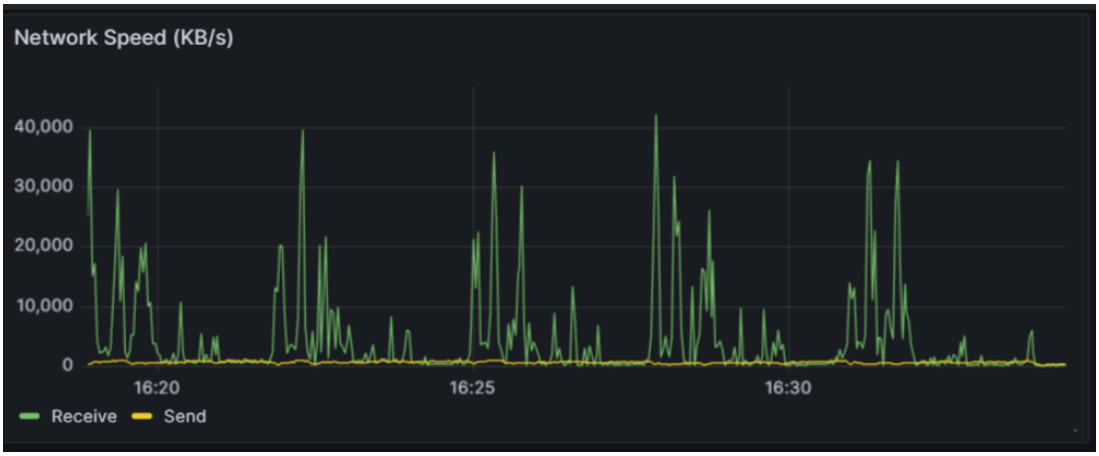


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

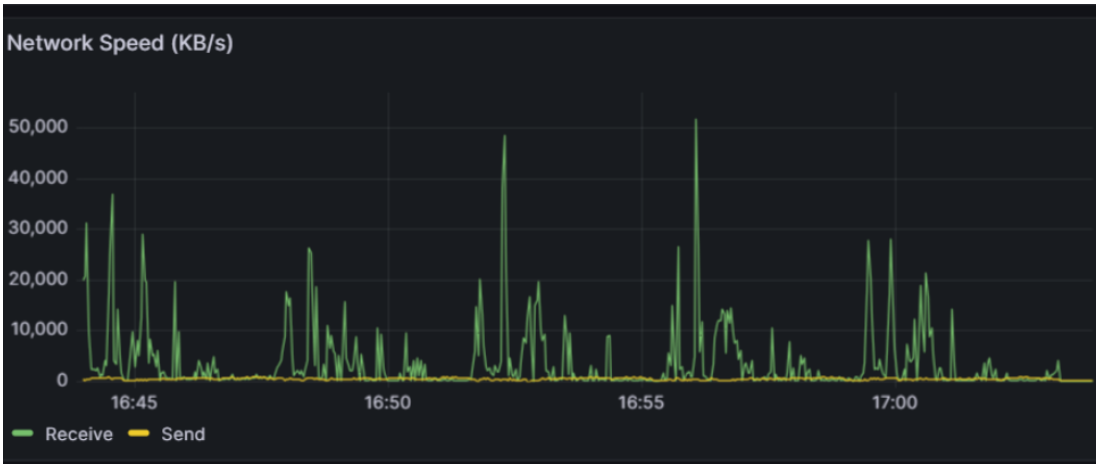


MD Core #2

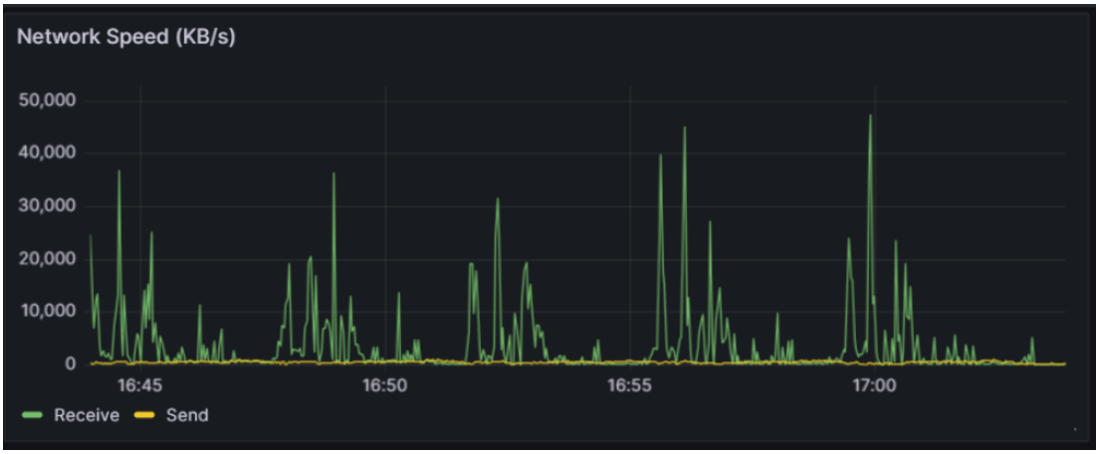


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1

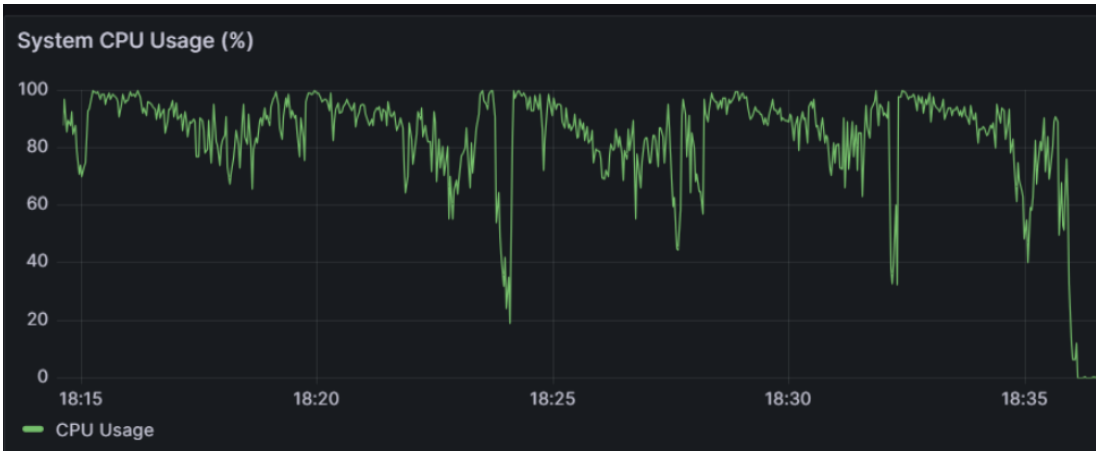


MD Core #2



CPU usage

MD Core #1

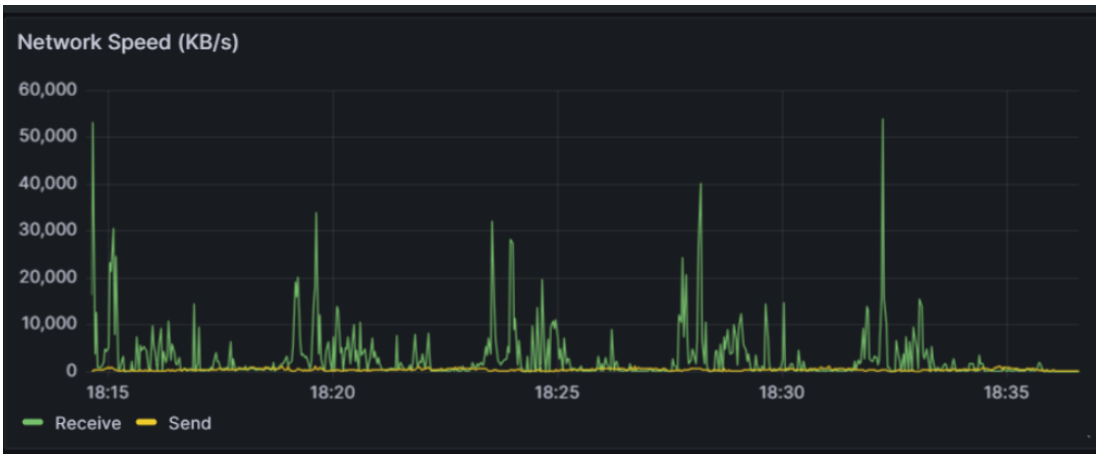


MD Core #2

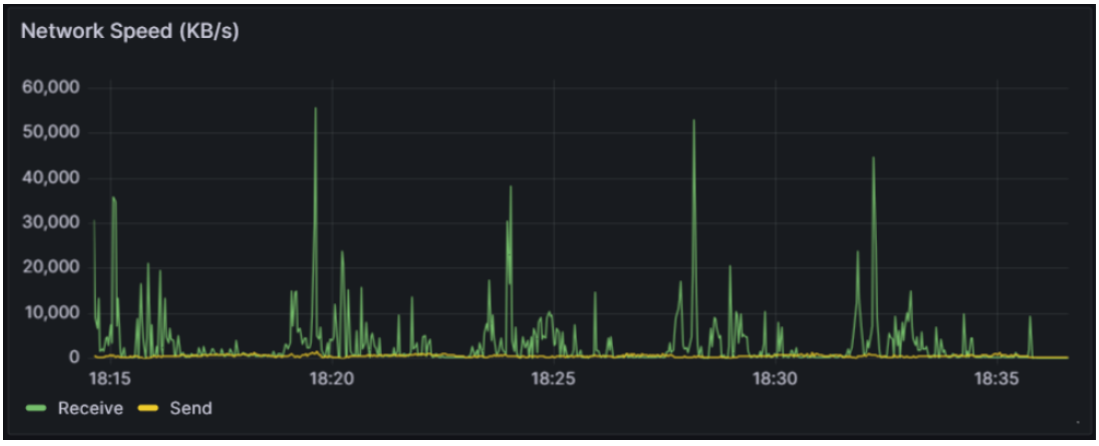


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

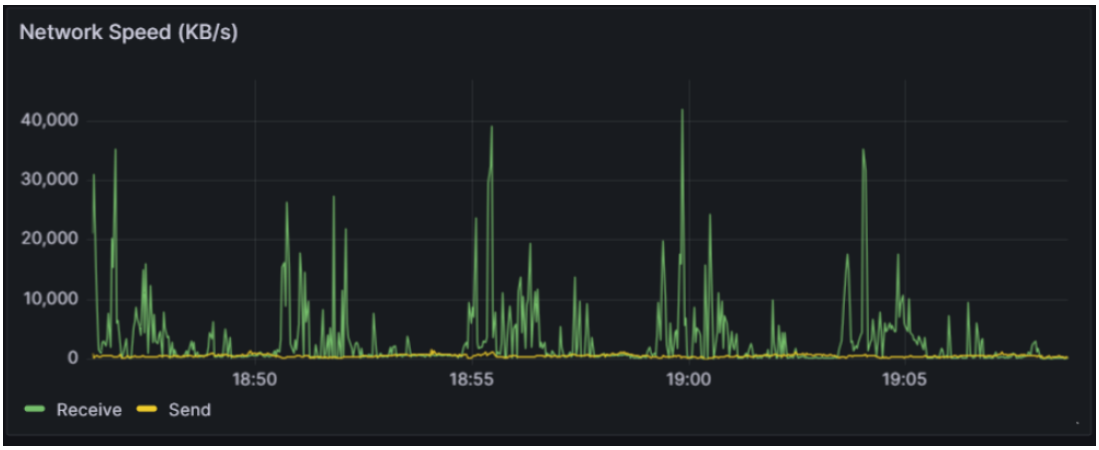


MD Core #2

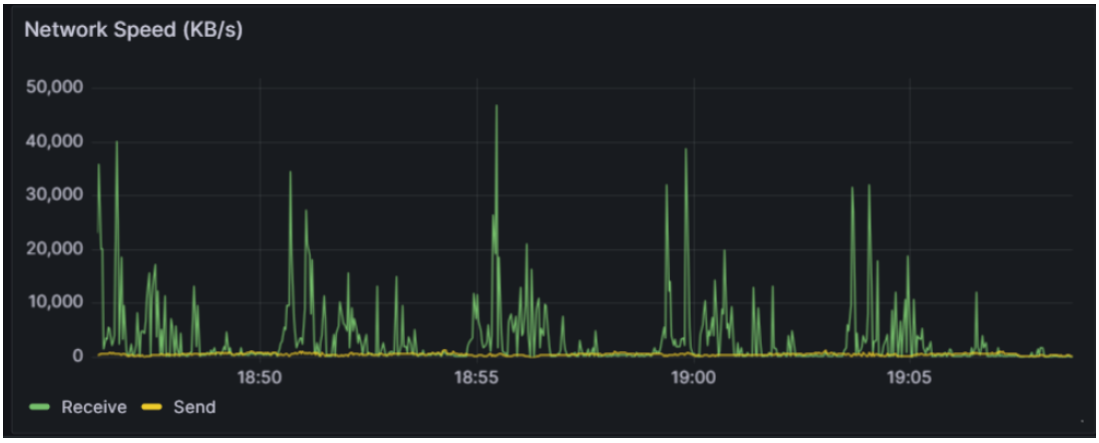


Network usage

MD Core #1



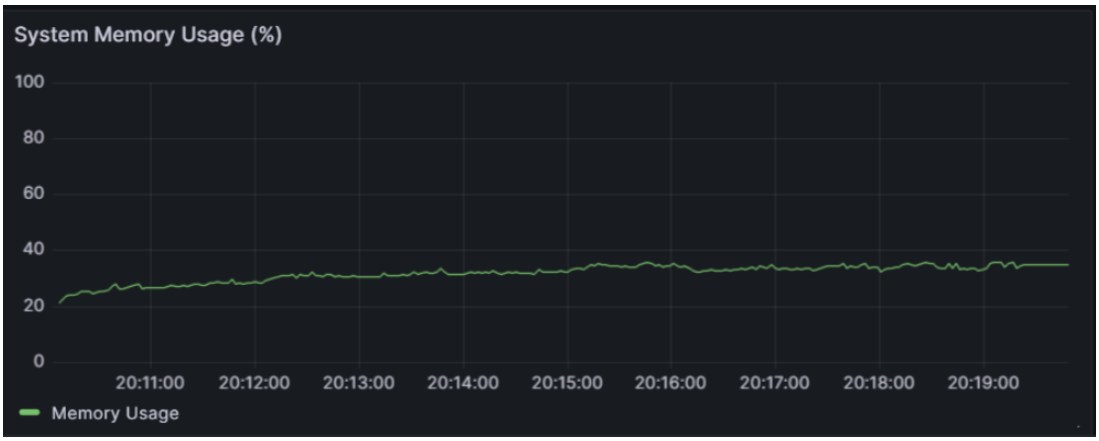
MD Core #2



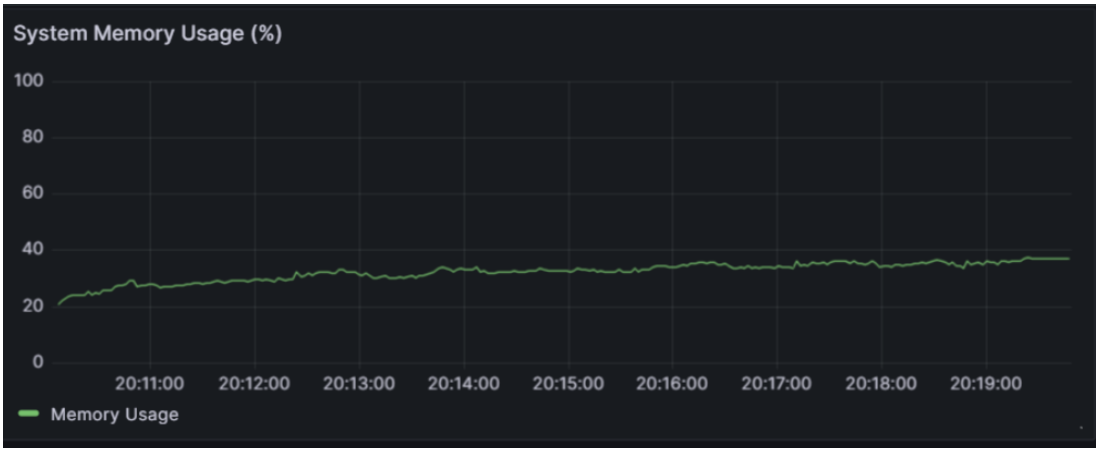
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1

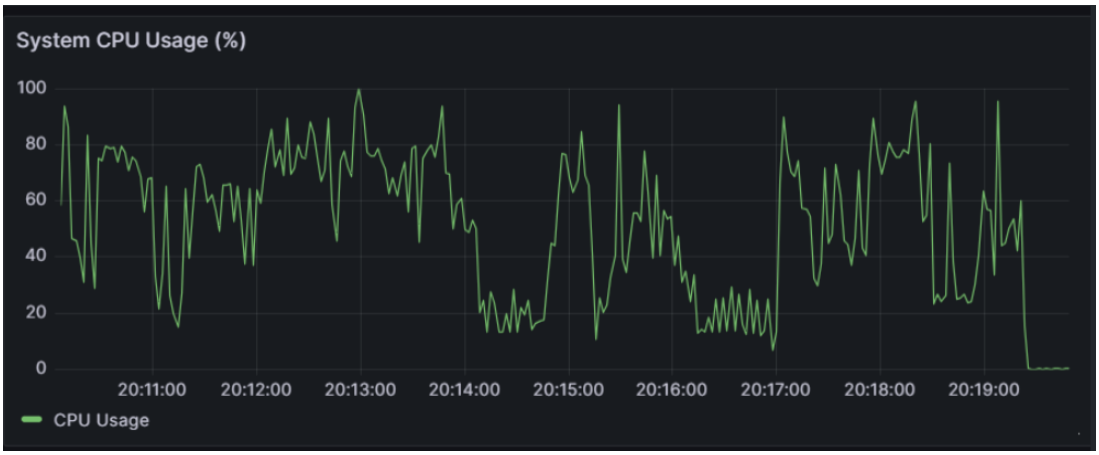


MD Core #2



CPU usage

MD Core #1

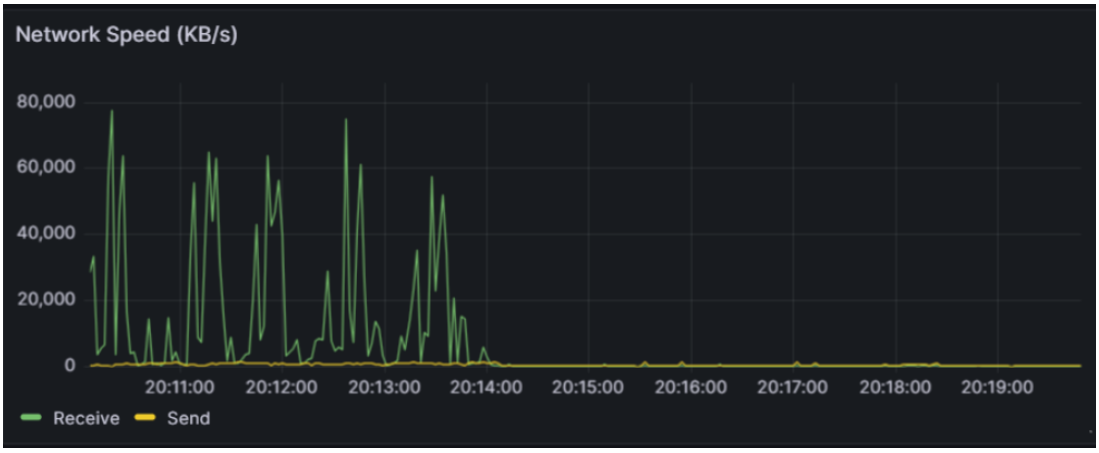


MD Core #2

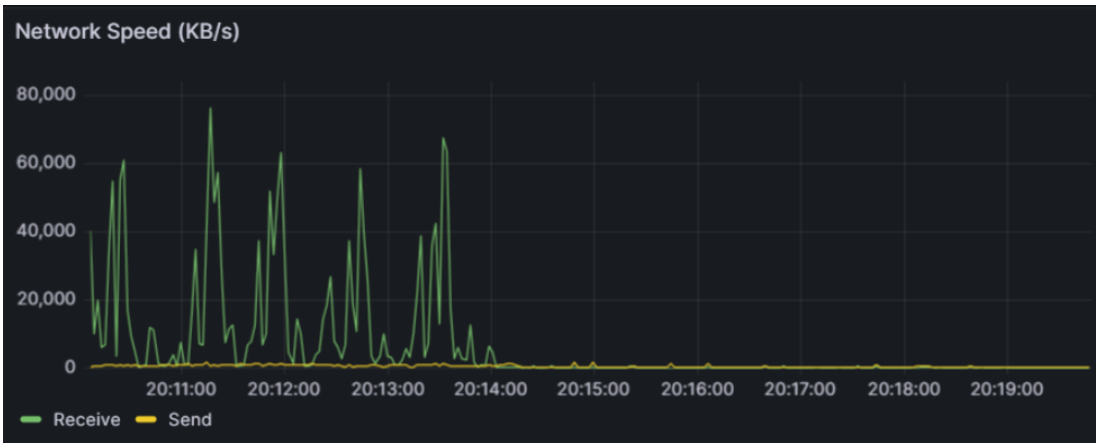


Network usage

MD Core #1



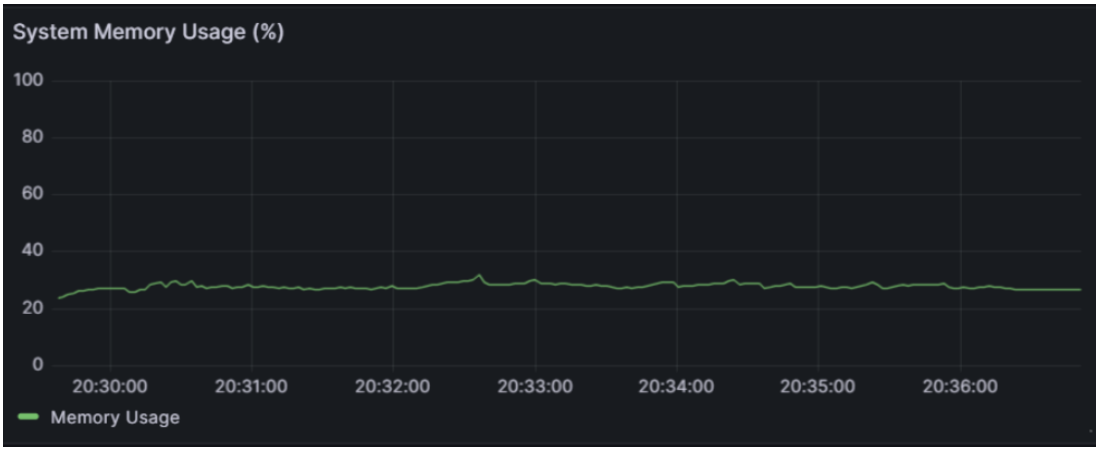
MD Core #2



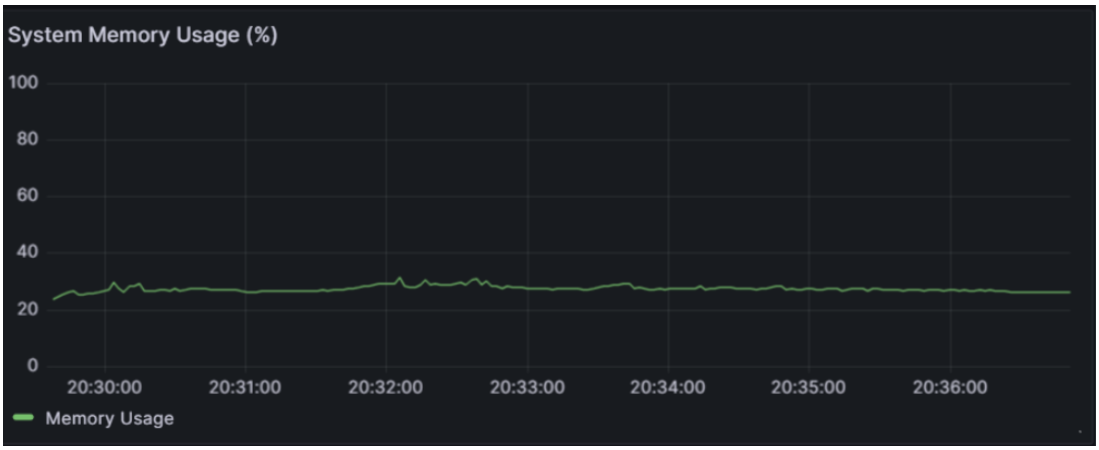
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1

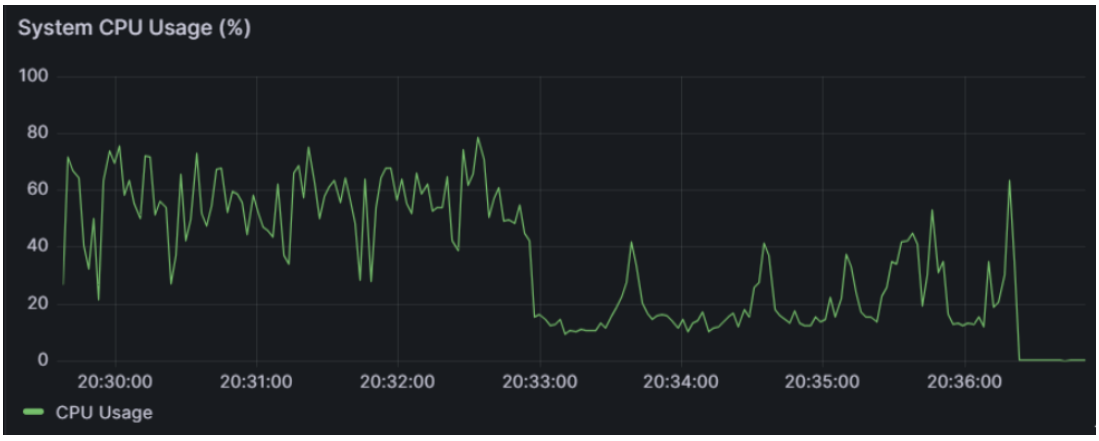


MD Core #2

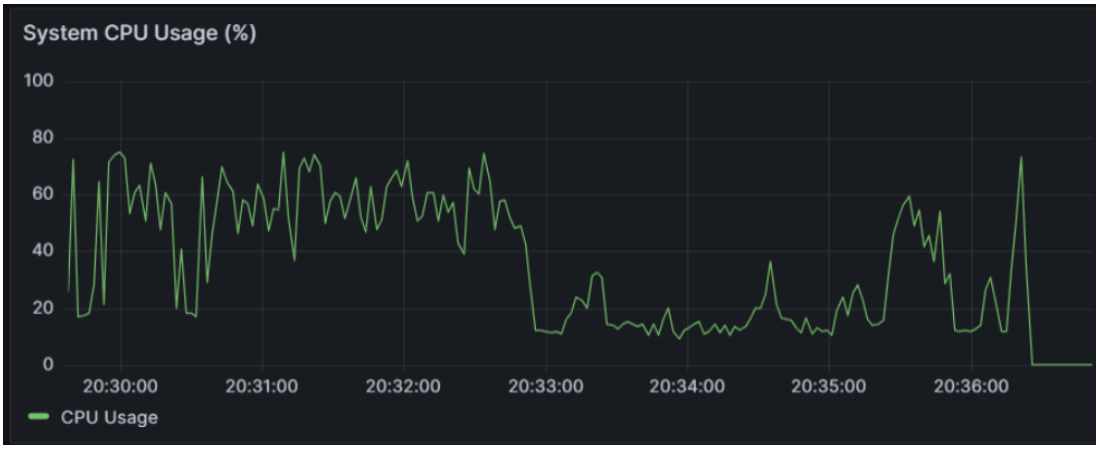


CPU usage

MD Core #1

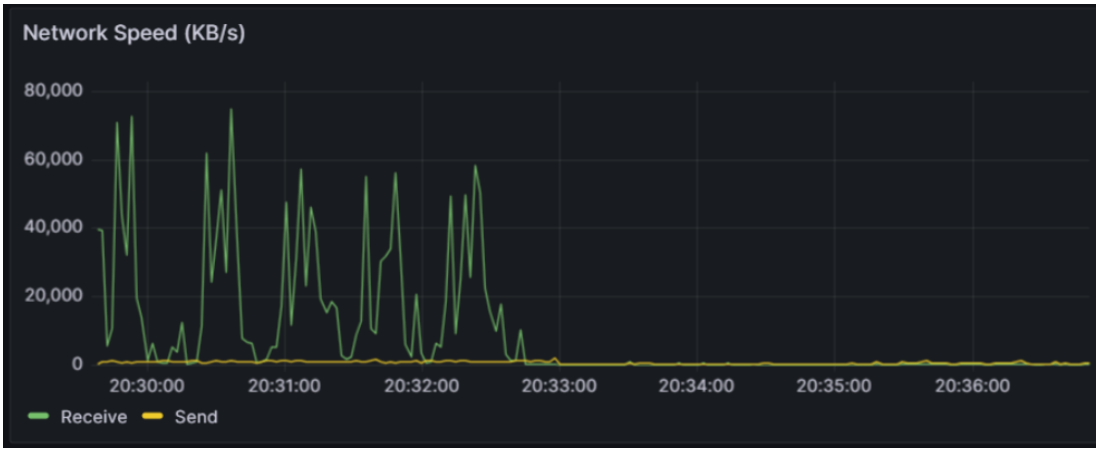


MD Core #2

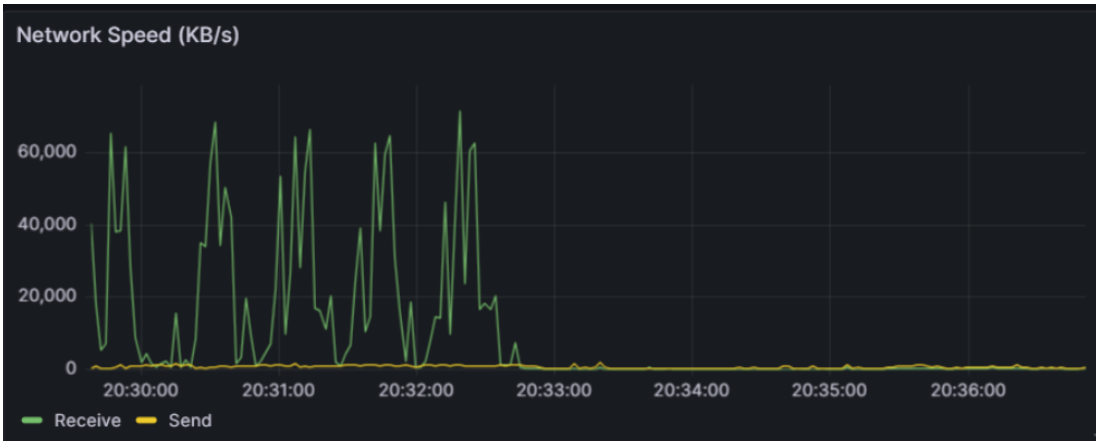


Network usage

MD Core #1



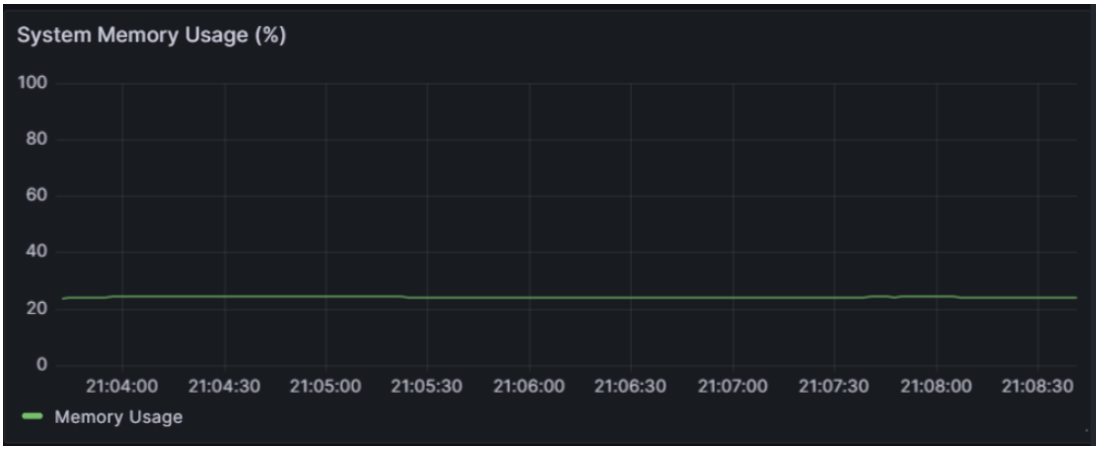
MD Core #2



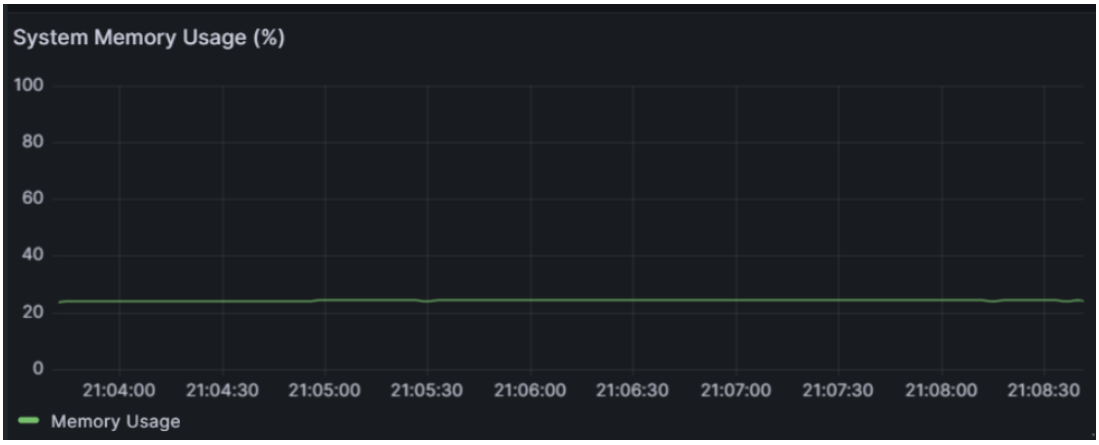
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

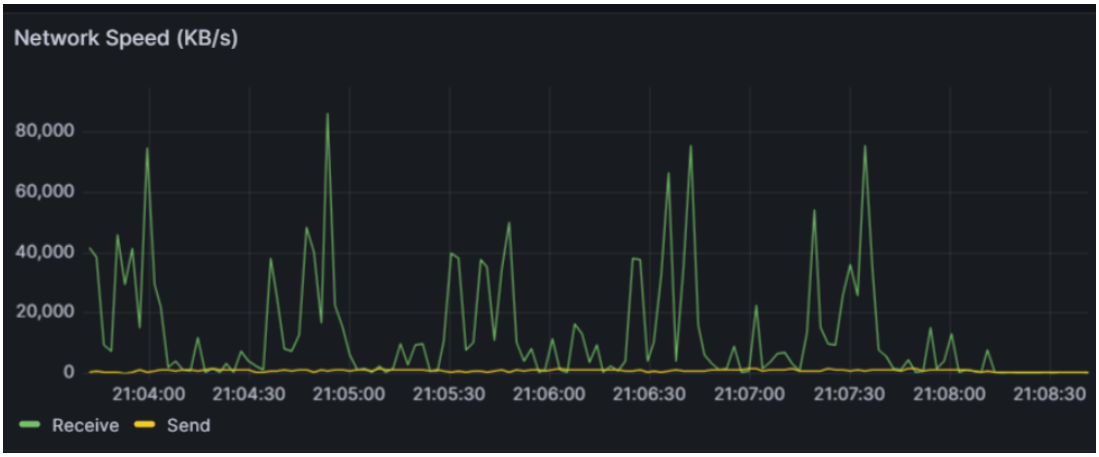


MD Core #2

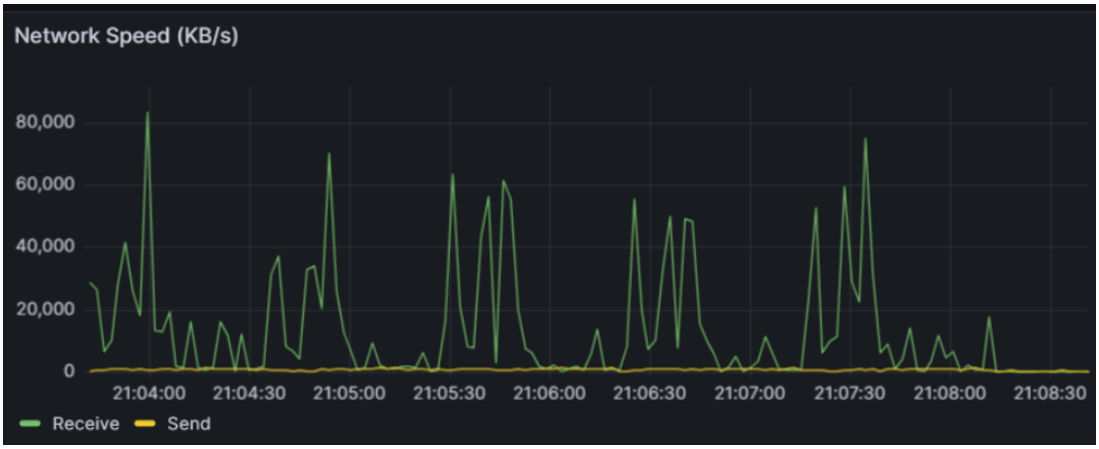


Network usage

MD Core #1



MD Core #2

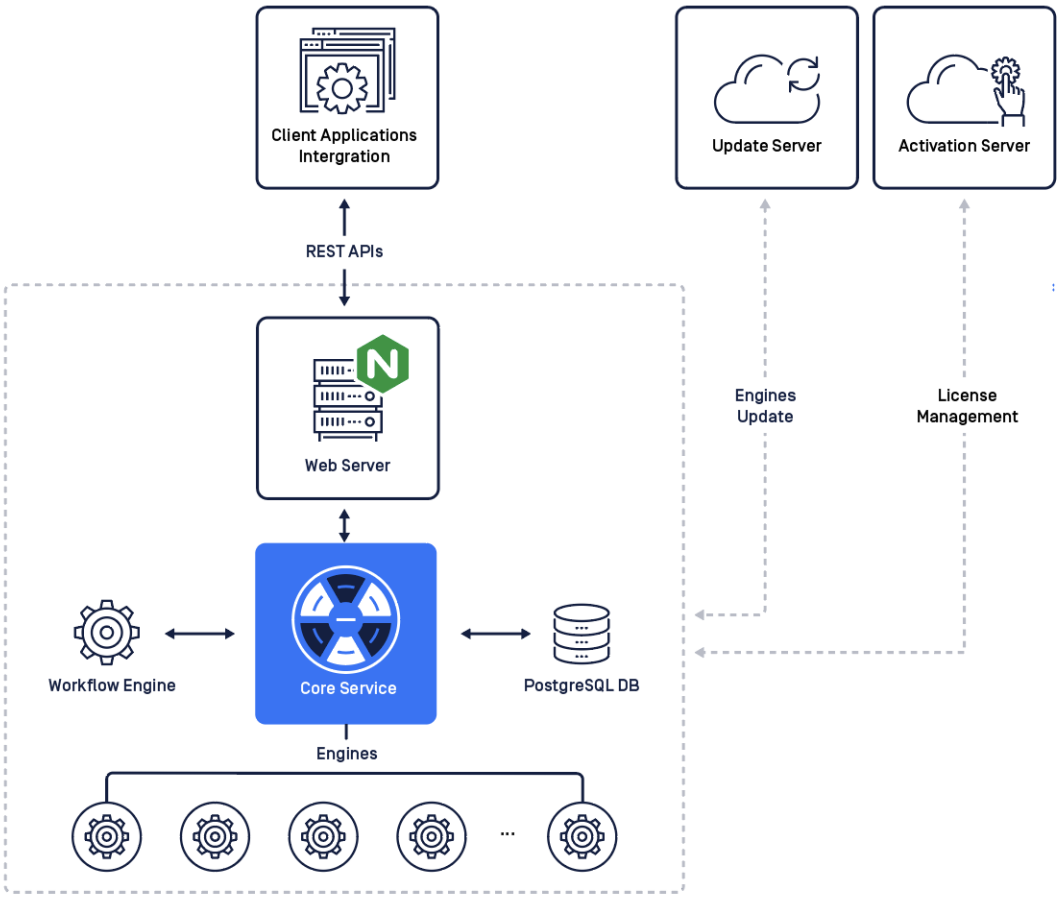

Network Speed (KB/s)
80,000
60,000
40,000
20,000
0
21:04:00
21:04:30
21:05:00
21:05:30
21:06:00
21:06:30
21:07:00
21:07:30
21:08:00
21:08:30
Receive
Send

MetaDefender Core Documentation

The users can consult this web page or, alternatively, they can download the manual in pdf format from the link below:

MetaDefender Core manual (SHA256:).

Recommended System Configuration



MetaDefender Core supports a wide range of deployment options, including air-gapped environments, providing the flexibility to choose the environment that best suits customers' requirements and infrastructure.

- Physical server.
- Virtualization platforms like VMware, Hyper-V and XenServer.
- Cloud Deployment.
- Container Deployment.

> **Note**
> Before installing MetaDefender Core, please refer to the **recommended minimum system configuration** listed below. The requirements are based on a standardized customer deployment and may require more or less resources to achieve the level of performance desired in your environment. Please note that the server specifications are built to allow a high volume daily processing. Please refer to Performance and Load Estimation for more details.

The recommendations below are for MetaDefender Core, API usage only. For any other use cases, please consult the user guide of the licensed products for accurate recommendations.

For certain use cases these might be adjusted and customized on specific needs and SLAs. We highly recommend to engage our ProServ team to assist in fine tuning MetaDefender and get the maximum performance out of your systems.

Microsoft Windows Deployments

[Windows] Supported Operating Systems

Only support 64-bit platforms.

- Windows 10, Windows 11 23H2, 11 IoT (22H2) only for End-User systems (e.g. Kiosk deployments)
  - Based on Microsoft License Terms, you're not allowed to use Windows Desktop as a server, therefore, MetaDefender Core can't be used on these systems for server use cases.
- Windows Server 2016, 2019, 2022

> **Disclaimer**
> End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

> **Plan for legacy OS support**
> OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:
>
> - Ubuntu 20.04 after **April 2025**.
> - Amazon Linux 2 after **June 2025**.
> - Windows 10 after **October 2025**.
>
> If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

[Windows] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 8	8	8 GB	16 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 12	16	16 GB	24 GB + 50 GB + 100 GB
MetaDefender Core 16	16	16 GB	32 GB + 50 GB + 100 GB
MetaDefender Core 20	32	16 GB	40 GB + 50 GB + 100 GB
MetaDefender Core MAX	32	32 GB	120 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core is installed.
- For a better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.
- SBOM processing requires more disk space, please allow at least 15GB free disk for the engine to work properly.

[Windows] Third Party Dependencies

- .NET framework 4.5 or above
- Microsoft Visual C++ Redistributable for Visual Studio 2015-2022 (version 14.38 or higher); x64 version.
- PostgreSQL: Only applicable when using a pre-installed PostgreSQL server running remotely.
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Some engines also have dependencies as described below:

Engine Name	Dependency
Deep CDR	Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 5.8.0 or above)
ESET	MetaDefender Core v4 temporary directory should have more than 200MB free disk space
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
Proactive DLP	• Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 2.0 or above) • .NET 6 dependencies • CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
Scrutiny	Microsoft Visual C++ 2015 - 2019 Redistributable x86 and x64
Systweak	.NET runtime 6.0, 7.0 or 8.0
Bkav Pro	Microsoft Visual C++ 2015 - 2022 Redistributable x64

[Windows] Installation Details

MetaDefender Core on Windows uses "C:\Program Files\OPSWAT" folder for storing resources or the installation directory.

MetaDefender will use its resources folder to store temp files as part of the analysis. **It's recommended to exclude this folder from real-time protection monitoring**. Knowledge base:

- How to set up exclusions on anti-malware software to prevent disruption
- Mitigating scan failure triggered by real-time protection service
- Can local AVs interrupt ongoing scans?

Linux Deployments

[Linux] Supported Operating Systems

Only support 64-bit platforms.

- Red Hat Enterprise Linux 8, 9
- Rocky Linux 9
- Debian 11, 12
- Ubuntu 20.04, 22.04
- Amazon Linux 2 is supported via Amazon Machine Image (AMI)

Plan for legacy OS support

OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:

- Ubuntu 20.04 after **April 2025**.
- Amazon Linux 2 after **June 2025**.
- Windows 10 after **October 2025**.

If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

For Hardended Linux OS

Some AV engines could be failed to initialize and run on MetaDefender Core (e.g. ESET, Kaspersky) due to execution permission required by them on `/var/tmp/ometascan` folder.

Please follow steps at Why have the ESET and Kaspersky scan engines failed to initialize on the hardened Linux OS? for remediation details.

Disclaimer

End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

[Linux] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 5	4	4 GB	10 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 10	8	8 GB	20 GB + 50 GB + 100 GB
MetaDefender Core MAX	16	16 GB	40 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core resides on.
- For better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.

[Linux] Third Party Dependencies

Dependencies list:

- openssl
- grep
- lib32stdc++6 (>= 4.5)
- libc6-i386 (>= 2.10)
- procps
- zlib1g
- libcurl3 (>= 7.19.7)
- libcurl4
- postgresql:
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Not all the above dependencies will need to be installed, it is dependent on different Unix distro and version

Some engines also have dependencies as described below:

Engine name	Dependency
Archive engine	• To extract password protected MS Office files, you need to install .NET 6 dependencies, libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • Rocky Linux 9.x/Red Hat 9.x: libnsl
Deep CDR	• Rocky Linux 9.x: libgomp, ncurses-compat-libs, libnsl • CentOS: libgomp, ncurses-compat-libs • Red Hat 8.x: libgomp, ncurses-compat-libs, libnsl • Red Hat 9.x: libgomp, ncurses-compat-libs, libnsl, compat-openssl11 • Ubuntu/Debian: libgomp1, libncurses5 • libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • .NET 6 dependencies
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
FileType engine	Rocky Linux 9.x/Red Hat 9.x: libnsl
Proactive DLP	libgdiplus, mscorefonts .NET 6 dependencies CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
RocketCyber	libgomp

Note: Some dependencies may need to be installed from external repositories, not from the OS default repositories.

[Linux] Installation Details

MetaDefender Core default installation path is using /var folder for storing resources:

- `/var/lib/ometascan` : installation folder with all its resources (database, DLP processed / CDR sanitized / quarantined file storage, engine package and definition updates).
- `/var/log/ometascan` : application logs.
- `/var/run/ometascan` : runtime folder for Core related processes.
- `/var/lib/ometascan` : all successful deployed engines data files.
- `/usr/lib/ometascan` : all shared libraries required by MetaDefender Core, bundled PostgreSQL server related binaries and libraries.
- `/var/tmp/ometascan/resources` : all temporary processing files for MetaDefender Core to process.

Storage Usage Information

Based on the configuration, MetaDefender Core will need additional disk space to store analysis/result data, quarantined files and temporary files:

	Data Storage
Analysis Reports (Processing History)	Approximately 4.5GB is required for every 1M analysis reports stored. • 100 GB is recommended as a baseline. Database storage requirements are affected by Data Retention settings, engine settings and scanning load.
Quarantined Files	Depends on the customers' dataset and configuration
Sanitized Files	Depends on the customers' dataset and configuration
Temporary processing files	Depends on the customers' dataset and configuration • A 1GB non-archive file requires 1GB free disk space at least. • A 1GB archive file requires (1 * 2)GB free disk space at least. • 50GB is a baseline and should be adjusted based on observation of the solution within the environment, with consideration to peak times and overall file makeup (e.g. complexity, size, archives).
Note	Options are available to control how large and/or complex files are handled to limit the amount of system resources required. See Workflow Configuration Template for more information.

Custom Engines

The recommendations above are specific for MetaDefender pre-packaged bundles.

However for additional Custom Engines, please review the Knowledge base to review additional requirements (if any) for the selected engine.

Browser Requirements for the MetaDefender Core Management Console

One of the following browsers is suggested to view the MetaDefender Core Management Console:

- Microsoft Edge
- Chrome
- Firefox
- Safari

Chrome, Firefox, Safari and Microsoft Edge browsers are tested with the latest available version at the time of release.

Network Bandwidth

In online environment where MetaDefender Core is configured to download engine update packages directly from OPSWAT update server source, then using gigabit network speed (125 MB/sec) is highly recommended to avoid potential network related issue while downloading files, or at least 100 Mbps (12.5 MB/sec) network speed.

Allowlisting URLs

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts':

- https://activation.dl.opswat.com - for license online activation
- https://update.dl.opswat.com - for engines and databases download

Note: *IP address-based allowlisting on your firewall might fail after some time since OPSWAT uses CDN (AWS Cloudfront) to faster delivery updates over the world, and IP address of edge servers might change over time.*

Depending on your location, connections with OPSWAT cloud-based servers be forwarded to specific AWS Cloudfront's edge locations, each is assigned with its own unique IP address ranges. You might also need to allow correct AWS Cloudfront's corresponding IP address ranges by referring to https://ip-ranges.amazonaws.com/ip-ranges.json

Warning

Even the OPSWAT update servers host updates for all of the available engines we support, sometimes custom engines might try to connect to their own cloud for updates, but this can be disabled in firewall and they will be updated just from OPSWAT cloud servers instead.

Differences Between MetaDefender Core v4 and v5

We are very excited to inform you about the release of version 5.0.0 of our MetaDefender Core!

The new version is designed to enhance the product performance, user experience, as well as to provide more flexibility, security, and scalability for our customers.

At the same time, we are also committed to keeping backward compatibility for whom running MetaDefender Core v4, and avoid the need for updating your client integration to move forward with MetaDefender Core v5.

v4 REST API Integration Supported
All public REST API endpoints available on MetaDefender Core v4 will be continously supported on v5.

v4 Product Major Features Supported
All product major features available on MetaDefender Core 4.21.2 (the latest one on v4) will be continously supported on v5.

Seamless Product Upgrade
Upgrading from MetaDefender Core v4 to v5 will be supported and covered with data migration without additional manual steps from users.

Licensing information will be also migrated.

Below are some highlights of the first v5 release, MetaDefender Core 5.0.0:

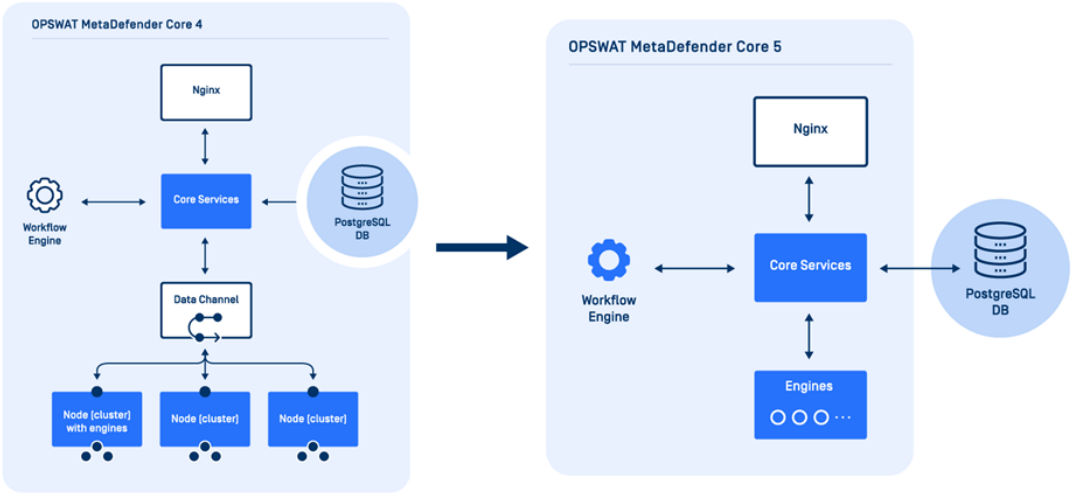
New product architecture

This version introduces a simpler product architecture that speeds up integration and automation, file processing flow, and immensely enhance the product performance and efficiency.

Earlier versions utilized MetaDefender Node concept, designed to support scaling of the scanning infrastructure by distributing scan requests among several different processing Nodes. Product licensing was based on number of Node instances. The benefit of having such a distributed infrastructure is that based on Node loads, Metadefender Core server can always choose the most appropriate Node to assign a new scan task to. However, having nodes in the v4 architecture caused more complexity and possible latency in the processing flow and entailed maintenance cost.

In this release, we retired multi-Node deployment and restructured MetaDefender Core's backend. This alteration significantly improves file processing speed and product efficiency, as well as eliminates extra cost for maintaining node services. Users now can simply install additional MetaDefender Core instances for scalability requirement,

as well as configure their own load balancer to coordinate tasks among multiple MetaDefender Core instances. Accordingly, product licensing is now based on number of the product instances.



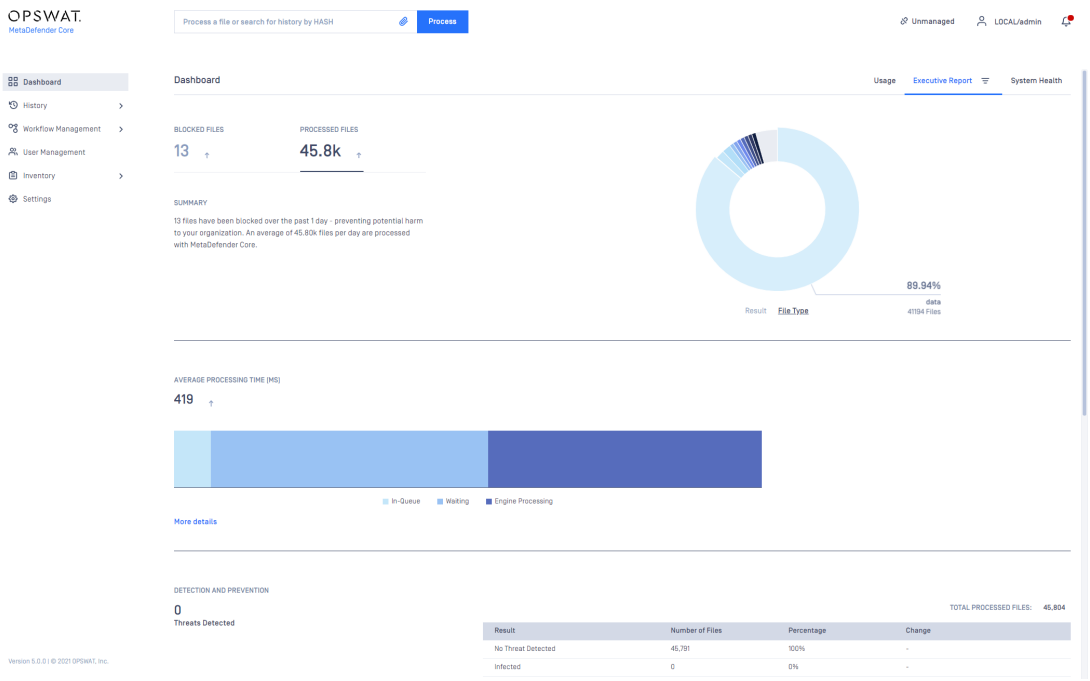
Please be informed that

- All existing administrative APIs related to "Node" on v4 will be kept as it is on v5 to ensure backward compatibility with existing client integrations.
- Applicable settings defined in Windows registry (Windows) or ometascan-node.conf config file (Linux) for Node will be migrated to Core. Details: MetaDefender Configuration

Brand-new product user interface

The second notable update in this version is a brand-new UI for the management console, which was designed with the focus on the quality and thoughtfulness of the user experience. The new administrative console UI provides an advanced set of Core features and observable processing status.

Comprehensive executive dashboard reporting offers dynamic, real-time and actionable threat insights. It also provides visibility of your security posture allowing you to paint a picture of security that aligns with the overall business. The report includes easy-to-read charts, graphs, and statistic explanation that help your security professionals reduce clutter, gather information quicker, spot threat activity early and act swiftly.



We're continuing to improve our product UI to provide users an excellent experience with the application and overall usability. Please expect more powerful UI features coming in next releases.

New compression archive engine for archive sanitization

In previous version, MetaDefender Core's archive engine implemented four functions including decryption, extraction, encryption and compression that with a high volume of files it takes time to process them all. For the purpose of enhancing archive handling performance, this release comes with a separated compression archive engine, which dedicatedly processes archive sanitization (compression and encryption). It is also created to facilitate maintenance and further feature development.

Archive sanitization feature now requires Deep CDR license

As to respect sanitization practice and ensure unknown threats prevention as designed, now starting on version 5, it's mandatory to have Deep CDR licensed for archive sanitization feature enablement on MetaDefender Core workflow configurations.

Archive handing related engines changed

The legacy "Archive" engine on MetaDefender Core v4 will be changed its name to "Archive Extraction" engine, even when you are still on MetaDefender Core v4.

A brand new "compression archive engine" is to dedicatedly handle archive sanitization in v5 (archive compression), that helps relevant archive handling performance improvement, easier for maintainance and further feature development.

Notes when you are still on MetaDefender Core v4 (not upgraded yet)

The legacy "Archive" engine (now name changed to "Archive Extraction" engine) on MetaDefender Core v4.x will still cover both extraction and compression archive tasks as usual.

Even though your MetaDefender Core v4.x might now have a new "Archive Compression" engine auto downloaded, but that engine cannot be deployed (marked as "permanently failed" in relevant engine status check API). This behavior is expected, and not causing any impact to MetaDefender Core processing flow, or its performance though.

- Just in case your client integration has a check for engine status before triggering scan request, then you might want to disable "Archive Compression" engine status on MetaDefender Core v4 using REST API. Detailed instruction: Archive Compression Engine Permanently Failed on MetaDefender Core v4
- When you later upgrade to MetaDefender Core v5 from v4, then you might need to manually re-enable "Archive Compression" engine to make it active and running on MetaDefender Core v5.

Notes for licensing

For offline deployment (MetaDefender Core is not connected to the Internet directly for online update), you will need to re-activate again on OPSWAT portal to retrieve latest engine list in .yml file including "Archive Compression" engine if you are currently licensed with Deep CDR.

Meanwhile, for online deployment, this manual procedure is not required since MetaDefender Core will cover that itself.

Import & export configurations

Not only workflow settings like in previous versions, but also all other general settings (update, security, data retention, server, hashes to skip) and most importantly, user settings, will be able to export and import on MetaDefender Core version 5.

Learn more at Import/Export configuration

Backward compatibility supported

The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

Windows application event logging disabled by default

Applicable only to Windows OS based MetaDefender Core.

Help avoid duplicated logging with separate file option, and improve overall processing performance. Of course, when being still valid to you, you are still supported to enable it back.

CSRF token header renamed

CSRF token originally created for security strengthening and being used by MetaDefender Core management console UI since version 4.21.0 via header `oms_csrf_token`

To get it compliant with some load balancers which do not accept underscore in header name by default, and save you from additional load balancer configuration step, we have now changed the header name to `oms-csrf-token`

This change does not impact to any REST API based client integration.

More comprehensive and accurate error messages on REST APIs

Impacting on following REST API endpoints:

- `PUT /admin/config/session`
- `PUT /admin/config/update`
- `PUT /admin/config/scan`
- `PUT /admin/config/skip`
- `PUT /admin/config/auditlog`
- `PUT /admin/config/webhook`
- `PUT /admin/config/rule`

Upload time and queue time logic updated in JSON scan response

Impacting on following REST API endpoints:

- `GET /file/{data_id}`
- `GET /hash/{md5/sha1/sha256}`
- `GET /archive/{data_id}`
- `POST /file/{async}`

Added new field:

- `receive_data_timestamp`: The timestamp when upload progress started (first byte received) (in milliseconds)
- `upload_time`: Total time elapsed for upload process (in milliseconds)

Modified existing field (new logic):

> **Backward compatibility supported**
> The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

- `upload_timestamp`: The timestamp when upload progress finished (all bytes received) (in milliseconds)
- `queue_time`: Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue time = start_time - upload_ timestamp) (in milliseconds)

Release version 5.0.0 also comes with variety of outstanding features and improvements.

For detailed information about our latest release please view our Release notes

Installation

Starting with MetaDefender Core 4.20.0, the centralized/shared database deployment model is supported (all MetaDefender Core instances share the same database), along with traditional standalone database mode (every MetaDefender Core instance has its own database to manage).

- Standalone mode
- Shared Database Mode

Installation

1. Download the package of your choice from My OPSWAT
2. Install the package on your computer via the Command Line or via the Wizard
3. Open a web browser and point to http://<server name or IP>:<port>
    - The default port is **8008**
    - In case of problem check Inaccessible Management Console page
4. Complete the required steps of the basic configuration wizard
5. You must activate this deployment to use its features

Installation notes

- If the MetaDefender Core package **dependencies are not installed** on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.
- During installation the databases might need to be upgraded. This could take noticeable time depending on database size (eg.: length of scan history).

Standalone mode

Wizard Installation

The Install Wizard is only for the Windows installer (.msi file).

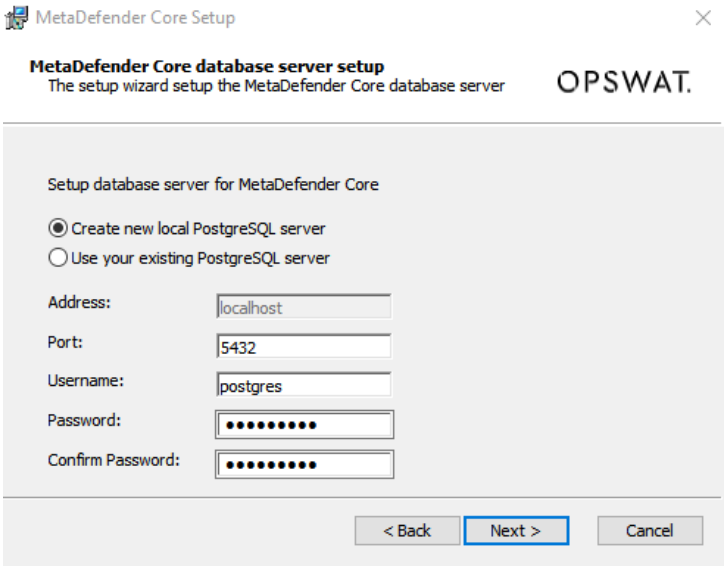
Since MetaDefender Core 4.19.0, we have switched to using PostgreSQL DBMS which needs a dedicated database server. MetaDefender Core supports two modes to setup PostgreSQL server:

Local (Bundle) PostgreSQL server

MetaDefender Core will install a new PostgreSQL server locally in the same box with the product.

> **PostgreSQL user credentials**
> Unicode characters are NOT supported for "Username" and "Password"
>
> "Password" does not accept # (hash) as starting character (e.g. #ABC)

> **The ignition file**
> If the ignition file exist in the designated location, then MetaDefender Core will use the details defined in the ignition file, rather than what is passed via the installation wizard.
>
> More info about the ignition file based installation: Command Line Installation

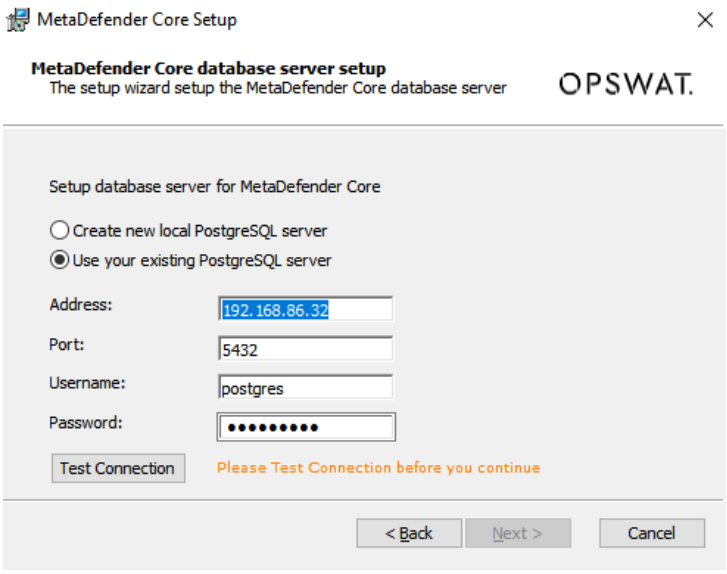


Remote (Existing) PostgreSQL server

MetaDefender Core will connect to remote pre-installed PostgreSQL running remotely.

> **PostgreSQL connection configuration**
> "Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

> **PostgreSQL user privilege requirements**
> MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.
>
> Details: PostgreSQL User Privilege Requirements



Command Line Installation

Ignition File

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Make sure to create the ignition file, if it doesn't exists before you go ahead and install MetaDefender Core via command line.

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

MetaDefender Core supports two modes to setup PostgreSQL server:

1) Local PostgreSQL server

- MetaDefender Core will install a new PostgreSQL server locally, next to the product.
  - A sample iginition file for PostgreSQL server information (***Only non-Unicode characters supported for "user" and "password"__***):

ometascan.conf bash

```
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
```

2) Remote PostgreSQL server

> **Info**
>
> **PostgreSQL version 12 (at least 12.3) or version 14 is required.**
>
> According to PostgreSQL notice, version 12 will stop receiving fixes on November 14, 2024, so we suggest that you use version 14.

MetaDefender Core will connect to a pre-installed PostgreSQL server running remotely and will setup its database.

Make sure to follow steps at Open Connection On PostgreSQL Server first, so that your remote existing PostgreSQL server to accept upcoming connection from MetaDefender Core.

A sample ignition file for PostgreSQL server information:

ometascan.conf bash

```
[dbserver]
type=remote
host=192.168.86.32
port=5432
user=existingserver_admin_user
password=existing_server_pass
```

After installing MetaDefender Core successfully, you may want to remove all credentials info created for [dbserver] section in ignition file for security reason.

> **Warning**
> If the MetaDefender Core package dependencies are not installed on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.

Debian package (.deb)

bash

```
sudo dpkg -i <filename> || sudo apt-get install -f
```

On Red Hat Enterprise 7.x Linux / CentOS package (.rpm)

bash

```
sudo yum install <filename>
```

For systems which enabled GPD check flag:

bash

```
sudo yum install --nogpgcheck <filename>
```

Red Hat Enterprise Linux 8.x package (.rpm)

Install the latest PostgreSQL repository RPM:

> **Note**
> Due to the shorter support cycle on Red Hat Enterprise Linux, all supported versions of PostgreSQL are not available on this platform. The PostgreSQL Yum Repository will integrate with the normal systems and patch management, and provide automatic updates for all supported versions of PostgreSQL throughout the supported lifetime of PostgreSQL. Reference: https://www.postgresql.org/download/linux/redhat/

bash

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

In case installing a remote PostgreSQL on either the same MetaDefender Core server or another server, you need to install the following additional package. PostgreSQL version 12.3 or newer is recommended.

bash

```
sudo dnf install -y postgresql12-contrib
```

Install MetaDefender Core:

bash

```
sudo dnf install -y <filename>
```

Windows package (.msi)

On Windows systems it is possible to install the product by running the corresponding .msi file.

From command line interface it is also possible to install the product by executing

bash

```
msiexec /qn /i <msi file name> <option key>=<option value>
```

where the possible keys and their default values are the following:

Key	Default Value	Description
INSTALLFOLDER	\Program Files\OPSWAT\MetaDefender Core	Customize installation folder for product Example: `INSTALLFOLDER="D:\Products"`
RESTADDRESS	*	REST interface binding IPv4 or IPv6 address ('*' means that service listens on all IPv4 and IPv6 interfaces)
RESTPORT	8008	REST interface binding port
REPORT_ENGINE_ISSUE	true	Enable reporting of engine issue count. (possible values: "true" or "false").

Note

Reporting of engine issue count

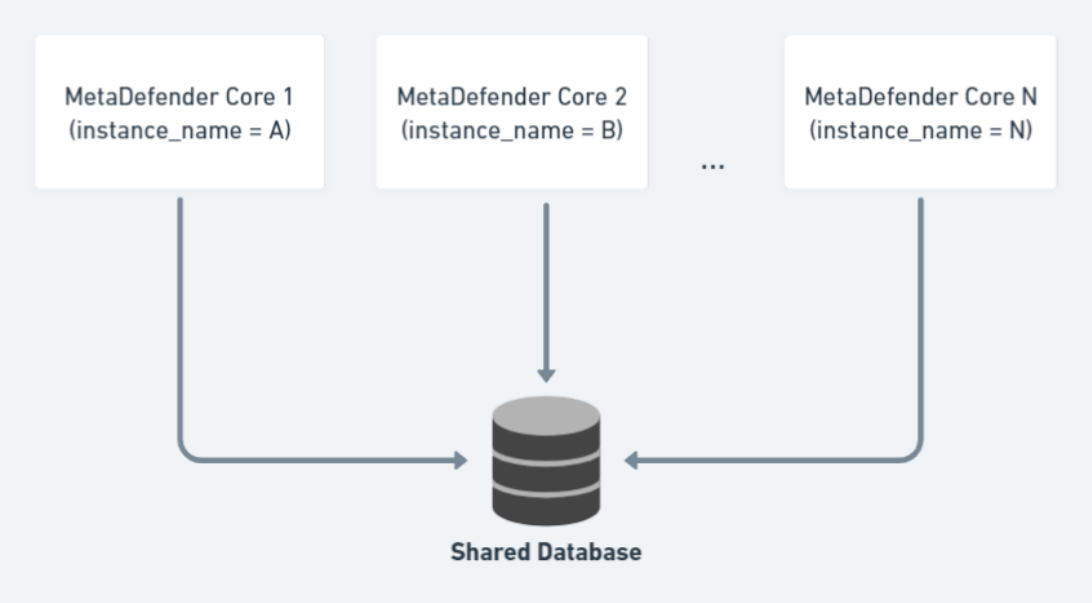
If reporting of engine issue count is enabled, MetaDefender Core server will send only the **number** of initialization errors and **number** of unexpected stops for the specific db/engine version. This information is sent over a HTTPS channel when the product downloads the latest package descriptors. This information is used for early detection of any specific 3rd party engine quality issues.

For details on using msiexec please consult Windows installer documentation.

Shared Database Mode

Starting with MetaDefender 4.20.0, you can use a centralized database deployment model, where all MetaDefender Core instances can share the same external database.

That helps auto-sync all product configurations (**except Single-Sign-On** that requires unique settings for each Core), processing history, live/on-going scan result query, audit, statistics data.



This article demonstrates how to install and setup MetaDefender Core 4.20.0 or newer with shared database mode.

Important notes & current limitations:

Currently not supported

- Does not support Central Management use-case with shared DB mode.
- Does not support to switch from shared database mode back to standalone database mode.
- Does not support to change instance_name of MetaDefender Core in shared database deployment model after installation.
- Does not support upgrading older MetaDefender Core from non-persistent mode to using the shared database mode.
- Does not support upgrading older MetaDefender Core from standalone mode to using the shared database mode.
- Does not support to upgrade multiple MetaDefender Core instances at once in shared database mode
    - Stop all Core instances' scanning service before performing the upgrade.
    - Upgrade each Core server one by one instead.

- For each Core, when the upgrade finished, then you can start using that Core for scanning without waiting for the other Core instances' upgrade to be done.

> **Disclaimer**
> MetaDefender Core does not support mode switching.

Requirements for shared database setup:

- The central remote PostgreSQL server (for shared database) could be utilized with a built-in (native) PostgreSQL database server installed with one of the MetaDefender Core instances in the network. Or using a dedicated existing PostgreSQL database server (see detailed requirements for PostgreSQL version at Recommended System Configuration).
- Before setting up, make sure to follow steps at Open Connection On PostgreSQL Server so that all MetaDefender Core instances could connect to the central remote PostgreSQL server.
- MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core, details: PostgreSQL User Privilege Requirements
- All MetaDefender Core instances must be installed on the same Operating System (Windows / Linux).
- The first MetaDefender Core instance connects to the shared database must be ready first (license activated, users/groups/roles created) before setting up other MetaDefender Core instances to connect and use that shared database.

Engine update:

- If engine updates is configured to be via Folder source, all MetaDefender Core instances must be able to access to the configured update folder, and the option DELETE FILES AFTER IMPORT will be unchecked and prevented for modification.

Authentication:

- Single-Sign-On (SSO) authentication settings (if using) cannot be shared among MetaDefender Core instances because SSO requires an unique login URL for every different Core.
- If an user is locked for login on a certain MetaDefender Core instance(e.g. exceeded the number of failed login times), that user is not prevented for logging in on any other instances.
- If an user is locked for sign-in on a certain MetaDefender Core instance (e.g. exceeded the number of failed login times), if the admin releases lockout on that user on that MetaDefender Core instance, that user is still not able to login on other MetaDefender Core instances.

Sanitized / DLP / Quarantined location changed after MetaDefender Core install:

Windows:

Shared storage path will be stored in 3 strings under key global in registry: quarantinepath, dlppath, sanitizepath as below

(Default)	REG_SZ	(value not set)
dbmode	REG_SZ	4
dlppath	REG_SZ	\\10.40.50.100\test
instance_name	REG_SZ	481rvj
quarantinepath	REG_SZ	\\10.40.50.100\test
restaddress	REG_SZ	*
restport	REG_SZ	8008
sanitizepath	REG_SZ	\\10.40.50.100\test

You can set either 1, 2 or 3 different paths for the 3 settings above (means that quarantined, DLP and sanitized files will be stored in same or different storages corresponding). In case shared storage is not set in installation, values of 3 settings above will be empty.

Restart MetaDefender Core service after changed.

Linux:

Open file `/etc/ometascan/ometascan.conf` and set shared storage path for 3 keys quarantinepath, dlppath, sanitizepath under section [global]

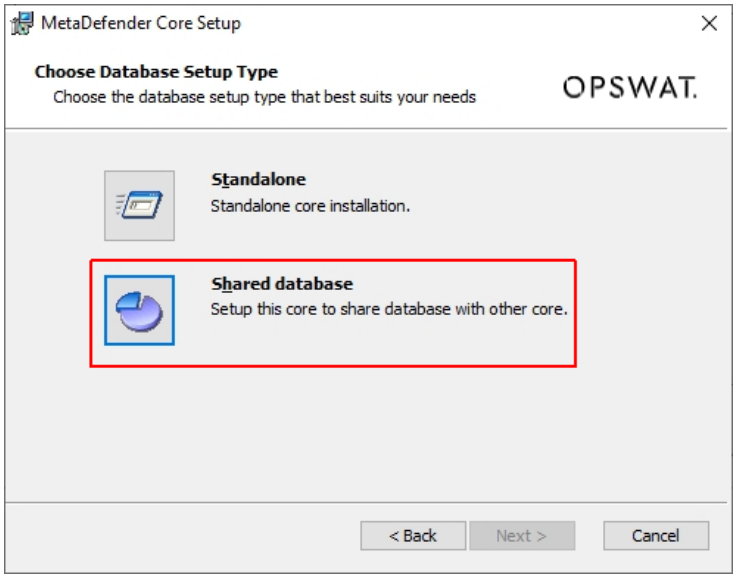
Restart ometascan service after changed.

Wizard Installation (Windows Only)

> **Info**
> Only applicable to MetaDefender Core 4.20.0 or newer using wizard on Windows OS.

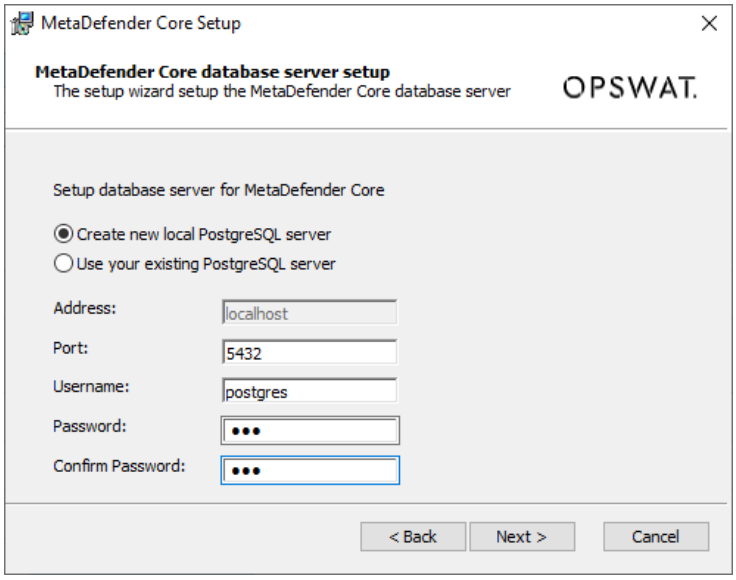
Clean install scenario

1. **Choose Shared database option:**

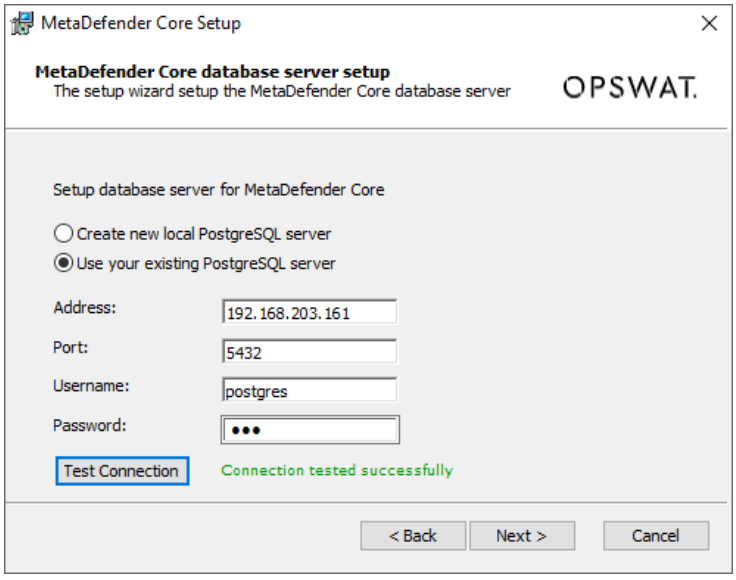


2. **Setup database server for MetaDefender Core:**

You can choose to create a native / built-in PostgreSQL server and setup the shared database on (then other MetaDefender Core instances will want to connect to that PostgreSQL server afterwards)



Or choose to connect to a pre-setup existing PostgreSQL server:



PostgreSQL connection configuration

"Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

PostgreSQL user privilege requirements

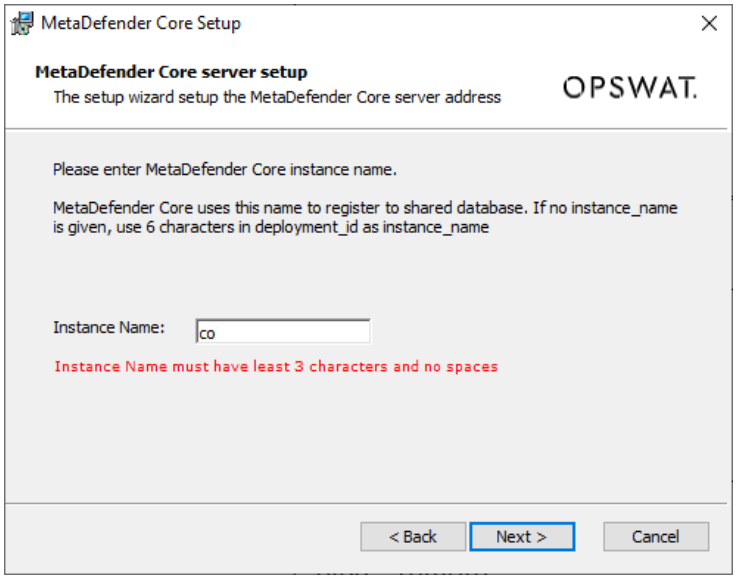
MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.

Details: PostgreSQL User Privilege Requirements

3. **Create an unique instance name for that MetaDefender Core (different from other MetaDefender Core instances):**

Instance name must contain at least 3 characters and no white space.

If you choose to leave it blank, which means it will use the default instance name with 6 characters extracted from that MetaDefender Core deployment ID.

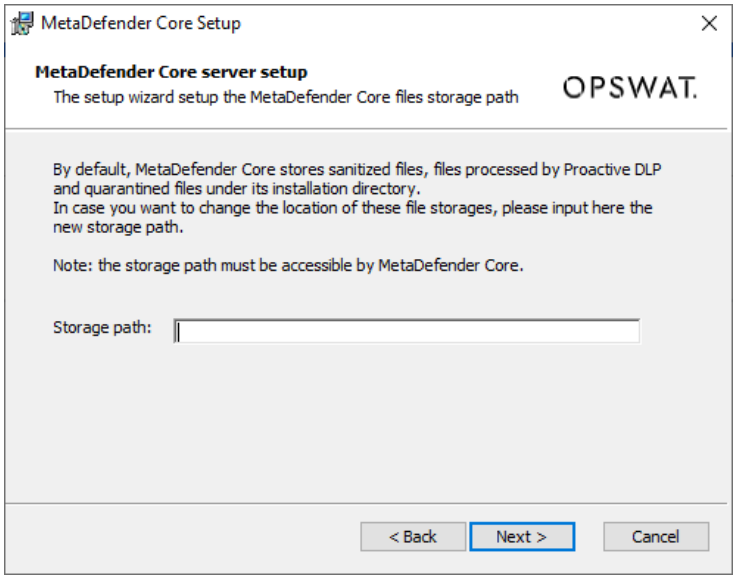


4. **Input the file storage path (sanitized, DLP, quarantined files)**

In the shared database model, all MetaDefender Core instances could use the same folder path to store all their own sanitized, DLP, quarantined files after processed.

The default location is: ***<MetaDefender Core installation folder>\data\ {sanitized|dlp|quarantine}***

If you wish to use that default location, just leave the input box blank and press Next.



Otherwise, you can specify a different location of sanitized/dlp/quarantined files. It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core.

For example:

- C:\new_folder
- \192.168.1.1\new_folder

Command Line Installation

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Clean install scenario

1. Create ignition file if not existed

- Windows: **C:\OPSWAT\ometascan.conf**
- Linux: ***/etc/opswat/ometascan.conf***

An example of ignition file:

ometascan.conf bash

```
eula=true

[global]
dbmode=4
instance_name=MetaDefenderCoreA
storage_path=D:\path\

[user]
name=admin
password=admin
email=admin@local

[dbserver]
type=remote
host=192.168.1.1
port=5432
user=postgres
password=your_own_password
```

Field	Required	Mapping to wizard UI	Possible values
global/dbmode	optional default is 1 (standalone)		1 - Standalone 2 - Central Hub deployment 3 - Non-persistent **4 - Shared**
global/instance_name	optional default is 6 characters of deployment ID		Unicode and number At least 3 characters No whitespace
global/storage_path	optional default is empty		Path to storages (DLP, quarantined, sanitized files) It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core. For example: • C:\folder • \192.168.1.1\folder

2. Run installers using command line to install

Please refer to Command Line Installation for installation command.

Deployment automation support

Note

An Ignition file could be utilized for either / both of following use-cases:

1. Pre-define PostgreSQL server information for MetaDefender Core to connect and setup database: only required on Linux installation, or Windows with command line installation. Learn more details at:
   a. Standalone Mode > Command Line Installation (applicable on version 4.19.0 or newer).
   b. Shared Database Mode > Command Line Installation (only applicable on version 4.20.0 or newer).
2. Automating to accept EULA, create a default local admin user and import MetaDefender Core configurations upon installed: Those steps could be automated using ignition file described in this page.

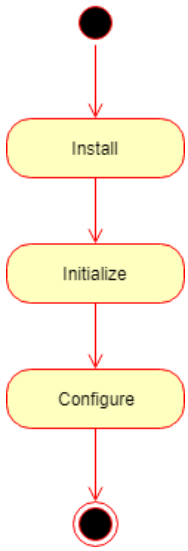
Ignition file location to create:

- **Windows**: `C:\OPSWAT\ometascan.conf`
- **Linux**: `/etc/opswat/ometascan.conf`

The product supports fully automated deployment. It means that it can be installed and configured with no human interaction.

The automated deployment can be split to three steps on a high level:

1. Installation
2. Initialization
3. Configuration



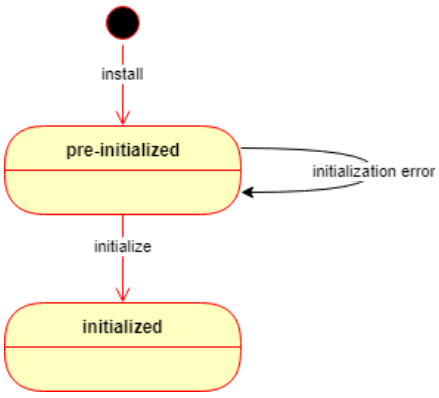
Installation

To automate the installation, install the product from the command line and provide the installation-time options as parameters to the installer. For further details see Command Line Installation

After the installation is complete, the product starts up and waits in a pre-initialized status. The product may be initialized in two ways:

1. Manually using the Wizard Setup or
2. Automatically using an *ignition file* (see below).

> **Warning**
> If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried after fixing the problem (e.g. placing the ignition file in its lookup location) and restarting the OPSWAT MetaDefender Core service.
>
> Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

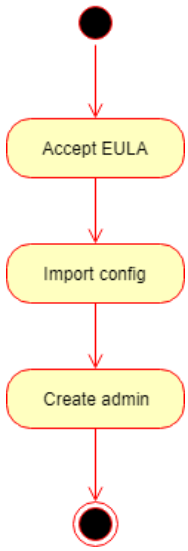


Initialization

Initialization is the process of bringing the product to an operable status.

Basically the initialization consists of the following steps:

1. Accept the End User License Agreement (EULA),
2. Import product configuration and
3. Create the first administrator user account.



Ignition file

The initialization process can be configured in a file called the *ignition file*.

> **Warning**
> In Linux, the ignition file and the folder /etc/opswat should have the permission 755

Ignition file location

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

The ignition file must be in conf format.

ometascan.conf bash

```
eula=true
[user]
name=admin
password=admin
email=admin@local
apikey=44c96cbb4e4e6c0a1d16bcb3c3da24def317
[config]
import=config_package.zip
import_password=123
[security]
session_cookie_samesite=Lax
[internal]
skip_startup_pages=true
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
private_username=my_internal_db_user
private_password=mypassword
[ocm]
ocm_url=https://my.opswat.com
reg_code=XXXXXXXX
```

Ignition file fields

The ignition file accepts the following fields:

Section	Key	Required	Accepted Values	Description
	eula	Optional (mandatory when having [user] section)	true false (default)	Whether to accept the End User License Agreement. This key must be set at the top of ignition file. This key must be set to "true" to accept the EULA. Any other value will cause the initialization to fail. When having [user] section defined, then this key must be set.
[user]		Optional		Initial administrator user account properties. The Administrator role is granted to the account.
	name	Optional	string	User name for the initial administrator user account.
	password	Optional	string	Password for the initial administrator user account. WARNING! Clear text password The password in this configuration file must be stored in its clear-text format and as so it may be visible for unauthorized parties.
	email	Optional	string	E-mail address for the initial administrator user account.
	apikey	Optional	string	API key for the initial administrator user account.
[config]		Optional		
	import	Optional	string	Path to a file in json/zip format that contains a previously exported configuration to be imported. Zip-format file contains a full configuration including user management settings, so it will overwrite [user] section.
	import_password	Optional	string	Password to decrypt .zip configuration package.
[security]		Optional		

Section	Key	Required	Accepted Values	Description
	session_cookie_samesite	Optional	Strict Lax (default) None	Declare if session cookie should be restricted to a first-party or same-site context.
[internal]		Optional		
	skip_startup_pages	Optional	true false (default)	Skip all welcome and other startup pages (e.g. upgrade result) when installing / upgrading MetaDefender, and going straight to the product index default page.
[dbserver]		Optional		
	type	Mandatory when having [dbserver] section	local remote	Where the PostgreSQL database server should physically locate. "remote" means PostgreSQL server and MetaDefender Core are not in the same machine.
	host	Mandatory when having [dbserver] section	string	IP address / domain name of the server where PostgreSQL server locates. "localhost" can be used when applicable.
	port	Mandatory when having [dbserver] section	number	Port of PostgreSQL server is listening for connections from clients (i.e. MetaDefender Core).
	user	Mandatory when having [dbserver] section	string	PostgreSQL server's user. SUPERUSER privilege is required for MetaDefender Core to setup its database and extensions for the first time. **Only non-Unicode characters supported.**

Section	Key	Required	Accepted Values	Description
	password	Mandatory when having [dbserver] section	string	PostgreSQL server's user credentials. **Only non-Unicode characters supported.**
	private_username	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
	private_password	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
[ocm]		Optional		
	ocm_url	Optional	string	URL to My OPSWAT server.
	reg_code	Optional	string	A registration code to enroll to My OPSWAT server.

Detailed initialization process

1) After the product has been started, it looks for the ignition file in the designated location.

Windows: C:\OPSWAT\ometascan.conf ** Linux: /etc/opswat/ometascan.conf

** 2) If an ignition file is found, then

2.1) It gets validated, and if it is valid, then based on the information found in the ignition file:

- The EULA is accepted,
- The configuration is imported,
- The administrator account is created.
- If any of the above steps fails, then the error is logged, and the initialization gets terminated.

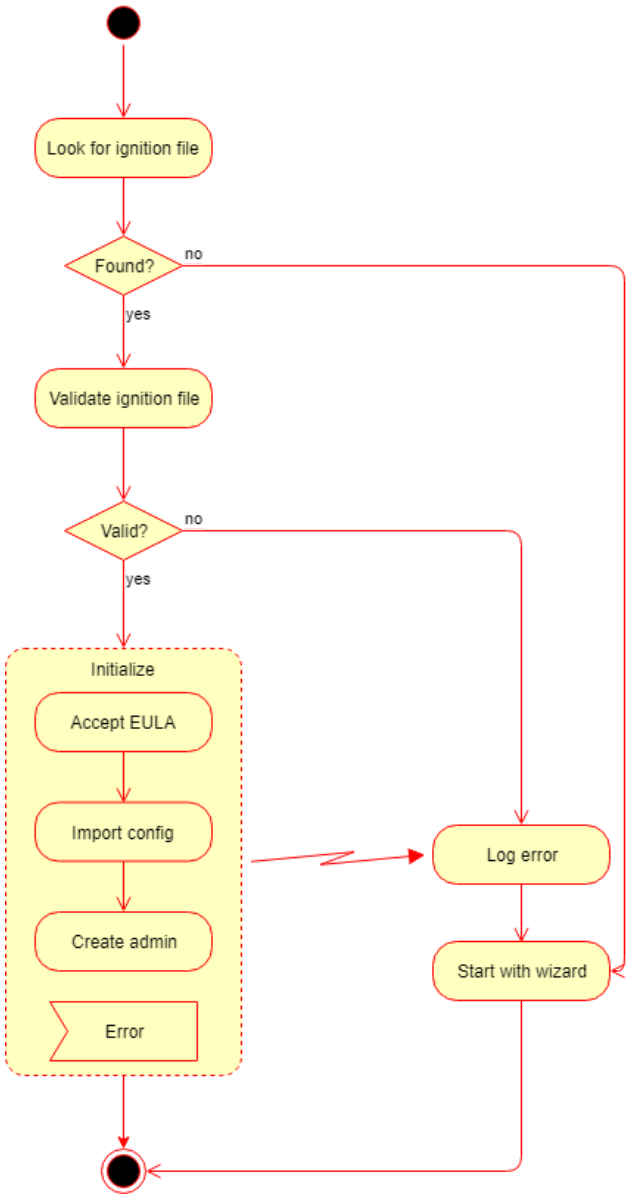
> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

2.2) If it is not valid, then the error is logged, and the initialization gets terminated.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

3) If there is no ignition file, then no initialization is performed.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.



Warning

If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried fixing the problem (e.g. placing the ignition file to its lookup location) and restarting the OPSWAT MetaDefender Core service.

Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

Configuration

After the initialization is complete, the product is ready with the default and the imported configuration.

This configuration can be later changed calling the configuration API functions. For further details about the API see Configuration related APIs API

CIS Level 1 Guidelines

Support Red Hat Enterprise Linux 9
For more details about Center for Internet Security (CIS) please refer to this document

Instruction steps

I. Install OpenSCAP

bash bash

```
yum install openscap-scanner scap-security-guide
```

II. Generate a result file and a HTML report using OpenSCAP scanner tool

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

III. Remediation of CIS Level 1 issues

Generate a remediation script based on the `ssg-rhel9-ds.xml` file:

bash

```
oscap xccdf generate fix --profile
xccdf_org.ssgproject.content_profile_cis_server_l1 --fix-type bash --output
remediations.sh /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

And execute remediation script.

bash

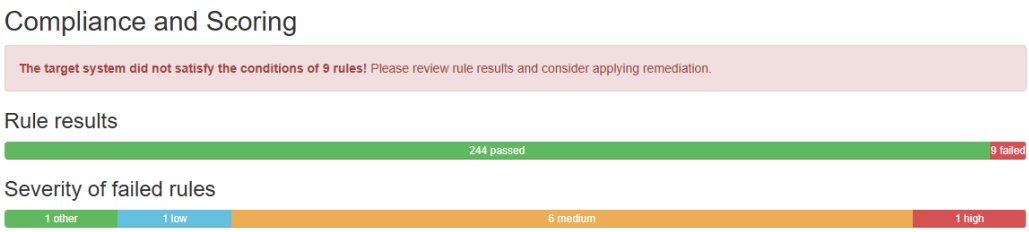
```
./remediations.sh
```

IV. Review the results after remediation

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

Example:



Notes

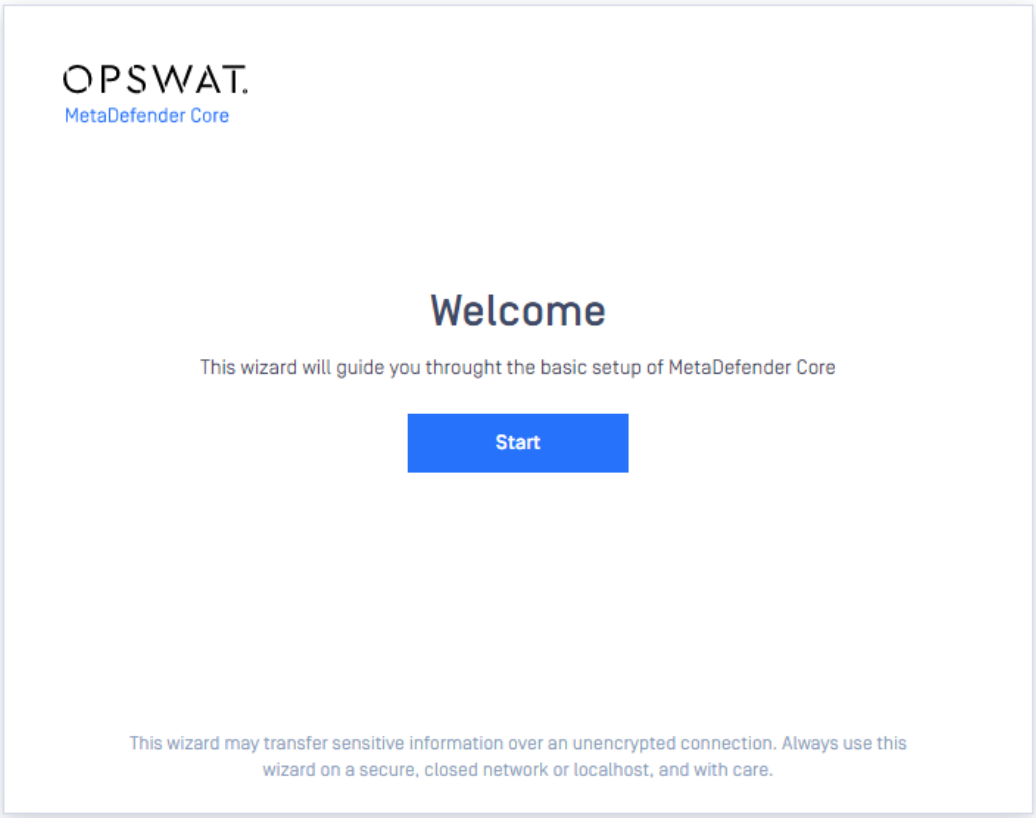
- CIS Level 1 requires `/tmp` folder to be mounted in a separate partition. Please ensure that that new partition have enough disk space for MetaDefender Core to run.
- CIS Level 1 requires that "Ensure No World-Writable Files Exist".
    - For now, when freshly installing MetaDefender Core, all its binary files meet the requirement.
    - When installing/updating engines, some engines might create additional files for its operation, and it might violate this requirement. In this case, you need to again execute the remediation script in the step III.

Wizard Setup

Introduction

Access the MetaDefender Core management console for the first time (i.e. http://localhost:8008), you are to complete a quick walkthrough for wizard setup in order to be able to use the product.

The administrative management console will be available only upon you have successfully finished this wizard stage.

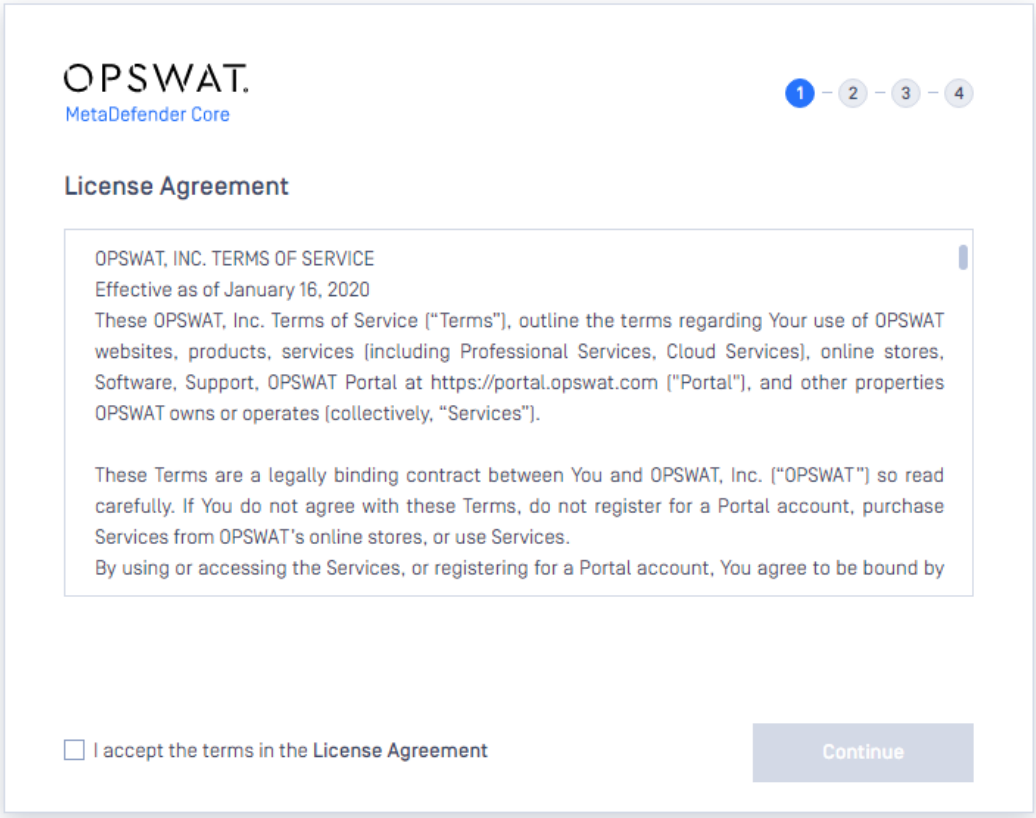


Warning

This wizard may transfer sensitive information over an unencrypted connection. Always use this wizard on a secure, closed network or localhost, and with care.

Basic configuration steps

End-User License Agreement



In the first page you can find the End-User License Agreement. You have to accept the terms before moving on.

Please read through the EULA (https://docs.opswat.com/policies/privacy-policy/portal-privacy-policy) carefully and if you agree with it, check `I accept the terms in the **License Agreement**` and click `Continue` to proceed.

Local Admin User Setup

The next step is to set up a default local administrator account.

This account will be the first one being able to access the administrative management console, and to create accounts for other users. You have to fill out all fields in this page to be able to move forward. When you are done, click NEXT to continue.

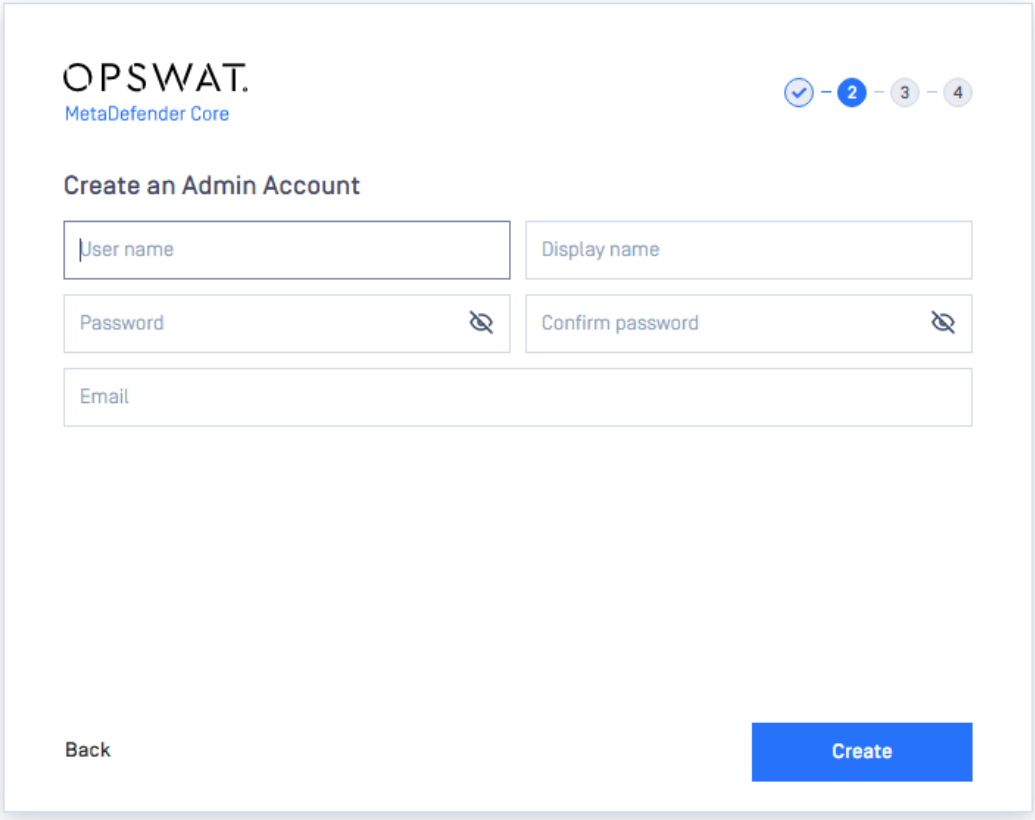
Info - User Directory

The administrator account, that is created via the basic configuration wizard, is always added to the LOCAL user directory as a member.

Learn more about user directory: User directories

The following information is required for the administrator account:

ACCOUNT NAME	The unique name of the account that is used at the time of login and in log messages for accountability.
ACCOUNT DISPLAY NAME	Name of the person bound to this account. This name (appended to the name of the account's user directory) is displayed in the top right corner of the administrative management console.
PASSWORD	Password of the user bound to this account that is used at the time of login. Passwords sent clear-text As long as TLS is not configured for the basic configuration wizard, passwords are sent clear-text over the network and may be disclosed to unauthorized parties. As a mitigation action: 1. Either use the wizard on *localhost* or on a direct network connection, or 2. Enabling HTTPS as soon as possible and change the password immediately if it has already been set.
EMAIL	Email address of the person bound to this account.



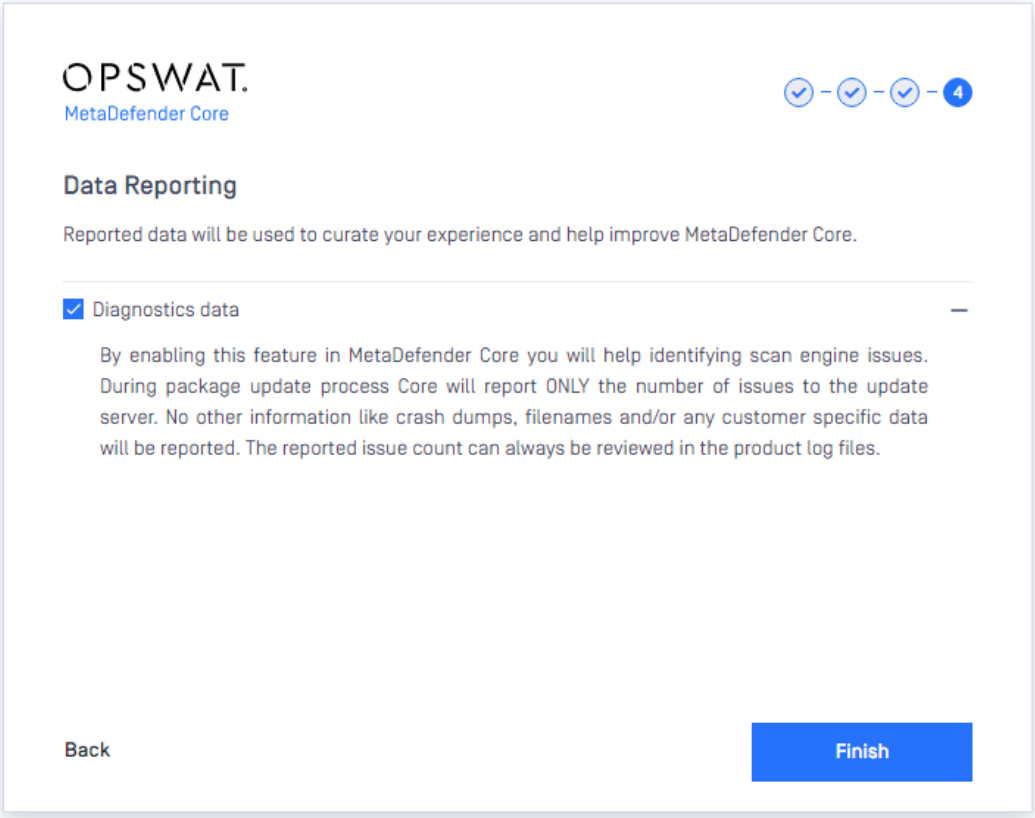
License activation

For license activation details see License Activation

Data Reporting

Enabling this option to allow MetaDefender Core collect its own engine issues and report back to OPSWAT for further analysis.

MetaDefender will not collect any other information.

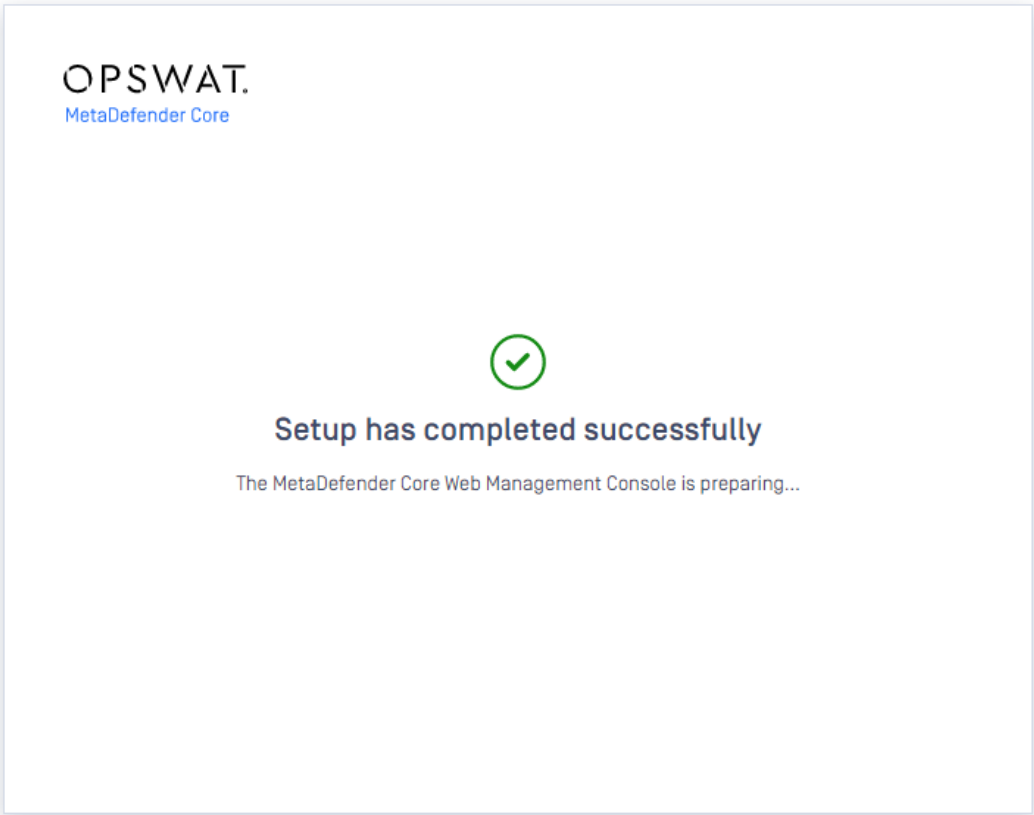


Wizard completion

After you have completed every steps you are ready to finish the wizard and start using the product. Click the `Fishish` button to complete the wizard.

> **Warning**
> The product's service will be restarted and the browser will be redirected to the administrative management console. This could take several seconds.

You can login to the Web Management Console with the administrator user that have just been created in the previous steps.



Setup HTTPS

Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network. Websites, like the Web Management Console, are able to use TLS to secure all communications between their servers and web browsers.

The TLS protocol aims primarily to provide confidentiality (privacy) and data integrity between two communicating computer applications.

Warning

No TLS for the wizard

By default, TLS is not enabled for the basic configuration wizard. As a consequence sessions between the wizard's backend and the browser may be insecure.

Info

Performing the same steps as for the Web Management Console, it is possible to set up TLS for the basic configuration wizard. Remember completing the TLS setup before launching the wizard.

For instructions to set up TLS see Enabling HTTPS

License Activation

##

To activate your installation go to the **Settings > License** menu in the Web Management Console. If you have no valid license, you will only see your installation's **deployment ID**. You will also see a warning in the Web Management Console header.

After successful activation, the product will start downloading the latest available scan engines and malware databases. You can follow the status of the scan engine installation on the **Inventory > Modules** page.

When all the licensed engines are installed, you can start using the MetaDefender Core to process files - File Processing.

Deployment ID

Each MetaDefender Core instance represented by an unique system fingerprint called `Deployment ID`

MetaDefender Core licensing is purely based on deployment ID activated. Therefore, everytime your MetaDefender Core is activated with a different deployment ID, then it will be captured as a different activation record accumulated into your total activation times allowed.

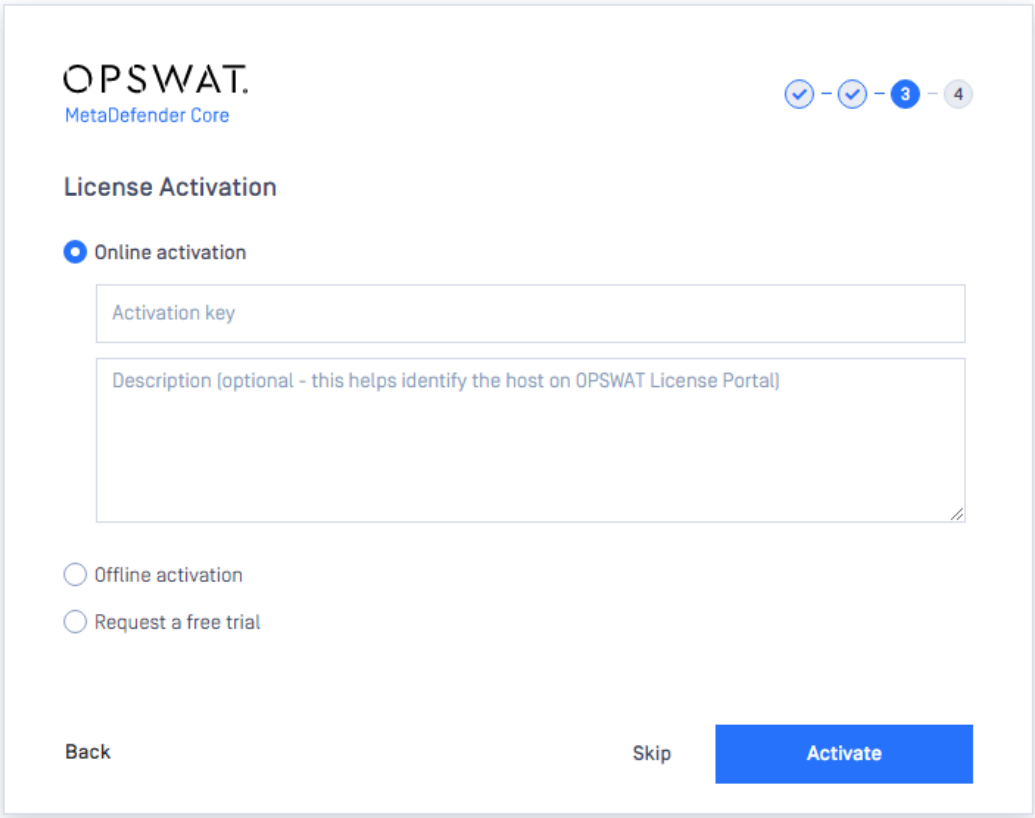
> **Notice**
> Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you alter the system where MetaDefender Core relies on (e.g. network card, OS hostname).
>
> Deployment ID is an UUID version 4 random based string. Learn more at: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you update the system where MetaDefender Core relies on.

Activation Modes

There are three different options for the product activation:

- Online activation: MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.
- Offline activation: Requiring you to upload a license file (.yml). That license file could be retrieved by activating your license via OPSWAT Portal. Follow the displayed instructions for details.
- Request trial key online: For evaluation purpose, you can receive an trial Activation key via email. Follow the displayed instructions for details.



Online activation

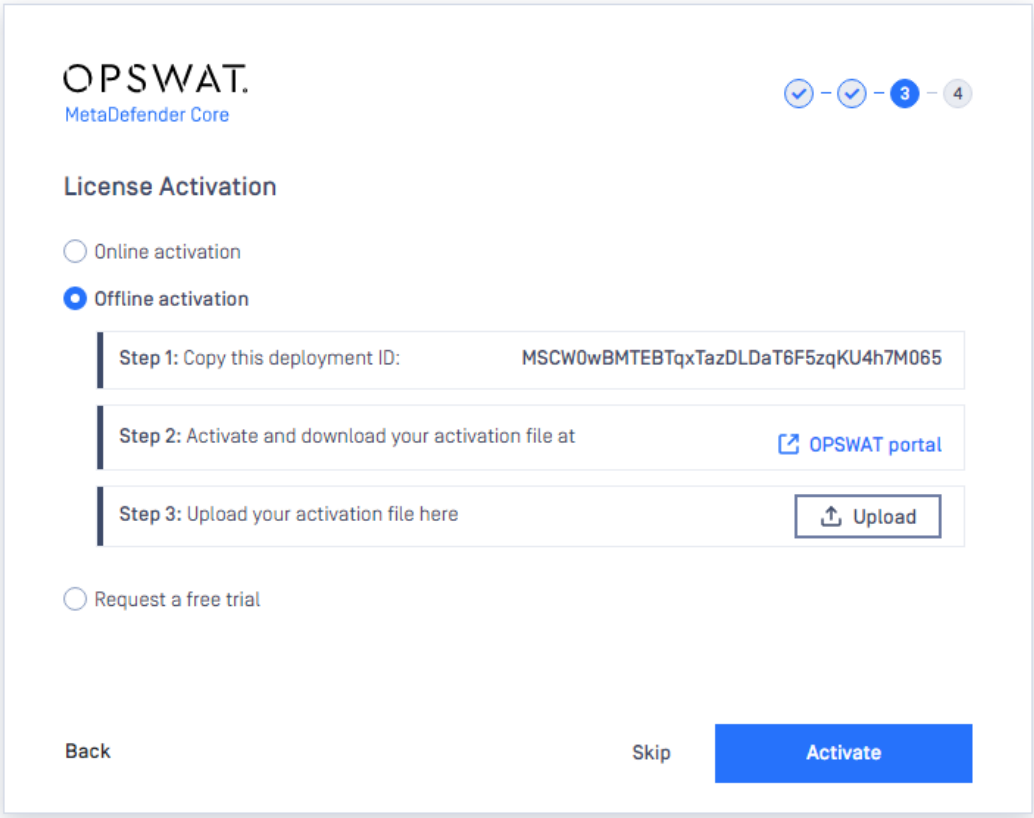
MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.

Required to input your Activation License key, and optional for description (but we recommend this for troubleshooting and auditing purpose if needed).

> **Warning**
> If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts' for license activation:
>
> - https://activation.dl.opswat.com Note: *IP address-based whitelisting on your firewall might fail after some time since OPSWAT uses CDN (Content Delivery Network) to faster delivery updates over the world, and IP address of edge servers might change over time.*

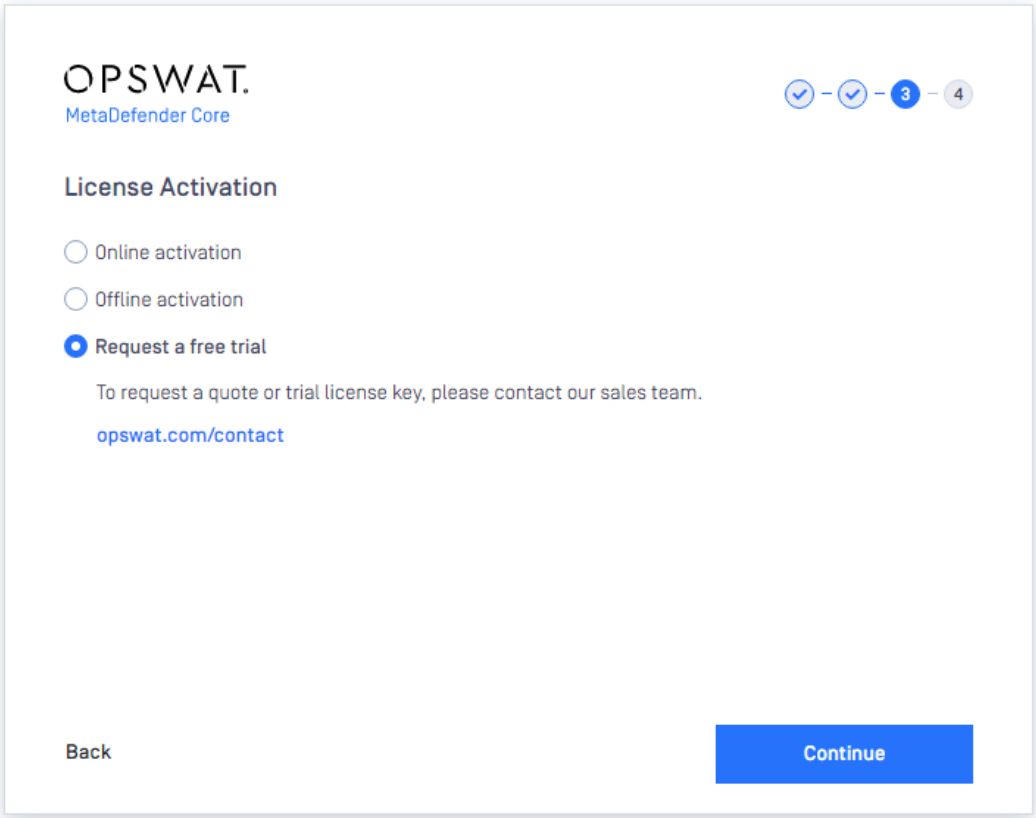
Offline activation

With no internet connection on the server the Metadefender v4 instance may be activated indirectly from a different machine, that has internet connection. The Deployment ID of the Metadefender v4 instance and the the Activation key received at the time of purchasing the product will be required. Follow the steps on the screen to activate the product offline.



Request trial key online

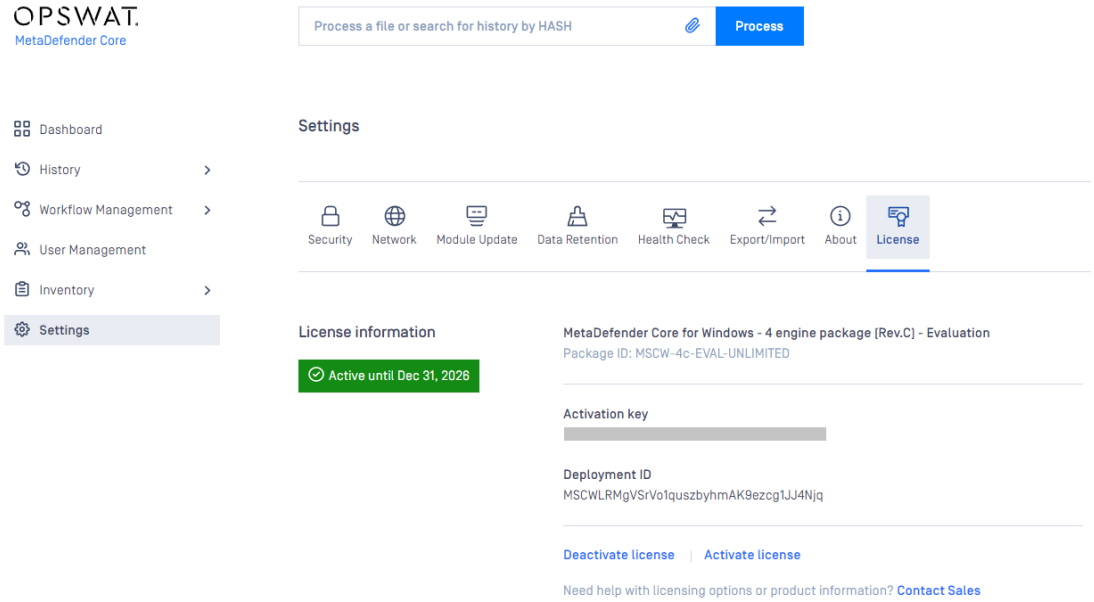
An evaluation license could be acquired by contacting our sales team.



Check Your License Details

For more license details and activating your MetaDefender Core, go to Settings > License menu tab on the Web Management Console:

- Product name: product name as on your order (e.g. MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation)
- Package ID: product identification as on your order (e.g. MSCW-4c-EVAL-UNLIMITED)
- Expiration: last day of license validity
- Activation key: key used for activating product
- Deployment ID: identification of this installation


OPSWAT.
MetaDefender Core
Process a file or search for history by HASH
Process
Dashboard
History
Workflow Management
User Management
Inventory
Settings
Settings
Security
Network
Module Update
Data Retention
Health Check
Export/Import
About
License
License information
Active until Dec 31, 2026
MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation
Package ID: MSCW-4c-EVAL-UNLIMITED
Activation key
Deployment ID
MSCWLRMgVSrVo1quszbyhmAK9ezcg1JJ4Njq
Deactivate license
Activate license
Need help with licensing options or product information? Contact Sales

Scripted license management

Using REST API calls there is an option for scripted activation and deactivation for MetaDefender Core servers with **no Internet connection or if your infrastructure is using automation to create/destroy MetaDefender Core instances**. In this chapter the steps of these two scenarios are described.

Requirements

- an installed MetaDefender Core instance without Internet connection
- another computer that has Internet connection and can run your activation scripts
- a manual or automated way to transfer data between the two computers

Activation steps

1. In order to activate MetaDefender Core, deployment ID and activation key are needed to generate the license file.
    a. Activation key should be purchased from OPSWAT.
    b. MetaDefender Core deployment ID can be queried by using REST API (for details see GET Get Current License Information API page)

> **Info**
> Save this deployment ID in your system, you might need this ID when the instance is unavailable at the time of deactivation.

2. Activate license and get the license file using the following URL, where <quantity> is 1.

activation call bash

```
curl -X GET https://activation.dl.opswat.com/activation?key=<activation-key>&deployment=<deployment unique-ID>&quantity=<quantity>
```

If the activation is successfully, a license file is downloaded. Save this file.

HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	license file	Activation was successfully.
200 Activation failed	error: '<user conform error message>'	Failed activation
200 Invalid parameter	error: 'Could not activate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not activate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
200 Invalid parameter	error: 'Could not activate your product because the quantity you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key, quantity or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Info
In case of any activation issue, contact OPSWAT support for help

3. Upload license file to MetaDefender Core: The license file should be uploaded to the MetaDefender Core v4 to activate the product. For details see page: POST Upload license key API

Deactivation steps

1. For deactivation, following infomation is required:
    a. Activation key
    b. MetaDefender Core deployment ID: can be queried by using REST API (for details see GET Get Current License Information API page)
2. Deactivate license using the following URL:

deactivation call bash

```
curl -X GET https://activation.dl.opswat.com/deactivation?key=<activation-
key>&deployment=<deployment unique-ID>
```

Response HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	result: ok	Successful deactivation
200 No active license found	error: 'Could not found any active license with the given parameters'	The license has not been activated yet or it has been deactivated already.
200 Invalid parameter	error: 'Could not deactivate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not deactivate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Important notes

Warning

Product activation is tied to several hardware, operating system and software parameters. In case of one or more major hardware or software parameter change the product might turn into deactivated status. Operating system updates, other software updates should not affecting the activation status.

If the product is online activated and has live Internet connection then in case of deployment ID change the product reactivates itself. If the product doesn't have Internet connection then the administrator is responsible to reactivate the product.

Because of the above administrator should consider licensing restrictions of the offline deployments if using any virtualization and/or containerization technologies.

File Processing

MetaDefender Core provides REST API interface for file processing business.

There are several ways to process files with MetaDefender Core:

- Analyze Files via Web Interface
- Process Files via REST API

Product Upgrading

> **Info**
> Simply run installer file, details could be found at Installation

> **Info**
> For command line mode, upgrading (e.g. on **Linux**), prior to running the installer file, please make sure to **create the ignition file** with PostgreSQL server information included. Details:
>
> - Standalone DB mode: Command Line Installation (Windows and Linux)
> - Shared DB mode: Command Line Installation

Recommendation for MetaDefender Core upgrade

Our goal is to help you execute the product upgrade continuously, efficiently, and correctly.

PostgreSQL service upgrade

- **Bundled PostgreSQL Service**: If you are using the bundled PostgreSQL server included in the installer, the PostgreSQL service will be automatically upgraded to the newer version (if available). No additional action is required for this process.
- **Remote PostgreSQL Service**: For remote PostgreSQL servers, customers are responsible for managing and upgrading the service independently. Upgrading MetaDefender Core does not update the remote PostgreSQL service.

Before the upgrade

Prepare a proper disaster recovery plan (DRP) including some common practices:

- Backup, or take a snapshot of your current working MetaDefender Core server.
  - This is to help quickly rollback for any upgrade failure might cause later for any reason.
- Optionally backup your PostgreSQL database, especially when using remote shared database mode.
  - Reference: https://www.postgresql.org/docs/current/backup.html
  - Please do it after stopping all linked MetaDefender Core instances to avoid potential data collision.
- Lastly, if backing up entire server is not feasible, then at least export MetaDefender Core configuration package Import/Export configuration just in case you'll need to start off a new server with current MetaDefender Core configuration to keep the scanning service running without any impact.

On the PostgreSQL server, double check server resource to ensure sufficient RAM memory and disk space to be preserved for the upgrade process.

- 8GB RAM available at least during the upgrade.

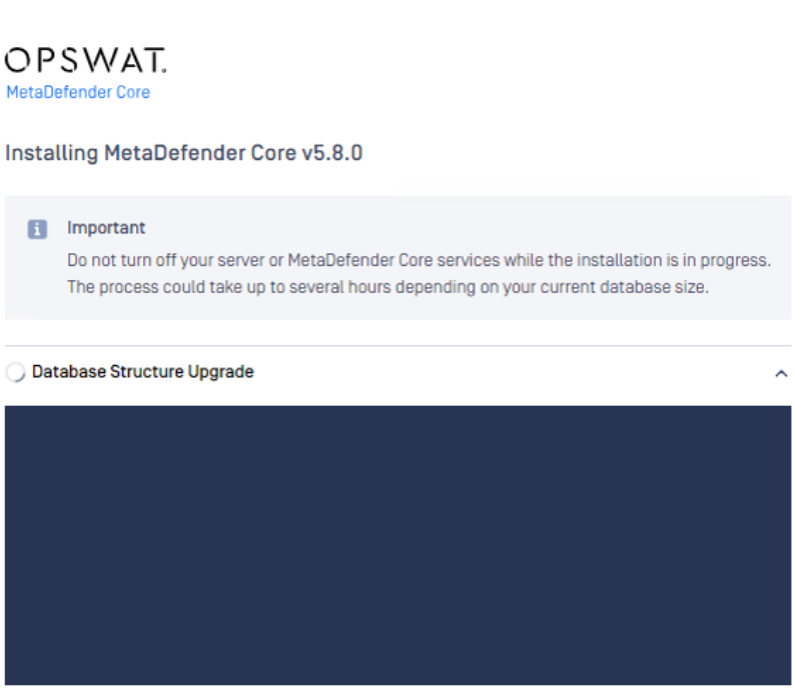
- Disk availability is greater than MetaDefender Core database size.

While doing the upgrade

Applicable for shared database deployment:

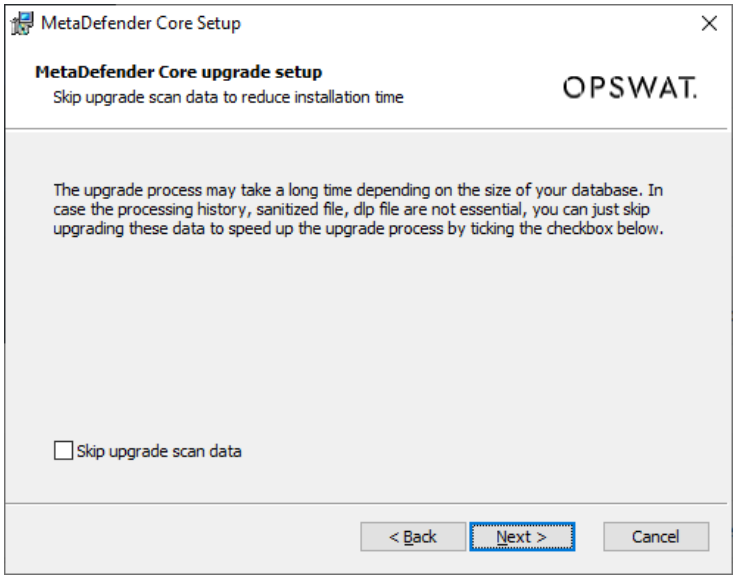
- Stop all MetaDefender Core instances' scanning service before performing the upgrade.
- Upgrade one (1) MetaDefender Core instance at a time.
- Proceed to upgrade MetaDefender Core instance where the bundle local PostgreSQL server is used for shared database setup.
- Otherwise is using remote PostgreSQL then the upgrade order does not matter.

For large size database, mostly for scan history, expecting long time to be spent on the upgrade process.



The product supports an option to skip data migration for scan history to expedite the process drastically and mitigate any failure risk. You can manually backup all data from your PostgreSQL server for offline usage if any.

Check "Skip upgrade scan data" to enable this feature while doing the upgrade.



or add following key in the ignition file:

none

```
[internal]
skip_upgrade_scan_data = 0
```

Troubleshooting

If the upgrade process ever runs into an error:

- Double check your network and services, applicable to remote PostgreSQL server.
- Optionally configure MetaDefender Core to skip migrating scan history.
  - Windows
    - Open Windows registry, and navigate to HKLM\Software\OPSWAT\Metascan
    - Add a new key `internal/skip_upgrade_scan_data` with value = 1
  - Linux
    - Open MetaDefender Core configuration file `/etc/ometascan/ometascan.conf`
    - Add new key into the file
      - [internal]
      - skip_upgrade_scan_data=1
  - Note: This key will be auto removed upon the upgrade finishes.
  - Then restart MetaDefender Core service to retry the upgrade.
    - Might retry this a few times until the upgrade is done.

Just in case the upgrade process still fails, please contact OPSWAT support with the support package - How to Create Support Package? This will help us analyze the root cause and figure out an optimal solution for your system failure.

Using external Load Balancer

Application Load Balancing (Layer 7)

This page provides information about recommended ways to use sessions with a Layer 7 load-balancer.

Most load-balancers have the ability to provide cookies for pinning subsequent traffic from a client to the appropriate server. This method is called sticky session, session persistence or session affinity. Using cookies efficiently requires the client to know when a cookie should be sent, should not be sent or should be deleted.

Sticky session load balancing

Each Metadefender Core instance has its own databases and application sessions that cannot be seen by other instances. Therefore, in order to get the related data/response to our queries we should ask the appropriate Metadefender Core v5 server. On the other hand, to keep the advantages of the used load balancing method, cookies should not be sent if it is not necessary.

Single file scanning

Step	Stage	Task	Cookie usage
1.	Sending file	Initiate processing a file on the client side. Send file through the load-balancer. (See POST File Analysis API)	Cookie should not be sent
2.	Sending file	Save the cookie and the data_id you got from the load-balancer	Save cookie
3.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Send cookie saved in step 2
4.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Send cookie saved in step 2

Batch scanning

Step	Stage	Task	Cookie usage
1.	Open batch	Initiate processing file(s) in batch. Request a batch ID through the load-balancer. (See POST Initiate Batch API)	Cookie should not be sent
2.	Open batch	Save the cookie and the batch_id you got from a Core server through the load-balancer.	Save cookie
3.	Sending files	Send file through the load-balancer. (See POST File Analysis (using batch header) API)	Send cookie saved in step 2
4.	Sending files	Save the data_id you got from the load-balancer.	-
	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Send cookie saved in step 2
5.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
6.	Getting batch status	See GET Batch Status API	Send cookie saved in step 2
7.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Send cookie saved in step 2
8.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API	Send cookie saved in step 2

If it does not matter which upstream server responds, the queries should be sent without cookie.

> **Info**
> It is recommended not to send cookies when it's not necessary to allow load-balancer to use its own method to share the load between Metadefender Core v5 servers.

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration through load-balancer

OPSWAT products that support HTTP load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	4.3.4	-
OPSWAT Client	Windows: 7.6.247.0 Mac: 10.4.243.0	-
MetaDefender Email Security	4.3.0	-
MetaDefender ICAP Server	4.3.0	-
MetaDefender Vault	1.3.0	-
MetaDefender for Secure Storage	2.0.7	-

DNS Load Balancing

Using this method is logically similar to Layer 7 load-balancing.

Briefly how it works

Client uses a domain name to send a query to a server. Client's DNS server has more "A" records for that name with different IPs. When a client resolves the server's domain name DNS server randomly chooses an IP for that name to send back. When a session is used on the application layer, client should know the IP address of the Core server that handles that specific session. In every other case, client should resolve the domain name with DNS query to let requests to be balanced between Metadefender Core servers.

Single file scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Sending file	Initiate processing a file on the client side. (See POST File Analysis API)	Use IP saved in step 2
4.	Sending file	Save the data_id got from the Core server	-
5.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Use IP saved in step 2
6.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Use IP saved in step 2

Batch scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Open batch	Initiate processing file(s) in batch. Request a batch ID. (See POST Initiate Batch API)	Use IP saved in step 2
4.	Sending files	Send file to the specific Core server with the basch ID saved in step 3. (See POST File Analysis (using batch header) API)	Use IP saved in step 2
5.	Sending files	Save the data_id you got from the Core server.	-
6.	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Use IP saved in step 2
7.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
8.	Getting batch status	See GET Batch Status API	Use IP saved in step 2
9.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Use IP saved in step 2
10.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API).	Use IP saved in step 2

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration via DNS load-balancing

OPSWAT products that support DNS load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	does not support yet	-
MetaDefender Client	does not support yet	-
MetaDefender Email Security	does not support yet	-
MetaDefender ICAP Server	does not support yet	-
MetaDefender Vault	does not support yet	-

SSL connection for PostgreSQL communication

> **Note**
> The SSL connection configuration is possible thanks to the support made by PostgreSQL. For reference: https://www.postgresql.org/docs/14/ssl-tcp.html
>
> MetaDefender Core is tested against this mode on particular version 4.21.1

Step 1: Generate SSL certificate for PostgreSQL server

> **Note**
> Following instructions are for self-sign certificate. For production environment, making sure to use the certificate signed by your trusted CA.

1) Generate a private key

bash

```
openssl genrsa -des3 -out server.key 1024
```

Sample output:

bash

```
Generating RSA private key, 1024 bit long modulus ........++++++
...............++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

Type your desired password / pass phrase to encrypt the private key

2) Remove the pass phrase to automatically start up the PostgreSQL server

bash

```
openssl rsa -in server.key -out server.key
```

> **Note**
> On Linux, make sure to set permission on the server.key file

bash

```
chmod og-rwx server.key
```

3) Create a self-signed certificate

bash

```
openssl req -new -key server.key -days 3650 -out server.crt -x509
```

You will be prompted to enter detailed information which is incorporated into your self-signed certificate request.

none

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:CA
Locality Name (eg, city) [Default City]:San Francisco
Organization Name (eg, company) [Default Company Ltd]:OPSWAT
Unit Name (eg, section) []:OPSWAT
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:dummy_email@opswat.com
```

For self-signed certificate, use the server ceriticate as the trusted root certificate:

bash

```
cp server.crt root.crt
```

Copy **server.key**, **server.crt**, **root.crt** to PostgreSQL data folder. For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data**
- Linux: **/var/lib/ometascan/pg_data/**

Step 2: Configure PostgreSQL server for SSL authentication connection

1) Create a custom config file (e.g. ssl.conf) for PostgreSQL.

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\postgres\ssl.conf**
- Linux: **/var/lib/ometascan/postgres/ssl.conf**

2) Content of ssl.conf:

none

```
ssl = on
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
```

3) Modify pg_hba.conf

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data\pg_hba.conf**
- Linux: **/var/lib/ometascan/pg_data/pg_hba.conf**

Add following (modify **::1/128** if needed to match with your environment requirement)

none

```
hostssl all all      ::1/128      scram-sha-256
```

After the modify, the content of pg_hba.conf should be like this (example):

> **Note**
> In a Linux environment, a local connection is required to upgrade from version 5.5.1 or earlier to version 5.6.0 or later.

none

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local         DATABASE  USER  METHOD  [OPTIONS]
# host          DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl     DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostgssenc    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnogssenc  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
local    all all                    scram-sha-256
hostssl all all      ::1/128      scram-sha-256

#More content.......
```

4) Reload the services

- Standalone DB mode: Restart **ometascan** service (MetaDefender Core service)
- Shared DB mode: Restart **ometascan-pg** / **ometascan-postgresql** service (MetaDefender Core PostgreSQL service) and then **ometascan** service (MetaDefender Core service)

Test your SSL database connection:

bash

```
psql.exe "sslmode=require dbname=postgres user=postgres host=localhost port=5432"
Password for user postgres:
psql (12.7)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
Type "help" for help.
```

This message indicate that you are now using SSL connection:

bash

```
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
```

Performance and Load Estimation

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Some factors that affect performance

- MetaDefender Core version
- MetaDefender Core engine package and configuration
  - set of engines (which and how many)
  - product configuration (e.g., thread pool size)
- System environment
  - server profile (CPU, RAM, hard disk)
  - client application location - remote or local
  - system caching and engine level caching
- Dataset
  - encrypted or decrypted
  - file types
    - different file types (e.g., document, image, executable)
    - archive file or compound document format files
  - file size
  - bad or unknown (assume to be clean)
- Performance tool

How test results are calculated

Performance (mainly scanning speed) is measured by throughput rather than unit speed. For example, if it takes 10 seconds to process 1 file, and it also takes 10 seconds to process 10 files, then performance is quantified as 1 second per file, rather than 10 seconds.

- total time / total number of files processed: 10 seconds / 10 files = 1 second / file.

Standalone DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.2.416.0

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

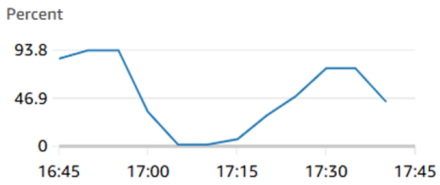
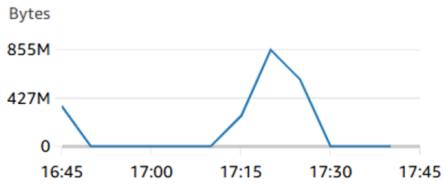
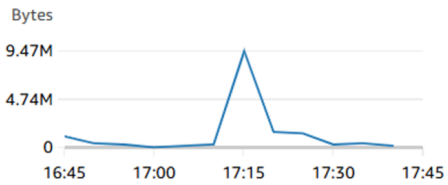
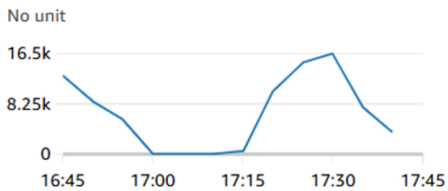
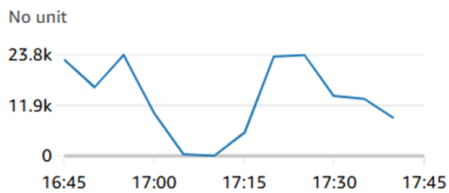
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV)	24.2	47.2 / 80	37.3 / 41.1		315,310	0.0114
Deep CDR	19.9	48.9 / 81.8	32.6 / 37		383,442	0.0094
Proactive DLP	17.1	36.8 / 58.2	30.4 / 33.6		446,228	0.0081
Vulnerability	10.1	59.2 / 85	29.3 / 33.3		755,495	0.0048

MetaDefender Core with common engine packages:

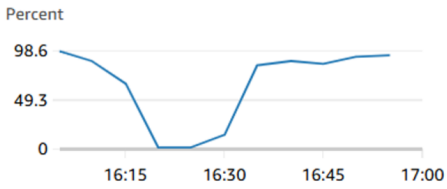
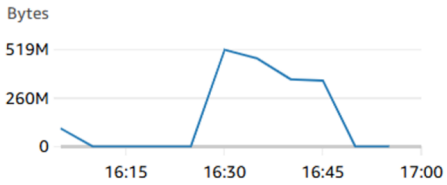
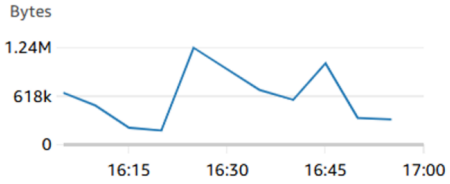
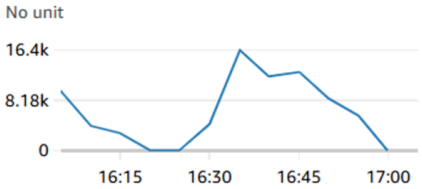
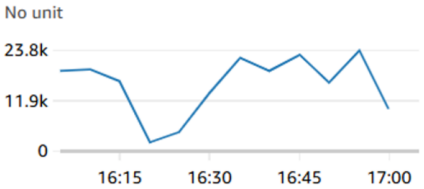
Use case	Scan duration	Avg./Max CPU usage	Avg./Max RAM usage	Avg./Max Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV) + Deep CDR	30.8	68.1 / 88	40.1 / 46.8		247,744	0.0145
Metascan (8 AV) + Deep CDR + Proactive DLP	31.2	74.2 / 98.6	41.5 / 51.1		244,567.3	0.0147
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	31.8	73.5 / 98	40.6 / 51.5		239,953	0.015

System resource utilization charts

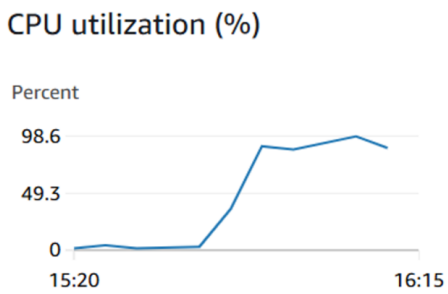
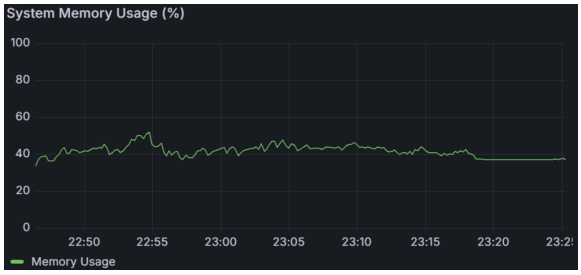
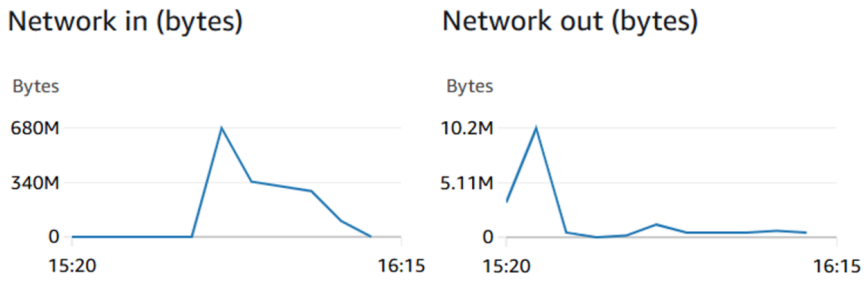
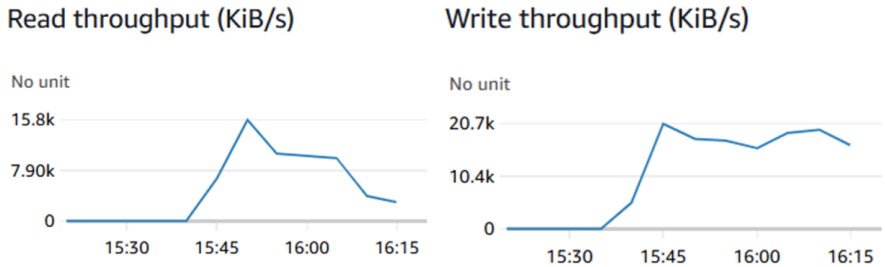
MetaDefender Core with Metascan (8 AV engines):

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

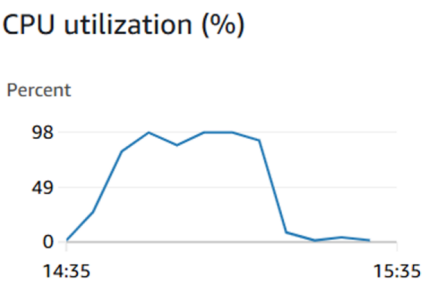
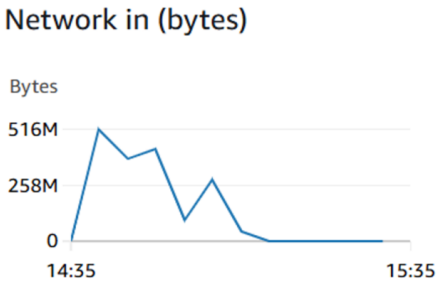
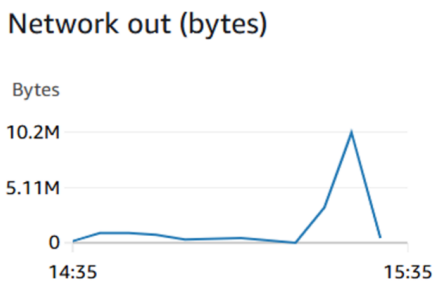
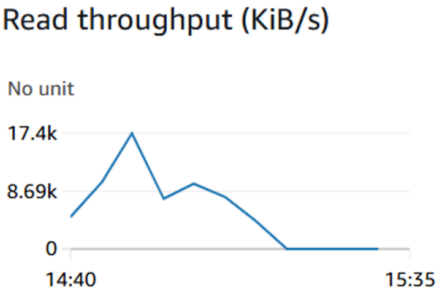
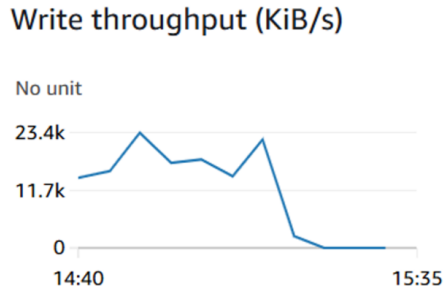
MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

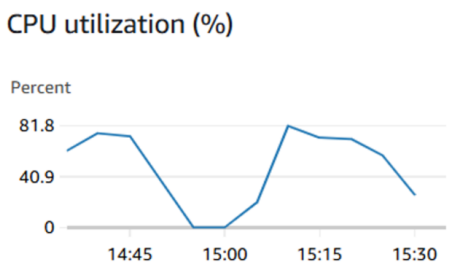
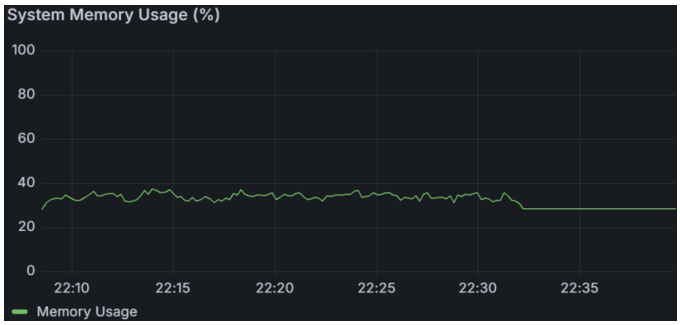
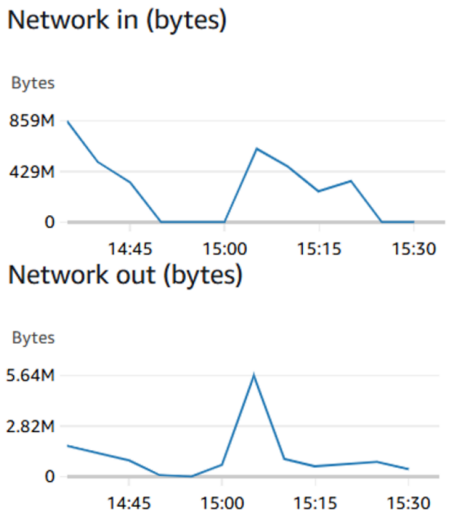
MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

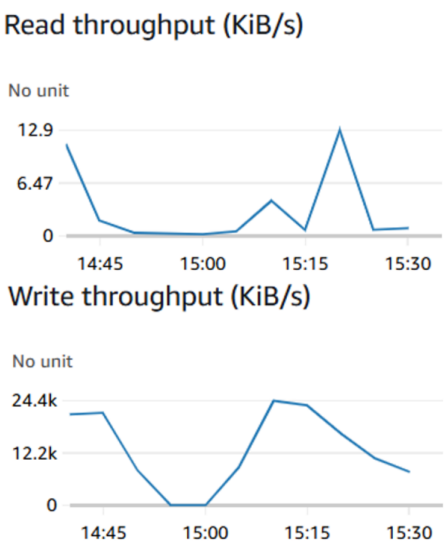
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

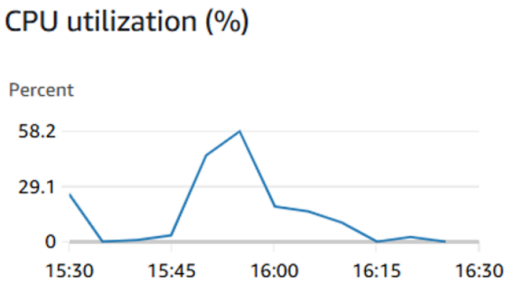
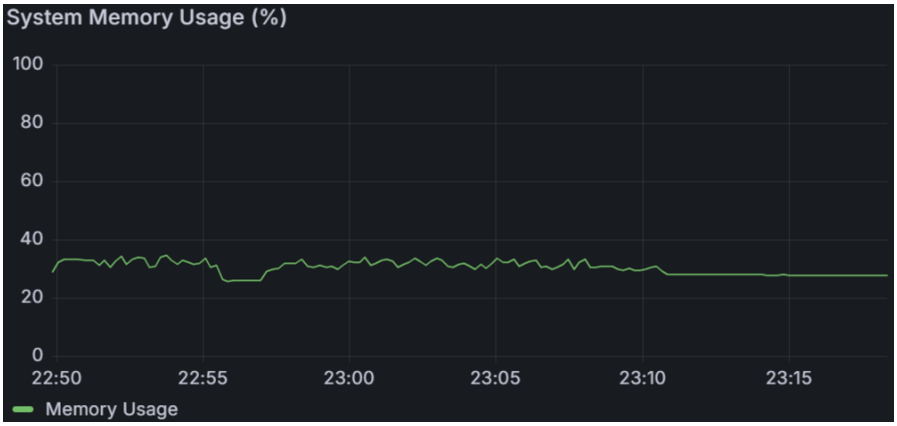
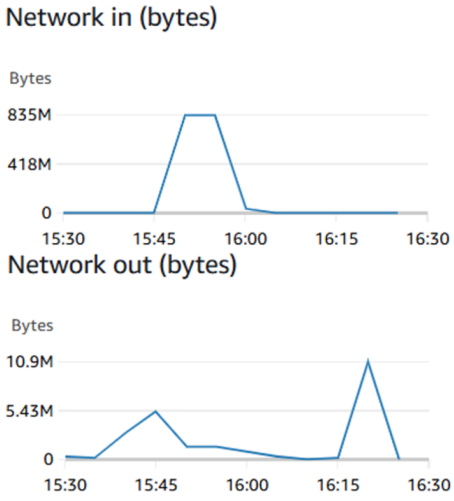
	Utilization charts
CPU usage	
Memory usage	
Network usage	 
Disk usage	 

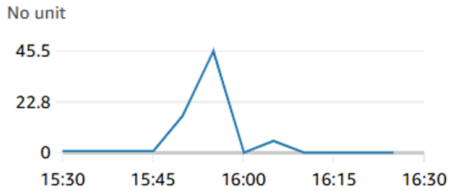
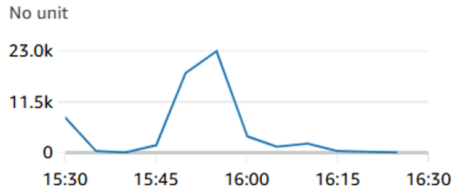
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

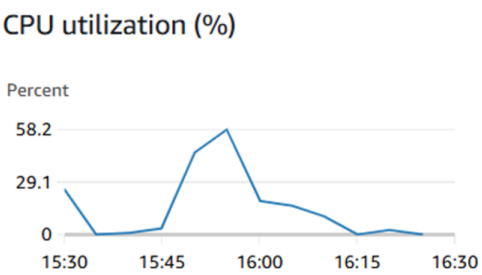
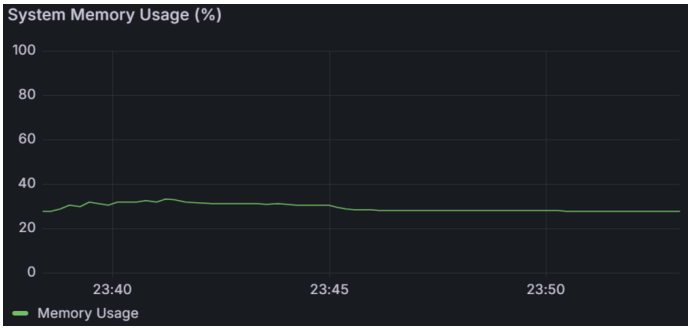
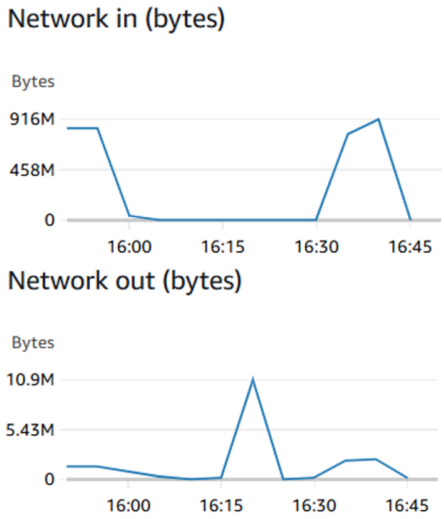
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

MetaDefender Core with Vulnerability:

	Utilization charts
CPU usage	
Memory usage	
Network usage	

Utilization charts

Disk usage



Standalone DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Ubuntu 22.04	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.57-236

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

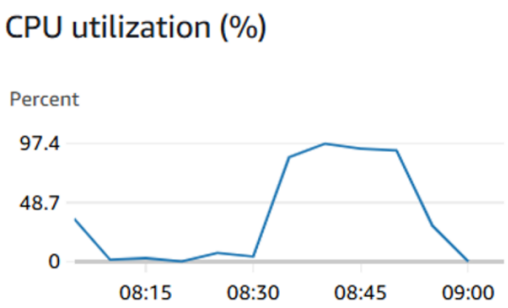
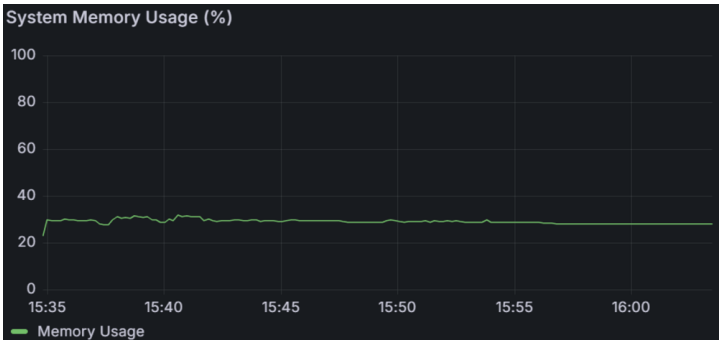
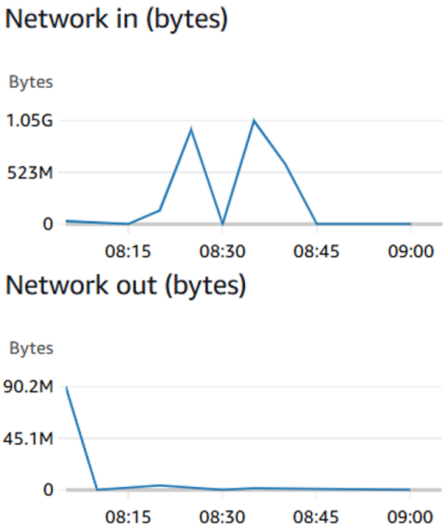
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV)	30.2	69 / 98	29 / 32		252,666	0.0142
Deep CDR	19.4	48 / 62	36 / 49		393,325	0.009
Proactive DLP	16.5	34 / 39	32 / 48		462,455	0.008
Vulnerability	12	50 / 65	46 / 46		635,875	0.006

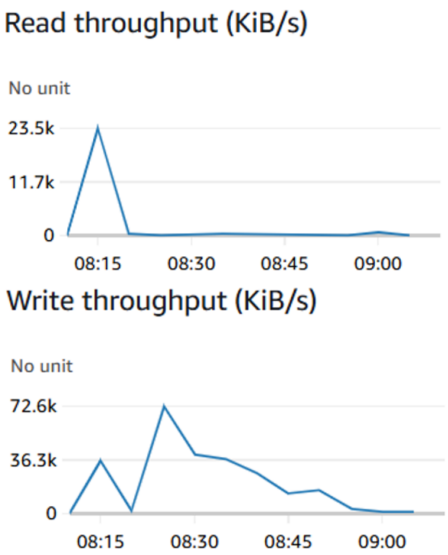
MetaDefender Core with common engine packages:

Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV) + Deep CDR	39.8	78 / 99	62 / 73		191,721	0.0188
Metascan (10 AV) + Deep CDR + Proactive DLP	43.5	61 / 84	47 / 67		175,414	0.0205
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	44.8	62 / 88	69 / 83		170,324	0.0211

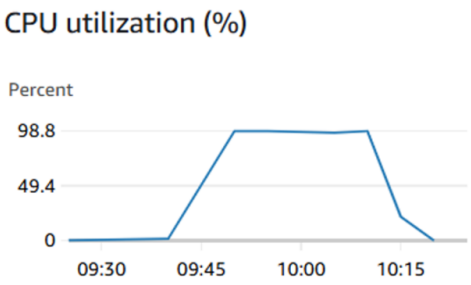
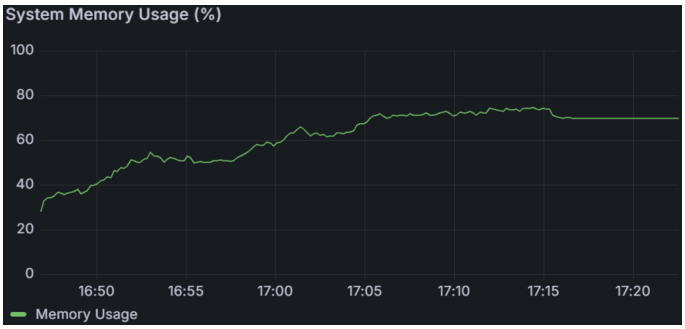
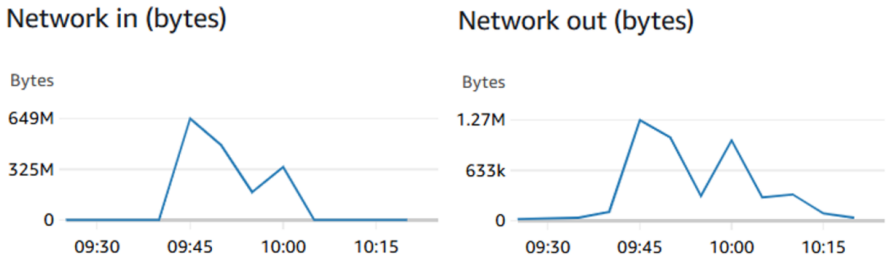
System resource utilization charts

MetaDefender Core with Metascan (10 AV engines) :

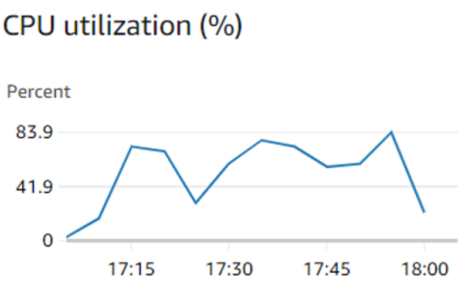
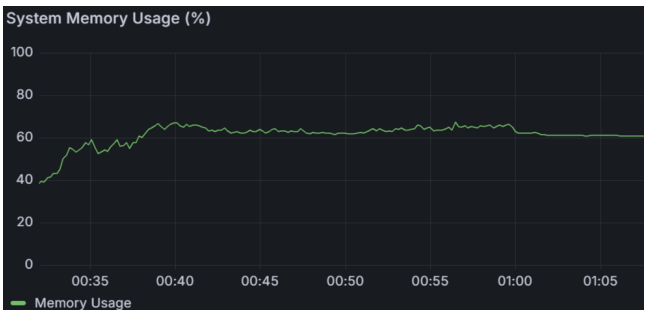
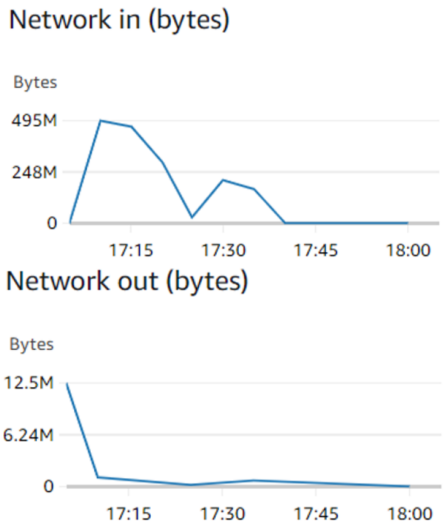
	Utilization charts
CPU usage	
Memory usage	
Network usage	

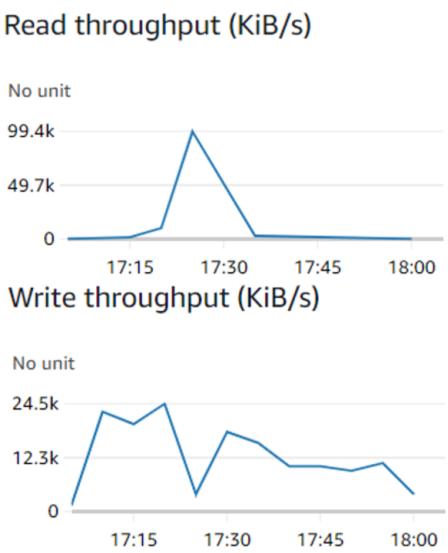
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR :

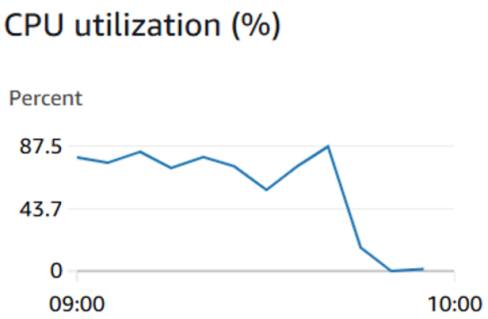
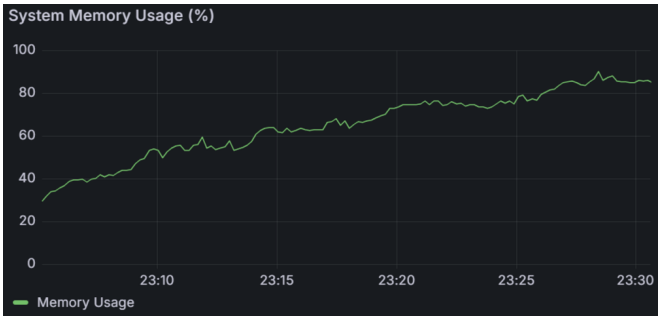
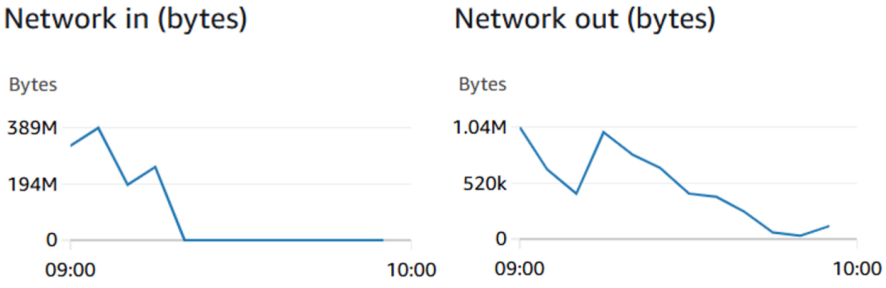
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP :

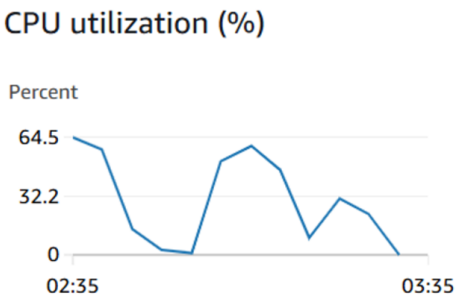
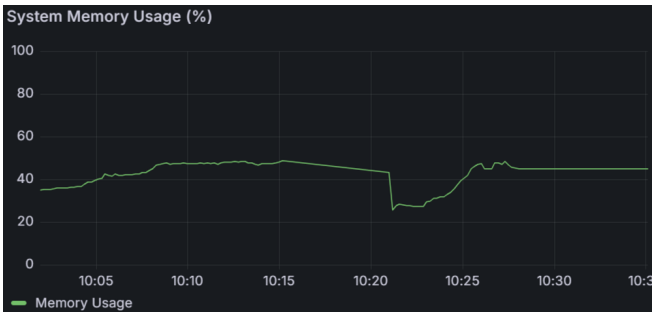
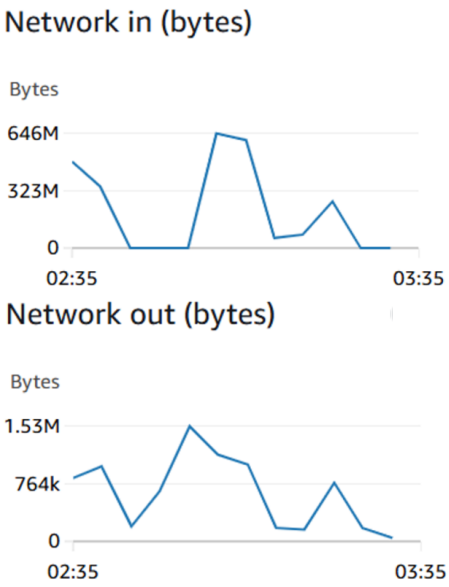
	Utilization charts
CPU usage	
Memory usage	
Network usage	

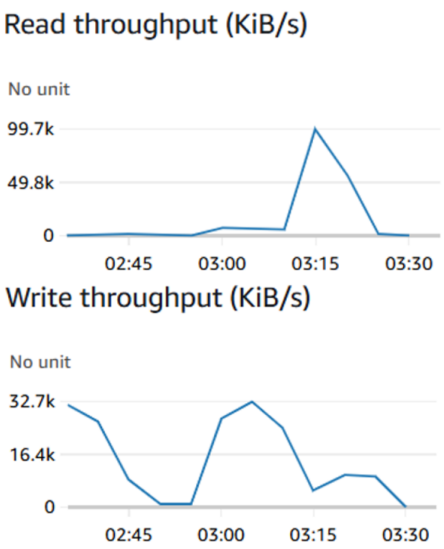
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP + Vulnerability :

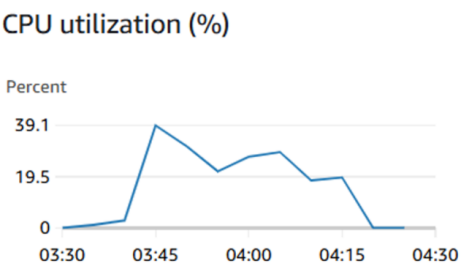
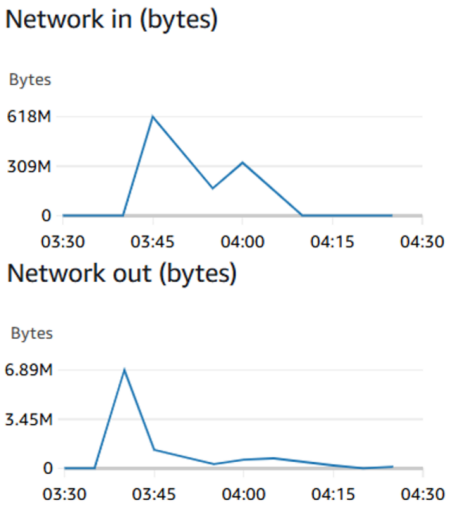
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

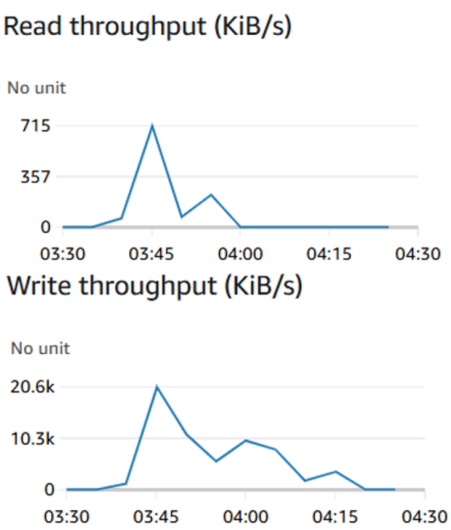
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

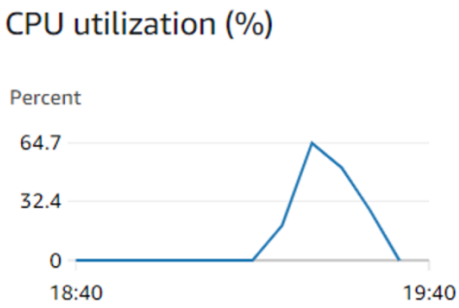
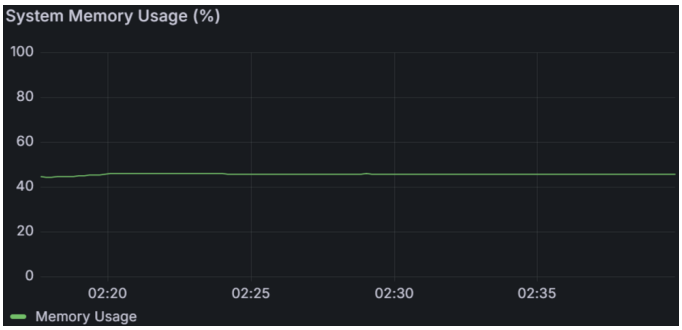
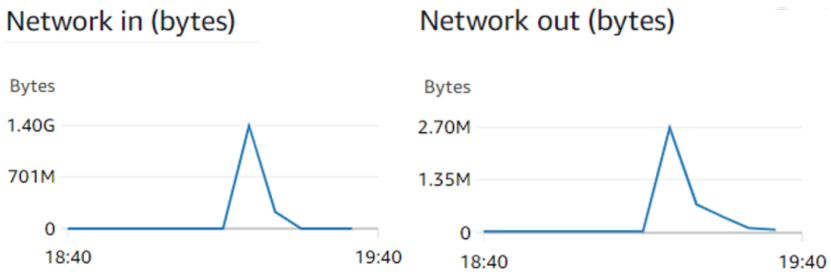
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability :

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

Shared DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

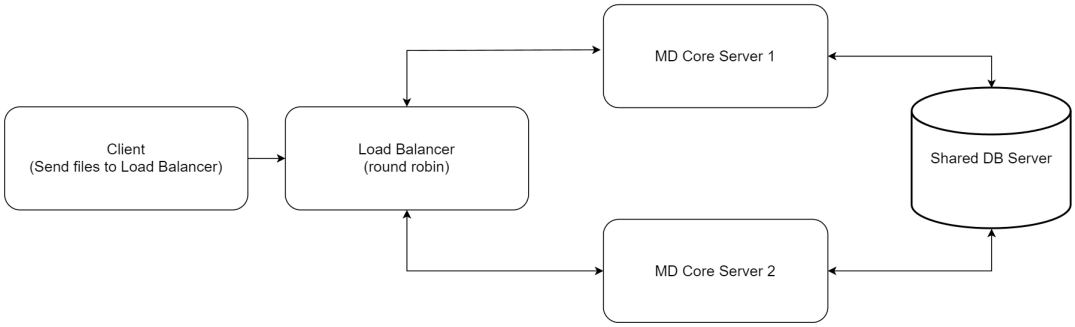
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV)	11.9	37.0	41.9	30.1	30.4	Receive: 6441.1 Send: 769.6	Receive: 6200.2 Send: 693.7	640,821.2	0.0
Deep CDR	10.5	60.1	66.7	30.3	32.2	Receive: 6379.8 Send: 594.8	Receive: 7146.1 Send: 609.6	729,368.5	0.0
Proactive DLP	7.8	38.7	37.5	29.7	31.0	Receive: 8842.6 Send: 791.8	Receive: 8635.6 Send: 729.1	965,455.7	0.0
Vulnerability	4.4	52.2	53.0	26.0	26.9	Receive: 16,215.5 Send: 1006.8	Receive: 15,785.7 Send: 997.2	1714,449.3	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV) + Deep CDR	15.4	72.4	75	34.3	35.4	Receive: 5436.6 Send: 611.3	Receive: 5589.3 Send: 607.2	495,697.4	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP	16.9	77.2	80.2	36.4	37.9	Receive: 5254 Send: 585.4	Receive: 5041.3 Send: 576.1	450,657.6	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	18.0	72.1	80.4	38.6	38.0	Receive: 4504.3 Send: 574.1	Receive: 4649.5 Send: 571.1	423,010	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

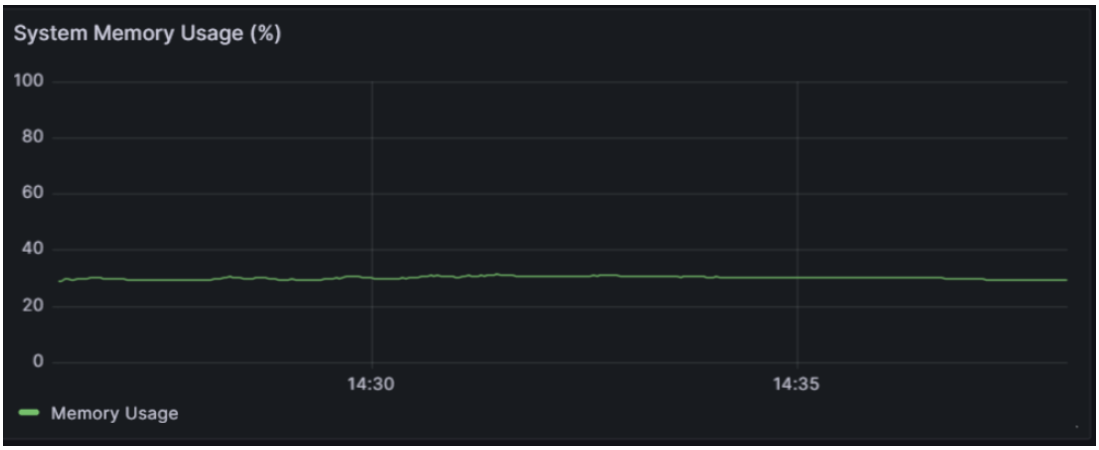
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2

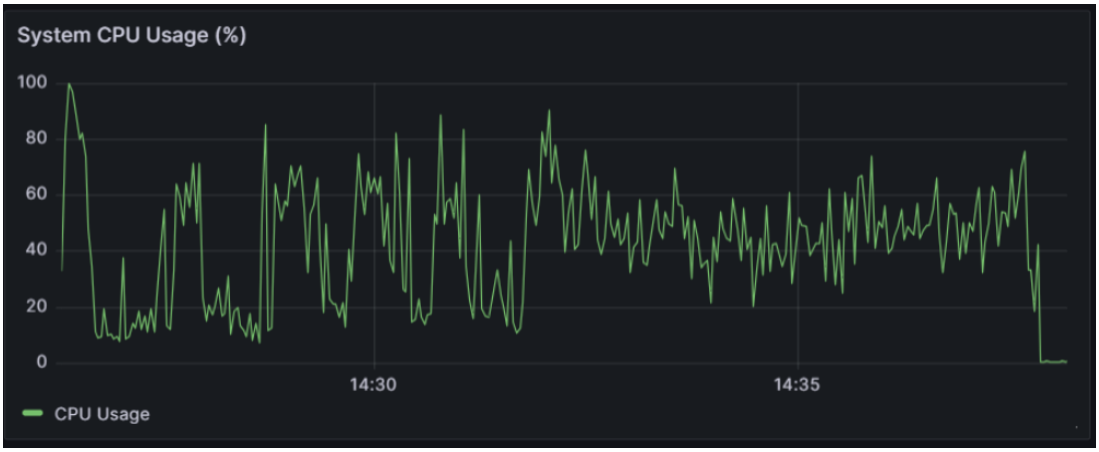


CPU usage

MD Core #1

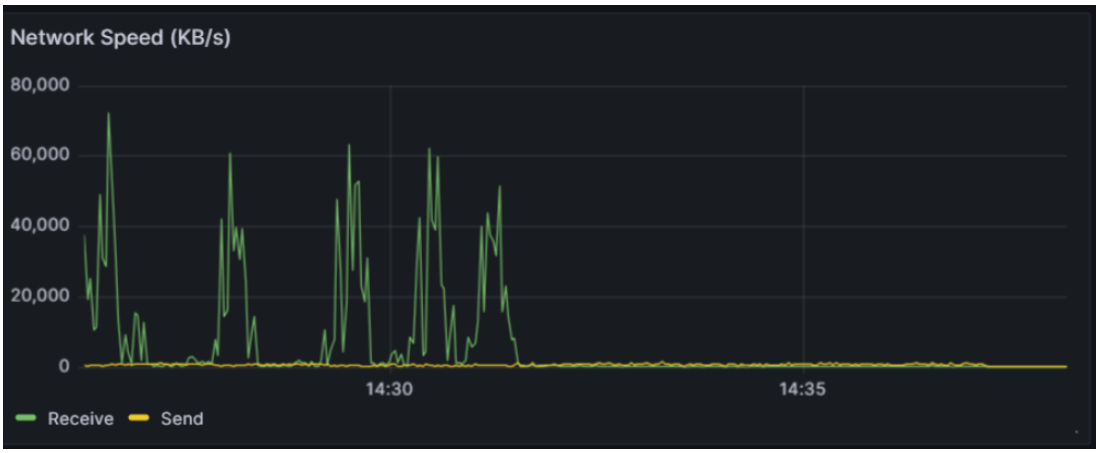


MD Core #2

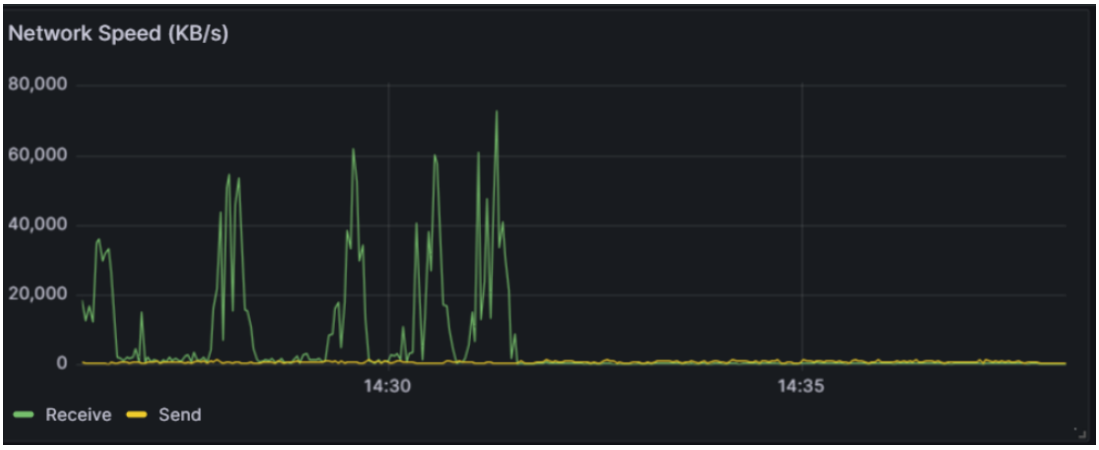


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2

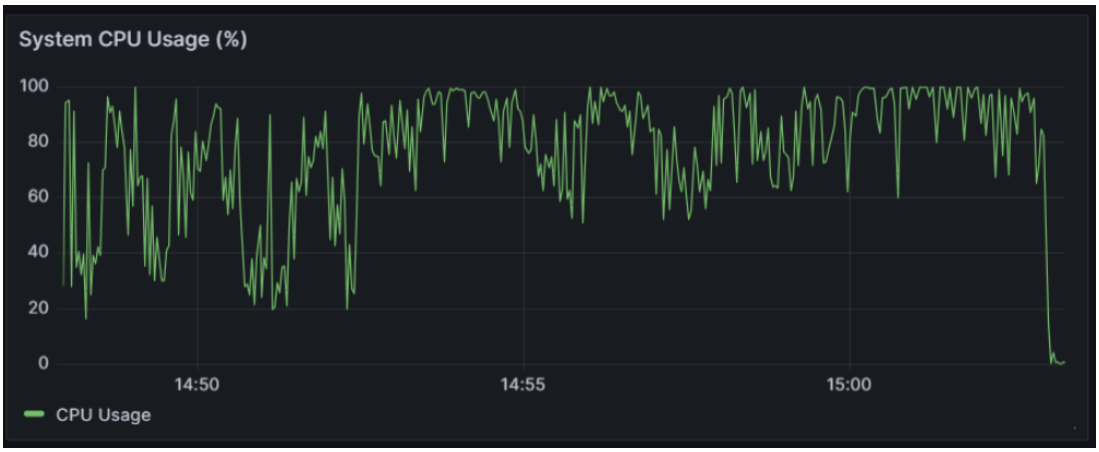


CPU usage

MD Core #1

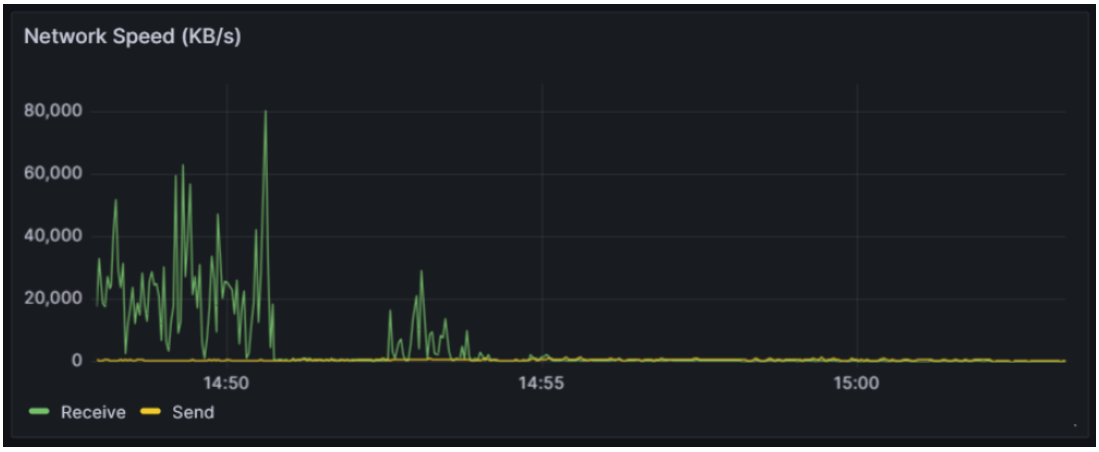


MD Core #2



Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

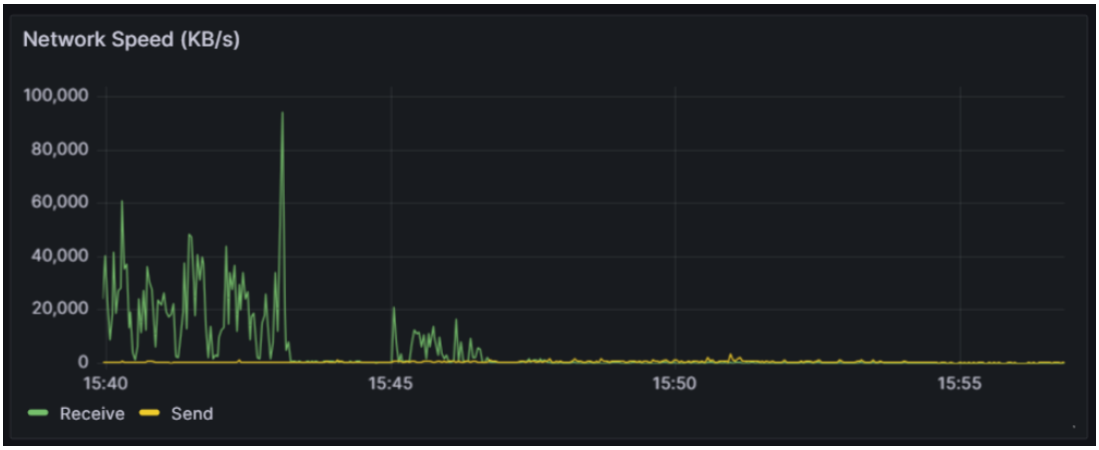


MD Core #2

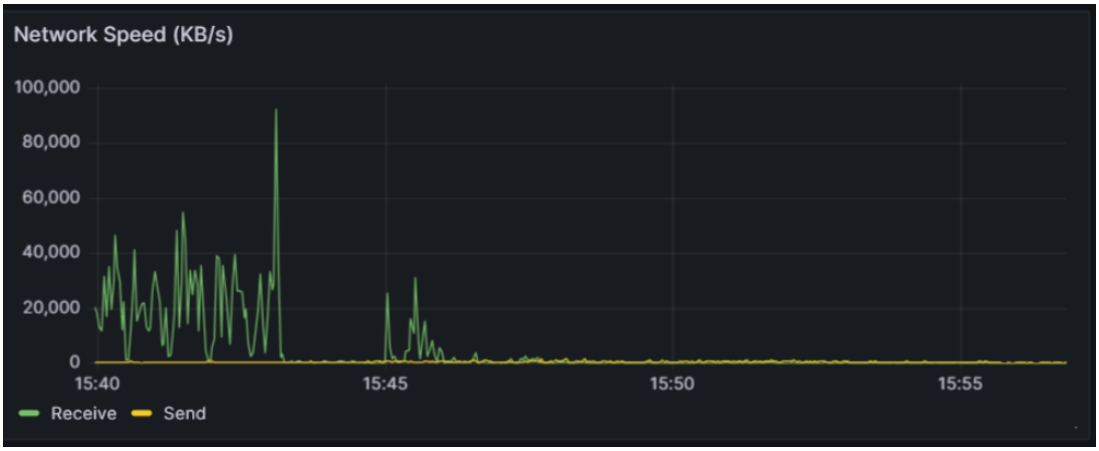


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1

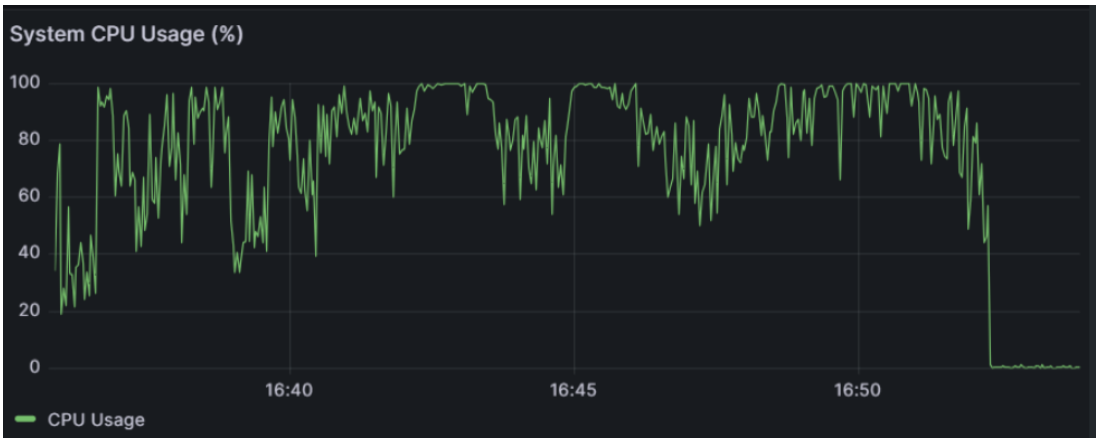


MD Core #2



CPU usage

MD Core #1

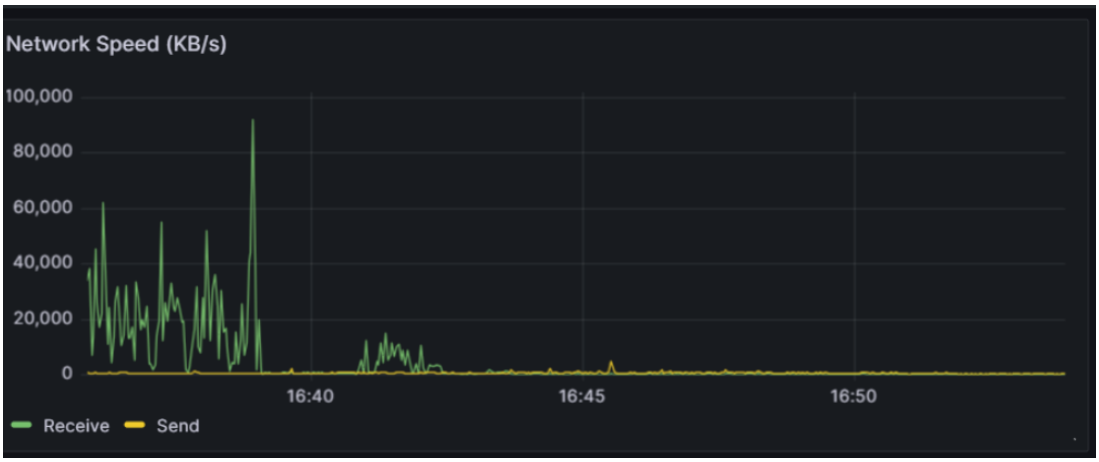


MD Core #2

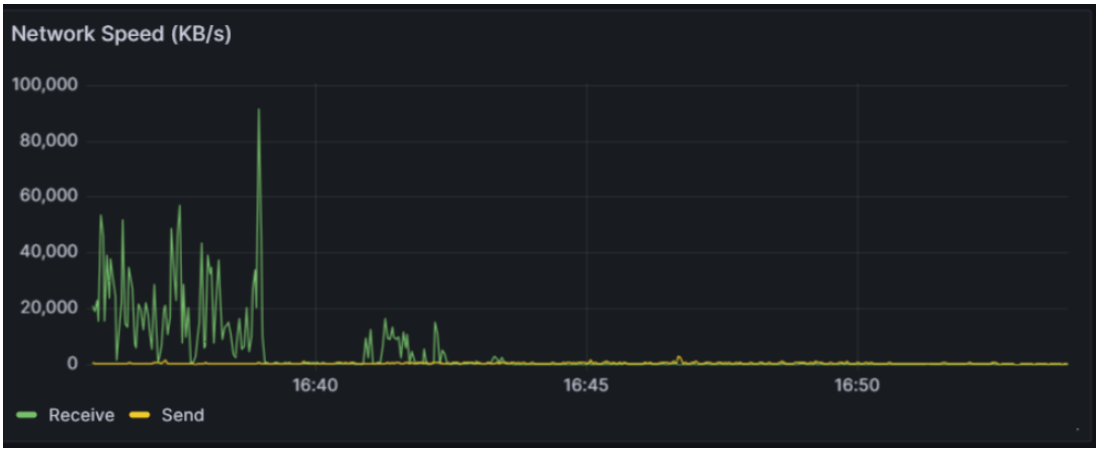


Network usage

MD Core #1



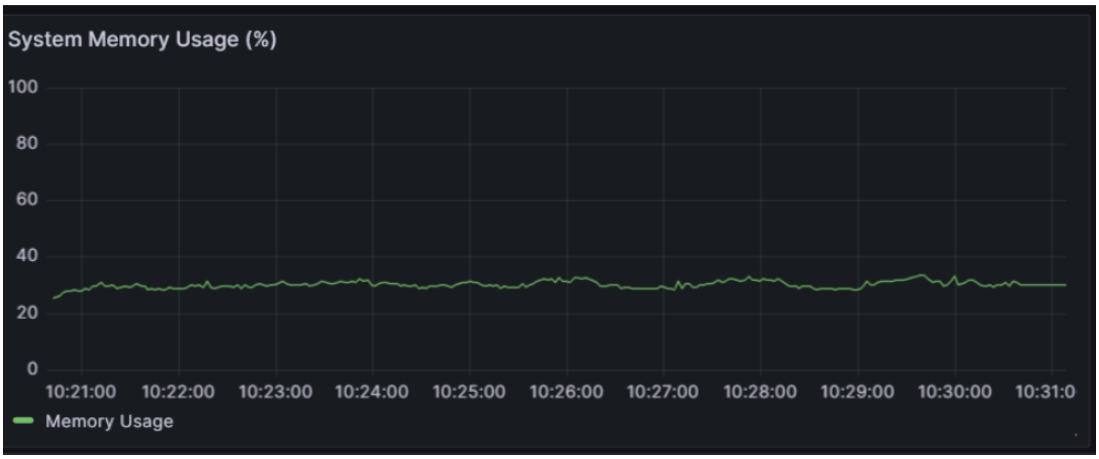
MD Core #2



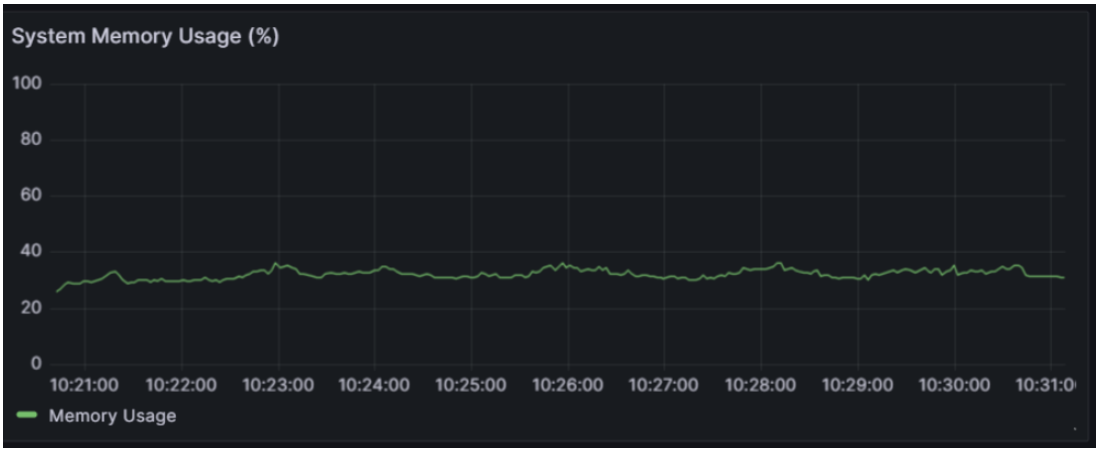
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

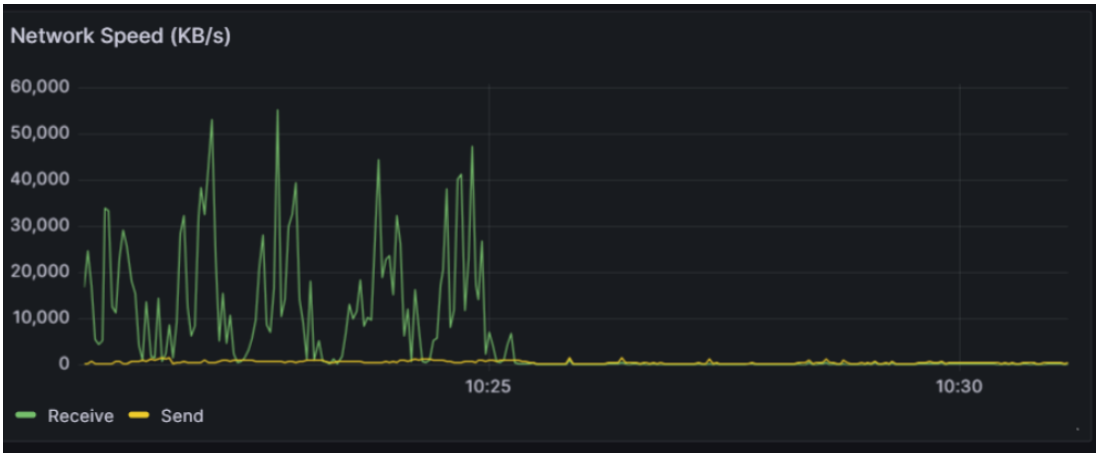


MD Core #2

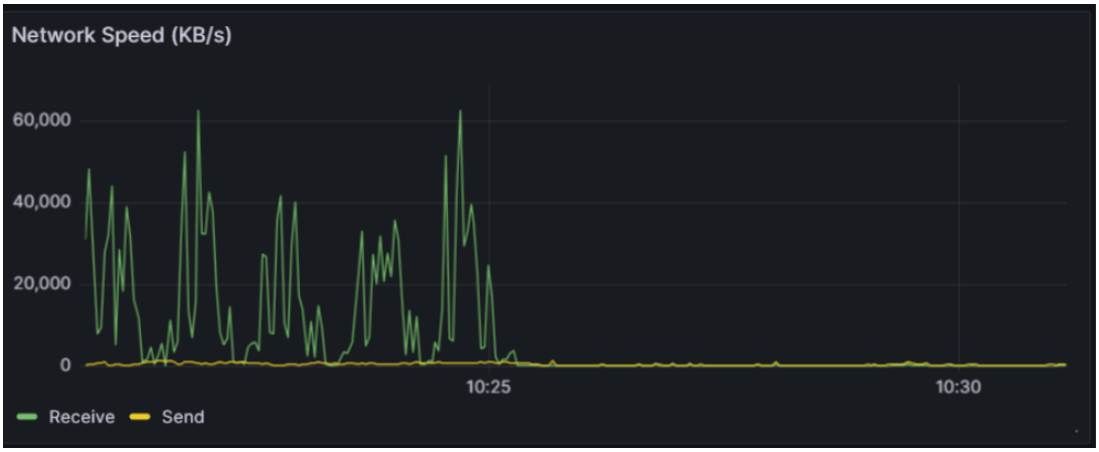


Network usage

MD Core #1



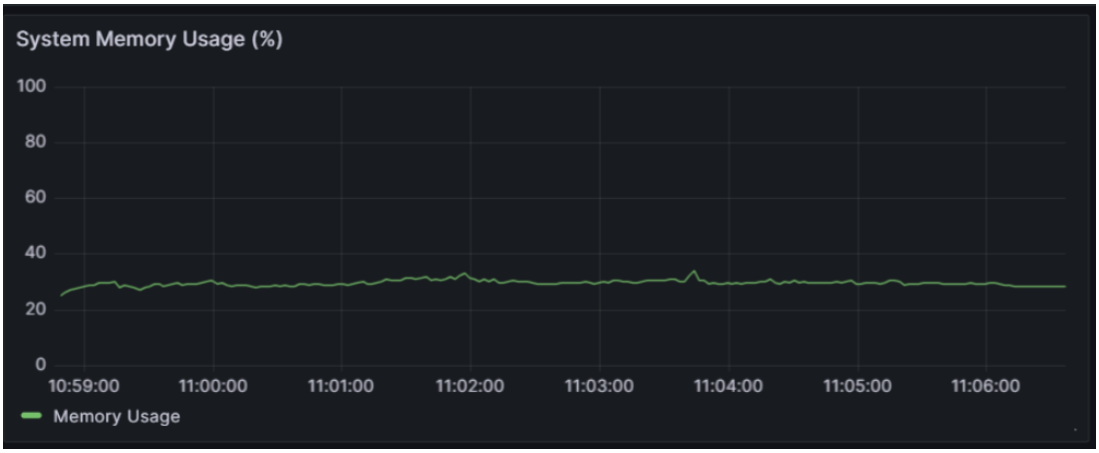
MD Core #2



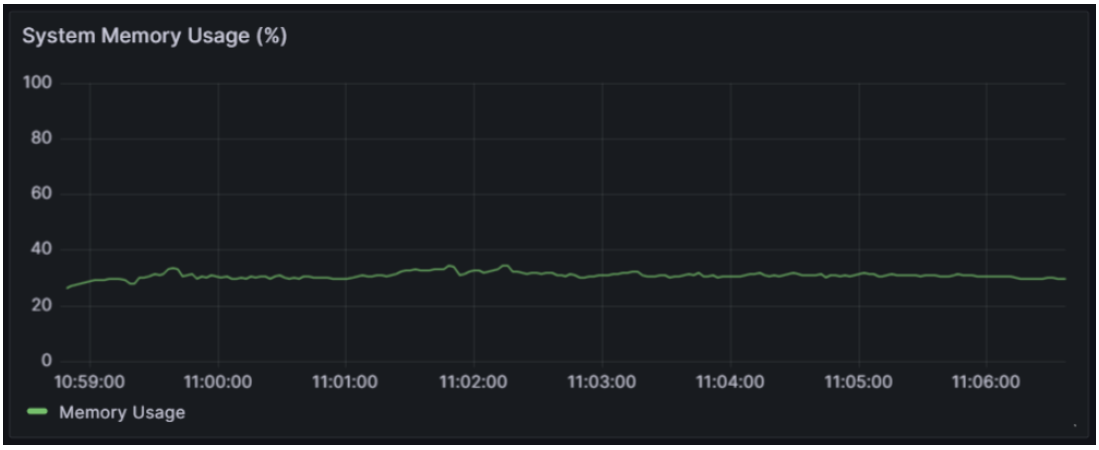
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

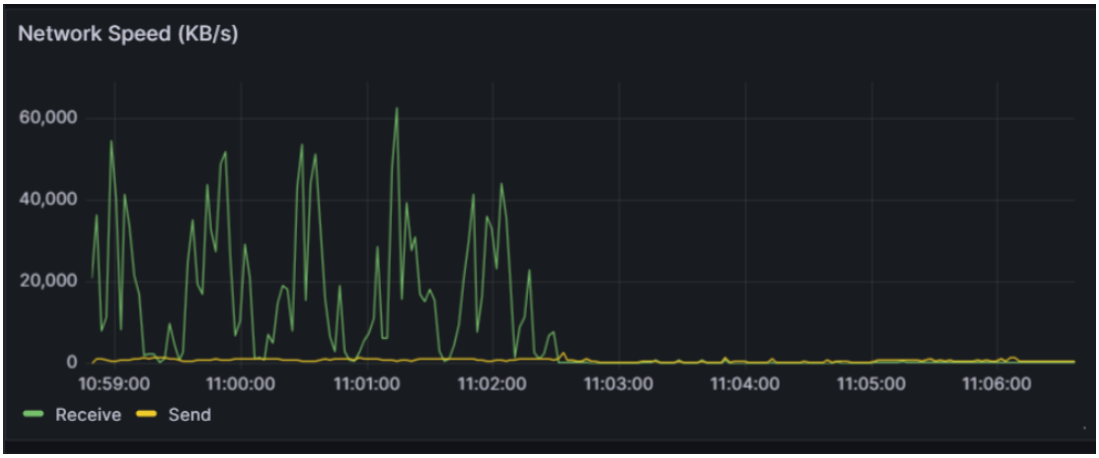


MD Core #2

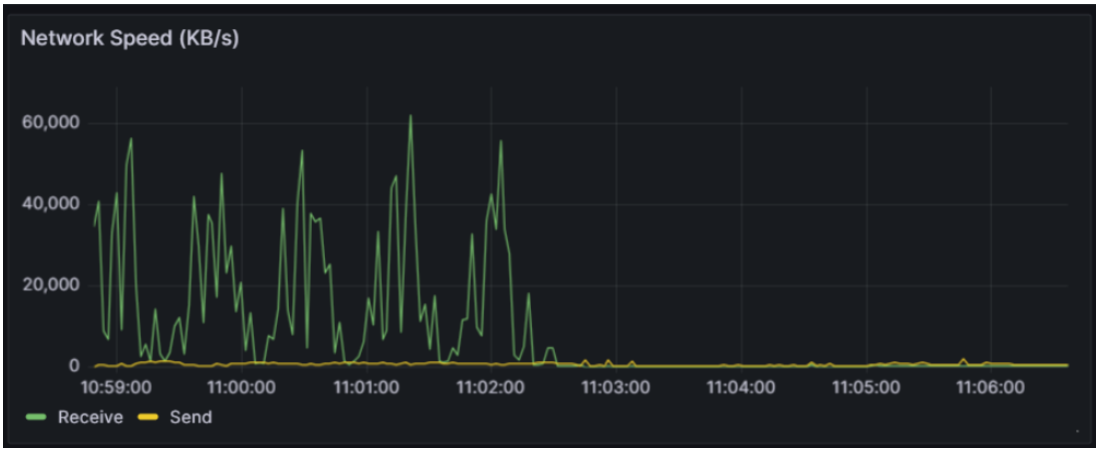


Network usage

MD Core #1



MD Core #2



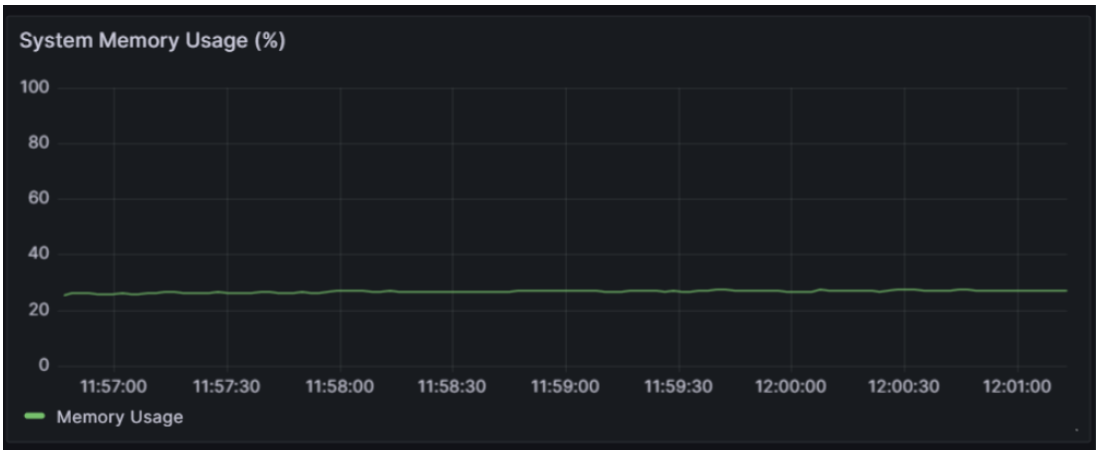
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

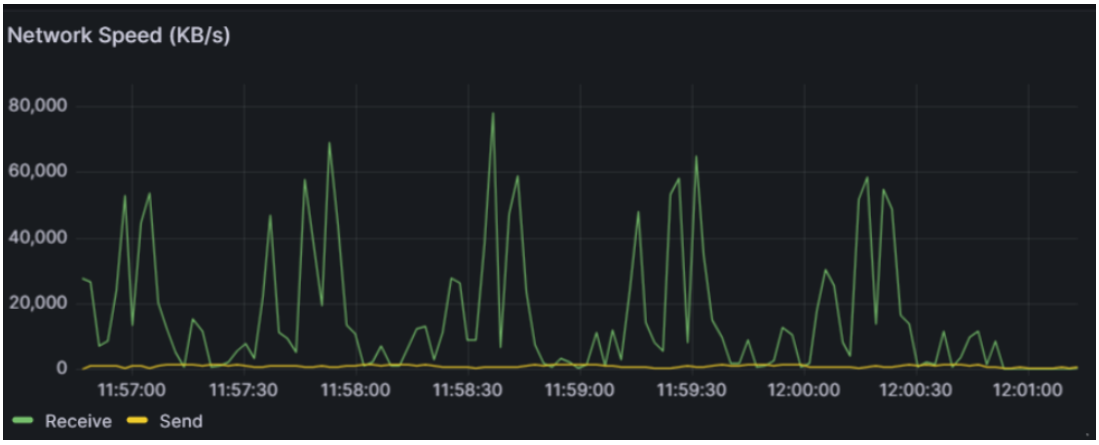


MD Core #2

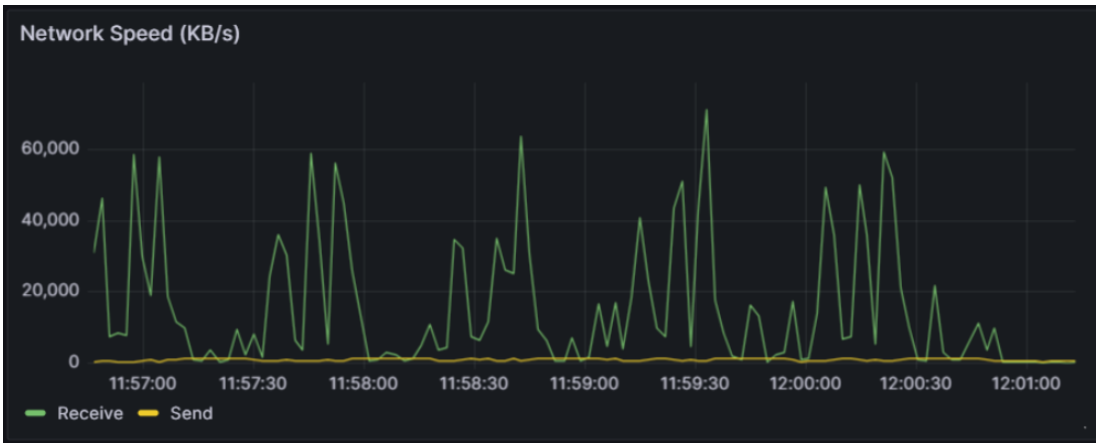


Network usage

MD Core #1



MD Core #2



Shared DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

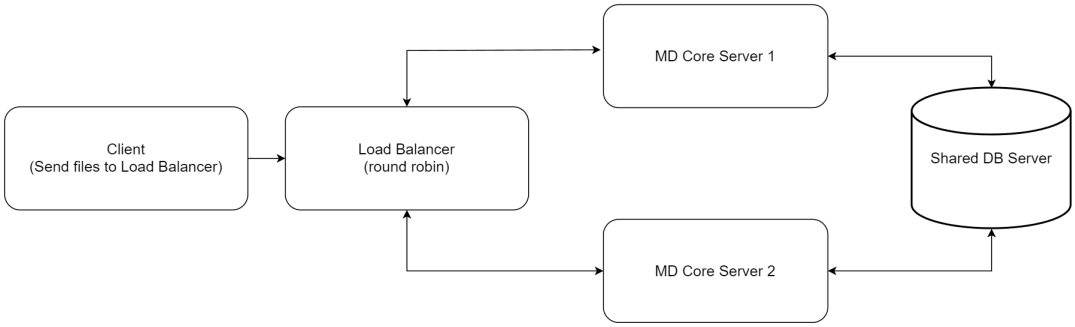
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV)	15.5	63.1	73.3	25.2	25.8	Receive: 4555.9 Send: 694.4	Receive: 4516.2 Send: 692.4	492,620.5	0.0
Deep CDR	9.70	50.7	51.2	31.9	32.5	Receive: 7062.9 Send: 623.2	Receive: 6653 Send: 654.1	786,189.6	0.0
Proactive DLP	7.2	35.2	35.0	27.9	27.5	Receive: 9047.4 Send: 802.4	Receive: 9859.8 Send: 787.5	1,055,110.8	0.0
Vulnerability	5.0	45.7	49.3	24.4	24.5	Receive: 13,585.7 Send: 966.7	Receive: 13,505.8 Send: 987.4	1,519,562.1	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV) + Deep CDR	19.9	79.1	77.6	34.0	34.2	Receive: 3785.9 Send: 573.8	Receive: 3663.1 Send: 577.7	382,716	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP	21.6	80.3	81.5	37.7	44.4	Receive: 3305.6 Send: 543.3	Receive: 3414.3 Send: 556.2	353,694.9	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	22.5	84.2	79.8	37.9	44.7	Receive: 3267.9 Send: 554.9	Receive: 3398.2 Send: 553.8	338,507.6	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

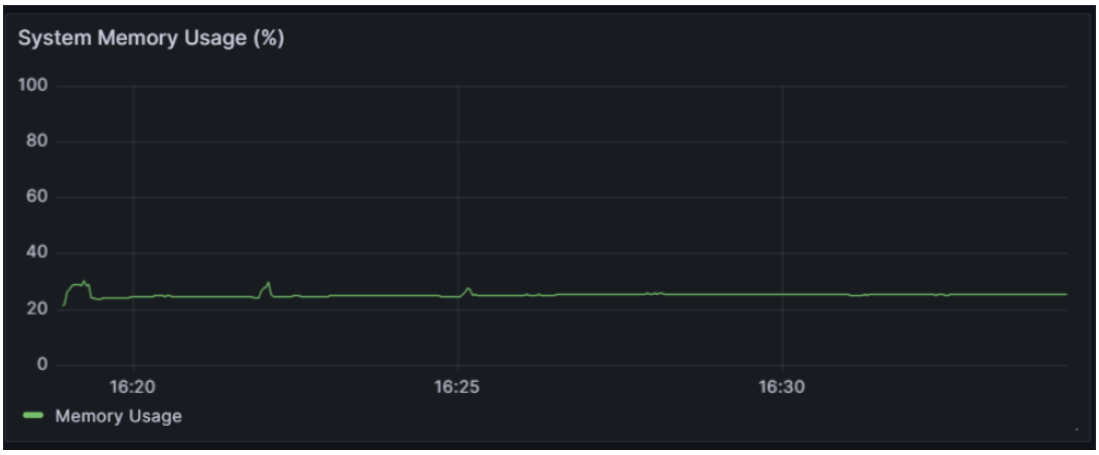
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

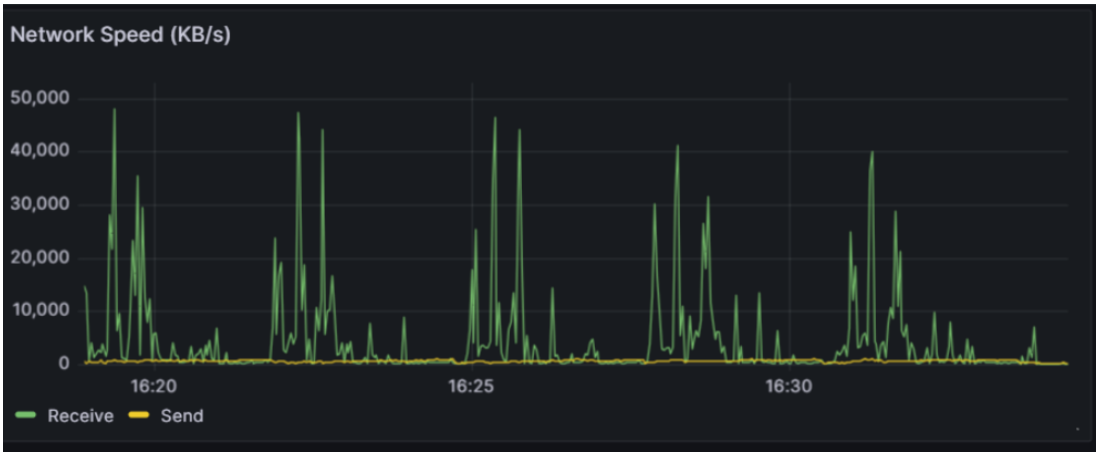


MD Core #2

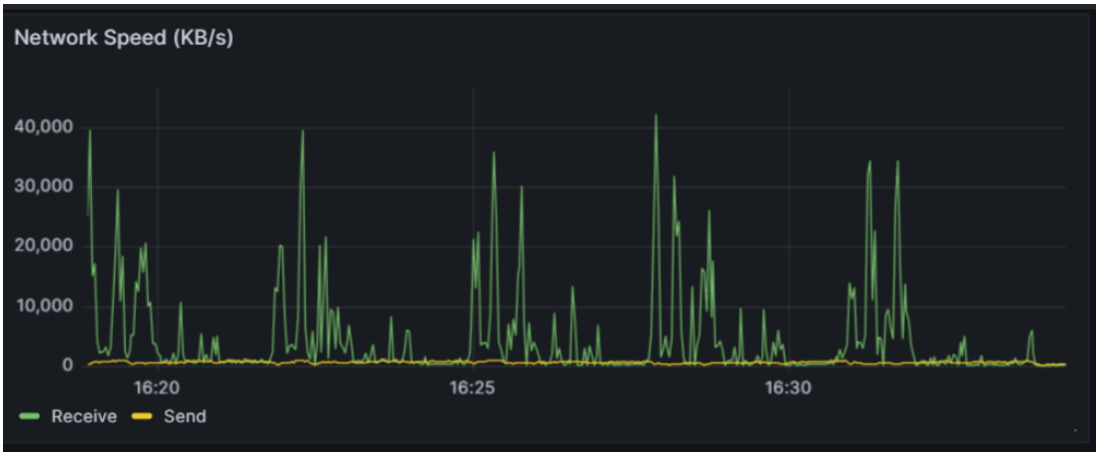


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

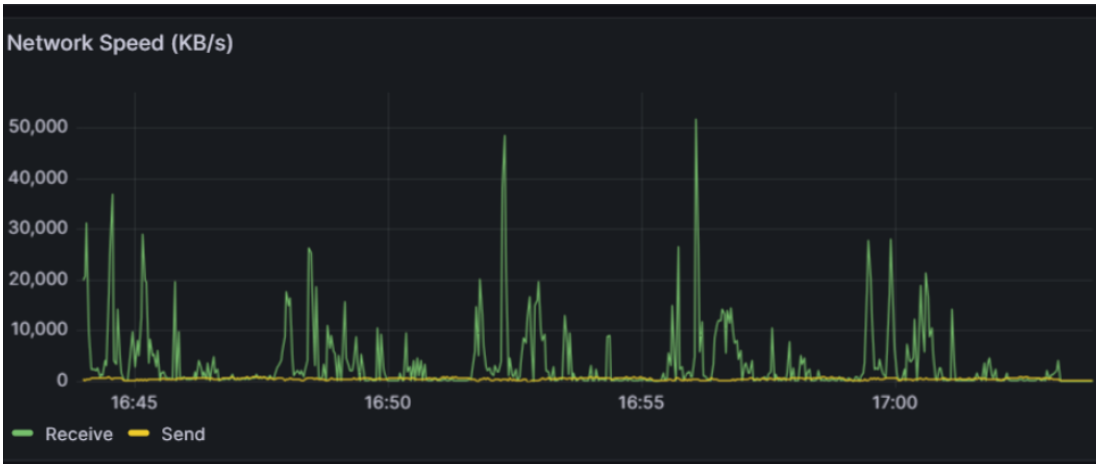


MD Core #2

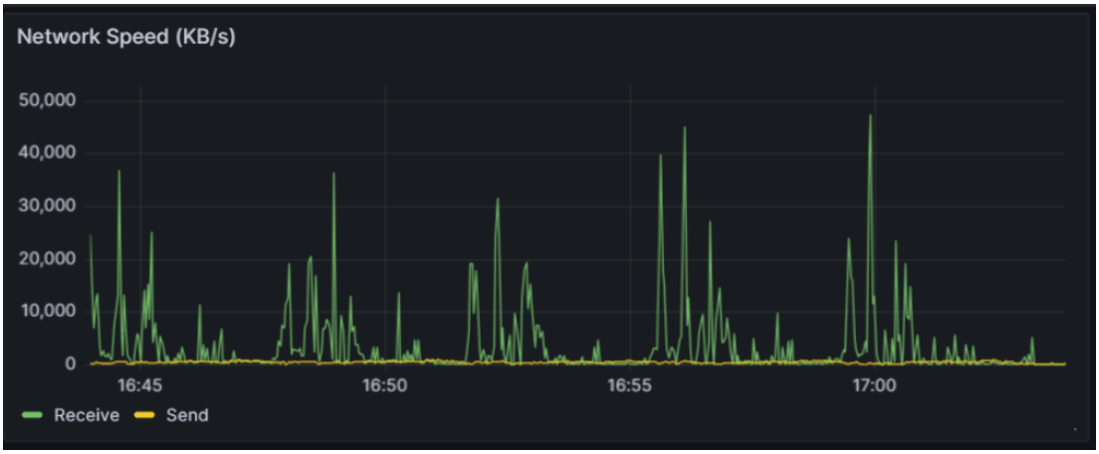


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1

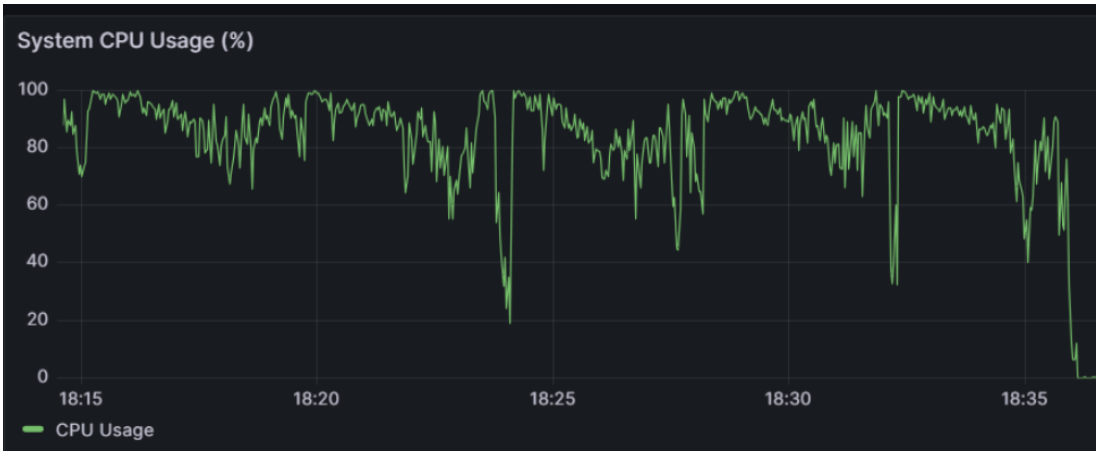


MD Core #2



CPU usage

MD Core #1

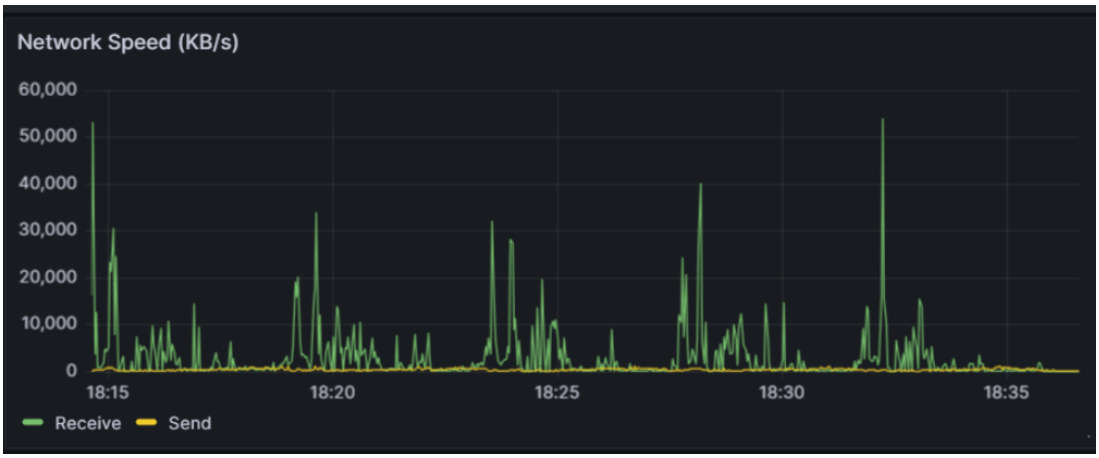


MD Core #2

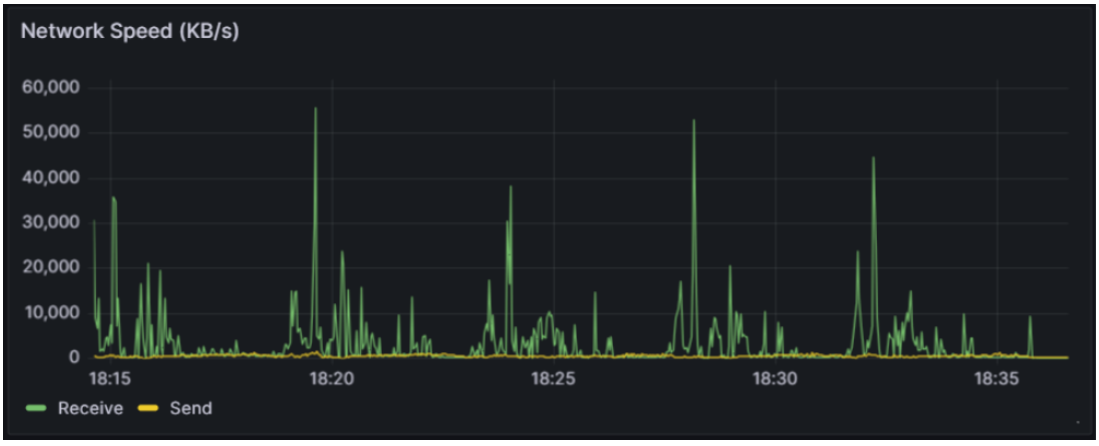


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

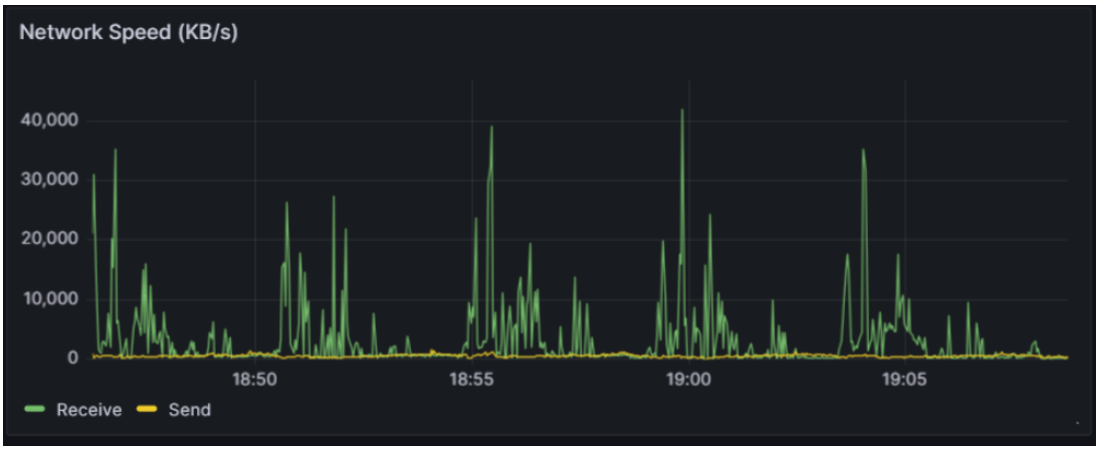


MD Core #2

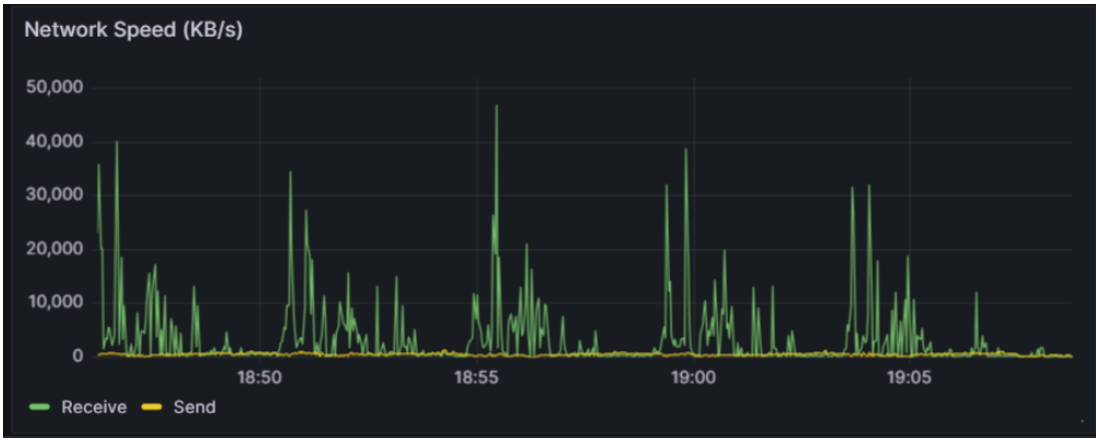


Network usage

MD Core #1



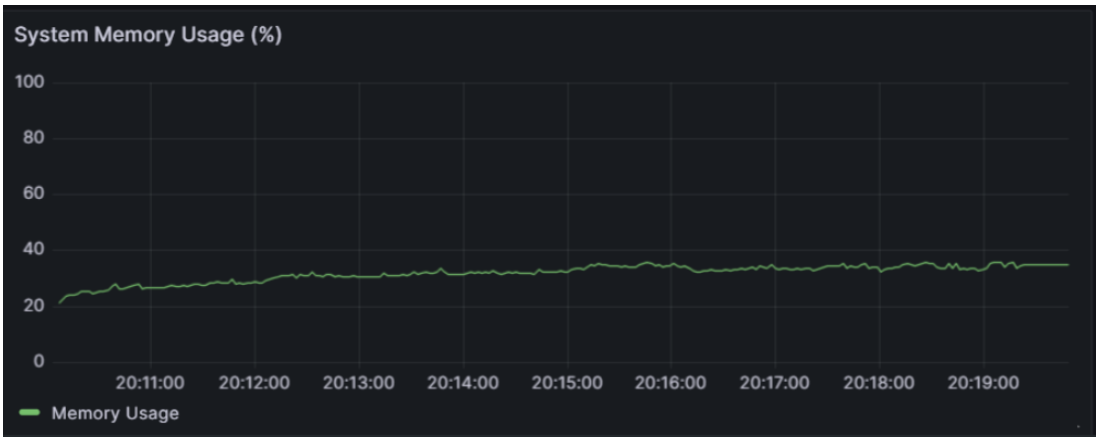
MD Core #2



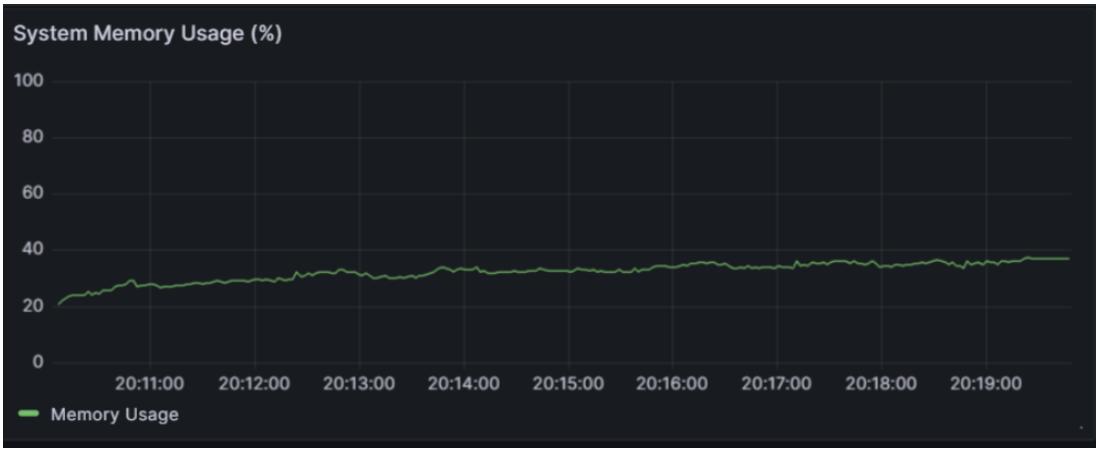
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1

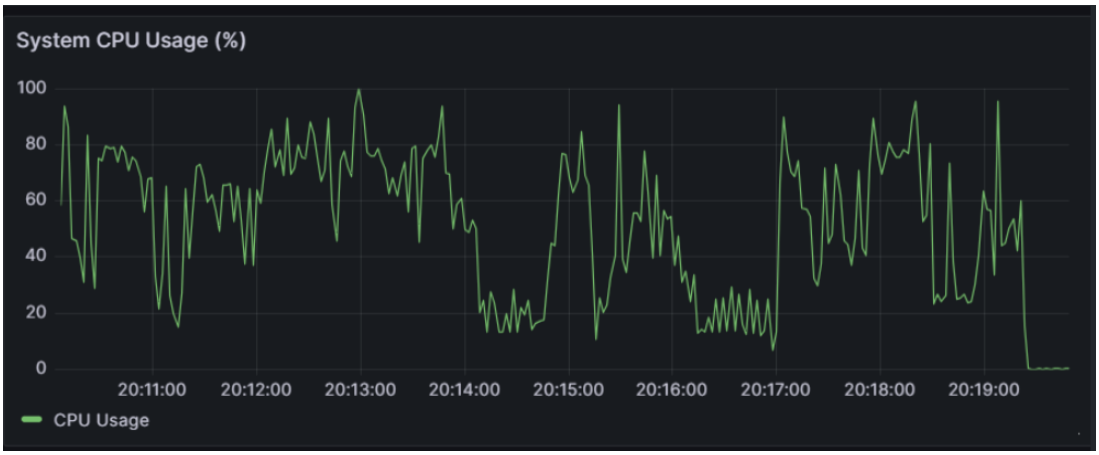


MD Core #2



CPU usage

MD Core #1

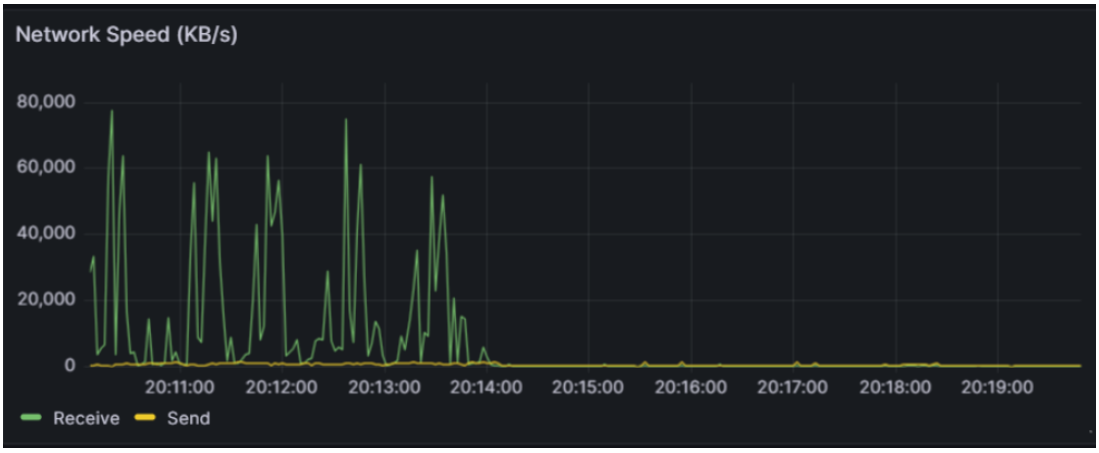


MD Core #2

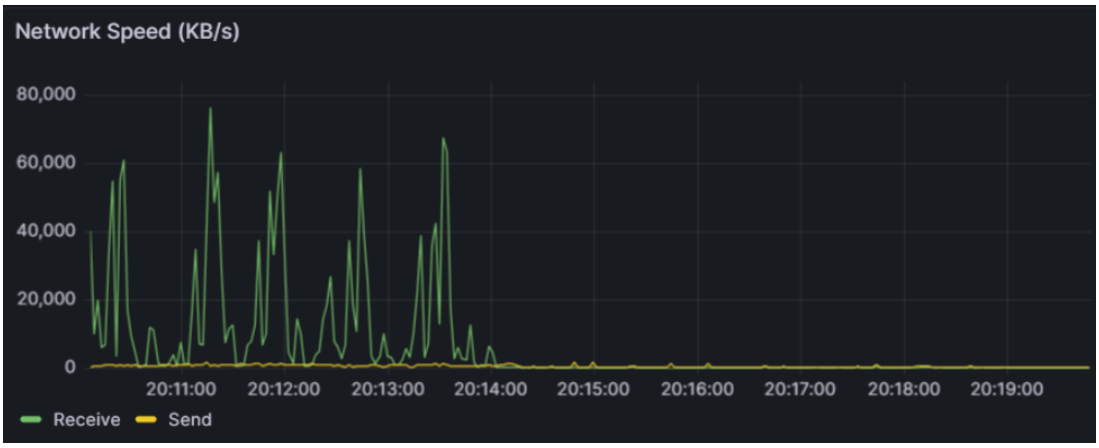


Network usage

MD Core #1



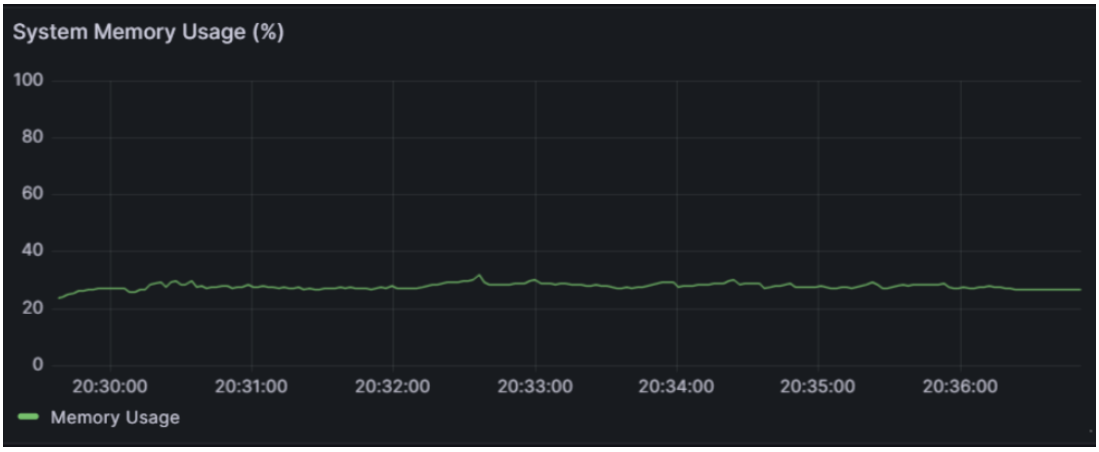
MD Core #2



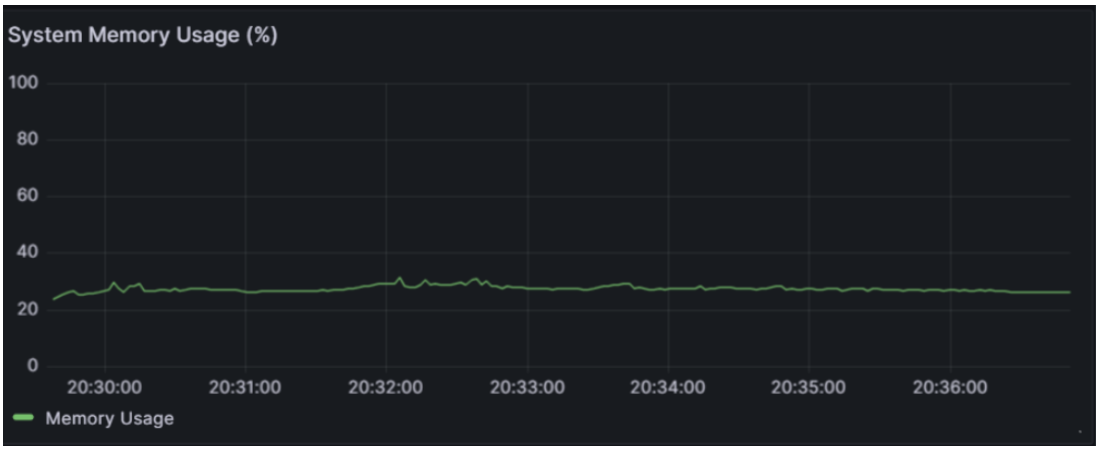
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1

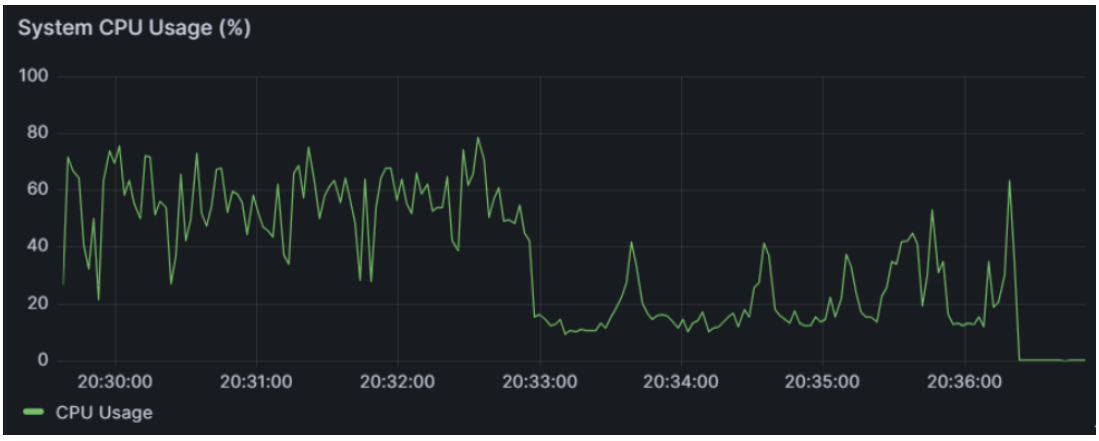


MD Core #2

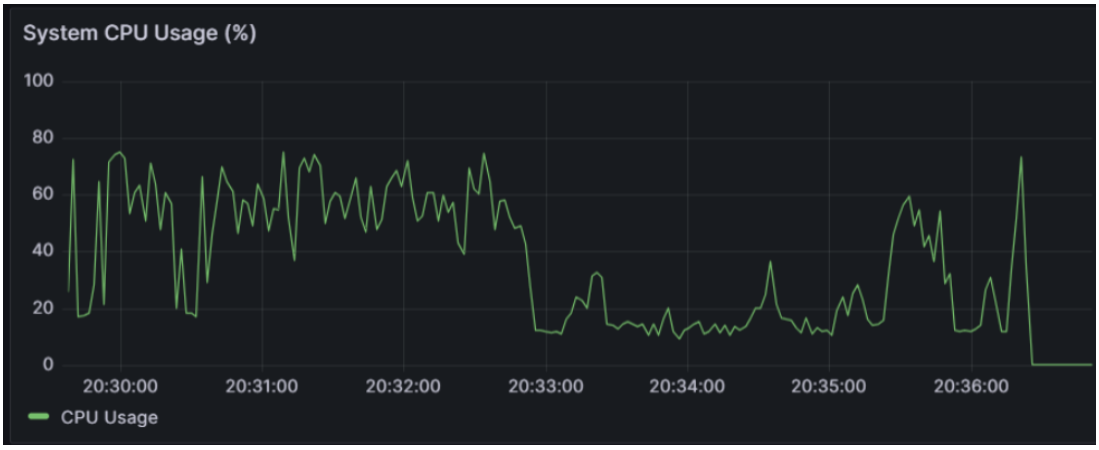


CPU usage

MD Core #1

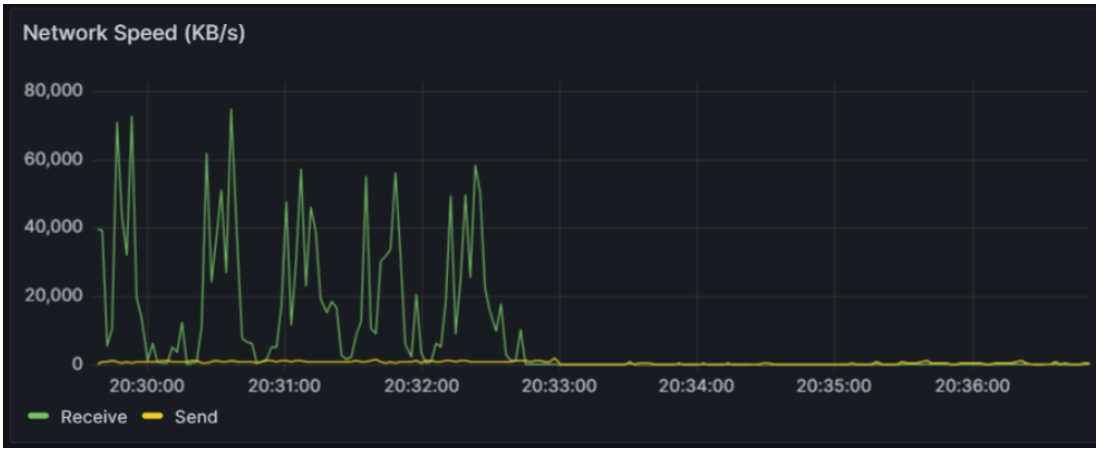


MD Core #2

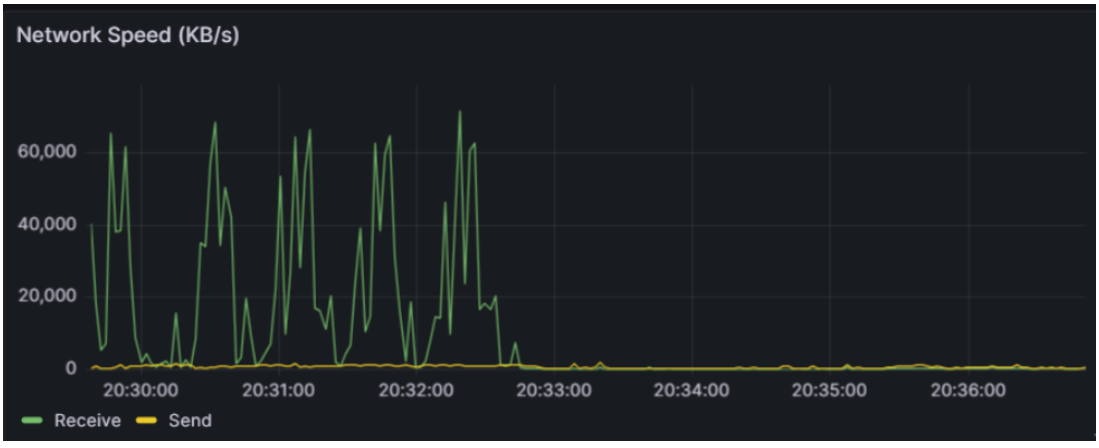


Network usage

MD Core #1



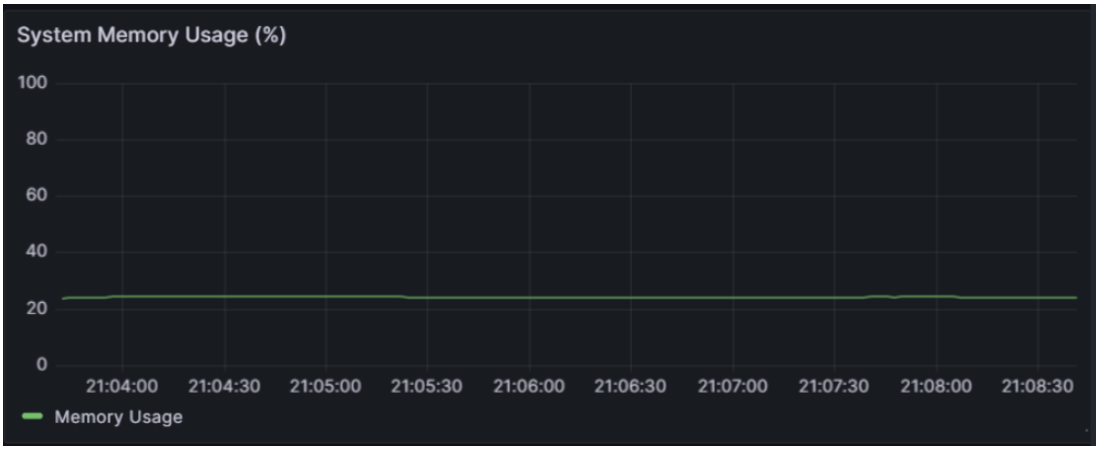
MD Core #2



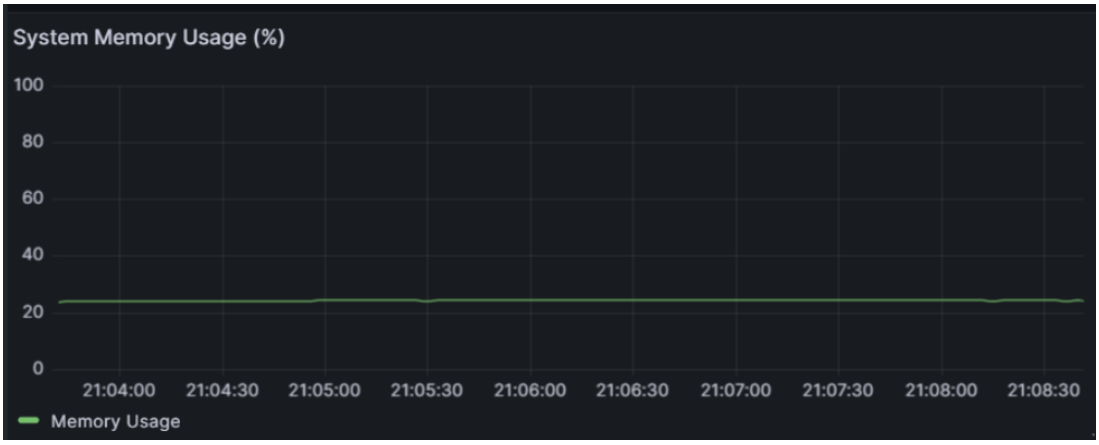
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

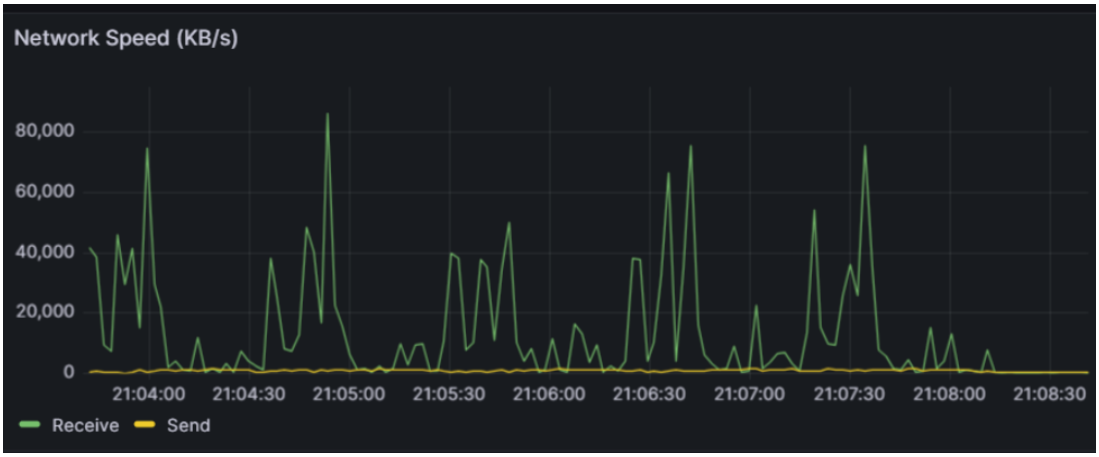


MD Core #2

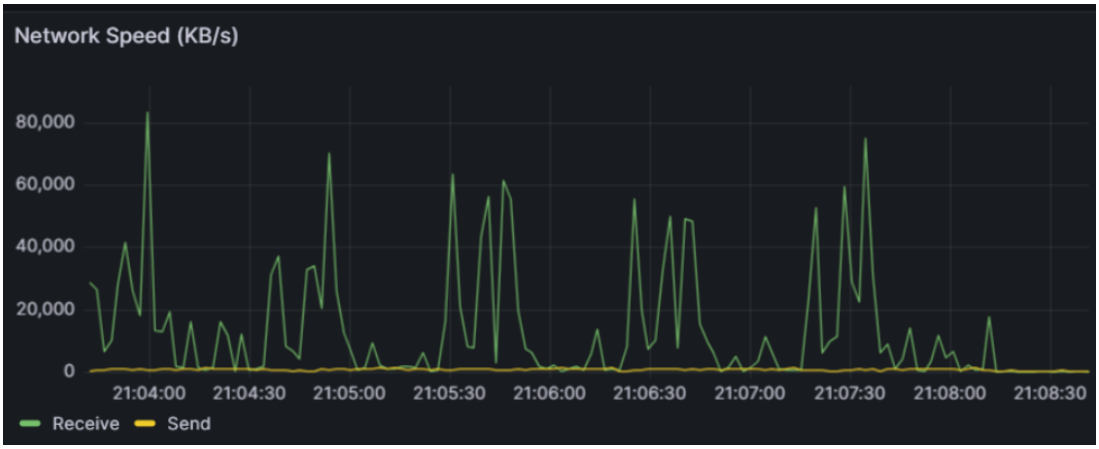


Network usage

MD Core #1



MD Core #2

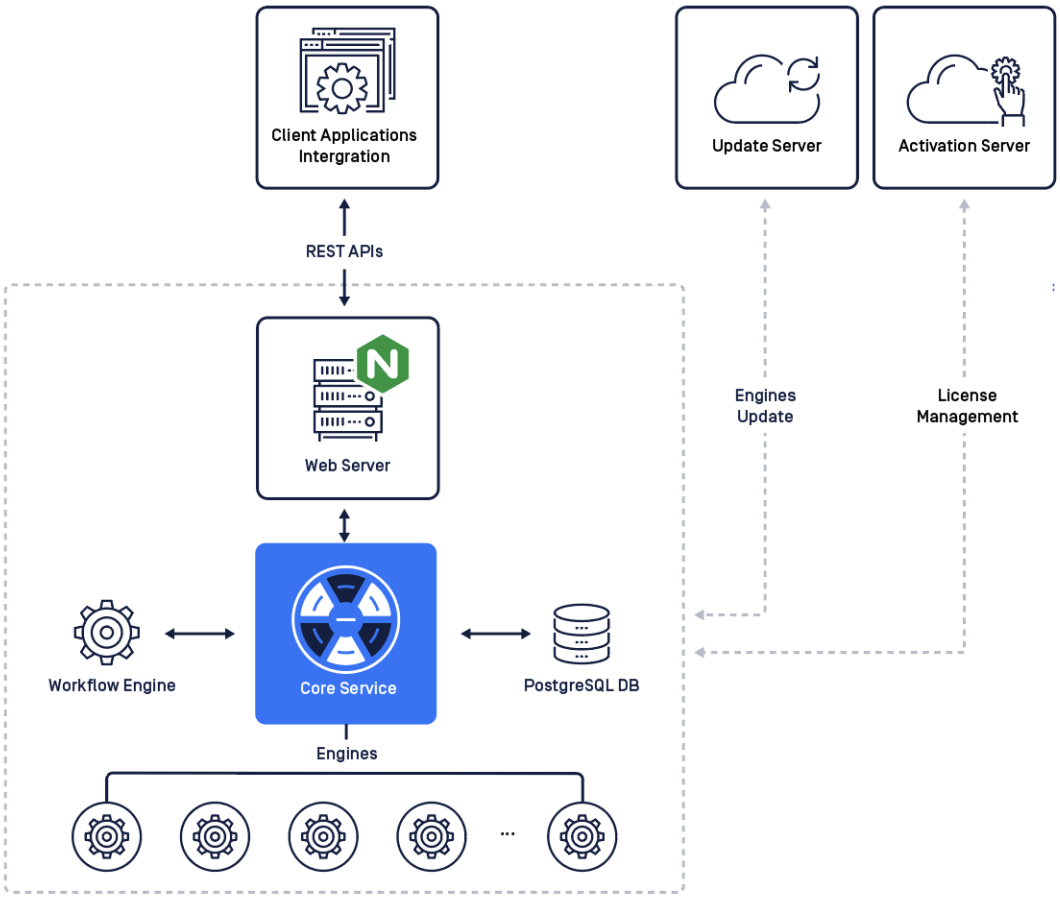

Network Speed (KB/s)
80,000
60,000
40,000
20,000
0
21:04:00
21:04:30
21:05:00
21:05:30
21:06:00
21:06:30
21:07:00
21:07:30
21:08:00
21:08:30
Receive
Send

MetaDefender Core Documentation

The users can consult this web page or, alternatively, they can download the manual in pdf format from the link below:

MetaDefender Core manual (SHA256:).

Recommended System Configuration



MetaDefender Core supports a wide range of deployment options, including air-gapped environments, providing the flexibility to choose the environment that best suits customers' requirements and infrastructure.

- Physical server.
- Virtualization platforms like VMware, Hyper-V and XenServer.
- Cloud Deployment.
- Container Deployment.

> **Note**
> Before installing MetaDefender Core, please refer to the **recommended minimum system configuration** listed below. The requirements are based on a standardized customer deployment and may require more or less resources to achieve the level of performance desired in your environment. Please note that the server specifications are built to allow a high volume daily processing. Please refer to Performance and Load Estimation for more details.

The recommendations below are for MetaDefender Core, API usage only. For any other use cases, please consult the user guide of the licensed products for accurate recommendations.

For certain use cases these might be adjusted and customized on specific needs and SLAs. We highly recommend to engage our ProServ team to assist in fine tuning MetaDefender and get the maximum performance out of your systems.

Microsoft Windows Deployments

[Windows] Supported Operating Systems

Only support 64-bit platforms.

- Windows 10, Windows 11 23H2, 11 IoT (22H2) only for End-User systems (e.g. Kiosk deployments)
  - Based on Microsoft License Terms, you're not allowed to use Windows Desktop as a server, therefore, MetaDefender Core can't be used on these systems for server use cases.
- Windows Server 2016, 2019, 2022

> **Disclaimer**
> End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

> **Plan for legacy OS support**
> OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:
>
> - Ubuntu 20.04 after **April 2025**.
> - Amazon Linux 2 after **June 2025**.
> - Windows 10 after **October 2025**.
>
> If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

[Windows] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 8	8	8 GB	16 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 12	16	16 GB	24 GB + 50 GB + 100 GB
MetaDefender Core 16	16	16 GB	32 GB + 50 GB + 100 GB
MetaDefender Core 20	32	16 GB	40 GB + 50 GB + 100 GB
MetaDefender Core MAX	32	32 GB	120 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core is installed.
- For a better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.
- SBOM processing requires more disk space, please allow at least 15GB free disk for the engine to work properly.

[Windows] Third Party Dependencies

- .NET framework 4.5 or above
- Microsoft Visual C++ Redistributable for Visual Studio 2015-2022 (version 14.38 or higher); x64 version.
- PostgreSQL: Only applicable when using a pre-installed PostgreSQL server running remotely.
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Some engines also have dependencies as described below:

Engine Name	Dependency
Deep CDR	Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 5.8.0 or above)
ESET	MetaDefender Core v4 temporary directory should have more than 200MB free disk space
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
Proactive DLP	• Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 2.0 or above) • .NET 6 dependencies • CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
Scrutiny	Microsoft Visual C++ 2015 - 2019 Redistributable x86 and x64
Systweak	.NET runtime 6.0, 7.0 or 8.0
Bkav Pro	Microsoft Visual C++ 2015 - 2022 Redistributable x64

[Windows] Installation Details

MetaDefender Core on Windows uses "C:\Program Files\OPSWAT" folder for storing resources or the installation directory.

MetaDefender will use its resources folder to store temp files as part of the analysis. **It's recommended to exclude this folder from real-time protection monitoring**. Knowledge base:

- How to set up exclusions on anti-malware software to prevent disruption
- Mitigating scan failure triggered by real-time protection service
- Can local AVs interrupt ongoing scans?

Linux Deployments

[Linux] Supported Operating Systems

Only support 64-bit platforms.

- Red Hat Enterprise Linux 8, 9
- Rocky Linux 9
- Debian 11, 12
- Ubuntu 20.04, 22.04
- Amazon Linux 2 is supported via Amazon Machine Image (AMI)

Plan for legacy OS support

OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:

- Ubuntu 20.04 after **April 2025**.
- Amazon Linux 2 after **June 2025**.
- Windows 10 after **October 2025**.

If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

For Hardended Linux OS

Some AV engines could be failed to initialize and run on MetaDefender Core (e.g. ESET, Kaspersky) due to execution permission required by them on `/var/tmp/ometascan` folder.

Please follow steps at Why have the ESET and Kaspersky scan engines failed to initialize on the hardened Linux OS? for remediation details.

Disclaimer

End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

[Linux] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 5	4	4 GB	10 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 10	8	8 GB	20 GB + 50 GB + 100 GB
MetaDefender Core MAX	16	16 GB	40 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core resides on.
- For better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.

[Linux] Third Party Dependencies

Dependencies list:

- openssl
- grep
- lib32stdc++6 (>= 4.5)
- libc6-i386 (>= 2.10)
- procps
- zlib1g
- libcurl3 (>= 7.19.7)
- libcurl4
- postgresql:
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Not all the above dependencies will need to be installed, it is dependent on different Unix distro and version

Some engines also have dependencies as described below:

Engine name	Dependency
Archive engine	• To extract password protected MS Office files, you need to install .NET 6 dependencies, libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • Rocky Linux 9.x/Red Hat 9.x: libnsl
Deep CDR	• Rocky Linux 9.x: libgomp, ncurses-compat-libs, libnsl • CentOS: libgomp, ncurses-compat-libs • Red Hat 8.x: libgomp, ncurses-compat-libs, libnsl • Red Hat 9.x: libgomp, ncurses-compat-libs, libnsl, compat-openssl11 • Ubuntu/Debian: libgomp1, libncurses5 • libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • .NET 6 dependencies
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
FileType engine	Rocky Linux 9.x/Red Hat 9.x: libnsl
Proactive DLP	libgdiplus, mscorefonts .NET 6 dependencies CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
RocketCyber	libgomp

Note: Some dependencies may need to be installed from external repositories, not from the OS default repositories.

[Linux] Installation Details

MetaDefender Core default installation path is using /var folder for storing resources:

- `/var/lib/ometascan` : installation folder with all its resources (database, DLP processed / CDR sanitized / quarantined file storage, engine package and definition updates).
- `/var/log/ometascan` : application logs.
- `/var/run/ometascan` : runtime folder for Core related processes.
- `/var/lib/ometascan` : all successful deployed engines data files.
- `/usr/lib/ometascan` : all shared libraries required by MetaDefender Core, bundled PostgreSQL server related binaries and libraries.
- `/var/tmp/ometascan/resources` : all temporary processing files for MetaDefender Core to process.

Storage Usage Information

Based on the configuration, MetaDefender Core will need additional disk space to store analysis/result data, quarantined files and temporary files:

	Data Storage
Analysis Reports (Processing History)	Approximately 4.5GB is required for every 1M analysis reports stored. • 100 GB is recommended as a baseline. Database storage requirements are affected by Data Retention settings, engine settings and scanning load.
Quarantined Files	Depends on the customers' dataset and configuration
Sanitized Files	Depends on the customers' dataset and configuration
Temporary processing files	Depends on the customers' dataset and configuration • A 1GB non-archive file requires 1GB free disk space at least. • A 1GB archive file requires (1 * 2)GB free disk space at least. • 50GB is a baseline and should be adjusted based on observation of the solution within the environment, with consideration to peak times and overall file makeup (e.g. complexity, size, archives).
Note	Options are available to control how large and/or complex files are handled to limit the amount of system resources required. See Workflow Configuration Template for more information.

Custom Engines

The recommendations above are specific for MetaDefender pre-packaged bundles.

However for additional Custom Engines, please review the Knowledge base to review additional requirements (if any) for the selected engine.

Browser Requirements for the MetaDefender Core Management Console

One of the following browsers is suggested to view the MetaDefender Core Management Console:

- Microsoft Edge
- Chrome
- Firefox
- Safari

Chrome, Firefox, Safari and Microsoft Edge browsers are tested with the latest available version at the time of release.

Network Bandwidth

In online environment where MetaDefender Core is configured to download engine update packages directly from OPSWAT update server source, then using gigabit network speed (125 MB/sec) is highly recommended to avoid potential network related issue while downloading files, or at least 100 Mbps (12.5 MB/sec) network speed.

Allowlisting URLs

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts':

- https://activation.dl.opswat.com - for license online activation
- https://update.dl.opswat.com - for engines and databases download

Note: *IP address-based allowlisting on your firewall might fail after some time since OPSWAT uses CDN (AWS Cloudfront) to faster delivery updates over the world, and IP address of edge servers might change over time.*

Depending on your location, connections with OPSWAT cloud-based servers be forwarded to specific AWS Cloudfront's edge locations, each is assigned with its own unique IP address ranges. You might also need to allow correct AWS Cloudfront's corresponding IP address ranges by referring to https://ip-ranges.amazonaws.com/ip-ranges.json

Warning

Even the OPSWAT update servers host updates for all of the available engines we support, sometimes custom engines might try to connect to their own cloud for updates, but this can be disabled in firewall and they will be updated just from OPSWAT cloud servers instead.

Differences Between MetaDefender Core v4 and v5

We are very excited to inform you about the release of version 5.0.0 of our MetaDefender Core!

The new version is designed to enhance the product performance, user experience, as well as to provide more flexibility, security, and scalability for our customers.

At the same time, we are also committed to keeping backward compatibility for whom running MetaDefender Core v4, and avoid the need for updating your client integration to move forward with MetaDefender Core v5.

v4 REST API Integration Supported
All public REST API endpoints available on MetaDefender Core v4 will be continously supported on v5.

v4 Product Major Features Supported
All product major features available on MetaDefender Core 4.21.2 (the latest one on v4) will be continously supported on v5.

Seamless Product Upgrade
Upgrading from MetaDefender Core v4 to v5 will be supported and covered with data migration without additional manual steps from users.

Licensing information will be also migrated.

Below are some highlights of the first v5 release, MetaDefender Core 5.0.0:

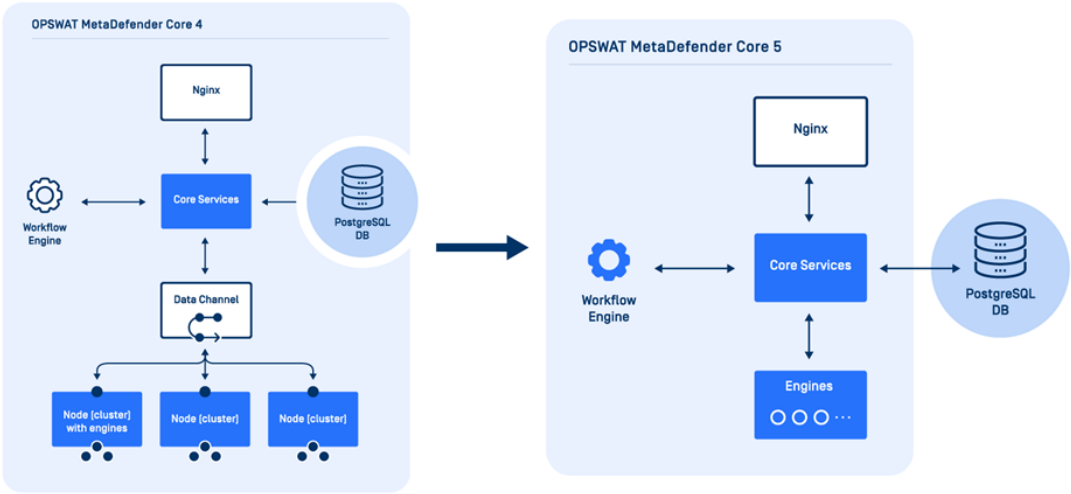
New product architecture

This version introduces a simpler product architecture that speeds up integration and automation, file processing flow, and immensely enhance the product performance and efficiency.

Earlier versions utilized MetaDefender Node concept, designed to support scaling of the scanning infrastructure by distributing scan requests among several different processing Nodes. Product licensing was based on number of Node instances. The benefit of having such a distributed infrastructure is that based on Node loads, Metadefender Core server can always choose the most appropriate Node to assign a new scan task to. However, having nodes in the v4 architecture caused more complexity and possible latency in the processing flow and entailed maintenance cost.

In this release, we retired multi-Node deployment and restructured MetaDefender Core's backend. This alteration significantly improves file processing speed and product efficiency, as well as eliminates extra cost for maintaining node services. Users now can simply install additional MetaDefender Core instances for scalability requirement,

as well as configure their own load balancer to coordinate tasks among multiple MetaDefender Core instances. Accordingly, product licensing is now based on number of the product instances.



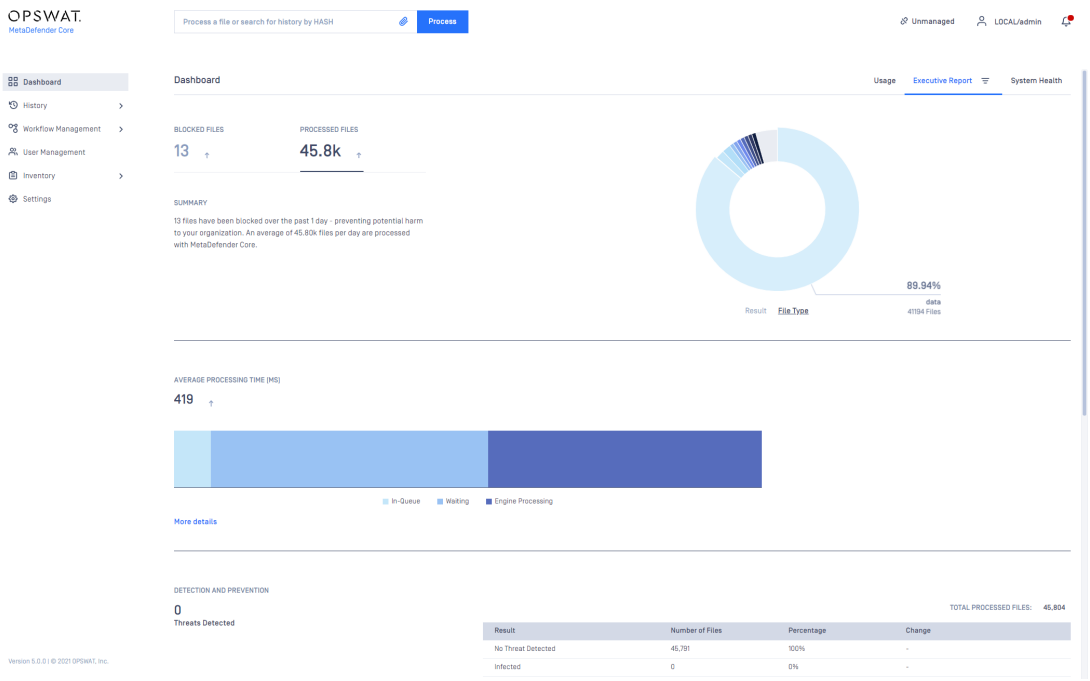
Please be informed that

- All existing administrative APIs related to "Node" on v4 will be kept as it is on v5 to ensure backward compatibility with existing client integrations.
- Applicable settings defined in Windows registry (Windows) or ometascan-node.conf config file (Linux) for Node will be migrated to Core. Details: MetaDefender Configuration

Brand-new product user interface

The second notable update in this version is a brand-new UI for the management console, which was designed with the focus on the quality and thoughtfulness of the user experience. The new administrative console UI provides an advanced set of Core features and observable processing status.

Comprehensive executive dashboard reporting offers dynamic, real-time and actionable threat insights. It also provides visibility of your security posture allowing you to paint a picture of security that aligns with the overall business. The report includes easy-to-read charts, graphs, and statistic explanation that help your security professionals reduce clutter, gather information quicker, spot threat activity early and act swiftly.



We're continuing to improve our product UI to provide users an excellent experience with the application and overall usability. Please expect more powerful UI features coming in next releases.

New compression archive engine for archive sanitization

In previous version, MetaDefender Core's archive engine implemented four functions including decryption, extraction, encryption and compression that with a high volume of files it takes time to process them all. For the purpose of enhancing archive handling performance, this release comes with a separated compression archive engine, which dedicatedly processes archive sanitization (compression and encryption). It is also created to facilitate maintenance and further feature development.

Archive sanitization feature now requires Deep CDR license

As to respect sanitization practice and ensure unknown threats prevention as designed, now starting on version 5, it's mandatory to have Deep CDR licensed for archive sanitization feature enablement on MetaDefender Core workflow configurations.

Archive handing related engines changed

The legacy "Archive" engine on MetaDefender Core v4 will be changed its name to "Archive Extraction" engine, even when you are still on MetaDefender Core v4.

A brand new "compression archive engine" is to dedicatedly handle archive sanitization in v5 (archive compression), that helps relevant archive handling performance improvement, easier for maintainance and further feature development.

Notes when you are still on MetaDefender Core v4 (not upgraded yet)

The legacy "Archive" engine (now name changed to "Archive Extraction" engine) on MetaDefender Core v4.x will still cover both extraction and compression archive tasks as usual.

Even though your MetaDefender Core v4.x might now have a new "Archive Compression" engine auto downloaded, but that engine cannot be deployed (marked as "permanently failed" in relevant engine status check API). This behavior is expected, and not causing any impact to MetaDefender Core processing flow, or its performance though.

- Just in case your client integration has a check for engine status before triggering scan request, then you might want to disable "Archive Compression" engine status on MetaDefender Core v4 using REST API. Detailed instruction: Archive Compression Engine Permanently Failed on MetaDefender Core v4
- When you later upgrade to MetaDefender Core v5 from v4, then you might need to manually re-enable "Archive Compression" engine to make it active and running on MetaDefender Core v5.

Notes for licensing

For offline deployment (MetaDefender Core is not connected to the Internet directly for online update), you will need to re-activate again on OPSWAT portal to retrieve latest engine list in .yml file including "Archive Compression" engine if you are currently licensed with Deep CDR.

Meanwhile, for online deployment, this manual procedure is not required since MetaDefender Core will cover that itself.

Import & export configurations

Not only workflow settings like in previous versions, but also all other general settings (update, security, data retention, server, hashes to skip) and most importantly, user settings, will be able to export and import on MetaDefender Core version 5.

Learn more at Import/Export configuration

Backward compatibility supported

The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

Windows application event logging disabled by default

Applicable only to Windows OS based MetaDefender Core.

Help avoid duplicated logging with separate file option, and improve overall processing performance. Of course, when being still valid to you, you are still supported to enable it back.

CSRF token header renamed

CSRF token originally created for security strengthening and being used by MetaDefender Core management console UI since version 4.21.0 via header `oms_csrf_token`

To get it compliant with some load balancers which do not accept underscore in header name by default, and save you from additional load balancer configuration step, we have now changed the header name to `oms-csrf-token`

This change does not impact to any REST API based client integration.

More comprehensive and accurate error messages on REST APIs

Impacting on following REST API endpoints:

- `PUT /admin/config/session`
- `PUT /admin/config/update`
- `PUT /admin/config/scan`
- `PUT /admin/config/skip`
- `PUT /admin/config/auditlog`
- `PUT /admin/config/webhook`
- `PUT /admin/config/rule`

Upload time and queue time logic updated in JSON scan response

Impacting on following REST API endpoints:

- `GET /file/{data_id}`
- `GET /hash/{md5/sha1/sha256}`
- `GET /archive/{data_id}`
- `POST /file/{async}`

Added new field:

- `receive_data_timestamp`: The timestamp when upload progress started (first byte received) (in milliseconds)
- `upload_time`: Total time elapsed for upload process (in milliseconds)

Modified existing field (new logic):

> **Backward compatibility supported**
> The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

- `upload_timestamp`: The timestamp when upload progress finished (all bytes received) (in milliseconds)
- `queue_time`: Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue time = start_time - upload_ timestamp) (in milliseconds)

Release version 5.0.0 also comes with variety of outstanding features and improvements.

For detailed information about our latest release please view our Release notes

Installation

Starting with MetaDefender Core 4.20.0, the centralized/shared database deployment model is supported (all MetaDefender Core instances share the same database), along with traditional standalone database mode (every MetaDefender Core instance has its own database to manage).

- Standalone mode
- Shared Database Mode

Installation

1. Download the package of your choice from My OPSWAT
2. Install the package on your computer via the Command Line or via the Wizard
3. Open a web browser and point to http://<server name or IP>:<port>
    - The default port is **8008**
    - In case of problem check Inaccessible Management Console page
4. Complete the required steps of the basic configuration wizard
5. You must activate this deployment to use its features

Installation notes

- If the MetaDefender Core package **dependencies are not installed** on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.
- During installation the databases might need to be upgraded. This could take noticeable time depending on database size (eg.: length of scan history).

Standalone mode

Wizard Installation

The Install Wizard is only for the Windows installer (.msi file).

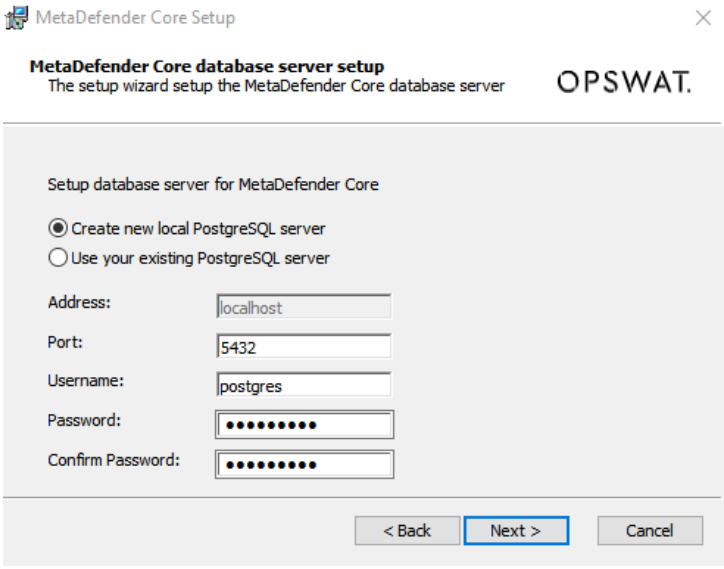
Since MetaDefender Core 4.19.0, we have switched to using PostgreSQL DBMS which needs a dedicated database server. MetaDefender Core supports two modes to setup PostgreSQL server:

Local (Bundle) PostgreSQL server

MetaDefender Core will install a new PostgreSQL server locally in the same box with the product.

> **PostgreSQL user credentials**
> Unicode characters are NOT supported for "Username" and "Password"
>
> "Password" does not accept # (hash) as starting character (e.g. #ABC)

> **The ignition file**
> If the ignition file exist in the designated location, then MetaDefender Core will use the details defined in the ignition file, rather than what is passed via the installation wizard.
>
> More info about the ignition file based installation: Command Line Installation

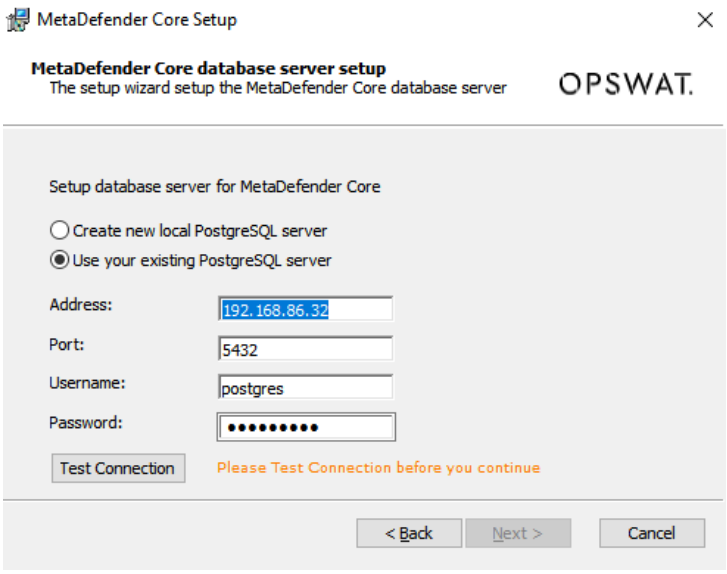


Remote (Existing) PostgreSQL server

MetaDefender Core will connect to remote pre-installed PostgreSQL running remotely.

> **PostgreSQL connection configuration**
> "Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

> **PostgreSQL user privilege requirements**
> MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.
>
> Details: PostgreSQL User Privilege Requirements



Command Line Installation

Ignition File

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Make sure to create the ignition file, if it doesn't exists before you go ahead and install MetaDefender Core via command line.

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

MetaDefender Core supports two modes to setup PostgreSQL server:

1) Local PostgreSQL server

- MetaDefender Core will install a new PostgreSQL server locally, next to the product.
  - A sample iginition file for PostgreSQL server information (***Only non-Unicode characters supported for "user" and "password"__***):

ometascan.conf bash

```
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
```

2) Remote PostgreSQL server

> **Info**
>
> **PostgreSQL version 12 (at least 12.3) or version 14 is required.**
>
> According to PostgreSQL notice, version 12 will stop receiving fixes on November 14, 2024, so we suggest that you use version 14.

MetaDefender Core will connect to a pre-installed PostgreSQL server running remotely and will setup its database.

Make sure to follow steps at Open Connection On PostgreSQL Server first, so that your remote existing PostgreSQL server to accept upcoming connection from MetaDefender Core.

A sample ignition file for PostgreSQL server information:

ometascan.conf bash

```
[dbserver]
type=remote
host=192.168.86.32
port=5432
user=existingserver_admin_user
password=existing_server_pass
```

After installing MetaDefender Core successfully, you may want to remove all credentials info created for [dbserver] section in ignition file for security reason.

> **Warning**
> If the MetaDefender Core package dependencies are not installed on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.

Debian package (.deb)

bash

```
sudo dpkg -i <filename> || sudo apt-get install -f
```

On Red Hat Enterprise 7.x Linux / CentOS package (.rpm)

bash

```
sudo yum install <filename>
```

For systems which enabled GPD check flag:

bash

```
sudo yum install --nogpgcheck <filename>
```

Red Hat Enterprise Linux 8.x package (.rpm)

Install the latest PostgreSQL repository RPM:

> **Note**
> Due to the shorter support cycle on Red Hat Enterprise Linux, all supported versions of PostgreSQL are not available on this platform. The PostgreSQL Yum Repository will integrate with the normal systems and patch management, and provide automatic updates for all supported versions of PostgreSQL throughout the supported lifetime of PostgreSQL. Reference: https://www.postgresql.org/download/linux/redhat/

bash

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

In case installing a remote PostgreSQL on either the same MetaDefender Core server or another server, you need to install the following additional package. PostgreSQL version 12.3 or newer is recommended.

bash

```
sudo dnf install -y postgresql12-contrib
```

Install MetaDefender Core:

bash

```
sudo dnf install -y <filename>
```

Windows package (.msi)

On Windows systems it is possible to install the product by running the corresponding .msi file.

From command line interface it is also possible to install the product by executing

bash

```
msiexec /qn /i <msi file name> <option key>=<option value>
```

where the possible keys and their default values are the following:

Key	Default Value	Description
INSTALLFOLDER	\Program Files\OPSWAT\MetaDefender Core	Customize installation folder for product Example: `INSTALLFOLDER="D:\Products"`
RESTADDRESS	*	REST interface binding IPv4 or IPv6 address ('*' means that service listens on all IPv4 and IPv6 interfaces)
RESTPORT	8008	REST interface binding port
REPORT_ENGINE_ISSUE	true	Enable reporting of engine issue count. (possible values: "true" or "false").

Note

Reporting of engine issue count

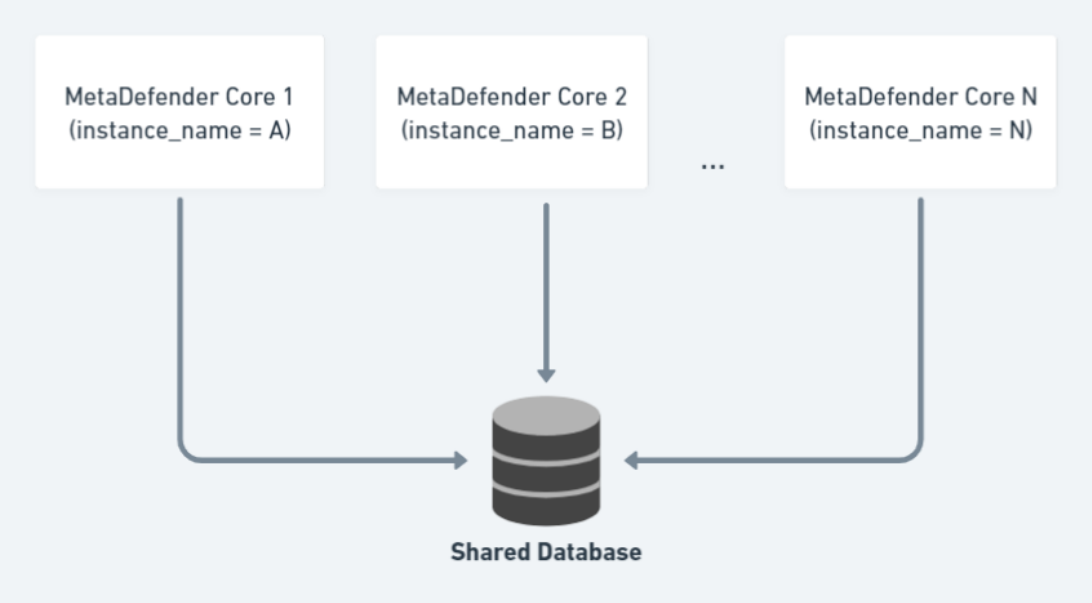
If reporting of engine issue count is enabled, MetaDefender Core server will send only the **number** of initialization errors and **number** of unexpected stops for the specific db/engine version. This information is sent over a HTTPS channel when the product downloads the latest package descriptors. This information is used for early detection of any specific 3rd party engine quality issues.

For details on using msiexec please consult Windows installer documentation.

Shared Database Mode

Starting with MetaDefender 4.20.0, you can use a centralized database deployment model, where all MetaDefender Core instances can share the same external database.

That helps auto-sync all product configurations (**except Single-Sign-On** that requires unique settings for each Core), processing history, live/on-going scan result query, audit, statistics data.



This article demonstrates how to install and setup MetaDefender Core 4.20.0 or newer with shared database mode.

Important notes & current limitations:

Currently not supported

- Does not support Central Management use-case with shared DB mode.
- Does not support to switch from shared database mode back to standalone database mode.
- Does not support to change instance_name of MetaDefender Core in shared database deployment model after installation.
- Does not support upgrading older MetaDefender Core from non-persistent mode to using the shared database mode.
- Does not support upgrading older MetaDefender Core from standalone mode to using the shared database mode.
- Does not support to upgrade multiple MetaDefender Core instances at once in shared database mode
    - Stop all Core instances' scanning service before performing the upgrade.
    - Upgrade each Core server one by one instead.

- For each Core, when the upgrade finished, then you can start using that Core for scanning without waiting for the other Core instances' upgrade to be done.

> **Disclaimer**
> MetaDefender Core does not support mode switching.

Requirements for shared database setup:

- The central remote PostgreSQL server (for shared database) could be utilized with a built-in (native) PostgreSQL database server installed with one of the MetaDefender Core instances in the network. Or using a dedicated existing PostgreSQL database server (see detailed requirements for PostgreSQL version at Recommended System Configuration).
- Before setting up, make sure to follow steps at Open Connection On PostgreSQL Server so that all MetaDefender Core instances could connect to the central remote PostgreSQL server.
- MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core, details: PostgreSQL User Privilege Requirements
- All MetaDefender Core instances must be installed on the same Operating System (Windows / Linux).
- The first MetaDefender Core instance connects to the shared database must be ready first (license activated, users/groups/roles created) before setting up other MetaDefender Core instances to connect and use that shared database.

Engine update:

- If engine updates is configured to be via Folder source, all MetaDefender Core instances must be able to access to the configured update folder, and the option DELETE FILES AFTER IMPORT will be unchecked and prevented for modification.

Authentication:

- Single-Sign-On (SSO) authentication settings (if using) cannot be shared among MetaDefender Core instances because SSO requires an unique login URL for every different Core.
- If an user is locked for login on a certain MetaDefender Core instance(e.g. exceeded the number of failed login times), that user is not prevented for logging in on any other instances.
- If an user is locked for sign-in on a certain MetaDefender Core instance (e.g. exceeded the number of failed login times), if the admin releases lockout on that user on that MetaDefender Core instance, that user is still not able to login on other MetaDefender Core instances.

Sanitized / DLP / Quarantined location changed after MetaDefender Core install:

Windows:

Shared storage path will be stored in 3 strings under key global in registry: quarantinepath, dlppath, sanitizepath as below

(Default)	REG_SZ	(value not set)
dbmode	REG_SZ	4
dlppath	REG_SZ	\\10.40.50.100\test
instance_name	REG_SZ	481rvj
quarantinepath	REG_SZ	\\10.40.50.100\test
restaddress	REG_SZ	*
restport	REG_SZ	8008
sanitizepath	REG_SZ	\\10.40.50.100\test

You can set either 1, 2 or 3 different paths for the 3 settings above (means that quarantined, DLP and sanitized files will be stored in same or different storages corresponding). In case shared storage is not set in installation, values of 3 settings above will be empty.

Restart MetaDefender Core service after changed.

Linux:

Open file `/etc/ometascan/ometascan.conf` and set shared storage path for 3 keys quarantinepath, dlppath, sanitizepath under section [global]

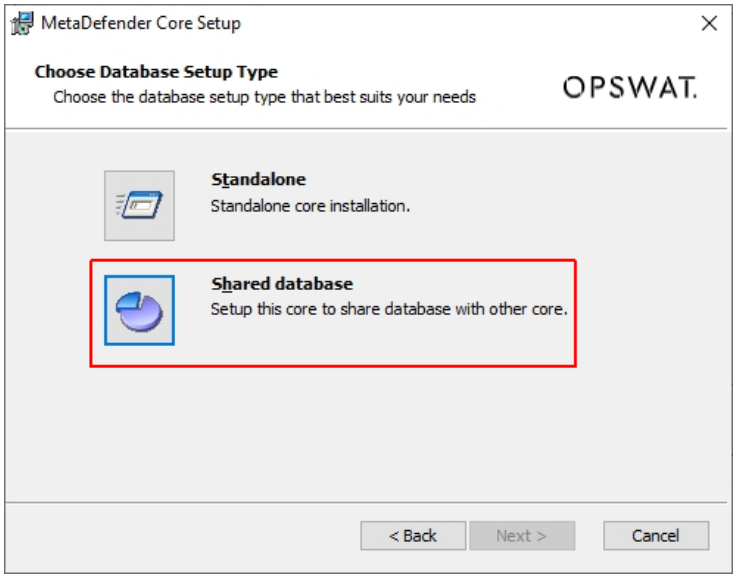
Restart ometascan service after changed.

Wizard Installation (Windows Only)

> **Info**
> Only applicable to MetaDefender Core 4.20.0 or newer using wizard on Windows OS.

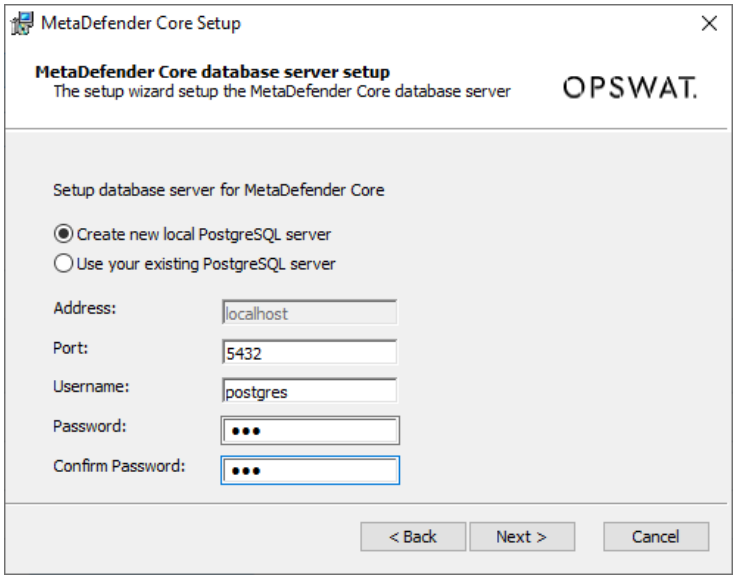
Clean install scenario

1. **Choose Shared database option:**

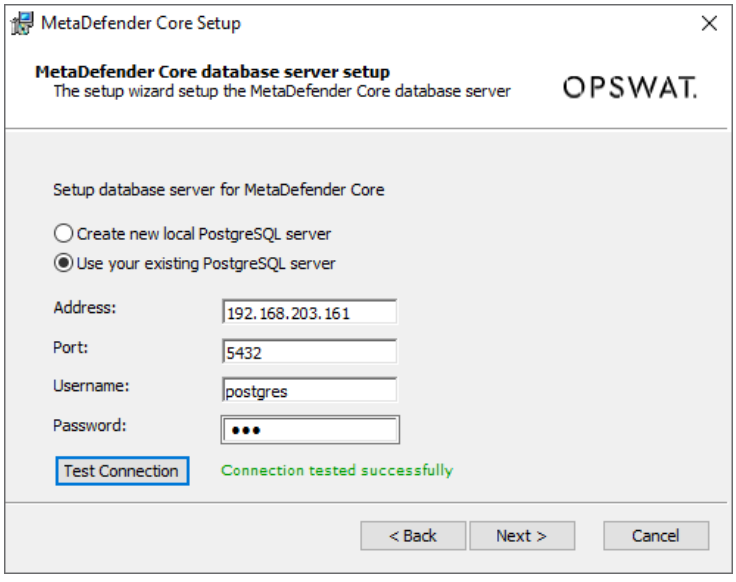


2. **Setup database server for MetaDefender Core:**

You can choose to create a native / built-in PostgreSQL server and setup the shared database on (then other MetaDefender Core instances will want to connect to that PostgreSQL server afterwards)



Or choose to connect to a pre-setup existing PostgreSQL server:



PostgreSQL connection configuration

"Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

PostgreSQL user privilege requirements

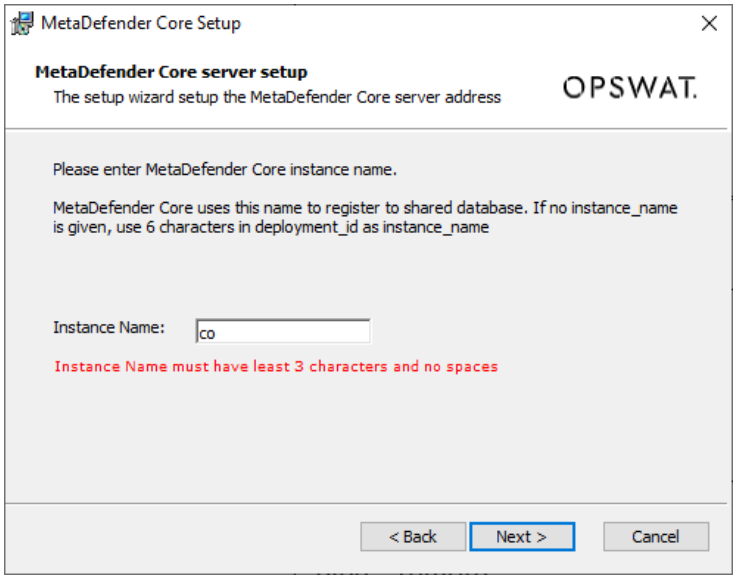
MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.

Details: PostgreSQL User Privilege Requirements

3. **Create an unique instance name for that MetaDefender Core (different from other MetaDefender Core instances):**

Instance name must contain at least 3 characters and no white space.

If you choose to leave it blank, which means it will use the default instance name with 6 characters extracted from that MetaDefender Core deployment ID.

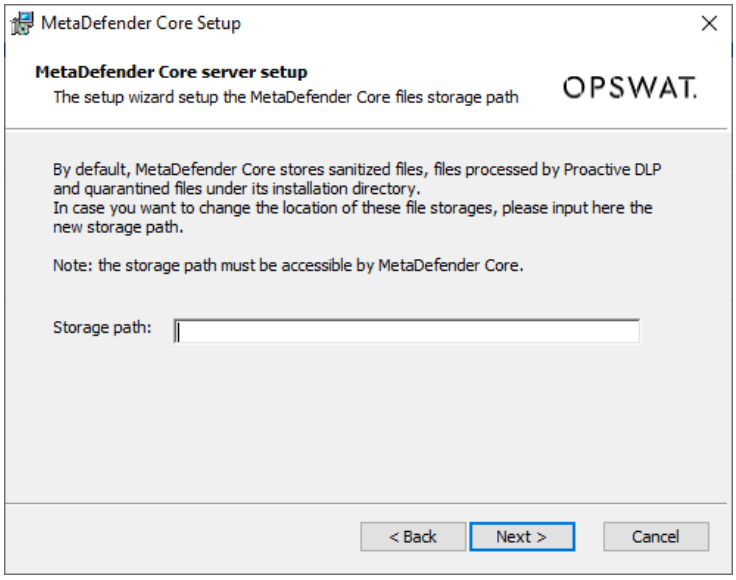


4. **Input the file storage path (sanitized, DLP, quarantined files)**

In the shared database model, all MetaDefender Core instances could use the same folder path to store all their own sanitized, DLP, quarantined files after processed.

The default location is: ***<MetaDefender Core installation folder>\data\ {sanitized|dlp|quarantine}***

If you wish to use that default location, just leave the input box blank and press Next.



Otherwise, you can specify a different location of sanitized/dlp/quarantined files. It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core.

For example:

- C:\new_folder
- \192.168.1.1\new_folder

Command Line Installation

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Clean install scenario

1. Create ignition file if not existed

- Windows: **C:\OPSWAT\ometascan.conf**
- Linux: ***/etc/opswat/ometascan.conf***

An example of ignition file:

ometascan.conf bash

```
eula=true

[global]
dbmode=4
instance_name=MetaDefenderCoreA
storage_path=D:\path\

[user]
name=admin
password=admin
email=admin@local

[dbserver]
type=remote
host=192.168.1.1
port=5432
user=postgres
password=your_own_password
```

Field	Required	Mapping to wizard UI	Possible values
global/dbmode	optional default is 1 (standalone)		1 - Standalone 2 - Central Hub deployment 3 - Non-persistent **4 - Shared**
global/instance_name	optional default is 6 characters of deployment ID		Unicode and number At least 3 characters No whitespace
global/storage_path	optional default is empty		Path to storages (DLP, quarantined, sanitized files) It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core. For example: • C:\folder • \192.168.1.1\folder

2. Run installers using command line to install

Please refer to Command Line Installation for installation command.

Deployment automation support

Note

An Ignition file could be utilized for either / both of following use-cases:

1. Pre-define PostgreSQL server information for MetaDefender Core to connect and setup database: only required on Linux installation, or Windows with command line installation. Learn more details at:
   a. Standalone Mode > Command Line Installation (applicable on version 4.19.0 or newer).
   b. Shared Database Mode > Command Line Installation (only applicable on version 4.20.0 or newer).
2. Automating to accept EULA, create a default local admin user and import MetaDefender Core configurations upon installed: Those steps could be automated using ignition file described in this page.

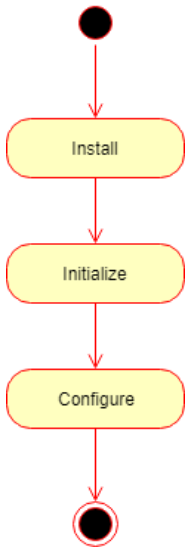
Ignition file location to create:

- **Windows**: `C:\OPSWAT\ometascan.conf`
- **Linux**: `/etc/opswat/ometascan.conf`

The product supports fully automated deployment. It means that it can be installed and configured with no human interaction.

The automated deployment can be split to three steps on a high level:

1. Installation
2. Initialization
3. Configuration



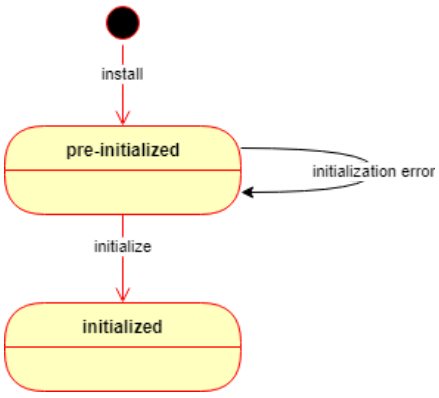
Installation

To automate the installation, install the product from the command line and provide the installation-time options as parameters to the installer. For further details see Command Line Installation

After the installation is complete, the product starts up and waits in a pre-initialized status. The product may be initialized in two ways:

1. Manually using the Wizard Setup or
2. Automatically using an *ignition file* (see below).

> **Warning**
> If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried after fixing the problem (e.g. placing the ignition file in its lookup location) and restarting the OPSWAT MetaDefender Core service.
>
> Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

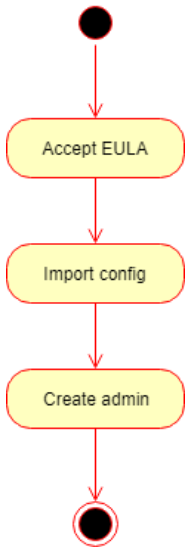


Initialization

Initialization is the process of bringing the product to an operable status.

Basically the initialization consists of the following steps:

1. Accept the End User License Agreement (EULA),
2. Import product configuration and
3. Create the first administrator user account.



Ignition file

The initialization process can be configured in a file called the *ignition file*.

> **Warning**
> In Linux, the ignition file and the folder /etc/opswat should have the permission 755

Ignition file location

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

The ignition file must be in conf format.

ometascan.conf bash

```
eula=true
[user]
name=admin
password=admin
email=admin@local
apikey=44c96cbb4e4e6c0a1d16bcb3c3da24def317
[config]
import=config_package.zip
import_password=123
[security]
session_cookie_samesite=Lax
[internal]
skip_startup_pages=true
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
private_username=my_internal_db_user
private_password=mypassword
[ocm]
ocm_url=https://my.opswat.com
reg_code=XXXXXXXX
```

Ignition file fields

The ignition file accepts the following fields:

Section	Key	Required	Accepted Values	Description
	eula	Optional (mandatory when having [user] section)	true false (default)	Whether to accept the End User License Agreement. This key must be set at the top of ignition file. This key must be set to "true" to accept the EULA. Any other value will cause the initialization to fail. When having [user] section defined, then this key must be set.
[user]		Optional		Initial administrator user account properties. The Administrator role is granted to the account.
	name	Optional	string	User name for the initial administrator user account.
	password	Optional	string	Password for the initial administrator user account. WARNING! Clear text password The password in this configuration file must be stored in its clear-text format and as so it may be visible for unauthorized parties.
	email	Optional	string	E-mail address for the initial administrator user account.
	apikey	Optional	string	API key for the initial administrator user account.
[config]		Optional		
	import	Optional	string	Path to a file in json/zip format that contains a previously exported configuration to be imported. Zip-format file contains a full configuration including user management settings, so it will overwrite [user] section.
	import_password	Optional	string	Password to decrypt .zip configuration package.
[security]		Optional		

Section	Key	Required	Accepted Values	Description
	session_cookie_samesite	Optional	Strict Lax (default) None	Declare if session cookie should be restricted to a first-party or same-site context.
[internal]		Optional		
	skip_startup_pages	Optional	true false (default)	Skip all welcome and other startup pages (e.g. upgrade result) when installing / upgrading MetaDefender, and going straight to the product index default page.
[dbserver]		Optional		
	type	Mandatory when having [dbserver] section	local remote	Where the PostgreSQL database server should physically locate. "remote" means PostgreSQL server and MetaDefender Core are not in the same machine.
	host	Mandatory when having [dbserver] section	string	IP address / domain name of the server where PostgreSQL server locates. "localhost" can be used when applicable.
	port	Mandatory when having [dbserver] section	number	Port of PostgreSQL server is listening for connections from clients (i.e. MetaDefender Core).
	user	Mandatory when having [dbserver] section	string	PostgreSQL server's user. SUPERUSER privilege is required for MetaDefender Core to setup its database and extensions for the first time. **Only non-Unicode characters supported.**

Section	Key	Required	Accepted Values	Description
	password	Mandatory when having [dbserver] section	string	PostgreSQL server's user credentials. **Only non-Unicode characters supported.**
	private_username	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
	private_password	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
[ocm]		Optional		
	ocm_url	Optional	string	URL to My OPSWAT server.
	reg_code	Optional	string	A registration code to enroll to My OPSWAT server.

Detailed initialization process

1) After the product has been started, it looks for the ignition file in the designated location.

Windows: C:\OPSWAT\ometascan.conf ** Linux: /etc/opswat/ometascan.conf

** 2) If an ignition file is found, then

2.1) It gets validated, and if it is valid, then based on the information found in the ignition file:

- The EULA is accepted,
- The configuration is imported,
- The administrator account is created.
- If any of the above steps fails, then the error is logged, and the initialization gets terminated.

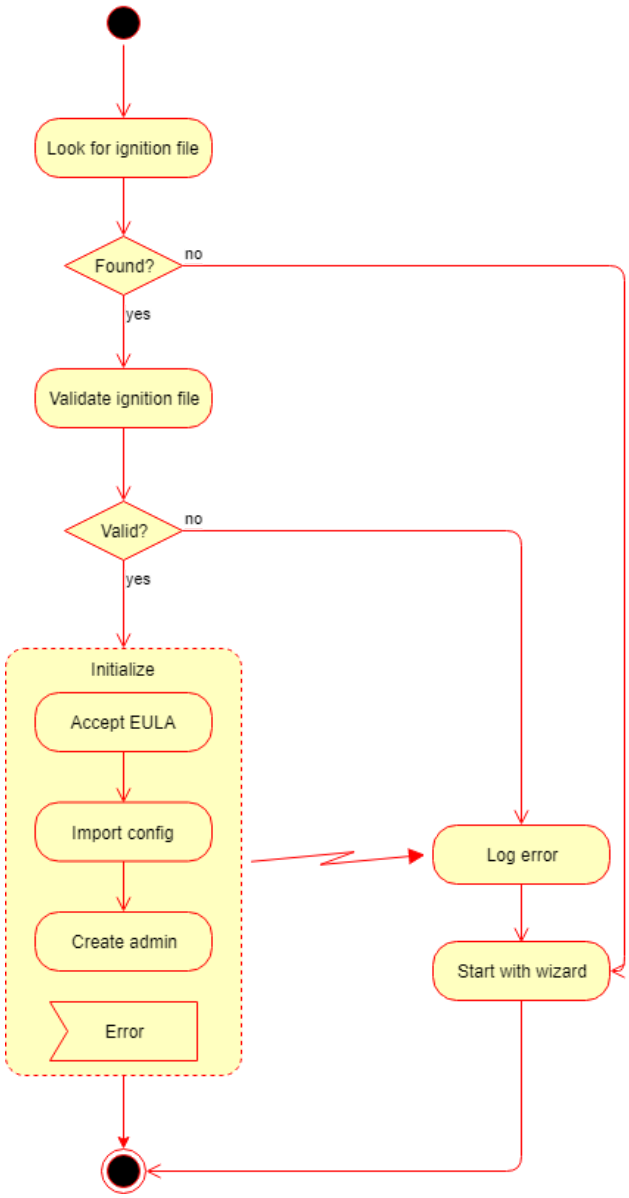
> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

2.2) If it is not valid, then the error is logged, and the initialization gets terminated.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

3) If there is no ignition file, then no initialization is performed.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.



Warning

If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried fixing the problem (e.g. placing the ignition file to its lookup location) and restarting the OPSWAT MetaDefender Core service.

Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

Configuration

After the initialization is complete, the product is ready with the default and the imported configuration.

This configuration can be later changed calling the configuration API functions. For further details about the API see Configuration related APIs API

CIS Level 1 Guidelines

Support Red Hat Enterprise Linux 9
For more details about Center for Internet Security (CIS) please refer to this document

Instruction steps

I. Install OpenSCAP

bash bash

```
yum install openscap-scanner scap-security-guide
```

II. Generate a result file and a HTML report using OpenSCAP scanner tool

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

III. Remediation of CIS Level 1 issues

Generate a remediation script based on the `ssg-rhel9-ds.xml` file:

bash

```
oscap xccdf generate fix --profile
xccdf_org.ssgproject.content_profile_cis_server_l1 --fix-type bash --output
remediations.sh /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

And execute remediation script.

bash

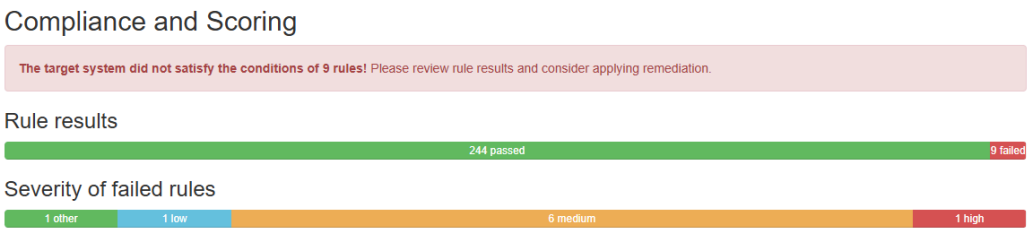
```
./remediations.sh
```

IV. Review the results after remediation

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

Example:



Notes

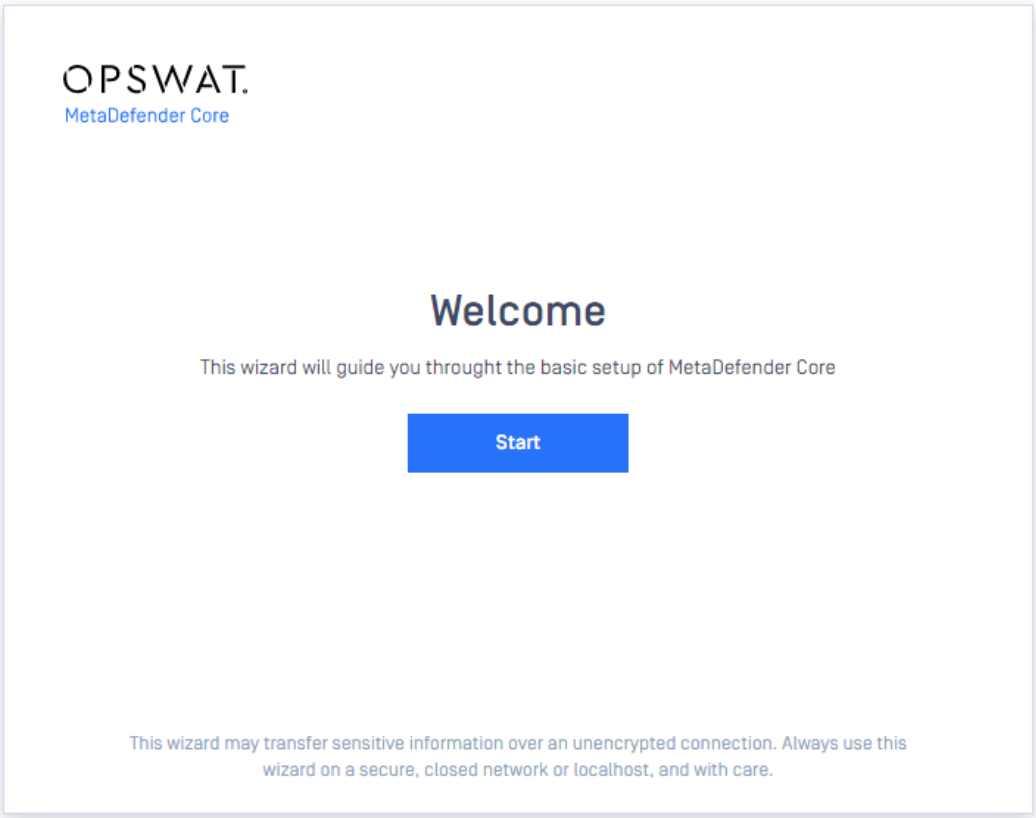
- CIS Level 1 requires `/tmp` folder to be mounted in a separate partition. Please ensure that that new partition have enough disk space for MetaDefender Core to run.
- CIS Level 1 requires that "Ensure No World-Writable Files Exist".
    - For now, when freshly installing MetaDefender Core, all its binary files meet the requirement.
    - When installing/updating engines, some engines might create additional files for its operation, and it might violate this requirement. In this case, you need to again execute the remediation script in the step III.

Wizard Setup

Introduction

Access the MetaDefender Core management console for the first time (i.e. http://localhost:8008), you are to complete a quick walkthrough for wizard setup in order to be able to use the product.

The administrative management console will be available only upon you have successfully finished this wizard stage.

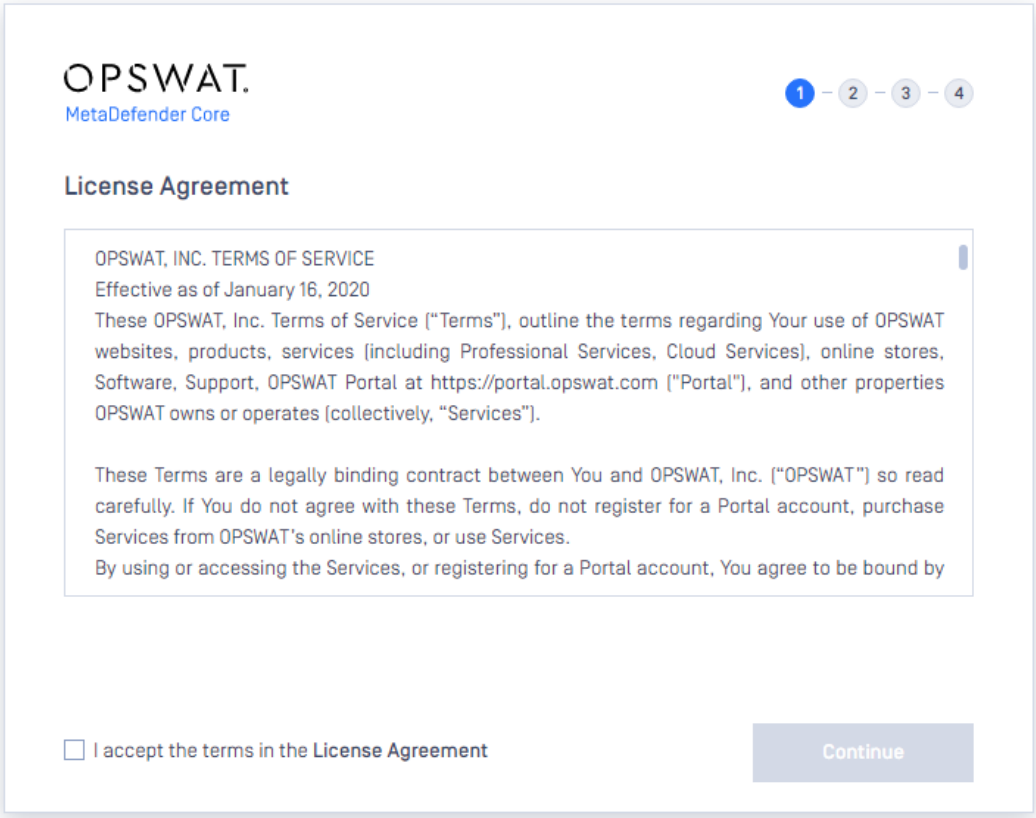


Warning

This wizard may transfer sensitive information over an unencrypted connection. Always use this wizard on a secure, closed network or localhost, and with care.

Basic configuration steps

End-User License Agreement



In the first page you can find the End-User License Agreement. You have to accept the terms before moving on.

Please read through the EULA (https://docs.opswat.com/policies/privacy-policy/portal-privacy-policy) carefully and if you agree with it, check `I accept the terms in the **License Agreement**` and click `Continue` to proceed.

Local Admin User Setup

The next step is to set up a default local administrator account.

This account will be the first one being able to access the administrative management console, and to create accounts for other users. You have to fill out all fields in this page to be able to move forward. When you are done, click NEXT to continue.

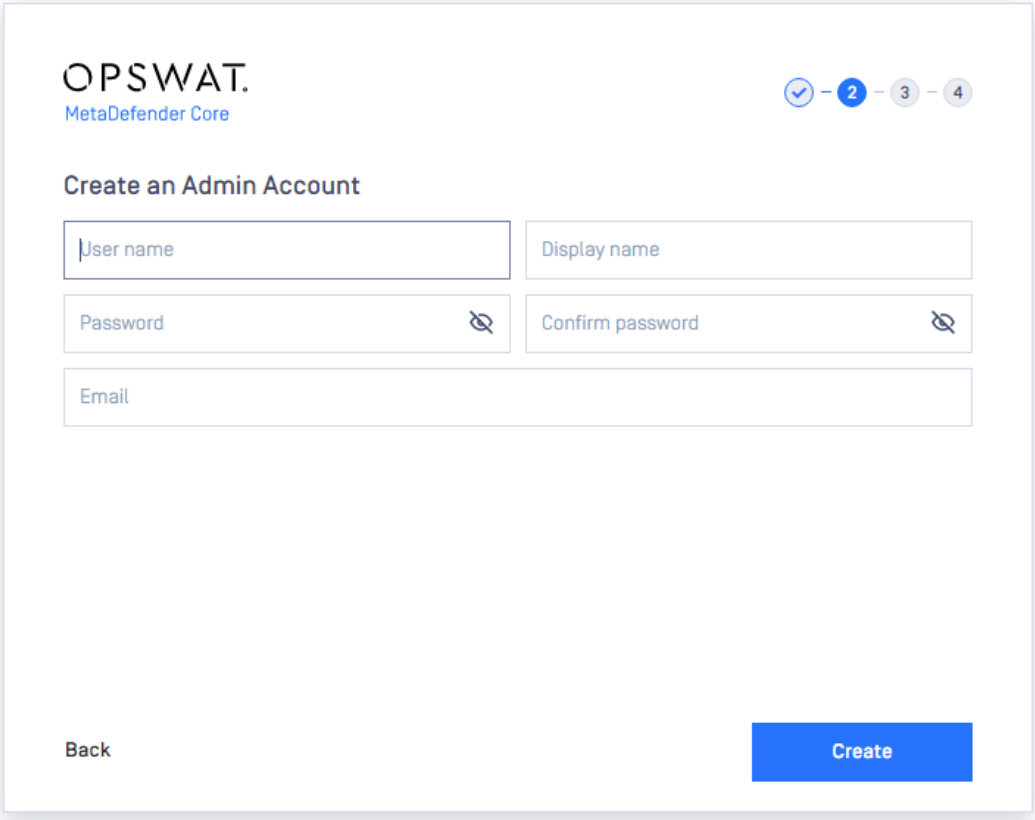
Info - User Directory

The administrator account, that is created via the basic configuration wizard, is always added to the LOCAL user directory as a member.

Learn more about user directory: User directories

The following information is required for the administrator account:

ACCOUNT NAME	The unique name of the account that is used at the time of login and in log messages for accountability.
ACCOUNT DISPLAY NAME	Name of the person bound to this account. This name (appended to the name of the account's user directory) is displayed in the top right corner of the administrative management console.
PASSWORD	Password of the user bound to this account that is used at the time of login. Passwords sent clear-text As long as TLS is not configured for the basic configuration wizard, passwords are sent clear-text over the network and may be disclosed to unauthorized parties. As a mitigation action: 1. Either use the wizard on *localhost* or on a direct network connection, or 2. Enabling HTTPS as soon as possible and change the password immediately if it has already been set.
EMAIL	Email address of the person bound to this account.



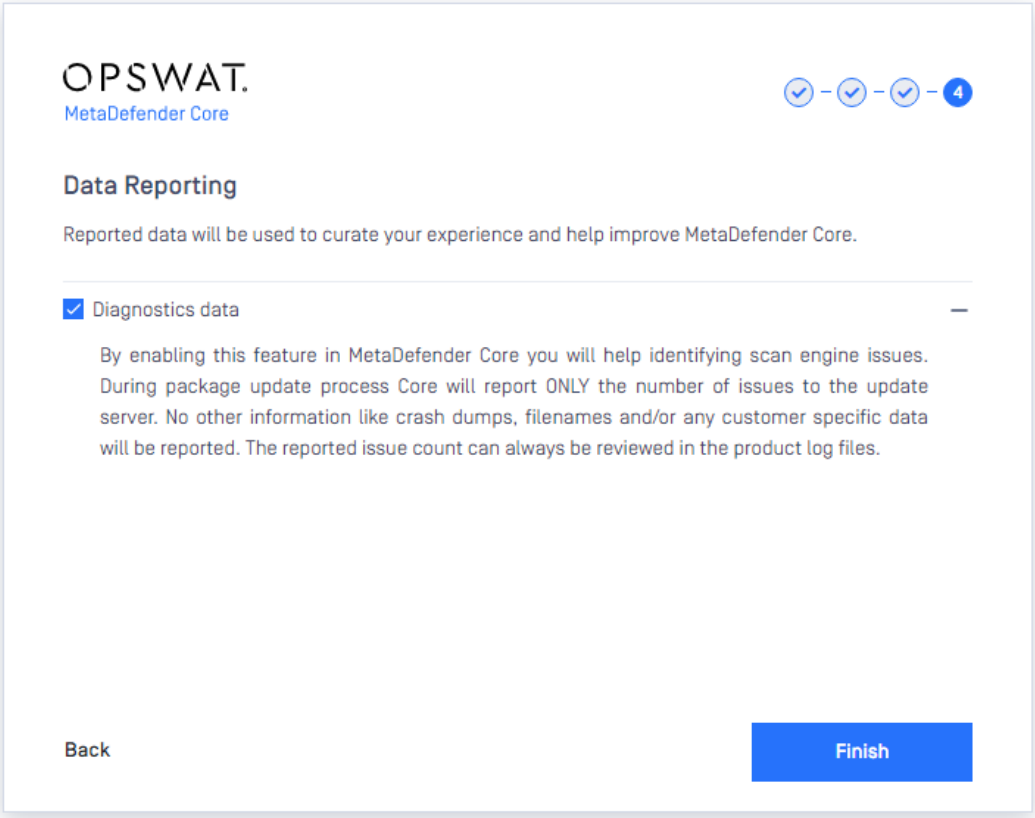
License activation

For license activation details see License Activation

Data Reporting

Enabling this option to allow MetaDefender Core collect its own engine issues and report back to OPSWAT for further analysis.

MetaDefender will not collect any other information.

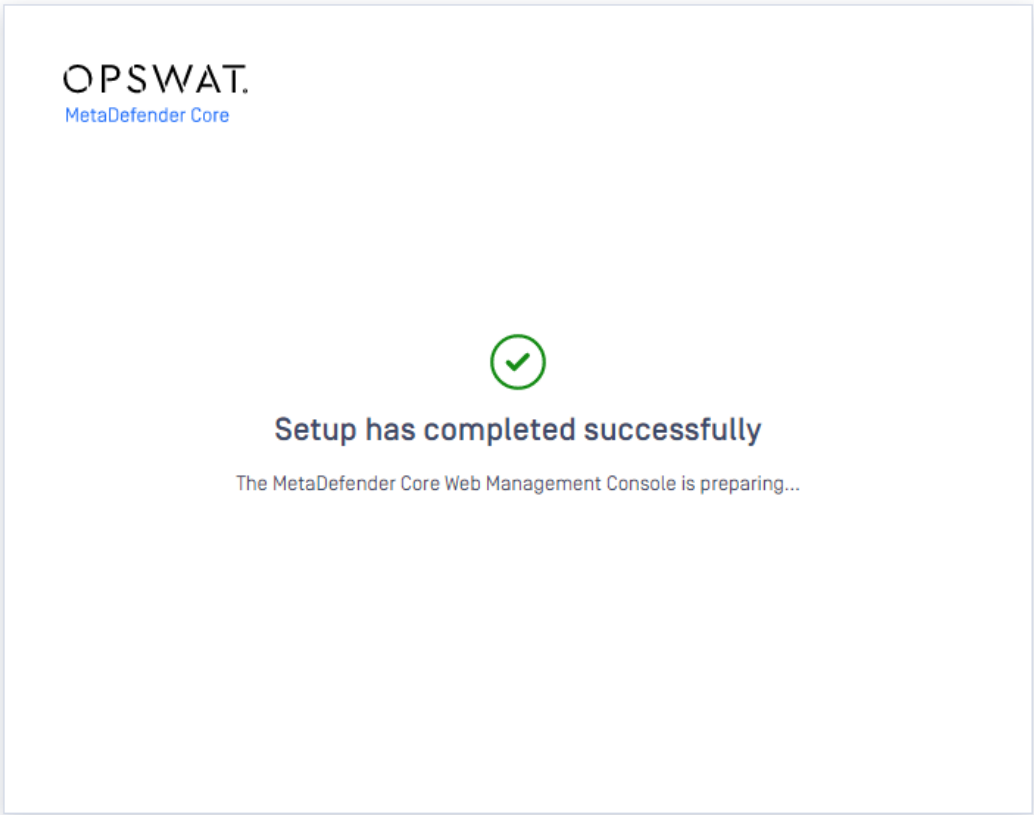


Wizard completion

After you have completed every steps you are ready to finish the wizard and start using the product. Click the `Fishish` button to complete the wizard.

> **Warning**
> The product's service will be restarted and the browser will be redirected to the administrative management console. This could take several seconds.

You can login to the Web Management Console with the administrator user that have just been created in the previous steps.



Setup HTTPS

Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network. Websites, like the Web Management Console, are able to use TLS to secure all communications between their servers and web browsers.

The TLS protocol aims primarily to provide confidentiality (privacy) and data integrity between two communicating computer applications.

Warning

No TLS for the wizard

By default, TLS is not enabled for the basic configuration wizard. As a consequence sessions between the wizard's backend and the browser may be insecure.

Info

Performing the same steps as for the Web Management Console, it is possible to set up TLS for the basic configuration wizard. Remember completing the TLS setup before launching the wizard.

For instructions to set up TLS see Enabling HTTPS

License Activation

##

To activate your installation go to the **Settings > License** menu in the Web Management Console. If you have no valid license, you will only see your installation's **deployment ID**. You will also see a warning in the Web Management Console header.

After successful activation, the product will start downloading the latest available scan engines and malware databases. You can follow the status of the scan engine installation on the **Inventory > Modules** page.

When all the licensed engines are installed, you can start using the MetaDefender Core to process files - File Processing.

Deployment ID

Each MetaDefender Core instance represented by an unique system fingerprint called `Deployment ID`

MetaDefender Core licensing is purely based on deployment ID activated. Therefore, everytime your MetaDefender Core is activated with a different deployment ID, then it will be captured as a different activation record accumulated into your total activation times allowed.

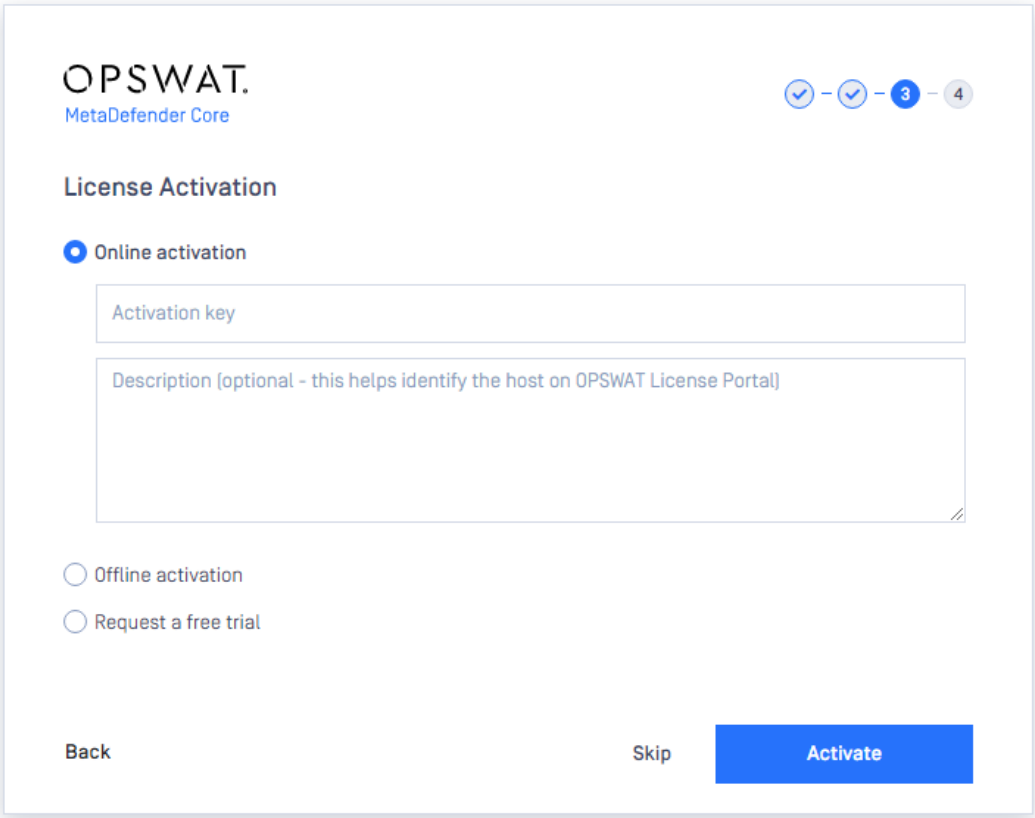
> **Notice**
> Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you alter the system where MetaDefender Core relies on (e.g. network card, OS hostname).
>
> Deployment ID is an UUID version 4 random based string. Learn more at: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you update the system where MetaDefender Core relies on.

Activation Modes

There are three different options for the product activation:

- Online activation: MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.
- Offline activation: Requiring you to upload a license file (.yml). That license file could be retrieved by activating your license via OPSWAT Portal. Follow the displayed instructions for details.
- Request trial key online: For evaluation purpose, you can receive an trial Activation key via email. Follow the displayed instructions for details.



Online activation

MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.

Required to input your Activation License key, and optional for description (but we recommend this for troubleshooting and auditing purpose if needed).

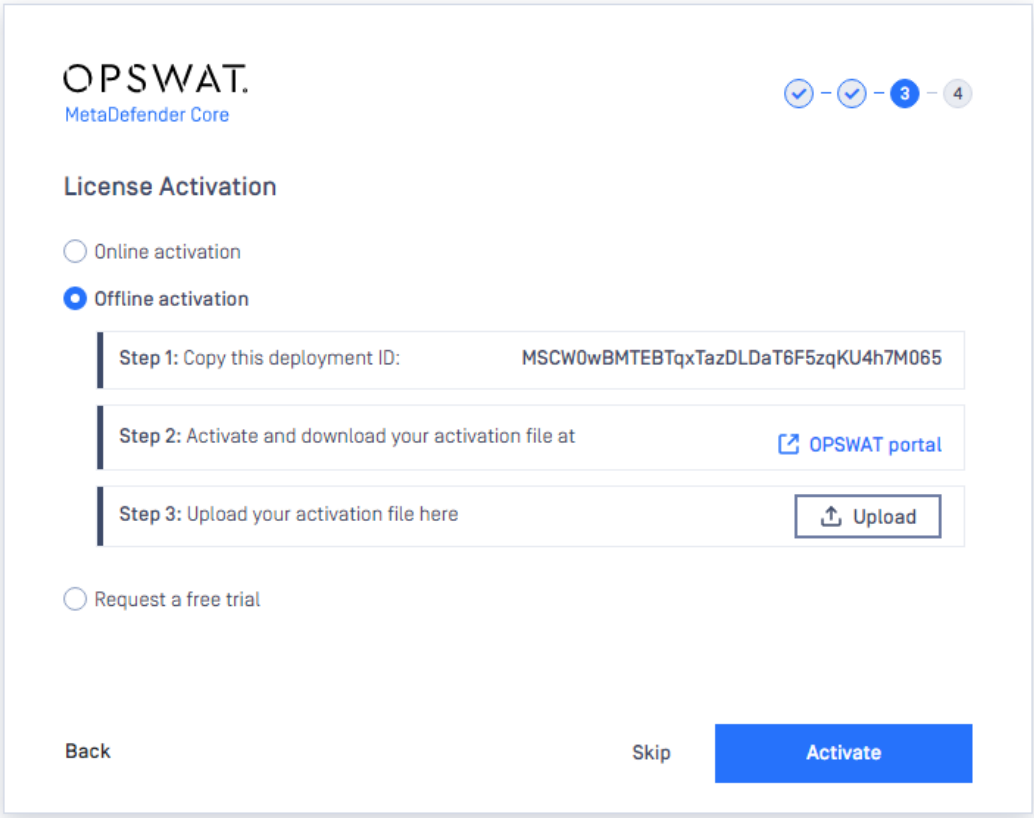
Warning

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts' for license activation:

- https://activation.dl.opswat.com Note: *IP address-based whitelisting on your firewall might fail after some time since OPSWAT uses CDN (Content Delivery Network) to faster delivery updates over the world, and IP address of edge servers might change over time.*

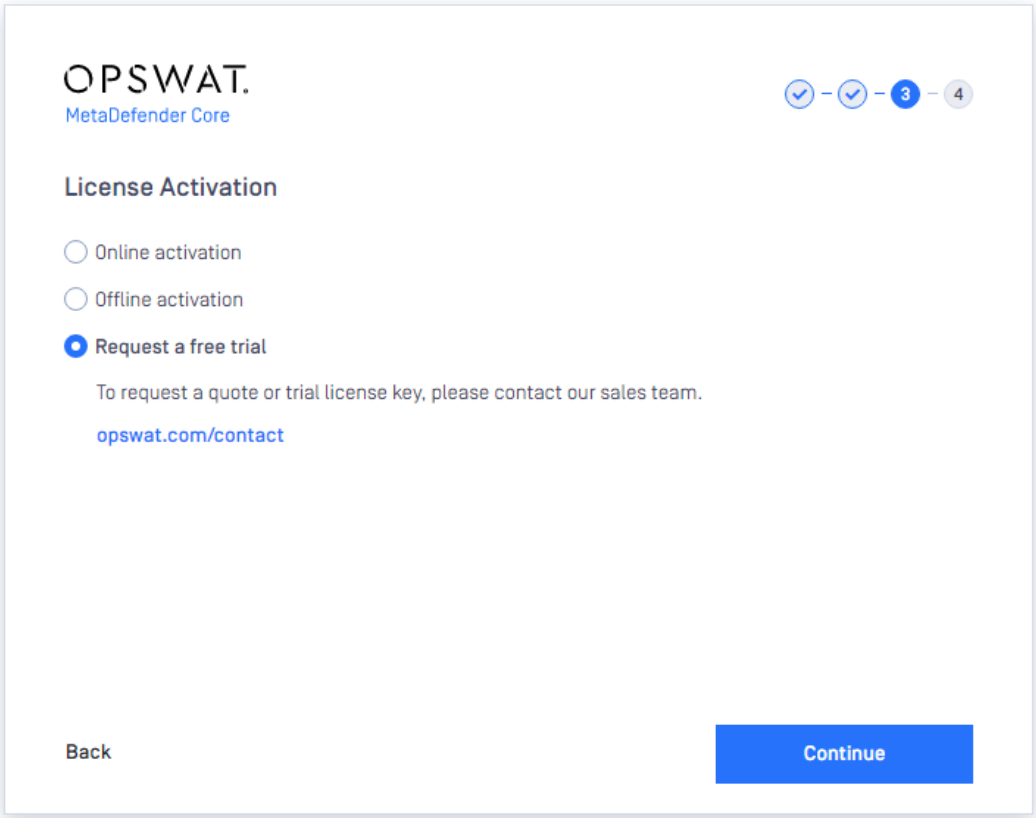
Offline activation

With no internet connection on the server the Metadefender v4 instance may be activated indirectly from a different machine, that has internet connection. The Deployment ID of the Metadefender v4 instance and the the Activation key received at the time of purchasing the product will be required. Follow the steps on the screen to activate the product offline.



Request trial key online

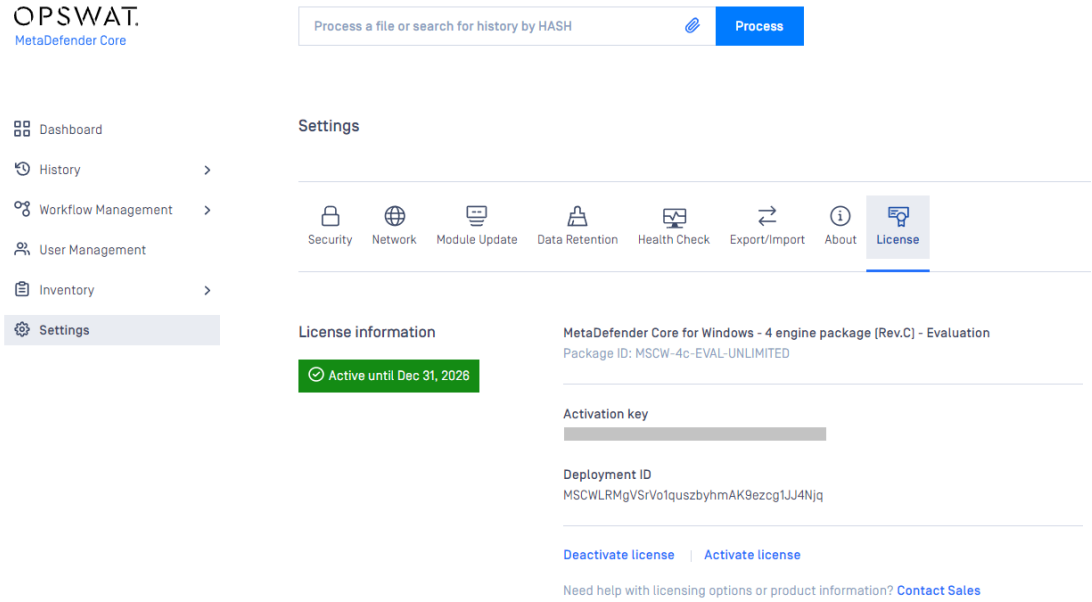
An evaluation license could be acquired by contacting our sales team.



Check Your License Details

For more license details and activating your MetaDefender Core, go to Settings > License menu tab on the Web Management Console:

- Product name: product name as on your order (e.g. MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation)
- Package ID: product identification as on your order (e.g. MSCW-4c-EVAL-UNLIMITED)
- Expiration: last day of license validity
- Activation key: key used for activating product
- Deployment ID: identification of this installation


OPSWAT.
MetaDefender Core
Process a file or search for history by HASH
Process
Dashboard
History
Workflow Management
User Management
Inventory
Settings
Settings
Security
Network
Module Update
Data Retention
Health Check
Export/Import
About
License
License information
Active until Dec 31, 2026
MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation
Package ID: MSCW-4c-EVAL-UNLIMITED
Activation key
Deployment ID
MSCWLRMgVSrVo1quszbyhmAK9ezcg1JJ4Njq
Deactivate license
Activate license
Need help with licensing options or product information? Contact Sales

Scripted license management

Using REST API calls there is an option for scripted activation and deactivation for MetaDefender Core servers with **no Internet connection or if your infrastructure is using automation to create/destroy MetaDefender Core instances**. In this chapter the steps of these two scenarios are described.

Requirements

- an installed MetaDefender Core instance without Internet connection
- another computer that has Internet connection and can run your activation scripts
- a manual or automated way to transfer data between the two computers

Activation steps

1. In order to activate MetaDefender Core, deployment ID and activation key are needed to generate the license file.
    a. Activation key should be purchased from OPSWAT.
    b. MetaDefender Core deployment ID can be queried by using REST API (for details see GET Get Current License Information API page)

> **Info**
> Save this deployment ID in your system, you might need this ID when the instance is unavailable at the time of deactivation.

2. Activate license and get the license file using the following URL, where <quantity> is 1.

activation call bash

```
curl -X GET https://activation.dl.opswat.com/activation?key=<activation-key>&deployment=<deployment unique-ID>&quantity=<quantity>
```

If the activation is successfully, a license file is downloaded. Save this file.

HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	license file	Activation was successfully.
200 Activation failed	error: '<user conform error message>'	Failed activation
200 Invalid parameter	error: 'Could not activate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not activate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
200 Invalid parameter	error: 'Could not activate your product because the quantity you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key, quantity or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Info
In case of any activation issue, contact OPSWAT support for help

3. Upload license file to MetaDefender Core: The license file should be uploaded to the MetaDefender Core v4 to activate the product. For details see page: POST Upload license key API

Deactivation steps

1. For deactivation, following infomation is required:
    a. Activation key
    b. MetaDefender Core deployment ID: can be queried by using REST API (for details see GET Get Current License Information API page)
2. Deactivate license using the following URL:

deactivation call bash

```
curl -X GET https://activation.dl.opswat.com/deactivation?key=<activation-
key>&deployment=<deployment unique-ID>
```

Response HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	result: ok	Successful deactivation
200 No active license found	error: 'Could not found any active license with the given parameters'	The license has not been activated yet or it has been deactivated already.
200 Invalid parameter	error: 'Could not deactivate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not deactivate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Important notes

Warning

Product activation is tied to several hardware, operating system and software parameters. In case of one or more major hardware or software parameter change the product might turn into deactivated status. Operating system updates, other software updates should not affecting the activation status.

If the product is online activated and has live Internet connection then in case of deployment ID change the product reactivates itself. If the product doesn't have Internet connection then the administrator is responsible to reactivate the product.

Because of the above administrator should consider licensing restrictions of the offline deployments if using any virtualization and/or containerization technologies.

File Processing

MetaDefender Core provides REST API interface for file processing business.

There are several ways to process files with MetaDefender Core:

- Analyze Files via Web Interface
- Process Files via REST API

Product Upgrading

> **Info**
> Simply run installer file, details could be found at Installation

> **Info**
> For command line mode, upgrading (e.g. on **Linux**), prior to running the installer file, please make sure to **create the ignition file** with PostgreSQL server information included. Details:
>
> - Standalone DB mode: Command Line Installation (Windows and Linux)
> - Shared DB mode: Command Line Installation

Recommendation for MetaDefender Core upgrade

Our goal is to help you execute the product upgrade continuously, efficiently, and correctly.

PostgreSQL service upgrade

- **Bundled PostgreSQL Service**: If you are using the bundled PostgreSQL server included in the installer, the PostgreSQL service will be automatically upgraded to the newer version (if available). No additional action is required for this process.
- **Remote PostgreSQL Service**: For remote PostgreSQL servers, customers are responsible for managing and upgrading the service independently. Upgrading MetaDefender Core does not update the remote PostgreSQL service.

Before the upgrade

Prepare a proper disaster recovery plan (DRP) including some common practices:

- Backup, or take a snapshot of your current working MetaDefender Core server.
  - This is to help quickly rollback for any upgrade failure might cause later for any reason.
- Optionally backup your PostgreSQL database, especially when using remote shared database mode.
  - Reference: https://www.postgresql.org/docs/current/backup.html
  - Please do it after stopping all linked MetaDefender Core instances to avoid potential data collision.
- Lastly, if backing up entire server is not feasible, then at least export MetaDefender Core configuration package Import/Export configuration just in case you'll need to start off a new server with current MetaDefender Core configuration to keep the scanning service running without any impact.

On the PostgreSQL server, double check server resource to ensure sufficient RAM memory and disk space to be preserved for the upgrade process.

- 8GB RAM available at least during the upgrade.

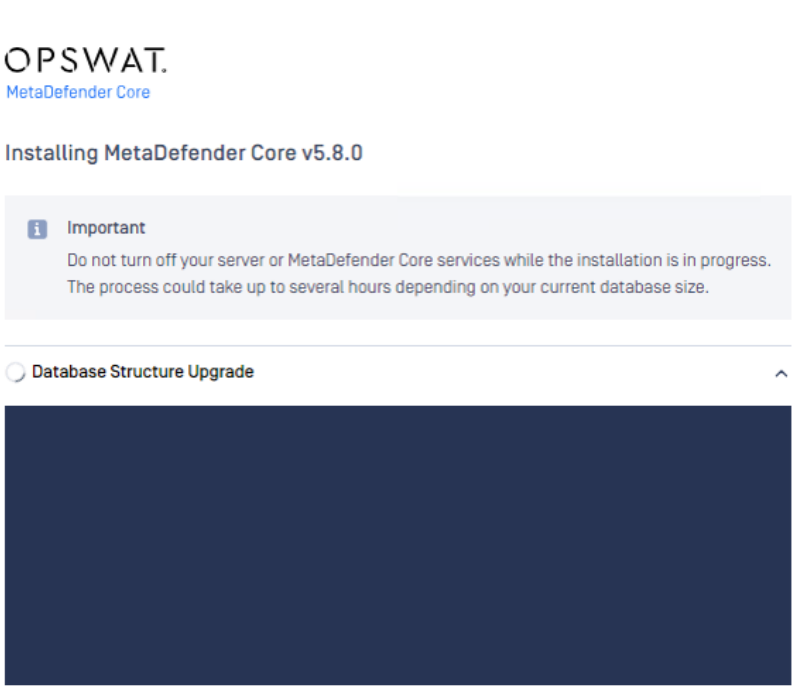
- Disk availability is greater than MetaDefender Core database size.

While doing the upgrade

Applicable for shared database deployment:

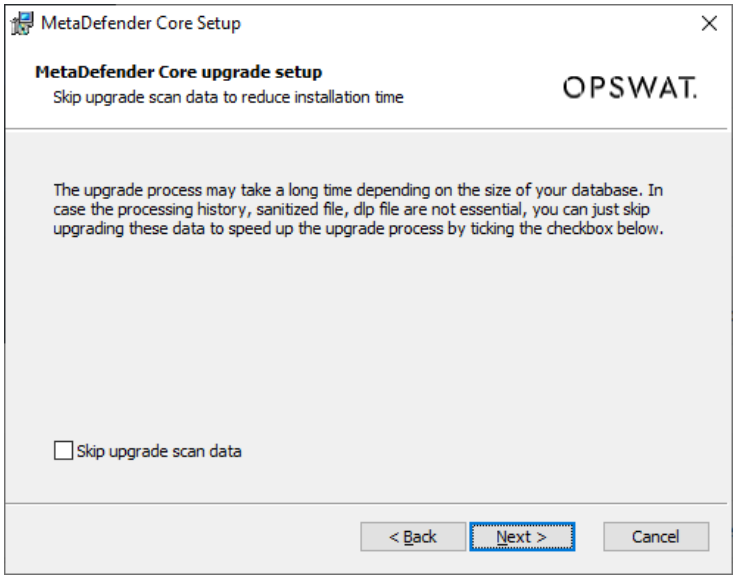
- Stop all MetaDefender Core instances' scanning service before performing the upgrade.
- Upgrade one (1) MetaDefender Core instance at a time.
- Proceed to upgrade MetaDefender Core instance where the bundle local PostgreSQL server is used for shared database setup.
- Otherwise is using remote PostgreSQL then the upgrade order does not matter.

For large size database, mostly for scan history, expecting long time to be spent on the upgrade process.



The product supports an option to skip data migration for scan history to expedite the process drastically and mitigate any failure risk. You can manually backup all data from your PostgreSQL server for offline usage if any.

Check "Skip upgrade scan data" to enable this feature while doing the upgrade.



or add following key in the ignition file:

none

```
[internal]
skip_upgrade_scan_data = 0
```

Troubleshooting

If the upgrade process ever runs into an error:

- Double check your network and services, applicable to remote PostgreSQL server.
- Optionally configure MetaDefender Core to skip migrating scan history.
  - Windows
    - Open Windows registry, and navigate to HKLM\Software\OPSWAT\Metascan
    - Add a new key `internal/skip_upgrade_scan_data` with value = 1
  - Linux
    - Open MetaDefender Core configuration file `/etc/ometascan/ometascan.conf`
    - Add new key into the file
      - [internal]
      - skip_upgrade_scan_data=1
  - Note: This key will be auto removed upon the upgrade finishes.
  - Then restart MetaDefender Core service to retry the upgrade.
    - Might retry this a few times until the upgrade is done.

Just in case the upgrade process still fails, please contact OPSWAT support with the support package - How to Create Support Package? This will help us analyze the root cause and figure out an optimal solution for your system failure.

Using external Load Balancer

Application Load Balancing (Layer 7)

This page provides information about recommended ways to use sessions with a Layer 7 load-balancer.

Most load-balancers have the ability to provide cookies for pinning subsequent traffic from a client to the appropriate server. This method is called sticky session, session persistence or session affinity. Using cookies efficiently requires the client to know when a cookie should be sent, should not be sent or should be deleted.

Sticky session load balancing

Each Metadefender Core instance has its own databases and application sessions that cannot be seen by other instances. Therefore, in order to get the related data/response to our queries we should ask the appropriate Metadefender Core v5 server. On the other hand, to keep the advantages of the used load balancing method, cookies should not be sent if it is not necessary.

Single file scanning

Step	Stage	Task	Cookie usage
1.	Sending file	Initiate processing a file on the client side. Send file through the load-balancer. (See POST File Analysis API)	Cookie should not be sent
2.	Sending file	Save the cookie and the data_id you got from the load-balancer	Save cookie
3.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Send cookie saved in step 2
4.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Send cookie saved in step 2

Batch scanning

Step	Stage	Task	Cookie usage
1.	Open batch	Initiate processing file(s) in batch. Request a batch ID through the load-balancer. (See POST Initiate Batch API)	Cookie should not be sent
2.	Open batch	Save the cookie and the batch_id you got from a Core server through the load-balancer.	Save cookie
3.	Sending files	Send file through the load-balancer. (See POST File Analysis (using batch header) API)	Send cookie saved in step 2
4.	Sending files	Save the data_id you got from the load-balancer.	-
	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Send cookie saved in step 2
5.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
6.	Getting batch status	See GET Batch Status API	Send cookie saved in step 2
7.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Send cookie saved in step 2
8.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API	Send cookie saved in step 2

If it does not matter which upstream server responds, the queries should be sent without cookie.

> **Info**
> It is recommended not to send cookies when it's not necessary to allow load-balancer to use its own method to share the load between Metadefender Core v5 servers.

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration through load-balancer

OPSWAT products that support HTTP load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	4.3.4	-
OPSWAT Client	Windows: 7.6.247.0 Mac: 10.4.243.0	-
MetaDefender Email Security	4.3.0	-
MetaDefender ICAP Server	4.3.0	-
MetaDefender Vault	1.3.0	-
MetaDefender for Secure Storage	2.0.7	-

DNS Load Balancing

Using this method is logically similar to Layer 7 load-balancing.

Briefly how it works

Client uses a domain name to send a query to a server. Client's DNS server has more "A" records for that name with different IPs. When a client resolves the server's domain name DNS server randomly chooses an IP for that name to send back. When a session is used on the application layer, client should know the IP address of the Core server that handles that specific session. In every other case, client should resolve the domain name with DNS query to let requests to be balanced between Metadefender Core servers.

Single file scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Sending file	Initiate processing a file on the client side. (See POST File Analysis API)	Use IP saved in step 2
4.	Sending file	Save the data_id got from the Core server	-
5.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Use IP saved in step 2
6.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Use IP saved in step 2

Batch scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Open batch	Initiate processing file(s) in batch. Request a batch ID. (See POST Initiate Batch API)	Use IP saved in step 2
4.	Sending files	Send file to the specific Core server with the basch ID saved in step 3. (See POST File Analysis (using batch header) API)	Use IP saved in step 2
5.	Sending files	Save the data_id you got from the Core server.	-
6.	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Use IP saved in step 2
7.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
8.	Getting batch status	See GET Batch Status API	Use IP saved in step 2
9.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Use IP saved in step 2
10.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API).	Use IP saved in step 2

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration via DNS load-balancing

OPSWAT products that support DNS load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	does not support yet	-
MetaDefender Client	does not support yet	-
MetaDefender Email Security	does not support yet	-
MetaDefender ICAP Server	does not support yet	-
MetaDefender Vault	does not support yet	-

SSL connection for PostgreSQL communication

> **Note**
> The SSL connection configuration is possible thanks to the support made by PostgreSQL. For reference: https://www.postgresql.org/docs/14/ssl-tcp.html
>
> MetaDefender Core is tested against this mode on particular version 4.21.1

Step 1: Generate SSL certificate for PostgreSQL server

> **Note**
> Following instructions are for self-sign certificate. For production environment, making sure to use the certificate signed by your trusted CA.

1) Generate a private key

bash

```
openssl genrsa -des3 -out server.key 1024
```

Sample output:

bash

```
Generating RSA private key, 1024 bit long modulus ........++++++
..............++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

Type your desired password / pass phrase to encrypt the private key

2) Remove the pass phrase to automatically start up the PostgreSQL server

bash

```
openssl rsa -in server.key -out server.key
```

> **Note**
> On Linux, make sure to set permission on the server.key file

bash

```
chmod og-rwx server.key
```

3) Create a self-signed certificate

bash

```
openssl req -new -key server.key -days 3650 -out server.crt -x509
```

You will be prompted to enter detailed information which is incorporated into your self-signed certificate request.

none

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:CA
Locality Name (eg, city) [Default City]:San Francisco
Organization Name (eg, company) [Default Company Ltd]:OPSWAT
Unit Name (eg, section) []:OPSWAT
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:dummy_email@opswat.com
```

For self-signed certificate, use the server ceriticate as the trusted root certificate:

bash

```
cp server.crt root.crt
```

Copy **server.key**, **server.crt**, **root.crt** to PostgreSQL data folder. For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data**
- Linux: **/var/lib/ometascan/pg_data/**

Step 2: Configure PostgreSQL server for SSL authentication connection

1) Create a custom config file (e.g. ssl.conf) for PostgreSQL.

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\postgres\ssl.conf**
- Linux: **/var/lib/ometascan/postgres/ssl.conf**

2) Content of ssl.conf:

none

```
ssl = on
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
```

3) Modify pg_hba.conf

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data\pg_hba.conf**
- Linux: **/var/lib/ometascan/pg_data/pg_hba.conf**

Add following (modify **::1/128** if needed to match with your environment requirement)

none

```
hostssl all all      ::1/128      scram-sha-256
```

After the modify, the content of pg_hba.conf should be like this (example):

> **Note**
> In a Linux environment, a local connection is required to upgrade from version 5.5.1 or earlier to version 5.6.0 or later.

none

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local         DATABASE  USER  METHOD  [OPTIONS]
# host          DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl     DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostgssenc    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnogssenc  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
local    all all                    scram-sha-256
hostssl all all      ::1/128      scram-sha-256

#More content.......
```

4) Reload the services

- Standalone DB mode: Restart **ometascan** service (MetaDefender Core service)
- Shared DB mode: Restart **ometascan-pg** / **ometascan-postgresql** service (MetaDefender Core PostgreSQL service) and then **ometascan** service (MetaDefender Core service)

Test your SSL database connection:

bash

```
psql.exe "sslmode=require dbname=postgres user=postgres host=localhost port=5432"
Password for user postgres:
psql (12.7)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
Type "help" for help.
```

This message indicate that you are now using SSL connection:

bash

```
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
```

Performance and Load Estimation

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Some factors that affect performance

- MetaDefender Core version
- MetaDefender Core engine package and configuration
  - set of engines (which and how many)
  - product configuration (e.g., thread pool size)
- System environment
  - server profile (CPU, RAM, hard disk)
  - client application location - remote or local
  - system caching and engine level caching
- Dataset
  - encrypted or decrypted
  - file types
    - different file types (e.g., document, image, executable)
    - archive file or compound document format files
  - file size
  - bad or unknown (assume to be clean)
- Performance tool

How test results are calculated

Performance (mainly scanning speed) is measured by throughput rather than unit speed. For example, if it takes 10 seconds to process 1 file, and it also takes 10 seconds to process 10 files, then performance is quantified as 1 second per file, rather than 10 seconds.

- total time / total number of files processed: 10 seconds / 10 files = 1 second / file.

Standalone DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.2.416.0

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

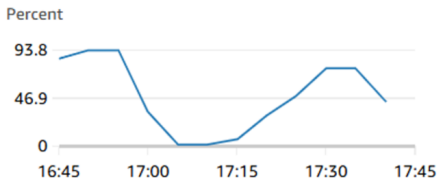
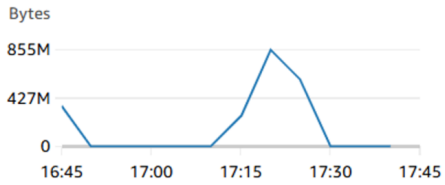
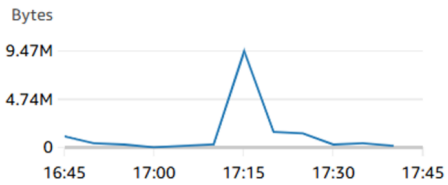
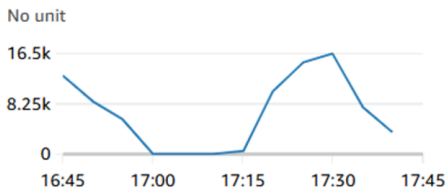
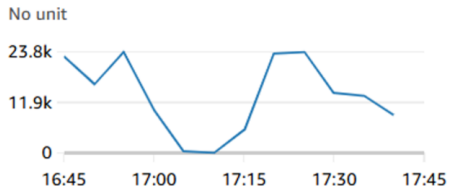
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV)	24.2	47.2 / 80	37.3 / 41.1		315,310	0.0114
Deep CDR	19.9	48.9 / 81.8	32.6 / 37		383,442	0.0094
Proactive DLP	17.1	36.8 / 58.2	30.4 / 33.6		446,228	0.0081
Vulnerability	10.1	59.2 / 85	29.3 / 33.3		755,495	0.0048

MetaDefender Core with common engine packages:

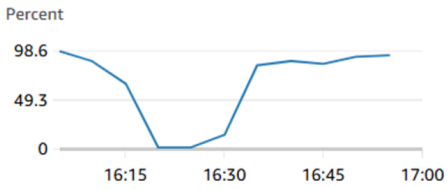
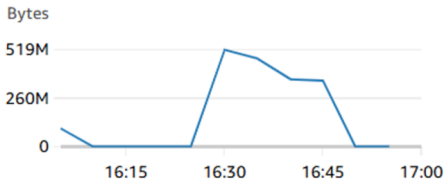
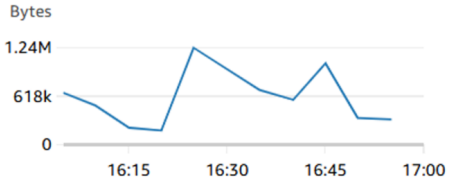
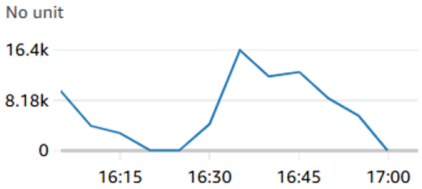
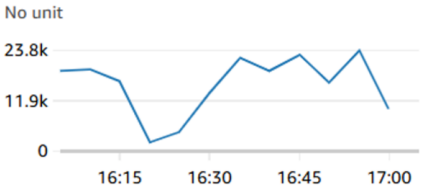
Use case	Scan duration	Avg./Max CPU usage	Avg./Max RAM usage	Avg./Max Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV) + Deep CDR	30.8	68.1 / 88	40.1 / 46.8		247,744	0.0145
Metascan (8 AV) + Deep CDR + Proactive DLP	31.2	74.2 / 98.6	41.5 / 51.1		244,567.3	0.0147
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	31.8	73.5 / 98	40.6 / 51.5		239,953	0.015

System resource utilization charts

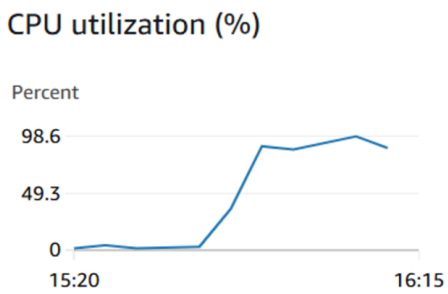
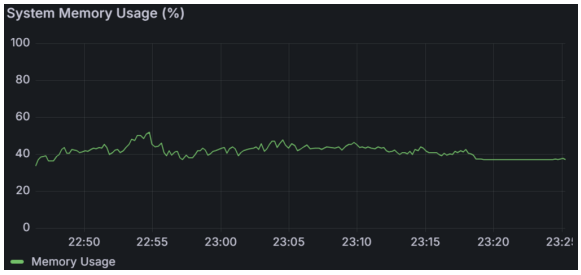
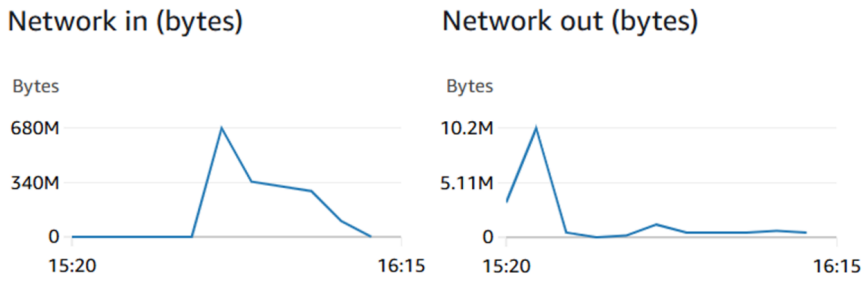
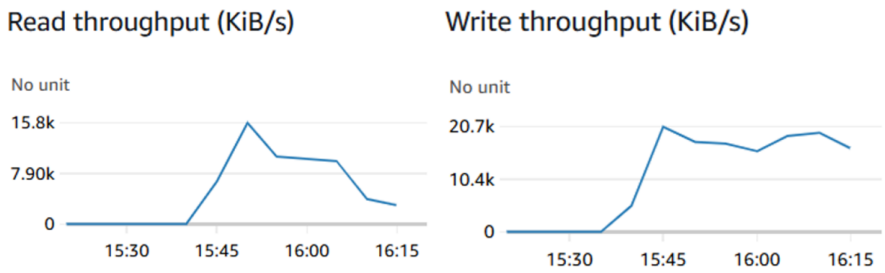
MetaDefender Core with Metascan (8 AV engines):

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

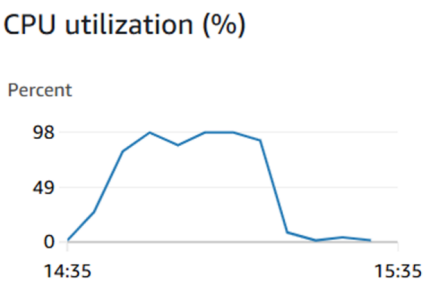
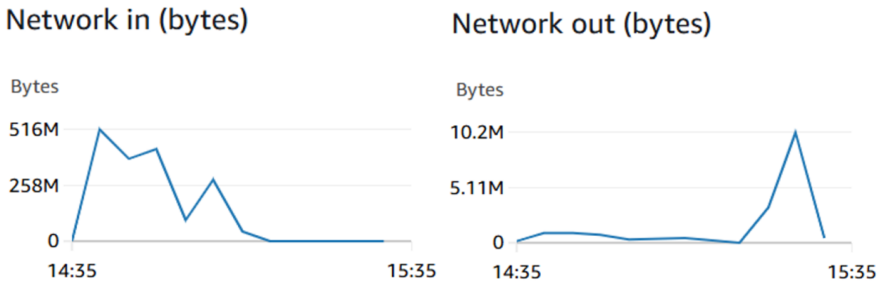
MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

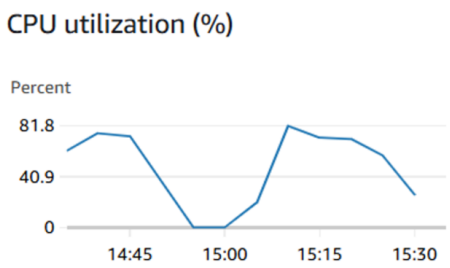
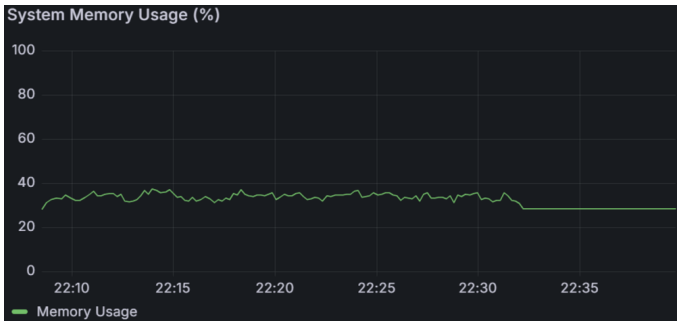
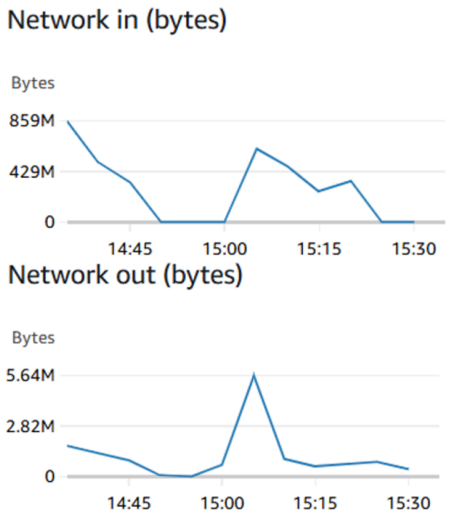
MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

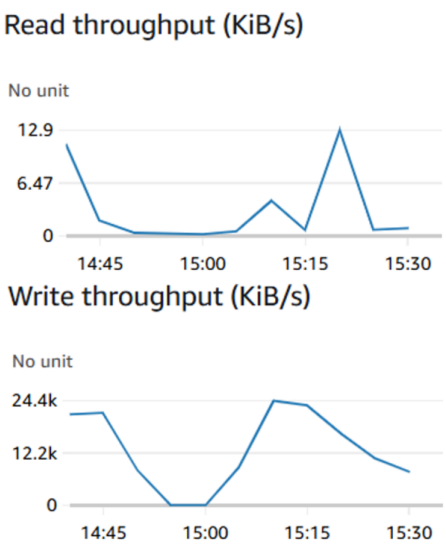
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

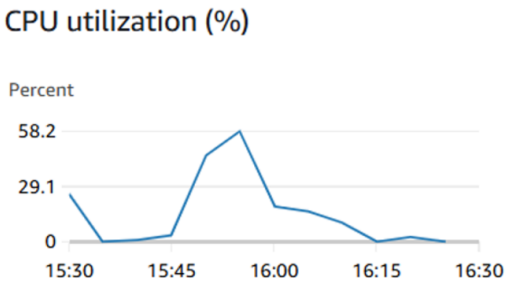
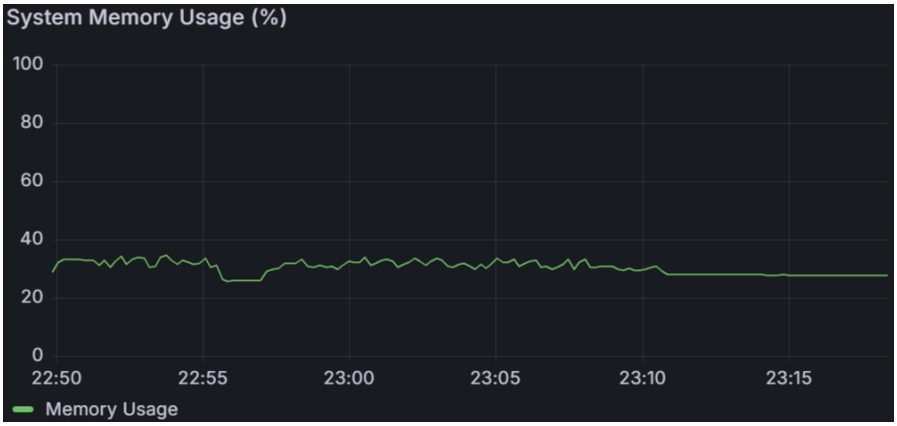
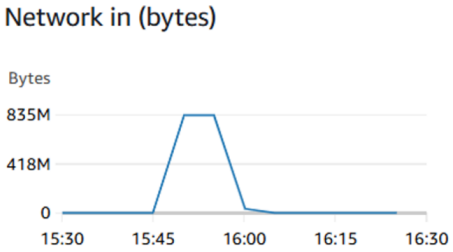
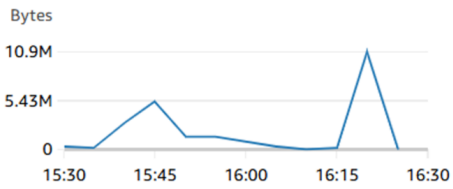
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

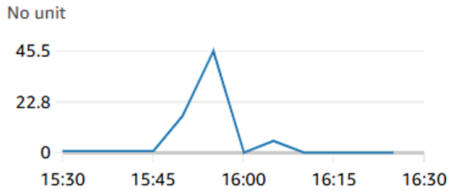
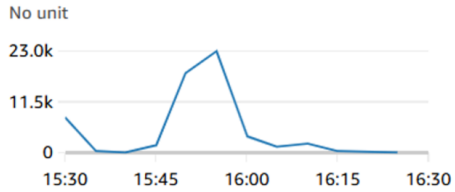
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

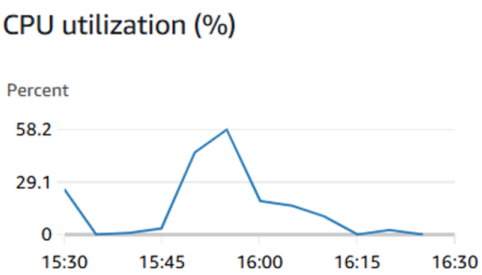
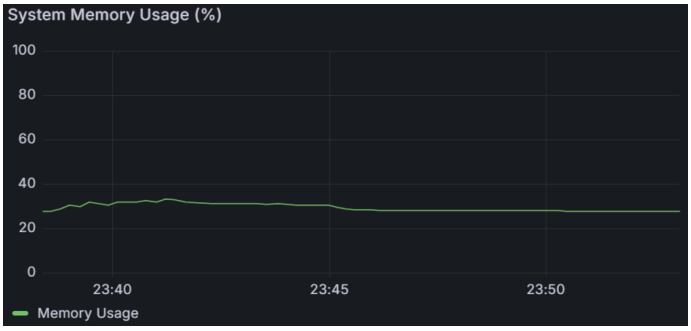
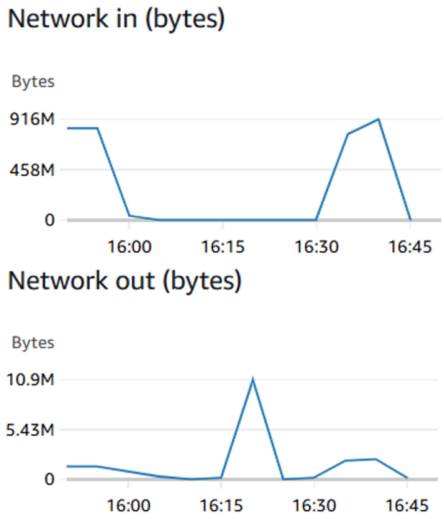
	Utilization charts
Disk usage	

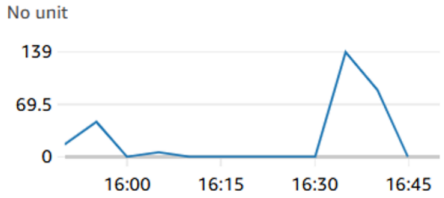
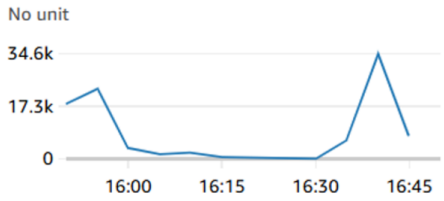
MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	 

	Utilization charts
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

MetaDefender Core with Vulnerability:

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

Standalone DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Ubuntu 22.04	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.57-236

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

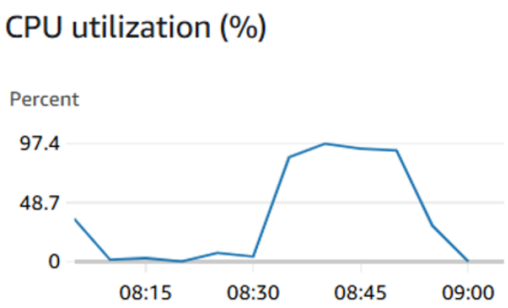
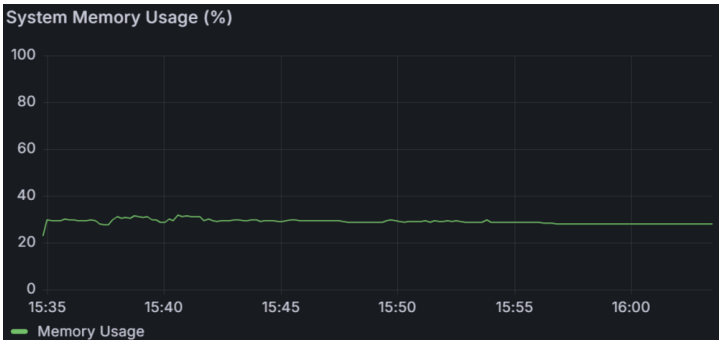
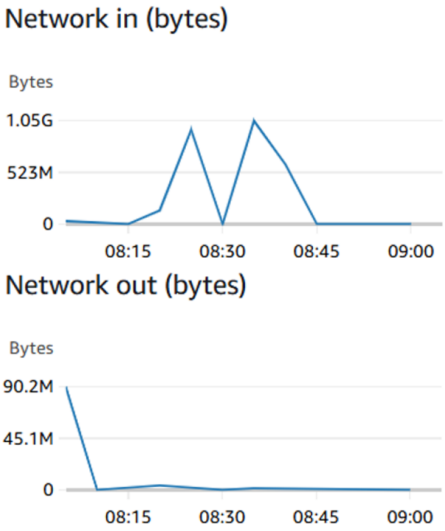
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV)	30.2	69 / 98	29 / 32		252,666	0.0142
Deep CDR	19.4	48 / 62	36 / 49		393,325	0.009
Proactive DLP	16.5	34 / 39	32 / 48		462,455	0.008
Vulnerability	12	50 / 65	46 / 46		635,875	0.006

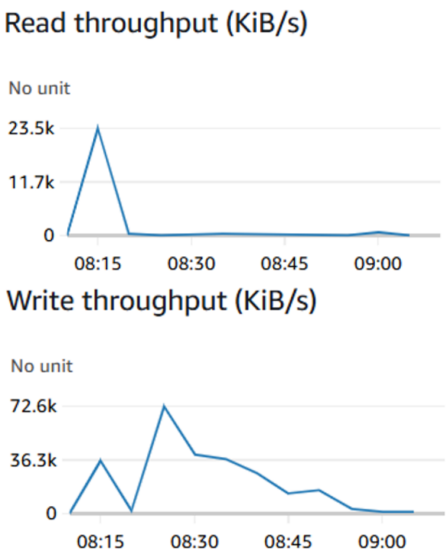
MetaDefender Core with common engine packages:

Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV) + Deep CDR	39.8	78 / 99	62 / 73		191,721	0.0188
Metascan (10 AV) + Deep CDR + Proactive DLP	43.5	61 / 84	47 / 67		175,414	0.0205
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	44.8	62 / 88	69 / 83		170,324	0.0211

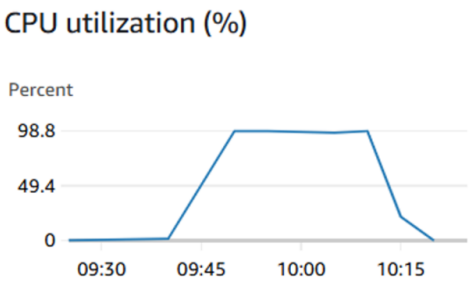
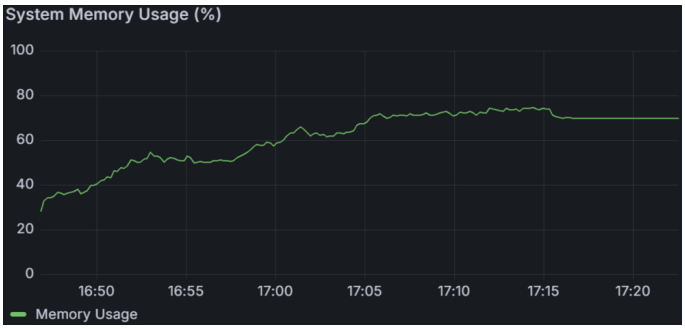
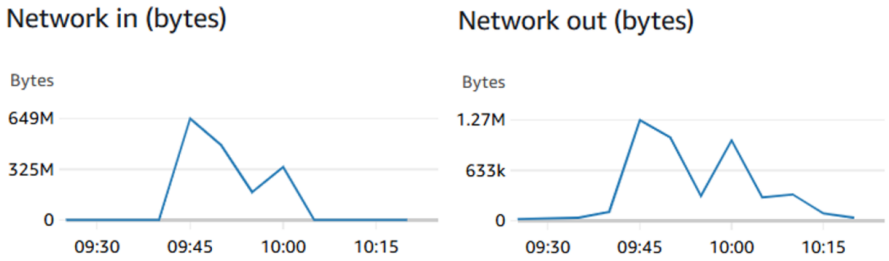
System resource utilization charts

MetaDefender Core with Metascan (10 AV engines) :

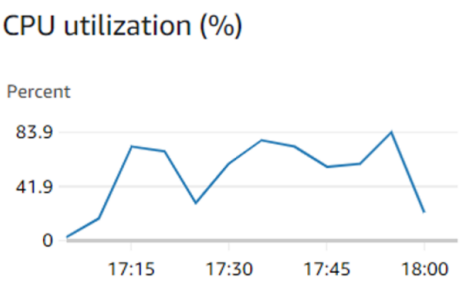
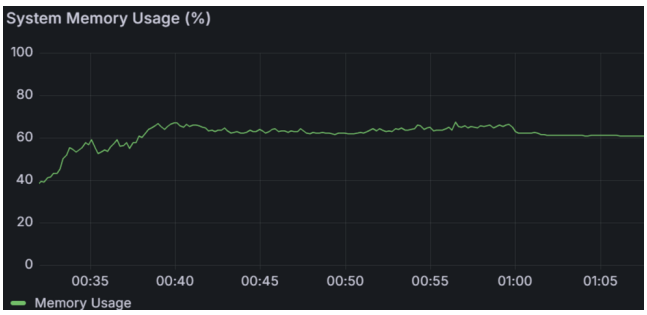
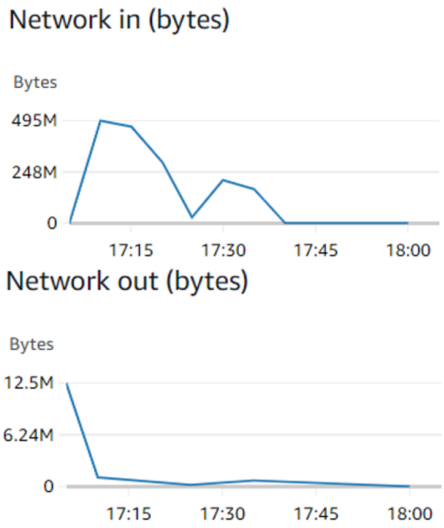
	Utilization charts
CPU usage	
Memory usage	
Network usage	

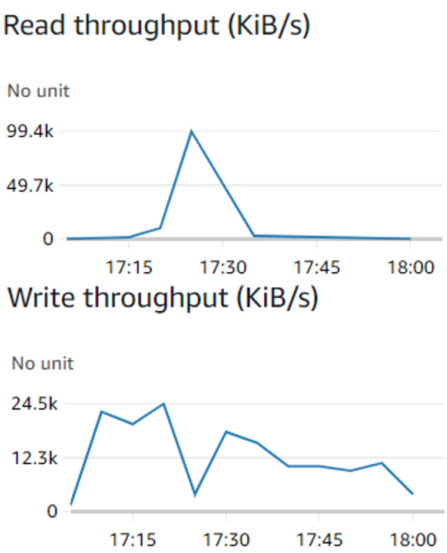
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR :

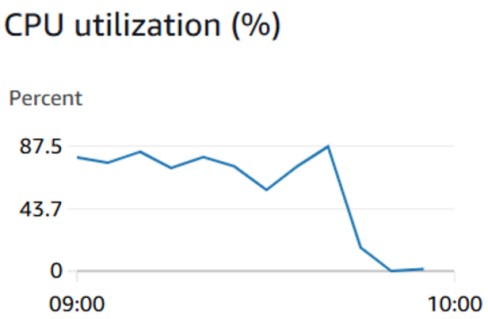
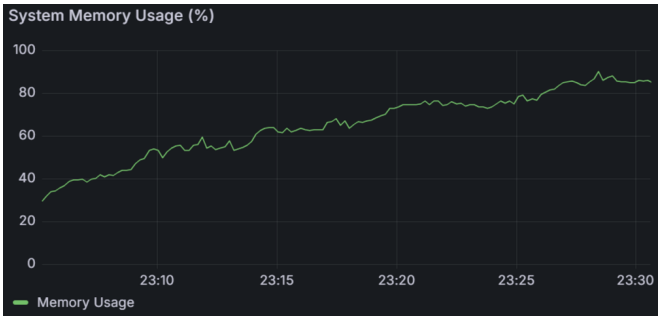
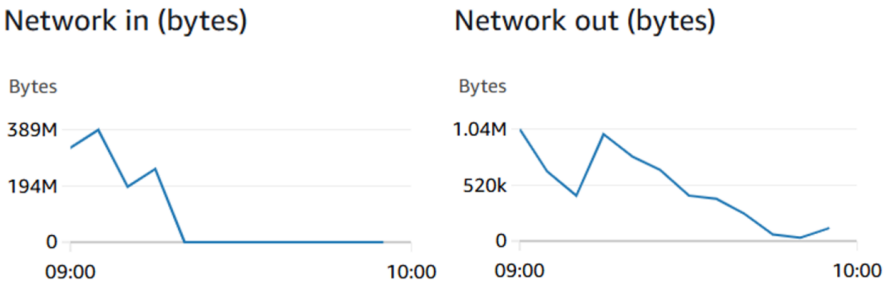
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP :

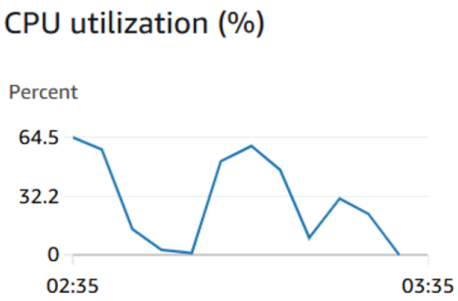
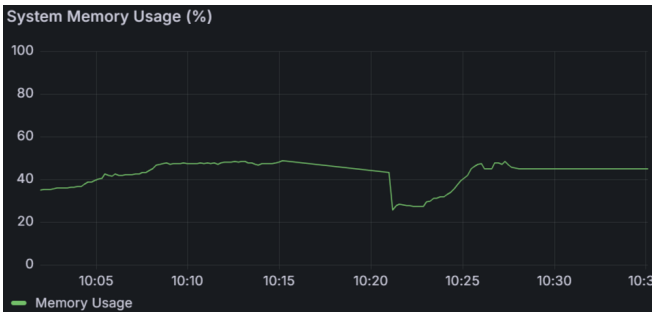
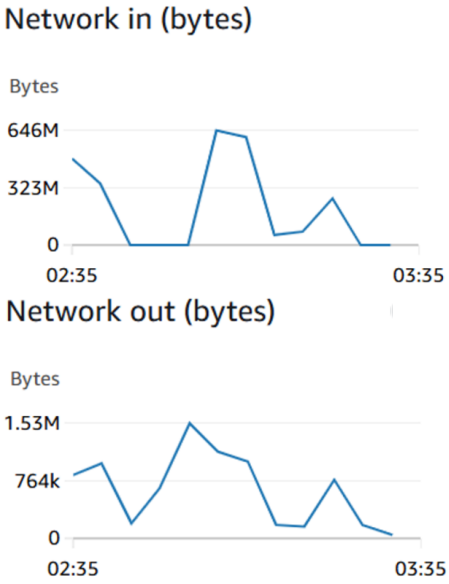
	Utilization charts
CPU usage	
Memory usage	
Network usage	

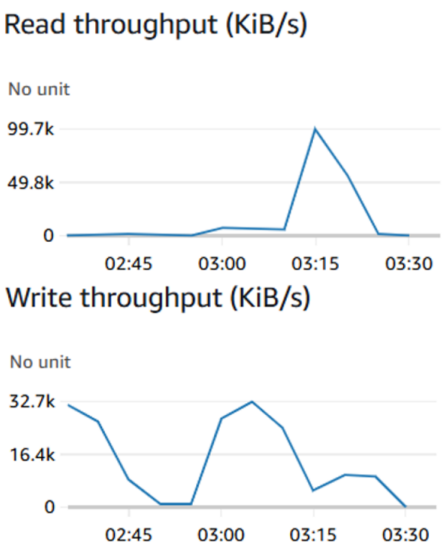
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP + Vulnerability :

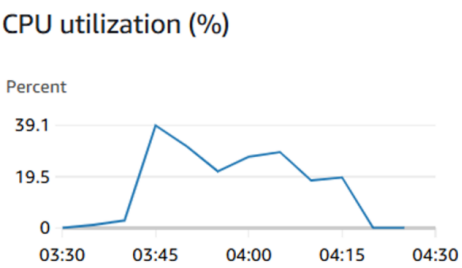
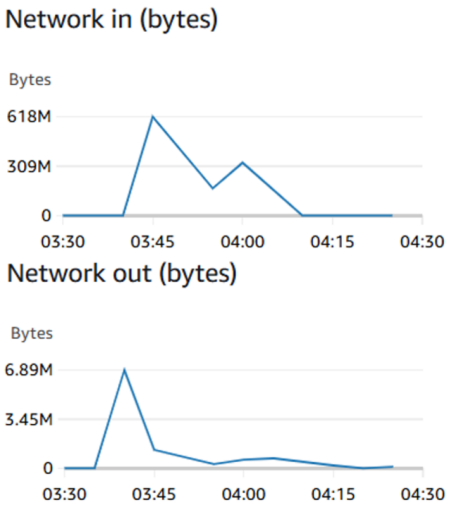
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

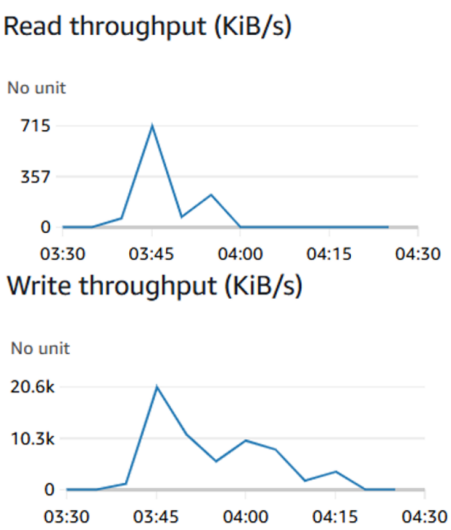
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

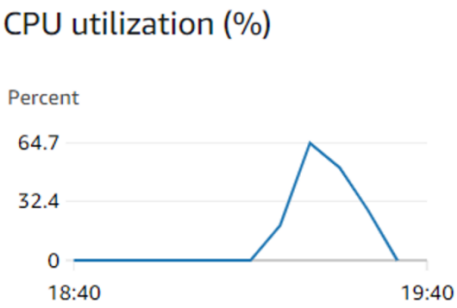
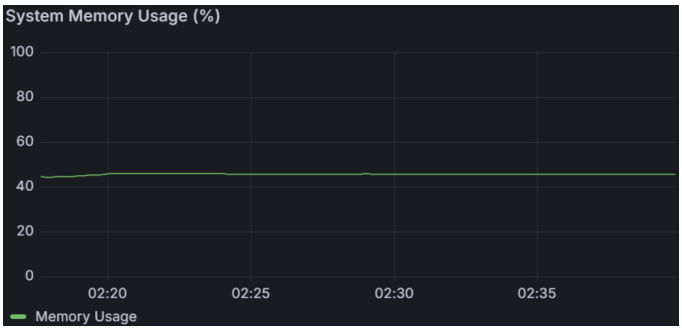
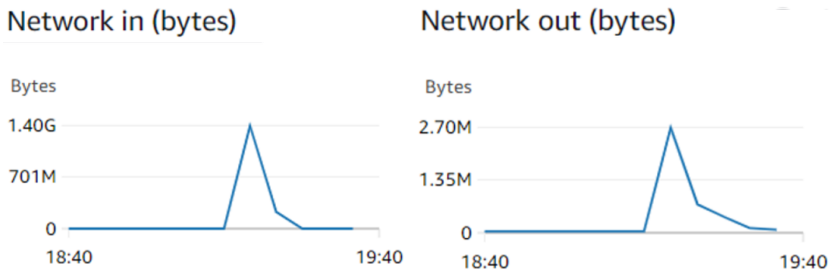
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability :

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

Shared DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

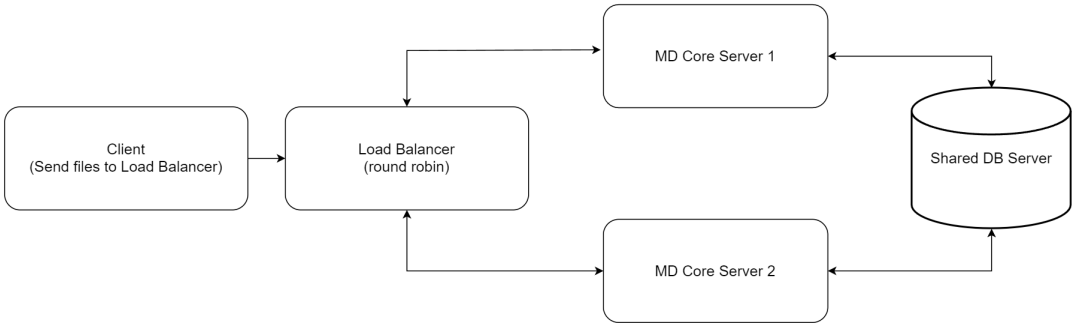
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV)	11.9	37.0	41.9	30.1	30.4	Receive: 6441.1 Send: 769.6	Receive: 6200.2 Send: 693.7	640,821.2	0.0
Deep CDR	10.5	60.1	66.7	30.3	32.2	Receive: 6379.8 Send: 594.8	Receive: 7146.1 Send: 609.6	729,368.5	0.0
Proactive DLP	7.8	38.7	37.5	29.7	31.0	Receive: 8842.6 Send: 791.8	Receive: 8635.6 Send: 729.1	965,455.7	0.0
Vulnerability	4.4	52.2	53.0	26.0	26.9	Receive: 16,215.5 Send: 1006.8	Receive: 15,785.7 Send: 997.2	1714,449.3	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV) + Deep CDR	15.4	72.4	75	34.3	35.4	Receive: 5436.6 Send: 611.3	Receive: 5589.3 Send: 607.2	495,697.4	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP	16.9	77.2	80.2	36.4	37.9	Receive: 5254 Send: 585.4	Receive: 5041.3 Send: 576.1	450,657.6	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	18.0	72.1	80.4	38.6	38.0	Receive: 4504.3 Send: 574.1	Receive: 4649.5 Send: 571.1	423,010	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

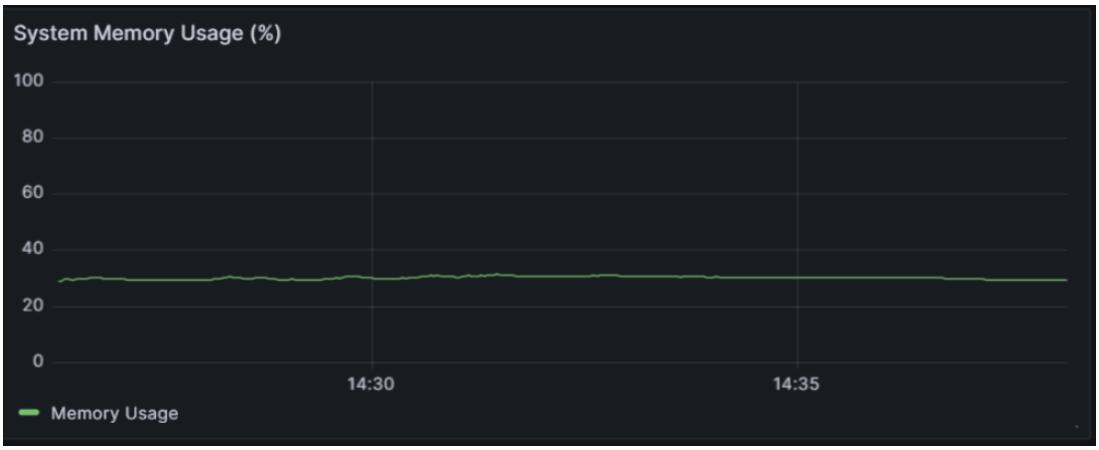
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2

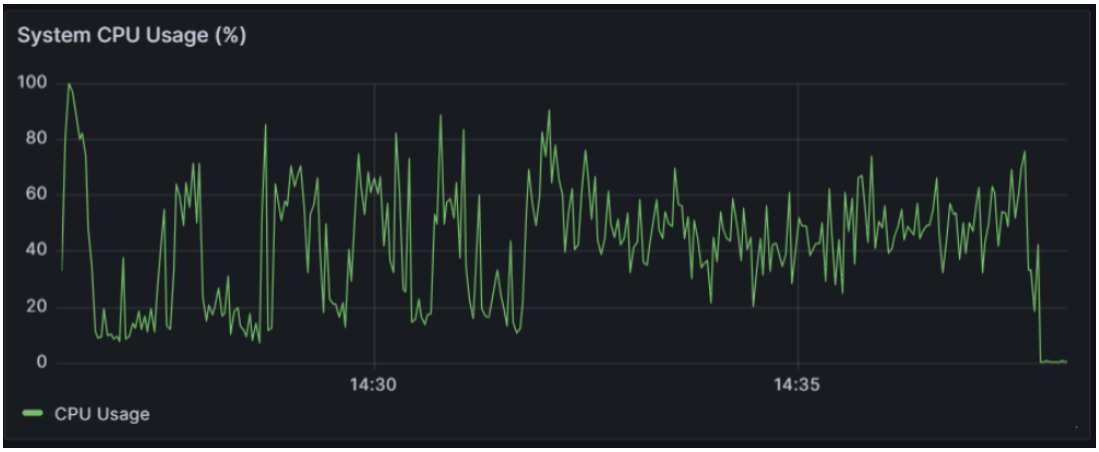


CPU usage

MD Core #1

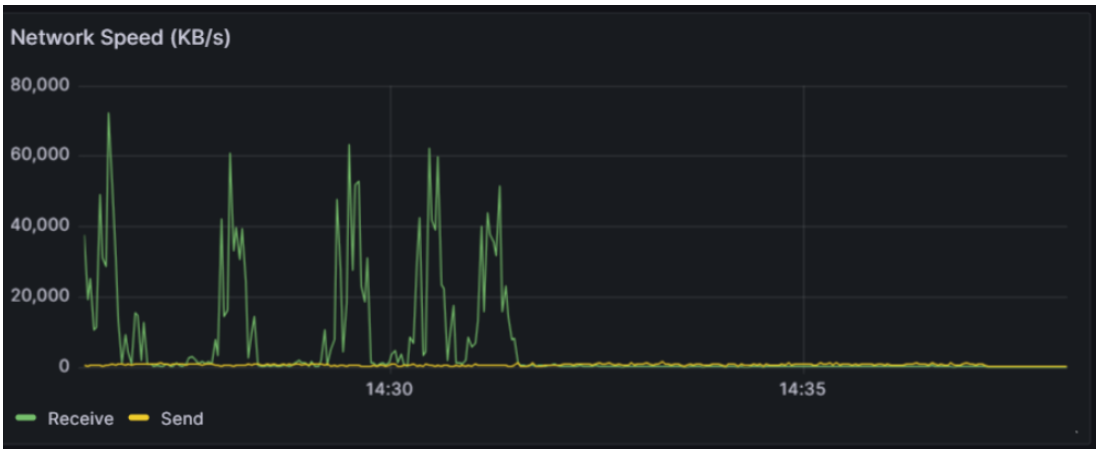


MD Core #2

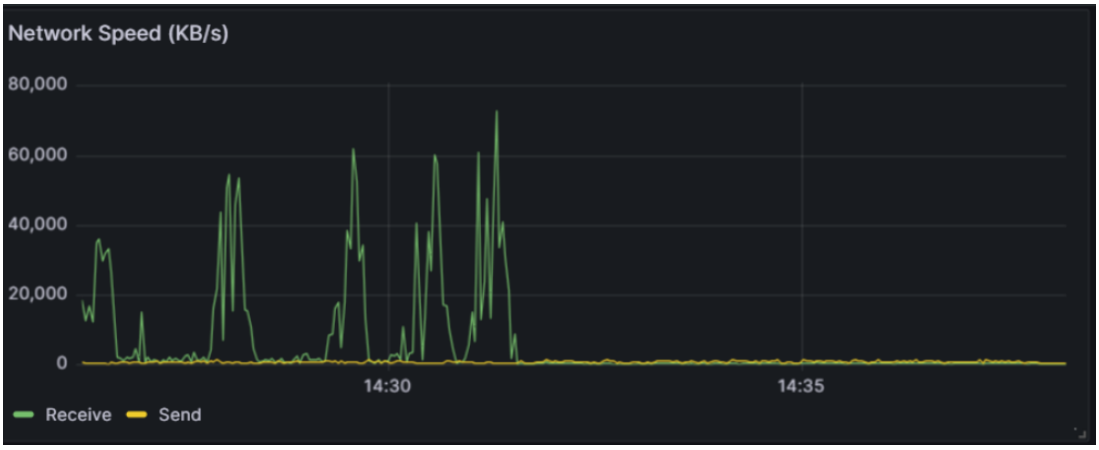


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2

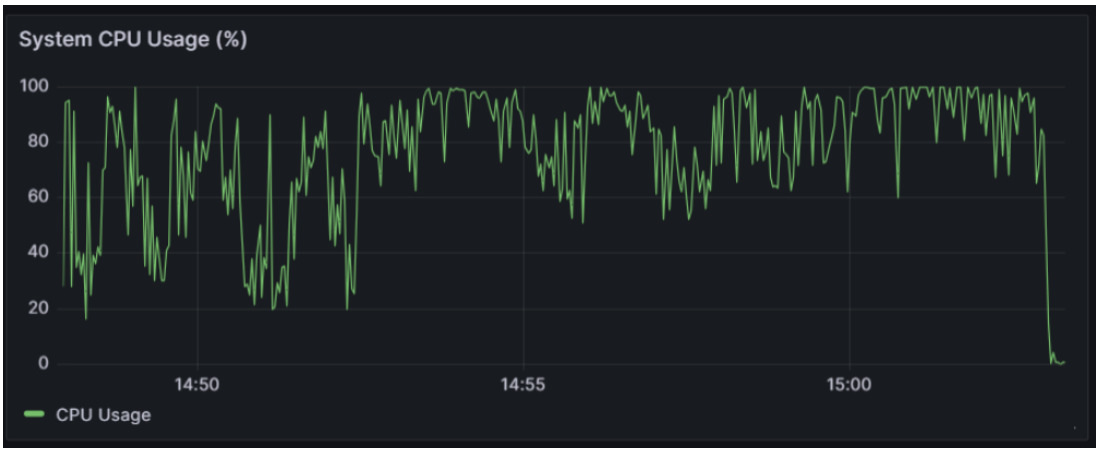


CPU usage

MD Core #1

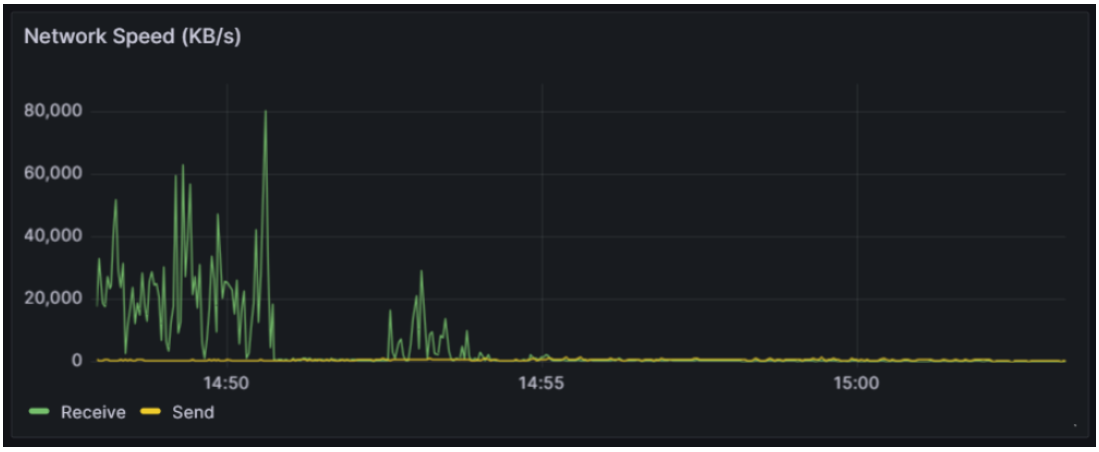


MD Core #2



Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

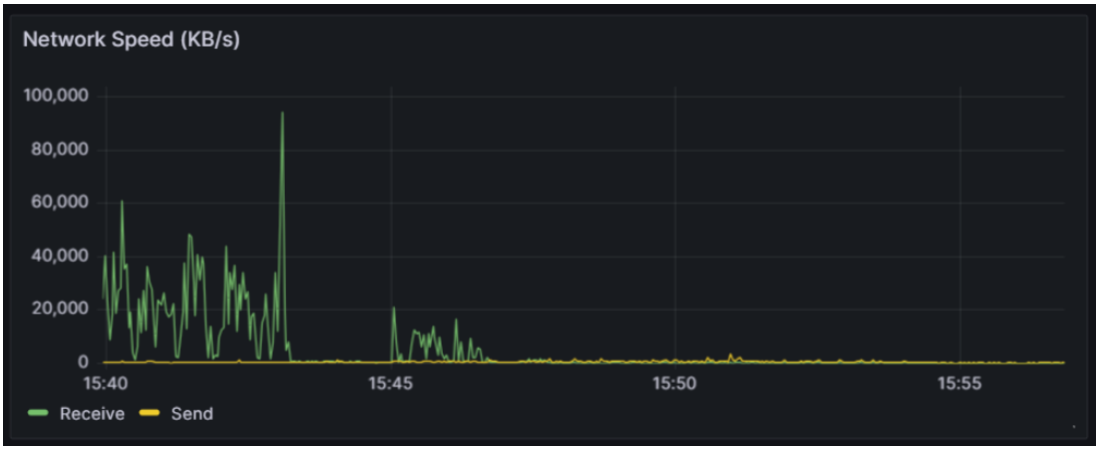


MD Core #2

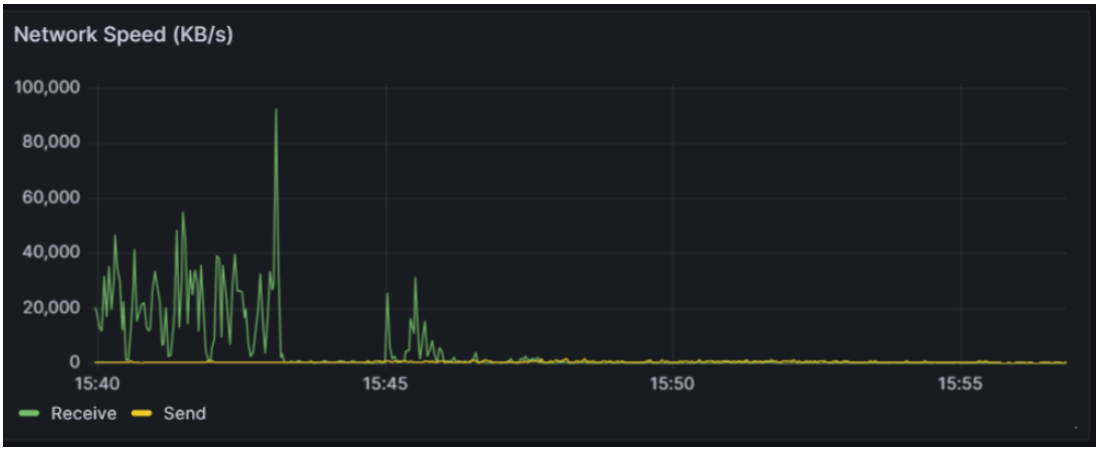


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1

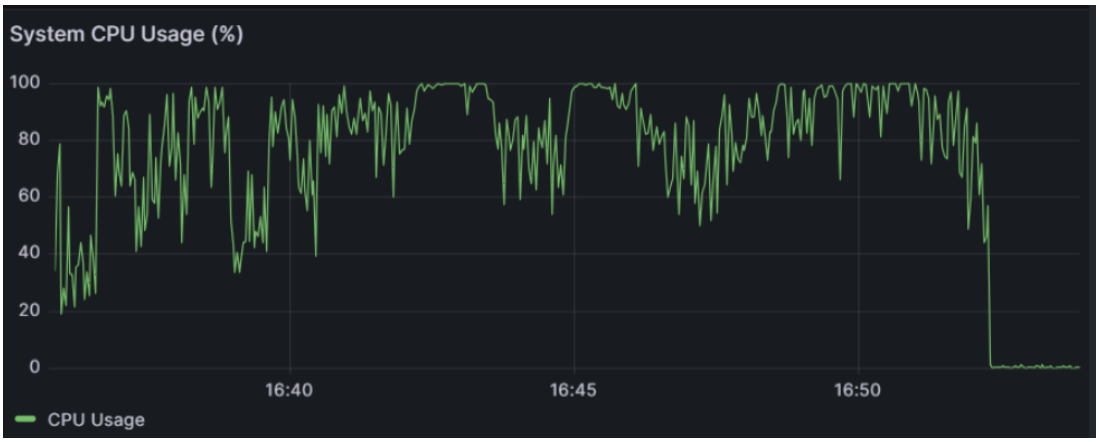


MD Core #2



CPU usage

MD Core #1

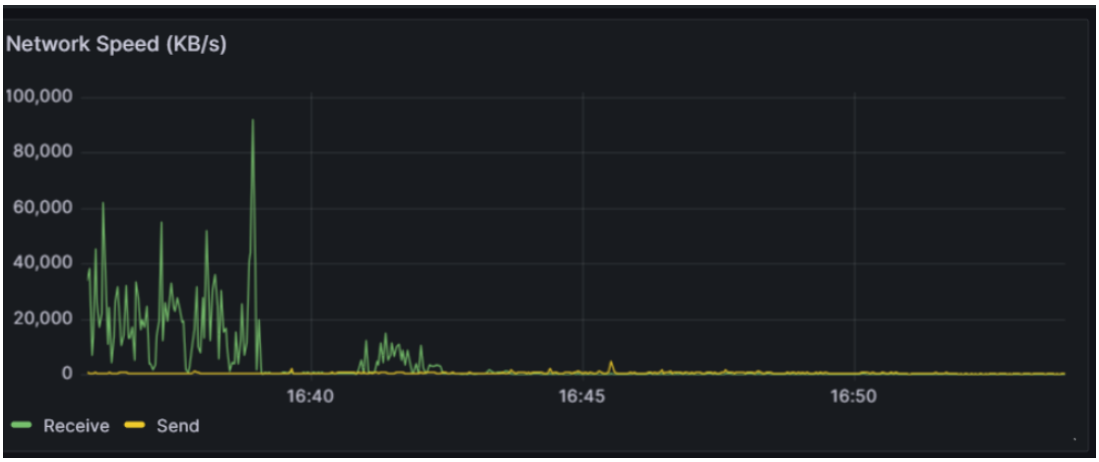


MD Core #2

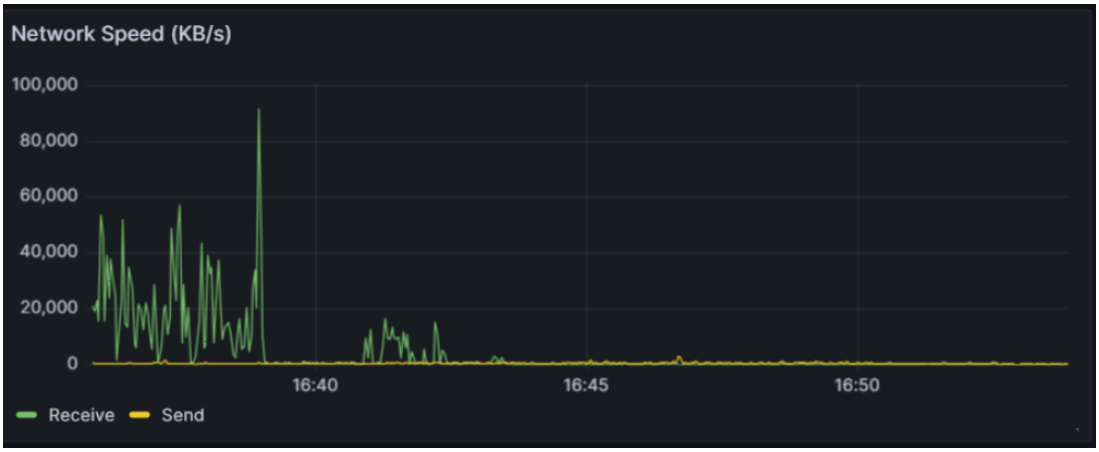


Network usage

MD Core #1



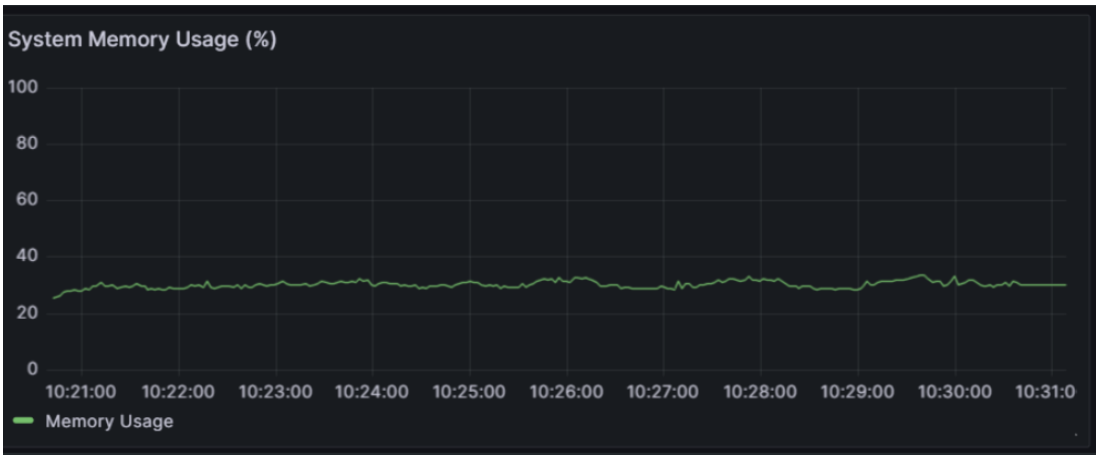
MD Core #2



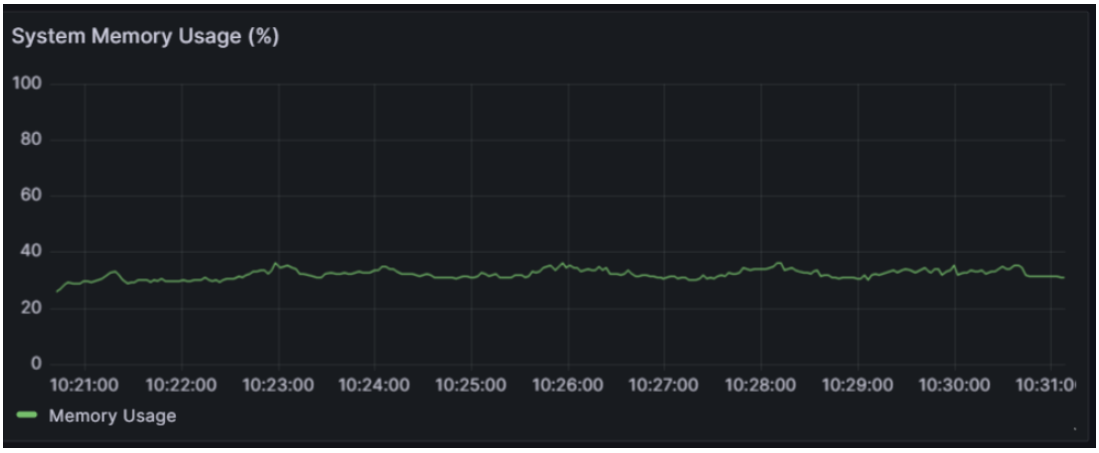
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

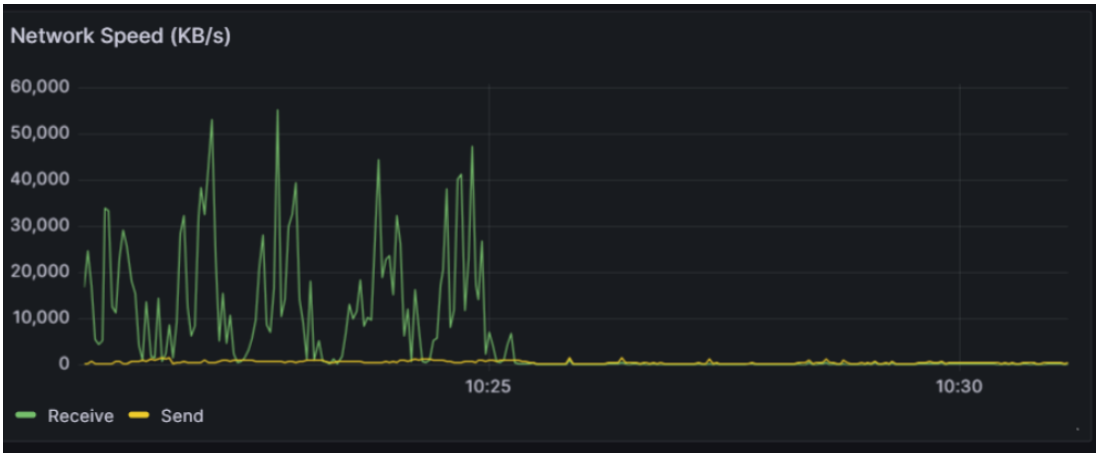


MD Core #2

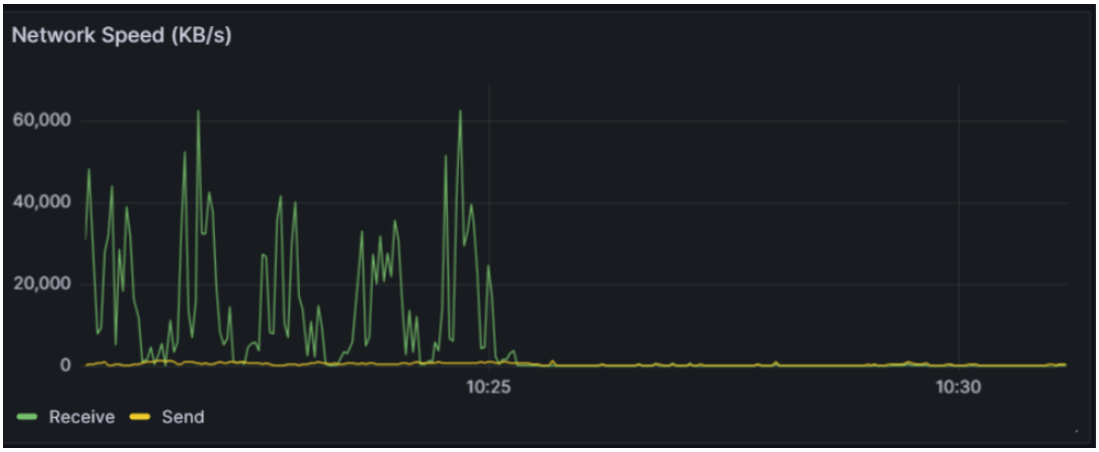


Network usage

MD Core #1



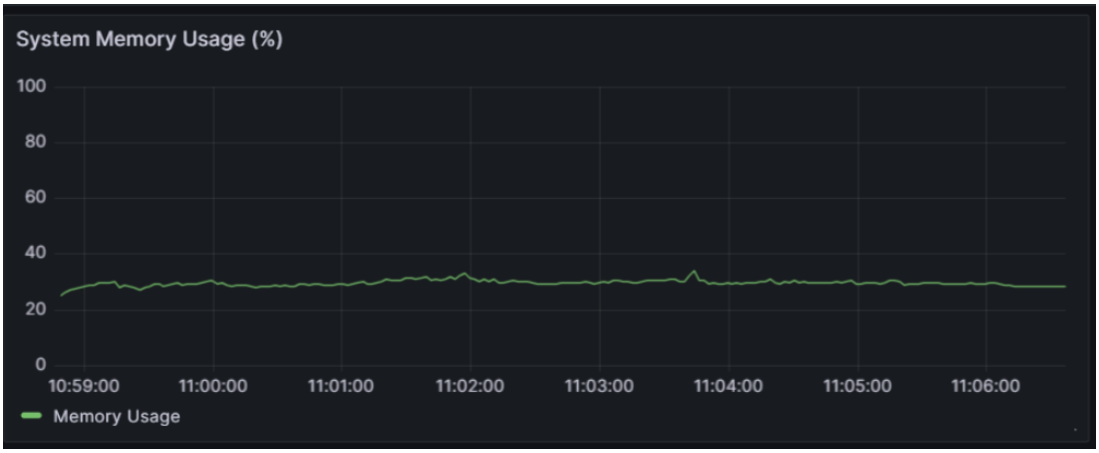
MD Core #2



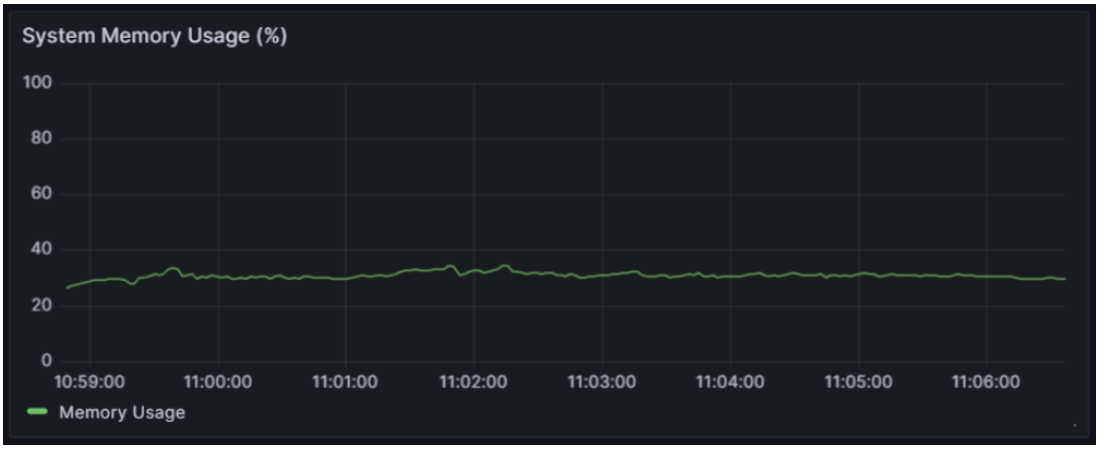
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

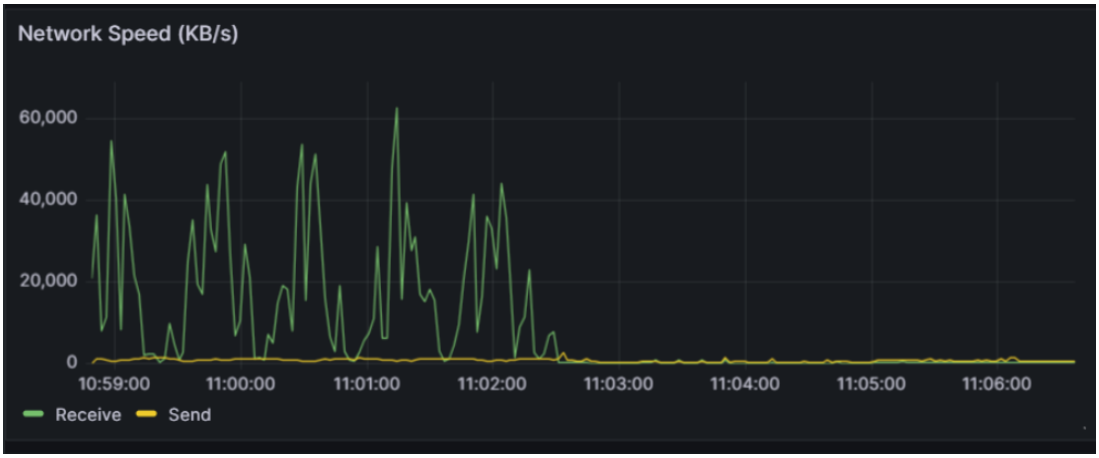


MD Core #2

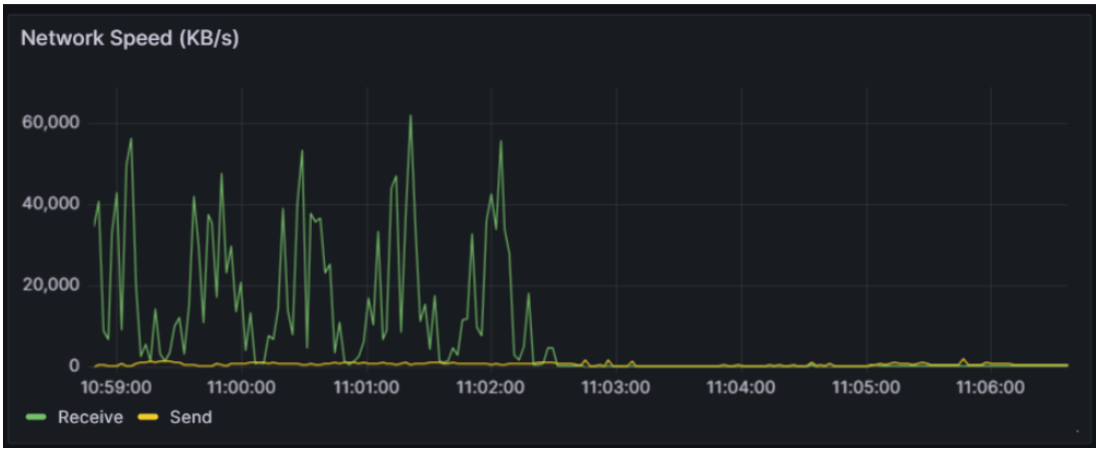


Network usage

MD Core #1



MD Core #2



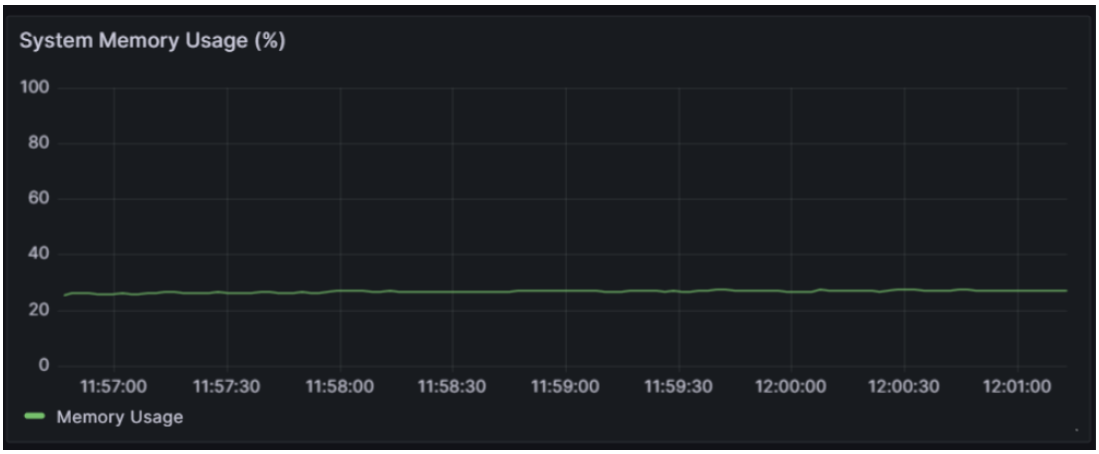
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

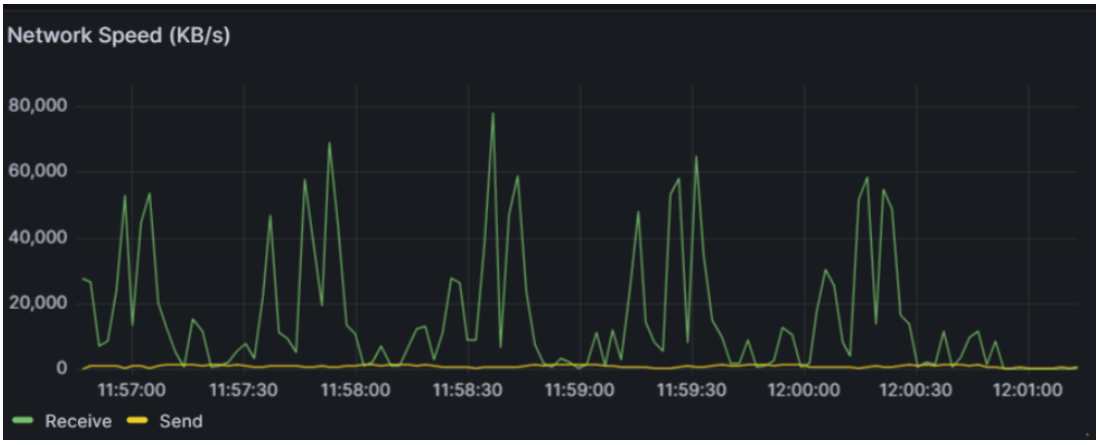


MD Core #2

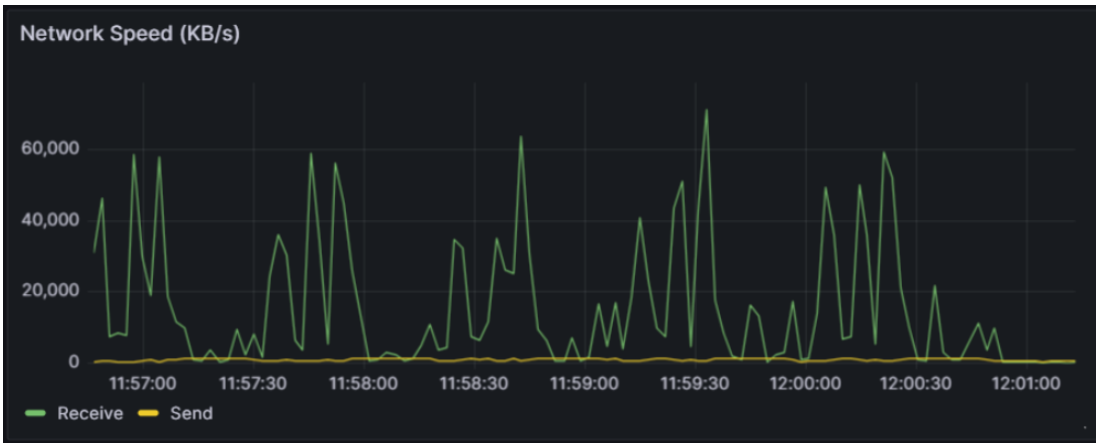


Network usage

MD Core #1



MD Core #2



Shared DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

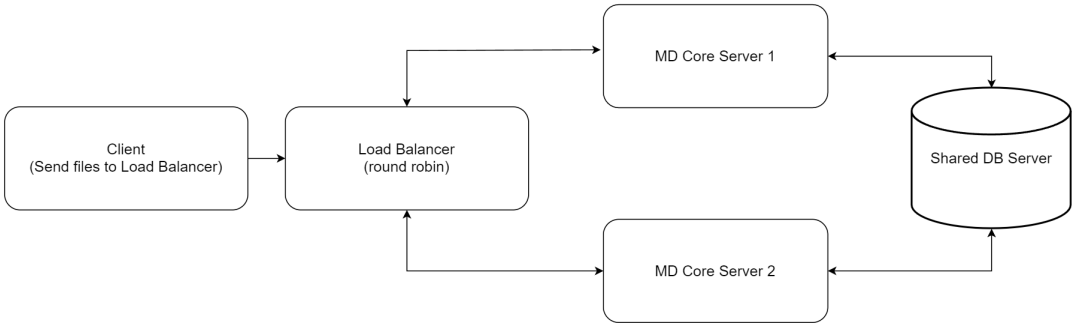
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV)	15.5	63.1	73.3	25.2	25.8	Receive: 4555.9 Send: 694.4	Receive: 4516.2 Send: 692.4	492,620.5	0.0
Deep CDR	9.70	50.7	51.2	31.9	32.5	Receive: 7062.9 Send: 623.2	Receive: 6653 Send: 654.1	786,189.6	0.0
Proactive DLP	7.2	35.2	35.0	27.9	27.5	Receive: 9047.4 Send: 802.4	Receive: 9859.8 Send: 787.5	1,055,110.8	0.0
Vulnerability	5.0	45.7	49.3	24.4	24.5	Receive: 13,585.7 Send: 966.7	Receive: 13,505.8 Send: 987.4	1,519,562.1	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV) + Deep CDR	19.9	79.1	77.6	34.0	34.2	Receive: 3785.9 Send: 573.8	Receive: 3663.1 Send: 577.7	382,716	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP	21.6	80.3	81.5	37.7	44.4	Receive: 3305.6 Send: 543.3	Receive: 3414.3 Send: 556.2	353,694.9	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	22.5	84.2	79.8	37.9	44.7	Receive: 3267.9 Send: 554.9	Receive: 3398.2 Send: 553.8	338,507.6	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

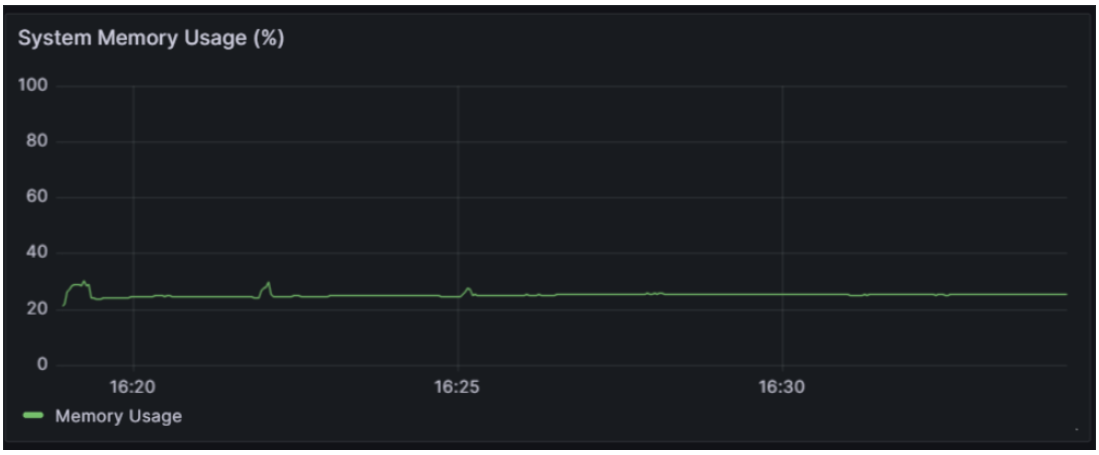
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

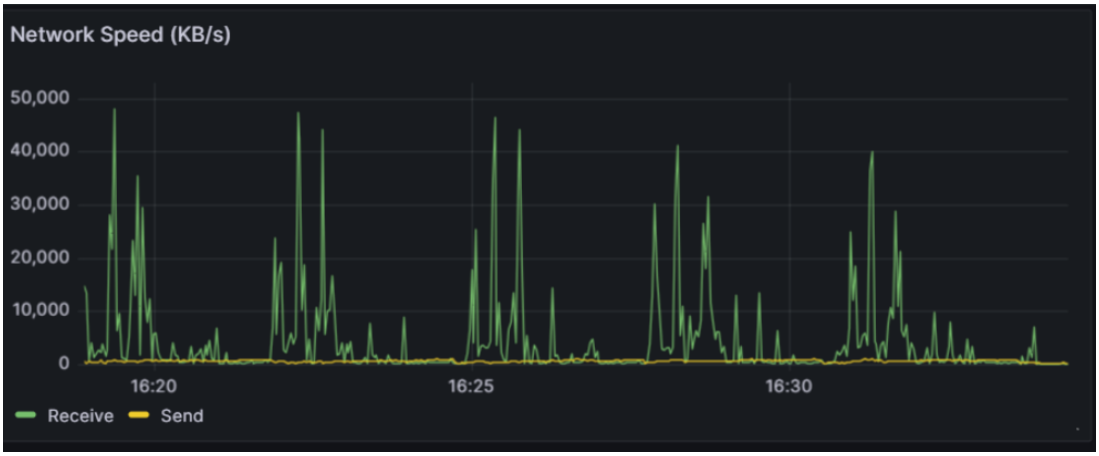


MD Core #2

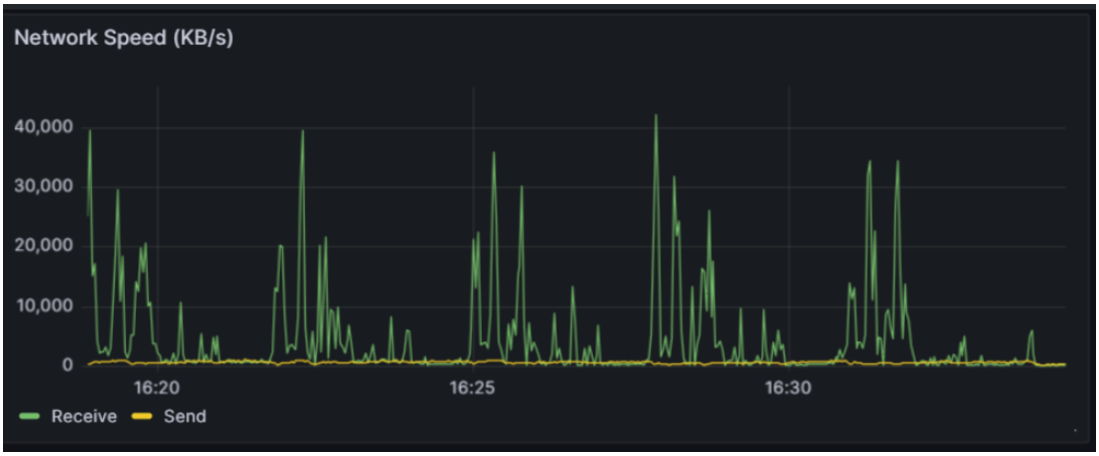


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

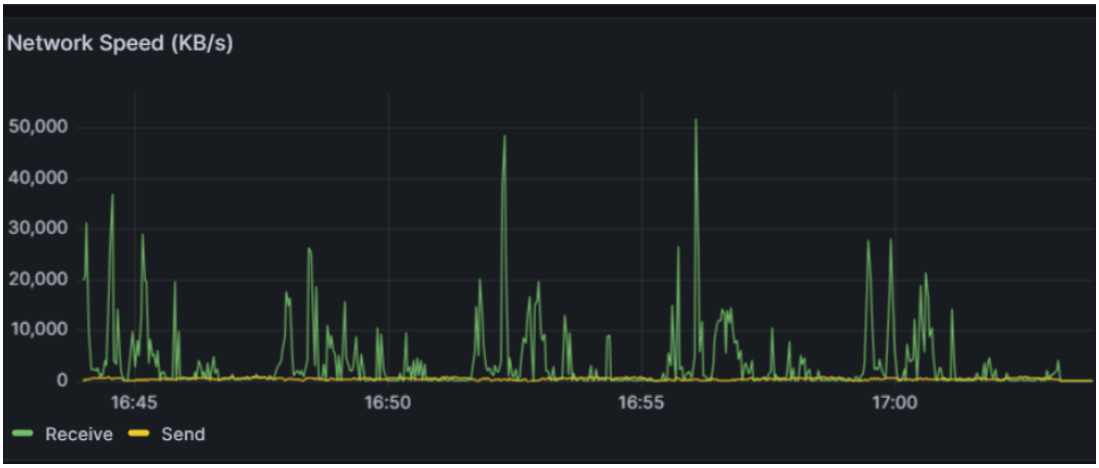


MD Core #2

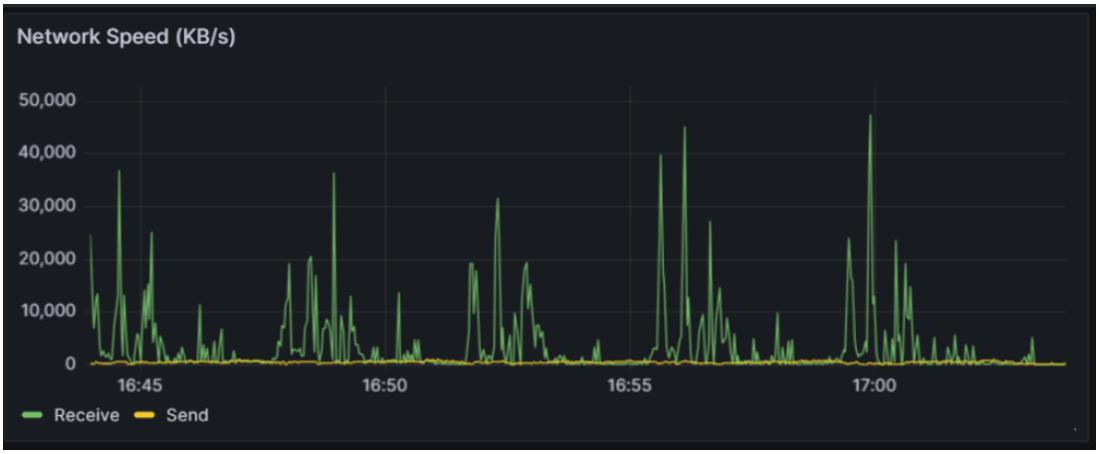


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1

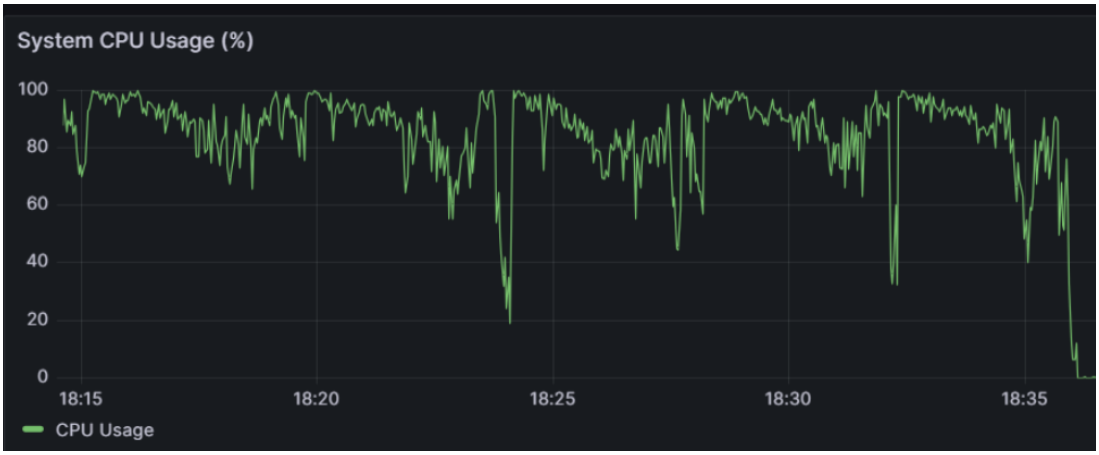


MD Core #2



CPU usage

MD Core #1

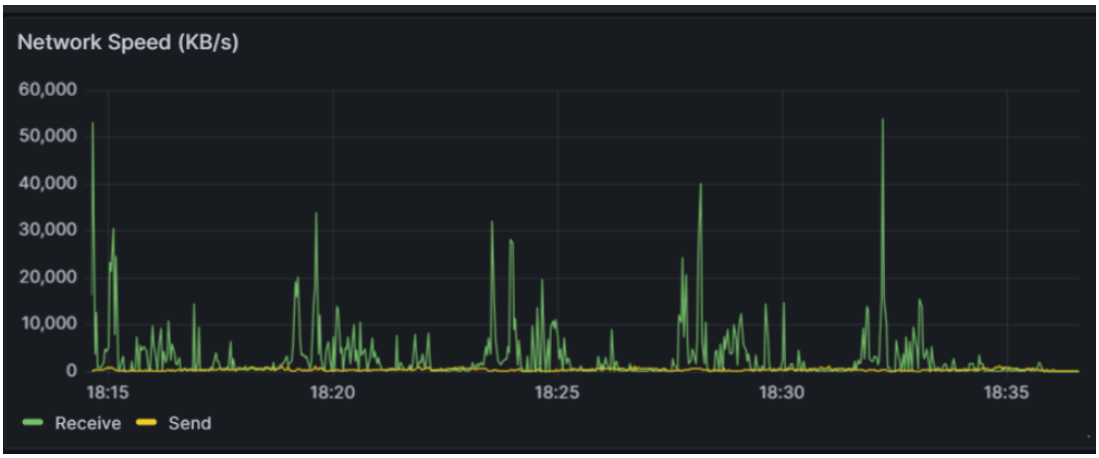


MD Core #2

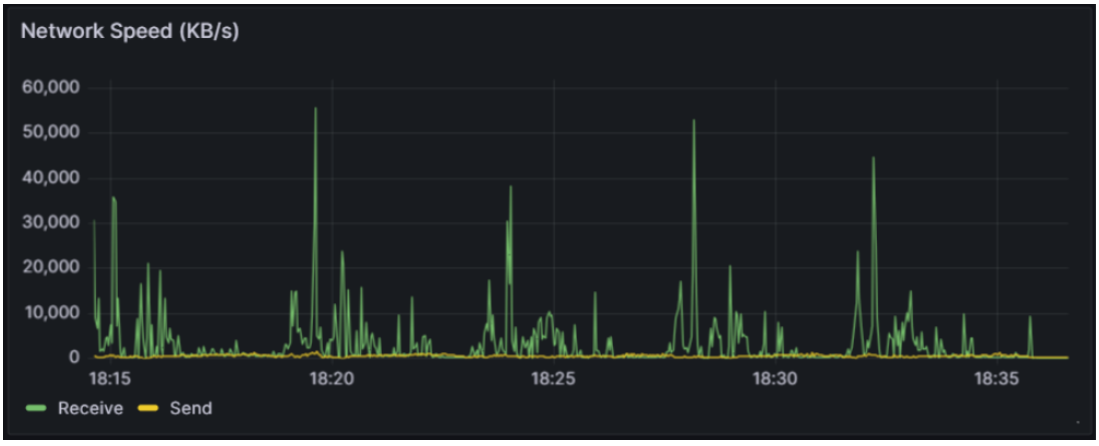


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

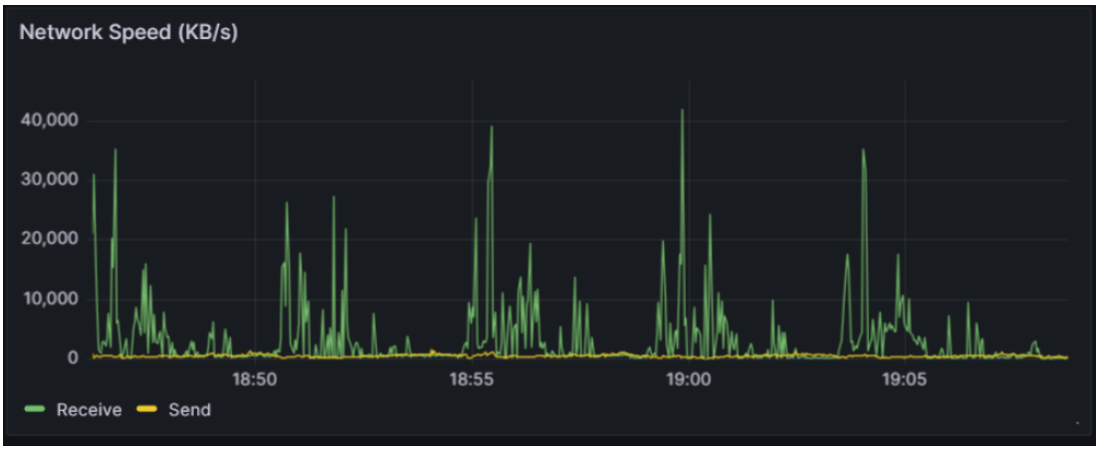


MD Core #2

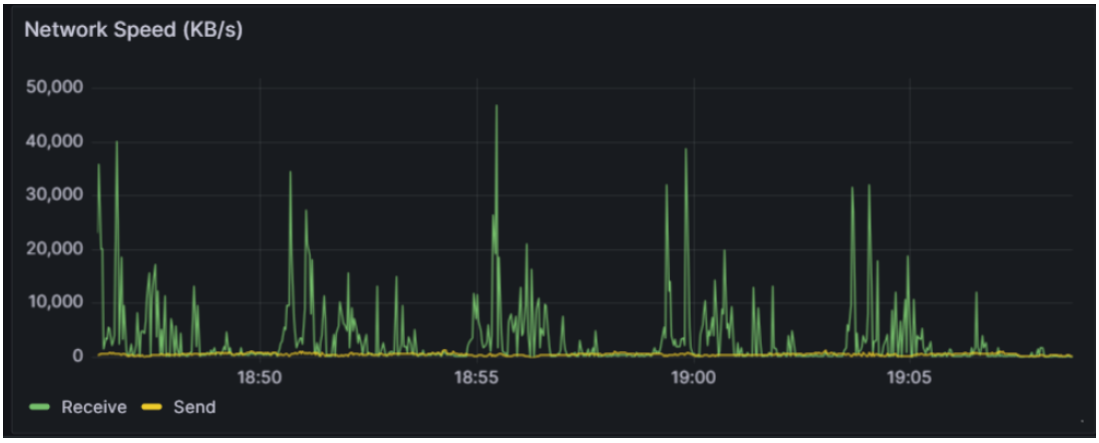


Network usage

MD Core #1



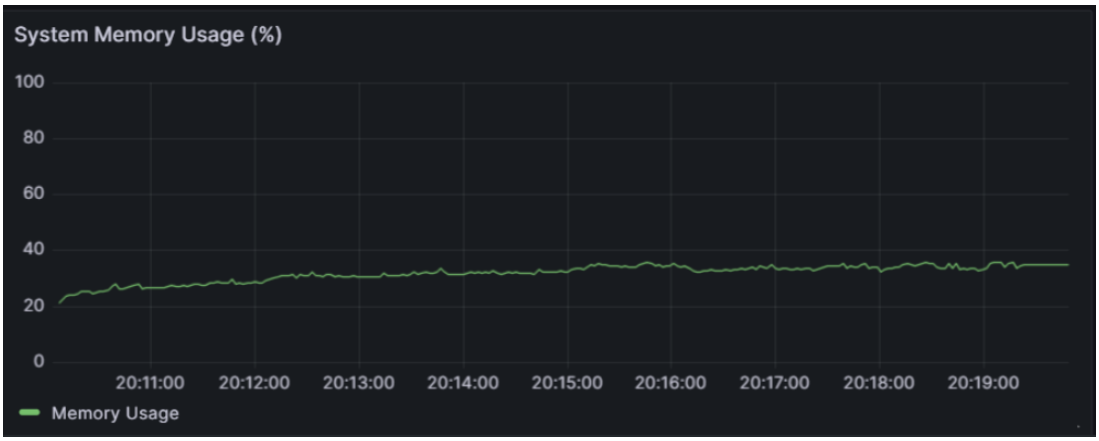
MD Core #2



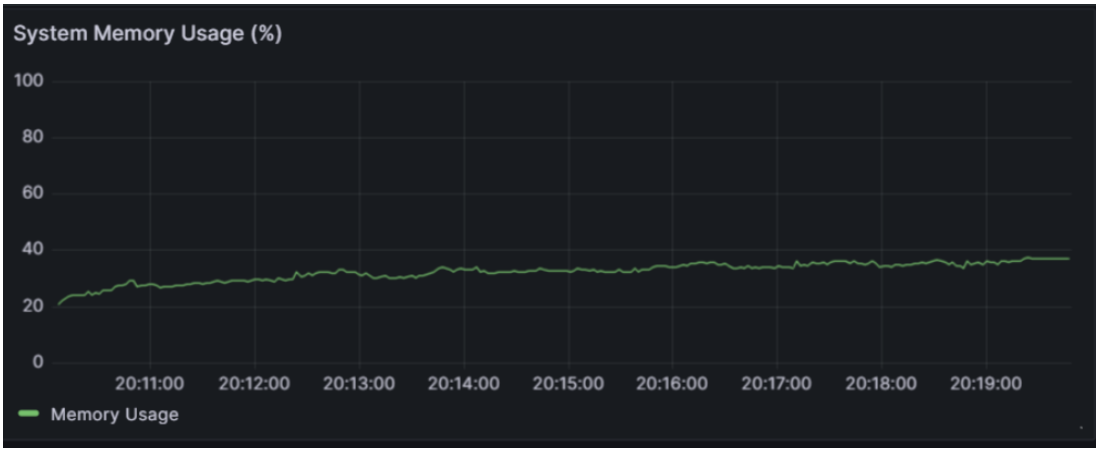
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1

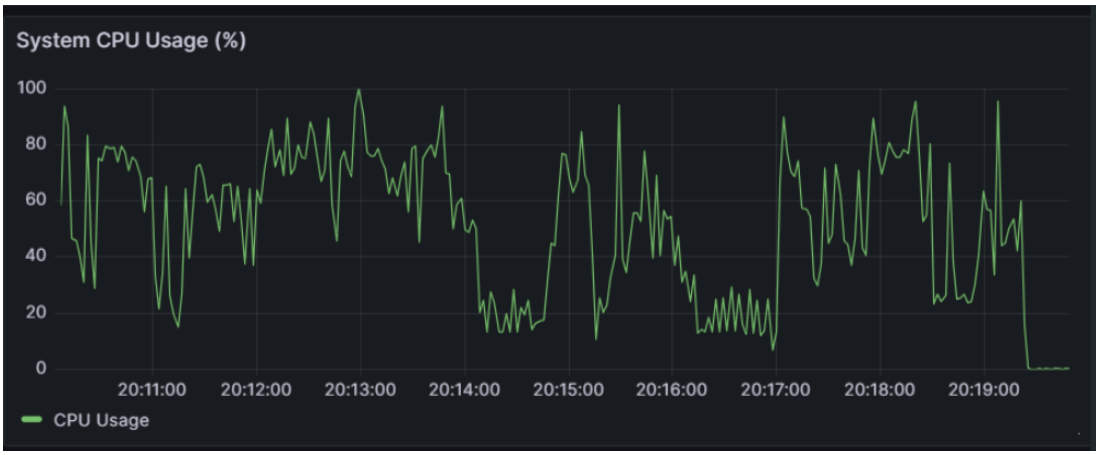


MD Core #2



CPU usage

MD Core #1

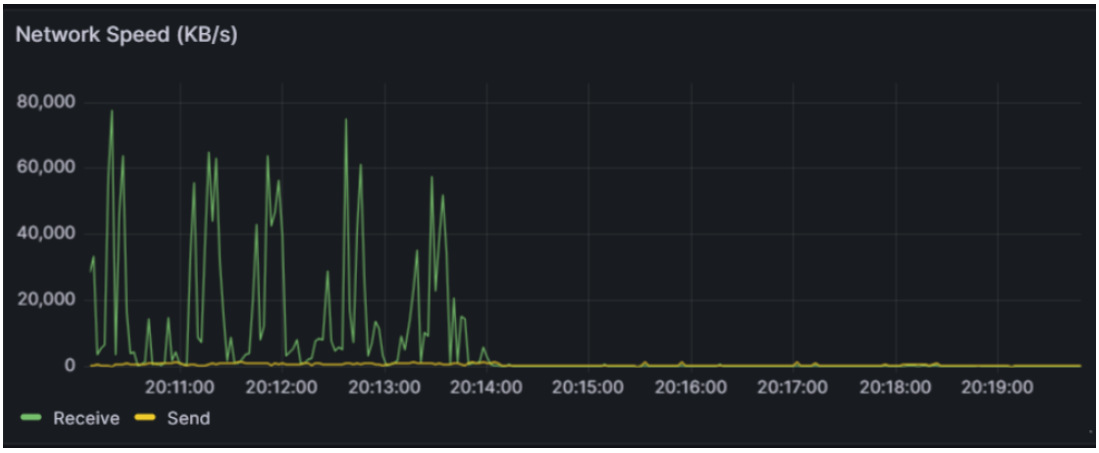


MD Core #2

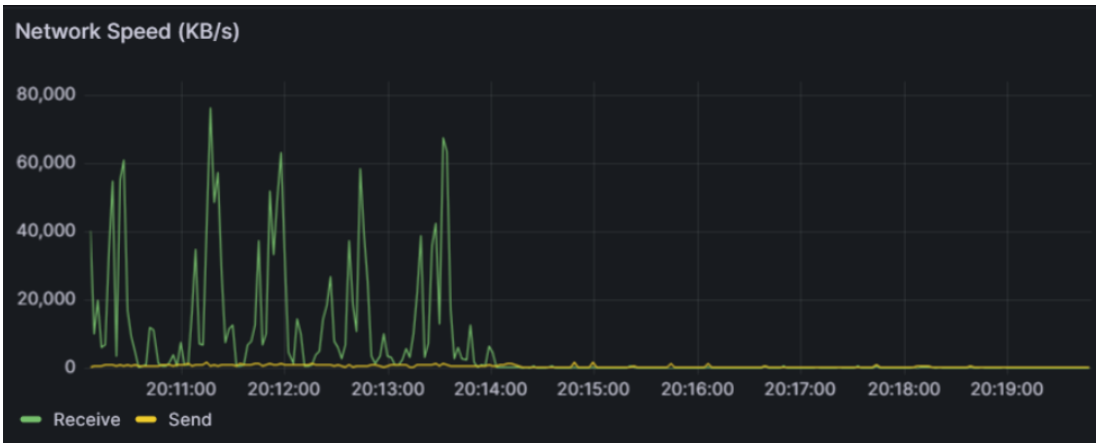


Network usage

MD Core #1



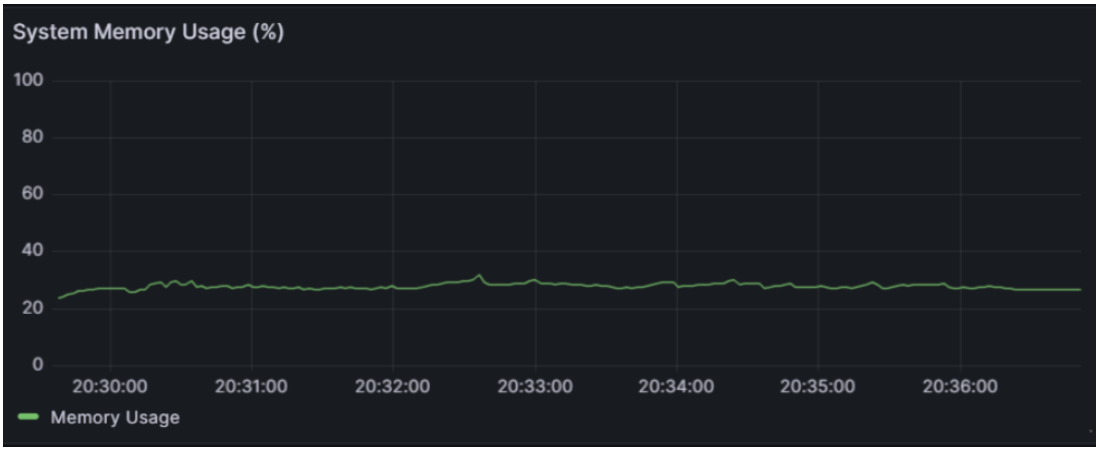
MD Core #2



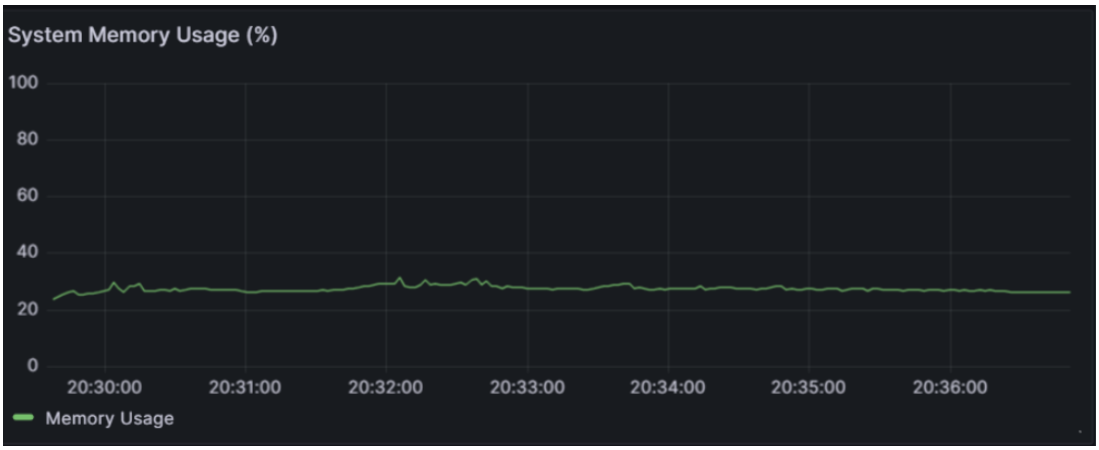
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1

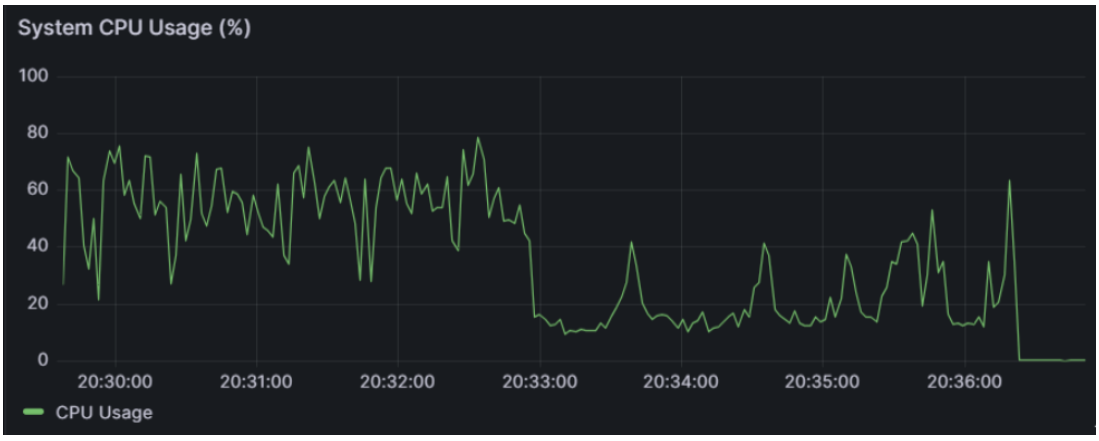


MD Core #2

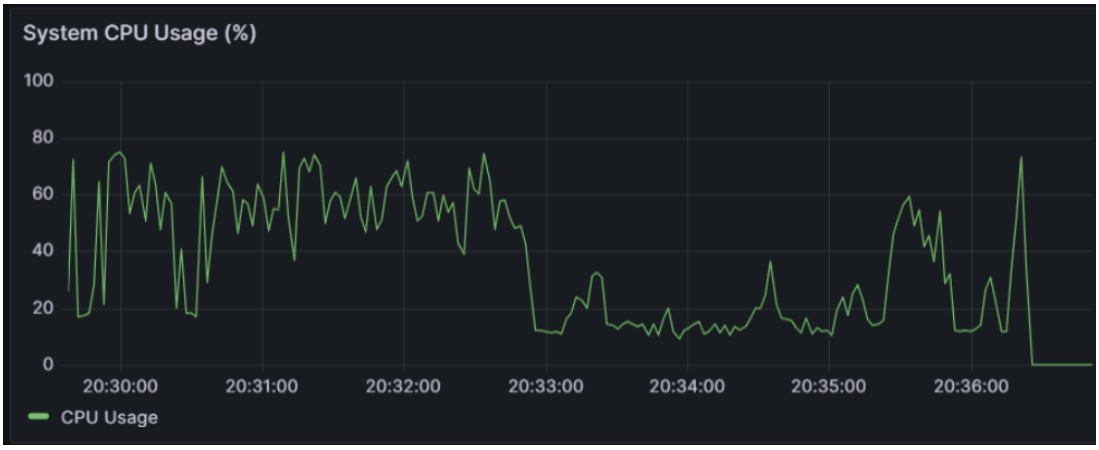


CPU usage

MD Core #1

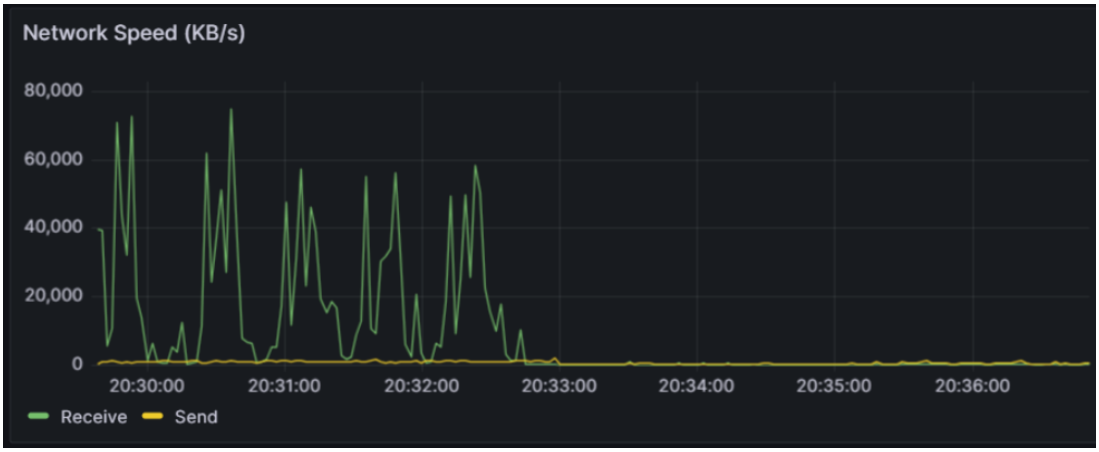


MD Core #2

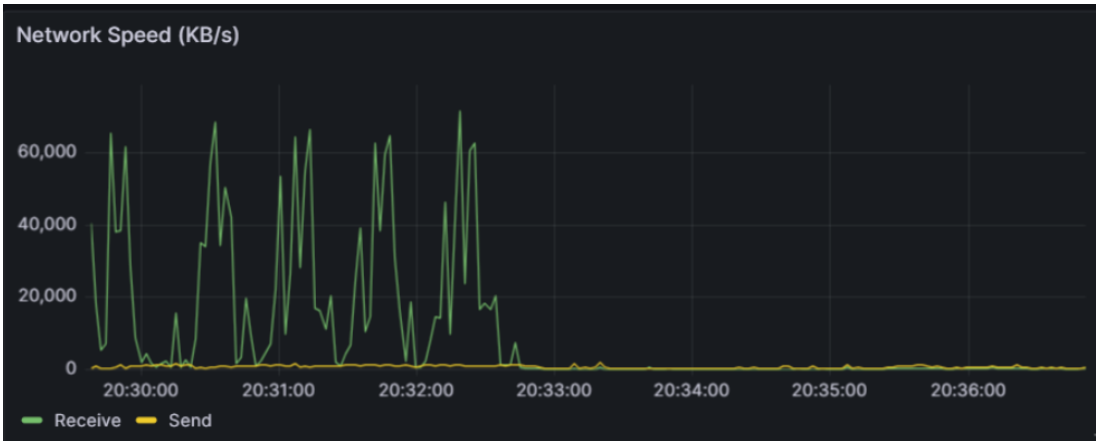


Network usage

MD Core #1



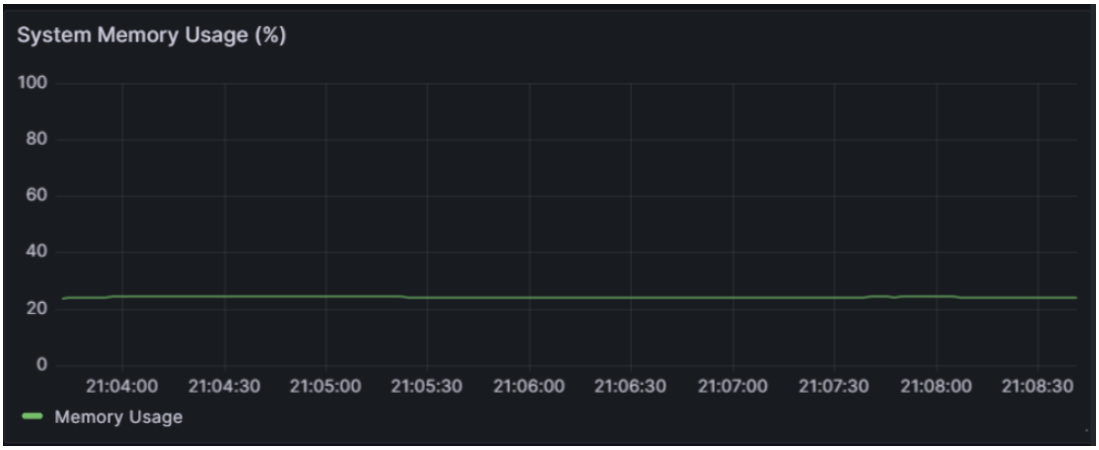
MD Core #2



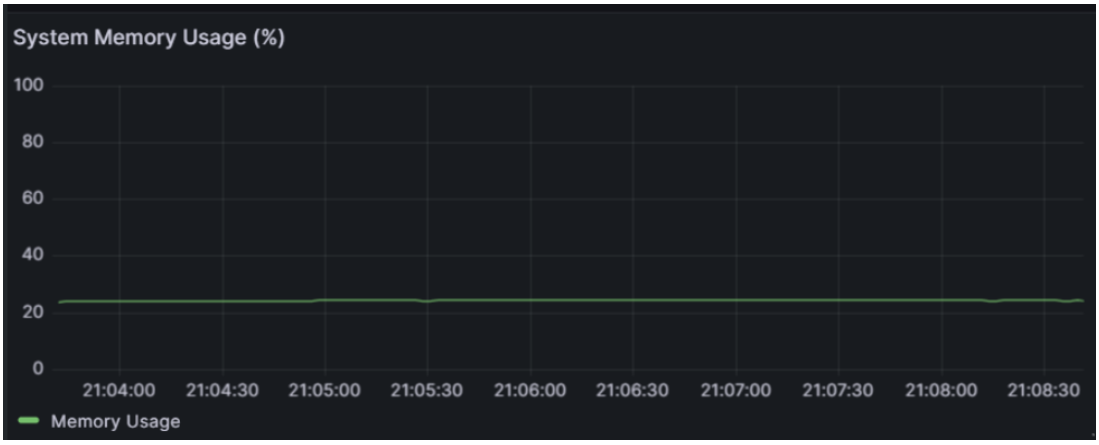
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

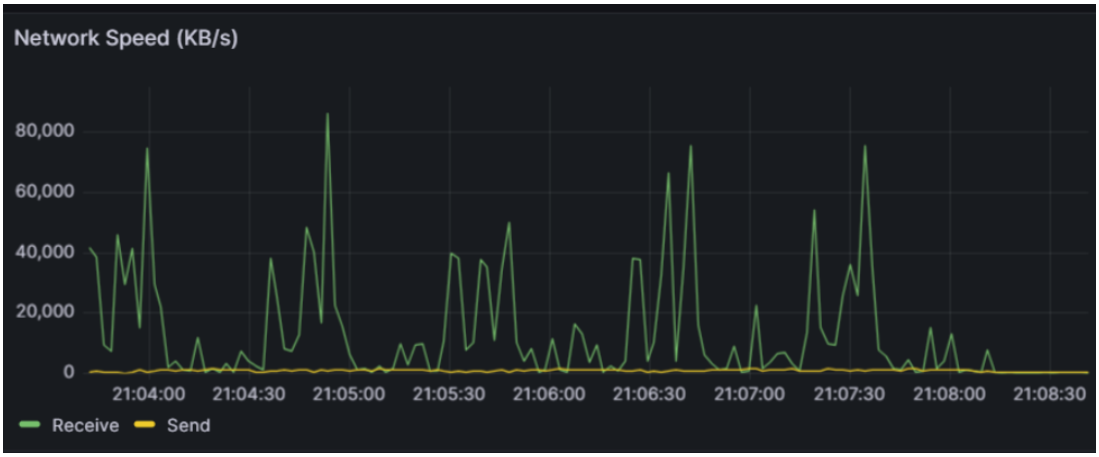


MD Core #2

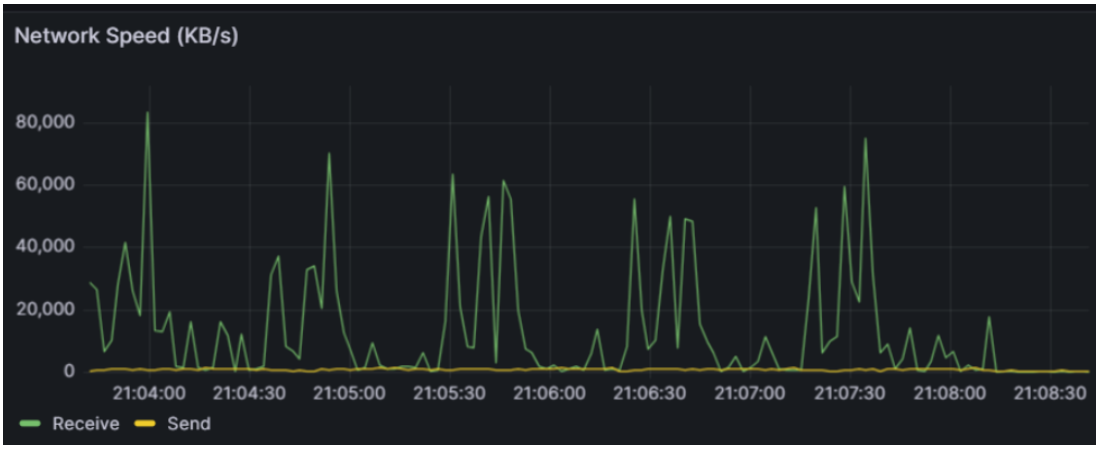


Network usage

MD Core #1



MD Core #2

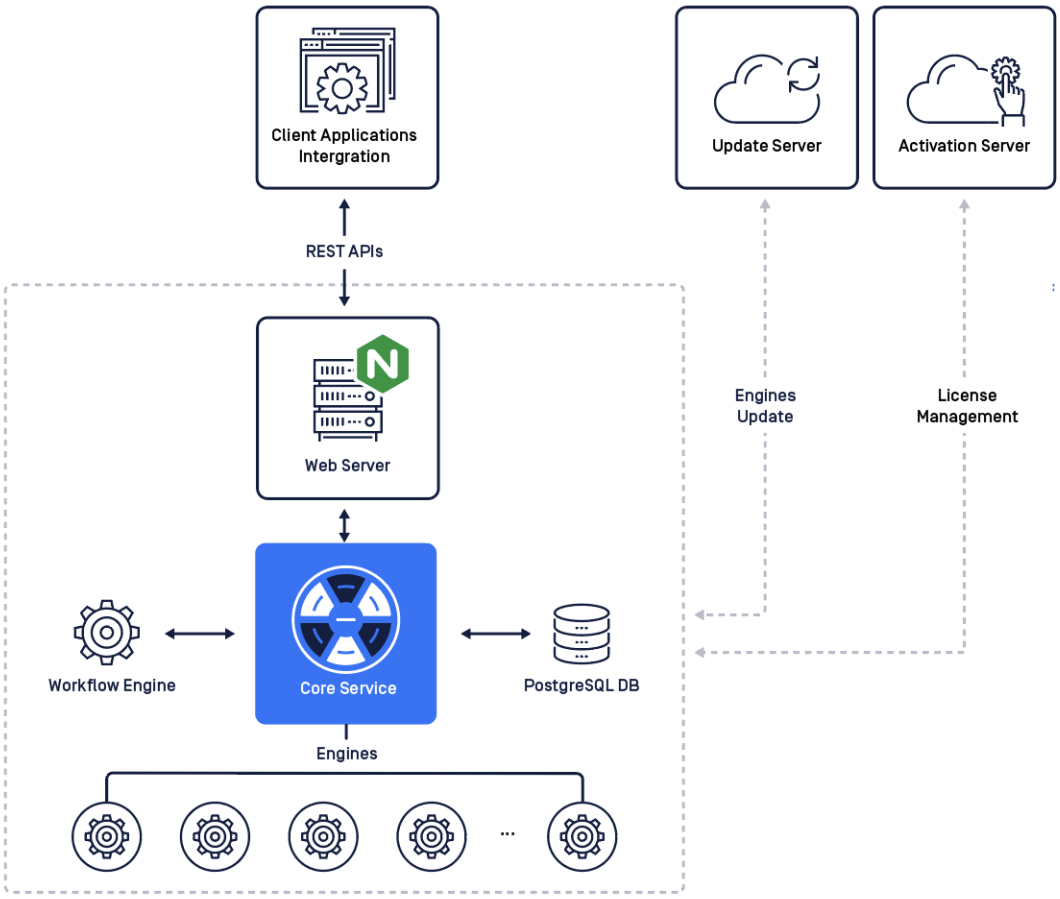

Network Speed (KB/s)
80,000
60,000
40,000
20,000
0
21:04:00
21:04:30
21:05:00
21:05:30
21:06:00
21:06:30
21:07:00
21:07:30
21:08:00
21:08:30
Receive
Send

MetaDefender Core Documentation

The users can consult this web page or, alternatively, they can download the manual in pdf format from the link below:

MetaDefender Core manual (SHA256:).

Recommended System Configuration



MetaDefender Core supports a wide range of deployment options, including air-gapped environments, providing the flexibility to choose the environment that best suits customers' requirements and infrastructure.

- Physical server.
- Virtualization platforms like VMware, Hyper-V and XenServer.
- Cloud Deployment.
- Container Deployment.

> **Note**
> Before installing MetaDefender Core, please refer to the **recommended minimum system configuration** listed below. The requirements are based on a standardized customer deployment and may require more or less resources to achieve the level of performance desired in your environment. Please note that the server specifications are built to allow a high volume daily processing. Please refer to Performance and Load Estimation for more details.

The recommendations below are for MetaDefender Core, API usage only. For any other use cases, please consult the user guide of the licensed products for accurate recommendations.

For certain use cases these might be adjusted and customized on specific needs and SLAs. We highly recommend to engage our ProServ team to assist in fine tuning MetaDefender and get the maximum performance out of your systems.

Microsoft Windows Deployments

[Windows] Supported Operating Systems

Only support 64-bit platforms.

- Windows 10, Windows 11 23H2, 11 IoT (22H2) only for End-User systems (e.g. Kiosk deployments)
  - Based on Microsoft License Terms, you're not allowed to use Windows Desktop as a server, therefore, MetaDefender Core can't be used on these systems for server use cases.
- Windows Server 2016, 2019, 2022

> **Disclaimer**
> End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

> **Plan for legacy OS support**
> OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:
>
> - Ubuntu 20.04 after **April 2025**.
> - Amazon Linux 2 after **June 2025**.
> - Windows 10 after **October 2025**.
>
> If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

[Windows] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 8	8	8 GB	16 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 12	16	16 GB	24 GB + 50 GB + 100 GB
MetaDefender Core 16	16	16 GB	32 GB + 50 GB + 100 GB
MetaDefender Core 20	32	16 GB	40 GB + 50 GB + 100 GB
MetaDefender Core MAX	32	32 GB	120 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core is installed.
- For a better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.
- SBOM processing requires more disk space, please allow at least 15GB free disk for the engine to work properly.

[Windows] Third Party Dependencies

- .NET framework 4.5 or above
- Microsoft Visual C++ Redistributable for Visual Studio 2015-2022 (version 14.38 or higher); x64 version.
- PostgreSQL: Only applicable when using a pre-installed PostgreSQL server running remotely.
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Some engines also have dependencies as described below:

Engine Name	Dependency
Deep CDR	Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 5.8.0 or above)
ESET	MetaDefender Core v4 temporary directory should have more than 200MB free disk space
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
Proactive DLP	• Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 2.0 or above) • .NET 6 dependencies • CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
Scrutiny	Microsoft Visual C++ 2015 - 2019 Redistributable x86 and x64
Systweak	.NET runtime 6.0, 7.0 or 8.0
Bkav Pro	Microsoft Visual C++ 2015 - 2022 Redistributable x64

[Windows] Installation Details

MetaDefender Core on Windows uses "C:\Program Files\OPSWAT" folder for storing resources or the installation directory.

MetaDefender will use its resources folder to store temp files as part of the analysis. **It's recommended to exclude this folder from real-time protection monitoring**. Knowledge base:

- How to set up exclusions on anti-malware software to prevent disruption
- Mitigating scan failure triggered by real-time protection service
- Can local AVs interrupt ongoing scans?

Linux Deployments

[Linux] Supported Operating Systems

Only support 64-bit platforms.

- Red Hat Enterprise Linux 8, 9
- Rocky Linux 9
- Debian 11, 12
- Ubuntu 20.04, 22.04
- Amazon Linux 2 is supported via Amazon Machine Image (AMI)

Plan for legacy OS support

OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:

- Ubuntu 20.04 after **April 2025**.
- Amazon Linux 2 after **June 2025**.
- Windows 10 after **October 2025**.

If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

For Hardended Linux OS

Some AV engines could be failed to initialize and run on MetaDefender Core (e.g. ESET, Kaspersky) due to execution permission required by them on `/var/tmp/ometascan` folder.

Please follow steps at Why have the ESET and Kaspersky scan engines failed to initialize on the hardened Linux OS? for remediation details.

Disclaimer

End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

[Linux] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 5	4	4 GB	10 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 10	8	8 GB	20 GB + 50 GB + 100 GB
MetaDefender Core MAX	16	16 GB	40 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core resides on.
- For better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.

[Linux] Third Party Dependencies

Dependencies list:

- openssl
- grep
- lib32stdc++6 (>= 4.5)
- libc6-i386 (>= 2.10)
- procps
- zlib1g
- libcurl3 (>= 7.19.7)
- libcurl4
- postgresql:
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Not all the above dependencies will need to be installed, it is dependent on different Unix distro and version

Some engines also have dependencies as described below:

Engine name	Dependency
Archive engine	• To extract password protected MS Office files, you need to install .NET 6 dependencies, libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • Rocky Linux 9.x/Red Hat 9.x: libnsl
Deep CDR	• Rocky Linux 9.x: libgomp, ncurses-compat-libs, libnsl • CentOS: libgomp, ncurses-compat-libs • Red Hat 8.x: libgomp, ncurses-compat-libs, libnsl • Red Hat 9.x: libgomp, ncurses-compat-libs, libnsl, compat-openssl11 • Ubuntu/Debian: libgomp1, libncurses5 • libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • .NET 6 dependencies
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
FileType engine	Rocky Linux 9.x/Red Hat 9.x: libnsl
Proactive DLP	libgdiplus, mscorefonts .NET 6 dependencies CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
RocketCyber	libgomp

Note: Some dependencies may need to be installed from external repositories, not from the OS default repositories.

[Linux] Installation Details

MetaDefender Core default installation path is using /var folder for storing resources:

- `/var/lib/ometascan` : installation folder with all its resources (database, DLP processed / CDR sanitized / quarantined file storage, engine package and definition updates).
- `/var/log/ometascan` : application logs.
- `/var/run/ometascan` : runtime folder for Core related processes.
- `/var/lib/ometascan` : all successful deployed engines data files.
- `/usr/lib/ometascan` : all shared libraries required by MetaDefender Core, bundled PostgreSQL server related binaries and libraries.
- `/var/tmp/ometascan/resources` : all temporary processing files for MetaDefender Core to process.

Storage Usage Information

Based on the configuration, MetaDefender Core will need additional disk space to store analysis/result data, quarantined files and temporary files:

	Data Storage
Analysis Reports (Processing History)	Approximately 4.5GB is required for every 1M analysis reports stored. • 100 GB is recommended as a baseline. Database storage requirements are affected by Data Retention settings, engine settings and scanning load.
Quarantined Files	Depends on the customers' dataset and configuration
Sanitized Files	Depends on the customers' dataset and configuration
Temporary processing files	Depends on the customers' dataset and configuration • A 1GB non-archive file requires 1GB free disk space at least. • A 1GB archive file requires (1 * 2)GB free disk space at least. • 50GB is a baseline and should be adjusted based on observation of the solution within the environment, with consideration to peak times and overall file makeup (e.g. complexity, size, archives).
Note	Options are available to control how large and/or complex files are handled to limit the amount of system resources required. See Workflow Configuration Template for more information.

Custom Engines

The recommendations above are specific for MetaDefender pre-packaged bundles.

However for additional Custom Engines, please review the Knowledge base to review additional requirements (if any) for the selected engine.

Browser Requirements for the MetaDefender Core Management Console

One of the following browsers is suggested to view the MetaDefender Core Management Console:

- Microsoft Edge
- Chrome
- Firefox
- Safari

Chrome, Firefox, Safari and Microsoft Edge browsers are tested with the latest available version at the time of release.

Network Bandwidth

In online environment where MetaDefender Core is configured to download engine update packages directly from OPSWAT update server source, then using gigabit network speed (125 MB/sec) is highly recommended to avoid potential network related issue while downloading files, or at least 100 Mbps (12.5 MB/sec) network speed.

Allowlisting URLs

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts':

- https://activation.dl.opswat.com - for license online activation
- https://update.dl.opswat.com - for engines and databases download

Note: *IP address-based allowlisting on your firewall might fail after some time since OPSWAT uses CDN (AWS Cloudfront) to faster delivery updates over the world, and IP address of edge servers might change over time.*

Depending on your location, connections with OPSWAT cloud-based servers be forwarded to specific AWS Cloudfront's edge locations, each is assigned with its own unique IP address ranges. You might also need to allow correct AWS Cloudfront's corresponding IP address ranges by referring to https://ip-ranges.amazonaws.com/ip-ranges.json

Warning

Even the OPSWAT update servers host updates for all of the available engines we support, sometimes custom engines might try to connect to their own cloud for updates, but this can be disabled in firewall and they will be updated just from OPSWAT cloud servers instead.

Differences Between MetaDefender Core v4 and v5

We are very excited to inform you about the release of version 5.0.0 of our MetaDefender Core!

The new version is designed to enhance the product performance, user experience, as well as to provide more flexibility, security, and scalability for our customers.

At the same time, we are also committed to keeping backward compatibility for whom running MetaDefender Core v4, and avoid the need for updating your client integration to move forward with MetaDefender Core v5.

v4 REST API Integration Supported
All public REST API endpoints available on MetaDefender Core v4 will be continously supported on v5.

v4 Product Major Features Supported
All product major features available on MetaDefender Core 4.21.2 (the latest one on v4) will be continously supported on v5.

Seamless Product Upgrade
Upgrading from MetaDefender Core v4 to v5 will be supported and covered with data migration without additional manual steps from users.

Licensing information will be also migrated.

Below are some highlights of the first v5 release, MetaDefender Core 5.0.0:

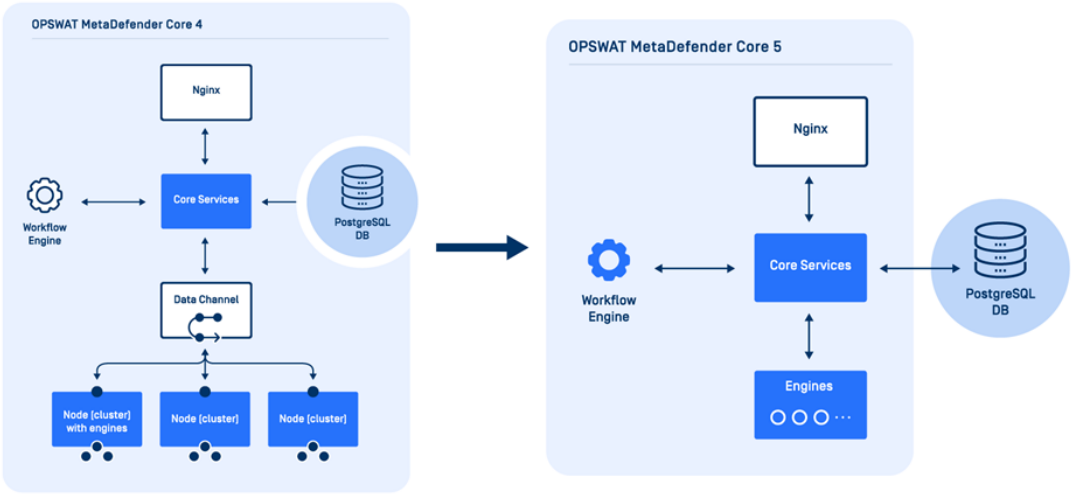
New product architecture

This version introduces a simpler product architecture that speeds up integration and automation, file processing flow, and immensely enhance the product performance and efficiency.

Earlier versions utilized MetaDefender Node concept, designed to support scaling of the scanning infrastructure by distributing scan requests among several different processing Nodes. Product licensing was based on number of Node instances. The benefit of having such a distributed infrastructure is that based on Node loads, Metadefender Core server can always choose the most appropriate Node to assign a new scan task to. However, having nodes in the v4 architecture caused more complexity and possible latency in the processing flow and entailed maintenance cost.

In this release, we retired multi-Node deployment and restructured MetaDefender Core's backend. This alteration significantly improves file processing speed and product efficiency, as well as eliminates extra cost for maintaining node services. Users now can simply install additional MetaDefender Core instances for scalability requirement,

as well as configure their own load balancer to coordinate tasks among multiple MetaDefender Core instances. Accordingly, product licensing is now based on number of the product instances.



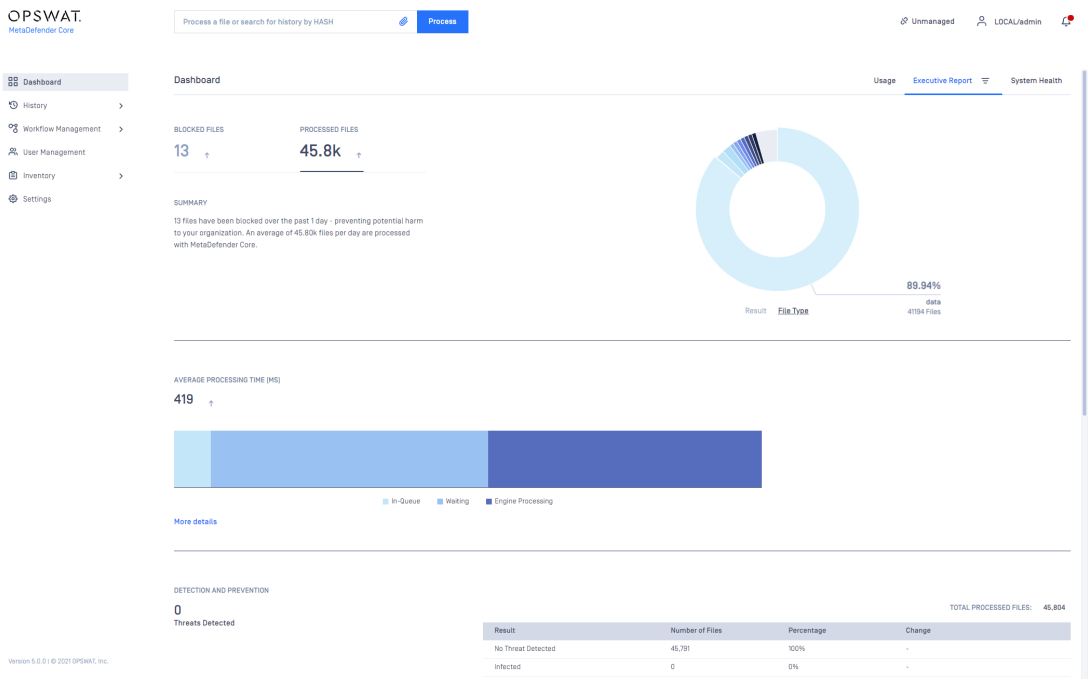
Please be informed that

- All existing administrative APIs related to "Node" on v4 will be kept as it is on v5 to ensure backward compatibility with existing client integrations.
- Applicable settings defined in Windows registry (Windows) or ometascan-node.conf config file (Linux) for Node will be migrated to Core. Details: MetaDefender Configuration

Brand-new product user interface

The second notable update in this version is a brand-new UI for the management console, which was designed with the focus on the quality and thoughtfulness of the user experience. The new administrative console UI provides an advanced set of Core features and observable processing status.

Comprehensive executive dashboard reporting offers dynamic, real-time and actionable threat insights. It also provides visibility of your security posture allowing you to paint a picture of security that aligns with the overall business. The report includes easy-to-read charts, graphs, and statistic explanation that help your security professionals reduce clutter, gather information quicker, spot threat activity early and act swiftly.



We're continuing to improve our product UI to provide users an excellent experience with the application and overall usability. Please expect more powerful UI features coming in next releases.

New compression archive engine for archive sanitization

In previous version, MetaDefender Core's archive engine implemented four functions including decryption, extraction, encryption and compression that with a high volume of files it takes time to process them all. For the purpose of enhancing archive handling performance, this release comes with a separated compression archive engine, which dedicatedly processes archive sanitization (compression and encryption). It is also created to facilitate maintenance and further feature development.

Archive sanitization feature now requires Deep CDR license

As to respect sanitization practice and ensure unknown threats prevention as designed, now starting on version 5, it's mandatory to have Deep CDR licensed for archive sanitization feature enablement on MetaDefender Core workflow configurations.

Archive handing related engines changed

The legacy "Archive" engine on MetaDefender Core v4 will be changed its name to "Archive Extraction" engine, even when you are still on MetaDefender Core v4.

A brand new "compression archive engine" is to dedicatedly handle archive sanitization in v5 (archive compression), that helps relevant archive handling performance improvement, easier for maintainance and further feature development.

Notes when you are still on MetaDefender Core v4 (not upgraded yet)

The legacy "Archive" engine (now name changed to "Archive Extraction" engine) on MetaDefender Core v4.x will still cover both extraction and compression archive tasks as usual.

Even though your MetaDefender Core v4.x might now have a new "Archive Compression" engine auto downloaded, but that engine cannot be deployed (marked as "permanently failed" in relevant engine status check API). This behavior is expected, and not causing any impact to MetaDefender Core processing flow, or its performance though.

- Just in case your client integration has a check for engine status before triggering scan request, then you might want to disable "Archive Compression" engine status on MetaDefender Core v4 using REST API. Detailed instruction: Archive Compression Engine Permanently Failed on MetaDefender Core v4
- When you later upgrade to MetaDefender Core v5 from v4, then you might need to manually re-enable "Archive Compression" engine to make it active and running on MetaDefender Core v5.

Notes for licensing

For offline deployment (MetaDefender Core is not connected to the Internet directly for online update), you will need to re-activate again on OPSWAT portal to retrieve latest engine list in .yml file including "Archive Compression" engine if you are currently licensed with Deep CDR.

Meanwhile, for online deployment, this manual procedure is not required since MetaDefender Core will cover that itself.

Import & export configurations

Not only workflow settings like in previous versions, but also all other general settings (update, security, data retention, server, hashes to skip) and most importantly, user settings, will be able to export and import on MetaDefender Core version 5.

Learn more at Import/Export configuration

Backward compatibility supported

The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

Windows application event logging disabled by default

Applicable only to Windows OS based MetaDefender Core.

Help avoid duplicated logging with separate file option, and improve overall processing performance. Of course, when being still valid to you, you are still supported to enable it back.

CSRF token header renamed

CSRF token originally created for security strengthening and being used by MetaDefender Core management console UI since version 4.21.0 via header `oms_csrf_token`

To get it compliant with some load balancers which do not accept underscore in header name by default, and save you from additional load balancer configuration step, we have now changed the header name to `oms-csrf-token`

This change does not impact to any REST API based client integration.

More comprehensive and accurate error messages on REST APIs

Impacting on following REST API endpoints:

- `PUT /admin/config/session`
- `PUT /admin/config/update`
- `PUT /admin/config/scan`
- `PUT /admin/config/skip`
- `PUT /admin/config/auditlog`
- `PUT /admin/config/webhook`
- `PUT /admin/config/rule`

Upload time and queue time logic updated in JSON scan response

Impacting on following REST API endpoints:

- `GET /file/{data_id}`
- `GET /hash/{md5/sha1/sha256}`
- `GET /archive/{data_id}`
- `POST /file/{async}`

Added new field:

- `receive_data_timestamp`: The timestamp when upload progress started (first byte received) (in milliseconds)
- `upload_time`: Total time elapsed for upload process (in milliseconds)

Modified existing field (new logic):

> **Backward compatibility supported**
> The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

- `upload_timestamp`: The timestamp when upload progress finished (all bytes received) (in milliseconds)
- `queue_time`: Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue time = start_time - upload_ timestamp) (in milliseconds)

Release version 5.0.0 also comes with variety of outstanding features and improvements.

For detailed information about our latest release please view our Release notes

Installation

Starting with MetaDefender Core 4.20.0, the centralized/shared database deployment model is supported (all MetaDefender Core instances share the same database), along with traditional standalone database mode (every MetaDefender Core instance has its own database to manage).

- Standalone mode
- Shared Database Mode

Installation

1. Download the package of your choice from My OPSWAT
2. Install the package on your computer via the Command Line or via the Wizard
3. Open a web browser and point to http://<server name or IP>:<port>
    - The default port is **8008**
    - In case of problem check Inaccessible Management Console page
4. Complete the required steps of the basic configuration wizard
5. You must activate this deployment to use its features

Installation notes

- If the MetaDefender Core package **dependencies are not installed** on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.
- During installation the databases might need to be upgraded. This could take noticeable time depending on database size (eg.: length of scan history).

Standalone mode

Wizard Installation

The Install Wizard is only for the Windows installer (.msi file).

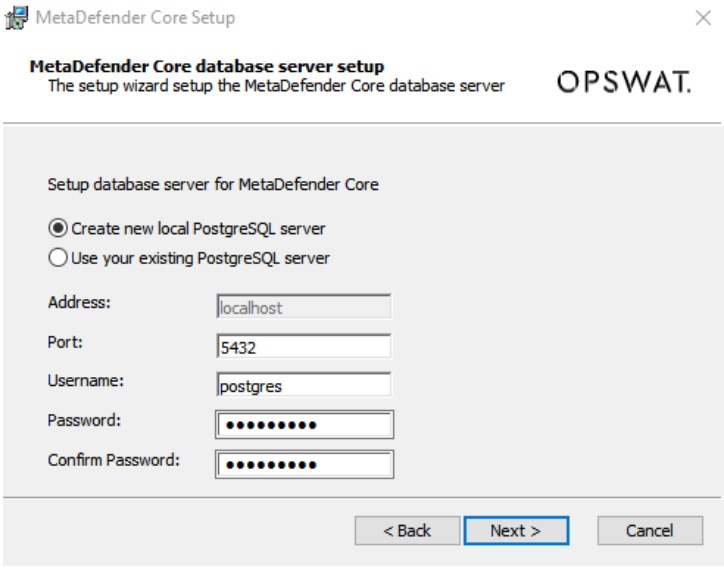
Since MetaDefender Core 4.19.0, we have switched to using PostgreSQL DBMS which needs a dedicated database server. MetaDefender Core supports two modes to setup PostgreSQL server:

Local (Bundle) PostgreSQL server

MetaDefender Core will install a new PostgreSQL server locally in the same box with the product.

> **PostgreSQL user credentials**
> Unicode characters are NOT supported for "Username" and "Password"
>
> "Password" does not accept # (hash) as starting character (e.g. #ABC)

> **The ignition file**
> If the ignition file exist in the designated location, then MetaDefender Core will use the details defined in the ignition file, rather than what is passed via the installation wizard.
>
> More info about the ignition file based installation: Command Line Installation

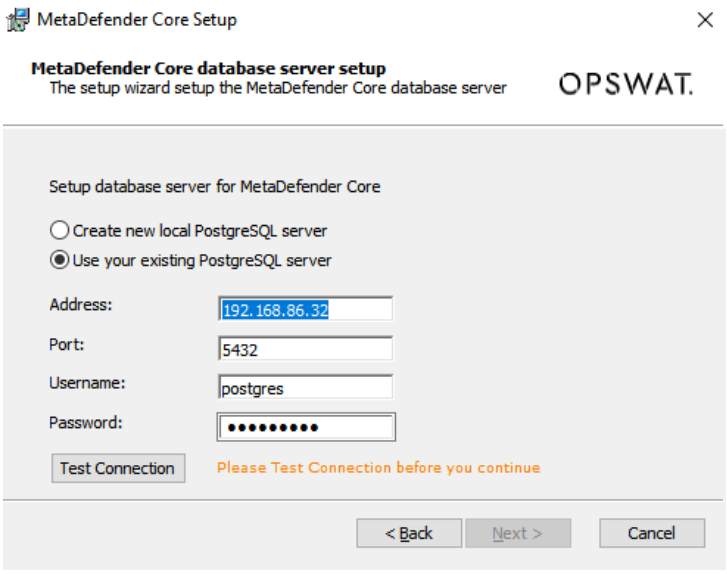


Remote (Existing) PostgreSQL server

MetaDefender Core will connect to remote pre-installed PostgreSQL running remotely.

> **PostgreSQL connection configuration**
> "Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

> **PostgreSQL user privilege requirements**
> MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.
>
> Details: PostgreSQL User Privilege Requirements



Command Line Installation

Ignition File

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Make sure to create the ignition file, if it doesn't exists before you go ahead and install MetaDefender Core via command line.

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

MetaDefender Core supports two modes to setup PostgreSQL server:

1) Local PostgreSQL server

- MetaDefender Core will install a new PostgreSQL server locally, next to the product.
    - A sample iginition file for PostgreSQL server information (***Only non-Unicode characters supported for "user" and "password"__***):

ometascan.conf bash

```
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
```

2) Remote PostgreSQL server

> **Info**
> **PostgreSQL version 12 (at least 12.3) or version 14 is required.**
>
> According to PostgreSQL notice, version 12 will stop receiving fixes on November 14, 2024, so we suggest that you use version 14.

MetaDefender Core will connect to a pre-installed PostgreSQL server running remotely and will setup its database.

Make sure to follow steps at Open Connection On PostgreSQL Server first, so that your remote existing PostgreSQL server to accept upcoming connection from MetaDefender Core.

A sample ignition file for PostgreSQL server information:

ometascan.conf bash

```
[dbserver]
type=remote
host=192.168.86.32
port=5432
user=existingserver_admin_user
password=existing_server_pass
```

After installing MetaDefender Core successfully, you may want to remove all credentials info created for [dbserver] section in ignition file for security reason.

> **Warning**
> If the MetaDefender Core package dependencies are not installed on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.

Debian package (.deb)

bash

```
sudo dpkg -i <filename> || sudo apt-get install -f
```

On Red Hat Enterprise 7.x Linux / CentOS package (.rpm)

bash

```
sudo yum install <filename>
```

For systems which enabled GPD check flag:

bash

```
sudo yum install --nogpgcheck <filename>
```

Red Hat Enterprise Linux 8.x package (.rpm)

Install the latest PostgreSQL repository RPM:

> **Note**
> Due to the shorter support cycle on Red Hat Enterprise Linux, all supported versions of PostgreSQL are not available on this platform. The PostgreSQL Yum Repository will integrate with the normal systems and patch management, and provide automatic updates for all supported versions of PostgreSQL throughout the supported lifetime of PostgreSQL. Reference: https://www.postgresql.org/download/linux/redhat/

bash

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

In case installing a remote PostgreSQL on either the same MetaDefender Core server or another server, you need to install the following additional package. PostgreSQL version 12.3 or newer is recommended.

bash

```
sudo dnf install -y postgresql12-contrib
```

Install MetaDefender Core:

bash

```
sudo dnf install -y <filename>
```

Windows package (.msi)

On Windows systems it is possible to install the product by running the corresponding .msi file.

From command line interface it is also possible to install the product by executing

bash

```
msiexec /qn /i <msi file name> <option key>=<option value>
```

where the possible keys and their default values are the following:

Key	Default Value	Description
INSTALLFOLDER	\Program Files\OPSWAT\MetaDefender Core	Customize installation folder for product Example: `INSTALLFOLDER="D:\Products"`
RESTADDRESS	*	REST interface binding IPv4 or IPv6 address ('*' means that service listens on all IPv4 and IPv6 interfaces)
RESTPORT	8008	REST interface binding port
REPORT_ENGINE_ISSUE	true	Enable reporting of engine issue count. (possible values: "true" or "false").

Note

Reporting of engine issue count

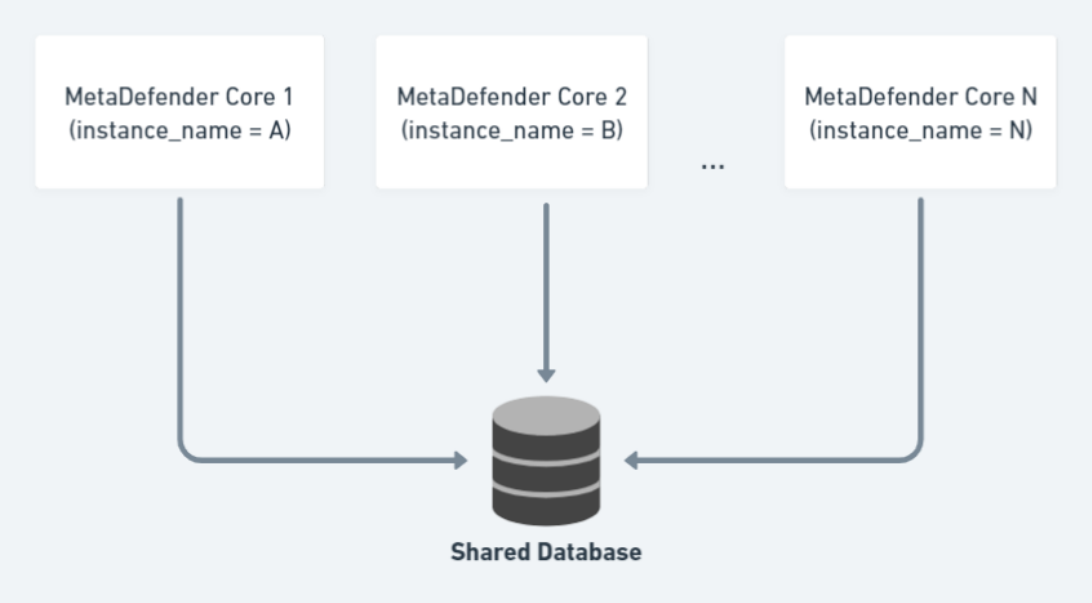
If reporting of engine issue count is enabled, MetaDefender Core server will send only the **number** of initialization errors and **number** of unexpected stops for the specific db/engine version. This information is sent over a HTTPS channel when the product downloads the latest package descriptors. This information is used for early detection of any specific 3rd party engine quality issues.

For details on using msiexec please consult Windows installer documentation.

Shared Database Mode

Starting with MetaDefender 4.20.0, you can use a centralized database deployment model, where all MetaDefender Core instances can share the same external database.

That helps auto-sync all product configurations (**except Single-Sign-On** that requires unique settings for each Core), processing history, live/on-going scan result query, audit, statistics data.



This article demonstrates how to install and setup MetaDefender Core 4.20.0 or newer with shared database mode.

Important notes & current limitations:

Currently not supported

- Does not support Central Management use-case with shared DB mode.
- Does not support to switch from shared database mode back to standalone database mode.
- Does not support to change instance_name of MetaDefender Core in shared database deployment model after installation.
- Does not support upgrading older MetaDefender Core from non-persistent mode to using the shared database mode.
- Does not support upgrading older MetaDefender Core from standalone mode to using the shared database mode.
- Does not support to upgrade multiple MetaDefender Core instances at once in shared database mode
    - Stop all Core instances' scanning service before performing the upgrade.
    - Upgrade each Core server one by one instead.

- For each Core, when the upgrade finished, then you can start using that Core for scanning without waiting for the other Core instances' upgrade to be done.

> **Disclaimer**
> MetaDefender Core does not support mode switching.

Requirements for shared database setup:

- The central remote PostgreSQL server (for shared database) could be utilized with a built-in (native) PostgreSQL database server installed with one of the MetaDefender Core instances in the network. Or using a dedicated existing PostgreSQL database server (see detailed requirements for PostgreSQL version at Recommended System Configuration).
- Before setting up, make sure to follow steps at Open Connection On PostgreSQL Server so that all MetaDefender Core instances could connect to the central remote PostgreSQL server.
- MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core, details: PostgreSQL User Privilege Requirements
- All MetaDefender Core instances must be installed on the same Operating System (Windows / Linux).
- The first MetaDefender Core instance connects to the shared database must be ready first (license activated, users/groups/roles created) before setting up other MetaDefender Core instances to connect and use that shared database.

Engine update:

- If engine updates is configured to be via Folder source, all MetaDefender Core instances must be able to access to the configured update folder, and the option DELETE FILES AFTER IMPORT will be unchecked and prevented for modification.

Authentication:

- Single-Sign-On (SSO) authentication settings (if using) cannot be shared among MetaDefender Core instances because SSO requires an unique login URL for every different Core.
- If an user is locked for login on a certain MetaDefender Core instance(e.g. exceeded the number of failed login times), that user is not prevented for logging in on any other instances.
- If an user is locked for sign-in on a certain MetaDefender Core instance (e.g. exceeded the number of failed login times), if the admin releases lockout on that user on that MetaDefender Core instance, that user is still not able to login on other MetaDefender Core instances.

Sanitized / DLP / Quarantined location changed after MetaDefender Core install:

Windows:

Shared storage path will be stored in 3 strings under key global in registry: quarantinepath, dlppath, sanitizepath as below

(Default)	REG_SZ	(value not set)
dbmode	REG_SZ	4
dlppath	REG_SZ	\\10.40.50.100\test
instance_name	REG_SZ	481rvj
quarantinepath	REG_SZ	\\10.40.50.100\test
restaddress	REG_SZ	*
restport	REG_SZ	8008
sanitizepath	REG_SZ	\\10.40.50.100\test

You can set either 1, 2 or 3 different paths for the 3 settings above (means that quarantined, DLP and sanitized files will be stored in same or different storages corresponding). In case shared storage is not set in installation, values of 3 settings above will be empty.

Restart MetaDefender Core service after changed.

Linux:

Open file `/etc/ometascan/ometascan.conf` and set shared storage path for 3 keys quarantinepath, dlppath, sanitizepath under section [global]

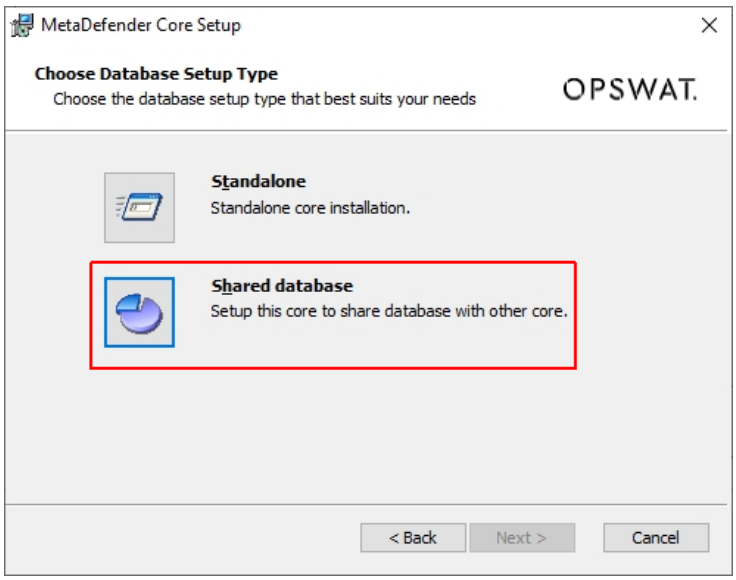
Restart ometascan service after changed.

Wizard Installation (Windows Only)

> **Info**
> Only applicable to MetaDefender Core 4.20.0 or newer using wizard on Windows OS.

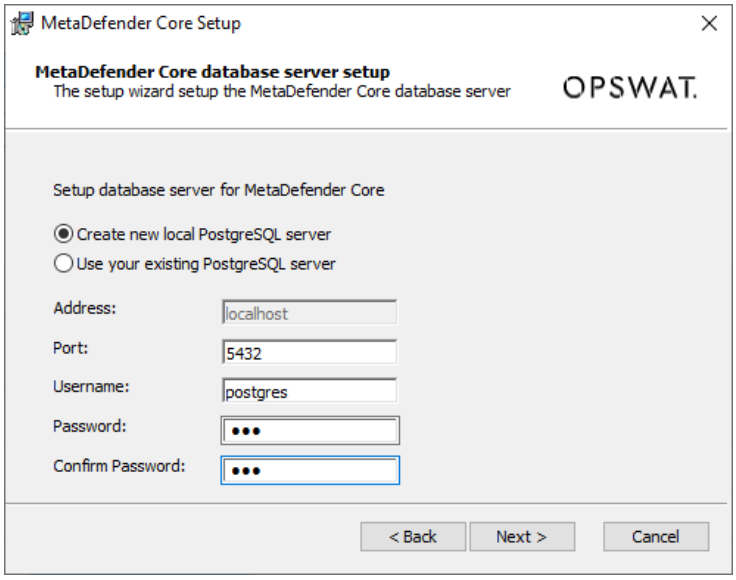
Clean install scenario

1. **Choose Shared database option:**

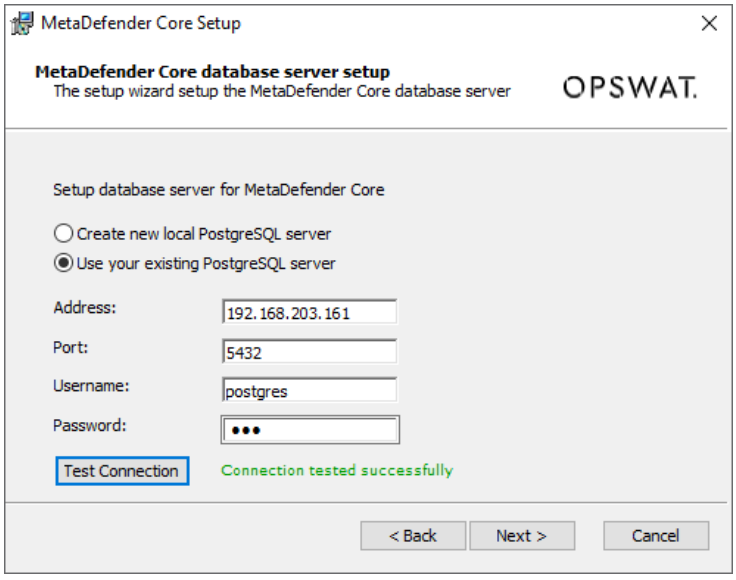


2. **Setup database server for MetaDefender Core:**

You can choose to create a native / built-in PostgreSQL server and setup the shared database on (then other MetaDefender Core instances will want to connect to that PostgreSQL server afterwards)



Or choose to connect to a pre-setup existing PostgreSQL server:



PostgreSQL connection configuration

"Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

PostgreSQL user privilege requirements

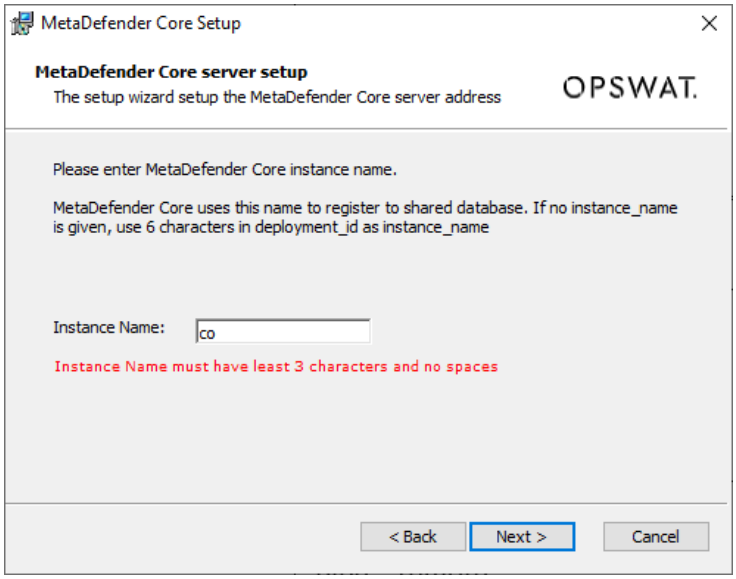
MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.

Details: PostgreSQL User Privilege Requirements

3. **Create an unique instance name for that MetaDefender Core (different from other MetaDefender Core instances):**

Instance name must contain at least 3 characters and no white space.

If you choose to leave it blank, which means it will use the default instance name with 6 characters extracted from that MetaDefender Core deployment ID.

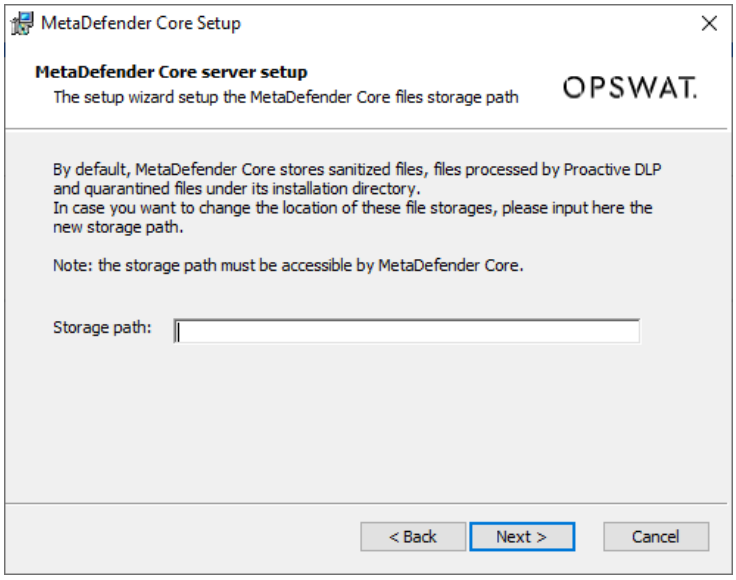


4. **Input the file storage path (sanitized, DLP, quarantined files)**

In the shared database model, all MetaDefender Core instances could use the same folder path to store all their own sanitized, DLP, quarantined files after processed.

The default location is: ***<MetaDefender Core installation folder>\data\ {sanitized|dlp|quarantine}***

If you wish to use that default location, just leave the input box blank and press Next.



Otherwise, you can specify a different location of sanitized/dlp/quarantined files. It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core.

For example:

- C:\new_folder
- \192.168.1.1\new_folder

Command Line Installation

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Clean install scenario

1. Create ignition file if not existed

- Windows: **C:\OPSWAT\ometascan.conf**
- Linux: ***/etc/opswat/ometascan.conf***

An example of ignition file:

ometascan.conf bash

```
eula=true

[global]
dbmode=4
instance_name=MetaDefenderCoreA
storage_path=D:\path\

[user]
name=admin
password=admin
email=admin@local

[dbserver]
type=remote
host=192.168.1.1
port=5432
user=postgres
password=your_own_password
```

Field	Required	Mapping to wizard UI	Possible values
global/dbmode	optional default is 1 (standalone)		1 - Standalone 2 - Central Hub deployment 3 - Non-persistent **4 - Shared**
global/instance_name	optional default is 6 characters of deployment ID		Unicode and number At least 3 characters No whitespace
global/storage_path	optional default is empty		Path to storages (DLP, quarantined, sanitized files) It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core. For example: • C:\folder • \192.168.1.1\folder

2. Run installers using command line to install

Please refer to Command Line Installation for installation command.

Deployment automation support

Note

An Ignition file could be utilized for either / both of following use-cases:

1. Pre-define PostgreSQL server information for MetaDefender Core to connect and setup database: only required on Linux installation, or Windows with command line installation. Learn more details at:
    a. Standalone Mode > Command Line Installation (applicable on version 4.19.0 or newer).
    b. Shared Database Mode > Command Line Installation (only applicable on version 4.20.0 or newer).
2. Automating to accept EULA, create a default local admin user and import MetaDefender Core configurations upon installed: Those steps could be automated using ignition file described in this page.

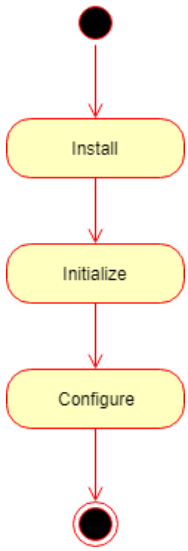
Ignition file location to create:

- **Windows**: `C:\OPSWAT\ometascan.conf`
- **Linux**: `/etc/opswat/ometascan.conf`

The product supports fully automated deployment. It means that it can be installed and configured with no human interaction.

The automated deployment can be split to three steps on a high level:

1. Installation
2. Initialization
3. Configuration



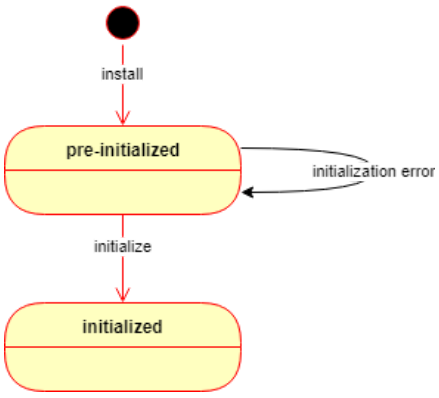
Installation

To automate the installation, install the product from the command line and provide the installation-time options as parameters to the installer. For further details see Command Line Installation

After the installation is complete, the product starts up and waits in a pre-initialized status. The product may be initialized in two ways:

1. Manually using the Wizard Setup or
2. Automatically using an *ignition file* (see below).

> **Warning**
> If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried after fixing the problem (e.g. placing the ignition file in its lookup location) and restarting the OPSWAT MetaDefender Core service.
>
> Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

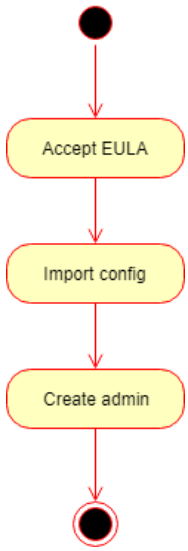


Initialization

Initialization is the process of bringing the product to an operable status.

Basically the initialization consists of the following steps:

1. Accept the End User License Agreement (EULA),
2. Import product configuration and
3. Create the first administrator user account.



Ignition file

The initialization process can be configured in a file called the *ignition file*.

> **Warning**
> In Linux, the ignition file and the folder /etc/opswat should have the permission 755

Ignition file location

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

The ignition file must be in conf format.

ometascan.conf bash

```
eula=true
[user]
name=admin
password=admin
email=admin@local
apikey=44c96cbb4e4e6c0a1d16bcb3c3da24def317
[config]
import=config_package.zip
import_password=123
[security]
session_cookie_samesite=Lax
[internal]
skip_startup_pages=true
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
private_username=my_internal_db_user
private_password=mypassword
[ocm]
ocm_url=https://my.opswat.com
reg_code=XXXXXXXX
```

Ignition file fields

The ignition file accepts the following fields:

Section	Key	Required	Accepted Values	Description
	eula	Optional (mandatory when having [user] section)	true false (default)	Whether to accept the End User License Agreement. This key must be set at the top of ignition file. This key must be set to "true" to accept the EULA. Any other value will cause the initialization to fail. When having [user] section defined, then this key must be set.
[user]		Optional		Initial administrator user account properties. The Administrator role is granted to the account.
	name	Optional	string	User name for the initial administrator user account.
	password	Optional	string	Password for the initial administrator user account. WARNING! Clear text password The password in this configuration file must be stored in its clear-text format and as so it may be visible for unauthorized parties.
	email	Optional	string	E-mail address for the initial administrator user account.
	apikey	Optional	string	API key for the initial administrator user account.
[config]		Optional		
	import	Optional	string	Path to a file in json/zip format that contains a previously exported configuration to be imported. Zip-format file contains a full configuration including user management settings, so it will overwrite [user] section.
	import_password	Optional	string	Password to decrypt .zip configuration package.
[security]		Optional		

Section	Key	Required	Accepted Values	Description
	session_cookie_samesite	Optional	Strict Lax (default) None	Declare if session cookie should be restricted to a first-party or same-site context.
[internal]		Optional		
	skip_startup_pages	Optional	true false (default)	Skip all welcome and other startup pages (e.g. upgrade result) when installing / upgrading MetaDefender, and going straight to the product index default page.
[dbserver]		Optional		
	type	Mandatory when having [dbserver] section	local remote	Where the PostgreSQL database server should physically locate. "remote" means PostgreSQL server and MetaDefender Core are not in the same machine.
	host	Mandatory when having [dbserver] section	string	IP address / domain name of the server where PostgreSQL server locates. "localhost" can be used when applicable.
	port	Mandatory when having [dbserver] section	number	Port of PostgreSQL server is listening for connections from clients (i.e. MetaDefender Core).
	user	Mandatory when having [dbserver] section	string	PostgreSQL server's user. SUPERUSER privilege is required for MetaDefender Core to setup its database and extensions for the first time. **Only non-Unicode characters supported.**

Section	Key	Required	Accepted Values	Description
	password	Mandatory when having [dbserver] section	string	PostgreSQL server's user credentials. **Only non-Unicode characters supported.**
	private_username	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
	private_password	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
[ocm]		Optional		
	ocm_url	Optional	string	URL to My OPSWAT server.
	reg_code	Optional	string	A registration code to enroll to My OPSWAT server.

Detailed initialization process

1) After the product has been started, it looks for the ignition file in the designated location.

Windows: C:\OPSWAT\ometascan.conf ** Linux: /etc/opswat/ometascan.conf

** 2) If an ignition file is found, then

2.1) It gets validated, and if it is valid, then based on the information found in the ignition file:

- The EULA is accepted,
- The configuration is imported,
- The administrator account is created.
- If any of the above steps fails, then the error is logged, and the initialization gets terminated.

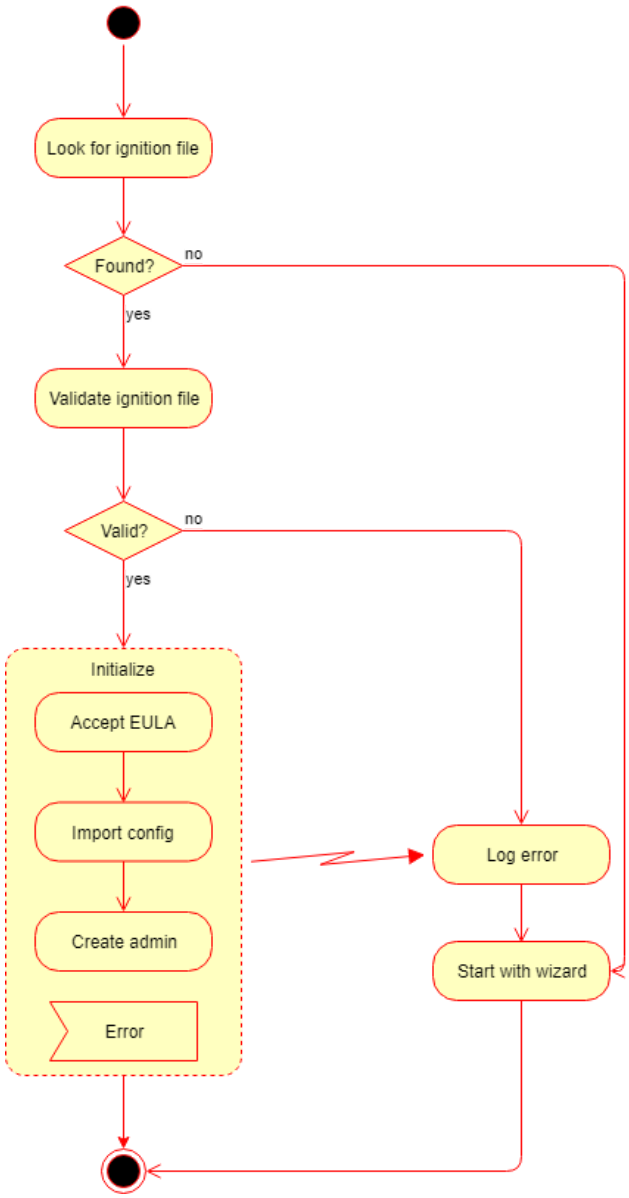
> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

2.2) If it is not valid, then the error is logged, and the initialization gets terminated.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

3) If there is no ignition file, then no initialization is performed.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.



Warning

If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried fixing the problem (e.g. placing the ignition file to its lookup location) and restarting the OPSWAT MetaDefender Core service.

Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

Configuration

After the initialization is complete, the product is ready with the default and the imported configuration.

This configuration can be later changed calling the configuration API functions. For further details about the API see Configuration related APIs API

CIS Level 1 Guidelines

Support Red Hat Enterprise Linux 9
For more details about Center for Internet Security (CIS) please refer to this document

Instruction steps

I. Install OpenSCAP

bash bash

```
yum install openscap-scanner scap-security-guide
```

II. Generate a result file and a HTML report using OpenSCAP scanner tool

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

III. Remediation of CIS Level 1 issues

Generate a remediation script based on the `ssg-rhel9-ds.xml` file:

bash

```
oscap xccdf generate fix --profile
xccdf_org.ssgproject.content_profile_cis_server_l1 --fix-type bash --output
remediations.sh /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

And execute remediation script.

bash

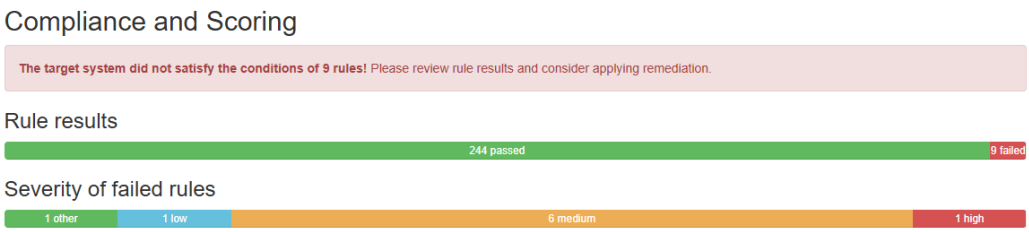
```
./remediations.sh
```

IV. Review the results after remediation

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

Example:



Notes

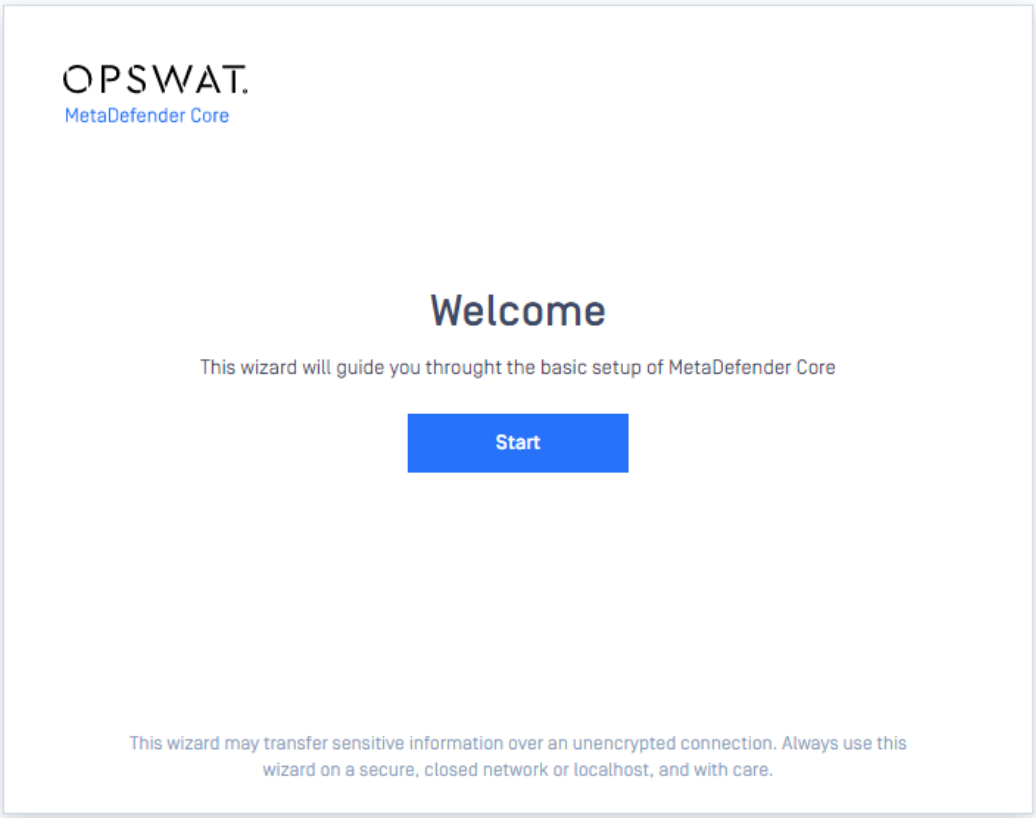
- CIS Level 1 requires `/tmp` folder to be mounted in a separate partition. Please ensure that that new partition have enough disk space for MetaDefender Core to run.
- CIS Level 1 requires that "Ensure No World-Writable Files Exist".
    - For now, when freshly installing MetaDefender Core, all its binary files meet the requirement.
    - When installing/updating engines, some engines might create additional files for its operation, and it might violate this requirement. In this case, you need to again execute the remediation script in the step III.

Wizard Setup

Introduction

Access the MetaDefender Core management console for the first time (i.e. http://localhost:8008), you are to complete a quick walkthrough for wizard setup in order to be able to use the product.

The administrative management console will be available only upon you have successfully finished this wizard stage.

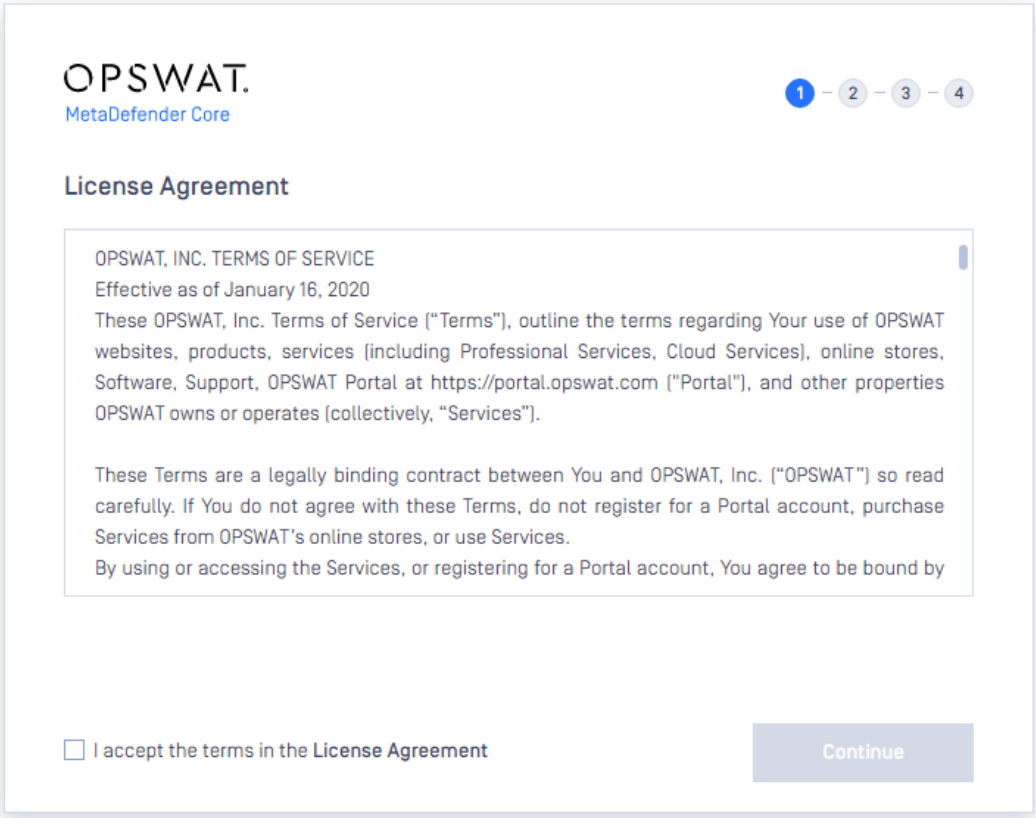


Warning

This wizard may transfer sensitive information over an unencrypted connection. Always use this wizard on a secure, closed network or localhost, and with care.

Basic configuration steps

End-User License Agreement



In the first page you can find the End-User License Agreement. You have to accept the terms before moving on.

Please read through the EULA (https://docs.opswat.com/policies/privacy-policy/portal-privacy-policy) carefully and if you agree with it, check `I accept the terms in the **License Agreement**` and click `Continue` to proceed.

Local Admin User Setup

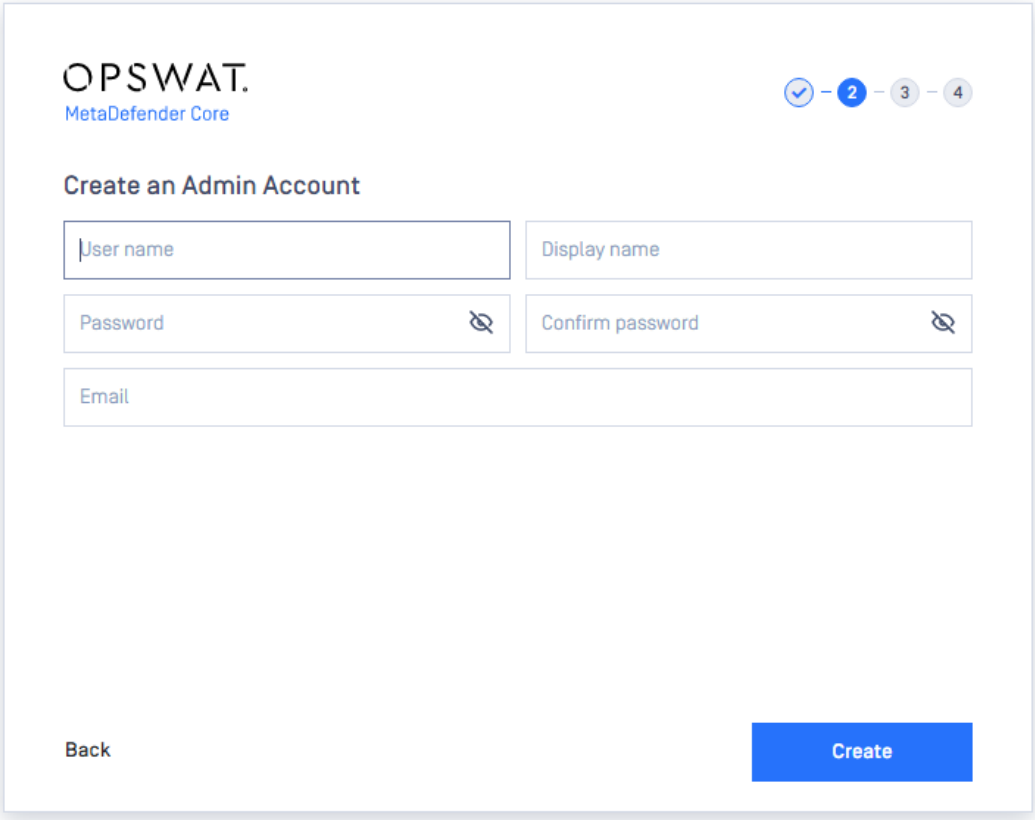
The next step is to set up a default local administrator account.

This account will be the first one being able to access the administrative management console, and to create accounts for other users. You have to fill out all fields in this page to be able to move forward. When you are done, click NEXT to continue.

> **Info - User Directory**
> The administrator account, that is created via the basic configuration wizard, is always added to the LOCAL user directory as a member.
>
> Learn more about user directory: User directories

The following information is required for the administrator account:

ACCOUNT NAME	The unique name of the account that is used at the time of login and in log messages for accountability.
ACCOUNT DISPLAY NAME	Name of the person bound to this account. This name (appended to the name of the account's user directory) is displayed in the top right corner of the administrative management console.
PASSWORD	Password of the user bound to this account that is used at the time of login. Passwords sent clear-text As long as TLS is not configured for the basic configuration wizard, passwords are sent clear-text over the network and may be disclosed to unauthorized parties. As a mitigation action: 1. Either use the wizard on *localhost* or on a direct network connection, or 2. Enabling HTTPS as soon as possible and change the password immediately if it has already been set.
EMAIL	Email address of the person bound to this account.



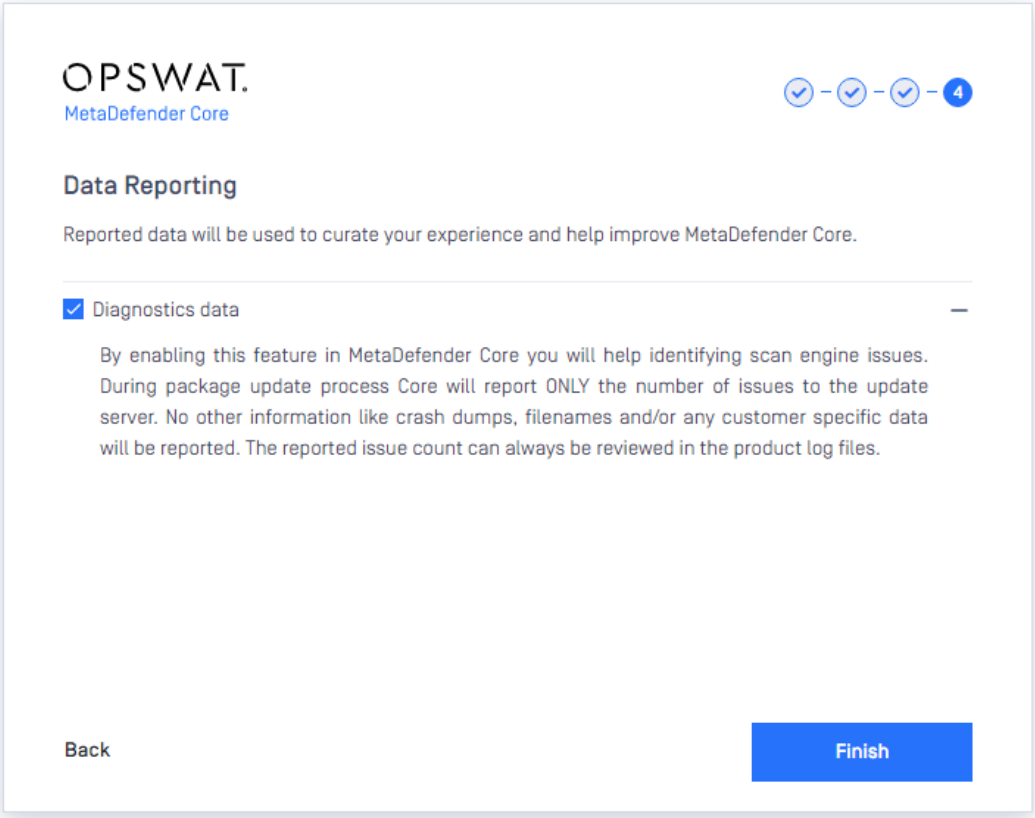
License activation

For license activation details see License Activation

Data Reporting

Enabling this option to allow MetaDefender Core collect its own engine issues and report back to OPSWAT for further analysis.

MetaDefender will not collect any other information.

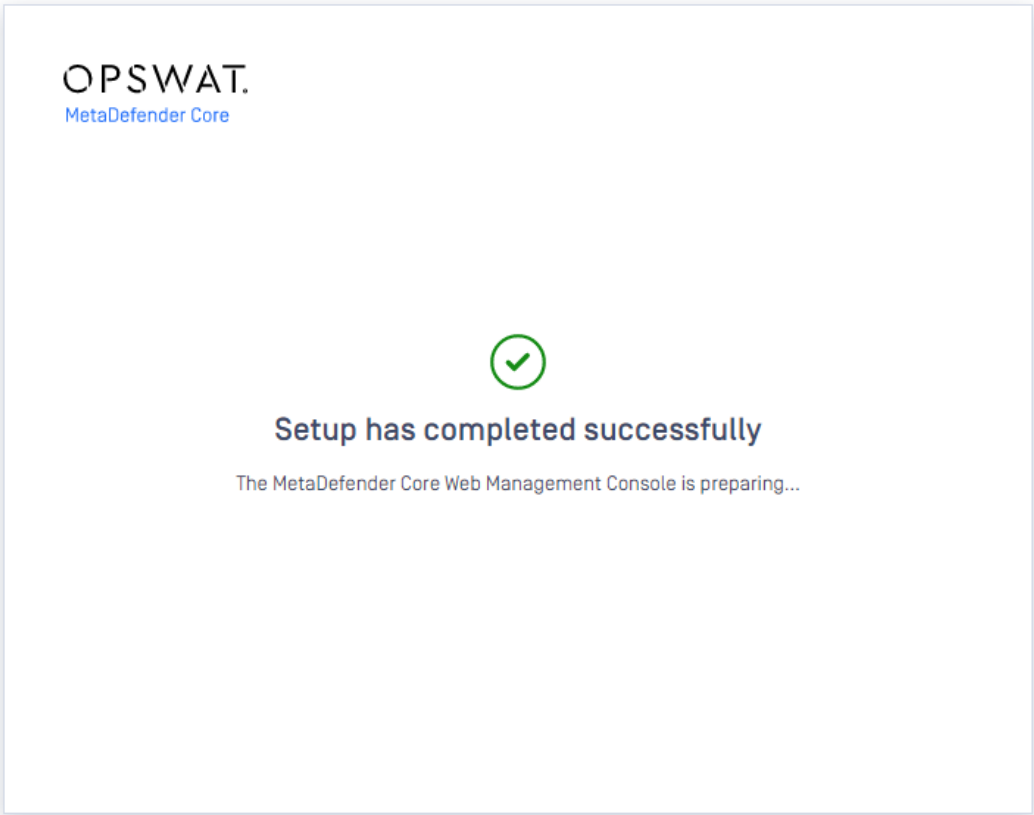


Wizard completion

After you have completed every steps you are ready to finish the wizard and start using the product. Click the `Fishish` button to complete the wizard.

> **Warning**
> The product's service will be restarted and the browser will be redirected to the administrative management console. This could take several seconds.

You can login to the Web Management Console with the administrator user that have just been created in the previous steps.



Setup HTTPS

Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network. Websites, like the Web Management Console, are able to use TLS to secure all communications between their servers and web browsers.

The TLS protocol aims primarily to provide confidentiality (privacy) and data integrity between two communicating computer applications.

Warning

No TLS for the wizard

By default, TLS is not enabled for the basic configuration wizard. As a consequence sessions between the wizard's backend and the browser may be insecure.

Info

Performing the same steps as for the Web Management Console, it is possible to set up TLS for the basic configuration wizard. Remember completing the TLS setup before launching the wizard.

For instructions to set up TLS see Enabling HTTPS

License Activation

##

To activate your installation go to the **Settings > License** menu in the Web Management Console. If you have no valid license, you will only see your installation's **deployment ID**. You will also see a warning in the Web Management Console header.

After successful activation, the product will start downloading the latest available scan engines and malware databases. You can follow the status of the scan engine installation on the **Inventory > Modules** page.

When all the licensed engines are installed, you can start using the MetaDefender Core to process files - File Processing.

Deployment ID

Each MetaDefender Core instance represented by an unique system fingerprint called `Deployment ID`

MetaDefender Core licensing is purely based on deployment ID activated. Therefore, everytime your MetaDefender Core is activated with a different deployment ID, then it will be captured as a different activation record accumulated into your total activation times allowed.

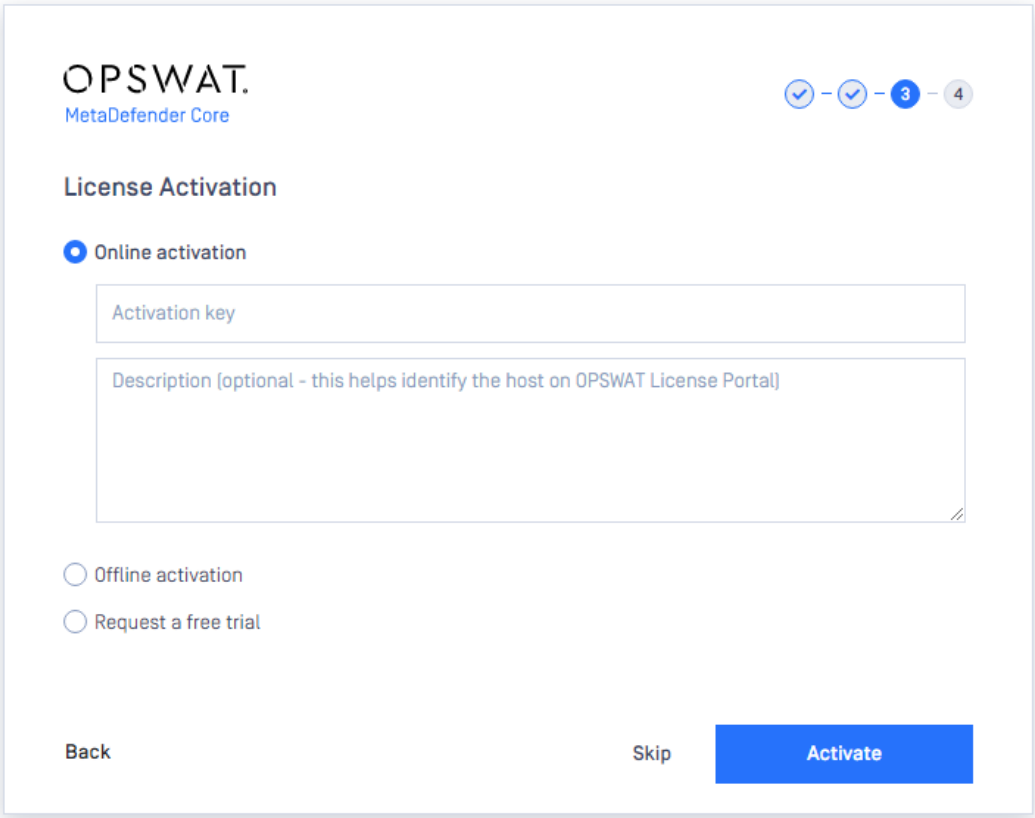
> **Notice**
> Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you alter the system where MetaDefender Core relies on (e.g. network card, OS hostname).
>
> Deployment ID is an UUID version 4 random based string. Learn more at: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you update the system where MetaDefender Core relies on.

Activation Modes

There are three different options for the product activation:

- Online activation: MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.
- Offline activation: Requiring you to upload a license file (.yml). That license file could be retrieved by activating your license via OPSWAT Portal. Follow the displayed instructions for details.
- Request trial key online: For evaluation purpose, you can receive an trial Activation key via email. Follow the displayed instructions for details.



Online activation

MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.

Required to input your Activation License key, and optional for description (but we recommend this for troubleshooting and auditing purpose if needed).

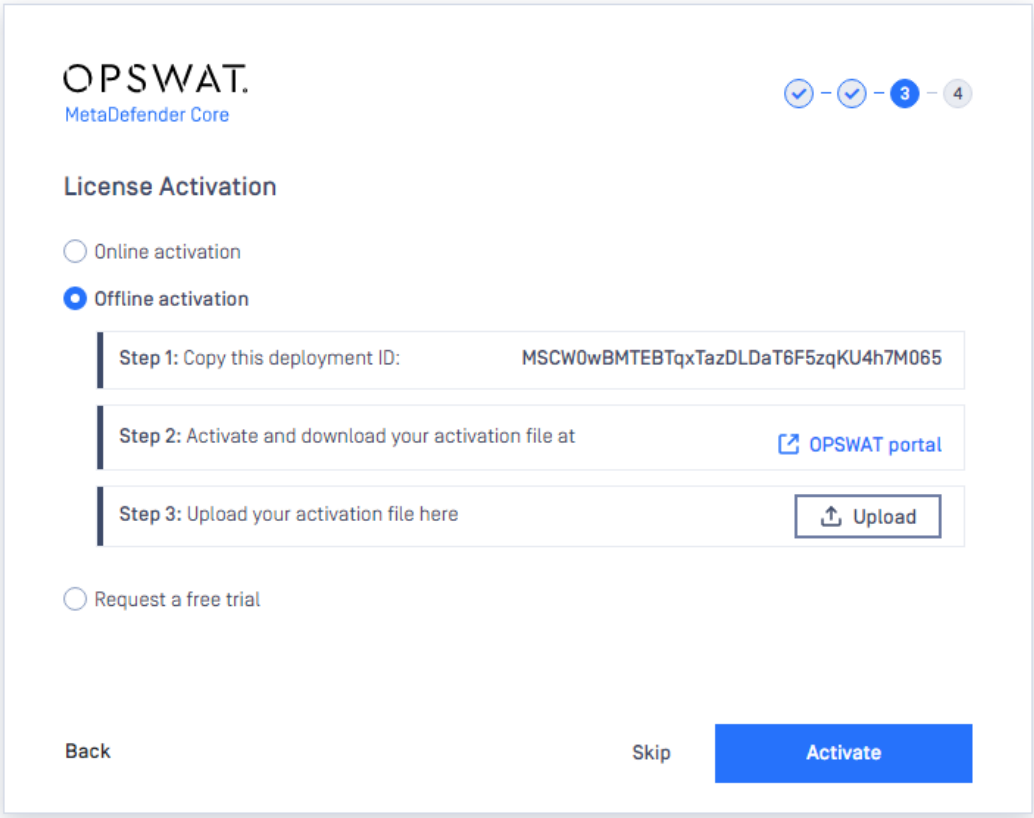
Warning

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts' for license activation:

- https://activation.dl.opswat.com Note: *IP address-based whitelisting on your firewall might fail after some time since OPSWAT uses CDN (Content Delivery Network) to faster delivery updates over the world, and IP address of edge servers might change over time.*

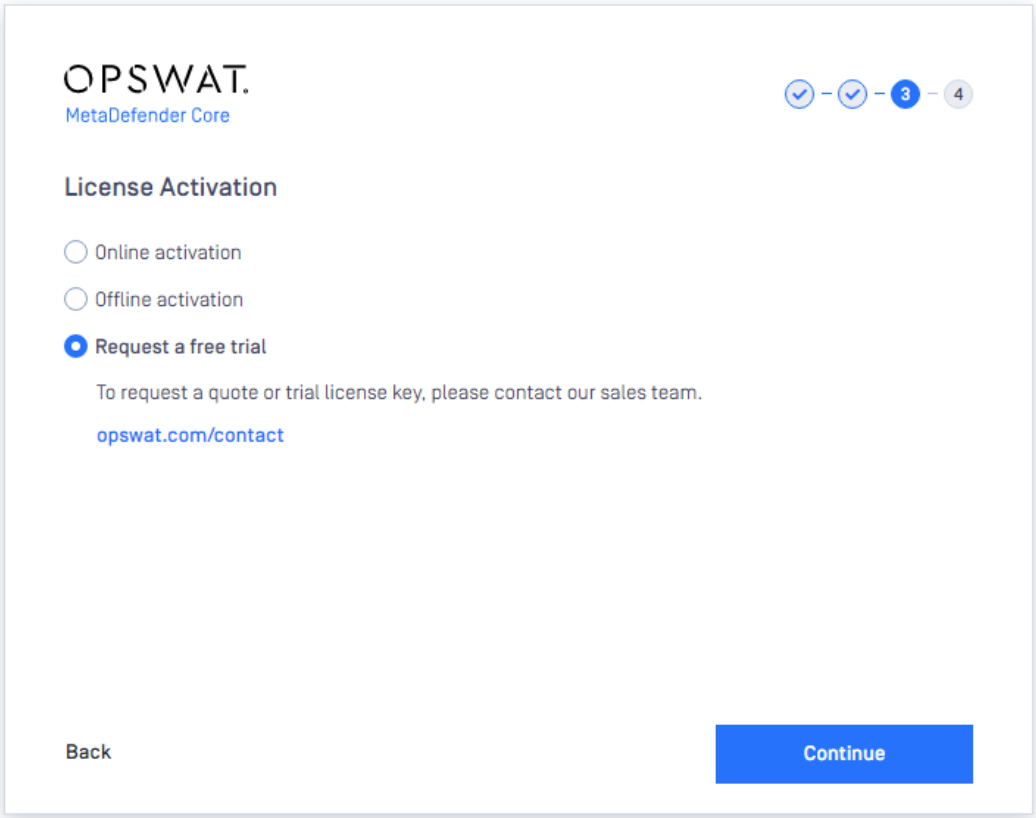
Offline activation

With no internet connection on the server the Metadefender v4 instance may be activated indirectly from a different machine, that has internet connection. The Deployment ID of the Metadefender v4 instance and the the Activation key received at the time of purchasing the product will be required. Follow the steps on the screen to activate the product offline.



Request trial key online

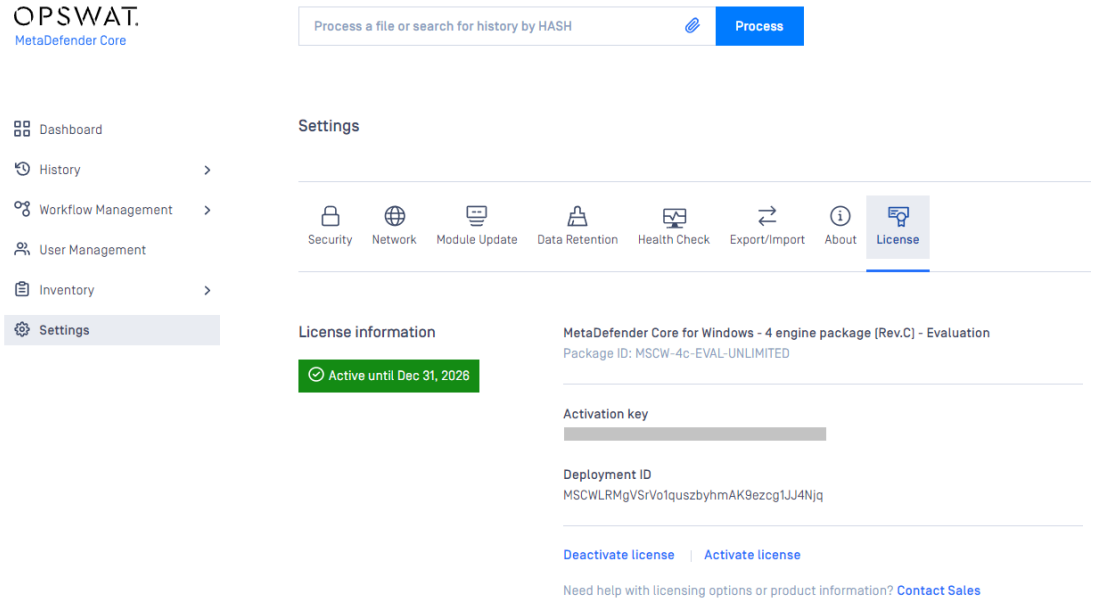
An evaluation license could be acquired by contacting our sales team.



Check Your License Details

For more license details and activating your MetaDefender Core, go to Settings > License menu tab on the Web Management Console:

- Product name: product name as on your order (e.g. MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation)
- Package ID: product identification as on your order (e.g. MSCW-4c-EVAL-UNLIMITED)
- Expiration: last day of license validity
- Activation key: key used for activating product
- Deployment ID: identification of this installation


OPSWAT.
MetaDefender Core
Process a file or search for history by HASH
Process
Dashboard
History
Workflow Management
User Management
Inventory
Settings
Settings
Security
Network
Module Update
Data Retention
Health Check
Export/Import
About
License
License information
Active until Dec 31, 2026
MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation
Package ID: MSCW-4c-EVAL-UNLIMITED
Activation key
Deployment ID
MSCWLRMgVSrVo1quszbyhmAK9ezcg1JJ4Njq
Deactivate license
Activate license
Need help with licensing options or product information? Contact Sales

Scripted license management

Using REST API calls there is an option for scripted activation and deactivation for MetaDefender Core servers with **no Internet connection or if your infrastructure is using automation to create/destroy MetaDefender Core instances**. In this chapter the steps of these two scenarios are described.

Requirements

- an installed MetaDefender Core instance without Internet connection
- another computer that has Internet connection and can run your activation scripts
- a manual or automated way to transfer data between the two computers

Activation steps

1. In order to activate MetaDefender Core, deployment ID and activation key are needed to generate the license file.
    a. Activation key should be purchased from OPSWAT.
    b. MetaDefender Core deployment ID can be queried by using REST API (for details see GET Get Current License Information API page)

> **Info**
> Save this deployment ID in your system, you might need this ID when the instance is unavailable at the time of deactivation.

2. Activate license and get the license file using the following URL, where <quantity> is 1.

activation call bash

```
curl -X GET https://activation.dl.opswat.com/activation?key=<activation-key>&deployment=<deployment unique-ID>&quantity=<quantity>
```

If the activation is successfully, a license file is downloaded. Save this file.

HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	license file	Activation was successfully.
200 Activation failed	error: '<user conform error message>'	Failed activation
200 Invalid parameter	error: 'Could not activate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not activate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
200 Invalid parameter	error: 'Could not activate your product because the quantity you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key, quantity or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Info
In case of any activation issue, contact OPSWAT support for help

3. Upload license file to MetaDefender Core: The license file should be uploaded to the MetaDefender Core v4 to activate the product. For details see page: POST Upload license key API

Deactivation steps

1. For deactivation, following infomation is required:
    a. Activation key
    b. MetaDefender Core deployment ID: can be queried by using REST API (for details see GET Get Current License Information API page)
2. Deactivate license using the following URL:

deactivation call bash

```
curl -X GET https://activation.dl.opswat.com/deactivation?key=<activation-
key>&deployment=<deployment unique-ID>
```

Response HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	result: ok	Successful deactivation
200 No active license found	error: 'Could not found any active license with the given parameters'	The license has not been activated yet or it has been deactivated already.
200 Invalid parameter	error: 'Could not deactivate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not deactivate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Important notes

Warning

Product activation is tied to several hardware, operating system and software parameters. In case of one or more major hardware or software parameter change the product might turn into deactivated status. Operating system updates, other software updates should not affecting the activation status.

If the product is online activated and has live Internet connection then in case of deployment ID change the product reactivates itself. If the product doesn't have Internet connection then the administrator is responsible to reactivate the product.

Because of the above administrator should consider licensing restrictions of the offline deployments if using any virtualization and/or containerization technologies.

File Processing

MetaDefender Core provides REST API interface for file processing business.

There are several ways to process files with MetaDefender Core:

- Analyze Files via Web Interface
- Process Files via REST API

Product Upgrading

> **Info**
> Simply run installer file, details could be found at Installation

> **Info**
> For command line mode, upgrading (e.g. on **Linux**), prior to running the installer file, please make sure to **create the ignition file** with PostgreSQL server information included. Details:
>
> - Standalone DB mode: Command Line Installation (Windows and Linux)
> - Shared DB mode: Command Line Installation

Recommendation for MetaDefender Core upgrade

Our goal is to help you execute the product upgrade continuously, efficiently, and correctly.

PostgreSQL service upgrade

- **Bundled PostgreSQL Service**: If you are using the bundled PostgreSQL server included in the installer, the PostgreSQL service will be automatically upgraded to the newer version (if available). No additional action is required for this process.
- **Remote PostgreSQL Service**: For remote PostgreSQL servers, customers are responsible for managing and upgrading the service independently. Upgrading MetaDefender Core does not update the remote PostgreSQL service.

Before the upgrade

Prepare a proper disaster recovery plan (DRP) including some common practices:

- Backup, or take a snapshot of your current working MetaDefender Core server.
  - This is to help quickly rollback for any upgrade failure might cause later for any reason.
- Optionally backup your PostgreSQL database, especially when using remote shared database mode.
  - Reference: https://www.postgresql.org/docs/current/backup.html
  - Please do it after stopping all linked MetaDefender Core instances to avoid potential data collision.
- Lastly, if backing up entire server is not feasible, then at least export MetaDefender Core configuration package Import/Export configuration just in case you'll need to start off a new server with current MetaDefender Core configuration to keep the scanning service running without any impact.

On the PostgreSQL server, double check server resource to ensure sufficient RAM memory and disk space to be preserved for the upgrade process.

- 8GB RAM available at least during the upgrade.

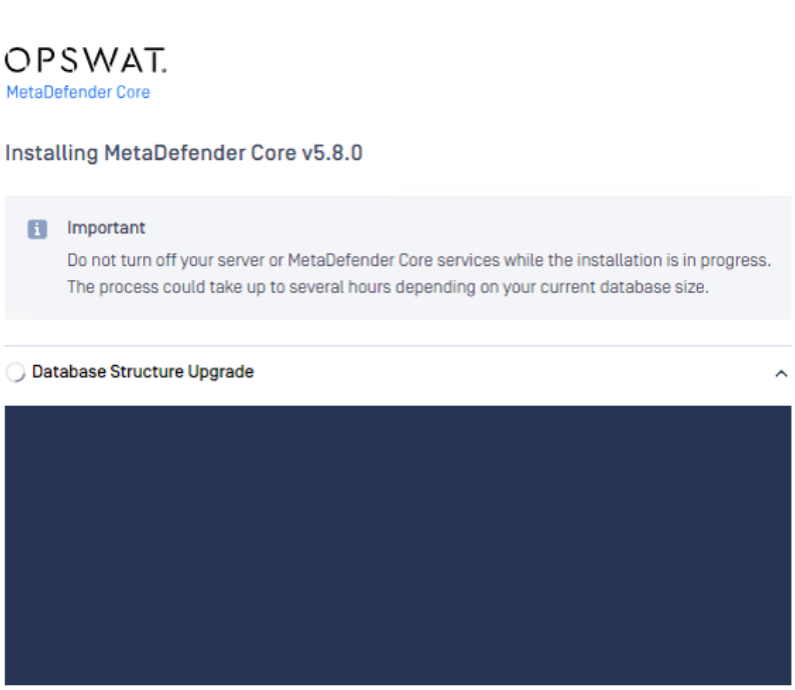
- Disk availability is greater than MetaDefender Core database size.

While doing the upgrade

Applicable for shared database deployment:

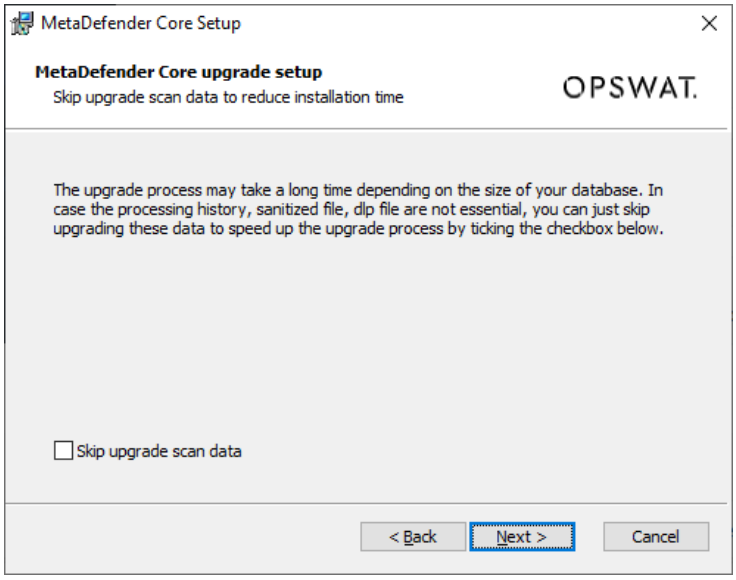
- Stop all MetaDefender Core instances' scanning service before performing the upgrade.
- Upgrade one (1) MetaDefender Core instance at a time.
- Proceed to upgrade MetaDefender Core instance where the bundle local PostgreSQL server is used for shared database setup.
- Otherwise is using remote PostgreSQL then the upgrade order does not matter.

For large size database, mostly for scan history, expecting long time to be spent on the upgrade process.



The product supports an option to skip data migration for scan history to expedite the process drastically and mitigate any failure risk. You can manually backup all data from your PostgreSQL server for offline usage if any.

Check "Skip upgrade scan data" to enable this feature while doing the upgrade.



or add following key in the ignition file:

none

```
[internal]
skip_upgrade_scan_data = 0
```

Troubleshooting

If the upgrade process ever runs into an error:

- Double check your network and services, applicable to remote PostgreSQL server.
- Optionally configure MetaDefender Core to skip migrating scan history.
  - Windows
    - Open Windows registry, and navigate to HKLM\Software\OPSWAT\Metascan
    - Add a new key `internal/skip_upgrade_scan_data` with value = 1
  - Linux
    - Open MetaDefender Core configuration file `/etc/ometascan/ometascan.conf`
    - Add new key into the file
      - [internal]
      - skip_upgrade_scan_data=1
  - Note: This key will be auto removed upon the upgrade finishes.
  - Then restart MetaDefender Core service to retry the upgrade.
    - Might retry this a few times until the upgrade is done.

Just in case the upgrade process still fails, please contact OPSWAT support with the support package - How to Create Support Package? This will help us analyze the root cause and figure out an optimal solution for your system failure.

Using external Load Balancer

Application Load Balancing (Layer 7)

This page provides information about recommended ways to use sessions with a Layer 7 load-balancer.

Most load-balancers have the ability to provide cookies for pinning subsequent traffic from a client to the appropriate server. This method is called sticky session, session persistence or session affinity. Using cookies efficiently requires the client to know when a cookie should be sent, should not be sent or should be deleted.

Sticky session load balancing

Each Metadefender Core instance has its own databases and application sessions that cannot be seen by other instances. Therefore, in order to get the related data/response to our queries we should ask the appropriate Metadefender Core v5 server. On the other hand, to keep the advantages of the used load balancing method, cookies should not be sent if it is not necessary.

Single file scanning

Step	Stage	Task	Cookie usage
1.	Sending file	Initiate processing a file on the client side. Send file through the load-balancer. (See POST File Analysis API)	Cookie should not be sent
2.	Sending file	Save the cookie and the data_id you got from the load-balancer	Save cookie
3.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Send cookie saved in step 2
4.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Send cookie saved in step 2

Batch scanning

Step	Stage	Task	Cookie usage
1.	Open batch	Initiate processing file(s) in batch. Request a batch ID through the load-balancer. (See POST Initiate Batch API)	Cookie should not be sent
2.	Open batch	Save the cookie and the batch_id you got from a Core server through the load-balancer.	Save cookie
3.	Sending files	Send file through the load-balancer. (See POST File Analysis (using batch header) API)	Send cookie saved in step 2
4.	Sending files	Save the data_id you got from the load-balancer.	-
	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Send cookie saved in step 2
5.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
6.	Getting batch status	See GET Batch Status API	Send cookie saved in step 2
7.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Send cookie saved in step 2
8.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API	Send cookie saved in step 2

If it does not matter which upstream server responds, the queries should be sent without cookie.

> **Info**
> It is recommended not to send cookies when it's not necessary to allow load-balancer to use its own method to share the load between Metadefender Core v5 servers.

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration through load-balancer

OPSWAT products that support HTTP load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	4.3.4	-
OPSWAT Client	Windows: 7.6.247.0 Mac: 10.4.243.0	-
MetaDefender Email Security	4.3.0	-
MetaDefender ICAP Server	4.3.0	-
MetaDefender Vault	1.3.0	-
MetaDefender for Secure Storage	2.0.7	-

DNS Load Balancing

Using this method is logically similar to Layer 7 load-balancing.

Briefly how it works

Client uses a domain name to send a query to a server. Client's DNS server has more "A" records for that name with different IPs. When a client resolves the server's domain name DNS server randomly chooses an IP for that name to send back. When a session is used on the application layer, client should know the IP address of the Core server that handles that specific session. In every other case, client should resolve the domain name with DNS query to let requests to be balanced between Metadefender Core servers.

Single file scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Sending file	Initiate processing a file on the client side. (See POST File Analysis API)	Use IP saved in step 2
4.	Sending file	Save the data_id got from the Core server	-
5.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Use IP saved in step 2
6.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Use IP saved in step 2

Batch scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Open batch	Initiate processing file(s) in batch. Request a batch ID. (See POST Initiate Batch API)	Use IP saved in step 2
4.	Sending files	Send file to the specific Core server with the basch ID saved in step 3. (See POST File Analysis (using batch header) API)	Use IP saved in step 2
5.	Sending files	Save the data_id you got from the Core server.	-
6.	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Use IP saved in step 2
7.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
8.	Getting batch status	See GET Batch Status API	Use IP saved in step 2
9.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Use IP saved in step 2
10.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API).	Use IP saved in step 2

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration via DNS load-balancing

OPSWAT products that support DNS load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	does not support yet	-
MetaDefender Client	does not support yet	-
MetaDefender Email Security	does not support yet	-
MetaDefender ICAP Server	does not support yet	-
MetaDefender Vault	does not support yet	-

SSL connection for PostgreSQL communication

> **Note**
> The SSL connection configuration is possible thanks to the support made by PostgreSQL. For reference: https://www.postgresql.org/docs/14/ssl-tcp.html
>
> MetaDefender Core is tested against this mode on particular version 4.21.1

Step 1: Generate SSL certificate for PostgreSQL server

> **Note**
> Following instructions are for self-sign certificate. For production environment, making sure to use the certificate signed by your trusted CA.

1) Generate a private key

bash

```
openssl genrsa -des3 -out server.key 1024
```

Sample output:

bash

```
Generating RSA private key, 1024 bit long modulus ........++++++
...............++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

Type your desired password / pass phrase to encrypt the private key

2) Remove the pass phrase to automatically start up the PostgreSQL server

bash

```
openssl rsa -in server.key -out server.key
```

> **Note**
> On Linux, make sure to set permission on the server.key file

bash

```
chmod og-rwx server.key
```

3) Create a self-signed certificate

bash

```
openssl req -new -key server.key -days 3650 -out server.crt -x509
```

You will be prompted to enter detailed information which is incorporated into your self-signed certificate request.

none

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:CA
Locality Name (eg, city) [Default City]:San Francisco
Organization Name (eg, company) [Default Company Ltd]:OPSWAT
Unit Name (eg, section) []:OPSWAT
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:dummy_email@opswat.com
```

For self-signed certificate, use the server ceriticate as the trusted root certificate:

bash

```
cp server.crt root.crt
```

Copy **server.key**, **server.crt**, **root.crt** to PostgreSQL data folder. For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data**
- Linux: **/var/lib/ometascan/pg_data/**

Step 2: Configure PostgreSQL server for SSL authentication connection

1) Create a custom config file (e.g. ssl.conf) for PostgreSQL.

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\postgres\ssl.conf**
- Linux: **/var/lib/ometascan/postgres/ssl.conf**

2) Content of ssl.conf:

none

```
ssl = on
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
```

3) Modify pg_hba.conf

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data\pg_hba.conf**
- Linux: **/var/lib/ometascan/pg_data/pg_hba.conf**

Add following (modify **::1/128** if needed to match with your environment requirement)

none

```
hostssl all all      ::1/128      scram-sha-256
```

After the modify, the content of pg_hba.conf should be like this (example):

> **Note**
> In a Linux environment, a local connection is required to upgrade from version 5.5.1 or earlier to version 5.6.0 or later.

none

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local         DATABASE  USER  METHOD  [OPTIONS]
# host          DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl     DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostgssenc    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnogssenc  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
local    all all                    scram-sha-256
hostssl all all      ::1/128      scram-sha-256

#More content.......
```

4) Reload the services

- Standalone DB mode: Restart **ometascan** service (MetaDefender Core service)
- Shared DB mode: Restart **ometascan-pg** / **ometascan-postgresql** service (MetaDefender Core PostgreSQL service) and then **ometascan** service (MetaDefender Core service)

Test your SSL database connection:

bash

```
psql.exe "sslmode=require dbname=postgres user=postgres host=localhost port=5432"
Password for user postgres:
psql (12.7)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
Type "help" for help.
```

This message indicate that you are now using SSL connection:

bash

```
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
```

Performance and Load Estimation

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Some factors that affect performance

- MetaDefender Core version
- MetaDefender Core engine package and configuration
  - set of engines (which and how many)
  - product configuration (e.g., thread pool size)
- System environment
  - server profile (CPU, RAM, hard disk)
  - client application location - remote or local
  - system caching and engine level caching
- Dataset
  - encrypted or decrypted
  - file types
    - different file types (e.g., document, image, executable)
    - archive file or compound document format files
  - file size
  - bad or unknown (assume to be clean)
- Performance tool

How test results are calculated

Performance (mainly scanning speed) is measured by throughput rather than unit speed. For example, if it takes 10 seconds to process 1 file, and it also takes 10 seconds to process 10 files, then performance is quantified as 1 second per file, rather than 10 seconds.

- total time / total number of files processed: 10 seconds / 10 files = 1 second / file.

Standalone DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.2.416.0

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

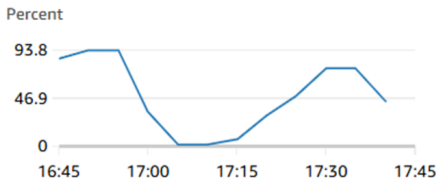
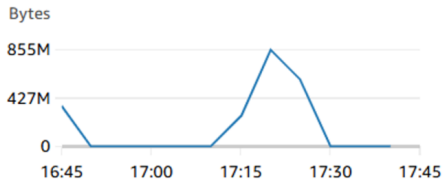
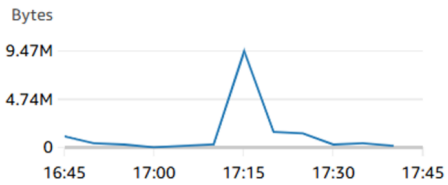
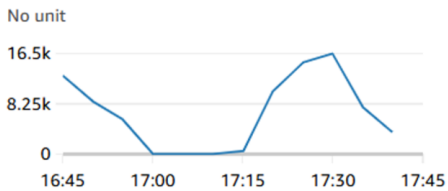
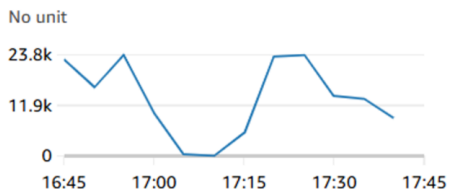
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV)	24.2	47.2 / 80	37.3 / 41.1		315,310	0.0114
Deep CDR	19.9	48.9 / 81.8	32.6 / 37		383,442	0.0094
Proactive DLP	17.1	36.8 / 58.2	30.4 / 33.6		446,228	0.0081
Vulnerability	10.1	59.2 / 85	29.3 / 33.3		755,495	0.0048

MetaDefender Core with common engine packages:

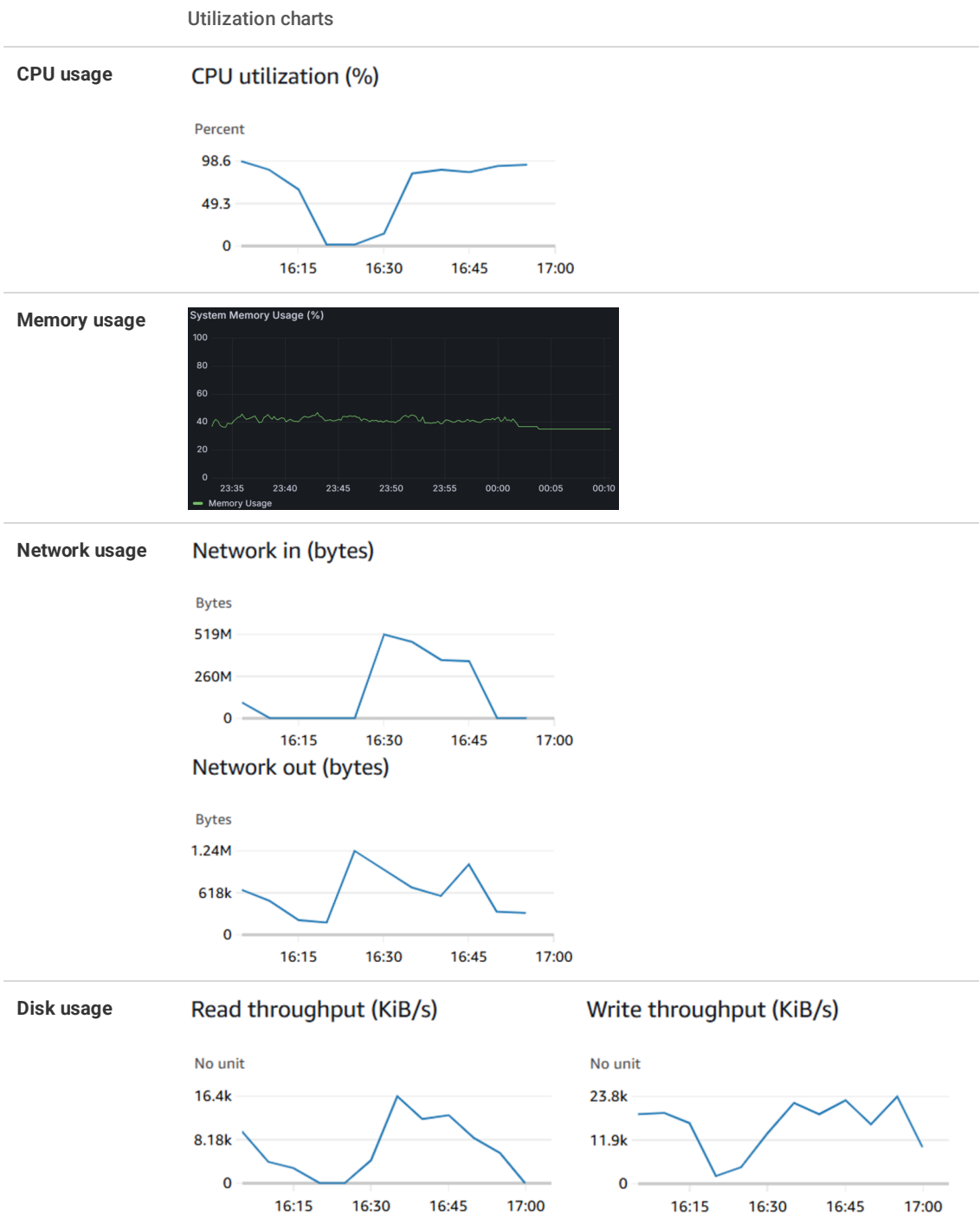
Use case	Scan duration	Avg./Max CPU usage	Avg./Max RAM usage	Avg./Max Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV) + Deep CDR	30.8	68.1 / 88	40.1 / 46.8		247,744	0.0145
Metascan (8 AV) + Deep CDR + Proactive DLP	31.2	74.2 / 98.6	41.5 / 51.1		244,567.3	0.0147
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	31.8	73.5 / 98	40.6 / 51.5		239,953	0.015

System resource utilization charts

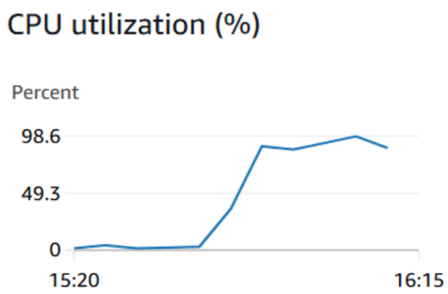
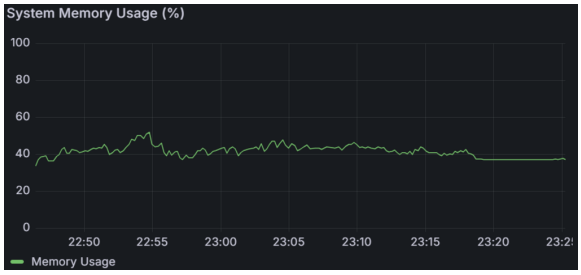
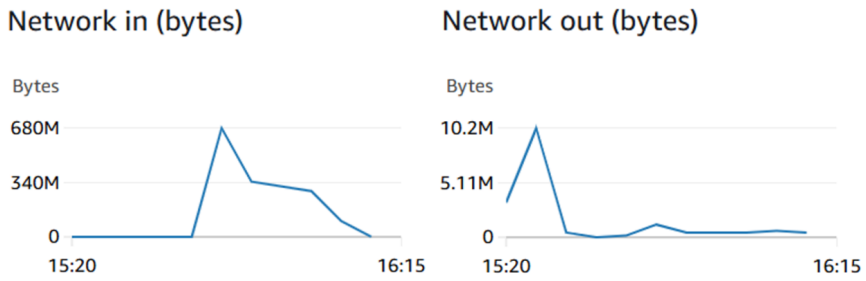
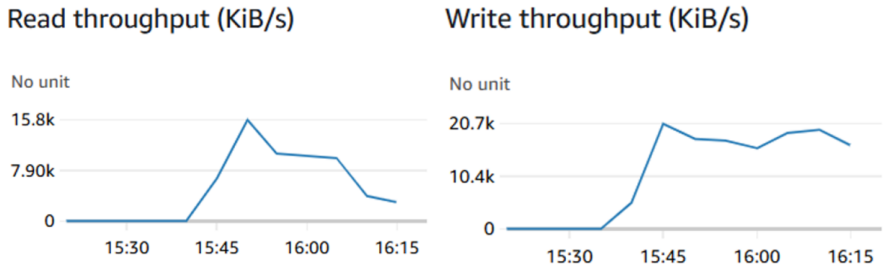
MetaDefender Core with Metascan (8 AV engines):

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

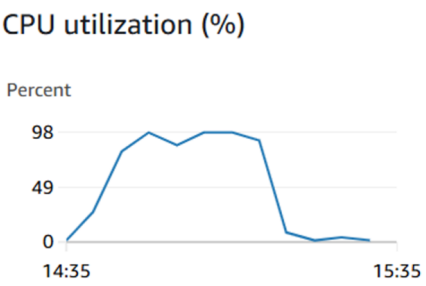
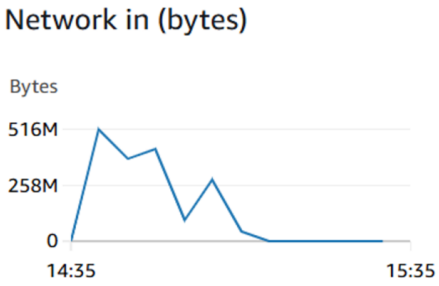
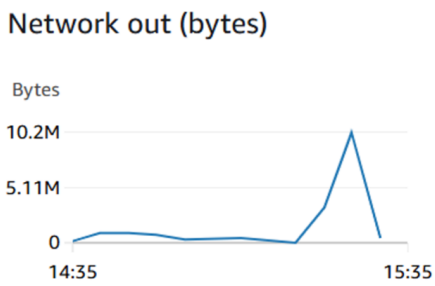
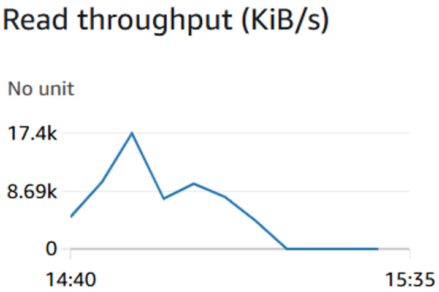
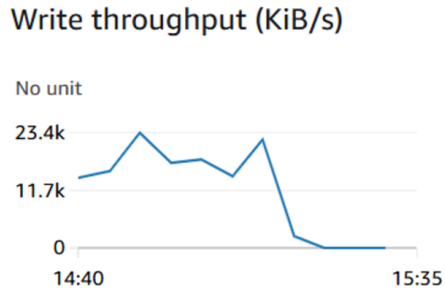
MetaDefender Core with Metascan (8 AV engines) + Deep CDR:



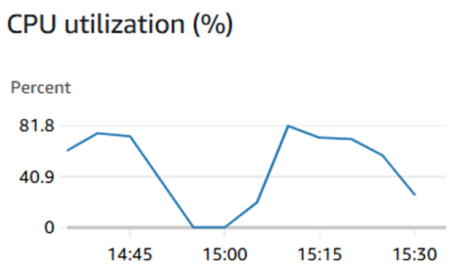
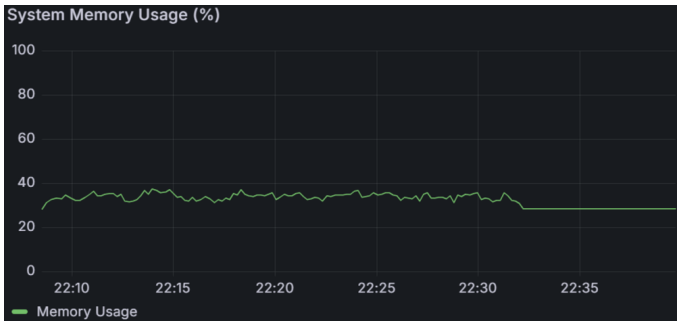
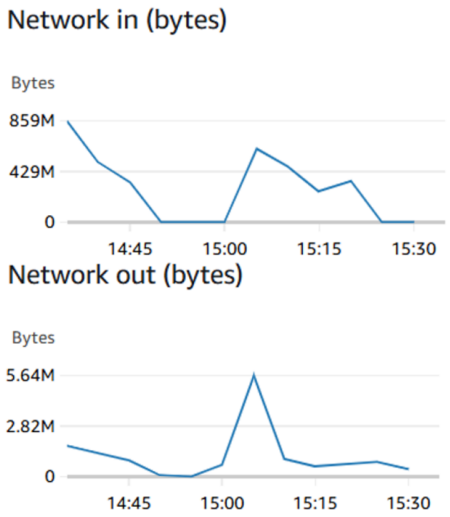
MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

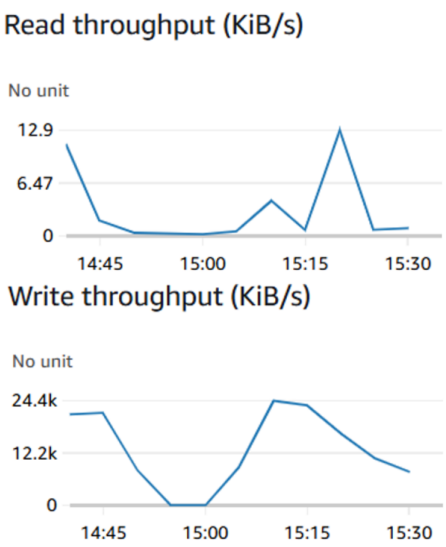
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

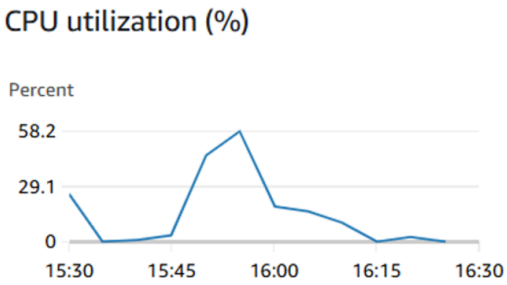
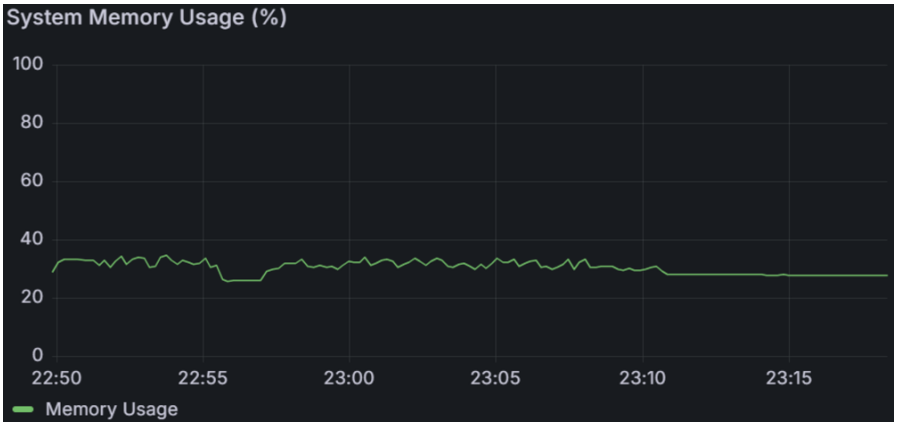
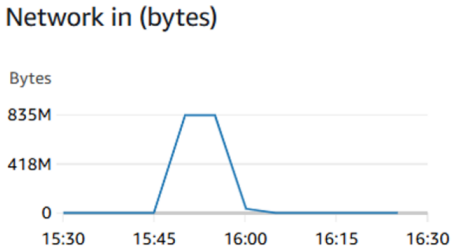
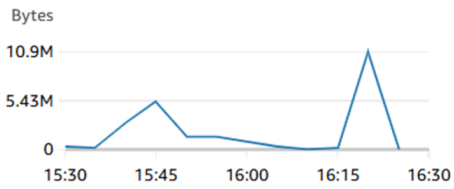
	Utilization charts
CPU usage	
Memory usage	
Network usage	 
Disk usage	 

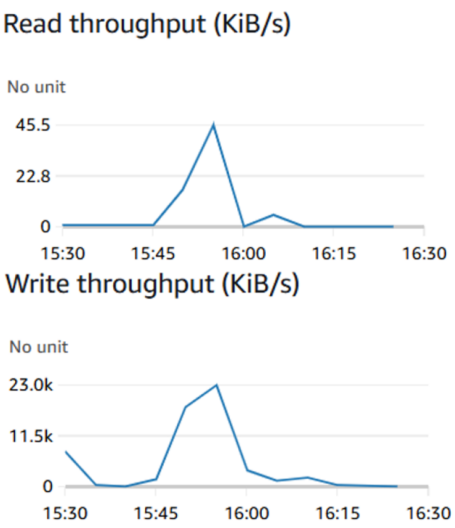
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

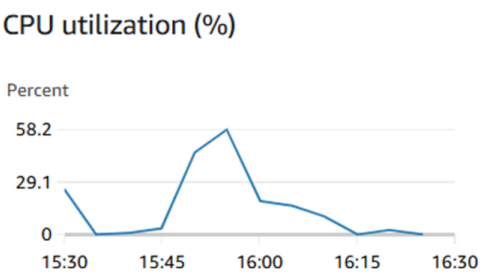
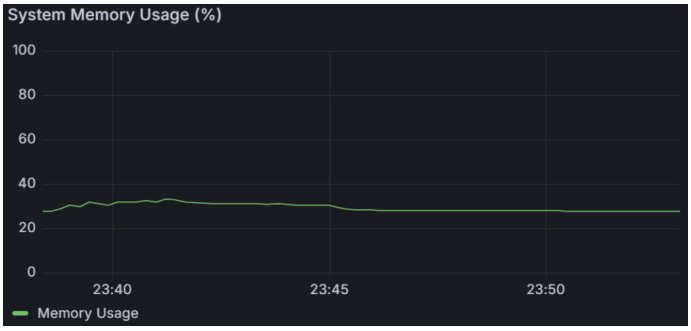
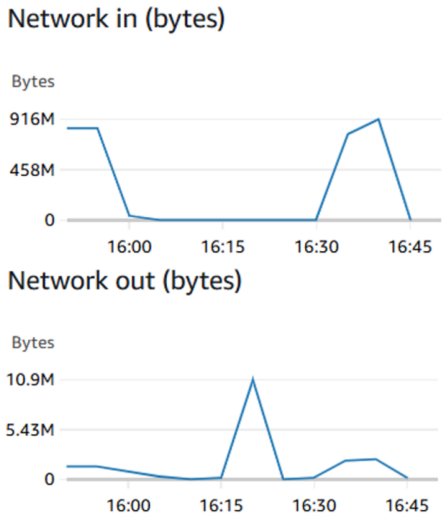
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	 

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability:

	Utilization charts
CPU usage	
Memory usage	
Network usage	

Utilization charts

Disk usage



Standalone DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Ubuntu 22.04	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.57-236

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

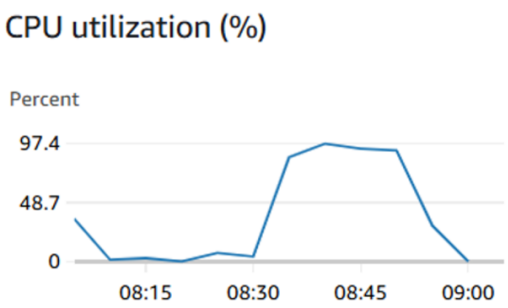
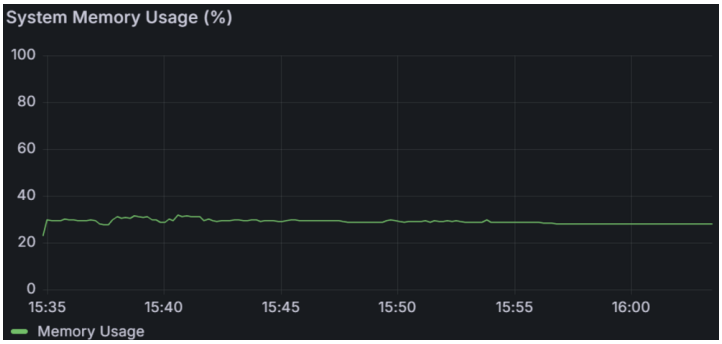
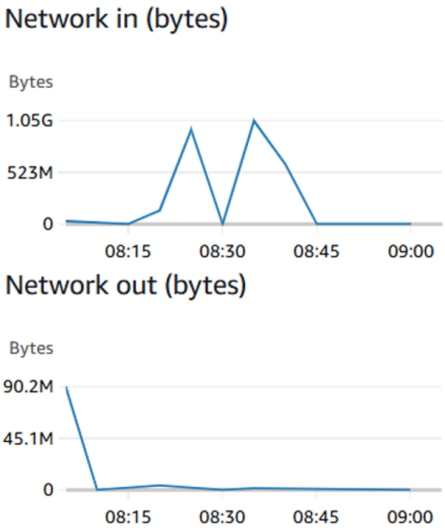
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV)	30.2	69 / 98	29 / 32		252,666	0.0142
Deep CDR	19.4	48 / 62	36 / 49		393,325	0.009
Proactive DLP	16.5	34 / 39	32 / 48		462,455	0.008
Vulnerability	12	50 / 65	46 / 46		635,875	0.006

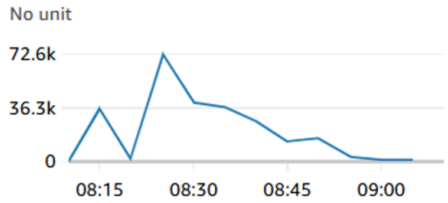
MetaDefender Core with common engine packages:

Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV) + Deep CDR	39.8	78 / 99	62 / 73		191,721	0.0188
Metascan (10 AV) + Deep CDR + Proactive DLP	43.5	61 / 84	47 / 67		175,414	0.0205
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	44.8	62 / 88	69 / 83		170,324	0.0211

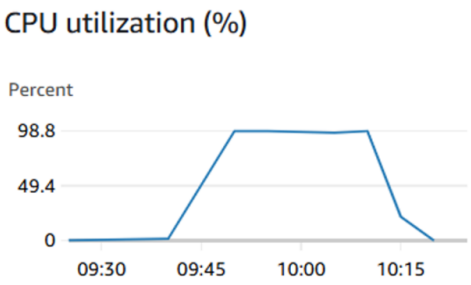
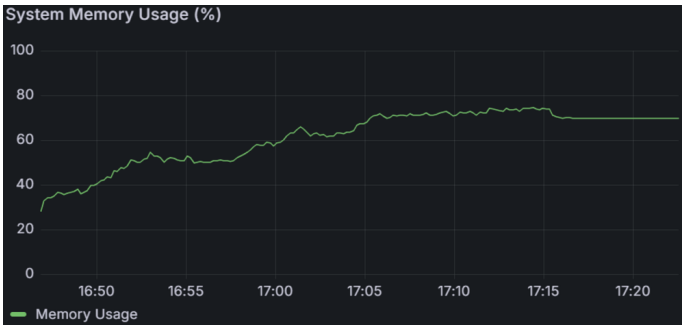
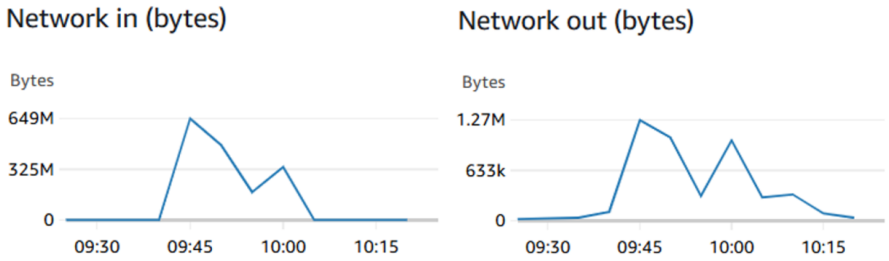
System resource utilization charts

MetaDefender Core with Metascan (10 AV engines) :

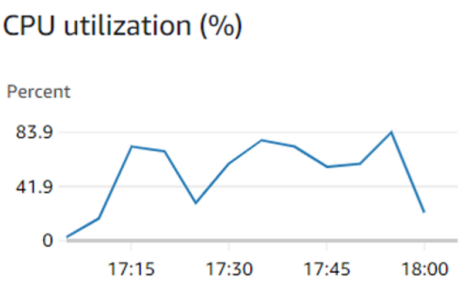
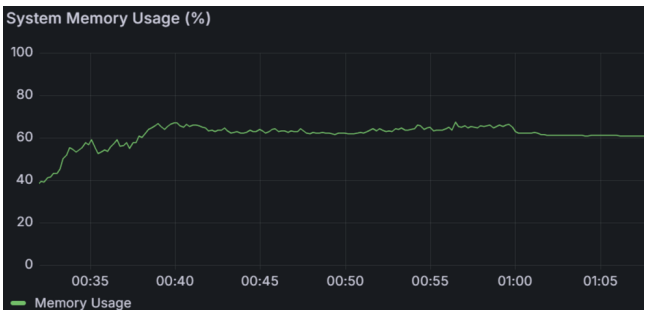
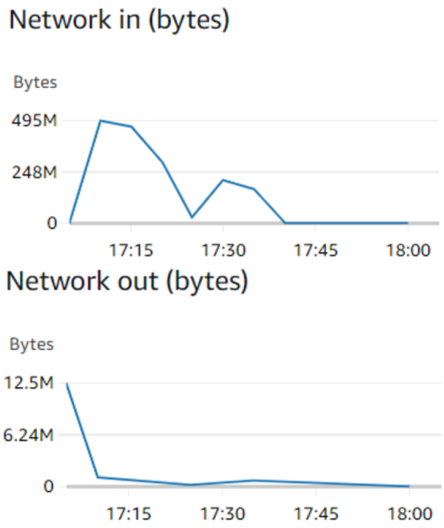
	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

MetaDefender Core with Metascan (10 AV engines) + Deep CDR :

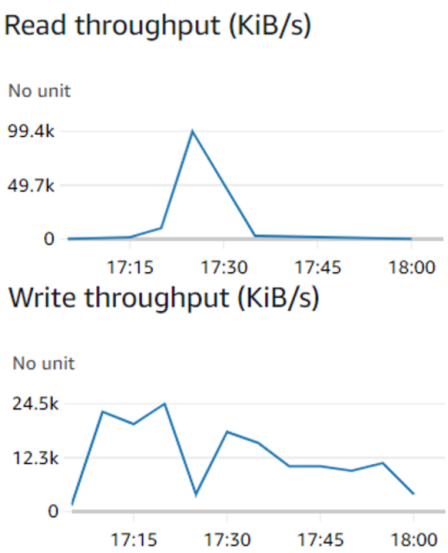
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP :

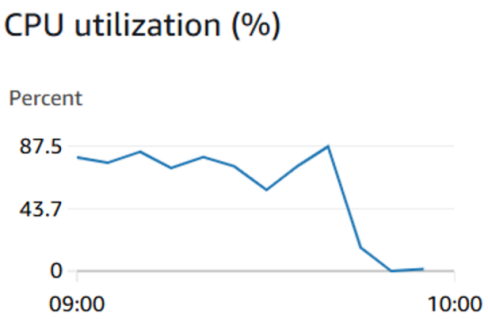
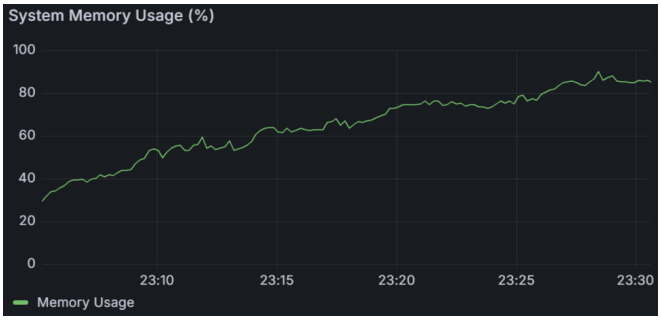
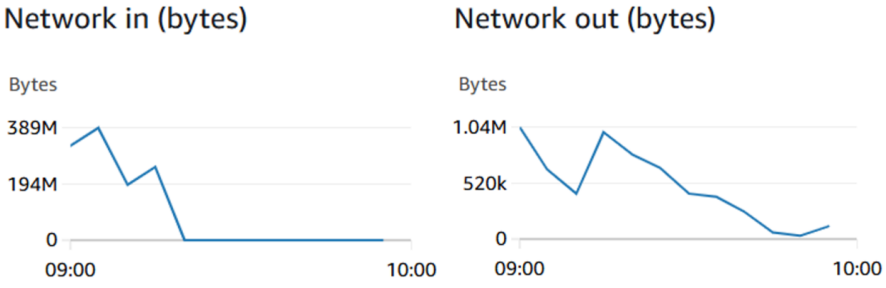
	Utilization charts
CPU usage	
Memory usage	
Network usage	

Utilization charts

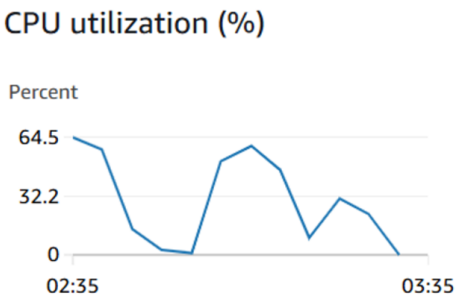
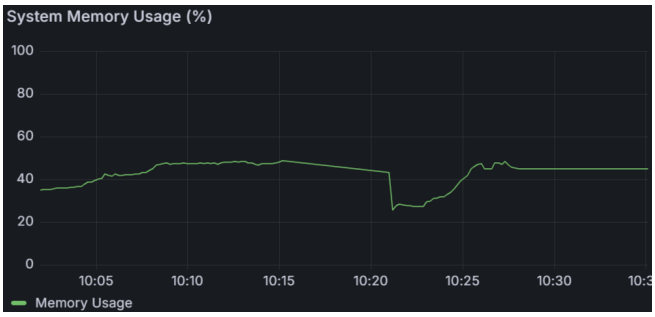
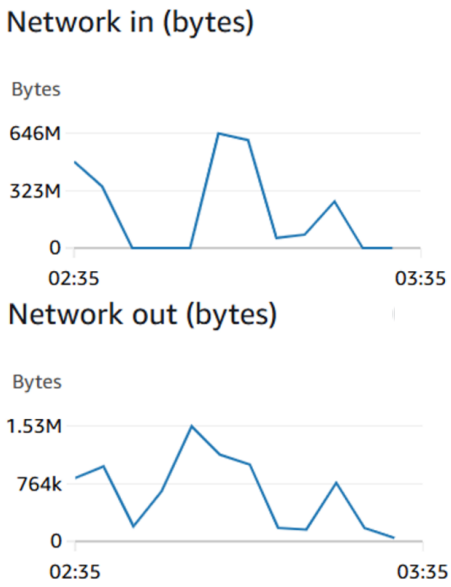
Disk usage

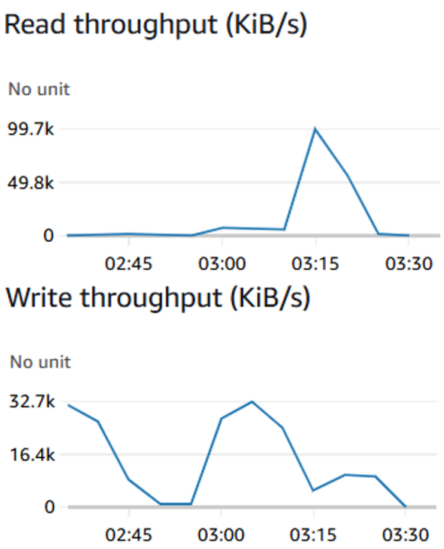


MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP + Vulnerability :

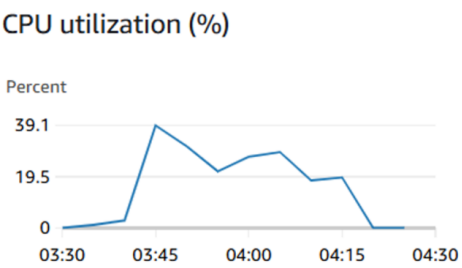
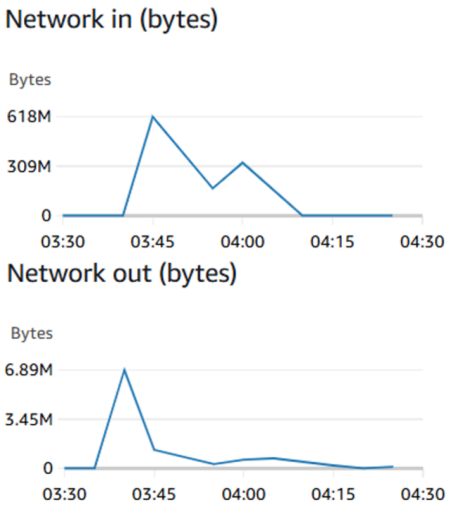
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

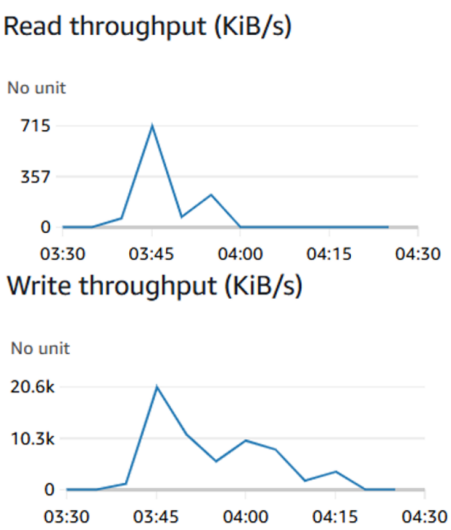
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

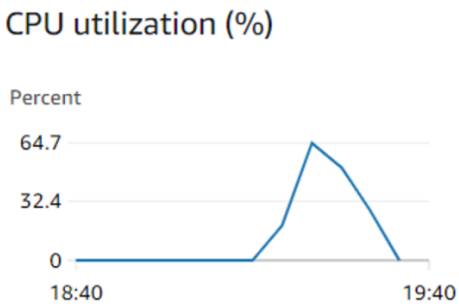
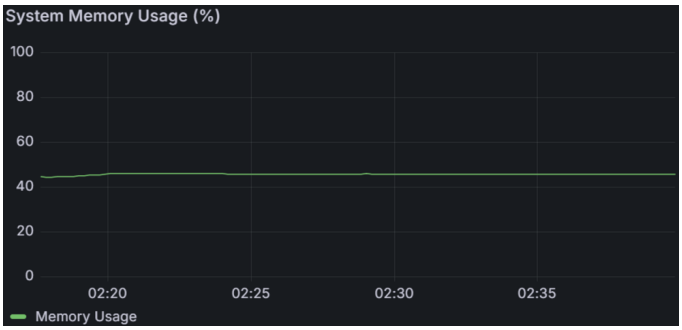
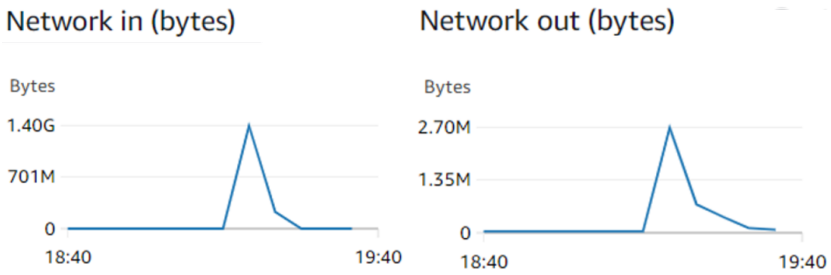
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability :

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

Shared DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

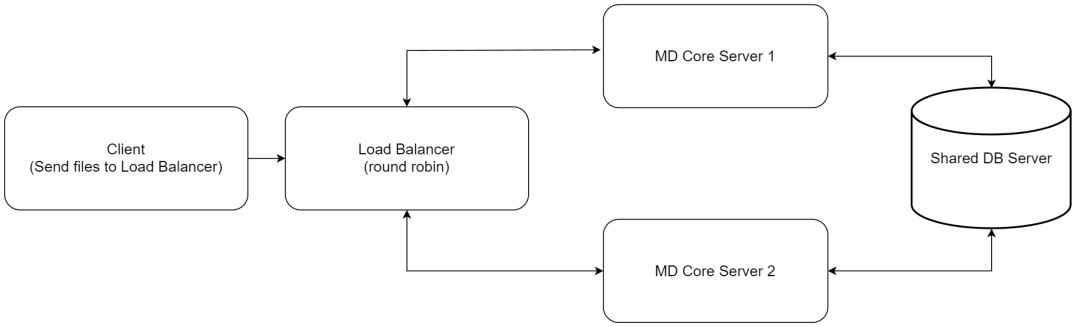
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV)	11.9	37.0	41.9	30.1	30.4	Receive: 6441.1 Send: 769.6	Receive: 6200.2 Send: 693.7	640,821.2	0.0
Deep CDR	10.5	60.1	66.7	30.3	32.2	Receive: 6379.8 Send: 594.8	Receive: 7146.1 Send: 609.6	729,368.5	0.0
Proactive DLP	7.8	38.7	37.5	29.7	31.0	Receive: 8842.6 Send: 791.8	Receive: 8635.6 Send: 729.1	965,455.7	0.0
Vulnerability	4.4	52.2	53.0	26.0	26.9	Receive: 16,215.5 Send: 1006.8	Receive: 15,785.7 Send: 997.2	1714,449.3	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV) + Deep CDR	15.4	72.4	75	34.3	35.4	Receive: 5436.6 Send: 611.3	Receive: 5589.3 Send: 607.2	495,697.4	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP	16.9	77.2	80.2	36.4	37.9	Receive: 5254 Send: 585.4	Receive: 5041.3 Send: 576.1	450,657.6	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	18.0	72.1	80.4	38.6	38.0	Receive: 4504.3 Send: 574.1	Receive: 4649.5 Send: 571.1	423,010	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

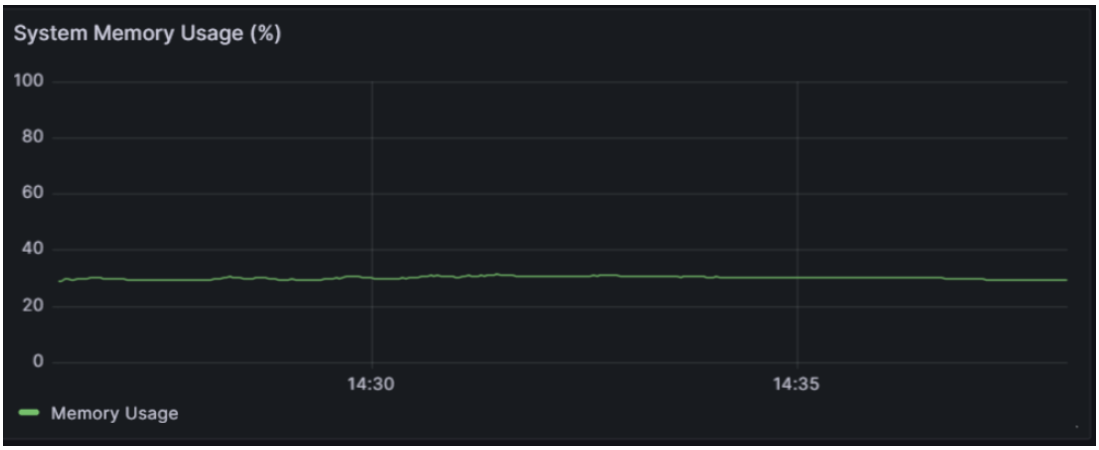
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2

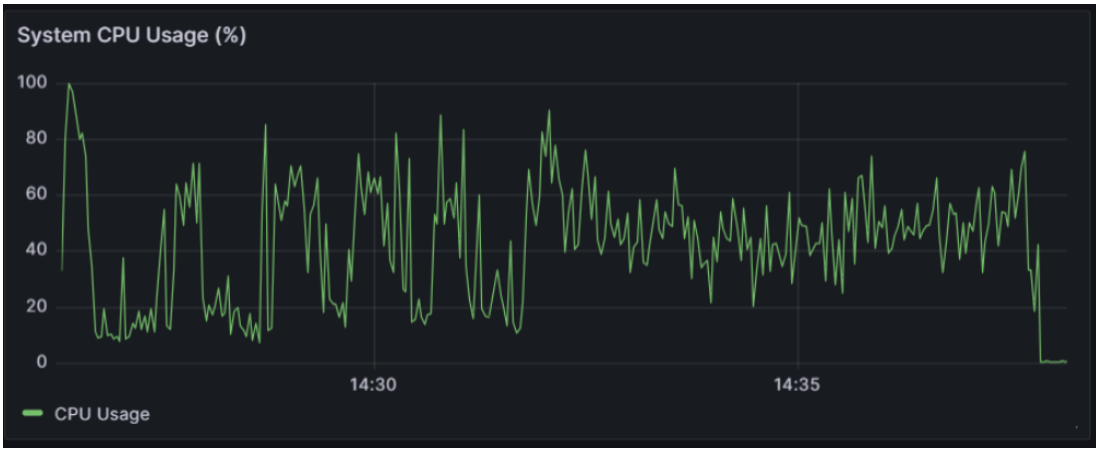


CPU usage

MD Core #1

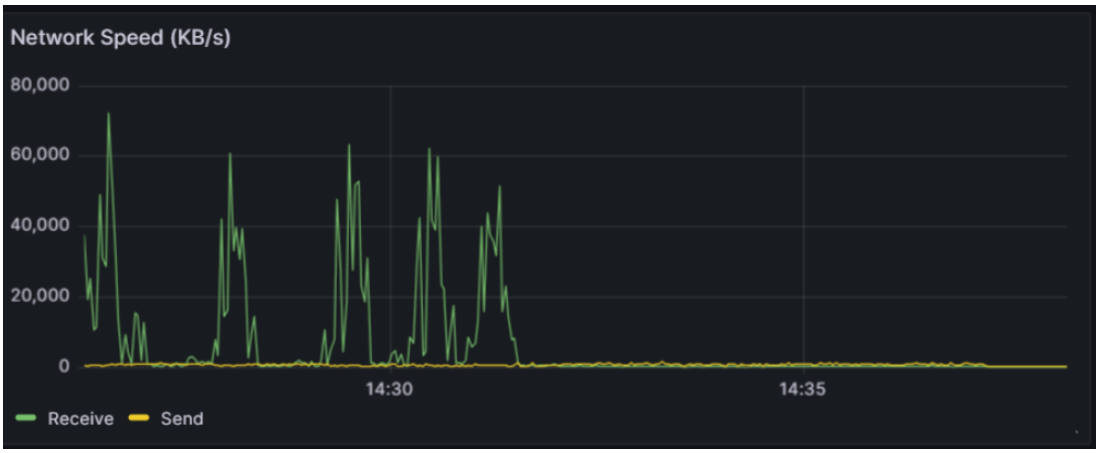


MD Core #2

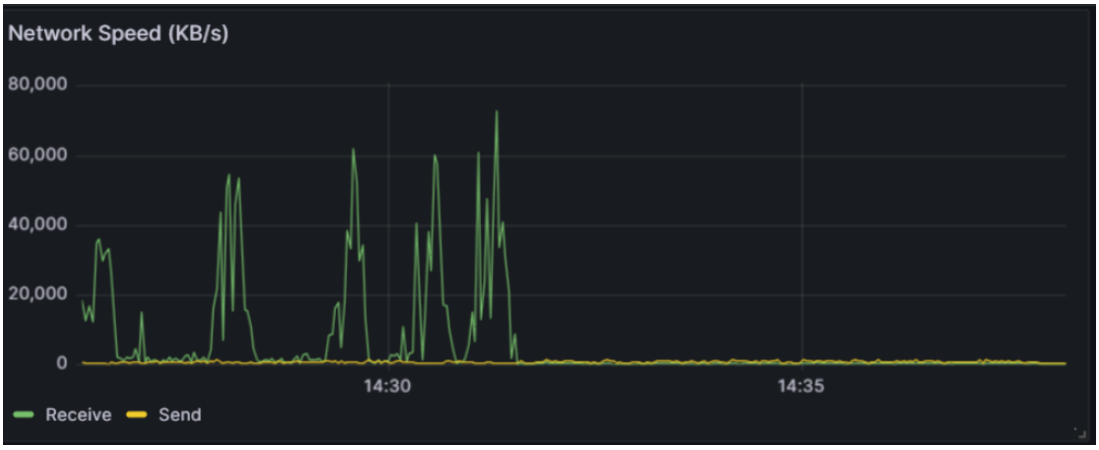


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2

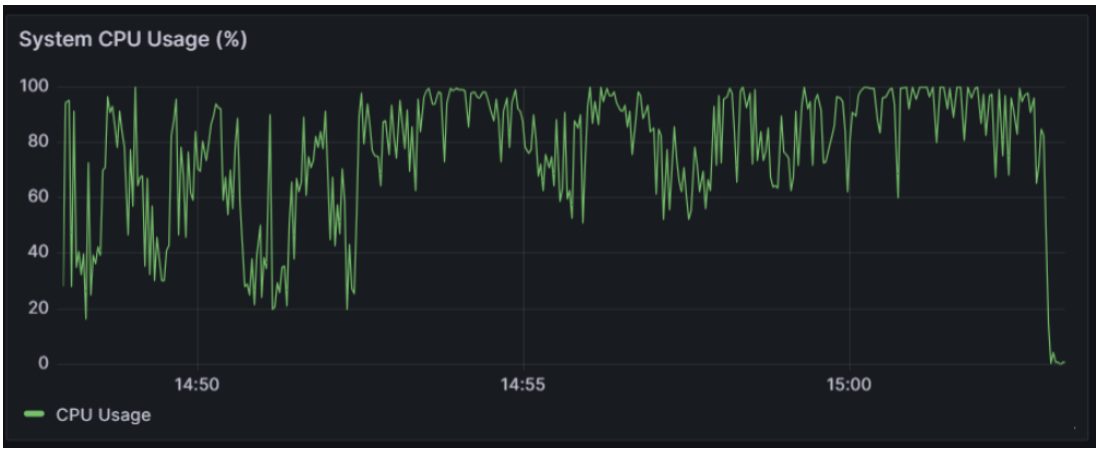


CPU usage

MD Core #1

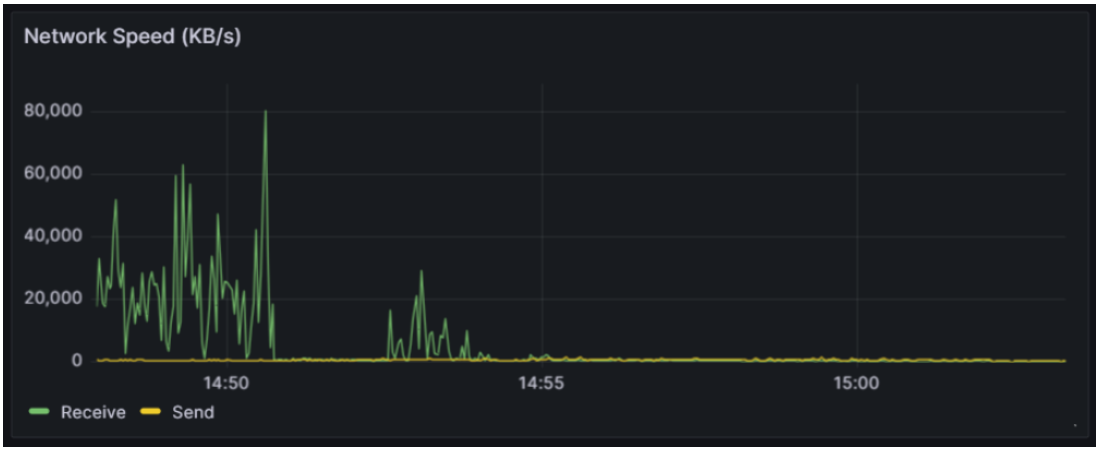


MD Core #2



Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

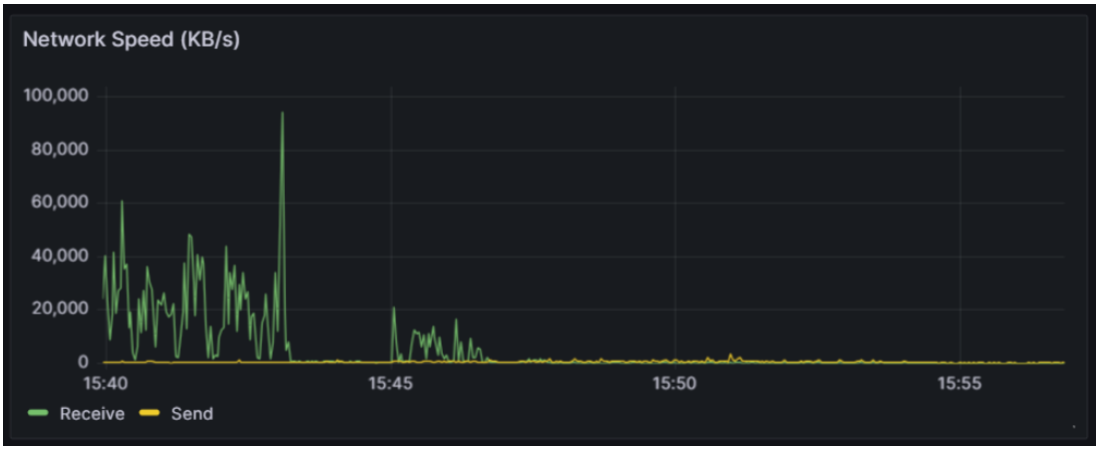


MD Core #2

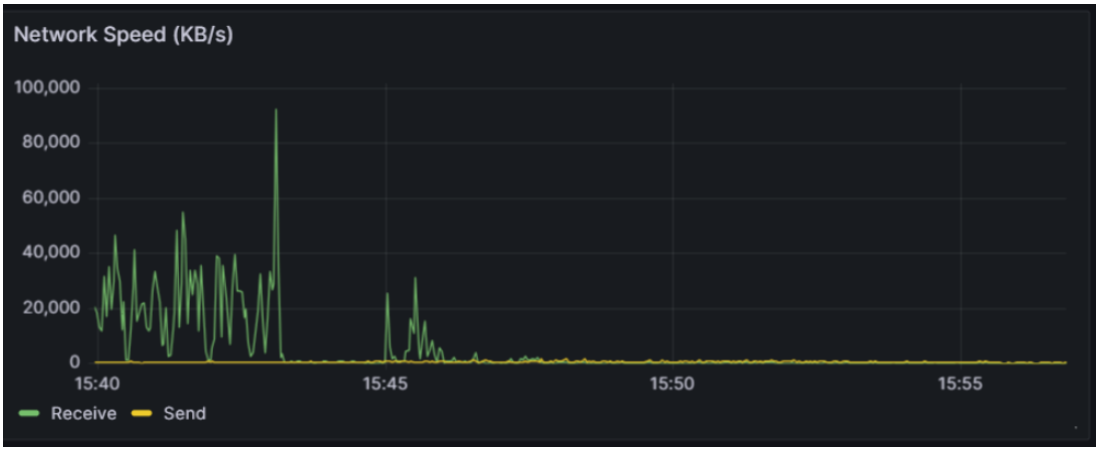


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1

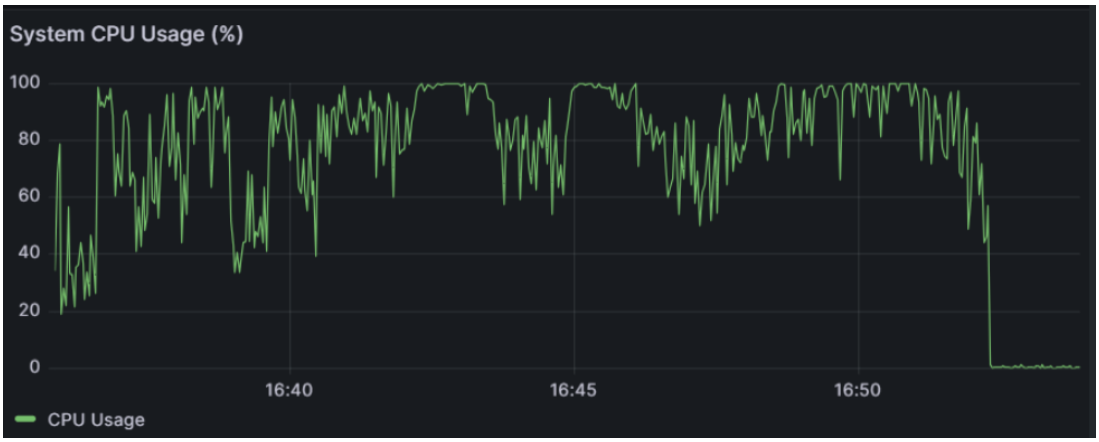


MD Core #2



CPU usage

MD Core #1

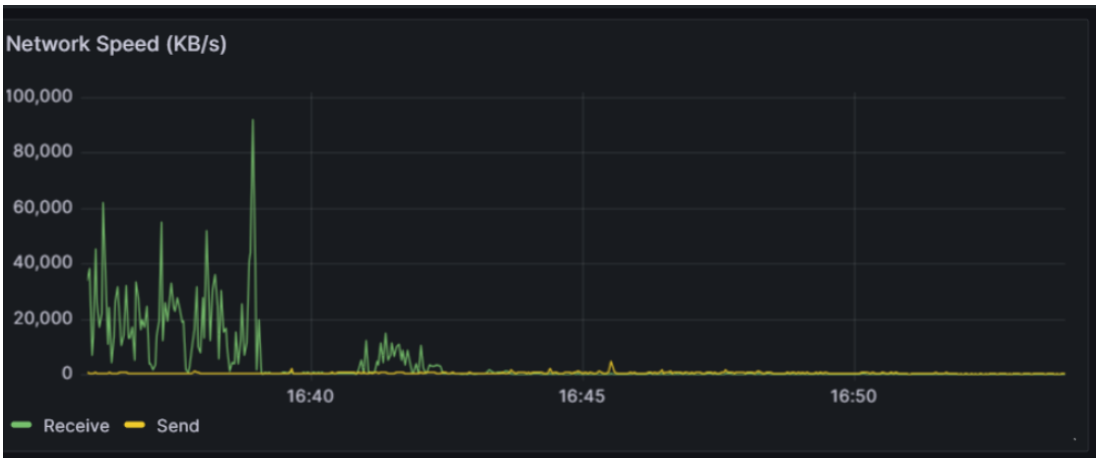


MD Core #2

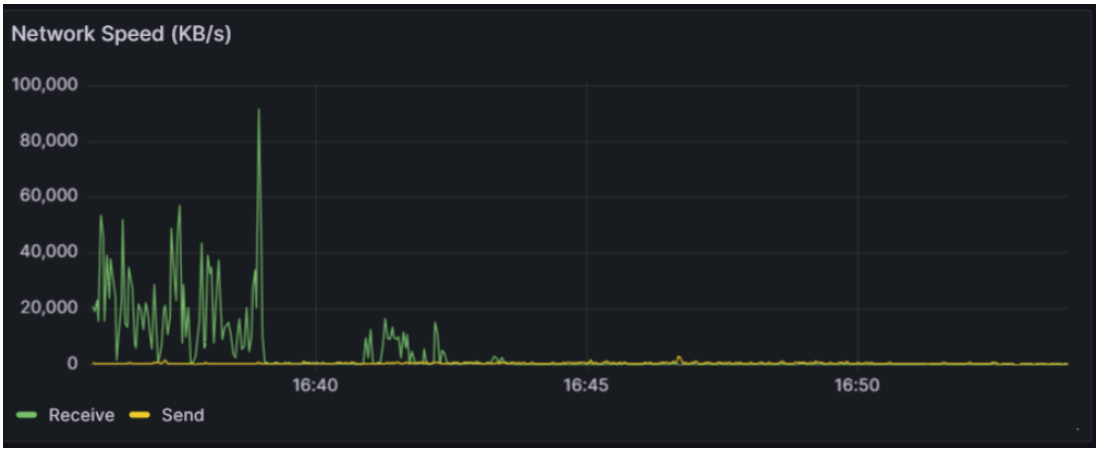


Network usage

MD Core #1



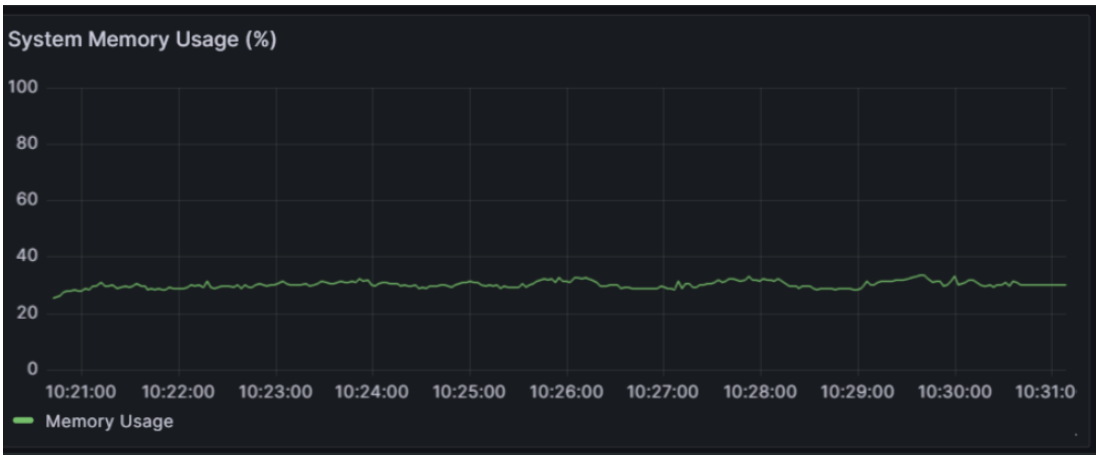
MD Core #2



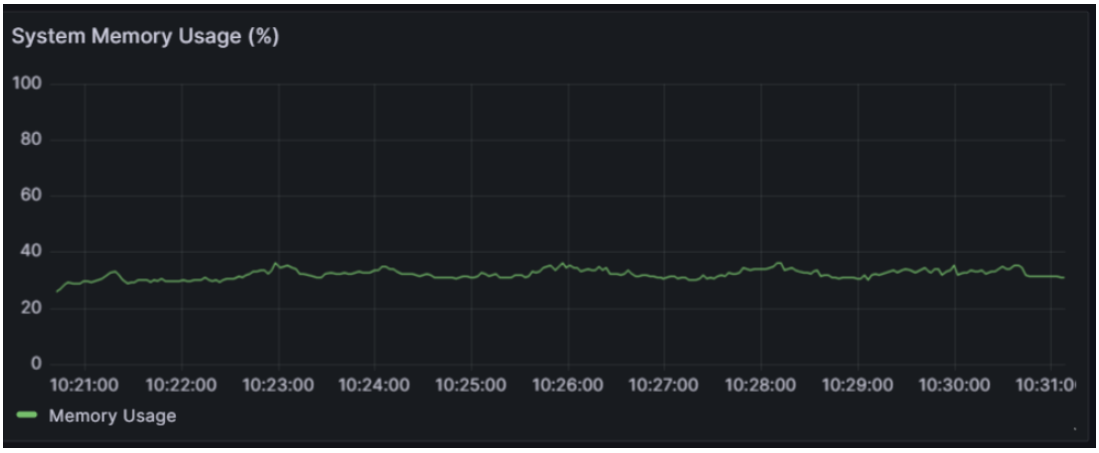
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

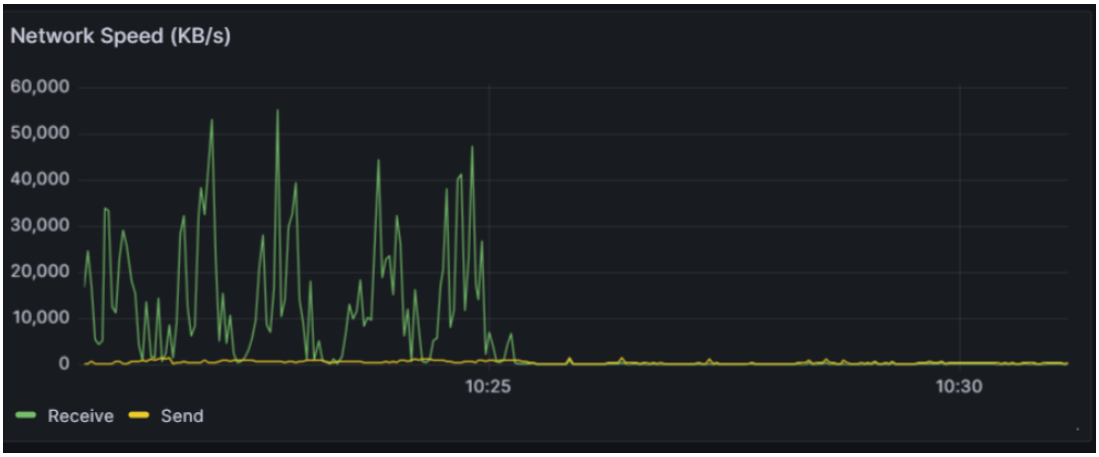


MD Core #2

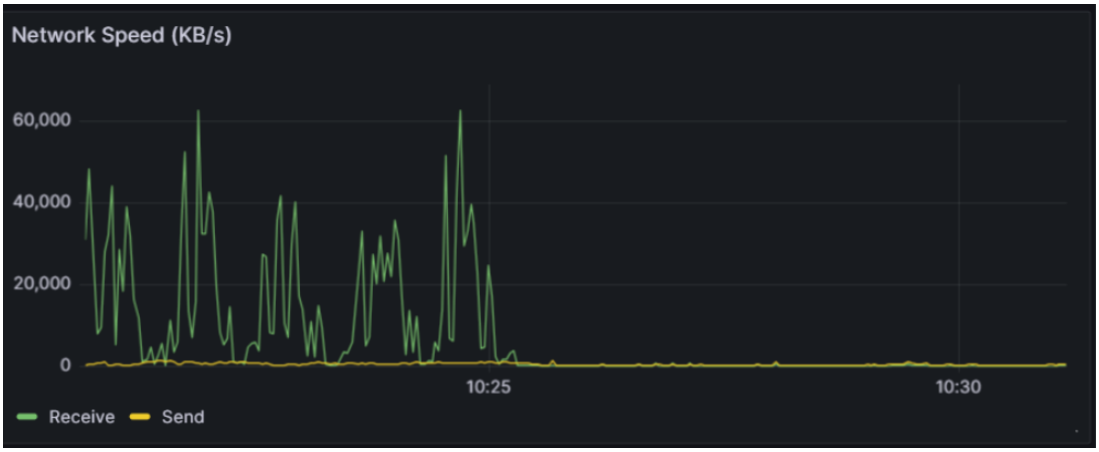


Network usage

MD Core #1



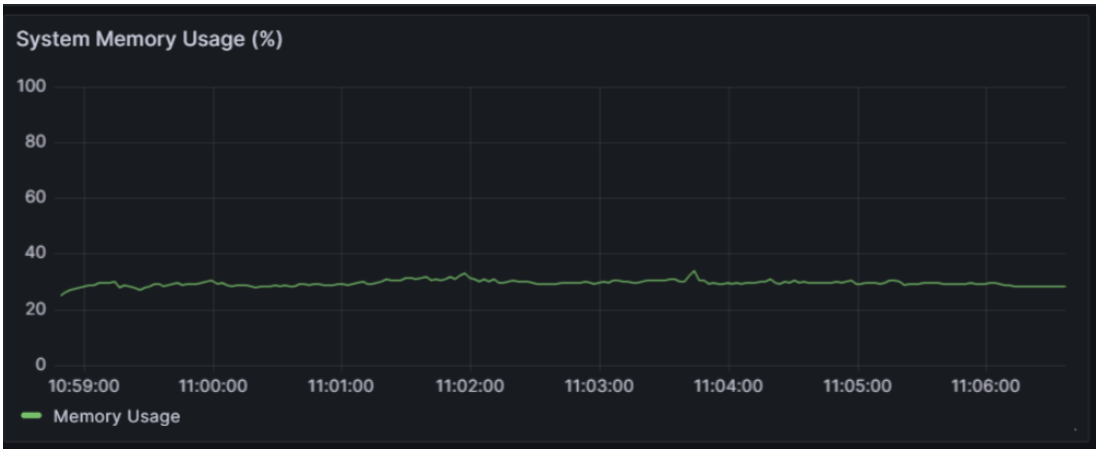
MD Core #2



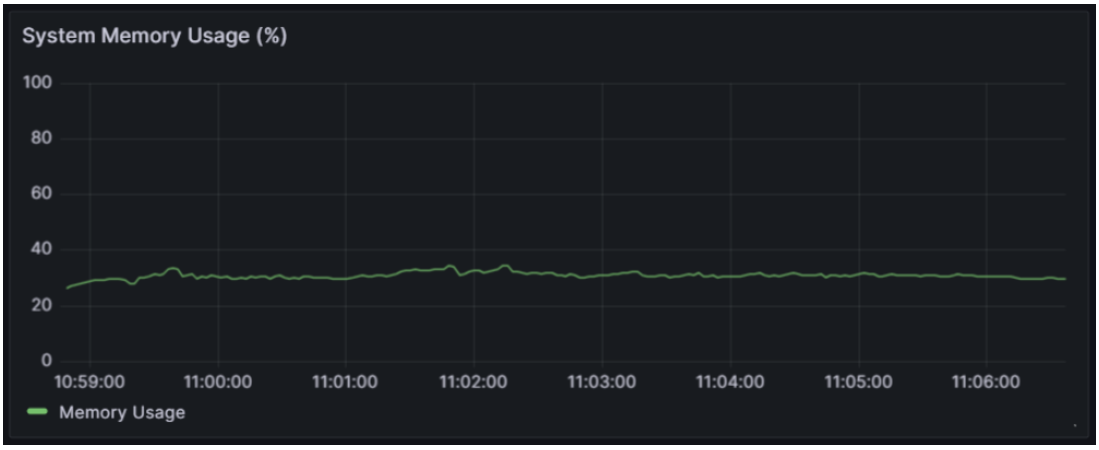
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

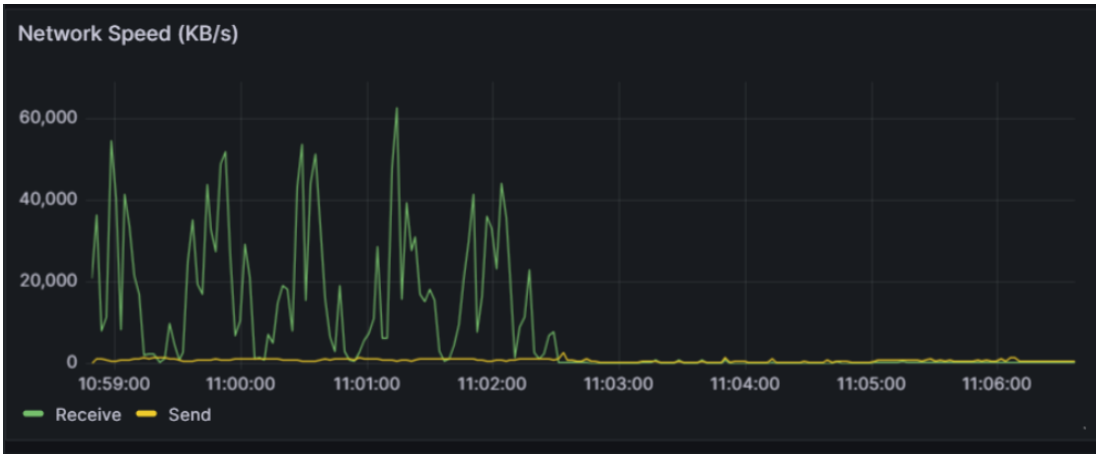


MD Core #2

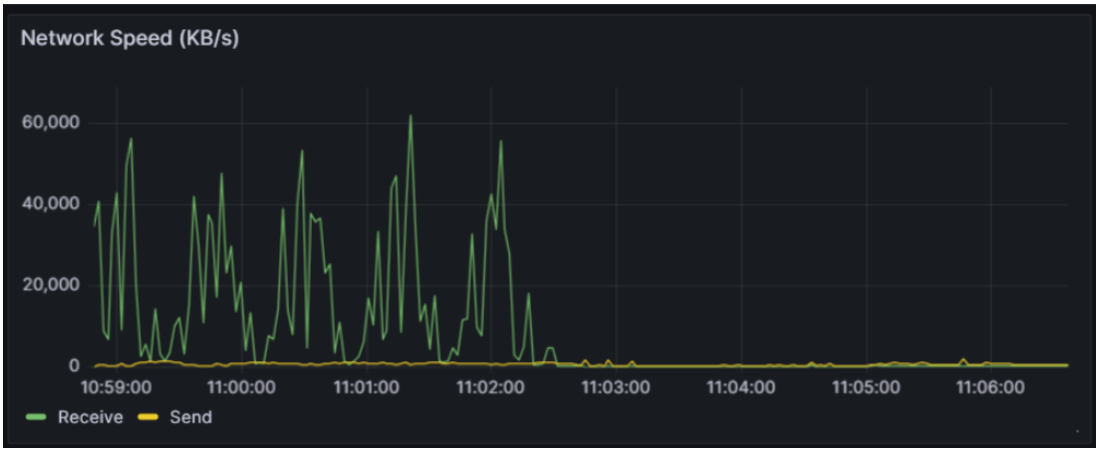


Network usage

MD Core #1



MD Core #2



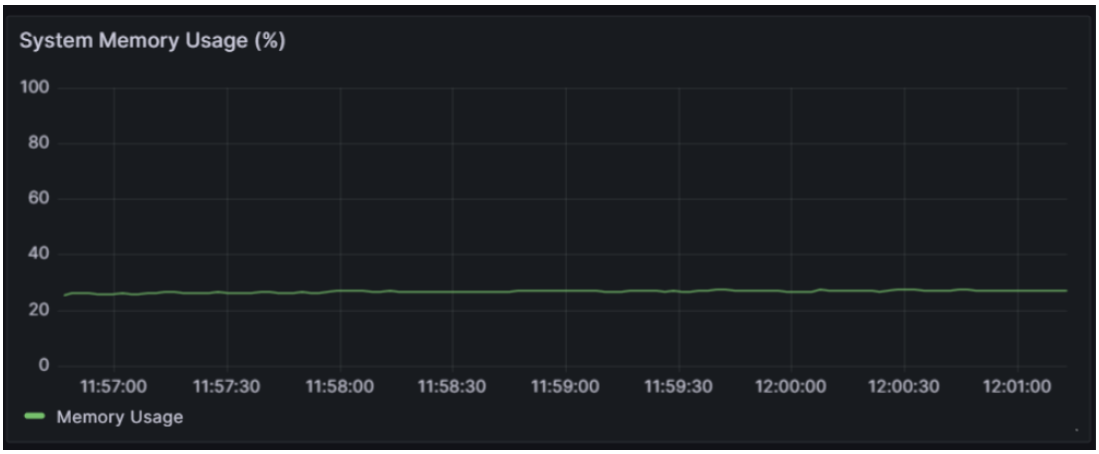
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

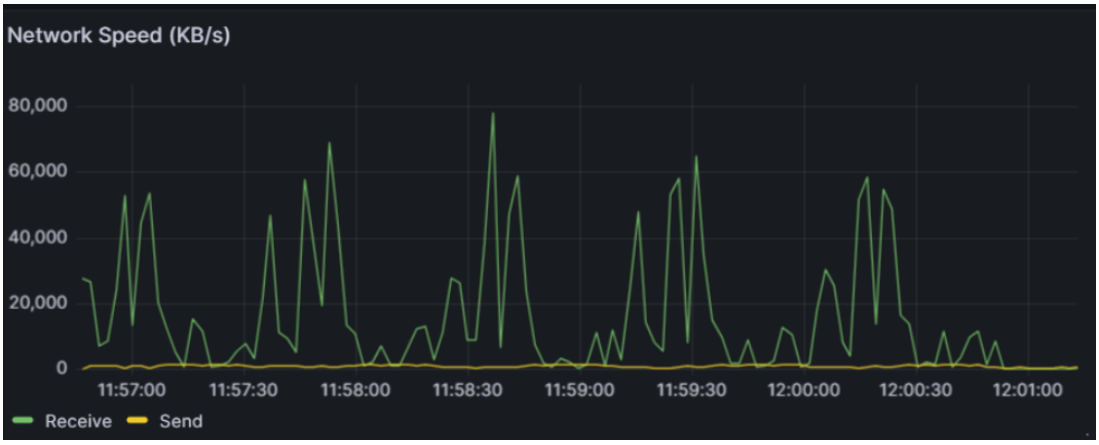


MD Core #2

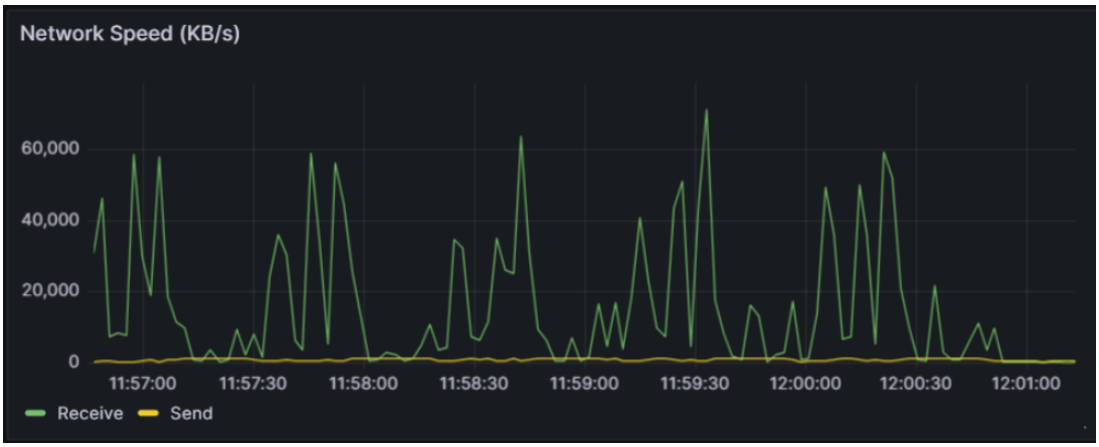


Network usage

MD Core #1



MD Core #2



Shared DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

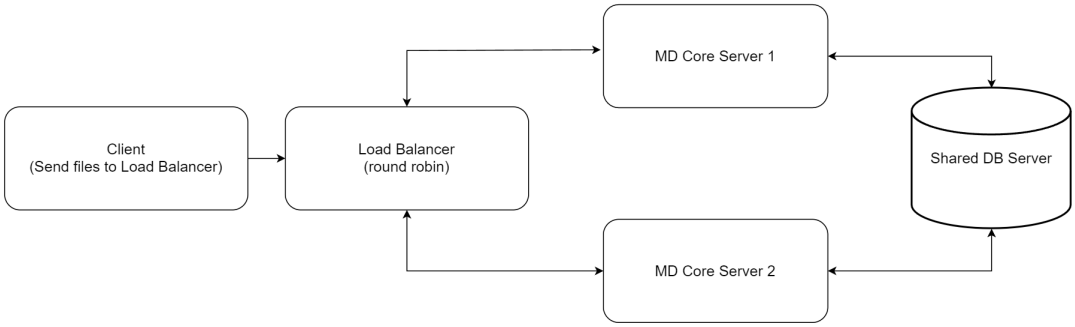
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV)	15.5	63.1	73.3	25.2	25.8	Receive: 4555.9 Send: 694.4	Receive: 4516.2 Send: 692.4	492,620.5	0.0
Deep CDR	9.70	50.7	51.2	31.9	32.5	Receive: 7062.9 Send: 623.2	Receive: 6653 Send: 654.1	786,189.6	0.0
Proactive DLP	7.2	35.2	35.0	27.9	27.5	Receive: 9047.4 Send: 802.4	Receive: 9859.8 Send: 787.5	1,055,110.8	0.0
Vulnerability	5.0	45.7	49.3	24.4	24.5	Receive: 13,585.7 Send: 966.7	Receive: 13,505.8 Send: 987.4	1,519,562.1	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV) + Deep CDR	19.9	79.1	77.6	34.0	34.2	Receive: 3785.9 Send: 573.8	Receive: 3663.1 Send: 577.7	382,716	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP	21.6	80.3	81.5	37.7	44.4	Receive: 3305.6 Send: 543.3	Receive: 3414.3 Send: 556.2	353,694.9	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	22.5	84.2	79.8	37.9	44.7	Receive: 3267.9 Send: 554.9	Receive: 3398.2 Send: 553.8	338,507.6	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

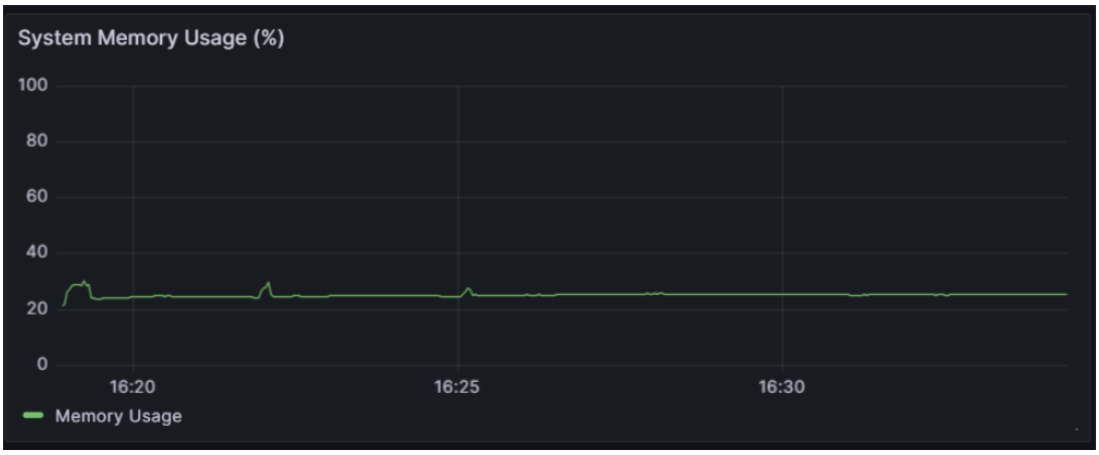
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

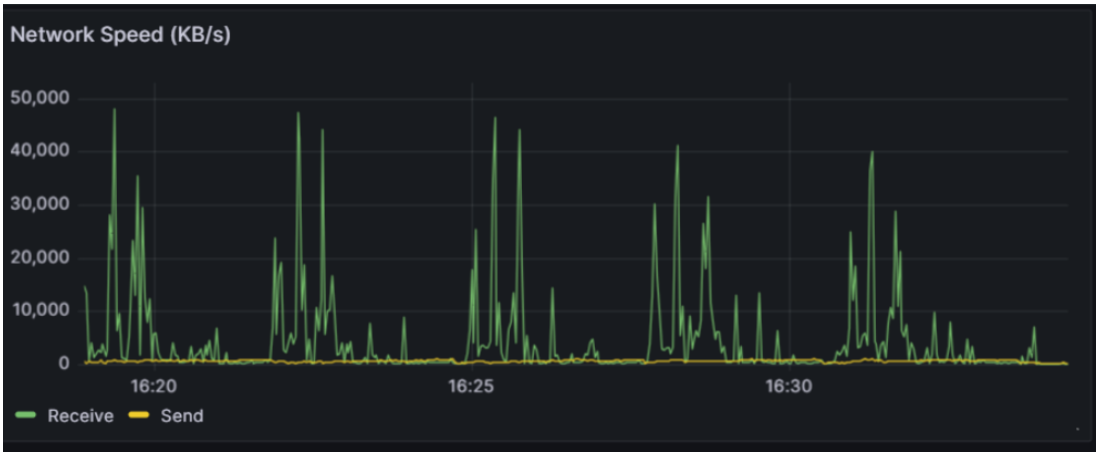


MD Core #2

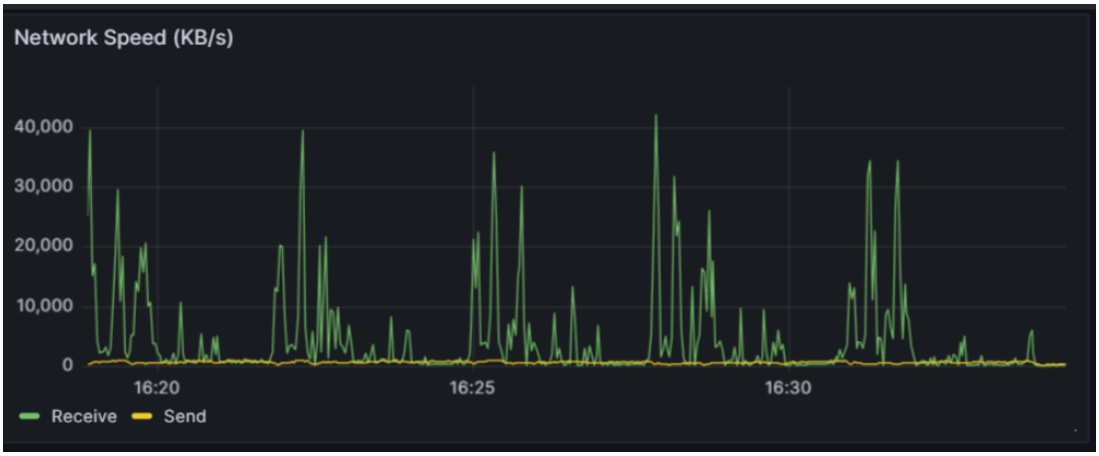


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

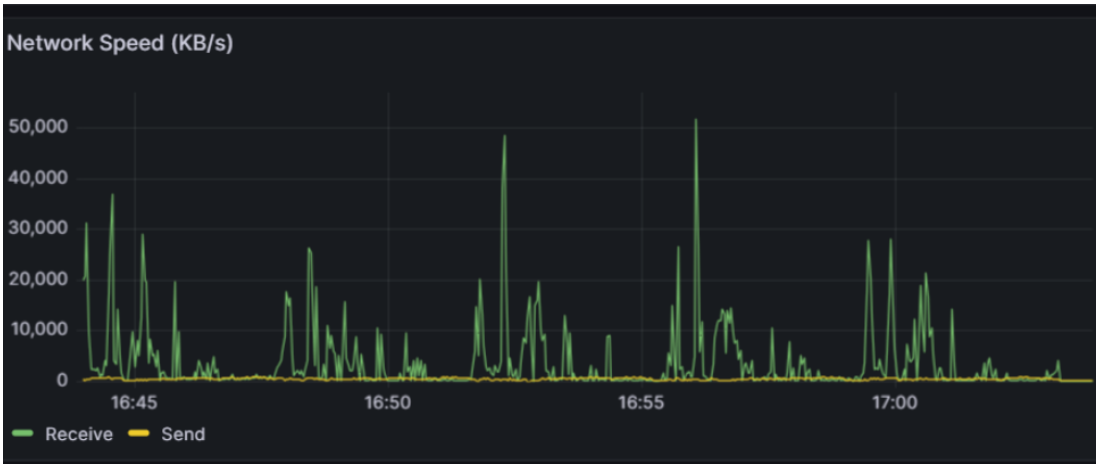


MD Core #2

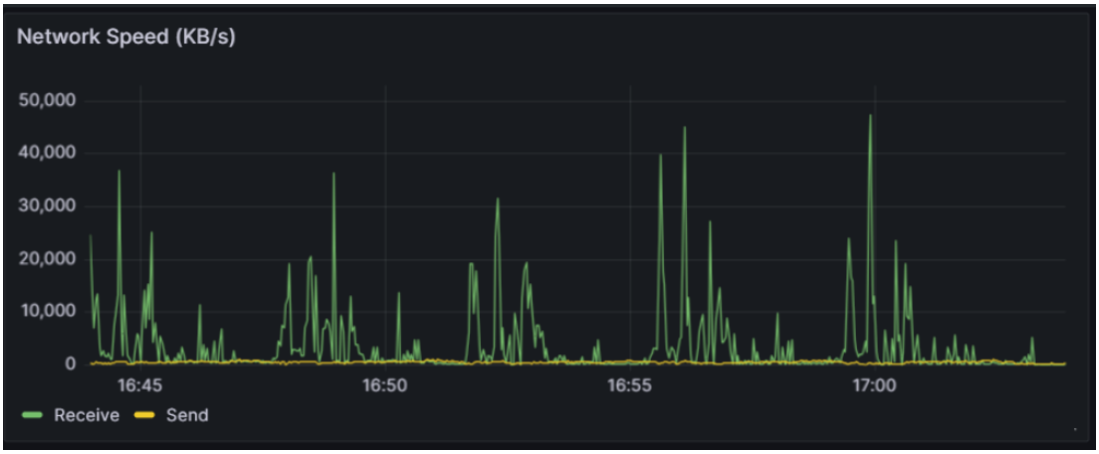


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1

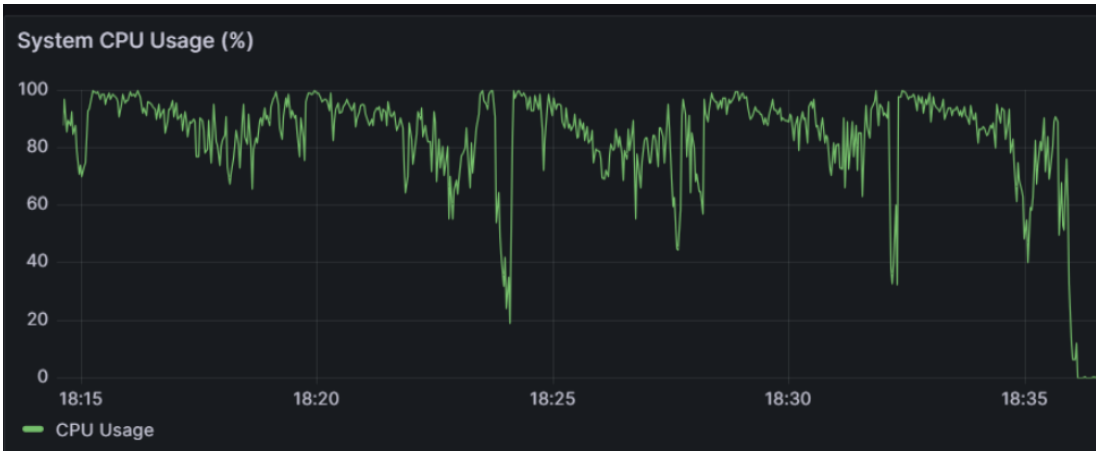


MD Core #2



CPU usage

MD Core #1

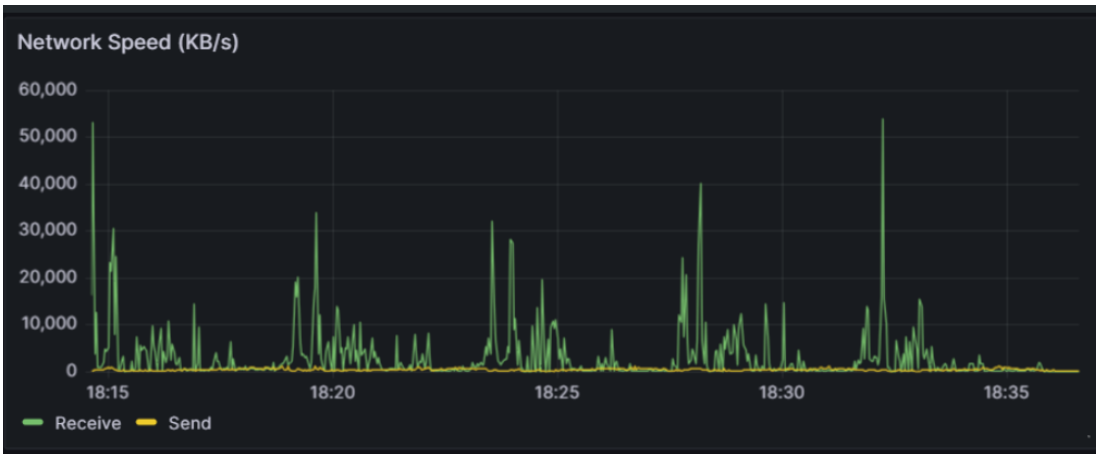


MD Core #2

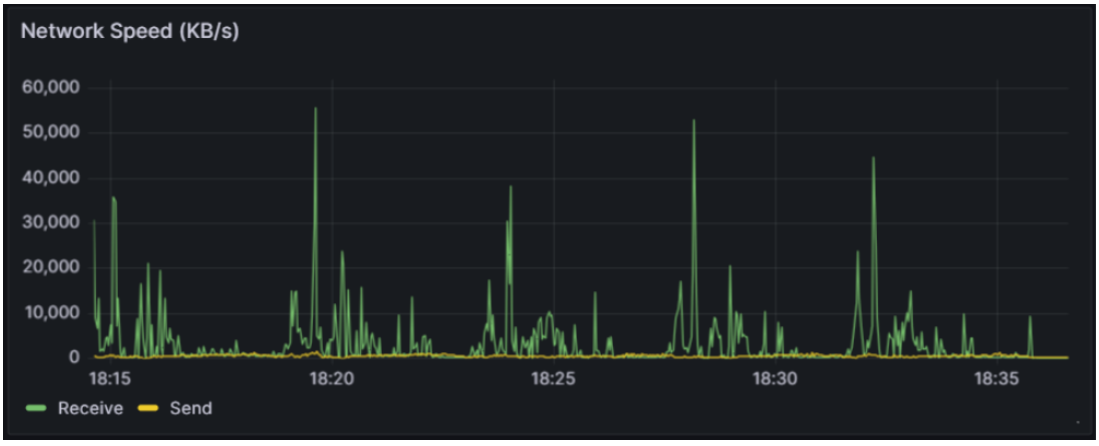


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

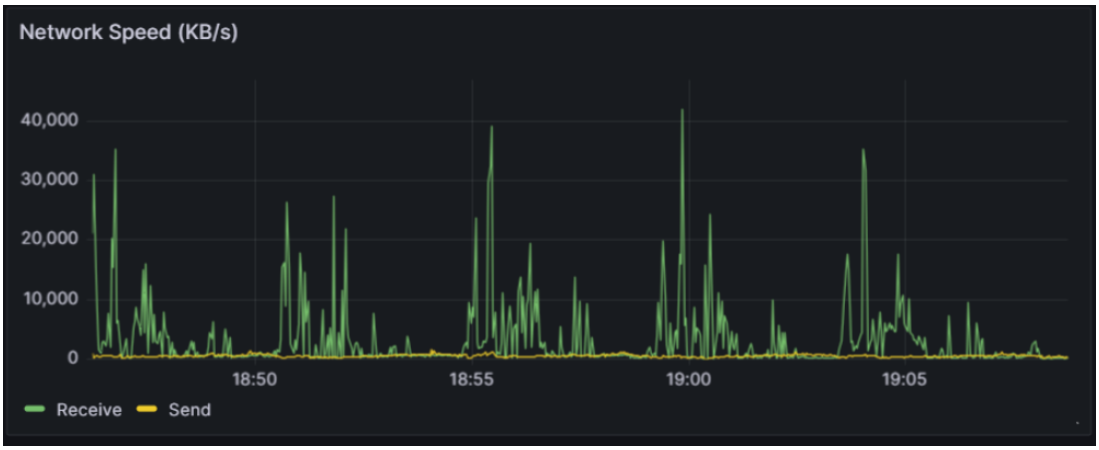


MD Core #2

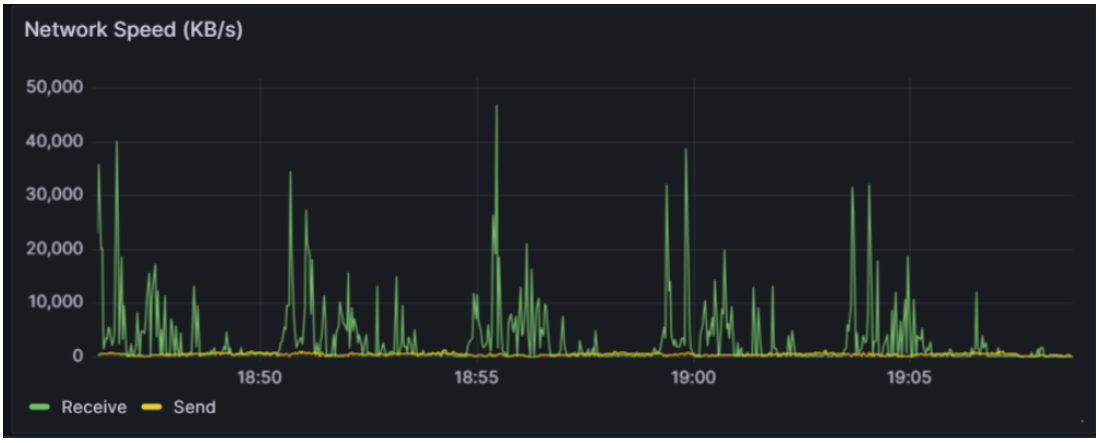


Network usage

MD Core #1



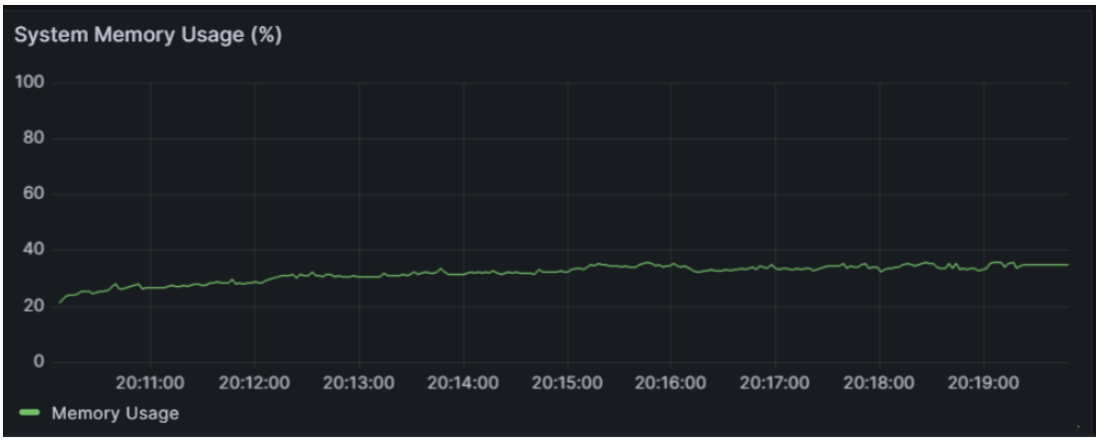
MD Core #2



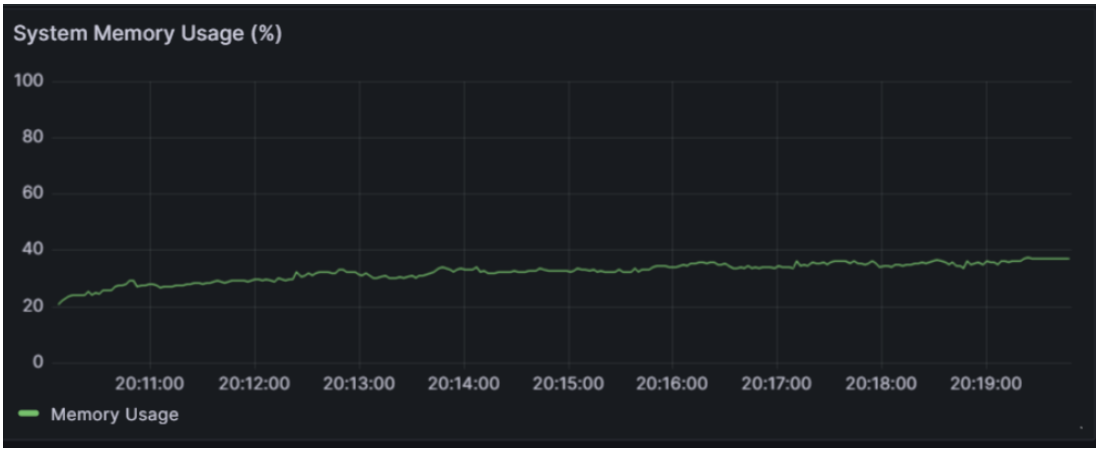
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1

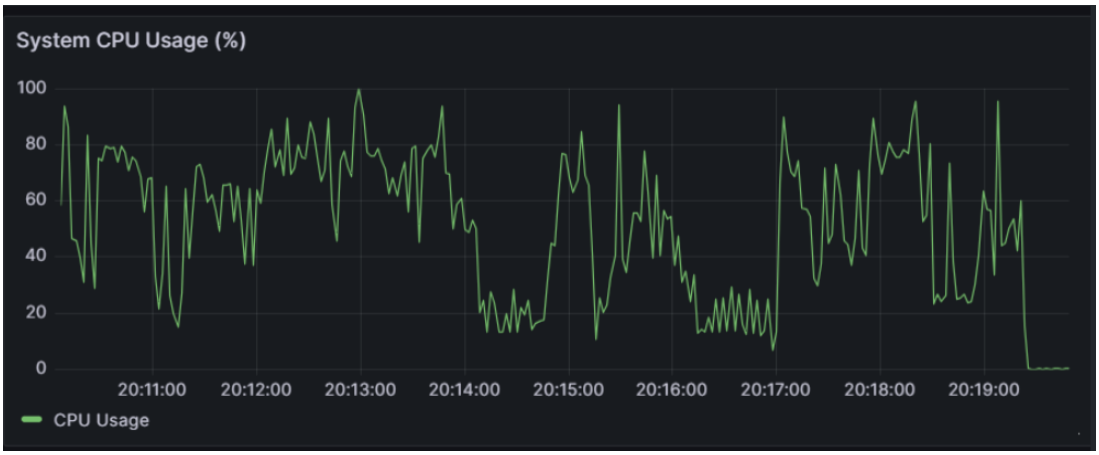


MD Core #2



CPU usage

MD Core #1

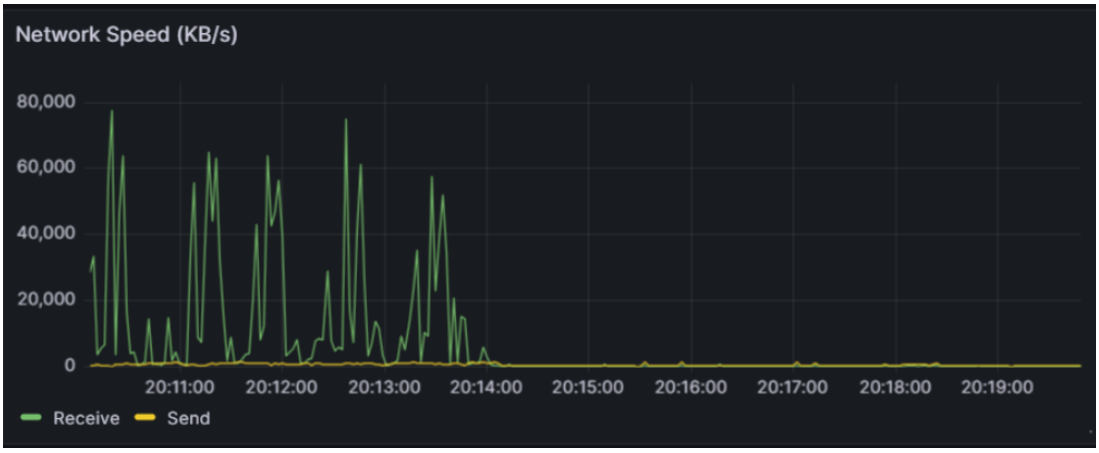


MD Core #2

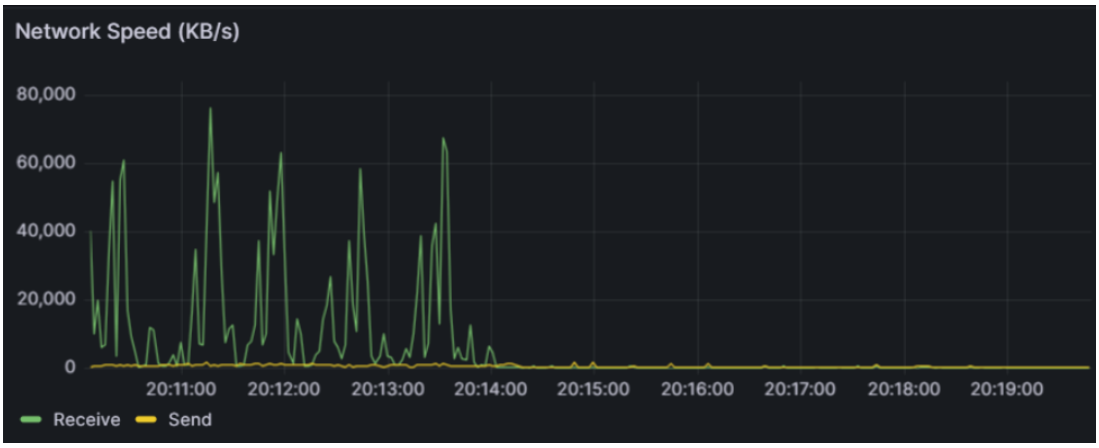


Network usage

MD Core #1



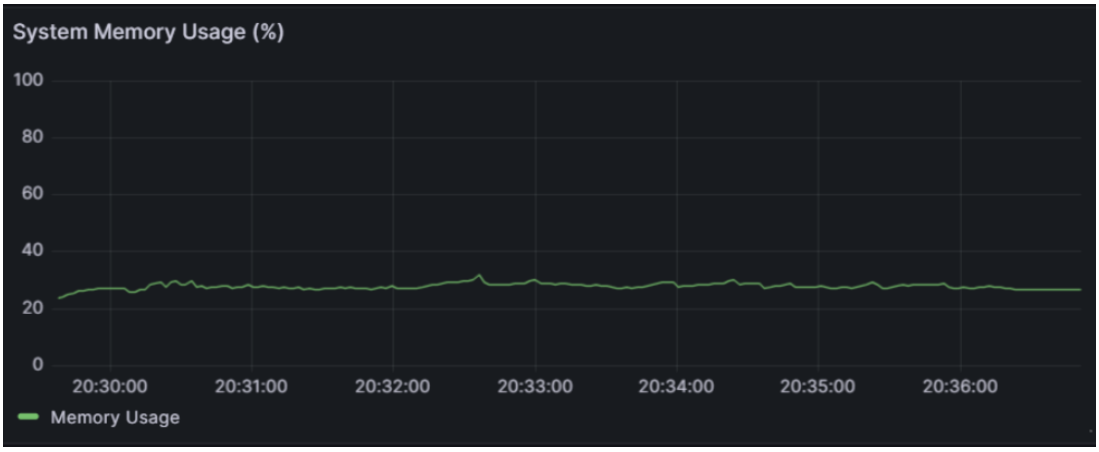
MD Core #2



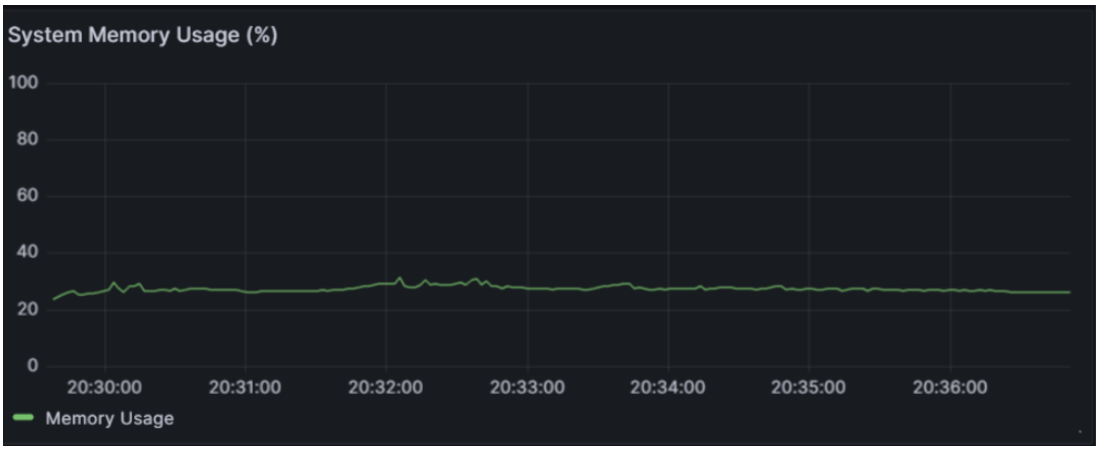
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1

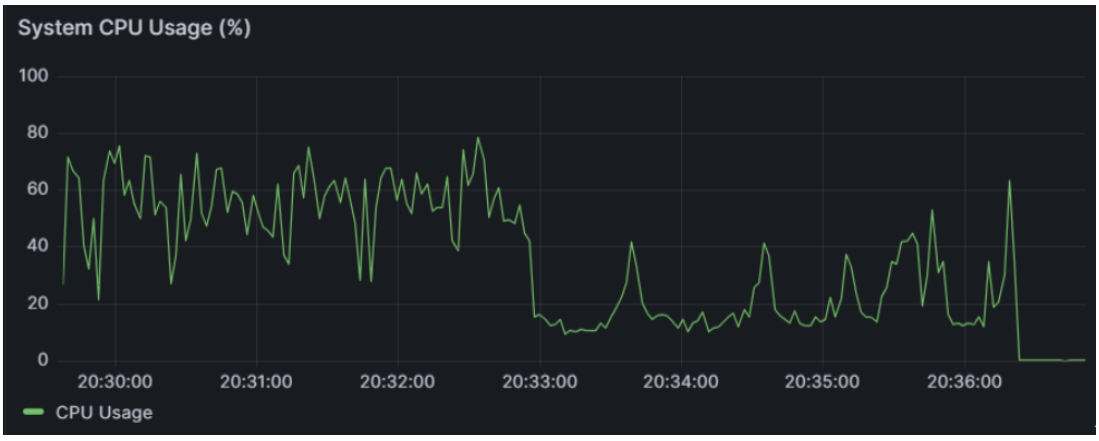


MD Core #2

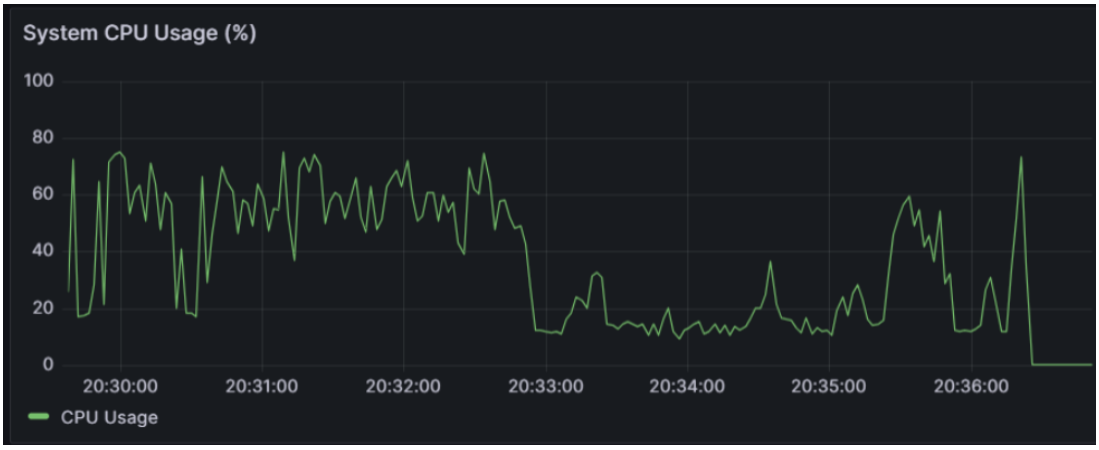


CPU usage

MD Core #1

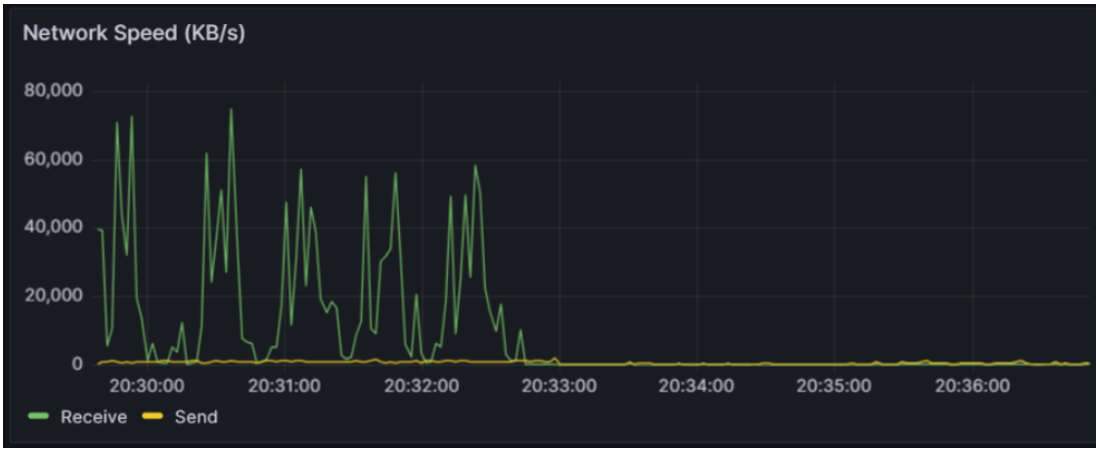


MD Core #2

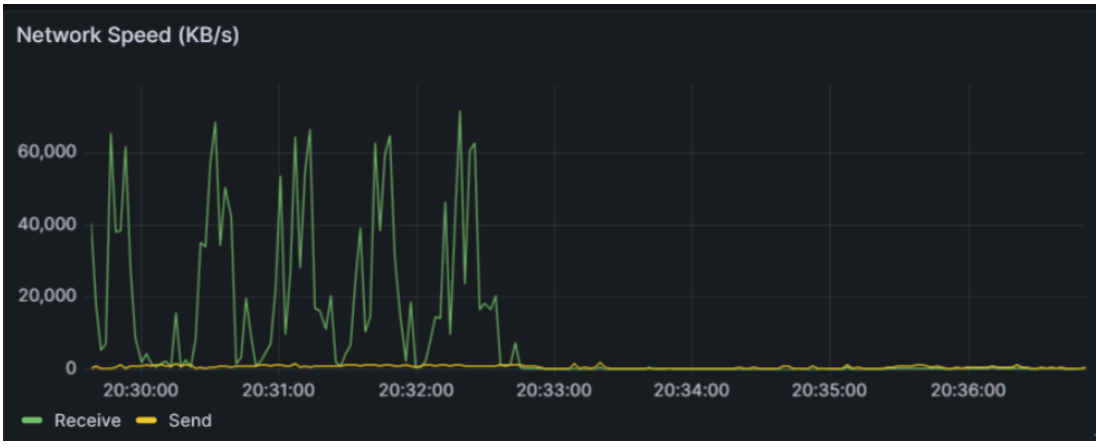


Network usage

MD Core #1



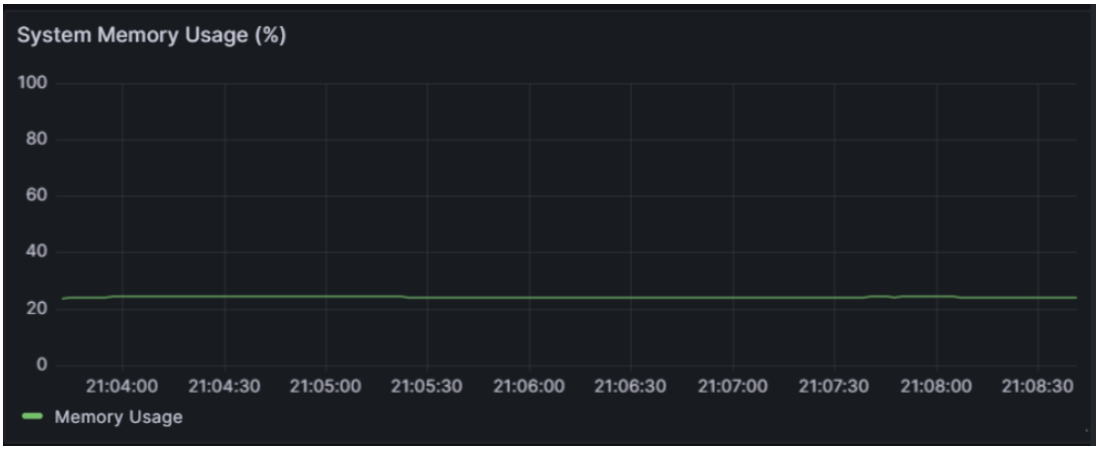
MD Core #2



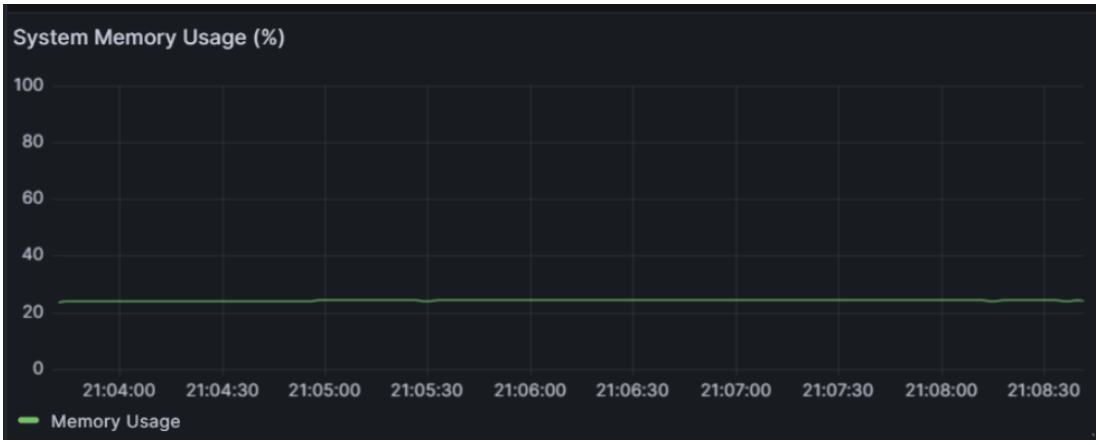
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

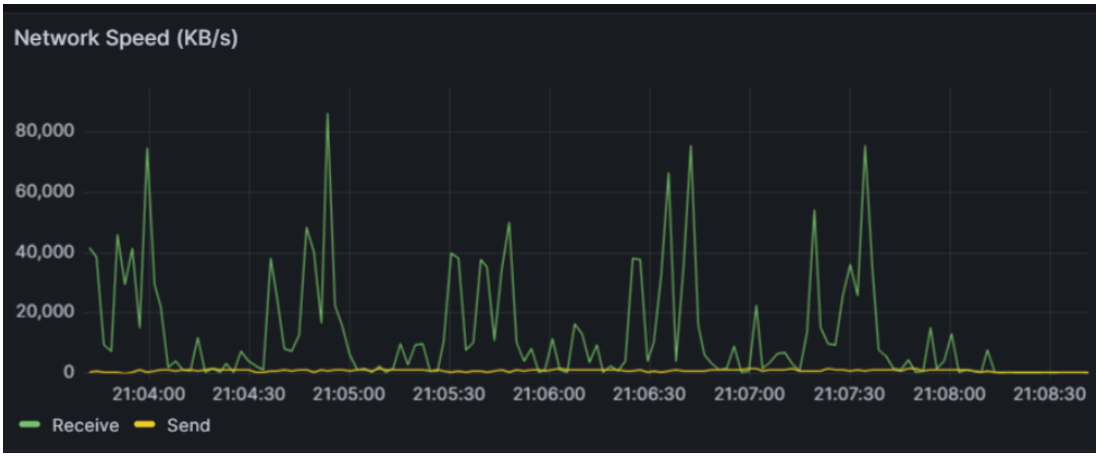


MD Core #2

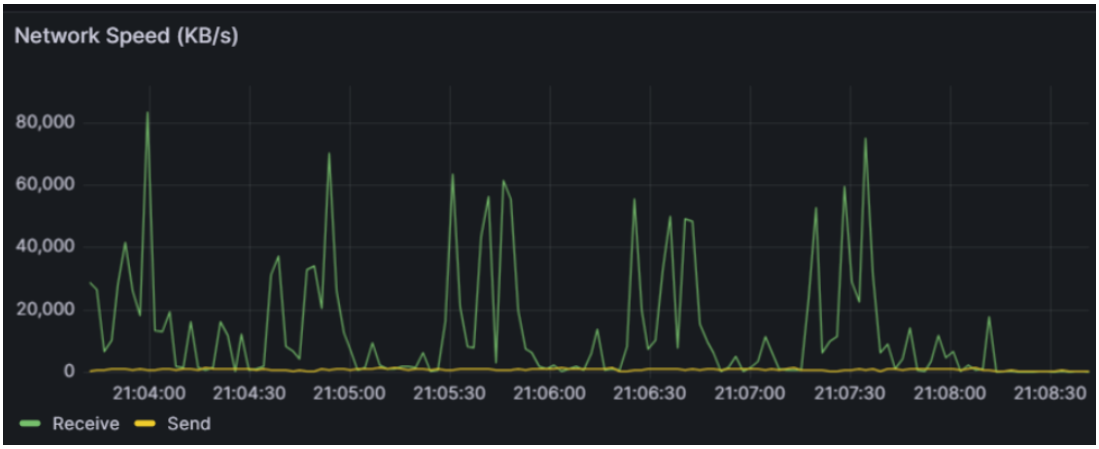


Network usage

MD Core #1



MD Core #2

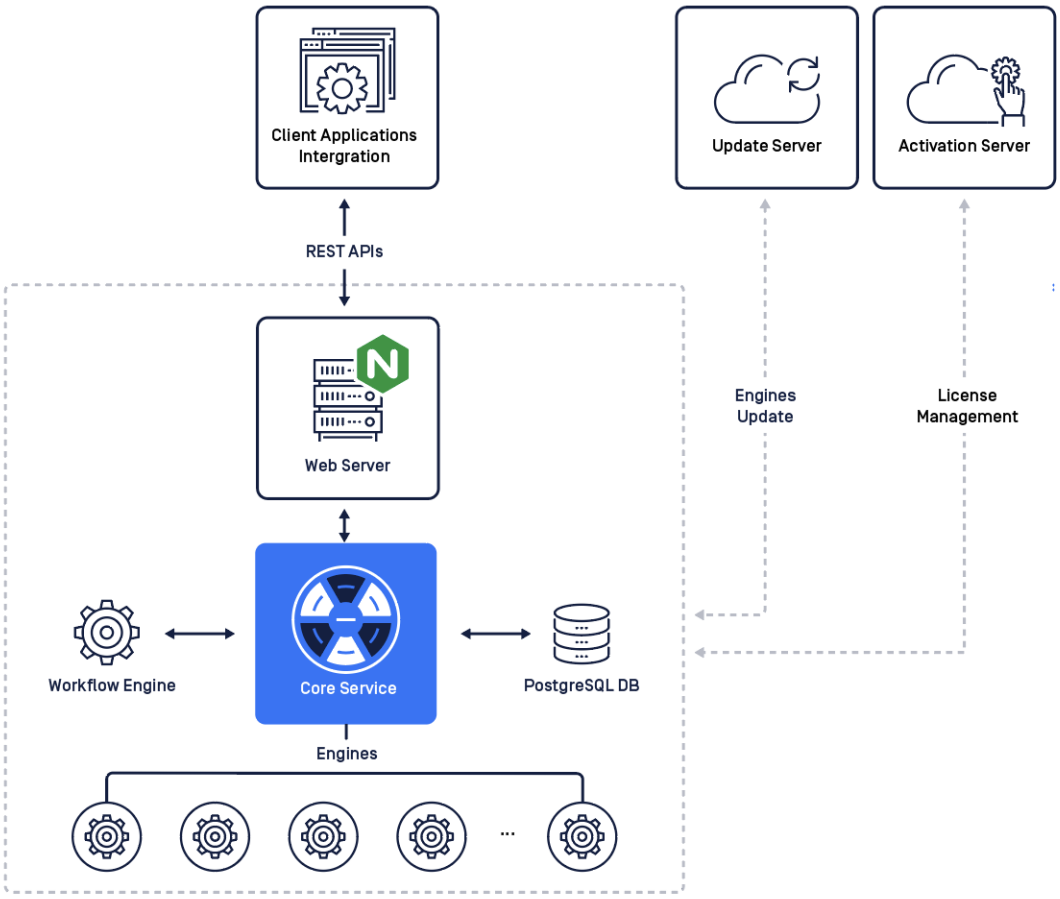

Network Speed (KB/s)
80,000
60,000
40,000
20,000
0
21:04:00
21:04:30
21:05:00
21:05:30
21:06:00
21:06:30
21:07:00
21:07:30
21:08:00
21:08:30
Receive
Send

MetaDefender Core Documentation

The users can consult this web page or, alternatively, they can download the manual in pdf format from the link below:

MetaDefender Core manual (SHA256:).

Recommended System Configuration



MetaDefender Core supports a wide range of deployment options, including air-gapped environments, providing the flexibility to choose the environment that best suits customers' requirements and infrastructure.

- Physical server.
- Virtualization platforms like VMware, Hyper-V and XenServer.
- Cloud Deployment.
- Container Deployment.

> **Note**
> Before installing MetaDefender Core, please refer to the **recommended minimum system configuration** listed below. The requirements are based on a standardized customer deployment and may require more or less resources to achieve the level of performance desired in your environment. Please note that the server specifications are built to allow a high volume daily processing. Please refer to Performance and Load Estimation for more details.

The recommendations below are for MetaDefender Core, API usage only. For any other use cases, please consult the user guide of the licensed products for accurate recommendations.

For certain use cases these might be adjusted and customized on specific needs and SLAs. We highly recommend to engage our ProServ team to assist in fine tuning MetaDefender and get the maximum performance out of your systems.

Microsoft Windows Deployments

[Windows] Supported Operating Systems

Only support 64-bit platforms.

- Windows 10, Windows 11 23H2, 11 IoT (22H2) only for End-User systems (e.g. Kiosk deployments)
  - Based on Microsoft License Terms, you're not allowed to use Windows Desktop as a server, therefore, MetaDefender Core can't be used on these systems for server use cases.
- Windows Server 2016, 2019, 2022

> **Disclaimer**
> End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

> **Plan for legacy OS support**
> OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:
>
> - Ubuntu 20.04 after **April 2025**.
> - Amazon Linux 2 after **June 2025**.
> - Windows 10 after **October 2025**.
>
> If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

[Windows] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 8	8	8 GB	16 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 12	16	16 GB	24 GB + 50 GB + 100 GB
MetaDefender Core 16	16	16 GB	32 GB + 50 GB + 100 GB
MetaDefender Core 20	32	16 GB	40 GB + 50 GB + 100 GB
MetaDefender Core MAX	32	32 GB	120 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core is installed.
- For a better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.
- SBOM processing requires more disk space, please allow at least 15GB free disk for the engine to work properly.

[Windows] Third Party Dependencies

- .NET framework 4.5 or above
- Microsoft Visual C++ Redistributable for Visual Studio 2015-2022 (version 14.38 or higher); x64 version.
- PostgreSQL: Only applicable when using a pre-installed PostgreSQL server running remotely.
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Some engines also have dependencies as described below:

Engine Name	Dependency
Deep CDR	Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 5.8.0 or above)
ESET	MetaDefender Core v4 temporary directory should have more than 200MB free disk space
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
Proactive DLP	• Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 2.0 or above) • .NET 6 dependencies • CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
Scrutiny	Microsoft Visual C++ 2015 - 2019 Redistributable x86 and x64
Systweak	.NET runtime 6.0, 7.0 or 8.0
Bkav Pro	Microsoft Visual C++ 2015 - 2022 Redistributable x64

[Windows] Installation Details

MetaDefender Core on Windows uses "C:\Program Files\OPSWAT" folder for storing resources or the installation directory.

MetaDefender will use its resources folder to store temp files as part of the analysis. **It's recommended to exclude this folder from real-time protection monitoring**. Knowledge base:

- How to set up exclusions on anti-malware software to prevent disruption
- Mitigating scan failure triggered by real-time protection service
- Can local AVs interrupt ongoing scans?

Linux Deployments

[Linux] Supported Operating Systems

Only support 64-bit platforms.

- Red Hat Enterprise Linux 8, 9
- Rocky Linux 9
- Debian 11, 12
- Ubuntu 20.04, 22.04
- Amazon Linux 2 is supported via Amazon Machine Image (AMI)

Plan for legacy OS support

OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:

- Ubuntu 20.04 after **April 2025**.
- Amazon Linux 2 after **June 2025**.
- Windows 10 after **October 2025**.

If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

For Hardended Linux OS

Some AV engines could be failed to initialize and run on MetaDefender Core (e.g. ESET, Kaspersky) due to execution permission required by them on `/var/tmp/ometascan` folder.

Please follow steps at Why have the ESET and Kaspersky scan engines failed to initialize on the hardened Linux OS? for remediation details.

Disclaimer

End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

[Linux] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 5	4	4 GB	10 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 10	8	8 GB	20 GB + 50 GB + 100 GB
MetaDefender Core MAX	16	16 GB	40 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core resides on.
- For better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.

[Linux] Third Party Dependencies

Dependencies list:

- openssl
- grep
- lib32stdc++6 (>= 4.5)
- libc6-i386 (>= 2.10)
- procps
- zlib1g
- libcurl3 (>= 7.19.7)
- libcurl4
- postgresql:
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Not all the above dependencies will need to be installed, it is dependent on different Unix distro and version

Some engines also have dependencies as described below:

Engine name	Dependency
Archive engine	• To extract password protected MS Office files, you need to install .NET 6 dependencies, libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • Rocky Linux 9.x/Red Hat 9.x: libnsl
Deep CDR	• Rocky Linux 9.x: libgomp, ncurses-compat-libs, libnsl • CentOS: libgomp, ncurses-compat-libs • Red Hat 8.x: libgomp, ncurses-compat-libs, libnsl • Red Hat 9.x: libgomp, ncurses-compat-libs, libnsl, compat-openssl11 • Ubuntu/Debian: libgomp1, libncurses5 • libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • .NET 6 dependencies
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
FileType engine	Rocky Linux 9.x/Red Hat 9.x: libnsl
Proactive DLP	libgdiplus, mscorefonts .NET 6 dependencies CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
RocketCyber	libgomp

Note: Some dependencies may need to be installed from external repositories, not from the OS default repositories.

[Linux] Installation Details

MetaDefender Core default installation path is using /var folder for storing resources:

- `/var/lib/ometascan` : installation folder with all its resources (database, DLP processed / CDR sanitized / quarantined file storage, engine package and definition updates).
- `/var/log/ometascan` : application logs.
- `/var/run/ometascan` : runtime folder for Core related processes.
- `/var/lib/ometascan` : all successful deployed engines data files.
- `/usr/lib/ometascan` : all shared libraries required by MetaDefender Core, bundled PostgreSQL server related binaries and libraries.
- `/var/tmp/ometascan/resources` : all temporary processing files for MetaDefender Core to process.

Storage Usage Information

Based on the configuration, MetaDefender Core will need additional disk space to store analysis/result data, quarantined files and temporary files:

	Data Storage
Analysis Reports (Processing History)	Approximately 4.5GB is required for every 1M analysis reports stored. • 100 GB is recommended as a baseline. Database storage requirements are affected by Data Retention settings, engine settings and scanning load.
Quarantined Files	Depends on the customers' dataset and configuration
Sanitized Files	Depends on the customers' dataset and configuration
Temporary processing files	Depends on the customers' dataset and configuration • A 1GB non-archive file requires 1GB free disk space at least. • A 1GB archive file requires (1 * 2)GB free disk space at least. • 50GB is a baseline and should be adjusted based on observation of the solution within the environment, with consideration to peak times and overall file makeup (e.g. complexity, size, archives).
Note	Options are available to control how large and/or complex files are handled to limit the amount of system resources required. See Workflow Configuration Template for more information.

Custom Engines

The recommendations above are specific for MetaDefender pre-packaged bundles.

However for additional Custom Engines, please review the Knowledge base to review additional requirements (if any) for the selected engine.

Browser Requirements for the MetaDefender Core Management Console

One of the following browsers is suggested to view the MetaDefender Core Management Console:

- Microsoft Edge
- Chrome
- Firefox
- Safari

Chrome, Firefox, Safari and Microsoft Edge browsers are tested with the latest available version at the time of release.

Network Bandwidth

In online environment where MetaDefender Core is configured to download engine update packages directly from OPSWAT update server source, then using gigabit network speed (125 MB/sec) is highly recommended to avoid potential network related issue while downloading files, or at least 100 Mbps (12.5 MB/sec) network speed.

Allowlisting URLs

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts':

- https://activation.dl.opswat.com - for license online activation
- https://update.dl.opswat.com - for engines and databases download

Note: *IP address-based allowlisting on your firewall might fail after some time since OPSWAT uses CDN (AWS Cloudfront) to faster delivery updates over the world, and IP address of edge servers might change over time.*

Depending on your location, connections with OPSWAT cloud-based servers be forwarded to specific AWS Cloudfront's edge locations, each is assigned with its own unique IP address ranges. You might also need to allow correct AWS Cloudfront's corresponding IP address ranges by referring to https://ip-ranges.amazonaws.com/ip-ranges.json

Warning

Even the OPSWAT update servers host updates for all of the available engines we support, sometimes custom engines might try to connect to their own cloud for updates, but this can be disabled in firewall and they will be updated just from OPSWAT cloud servers instead.

Differences Between MetaDefender Core v4 and v5

We are very excited to inform you about the release of version 5.0.0 of our MetaDefender Core!

The new version is designed to enhance the product performance, user experience, as well as to provide more flexibility, security, and scalability for our customers.

At the same time, we are also committed to keeping backward compatibility for whom running MetaDefender Core v4, and avoid the need for updating your client integration to move forward with MetaDefender Core v5.

v4 REST API Integration Supported
All public REST API endpoints available on MetaDefender Core v4 will be continously supported on v5.

v4 Product Major Features Supported
All product major features available on MetaDefender Core 4.21.2 (the latest one on v4) will be continously supported on v5.

Seamless Product Upgrade
Upgrading from MetaDefender Core v4 to v5 will be supported and covered with data migration without additional manual steps from users.

Licensing information will be also migrated.

Below are some highlights of the first v5 release, MetaDefender Core 5.0.0:

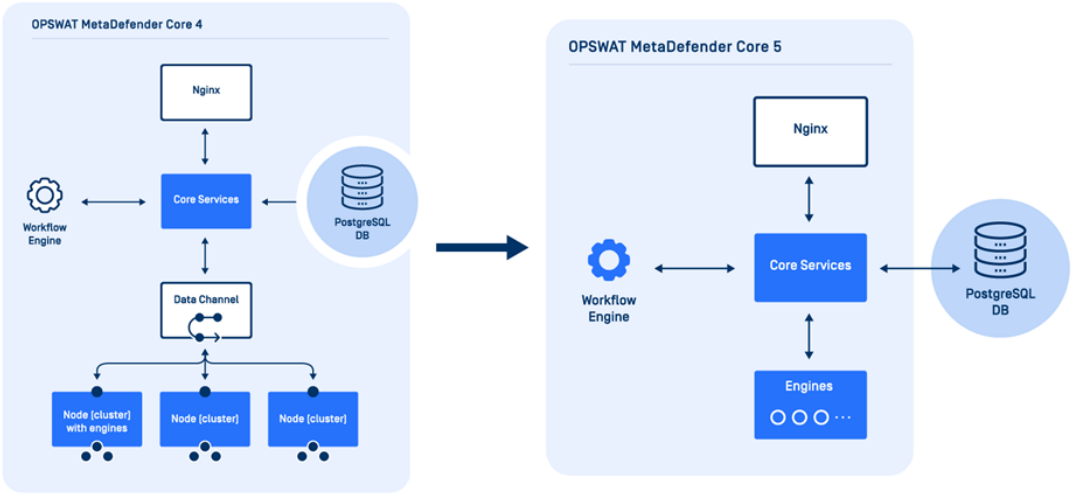
New product architecture

This version introduces a simpler product architecture that speeds up integration and automation, file processing flow, and immensely enhance the product performance and efficiency.

Earlier versions utilized MetaDefender Node concept, designed to support scaling of the scanning infrastructure by distributing scan requests among several different processing Nodes. Product licensing was based on number of Node instances. The benefit of having such a distributed infrastructure is that based on Node loads, Metadefender Core server can always choose the most appropriate Node to assign a new scan task to. However, having nodes in the v4 architecture caused more complexity and possible latency in the processing flow and entailed maintenance cost.

In this release, we retired multi-Node deployment and restructured MetaDefender Core's backend. This alteration significantly improves file processing speed and product efficiency, as well as eliminates extra cost for maintaining node services. Users now can simply install additional MetaDefender Core instances for scalability requirement,

as well as configure their own load balancer to coordinate tasks among multiple MetaDefender Core instances. Accordingly, product licensing is now based on number of the product instances.



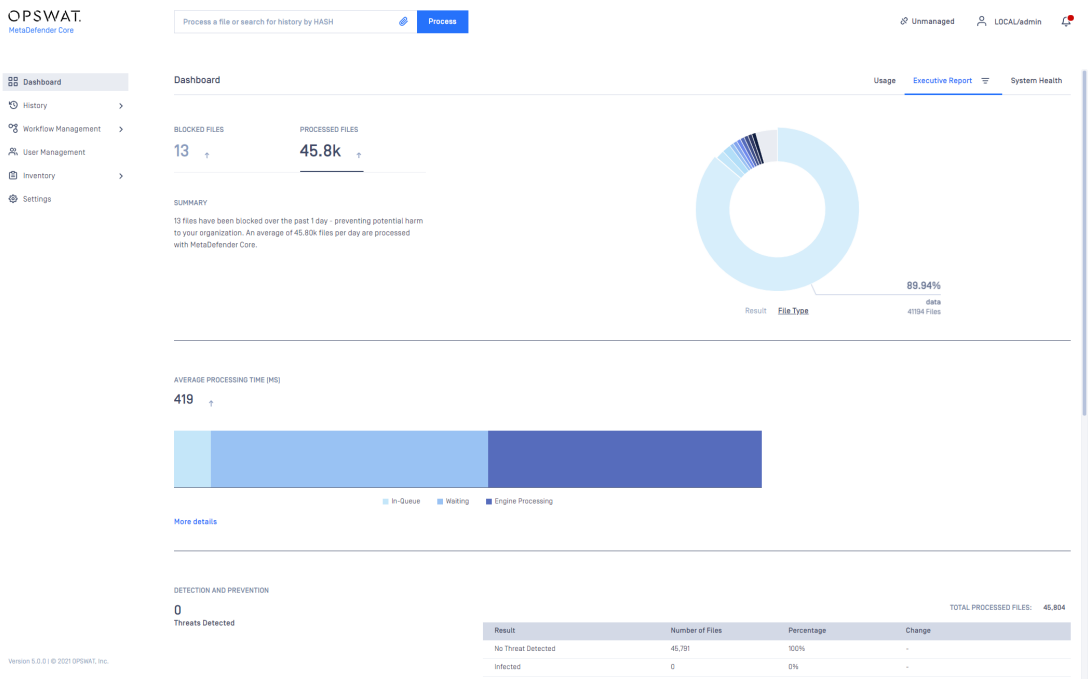
Please be informed that

- All existing administrative APIs related to "Node" on v4 will be kept as it is on v5 to ensure backward compatibility with existing client integrations.
- Applicable settings defined in Windows registry (Windows) or ometascan-node.conf config file (Linux) for Node will be migrated to Core. Details: MetaDefender Configuration

Brand-new product user interface

The second notable update in this version is a brand-new UI for the management console, which was designed with the focus on the quality and thoughtfulness of the user experience. The new administrative console UI provides an advanced set of Core features and observable processing status.

Comprehensive executive dashboard reporting offers dynamic, real-time and actionable threat insights. It also provides visibility of your security posture allowing you to paint a picture of security that aligns with the overall business. The report includes easy-to-read charts, graphs, and statistic explanation that help your security professionals reduce clutter, gather information quicker, spot threat activity early and act swiftly.



We're continuing to improve our product UI to provide users an excellent experience with the application and overall usability. Please expect more powerful UI features coming in next releases.

New compression archive engine for archive sanitization

In previous version, MetaDefender Core's archive engine implemented four functions including decryption, extraction, encryption and compression that with a high volume of files it takes time to process them all. For the purpose of enhancing archive handling performance, this release comes with a separated compression archive engine, which dedicatedly processes archive sanitization (compression and encryption). It is also created to facilitate maintenance and further feature development.

Archive sanitization feature now requires Deep CDR license

As to respect sanitization practice and ensure unknown threats prevention as designed, now starting on version 5, it's mandatory to have Deep CDR licensed for archive sanitization feature enablement on MetaDefender Core workflow configurations.

Archive handing related engines changed

The legacy "Archive" engine on MetaDefender Core v4 will be changed its name to "Archive Extraction" engine, even when you are still on MetaDefender Core v4.

A brand new "compression archive engine" is to dedicatedly handle archive sanitization in v5 (archive compression), that helps relevant archive handling performance improvement, easier for maintainance and further feature development.

Notes when you are still on MetaDefender Core v4 (not upgraded yet)

The legacy "Archive" engine (now name changed to "Archive Extraction" engine) on MetaDefender Core v4.x will still cover both extraction and compression archive tasks as usual.

Even though your MetaDefender Core v4.x might now have a new "Archive Compression" engine auto downloaded, but that engine cannot be deployed (marked as "permanently failed" in relevant engine status check API). This behavior is expected, and not causing any impact to MetaDefender Core processing flow, or its performance though.

- Just in case your client integration has a check for engine status before triggering scan request, then you might want to disable "Archive Compression" engine status on MetaDefender Core v4 using REST API. Detailed instruction: Archive Compression Engine Permanently Failed on MetaDefender Core v4
- When you later upgrade to MetaDefender Core v5 from v4, then you might need to manually re-enable "Archive Compression" engine to make it active and running on MetaDefender Core v5.

Notes for licensing

For offline deployment (MetaDefender Core is not connected to the Internet directly for online update), you will need to re-activate again on OPSWAT portal to retrieve latest engine list in .yml file including "Archive Compression" engine if you are currently licensed with Deep CDR.

Meanwhile, for online deployment, this manual procedure is not required since MetaDefender Core will cover that itself.

Import & export configurations

Not only workflow settings like in previous versions, but also all other general settings (update, security, data retention, server, hashes to skip) and most importantly, user settings, will be able to export and import on MetaDefender Core version 5.

Learn more at Import/Export configuration

Backward compatibility supported

The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

Windows application event logging disabled by default

Applicable only to Windows OS based MetaDefender Core.

Help avoid duplicated logging with separate file option, and improve overall processing performance. Of course, when being still valid to you, you are still supported to enable it back.

CSRF token header renamed

CSRF token originally created for security strengthening and being used by MetaDefender Core management console UI since version 4.21.0 via header `oms_csrf_token`

To get it compliant with some load balancers which do not accept underscore in header name by default, and save you from additional load balancer configuration step, we have now changed the header name to `oms-csrf-token`

This change does not impact to any REST API based client integration.

More comprehensive and accurate error messages on REST APIs

Impacting on following REST API endpoints:

- `PUT /admin/config/session`
- `PUT /admin/config/update`
- `PUT /admin/config/scan`
- `PUT /admin/config/skip`
- `PUT /admin/config/auditlog`
- `PUT /admin/config/webhook`
- `PUT /admin/config/rule`

Upload time and queue time logic updated in JSON scan response

Impacting on following REST API endpoints:

- `GET /file/{data_id}`
- `GET /hash/{md5/sha1/sha256}`
- `GET /archive/{data_id}`
- `POST /file/{async}`

Added new field:

- `receive_data_timestamp`: The timestamp when upload progress started (first byte received) (in milliseconds)
- `upload_time`: Total time elapsed for upload process (in milliseconds)

Modified existing field (new logic):

> **Backward compatibility supported**
> The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

- `upload_timestamp`: The timestamp when upload progress finished (all bytes received) (in milliseconds)
- `queue_time`: Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue time = start_time - upload_ timestamp) (in milliseconds)

Release version 5.0.0 also comes with variety of outstanding features and improvements.

For detailed information about our latest release please view our Release notes

Installation

Starting with MetaDefender Core 4.20.0, the centralized/shared database deployment model is supported (all MetaDefender Core instances share the same database), along with traditional standalone database mode (every MetaDefender Core instance has its own database to manage).

- Standalone mode
- Shared Database Mode

Installation

1. Download the package of your choice from My OPSWAT
2. Install the package on your computer via the Command Line or via the Wizard
3. Open a web browser and point to http://<server name or IP>:<port>
    - The default port is **8008**
    - In case of problem check Inaccessible Management Console page
4. Complete the required steps of the basic configuration wizard
5. You must activate this deployment to use its features

Installation notes

- If the MetaDefender Core package **dependencies are not installed** on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.
- During installation the databases might need to be upgraded. This could take noticeable time depending on database size (eg.: length of scan history).

Standalone mode

Wizard Installation

The Install Wizard is only for the Windows installer (.msi file).

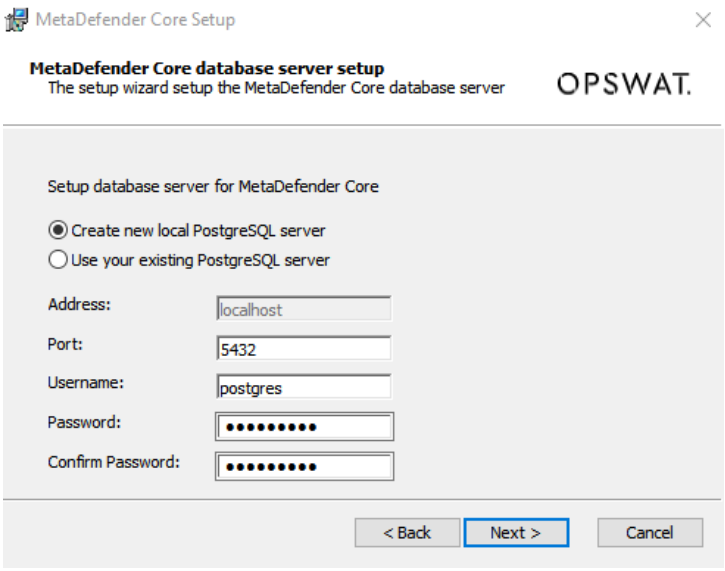
Since MetaDefender Core 4.19.0, we have switched to using PostgreSQL DBMS which needs a dedicated database server. MetaDefender Core supports two modes to setup PostgreSQL server:

Local (Bundle) PostgreSQL server

MetaDefender Core will install a new PostgreSQL server locally in the same box with the product.

> **PostgreSQL user credentials**
> Unicode characters are NOT supported for "Username" and "Password"
>
> "Password" does not accept # (hash) as starting character (e.g. #ABC)

> **The ignition file**
> If the ignition file exist in the designated location, then MetaDefender Core will use the details defined in the ignition file, rather than what is passed via the installation wizard.
>
> More info about the ignition file based installation: Command Line Installation

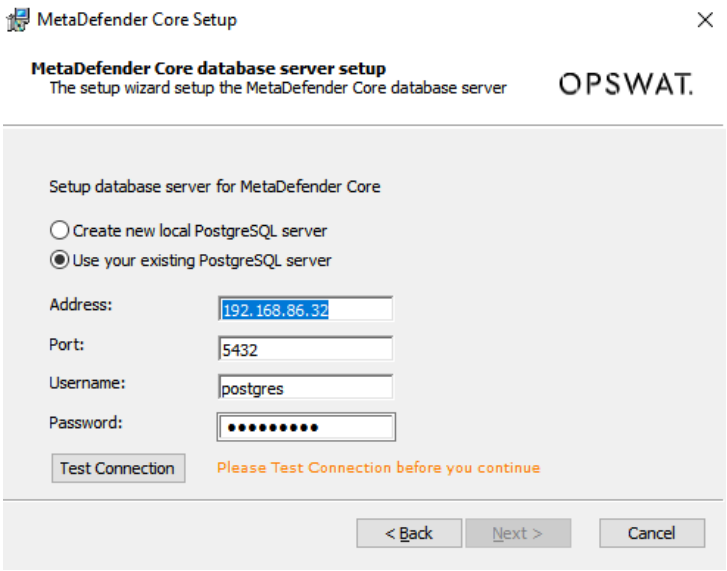


Remote (Existing) PostgreSQL server

MetaDefender Core will connect to remote pre-installed PostgreSQL running remotely.

> **PostgreSQL connection configuration**
> "Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

> **PostgreSQL user privilege requirements**
> MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.
>
> Details: PostgreSQL User Privilege Requirements



Command Line Installation

Ignition File

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Make sure to create the ignition file, if it doesn't exists before you go ahead and install MetaDefender Core via command line.

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

MetaDefender Core supports two modes to setup PostgreSQL server:

1) Local PostgreSQL server

- MetaDefender Core will install a new PostgreSQL server locally, next to the product.
  - A sample iginition file for PostgreSQL server information (***Only non-Unicode characters supported for "user" and "password"__***):

ometascan.conf bash

```
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
```

2) Remote PostgreSQL server

> **Info**
>
> **PostgreSQL version 12 (at least 12.3) or version 14 is required.**
>
> According to PostgreSQL notice, version 12 will stop receiving fixes on November 14, 2024, so we suggest that you use version 14.

MetaDefender Core will connect to a pre-installed PostgreSQL server running remotely and will setup its database.

Make sure to follow steps at Open Connection On PostgreSQL Server first, so that your remote existing PostgreSQL server to accept upcoming connection from MetaDefender Core.

A sample ignition file for PostgreSQL server information:

ometascan.conf bash

```
[dbserver]
type=remote
host=192.168.86.32
port=5432
user=existingserver_admin_user
password=existing_server_pass
```

After installing MetaDefender Core successfully, you may want to remove all credentials info created for [dbserver] section in ignition file for security reason.

> **Warning**
> If the MetaDefender Core package dependencies are not installed on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.

Debian package (.deb)

bash

```
sudo dpkg -i <filename> || sudo apt-get install -f
```

On Red Hat Enterprise 7.x Linux / CentOS package (.rpm)

bash

```
sudo yum install <filename>
```

For systems which enabled GPD check flag:

bash

```
sudo yum install --nogpgcheck <filename>
```

Red Hat Enterprise Linux 8.x package (.rpm)

Install the latest PostgreSQL repository RPM:

> **Note**
> Due to the shorter support cycle on Red Hat Enterprise Linux, all supported versions of PostgreSQL are not available on this platform. The PostgreSQL Yum Repository will integrate with the normal systems and patch management, and provide automatic updates for all supported versions of PostgreSQL throughout the supported lifetime of PostgreSQL. Reference: https://www.postgresql.org/download/linux/redhat/

bash

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

In case installing a remote PostgreSQL on either the same MetaDefender Core server or another server, you need to install the following additional package. PostgreSQL version 12.3 or newer is recommended.

bash

```
sudo dnf install -y postgresql12-contrib
```

Install MetaDefender Core:

bash

```
sudo dnf install -y <filename>
```

Windows package (.msi)

On Windows systems it is possible to install the product by running the corresponding .msi file.

From command line interface it is also possible to install the product by executing

bash

```
msiexec /qn /i <msi file name> <option key>=<option value>
```

where the possible keys and their default values are the following:

Key	Default Value	Description
INSTALLFOLDER	\Program Files\OPSWAT\MetaDefender Core	Customize installation folder for product Example: `INSTALLFOLDER="D:\Products"`
RESTADDRESS	*	REST interface binding IPv4 or IPv6 address ('*' means that service listens on all IPv4 and IPv6 interfaces)
RESTPORT	8008	REST interface binding port
REPORT_ENGINE_ISSUE	true	Enable reporting of engine issue count. (possible values: "true" or "false").

Note

Reporting of engine issue count

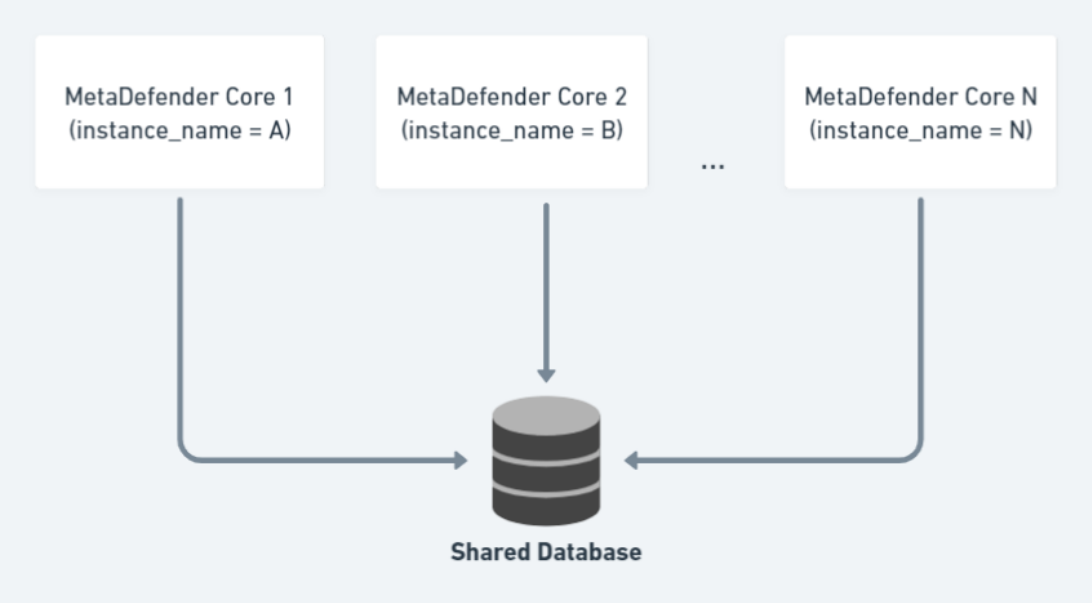
If reporting of engine issue count is enabled, MetaDefender Core server will send only the **number** of initialization errors and **number** of unexpected stops for the specific db/engine version. This information is sent over a HTTPS channel when the product downloads the latest package descriptors. This information is used for early detection of any specific 3rd party engine quality issues.

For details on using msiexec please consult Windows installer documentation.

Shared Database Mode

Starting with MetaDefender 4.20.0, you can use a centralized database deployment model, where all MetaDefender Core instances can share the same external database.

That helps auto-sync all product configurations (**except Single-Sign-On** that requires unique settings for each Core), processing history, live/on-going scan result query, audit, statistics data.



This article demonstrates how to install and setup MetaDefender Core 4.20.0 or newer with shared database mode.

Important notes & current limitations:

Currently not supported

- Does not support Central Management use-case with shared DB mode.
- Does not support to switch from shared database mode back to standalone database mode.
- Does not support to change instance_name of MetaDefender Core in shared database deployment model after installation.
- Does not support upgrading older MetaDefender Core from non-persistent mode to using the shared database mode.
- Does not support upgrading older MetaDefender Core from standalone mode to using the shared database mode.
- Does not support to upgrade multiple MetaDefender Core instances at once in shared database mode
    - Stop all Core instances' scanning service before performing the upgrade.
    - Upgrade each Core server one by one instead.

- For each Core, when the upgrade finished, then you can start using that Core for scanning without waiting for the other Core instances' upgrade to be done.

> **Disclaimer**
> MetaDefender Core does not support mode switching.

Requirements for shared database setup:

- The central remote PostgreSQL server (for shared database) could be utilized with a built-in (native) PostgreSQL database server installed with one of the MetaDefender Core instances in the network. Or using a dedicated existing PostgreSQL database server (see detailed requirements for PostgreSQL version at Recommended System Configuration).
- Before setting up, make sure to follow steps at Open Connection On PostgreSQL Server so that all MetaDefender Core instances could connect to the central remote PostgreSQL server.
- MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core, details: PostgreSQL User Privilege Requirements
- All MetaDefender Core instances must be installed on the same Operating System (Windows / Linux).
- The first MetaDefender Core instance connects to the shared database must be ready first (license activated, users/groups/roles created) before setting up other MetaDefender Core instances to connect and use that shared database.

Engine update:

- If engine updates is configured to be via Folder source, all MetaDefender Core instances must be able to access to the configured update folder, and the option DELETE FILES AFTER IMPORT will be unchecked and prevented for modification.

Authentication:

- Single-Sign-On (SSO) authentication settings (if using) cannot be shared among MetaDefender Core instances because SSO requires an unique login URL for every different Core.
- If an user is locked for login on a certain MetaDefender Core instance(e.g. exceeded the number of failed login times), that user is not prevented for logging in on any other instances.
- If an user is locked for sign-in on a certain MetaDefender Core instance (e.g. exceeded the number of failed login times), if the admin releases lockout on that user on that MetaDefender Core instance, that user is still not able to login on other MetaDefender Core instances.

Sanitized / DLP / Quarantined location changed after MetaDefender Core install:

Windows:

Shared storage path will be stored in 3 strings under key global in registry: quarantinepath, dlppath, sanitizepath as below

(Default)	REG_SZ	(value not set)
dbmode	REG_SZ	4
dlppath	REG_SZ	\\10.40.50.100\test
instance_name	REG_SZ	481rvj
quarantinepath	REG_SZ	\\10.40.50.100\test
restaddress	REG_SZ	*
restport	REG_SZ	8008
sanitizepath	REG_SZ	\\10.40.50.100\test

You can set either 1, 2 or 3 different paths for the 3 settings above (means that quarantined, DLP and sanitized files will be stored in same or different storages corresponding). In case shared storage is not set in installation, values of 3 settings above will be empty.

Restart MetaDefender Core service after changed.

Linux:

Open file `/etc/ometascan/ometascan.conf` and set shared storage path for 3 keys quarantinepath, dlppath, sanitizepath under section [global]

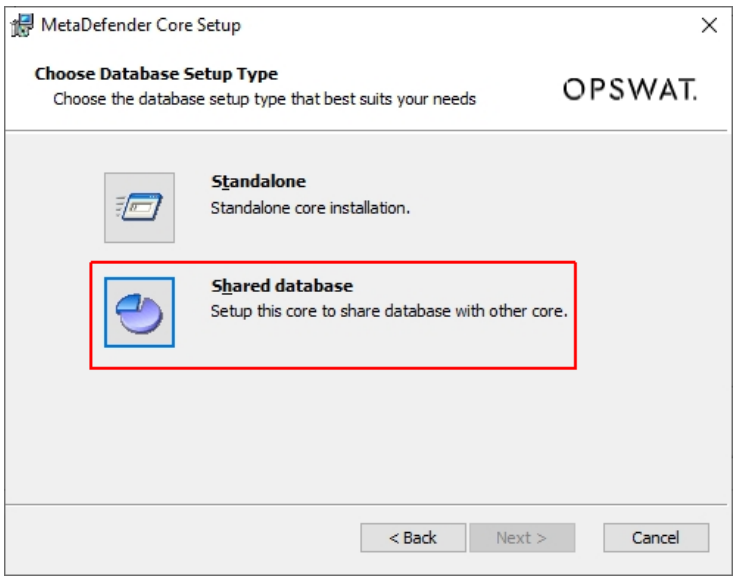
Restart ometascan service after changed.

Wizard Installation (Windows Only)

> **Info**
> Only applicable to MetaDefender Core 4.20.0 or newer using wizard on Windows OS.

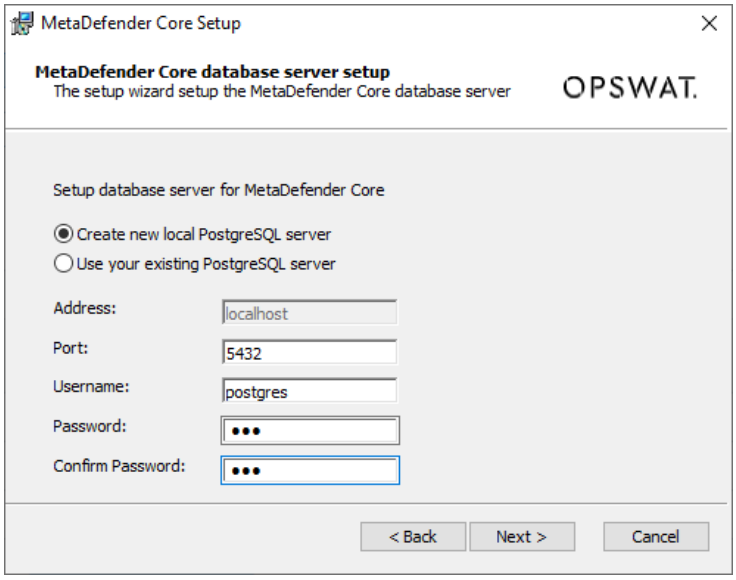
Clean install scenario

1. **Choose Shared database option:**

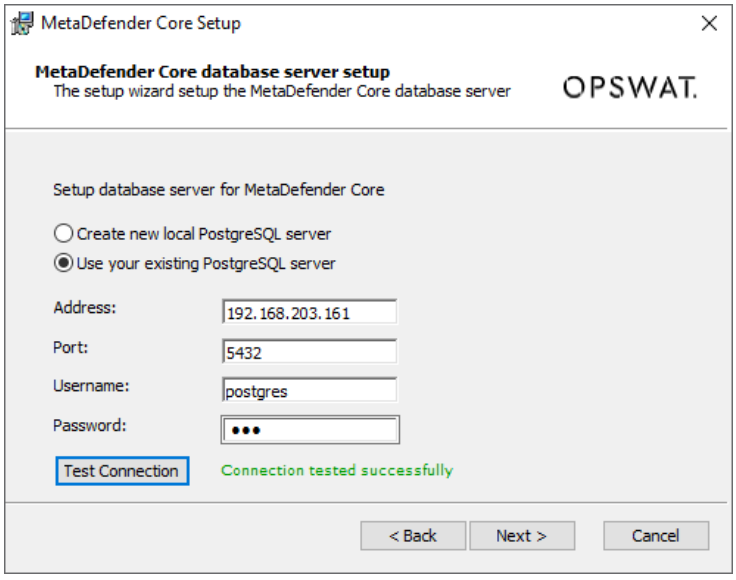


2. **Setup database server for MetaDefender Core:**

You can choose to create a native / built-in PostgreSQL server and setup the shared database on (then other MetaDefender Core instances will want to connect to that PostgreSQL server afterwards)



Or choose to connect to a pre-setup existing PostgreSQL server:



PostgreSQL connection configuration

"Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

PostgreSQL user privilege requirements

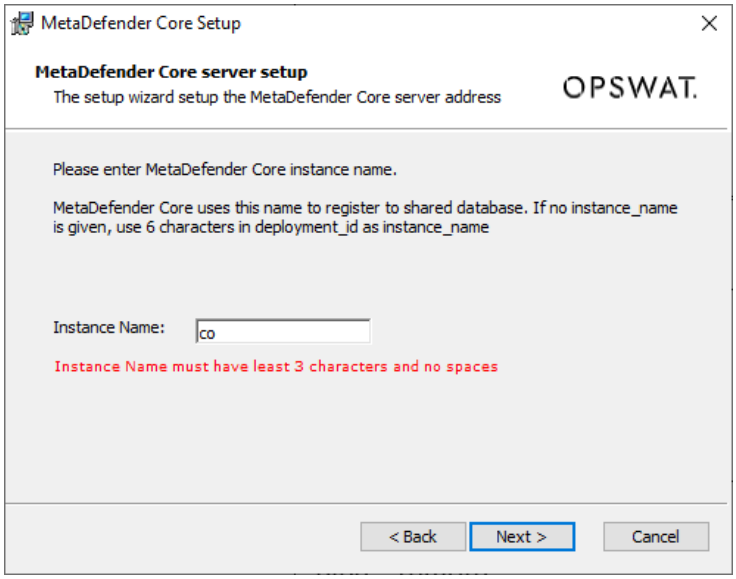
MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.

Details: PostgreSQL User Privilege Requirements

3. **Create an unique instance name for that MetaDefender Core (different from other MetaDefender Core instances):**

Instance name must contain at least 3 characters and no white space.

If you choose to leave it blank, which means it will use the default instance name with 6 characters extracted from that MetaDefender Core deployment ID.

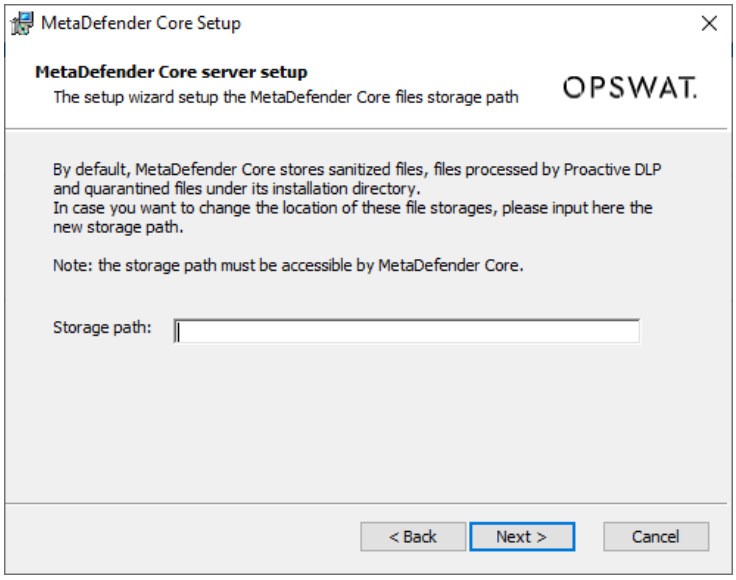


4. **Input the file storage path (sanitized, DLP, quarantined files)**

In the shared database model, all MetaDefender Core instances could use the same folder path to store all their own sanitized, DLP, quarantined files after processed.

The default location is: ***<MetaDefender Core installation folder>\data\ {sanitized|dlp|quarantine}***

If you wish to use that default location, just leave the input box blank and press Next.



Otherwise, you can specify a different location of sanitized/dlp/quarantined files. It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core.

For example:

- C:\new_folder
- \192.168.1.1\new_folder

Command Line Installation

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Clean install scenario

1. Create ignition file if not existed

- Windows: **C:\OPSWAT\ometascan.conf**
- Linux: ***/etc/opswat/ometascan.conf***

An example of ignition file:

ometascan.conf bash

```
eula=true

[global]
dbmode=4
instance_name=MetaDefenderCoreA
storage_path=D:\path\

[user]
name=admin
password=admin
email=admin@local

[dbserver]
type=remote
host=192.168.1.1
port=5432
user=postgres
password=your_own_password
```

Field	Required	Mapping to wizard UI	Possible values
global/dbmode	optional default is 1 (standalone)		1 - Standalone 2 - Central Hub deployment 3 - Non-persistent **4 - Shared**
global/instance_name	optional default is 6 characters of deployment ID		Unicode and number At least 3 characters No whitespace
global/storage_path	optional default is empty		Path to storages (DLP, quarantined, sanitized files) It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core. For example: • C:\folder • \192.168.1.1\folder

2. Run installers using command line to install

Please refer to Command Line Installation for installation command.

Deployment automation support

Note

An Ignition file could be utilized for either / both of following use-cases:

1. Pre-define PostgreSQL server information for MetaDefender Core to connect and setup database: only required on Linux installation, or Windows with command line installation. Learn more details at:
    a. Standalone Mode > Command Line Installation (applicable on version 4.19.0 or newer).
    b. Shared Database Mode > Command Line Installation (only applicable on version 4.20.0 or newer).
2. Automating to accept EULA, create a default local admin user and import MetaDefender Core configurations upon installed: Those steps could be automated using ignition file described in this page.

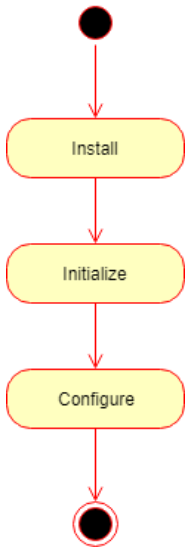
Ignition file location to create:

- **Windows**: `C:\OPSWAT\ometascan.conf`
- **Linux**: `/etc/opswat/ometascan.conf`

The product supports fully automated deployment. It means that it can be installed and configured with no human interaction.

The automated deployment can be split to three steps on a high level:

1. Installation
2. Initialization
3. Configuration



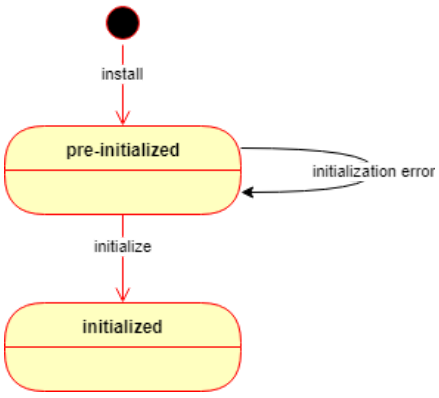
Installation

To automate the installation, install the product from the command line and provide the installation-time options as parameters to the installer. For further details see Command Line Installation

After the installation is complete, the product starts up and waits in a pre-initialized status. The product may be initialized in two ways:

1. Manually using the Wizard Setup or
2. Automatically using an *ignition file* (see below).

> **Warning**
> If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried after fixing the problem (e.g. placing the ignition file in its lookup location) and restarting the OPSWAT MetaDefender Core service.
>
> Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

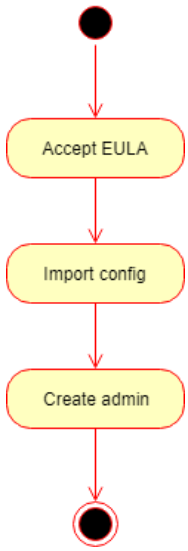


Initialization

Initialization is the process of bringing the product to an operable status.

Basically the initialization consists of the following steps:

1. Accept the End User License Agreement (EULA),
2. Import product configuration and
3. Create the first administrator user account.



Ignition file

The initialization process can be configured in a file called the *ignition file*.

> **Warning**
> In Linux, the ignition file and the folder /etc/opswat should have the permission 755

Ignition file location

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

The ignition file must be in conf format.

ometascan.conf bash

```
eula=true
[user]
name=admin
password=admin
email=admin@local
apikey=44c96cbb4e4e6c0a1d16bcb3c3da24def317
[config]
import=config_package.zip
import_password=123
[security]
session_cookie_samesite=Lax
[internal]
skip_startup_pages=true
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
private_username=my_internal_db_user
private_password=mypassword
[ocm]
ocm_url=https://my.opswat.com
reg_code=XXXXXXXX
```

Ignition file fields

The ignition file accepts the following fields:

Section	Key	Required	Accepted Values	Description
	eula	Optional (mandatory when having [user] section)	true false (default)	Whether to accept the End User License Agreement. This key must be set at the top of ignition file. This key must be set to "true" to accept the EULA. Any other value will cause the initialization to fail. When having [user] section defined, then this key must be set.
[user]		Optional		Initial administrator user account properties. The Administrator role is granted to the account.
	name	Optional	string	User name for the initial administrator user account.
	password	Optional	string	Password for the initial administrator user account. WARNING! Clear text password The password in this configuration file must be stored in its clear-text format and as so it may be visible for unauthorized parties.
	email	Optional	string	E-mail address for the initial administrator user account.
	apikey	Optional	string	API key for the initial administrator user account.
[config]		Optional		
	import	Optional	string	Path to a file in json/zip format that contains a previously exported configuration to be imported. Zip-format file contains a full configuration including user management settings, so it will overwrite [user] section.
	import_password	Optional	string	Password to decrypt .zip configuration package.
[security]		Optional		

Section	Key	Required	Accepted Values	Description
	session_cookie_samesite	Optional	Strict Lax (default) None	Declare if session cookie should be restricted to a first-party or same-site context.
[internal]		Optional		
	skip_startup_pages	Optional	true false (default)	Skip all welcome and other startup pages (e.g. upgrade result) when installing / upgrading MetaDefender, and going straight to the product index default page.
[dbserver]		Optional		
	type	Mandatory when having [dbserver] section	local remote	Where the PostgreSQL database server should physically locate. "remote" means PostgreSQL server and MetaDefender Core are not in the same machine.
	host	Mandatory when having [dbserver] section	string	IP address / domain name of the server where PostgreSQL server locates. "localhost" can be used when applicable.
	port	Mandatory when having [dbserver] section	number	Port of PostgreSQL server is listening for connections from clients (i.e. MetaDefender Core).
	user	Mandatory when having [dbserver] section	string	PostgreSQL server's user. SUPERUSER privilege is required for MetaDefender Core to setup its database and extensions for the first time. **Only non-Unicode characters supported.**

Section	Key	Required	Accepted Values	Description
	password	Mandatory when having [dbserver] section	string	PostgreSQL server's user credentials. **Only non-Unicode characters supported.**
	private_username	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
	private_password	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
[ocm]		Optional		
	ocm_url	Optional	string	URL to My OPSWAT server.
	reg_code	Optional	string	A registration code to enroll to My OPSWAT server.

Detailed initialization process

1) After the product has been started, it looks for the ignition file in the designated location.

Windows: C:\OPSWAT\ometascan.conf ** Linux: /etc/opswat/ometascan.conf

** 2) If an ignition file is found, then

2.1) It gets validated, and if it is valid, then based on the information found in the ignition file:

- The EULA is accepted,
- The configuration is imported,
- The administrator account is created.
- If any of the above steps fails, then the error is logged, and the initialization gets terminated.

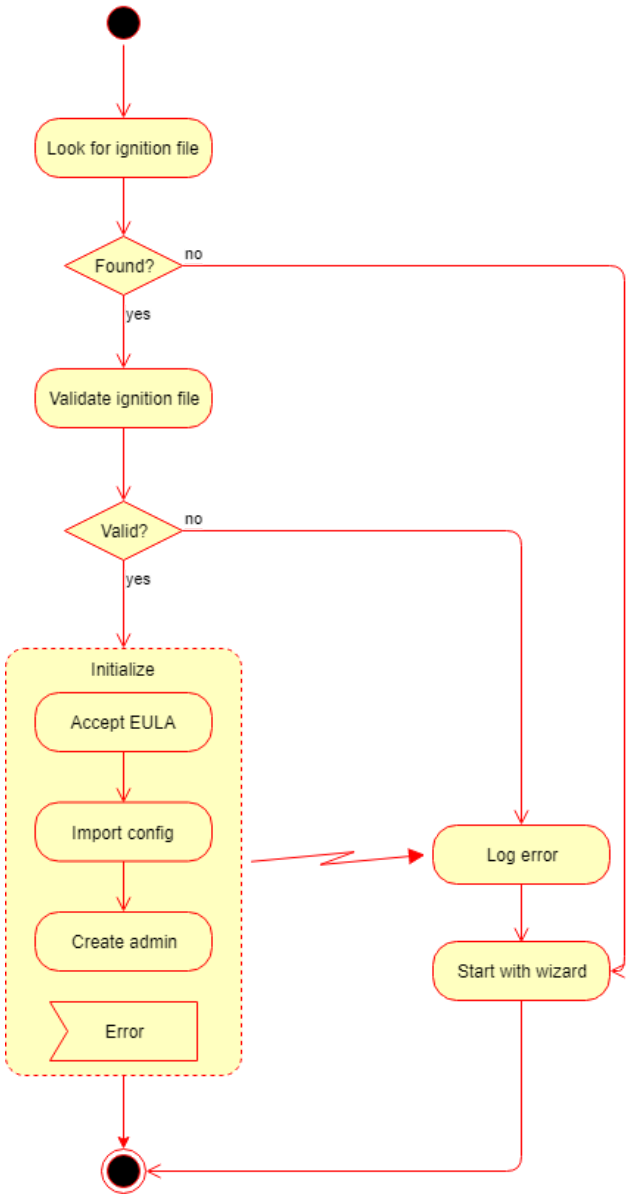
> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

2.2) If it is not valid, then the error is logged, and the initialization gets terminated.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

3) If there is no ignition file, then no initialization is performed.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.



Warning

If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried fixing the problem (e.g. placing the ignition file to its lookup location) and restarting the OPSWAT MetaDefender Core service.

Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

Configuration

After the initialization is complete, the product is ready with the default and the imported configuration.

This configuration can be later changed calling the configuration API functions. For further details about the API see Configuration related APIs API

CIS Level 1 Guidelines

Support Red Hat Enterprise Linux 9
For more details about Center for Internet Security (CIS) please refer to this document

Instruction steps

I. Install OpenSCAP

bash bash

```
yum install openscap-scanner scap-security-guide
```

II. Generate a result file and a HTML report using OpenSCAP scanner tool

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

III. Remediation of CIS Level 1 issues

Generate a remediation script based on the `ssg-rhel9-ds.xml` file:

bash

```
oscap xccdf generate fix --profile
xccdf_org.ssgproject.content_profile_cis_server_l1 --fix-type bash --output
remediations.sh /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

And execute remediation script.

bash

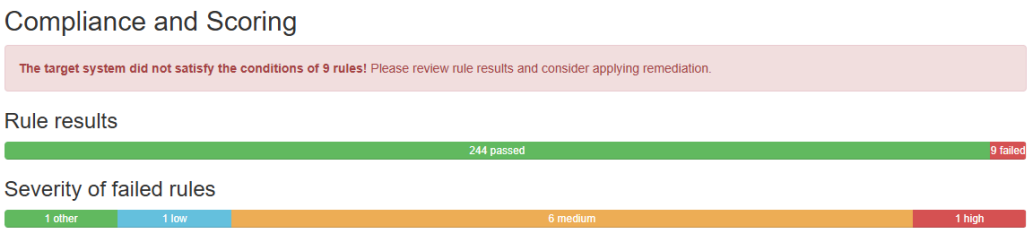
```
./remediations.sh
```

IV. Review the results after remediation

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

Example:



Notes

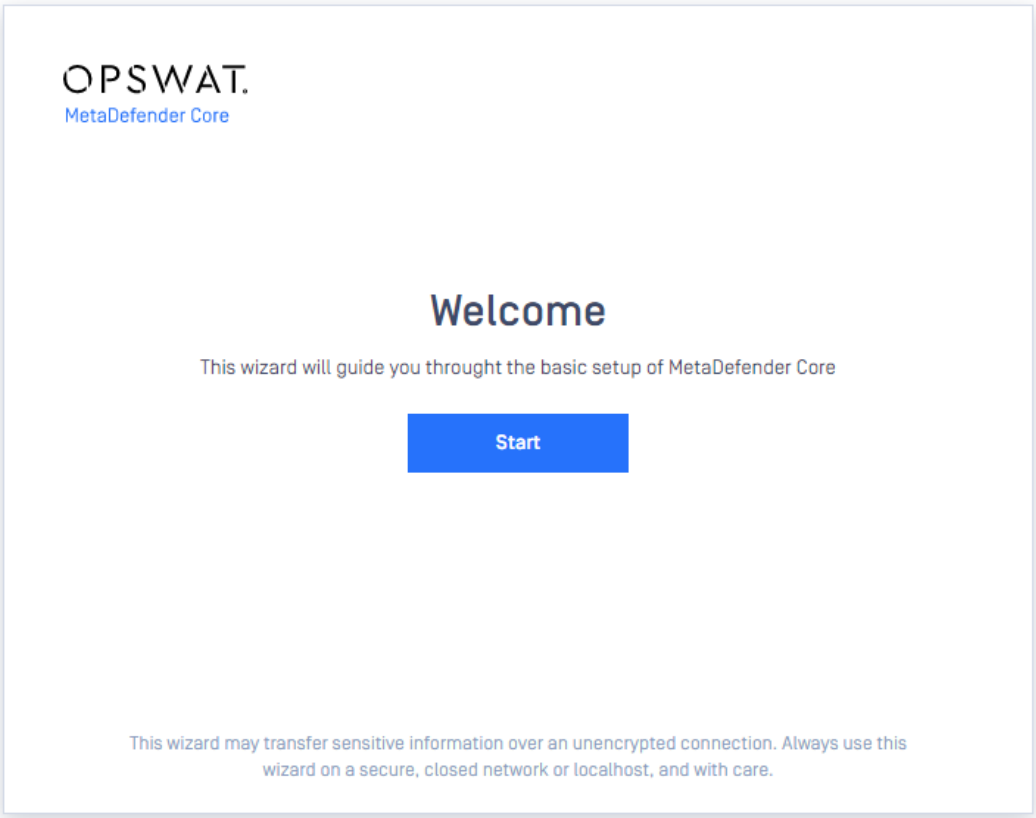
- CIS Level 1 requires `/tmp` folder to be mounted in a separate partition. Please ensure that that new partition have enough disk space for MetaDefender Core to run.
- CIS Level 1 requires that "Ensure No World-Writable Files Exist".
    - For now, when freshly installing MetaDefender Core, all its binary files meet the requirement.
    - When installing/updating engines, some engines might create additional files for its operation, and it might violate this requirement. In this case, you need to again execute the remediation script in the step III.

Wizard Setup

Introduction

Access the MetaDefender Core management console for the first time (i.e. http://localhost:8008), you are to complete a quick walkthrough for wizard setup in order to be able to use the product.

The administrative management console will be available only upon you have successfully finished this wizard stage.

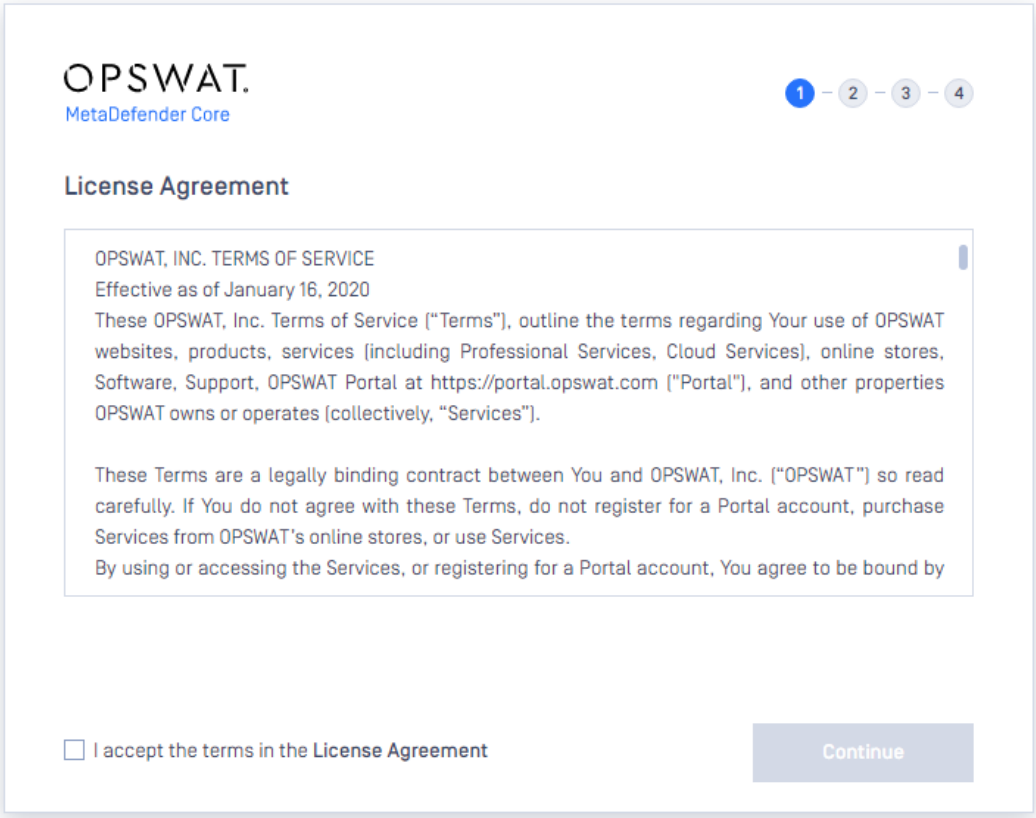


Warning

This wizard may transfer sensitive information over an unencrypted connection. Always use this wizard on a secure, closed network or localhost, and with care.

Basic configuration steps

End-User License Agreement



In the first page you can find the End-User License Agreement. You have to accept the terms before moving on.

Please read through the EULA (https://docs.opswat.com/policies/privacy-policy/portal-privacy-policy) carefully and if you agree with it, check `I accept the terms in the **License Agreement**` and click `Continue` to proceed.

Local Admin User Setup

The next step is to set up a default local administrator account.

This account will be the first one being able to access the administrative management console, and to create accounts for other users. You have to fill out all fields in this page to be able to move forward. When you are done, click NEXT to continue.

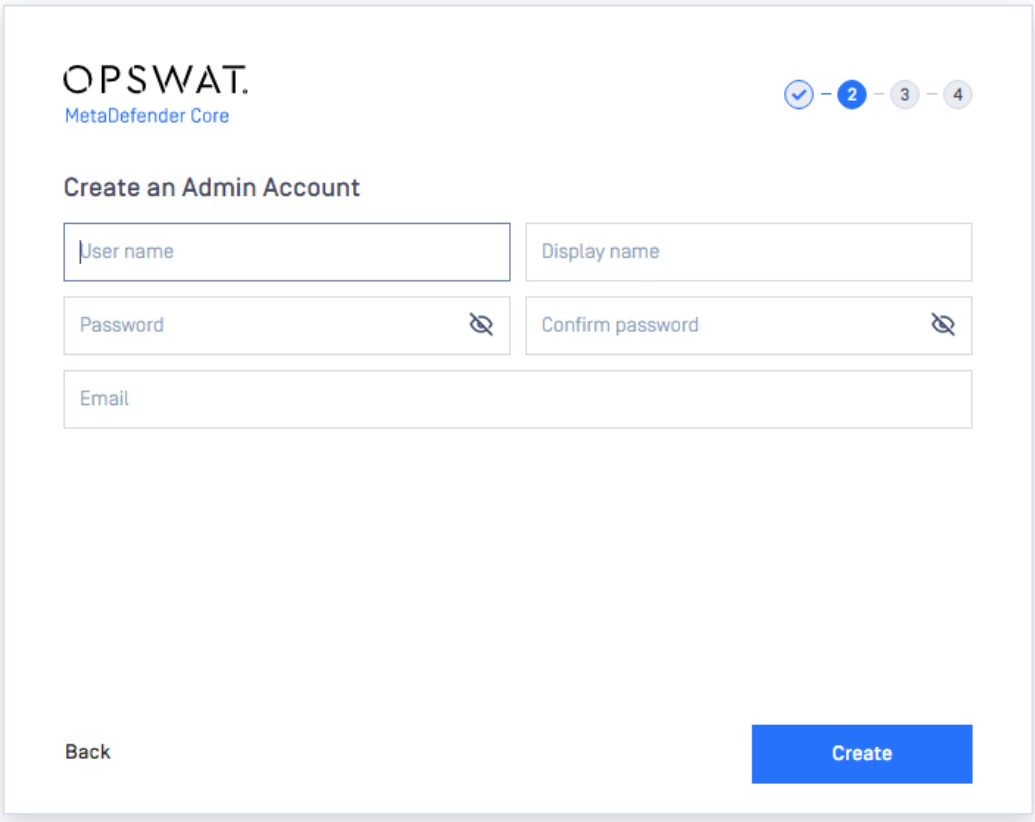
Info - User Directory

The administrator account, that is created via the basic configuration wizard, is always added to the LOCAL user directory as a member.

Learn more about user directory: User directories

The following information is required for the administrator account:

ACCOUNT NAME	The unique name of the account that is used at the time of login and in log messages for accountability.
ACCOUNT DISPLAY NAME	Name of the person bound to this account. This name (appended to the name of the account's user directory) is displayed in the top right corner of the administrative management console.
PASSWORD	Password of the user bound to this account that is used at the time of login. Passwords sent clear-text As long as TLS is not configured for the basic configuration wizard, passwords are sent clear-text over the network and may be disclosed to unauthorized parties. As a mitigation action: 1. Either use the wizard on *localhost* or on a direct network connection, or 2. Enabling HTTPS as soon as possible and change the password immediately if it has already been set.
EMAIL	Email address of the person bound to this account.



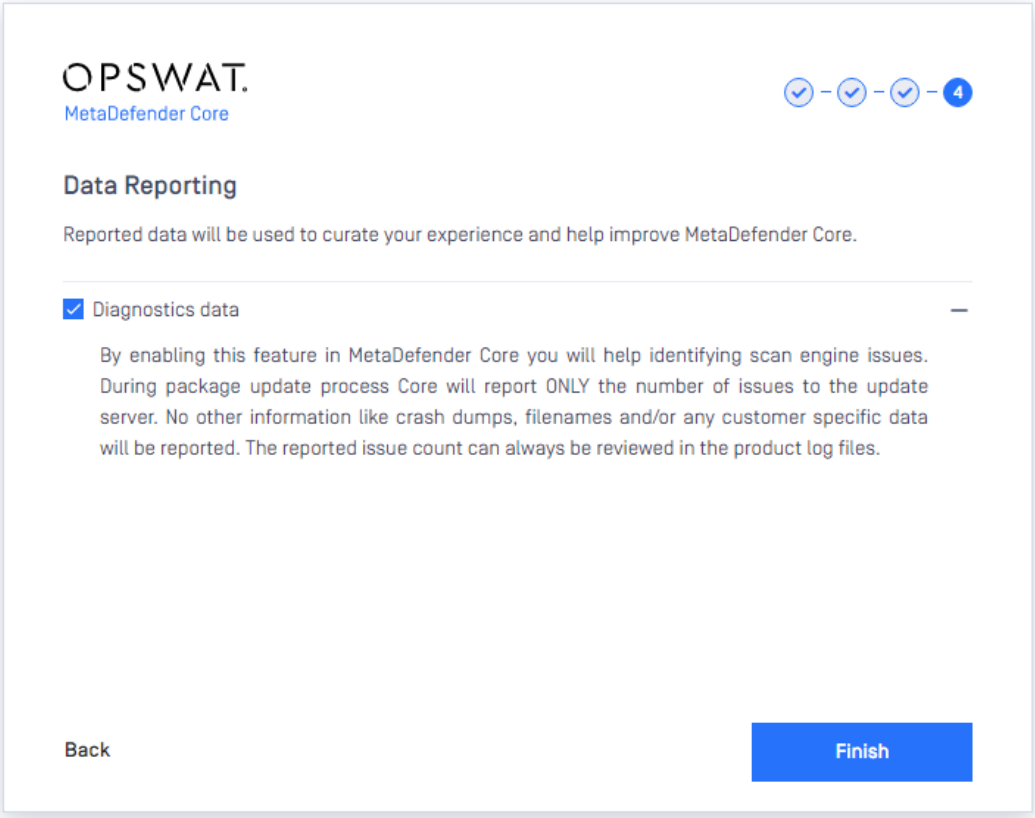
License activation

For license activation details see License Activation

Data Reporting

Enabling this option to allow MetaDefender Core collect its own engine issues and report back to OPSWAT for further analysis.

MetaDefender will not collect any other information.

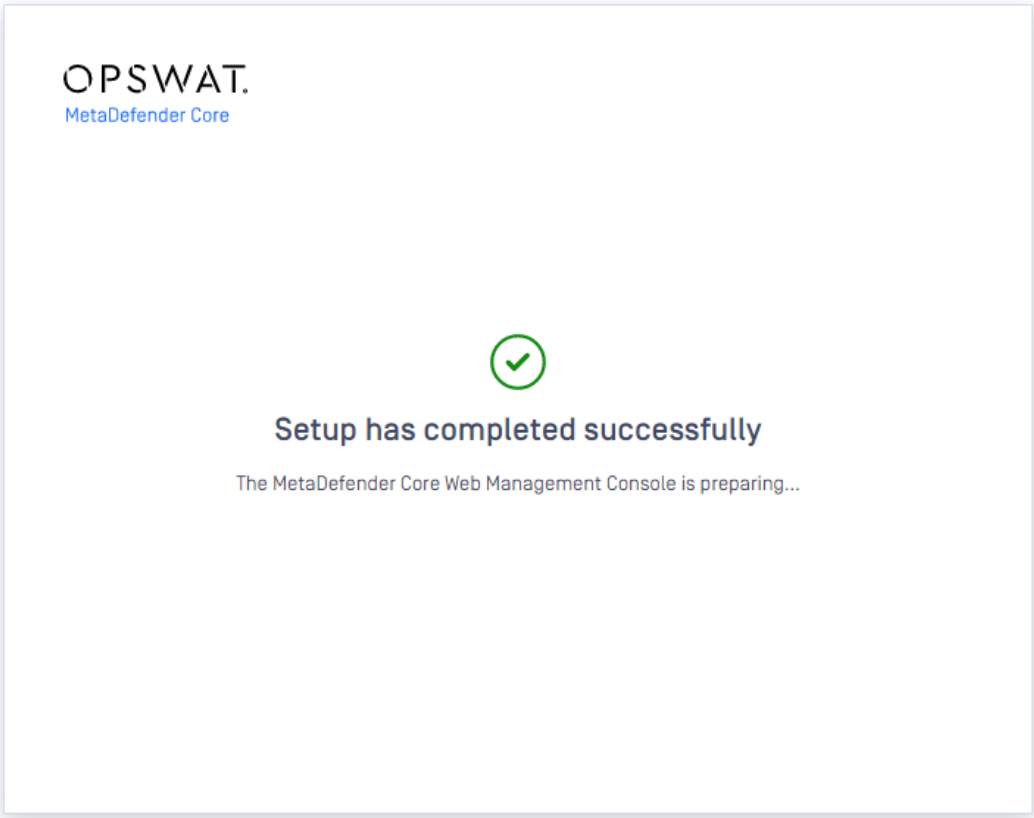


Wizard completion

After you have completed every steps you are ready to finish the wizard and start using the product. Click the `Fishish` button to complete the wizard.

> **Warning**
> The product's service will be restarted and the browser will be redirected to the administrative management console. This could take several seconds.

You can login to the Web Management Console with the administrator user that have just been created in the previous steps.



Setup HTTPS

Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network. Websites, like the Web Management Console, are able to use TLS to secure all communications between their servers and web browsers.

The TLS protocol aims primarily to provide confidentiality (privacy) and data integrity between two communicating computer applications.

Warning

No TLS for the wizard

By default, TLS is not enabled for the basic configuration wizard. As a consequence sessions between the wizard's backend and the browser may be insecure.

Info

Performing the same steps as for the Web Management Console, it is possible to set up TLS for the basic configuration wizard. Remember completing the TLS setup before launching the wizard.

For instructions to set up TLS see Enabling HTTPS

License Activation

##

To activate your installation go to the **Settings > License** menu in the Web Management Console. If you have no valid license, you will only see your installation's **deployment ID**. You will also see a warning in the Web Management Console header.

After successful activation, the product will start downloading the latest available scan engines and malware databases. You can follow the status of the scan engine installation on the **Inventory > Modules** page.

When all the licensed engines are installed, you can start using the MetaDefender Core to process files - File Processing.

Deployment ID

Each MetaDefender Core instance represented by an unique system fingerprint called `Deployment ID`

MetaDefender Core licensing is purely based on deployment ID activated. Therefore, everytime your MetaDefender Core is activated with a different deployment ID, then it will be captured as a different activation record accumulated into your total activation times allowed.

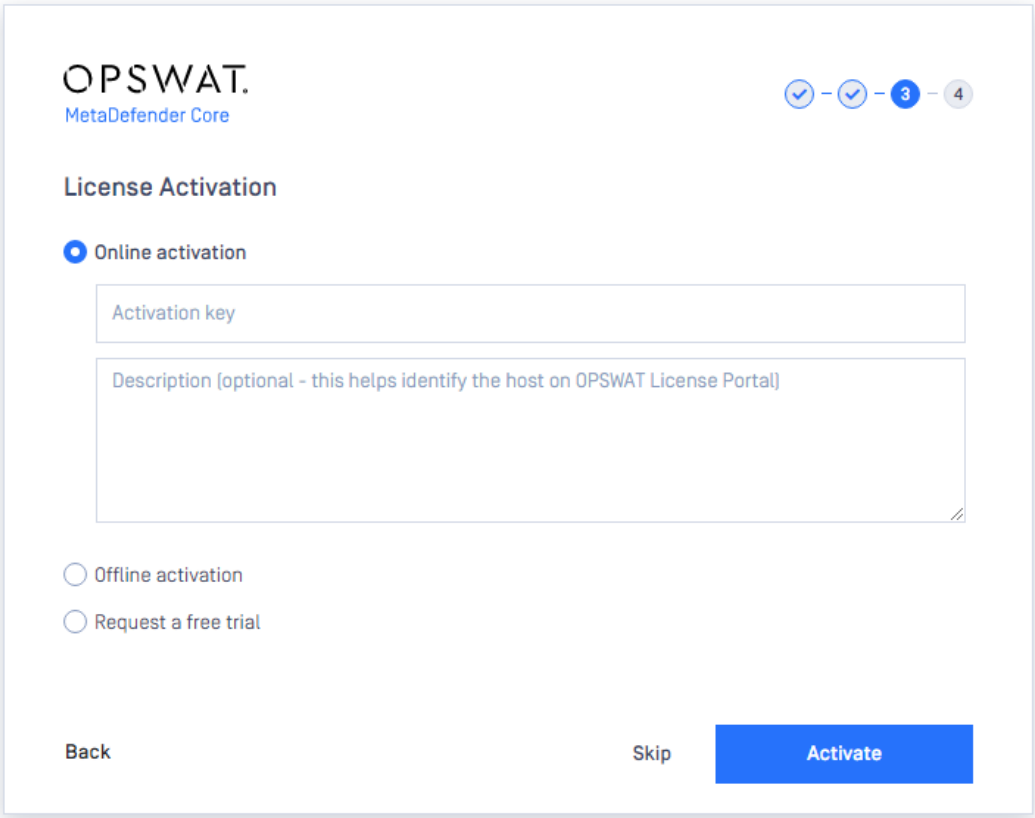
> **Notice**
> Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you alter the system where MetaDefender Core relies on (e.g. network card, OS hostname).
>
> Deployment ID is an UUID version 4 random based string. Learn more at: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you update the system where MetaDefender Core relies on.

Activation Modes

There are three different options for the product activation:

- Online activation: MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.
- Offline activation: Requiring you to upload a license file (.yml). That license file could be retrieved by activating your license via OPSWAT Portal. Follow the displayed instructions for details.
- Request trial key online: For evaluation purpose, you can receive an trial Activation key via email. Follow the displayed instructions for details.



Online activation

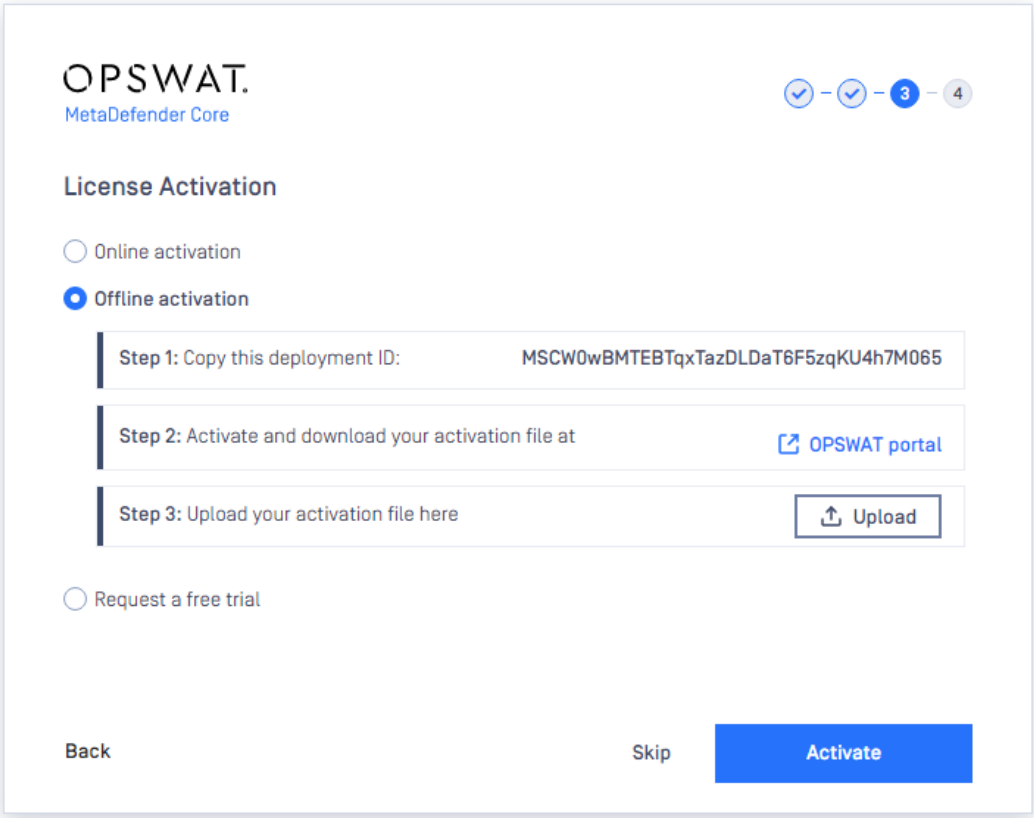
MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.

Required to input your Activation License key, and optional for description (but we recommend this for troubleshooting and auditing purpose if needed).

> **Warning**
> If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts' for license activation:
>
> - https://activation.dl.opswat.com Note: *IP address-based whitelisting on your firewall might fail after some time since OPSWAT uses CDN (Content Delivery Network) to faster delivery updates over the world, and IP address of edge servers might change over time.*

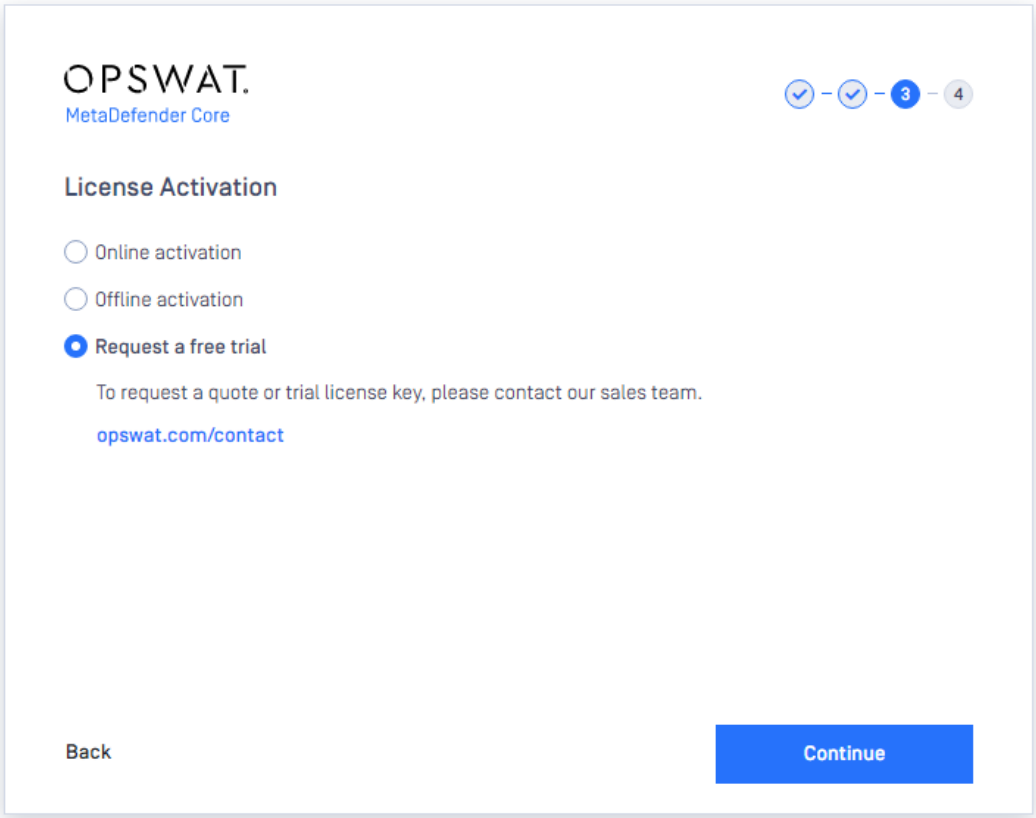
Offline activation

With no internet connection on the server the Metadefender v4 instance may be activated indirectly from a different machine, that has internet connection. The Deployment ID of the Metadefender v4 instance and the the Activation key received at the time of purchasing the product will be required. Follow the steps on the screen to activate the product offline.



Request trial key online

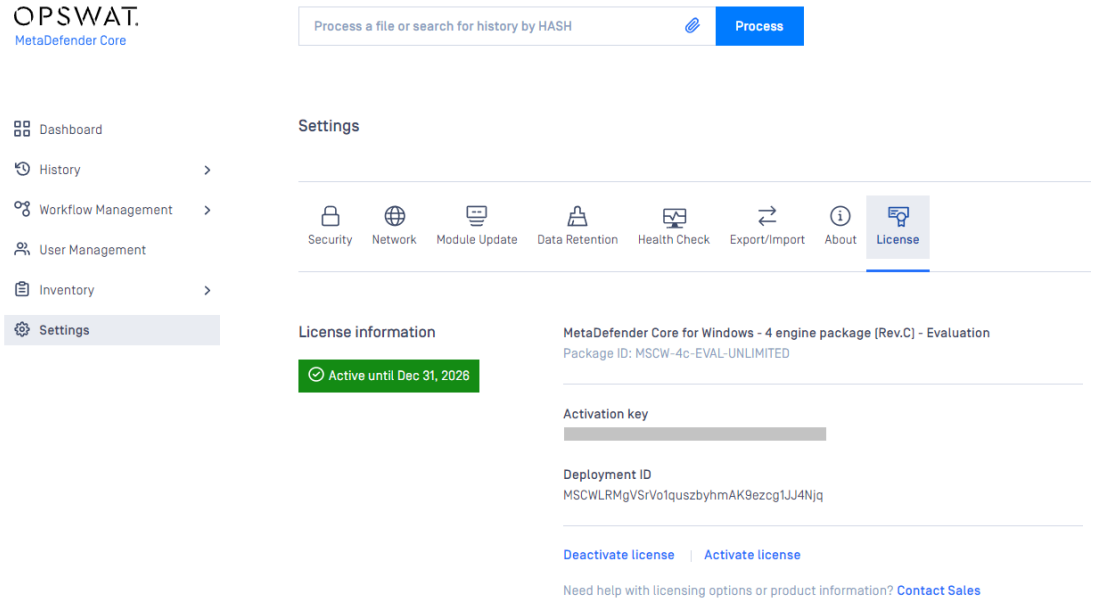
An evaluation license could be acquired by contacting our sales team.



Check Your License Details

For more license details and activating your MetaDefender Core, go to Settings > License menu tab on the Web Management Console:

- Product name: product name as on your order (e.g. MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation)
- Package ID: product identification as on your order (e.g. MSCW-4c-EVAL-UNLIMITED)
- Expiration: last day of license validity
- Activation key: key used for activating product
- Deployment ID: identification of this installation


OPSWAT.
MetaDefender Core
Process a file or search for history by HASH
Process
Dashboard
History
Workflow Management
User Management
Inventory
Settings
Settings
Security
Network
Module Update
Data Retention
Health Check
Export/Import
About
License
License information
Active until Dec 31, 2026
MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation
Package ID: MSCW-4c-EVAL-UNLIMITED
Activation key
Deployment ID
MSCWLRMgVSrVo1quszbyhmAK9ezcg1JJ4Njq
Deactivate license
Activate license
Need help with licensing options or product information? Contact Sales

Scripted license management

Using REST API calls there is an option for scripted activation and deactivation for MetaDefender Core servers with **no Internet connection or if your infrastructure is using automation to create/destroy MetaDefender Core instances**. In this chapter the steps of these two scenarios are described.

Requirements

- an installed MetaDefender Core instance without Internet connection
- another computer that has Internet connection and can run your activation scripts
- a manual or automated way to transfer data between the two computers

Activation steps

1. In order to activate MetaDefender Core, deployment ID and activation key are needed to generate the license file.
    a. Activation key should be purchased from OPSWAT.
    b. MetaDefender Core deployment ID can be queried by using REST API (for details see GET Get Current License Information API page)

> **Info**
> Save this deployment ID in your system, you might need this ID when the instance is unavailable at the time of deactivation.

2. Activate license and get the license file using the following URL, where <quantity> is 1.

activation call bash

```
curl -X GET https://activation.dl.opswat.com/activation?key=<activation-key>&deployment=<deployment unique-ID>&quantity=<quantity>
```

If the activation is successfully, a license file is downloaded. Save this file.

HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	license file	Activation was successfully.
200 Activation failed	error: '<user conform error message>'	Failed activation
200 Invalid parameter	error: 'Could not activate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not activate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
200 Invalid parameter	error: 'Could not activate your product because the quantity you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key, quantity or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Info
In case of any activation issue, contact OPSWAT support for help

3. Upload license file to MetaDefender Core: The license file should be uploaded to the MetaDefender Core v4 to activate the product. For details see page: POST Upload license key API

Deactivation steps

1. For deactivation, following infomation is required:
    a. Activation key
    b. MetaDefender Core deployment ID: can be queried by using REST API (for details see GET Get Current License Information API page)
2. Deactivate license using the following URL:

deactivation call bash

```
curl -X GET https://activation.dl.opswat.com/deactivation?key=<activation-
key>&deployment=<deployment unique-ID>
```

Response HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	result: ok	Successful deactivation
200 No active license found	error: 'Could not found any active license with the given parameters'	The license has not been activated yet or it has been deactivated already.
200 Invalid parameter	error: 'Could not deactivate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not deactivate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Important notes

Warning

Product activation is tied to several hardware, operating system and software parameters. In case of one or more major hardware or software parameter change the product might turn into deactivated status. Operating system updates, other software updates should not affecting the activation status.

If the product is online activated and has live Internet connection then in case of deployment ID change the product reactivates itself. If the product doesn't have Internet connection then the administrator is responsible to reactivate the product.

Because of the above administrator should consider licensing restrictions of the offline deployments if using any virtualization and/or containerization technologies.

File Processing

MetaDefender Core provides REST API interface for file processing business.

There are several ways to process files with MetaDefender Core:

- Analyze Files via Web Interface
- Process Files via REST API

Product Upgrading

> **Info**
> Simply run installer file, details could be found at Installation

> **Info**
> For command line mode, upgrading (e.g. on **Linux**), prior to running the installer file, please make sure to **create the ignition file** with PostgreSQL server information included. Details:
>
> - Standalone DB mode: Command Line Installation (Windows and Linux)
> - Shared DB mode: Command Line Installation

Recommendation for MetaDefender Core upgrade

Our goal is to help you execute the product upgrade continuously, efficiently, and correctly.

PostgreSQL service upgrade

- **Bundled PostgreSQL Service**: If you are using the bundled PostgreSQL server included in the installer, the PostgreSQL service will be automatically upgraded to the newer version (if available). No additional action is required for this process.
- **Remote PostgreSQL Service**: For remote PostgreSQL servers, customers are responsible for managing and upgrading the service independently. Upgrading MetaDefender Core does not update the remote PostgreSQL service.

Before the upgrade

Prepare a proper disaster recovery plan (DRP) including some common practices:

- Backup, or take a snapshot of your current working MetaDefender Core server.
  - This is to help quickly rollback for any upgrade failure might cause later for any reason.
- Optionally backup your PostgreSQL database, especially when using remote shared database mode.
  - Reference: https://www.postgresql.org/docs/current/backup.html
  - Please do it after stopping all linked MetaDefender Core instances to avoid potential data collision.
- Lastly, if backing up entire server is not feasible, then at least export MetaDefender Core configuration package Import/Export configuration just in case you'll need to start off a new server with current MetaDefender Core configuration to keep the scanning service running without any impact.

On the PostgreSQL server, double check server resource to ensure sufficient RAM memory and disk space to be preserved for the upgrade process.

- 8GB RAM available at least during the upgrade.

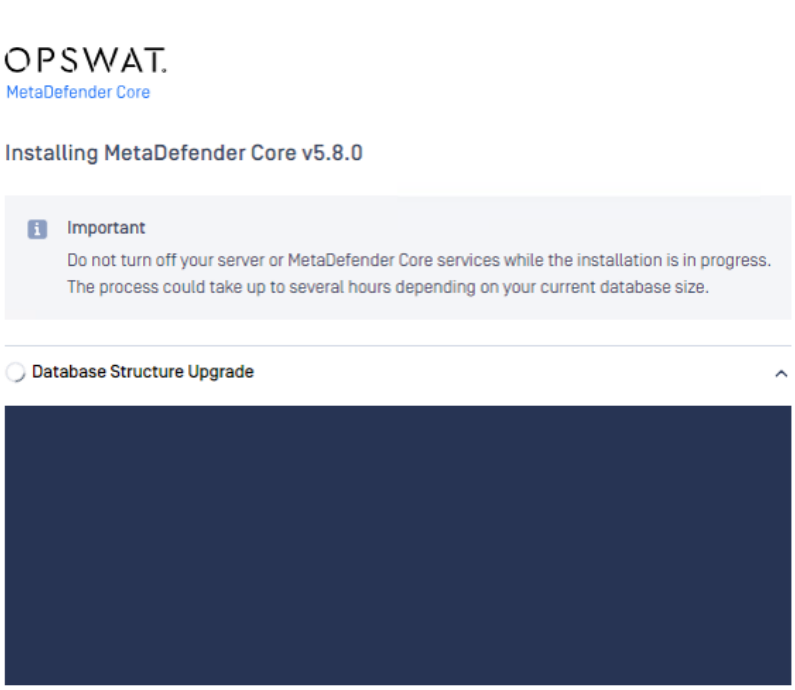
- Disk availability is greater than MetaDefender Core database size.

While doing the upgrade

Applicable for shared database deployment:

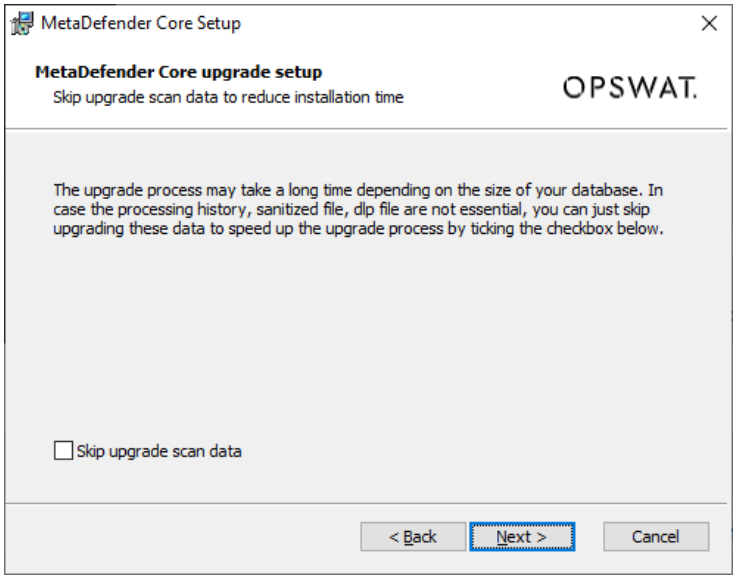
- Stop all MetaDefender Core instances' scanning service before performing the upgrade.
- Upgrade one (1) MetaDefender Core instance at a time.
- Proceed to upgrade MetaDefender Core instance where the bundle local PostgreSQL server is used for shared database setup.
- Otherwise is using remote PostgreSQL then the upgrade order does not matter.

For large size database, mostly for scan history, expecting long time to be spent on the upgrade process.



The product supports an option to skip data migration for scan history to expedite the process drastically and mitigate any failure risk. You can manually backup all data from your PostgreSQL server for offline usage if any.

Check "Skip upgrade scan data" to enable this feature while doing the upgrade.



or add following key in the ignition file:

none

```
[internal]
skip_upgrade_scan_data = 0
```

Troubleshooting

If the upgrade process ever runs into an error:

- Double check your network and services, applicable to remote PostgreSQL server.
- Optionally configure MetaDefender Core to skip migrating scan history.
  - Windows
    - Open Windows registry, and navigate to HKLM\Software\OPSWAT\Metascan
    - Add a new key `internal/skip_upgrade_scan_data` with value = 1
  - Linux
    - Open MetaDefender Core configuration file `/etc/ometascan/ometascan.conf`
    - Add new key into the file
      - [internal]
      - skip_upgrade_scan_data=1
  - Note: This key will be auto removed upon the upgrade finishes.
  - Then restart MetaDefender Core service to retry the upgrade.
    - Might retry this a few times until the upgrade is done.

Just in case the upgrade process still fails, please contact OPSWAT support with the support package - How to Create Support Package? This will help us analyze the root cause and figure out an optimal solution for your system failure.

Using external Load Balancer

Application Load Balancing (Layer 7)

This page provides information about recommended ways to use sessions with a Layer 7 load-balancer.

Most load-balancers have the ability to provide cookies for pinning subsequent traffic from a client to the appropriate server. This method is called sticky session, session persistence or session affinity. Using cookies efficiently requires the client to know when a cookie should be sent, should not be sent or should be deleted.

Sticky session load balancing

Each Metadefender Core instance has its own databases and application sessions that cannot be seen by other instances. Therefore, in order to get the related data/response to our queries we should ask the appropriate Metadefender Core v5 server. On the other hand, to keep the advantages of the used load balancing method, cookies should not be sent if it is not necessary.

Single file scanning

Step	Stage	Task	Cookie usage
1.	Sending file	Initiate processing a file on the client side. Send file through the load-balancer. (See POST File Analysis API)	Cookie should not be sent
2.	Sending file	Save the cookie and the data_id you got from the load-balancer	Save cookie
3.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Send cookie saved in step 2
4.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Send cookie saved in step 2

Batch scanning

Step	Stage	Task	Cookie usage
1.	Open batch	Initiate processing file(s) in batch. Request a batch ID through the load-balancer. (See POST Initiate Batch API)	Cookie should not be sent
2.	Open batch	Save the cookie and the batch_id you got from a Core server through the load-balancer.	Save cookie
3.	Sending files	Send file through the load-balancer. (See POST File Analysis (using batch header) API)	Send cookie saved in step 2
4.	Sending files	Save the data_id you got from the load-balancer.	-
	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Send cookie saved in step 2
5.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
6.	Getting batch status	See GET Batch Status API	Send cookie saved in step 2
7.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Send cookie saved in step 2
8.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API	Send cookie saved in step 2

If it does not matter which upstream server responds, the queries should be sent without cookie.

> **Info**
> It is recommended not to send cookies when it's not necessary to allow load-balancer to use its own method to share the load between Metadefender Core v5 servers.

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration through load-balancer

OPSWAT products that support HTTP load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	4.3.4	-
OPSWAT Client	Windows: 7.6.247.0 Mac: 10.4.243.0	-
MetaDefender Email Security	4.3.0	-
MetaDefender ICAP Server	4.3.0	-
MetaDefender Vault	1.3.0	-
MetaDefender for Secure Storage	2.0.7	-

DNS Load Balancing

Using this method is logically similar to Layer 7 load-balancing.

Briefly how it works

Client uses a domain name to send a query to a server. Client's DNS server has more "A" records for that name with different IPs. When a client resolves the server's domain name DNS server randomly chooses an IP for that name to send back. When a session is used on the application layer, client should know the IP address of the Core server that handles that specific session. In every other case, client should resolve the domain name with DNS query to let requests to be balanced between Metadefender Core servers.

Single file scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Sending file	Initiate processing a file on the client side. (See POST File Analysis API)	Use IP saved in step 2
4.	Sending file	Save the data_id got from the Core server	-
5.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Use IP saved in step 2
6.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Use IP saved in step 2

Batch scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Open batch	Initiate processing file(s) in batch. Request a batch ID. (See POST Initiate Batch API)	Use IP saved in step 2
4.	Sending files	Send file to the specific Core server with the basch ID saved in step 3. (See POST File Analysis (using batch header) API)	Use IP saved in step 2
5.	Sending files	Save the data_id you got from the Core server.	-
6.	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Use IP saved in step 2
7.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
8.	Getting batch status	See GET Batch Status API	Use IP saved in step 2
9.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Use IP saved in step 2
10.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API).	Use IP saved in step 2

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration via DNS load-balancing

OPSWAT products that support DNS load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	does not support yet	-
MetaDefender Client	does not support yet	-
MetaDefender Email Security	does not support yet	-
MetaDefender ICAP Server	does not support yet	-
MetaDefender Vault	does not support yet	-

SSL connection for PostgreSQL communication

> **Note**
> The SSL connection configuration is possible thanks to the support made by PostgreSQL. For reference: https://www.postgresql.org/docs/14/ssl-tcp.html
>
> MetaDefender Core is tested against this mode on particular version 4.21.1

Step 1: Generate SSL certificate for PostgreSQL server

> **Note**
> Following instructions are for self-sign certificate. For production environment, making sure to use the certificate signed by your trusted CA.

1) Generate a private key

bash

```
openssl genrsa -des3 -out server.key 1024
```

Sample output:

bash

```
Generating RSA private key, 1024 bit long modulus ........++++++
...............++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

Type your desired password / pass phrase to encrypt the private key

2) Remove the pass phrase to automatically start up the PostgreSQL server

bash

```
openssl rsa -in server.key -out server.key
```

> **Note**
> On Linux, make sure to set permission on the server.key file

bash

```
chmod og-rwx server.key
```

3) Create a self-signed certificate

bash

```
openssl req -new -key server.key -days 3650 -out server.crt -x509
```

You will be prompted to enter detailed information which is incorporated into your self-signed certificate request.

none

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:CA
Locality Name (eg, city) [Default City]:San Francisco
Organization Name (eg, company) [Default Company Ltd]:OPSWAT
Unit Name (eg, section) []:OPSWAT
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:dummy_email@opswat.com
```

For self-signed certificate, use the server ceriticate as the trusted root certificate:

bash

```
cp server.crt root.crt
```

Copy **server.key**, **server.crt**, **root.crt** to PostgreSQL data folder. For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data**
- Linux: **/var/lib/ometascan/pg_data/**

Step 2: Configure PostgreSQL server for SSL authentication connection

1) Create a custom config file (e.g. ssl.conf) for PostgreSQL.

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\postgres\ssl.conf**
- Linux: **/var/lib/ometascan/postgres/ssl.conf**

2) Content of ssl.conf:

none

```
ssl = on
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
```

3) Modify pg_hba.conf

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data\pg_hba.conf**
- Linux: **/var/lib/ometascan/pg_data/pg_hba.conf**

Add following (modify **::1/128** if needed to match with your environment requirement)

none

```
hostssl all all      ::1/128      scram-sha-256
```

After the modify, the content of pg_hba.conf should be like this (example):

> **Note**
> In a Linux environment, a local connection is required to upgrade from version 5.5.1 or earlier to version 5.6.0 or later.

none

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local         DATABASE  USER  METHOD  [OPTIONS]
# host          DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl     DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostgssenc    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnogssenc  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
local    all all                    scram-sha-256
hostssl all all      ::1/128      scram-sha-256

#More content.......
```

4) Reload the services

- Standalone DB mode: Restart **ometascan** service (MetaDefender Core service)
- Shared DB mode: Restart **ometascan-pg** / **ometascan-postgresql** service (MetaDefender Core PostgreSQL service) and then **ometascan** service (MetaDefender Core service)

Test your SSL database connection:

bash

```
psql.exe "sslmode=require dbname=postgres user=postgres host=localhost port=5432"
Password for user postgres:
psql (12.7)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
Type "help" for help.
```

This message indicate that you are now using SSL connection:

bash

```
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
```

Performance and Load Estimation

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Some factors that affect performance

- MetaDefender Core version
- MetaDefender Core engine package and configuration
  - set of engines (which and how many)
  - product configuration (e.g., thread pool size)
- System environment
  - server profile (CPU, RAM, hard disk)
  - client application location - remote or local
  - system caching and engine level caching
- Dataset
  - encrypted or decrypted
  - file types
    - different file types (e.g., document, image, executable)
    - archive file or compound document format files
  - file size
  - bad or unknown (assume to be clean)
- Performance tool

How test results are calculated

Performance (mainly scanning speed) is measured by throughput rather than unit speed. For example, if it takes 10 seconds to process 1 file, and it also takes 10 seconds to process 10 files, then performance is quantified as 1 second per file, rather than 10 seconds.

- total time / total number of files processed: 10 seconds / 10 files = 1 second / file.

Standalone DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.2.416.0

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

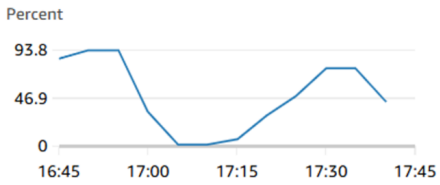
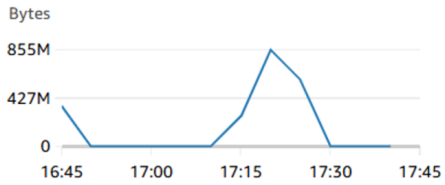
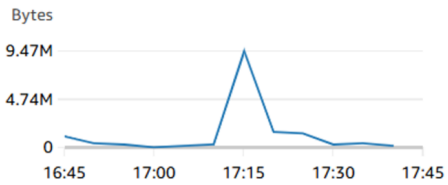
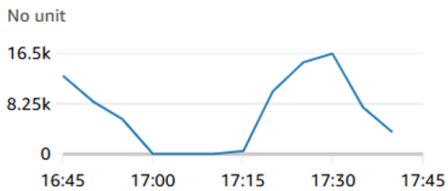
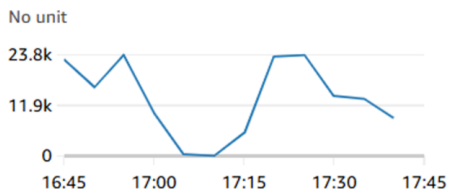
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV)	24.2	47.2 / 80	37.3 / 41.1		315,310	0.0114
Deep CDR	19.9	48.9 / 81.8	32.6 / 37		383,442	0.0094
Proactive DLP	17.1	36.8 / 58.2	30.4 / 33.6		446,228	0.0081
Vulnerability	10.1	59.2 / 85	29.3 / 33.3		755,495	0.0048

MetaDefender Core with common engine packages:

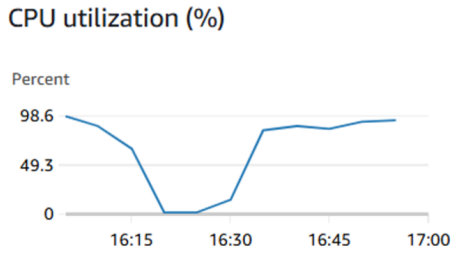
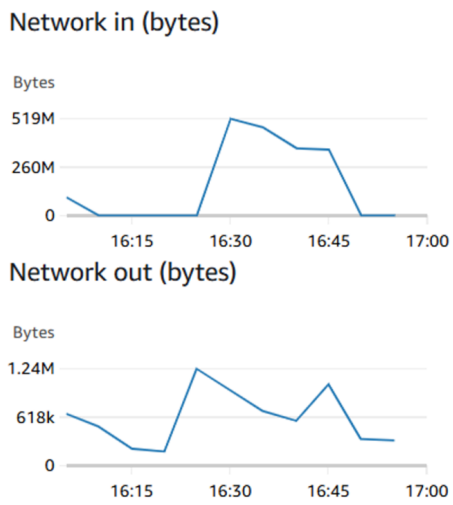
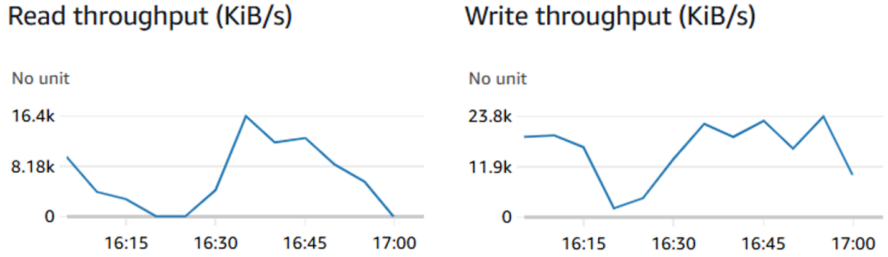
Use case	Scan duration	Avg./Max CPU usage	Avg./Max RAM usage	Avg./Max Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV) + Deep CDR	30.8	68.1 / 88	40.1 / 46.8		247,744	0.0145
Metascan (8 AV) + Deep CDR + Proactive DLP	31.2	74.2 / 98.6	41.5 / 51.1		244,567.3	0.0147
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	31.8	73.5 / 98	40.6 / 51.5		239,953	0.015

System resource utilization charts

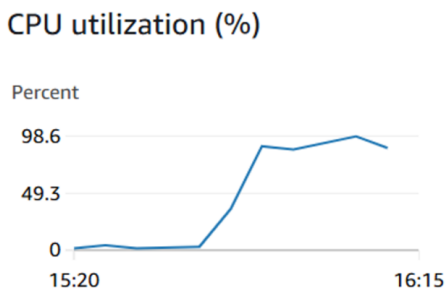
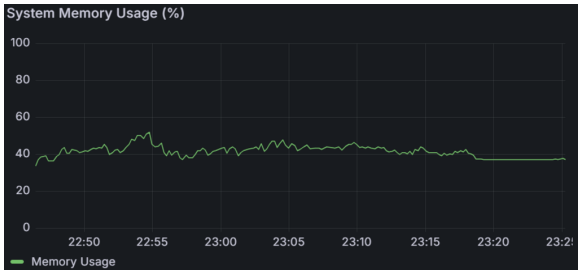
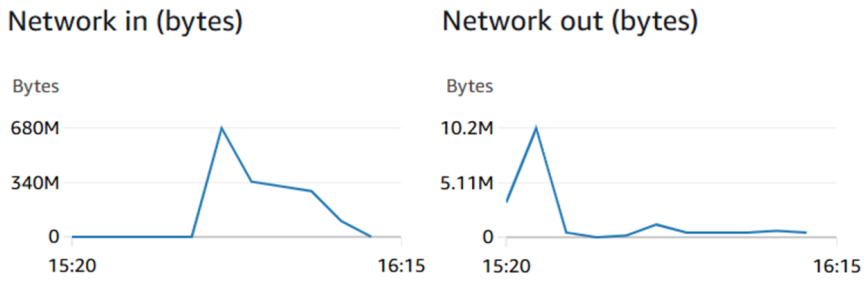
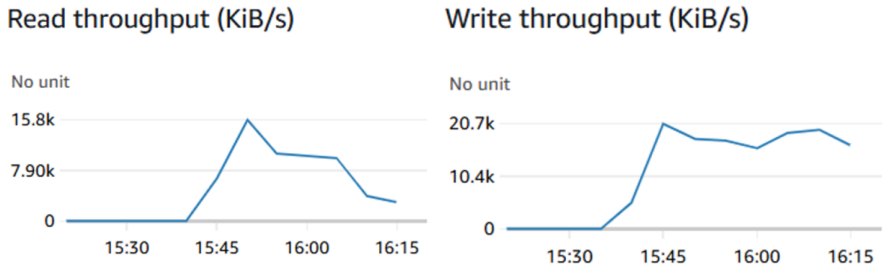
MetaDefender Core with Metascan (8 AV engines):

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

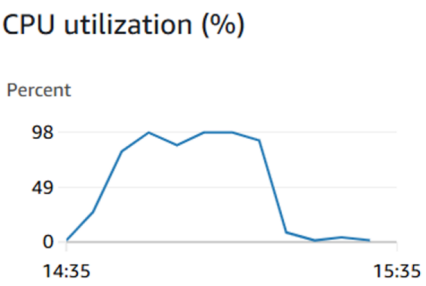
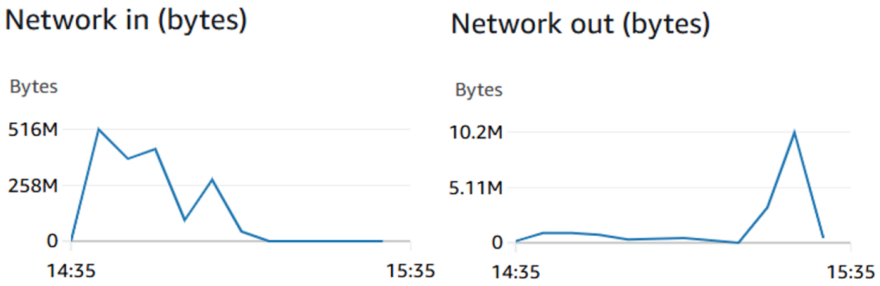
MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

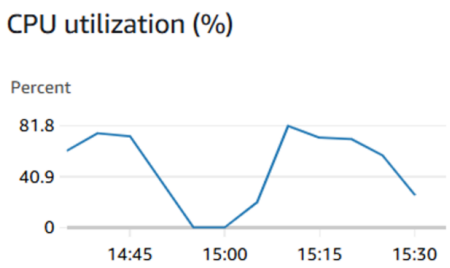
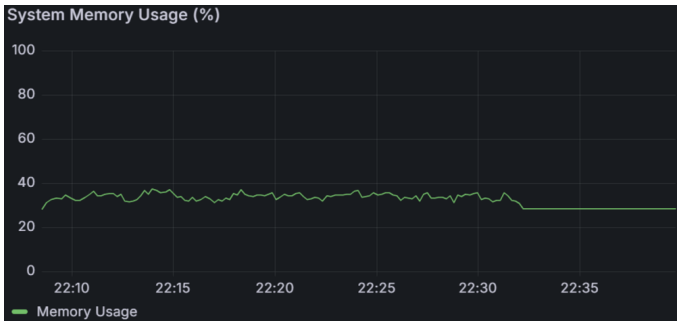
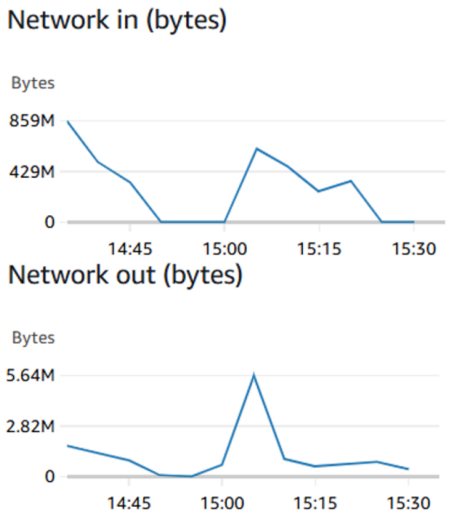
MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

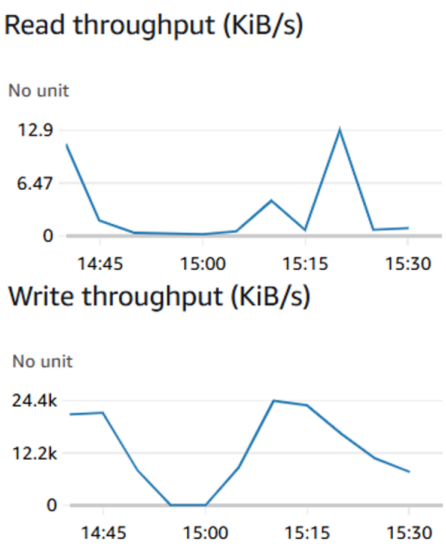
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

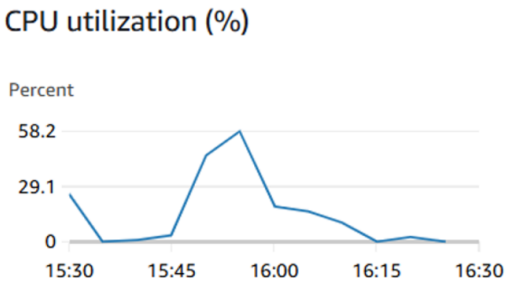
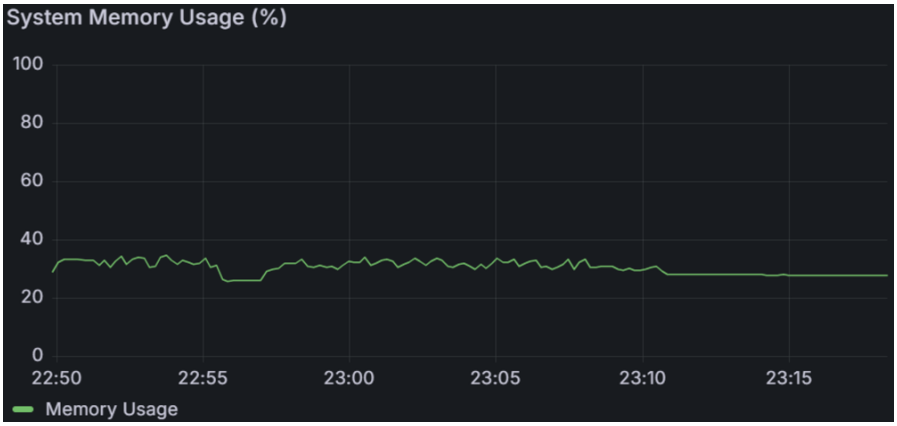
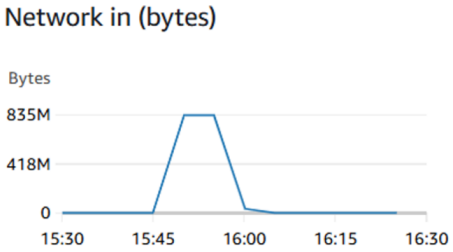
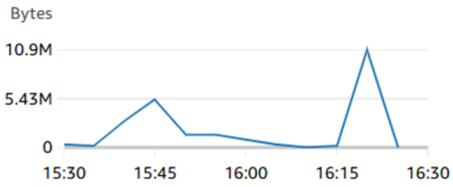
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

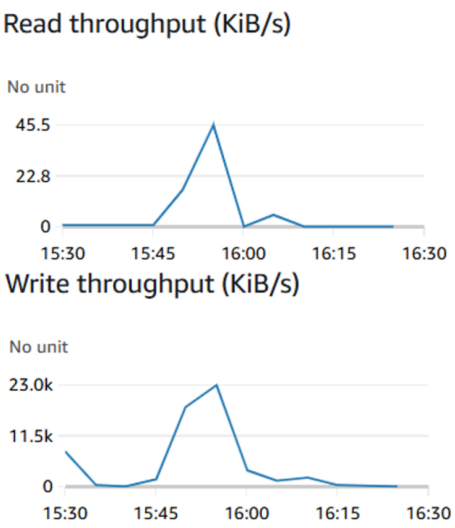
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

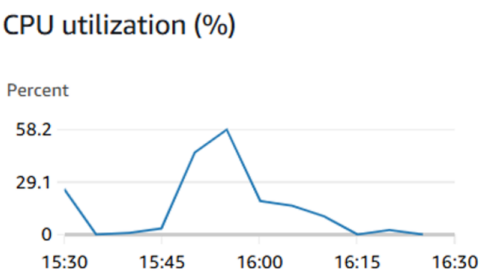
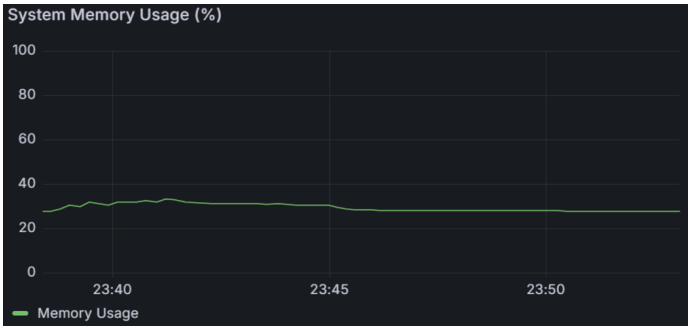
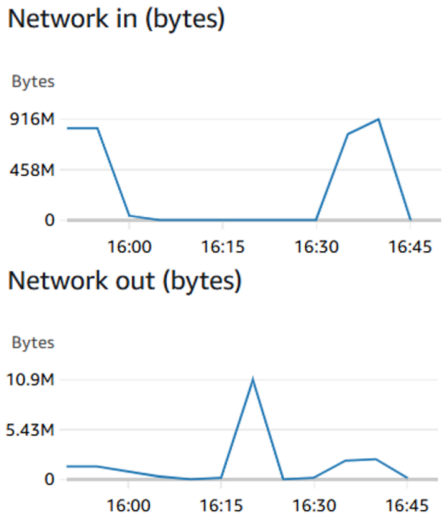
	Utilization charts
Disk usage	

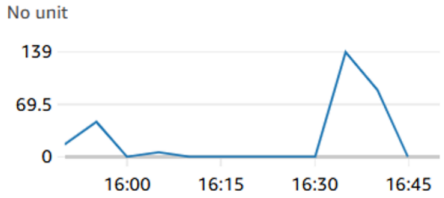
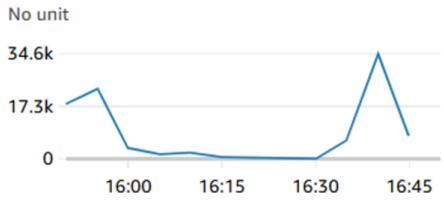
MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	 

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability:

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

Standalone DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Ubuntu 22.04	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.57-236

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

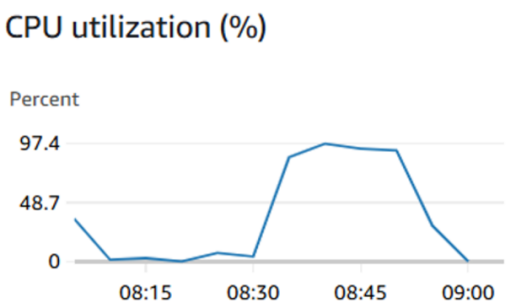
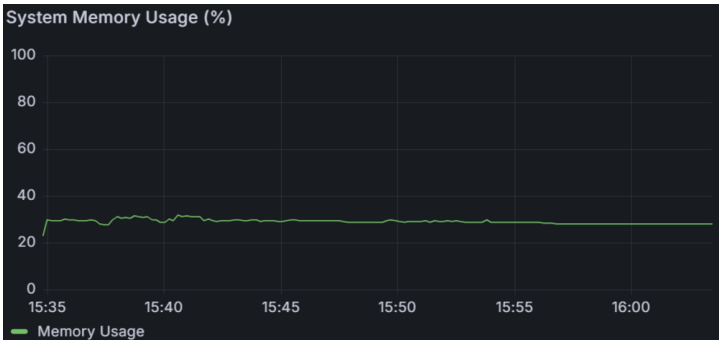
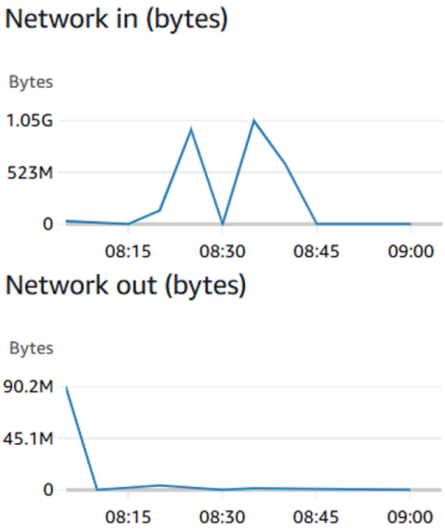
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV)	30.2	69 / 98	29 / 32		252,666	0.0142
Deep CDR	19.4	48 / 62	36 / 49		393,325	0.009
Proactive DLP	16.5	34 / 39	32 / 48		462,455	0.008
Vulnerability	12	50 / 65	46 / 46		635,875	0.006

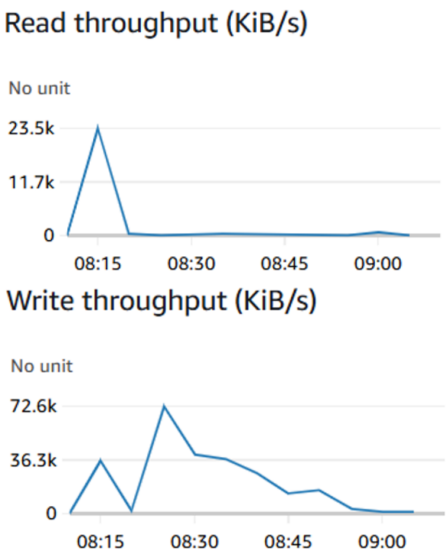
MetaDefender Core with common engine packages:

Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV) + Deep CDR	39.8	78 / 99	62 / 73		191,721	0.0188
Metascan (10 AV) + Deep CDR + Proactive DLP	43.5	61 / 84	47 / 67		175,414	0.0205
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	44.8	62 / 88	69 / 83		170,324	0.0211

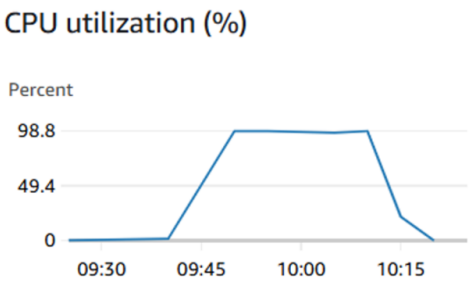
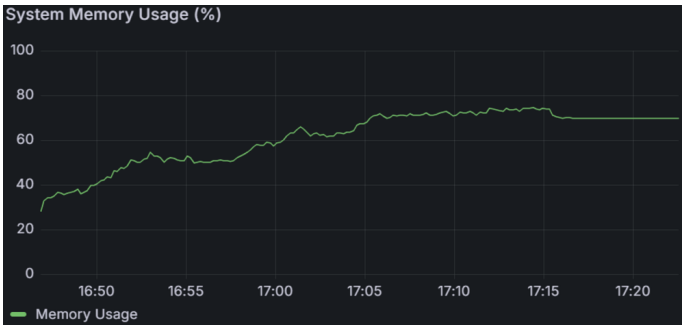
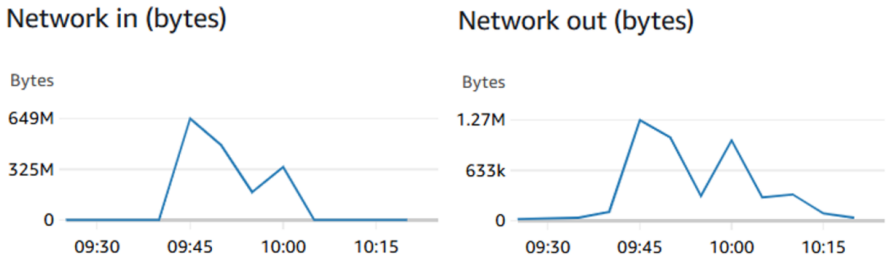
System resource utilization charts

MetaDefender Core with Metascan (10 AV engines) :

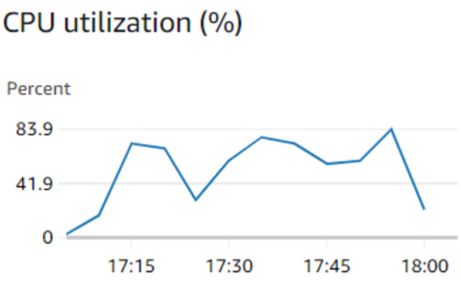
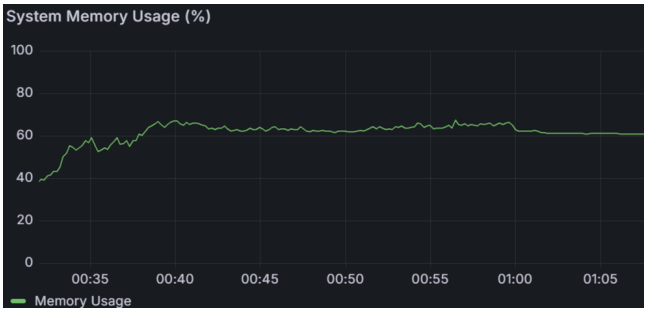
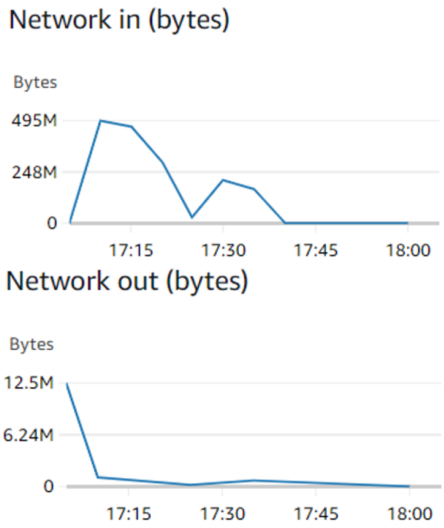
	Utilization charts
CPU usage	
Memory usage	
Network usage	

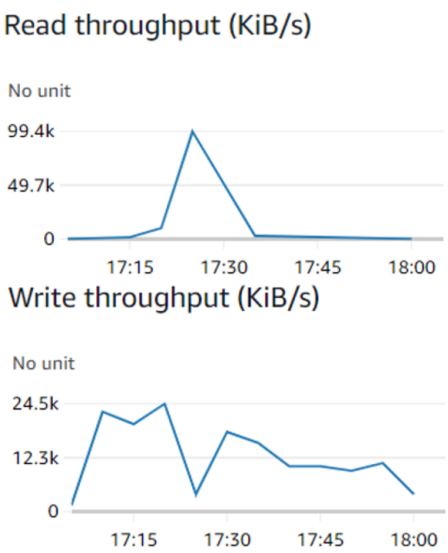
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR :

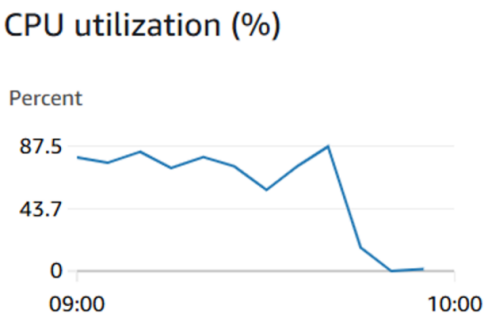
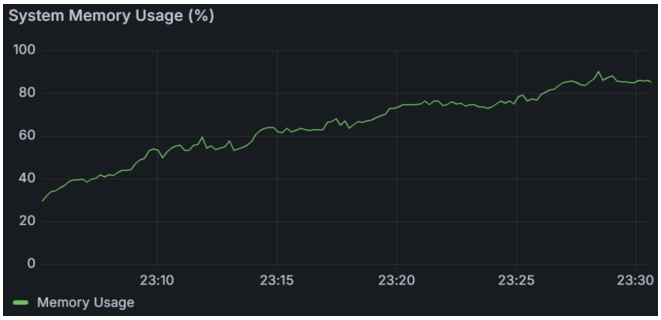
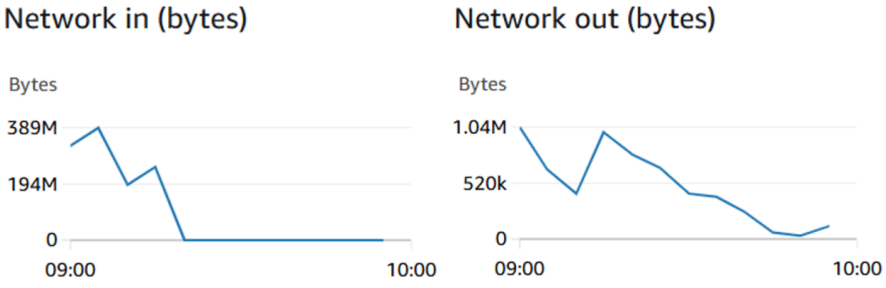
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP :

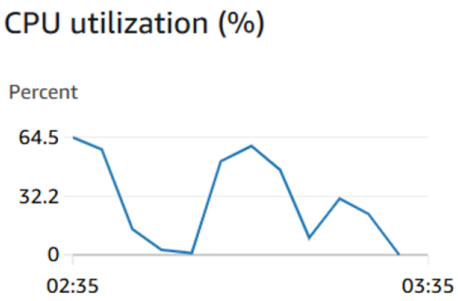
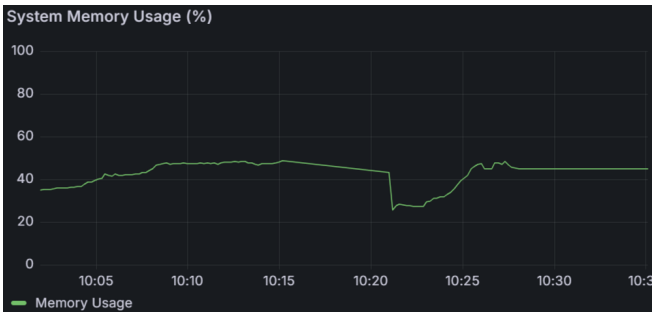
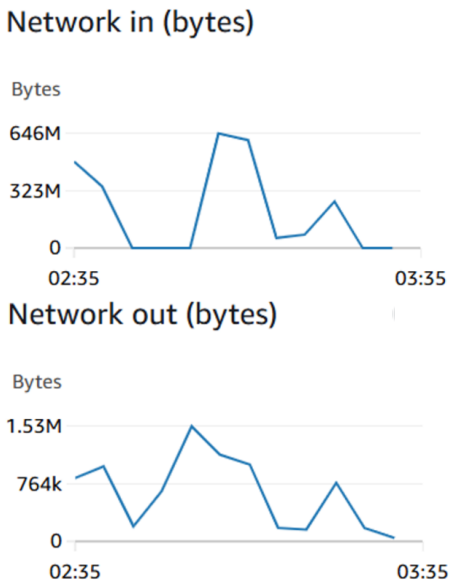
	Utilization charts
CPU usage	
Memory usage	
Network usage	

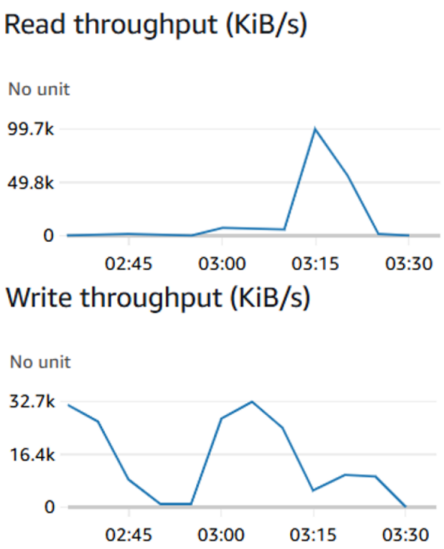
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP + Vulnerability :

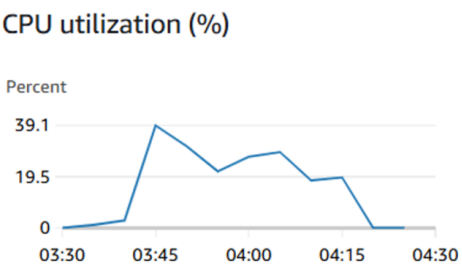
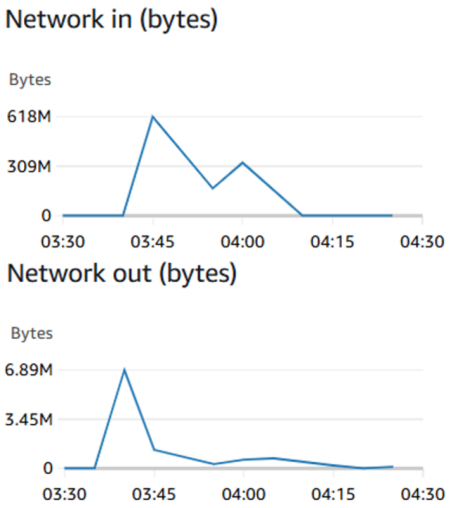
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

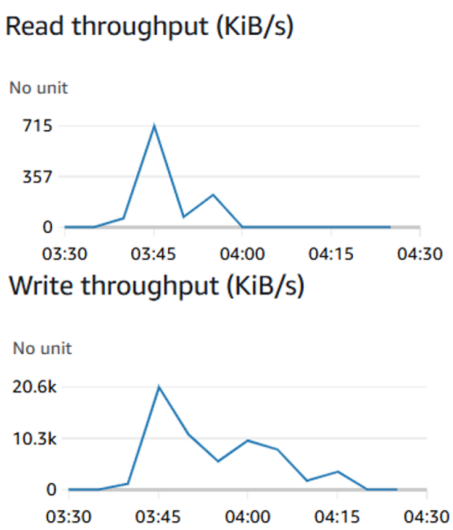
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

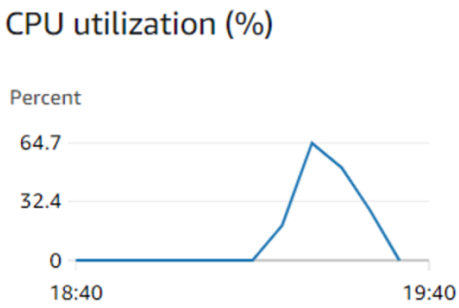
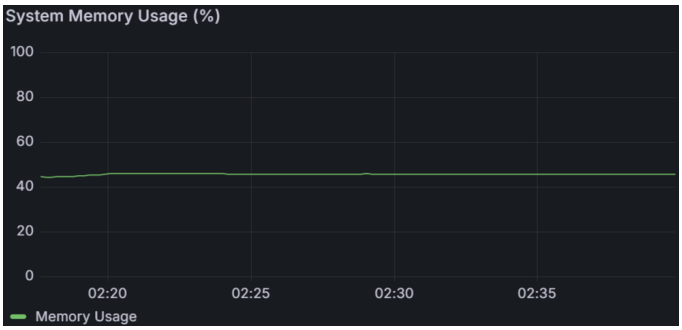
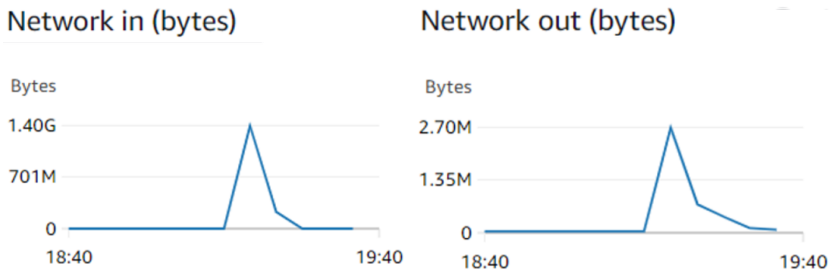
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability :

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

Shared DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

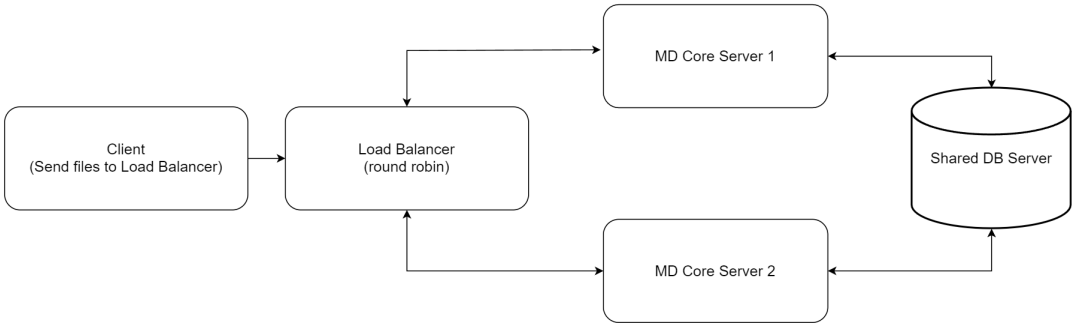
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV)	11.9	37.0	41.9	30.1	30.4	Receive: 6441.1 Send: 769.6	Receive: 6200.2 Send: 693.7	640,821.2	0.0
Deep CDR	10.5	60.1	66.7	30.3	32.2	Receive: 6379.8 Send: 594.8	Receive: 7146.1 Send: 609.6	729,368.5	0.0
Proactive DLP	7.8	38.7	37.5	29.7	31.0	Receive: 8842.6 Send: 791.8	Receive: 8635.6 Send: 729.1	965,455.7	0.0
Vulnerability	4.4	52.2	53.0	26.0	26.9	Receive: 16,215.5 Send: 1006.8	Receive: 15,785.7 Send: 997.2	1714,449.3	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV) + Deep CDR	15.4	72.4	75	34.3	35.4	Receive: 5436.6 Send: 611.3	Receive: 5589.3 Send: 607.2	495,697.4	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP	16.9	77.2	80.2	36.4	37.9	Receive: 5254 Send: 585.4	Receive: 5041.3 Send: 576.1	450,657.6	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	18.0	72.1	80.4	38.6	38.0	Receive: 4504.3 Send: 574.1	Receive: 4649.5 Send: 571.1	423,010	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

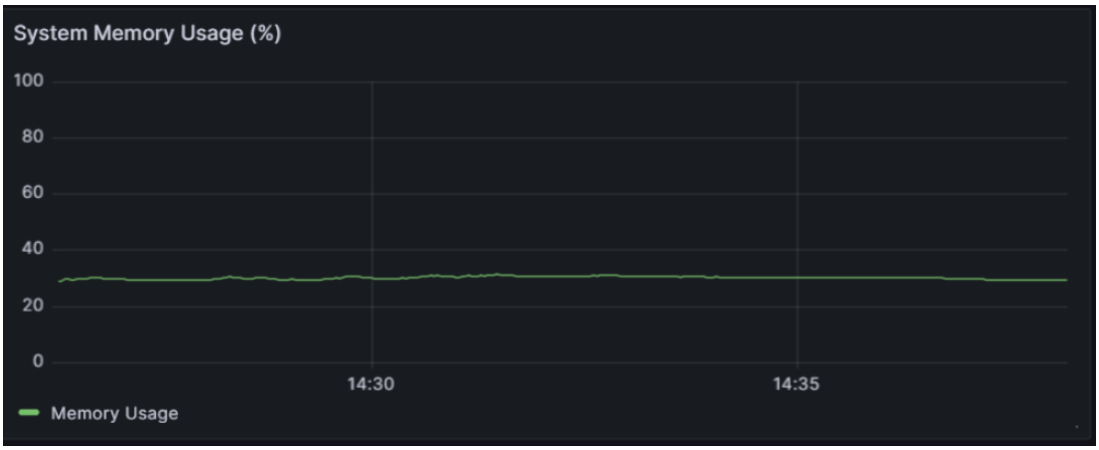
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2

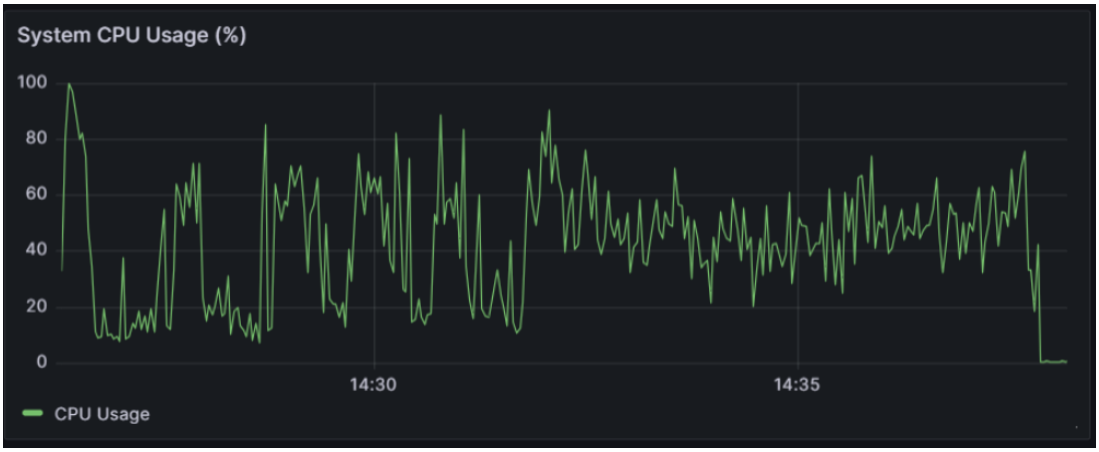


CPU usage

MD Core #1

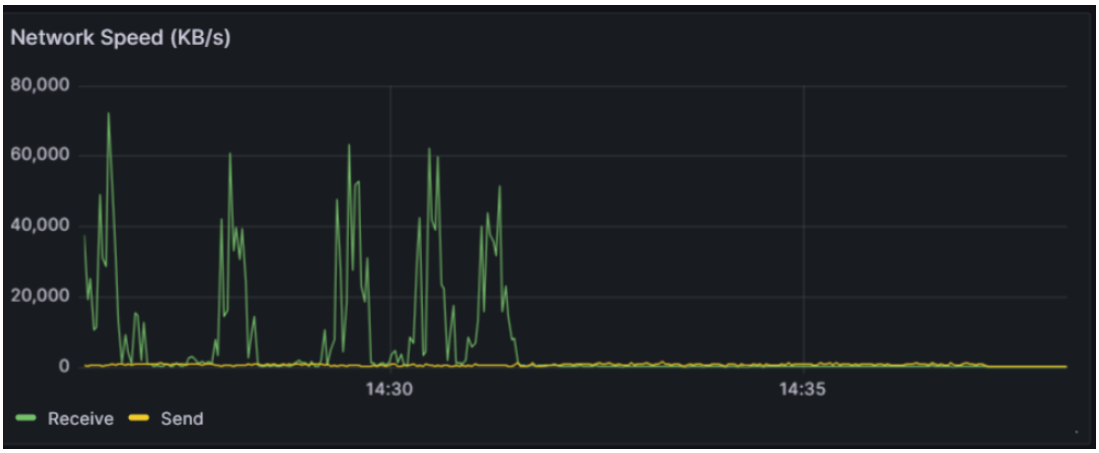


MD Core #2

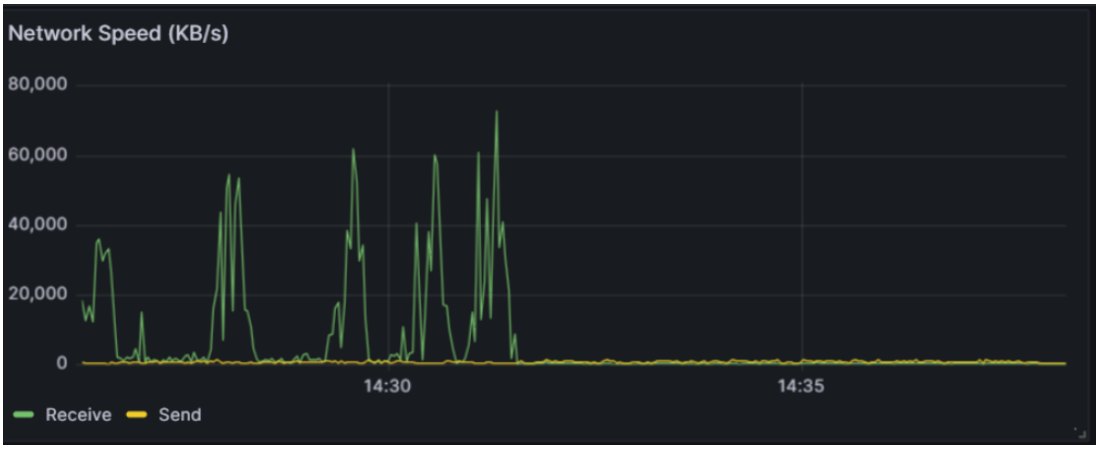


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2

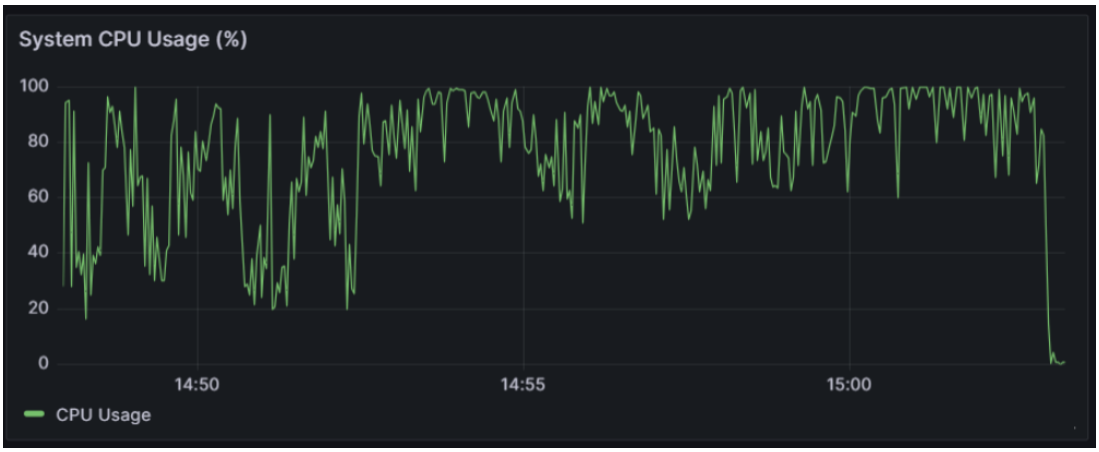


CPU usage

MD Core #1

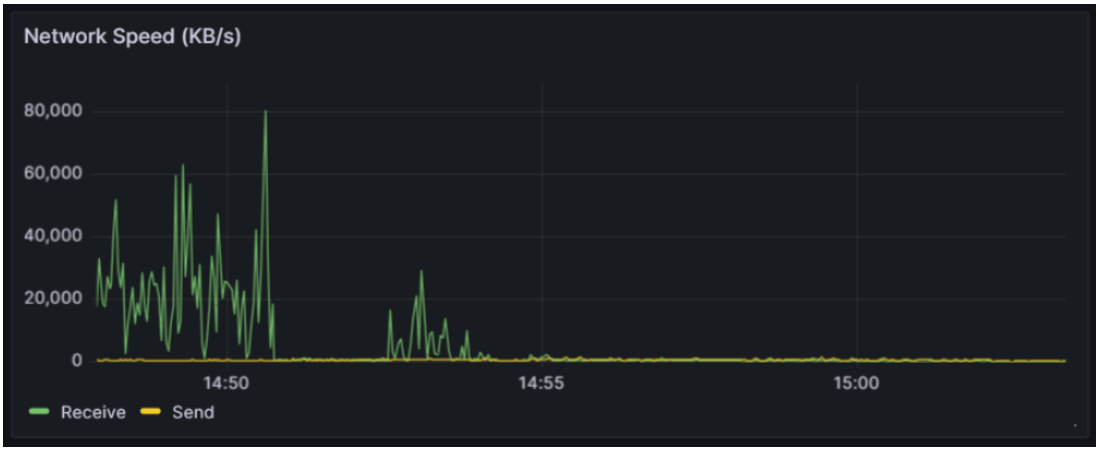


MD Core #2



Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

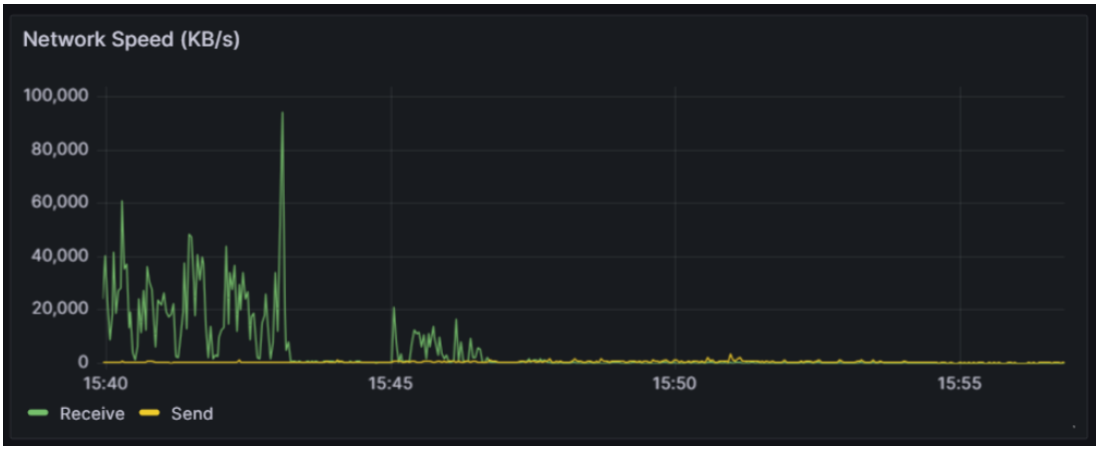


MD Core #2

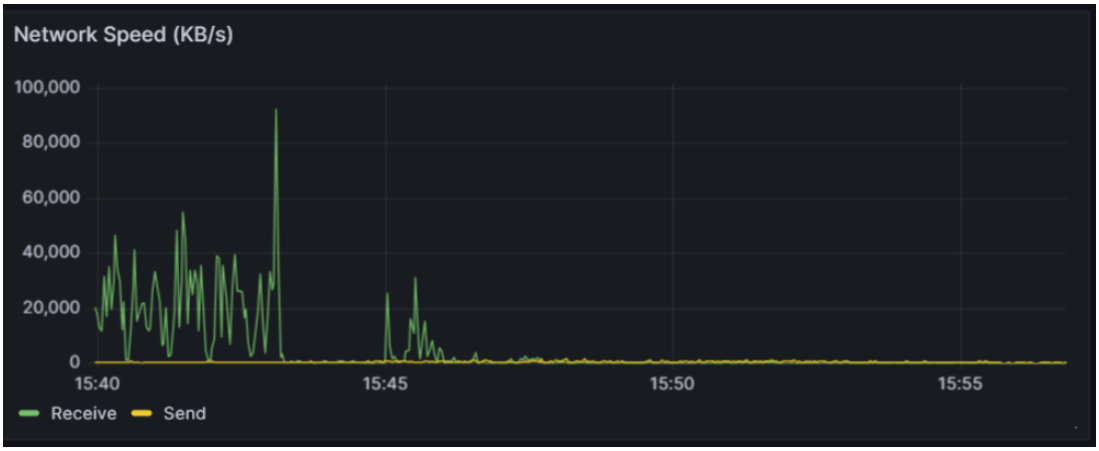


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1

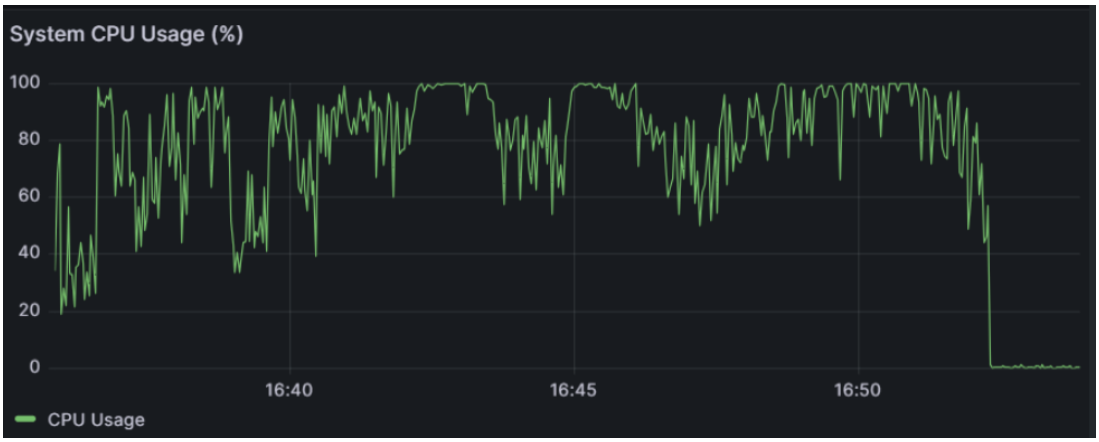


MD Core #2



CPU usage

MD Core #1

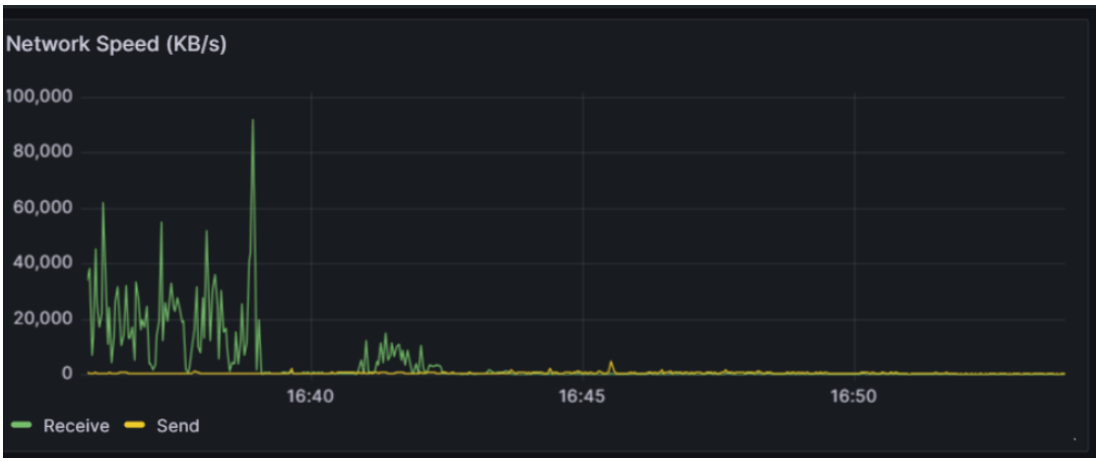


MD Core #2

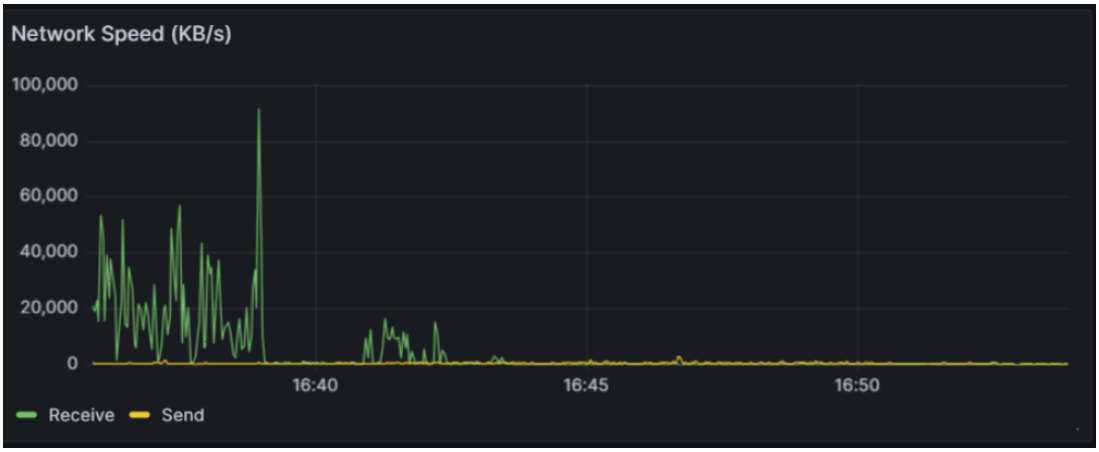


Network usage

MD Core #1



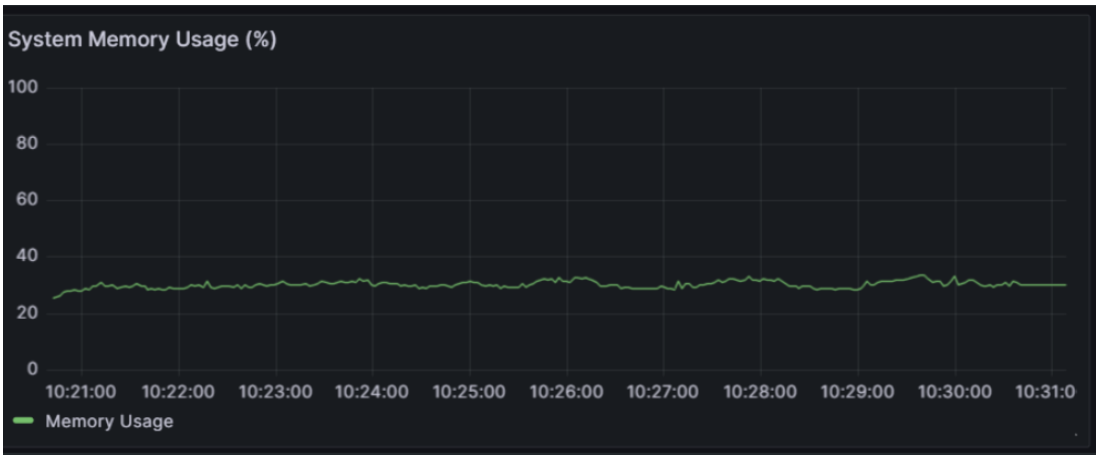
MD Core #2



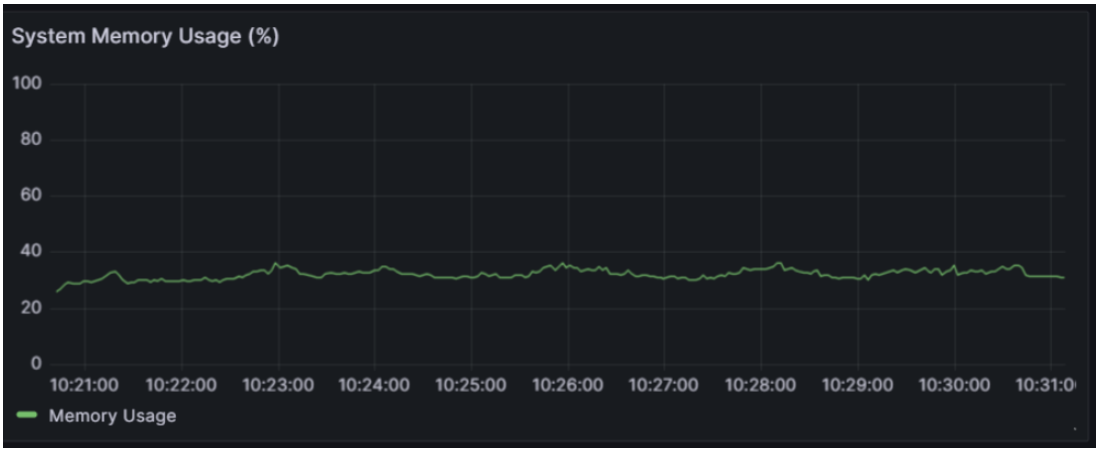
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

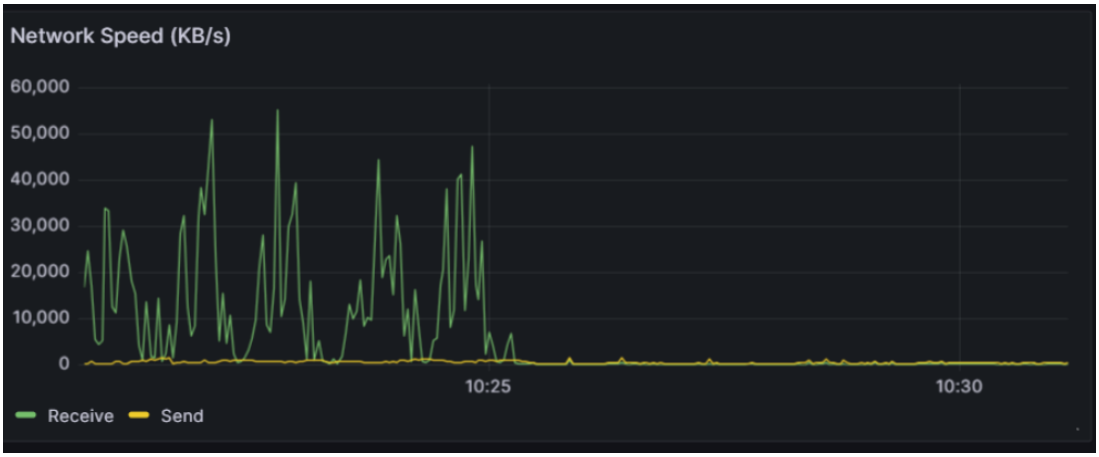


MD Core #2

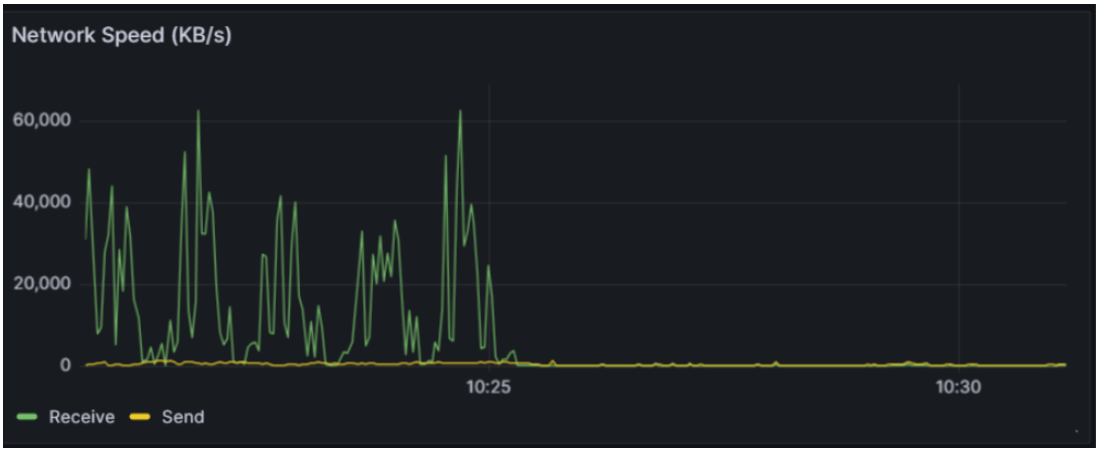


Network usage

MD Core #1



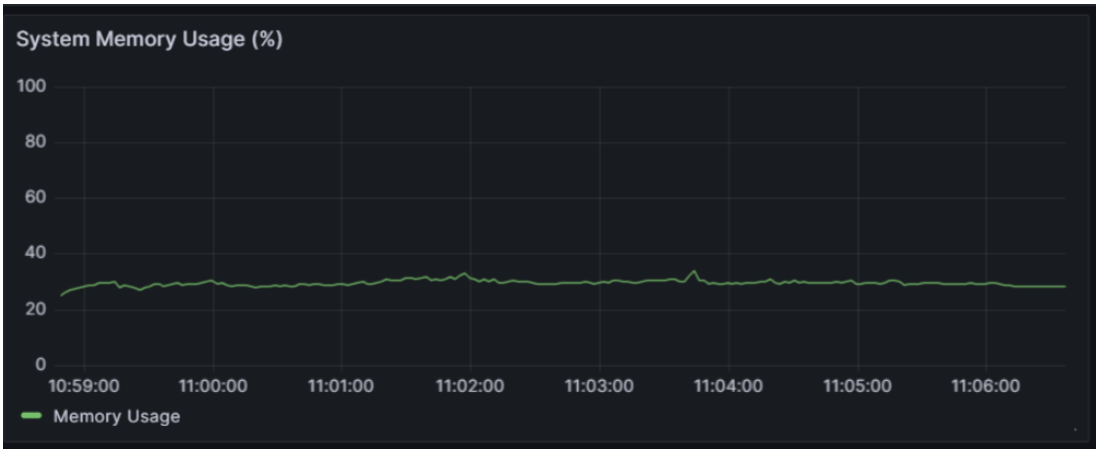
MD Core #2



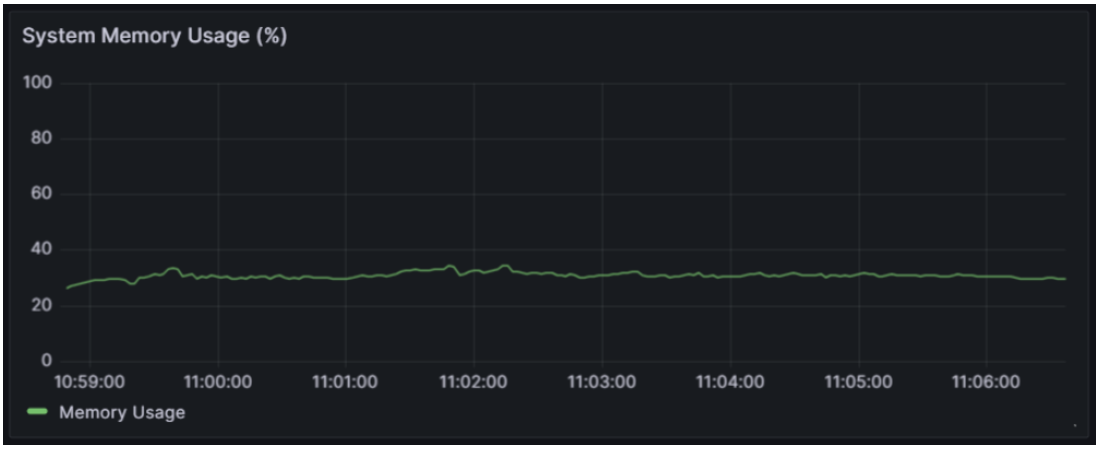
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

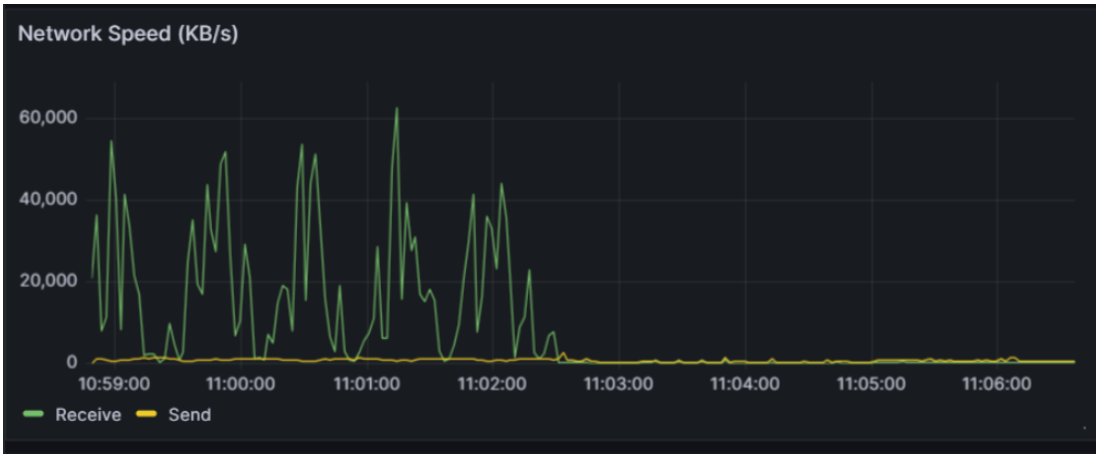


MD Core #2

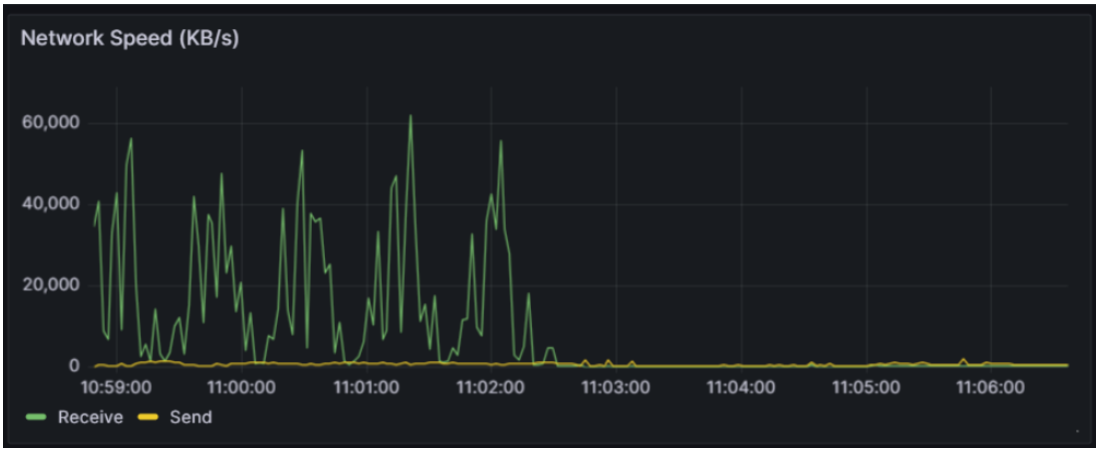


Network usage

MD Core #1



MD Core #2



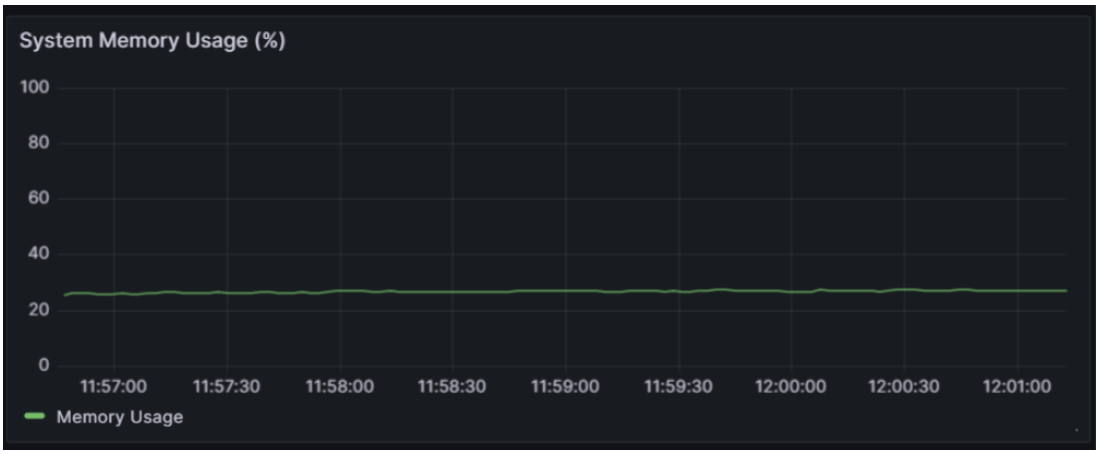
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

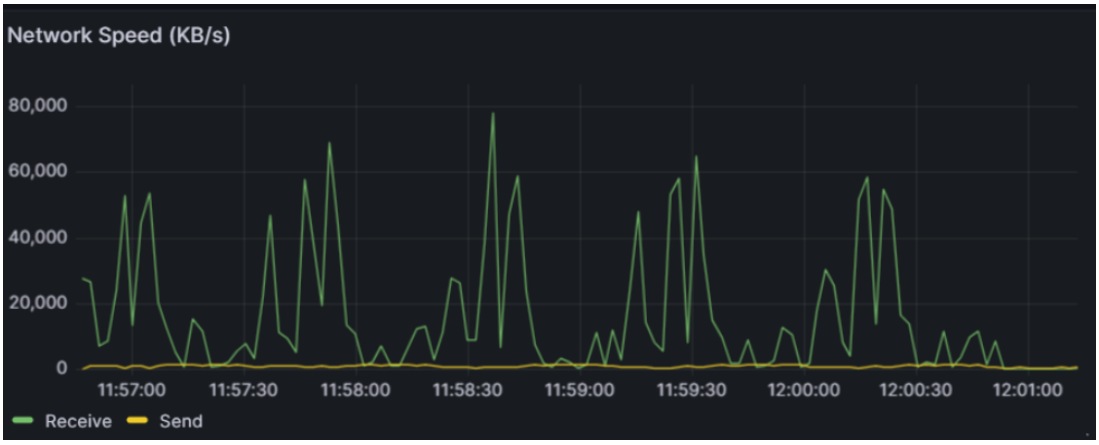


MD Core #2

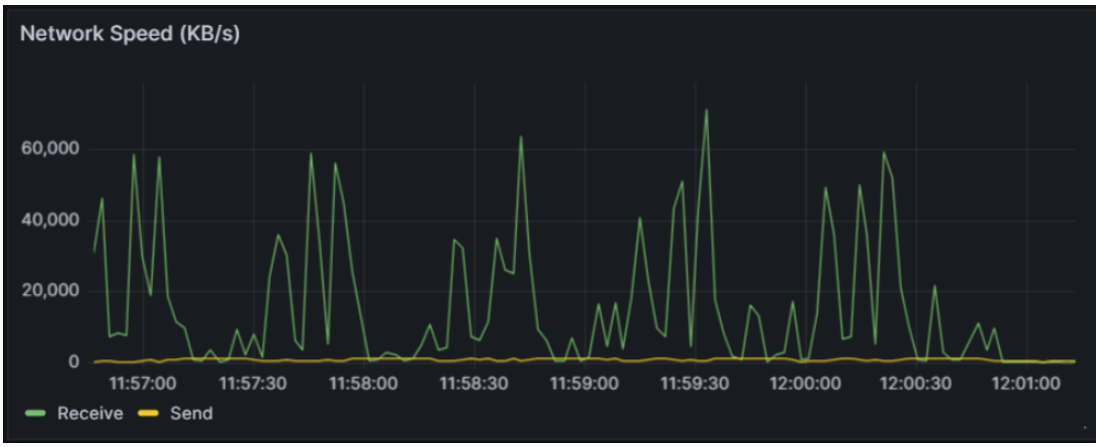


Network usage

MD Core #1



MD Core #2



Shared DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

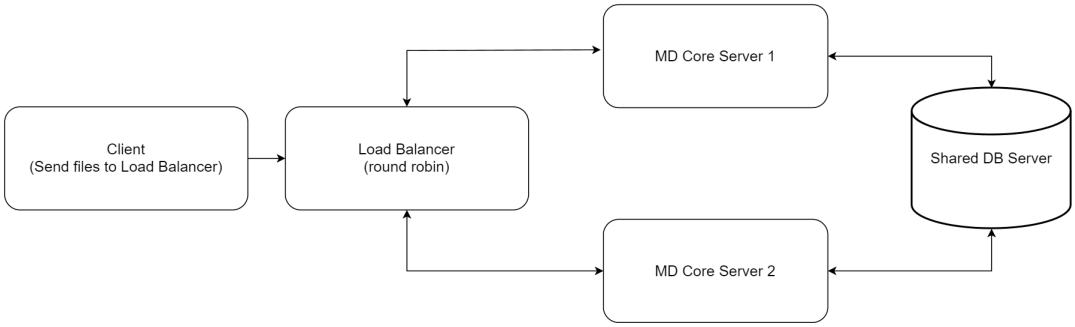
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV)	15.5	63.1	73.3	25.2	25.8	Receive: 4555.9 Send: 694.4	Receive: 4516.2 Send: 692.4	492,620.5	0.0
Deep CDR	9.70	50.7	51.2	31.9	32.5	Receive: 7062.9 Send: 623.2	Receive: 6653 Send: 654.1	786,189.6	0.0
Proactive DLP	7.2	35.2	35.0	27.9	27.5	Receive: 9047.4 Send: 802.4	Receive: 9859.8 Send: 787.5	1,055,110.8	0.0
Vulnerability	5.0	45.7	49.3	24.4	24.5	Receive: 13,585.7 Send: 966.7	Receive: 13,505.8 Send: 987.4	1,519,562.1	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV) + Deep CDR	19.9	79.1	77.6	34.0	34.2	Receive: 3785.9 Send: 573.8	Receive: 3663.1 Send: 577.7	382,716	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP	21.6	80.3	81.5	37.7	44.4	Receive: 3305.6 Send: 543.3	Receive: 3414.3 Send: 556.2	353,694.9	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	22.5	84.2	79.8	37.9	44.7	Receive: 3267.9 Send: 554.9	Receive: 3398.2 Send: 553.8	338,507.6	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

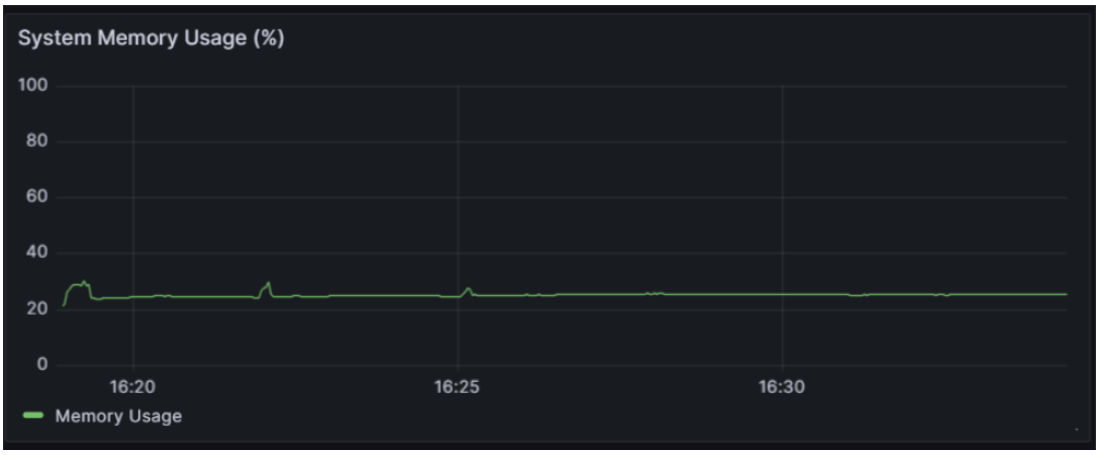
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

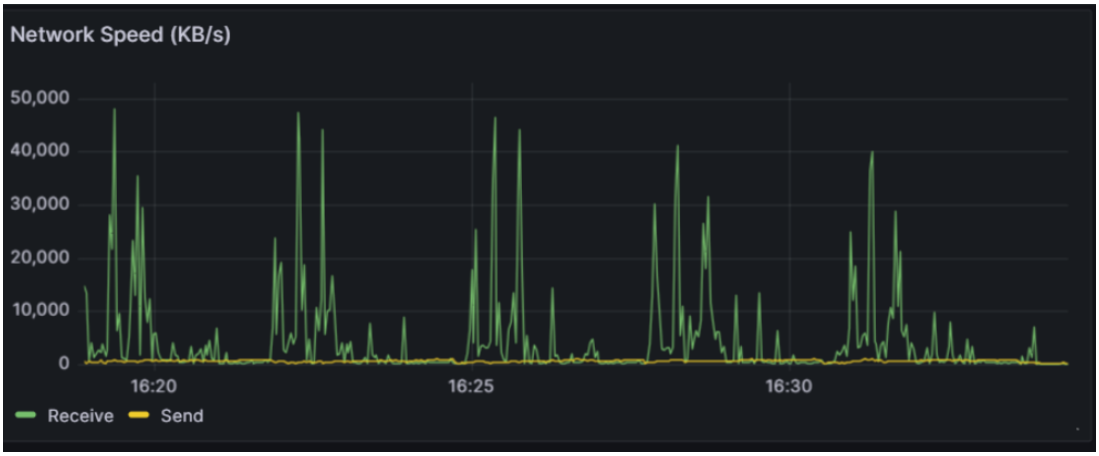


MD Core #2

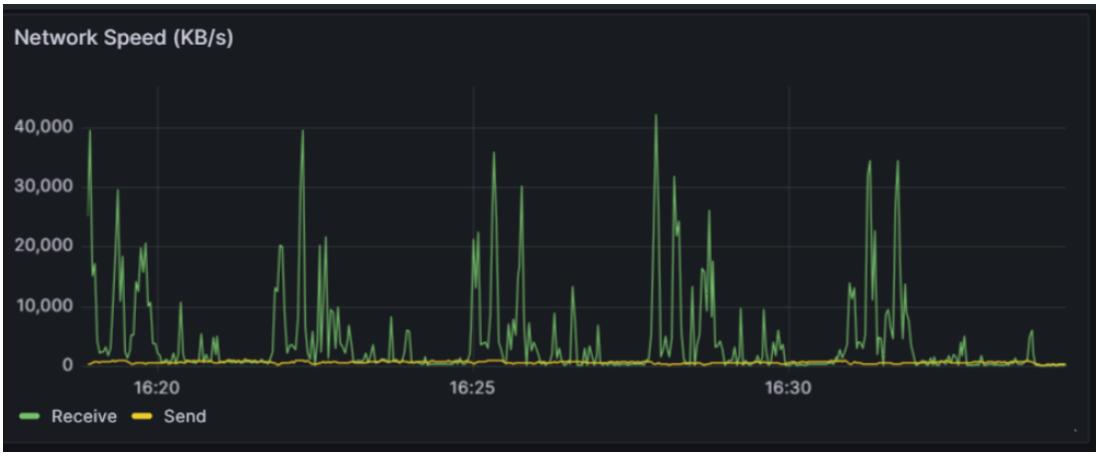


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

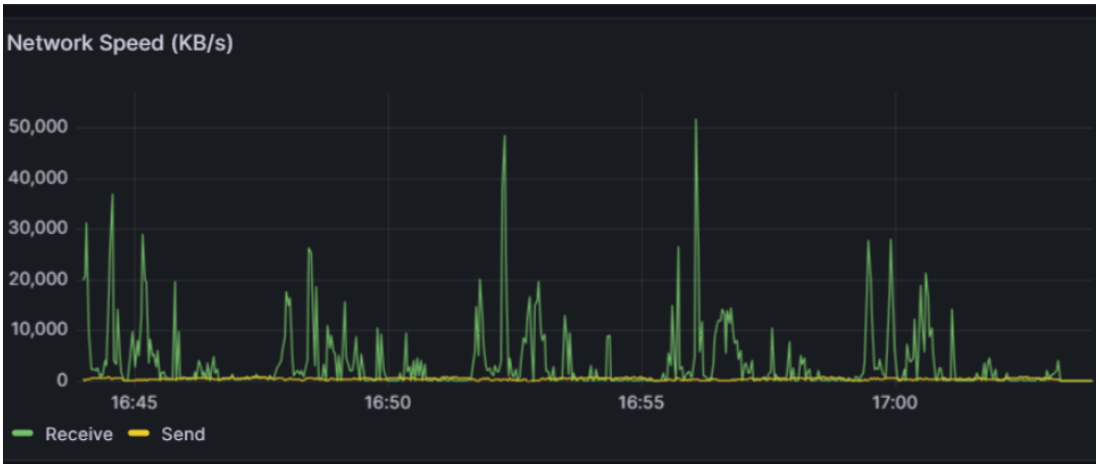


MD Core #2

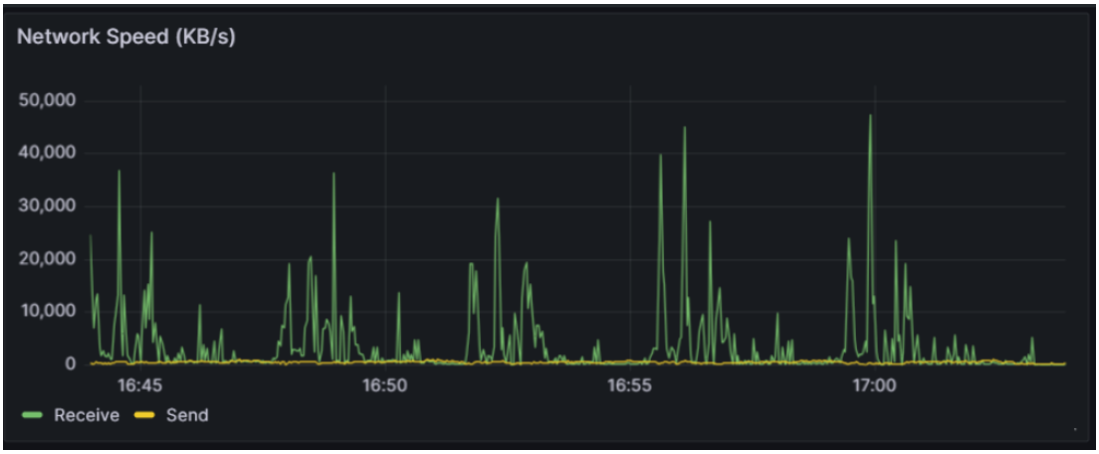


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1

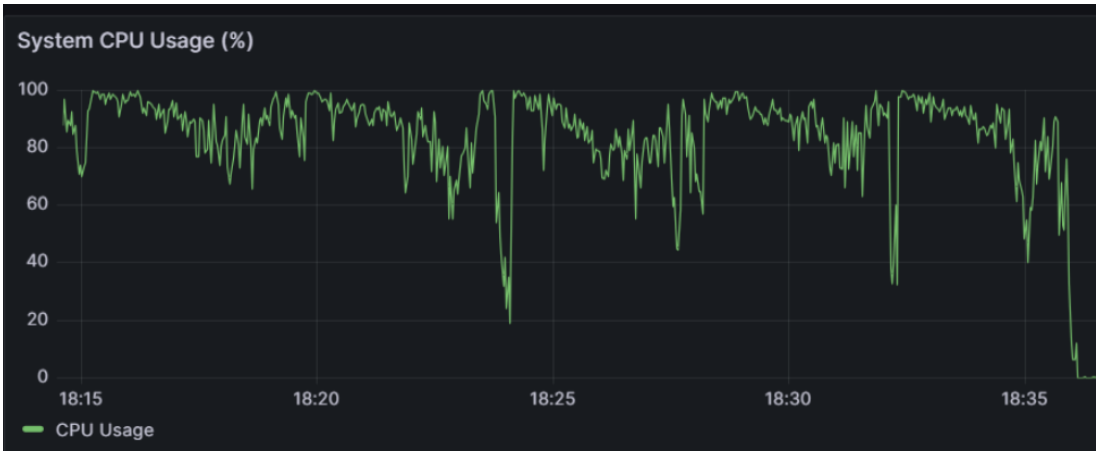


MD Core #2



CPU usage

MD Core #1

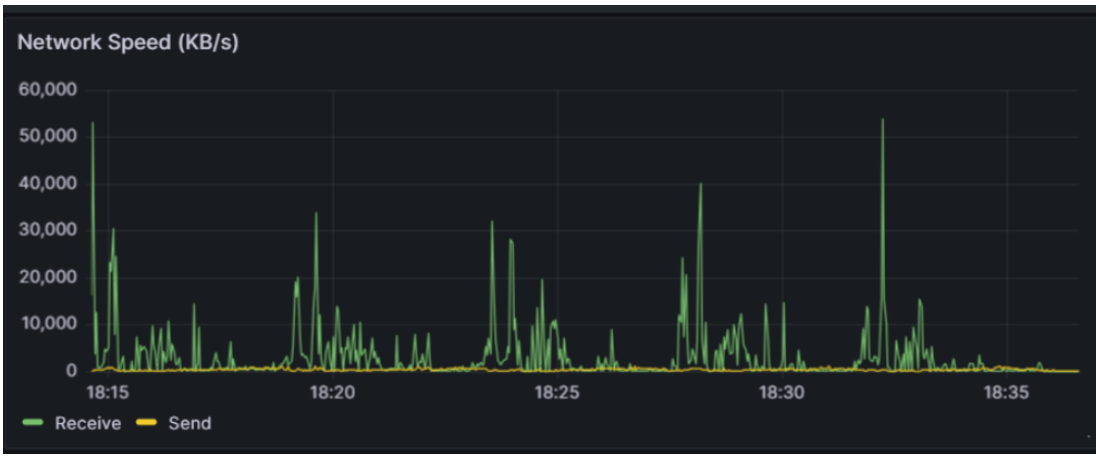


MD Core #2

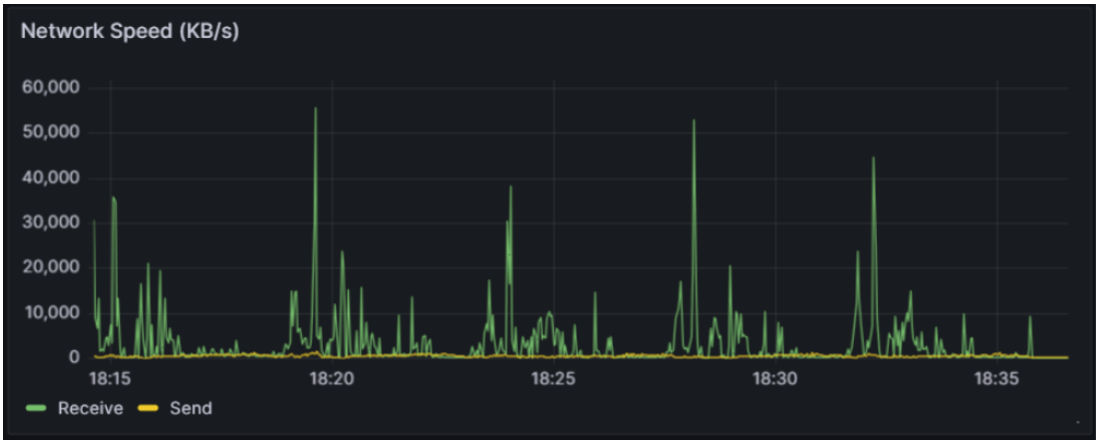


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

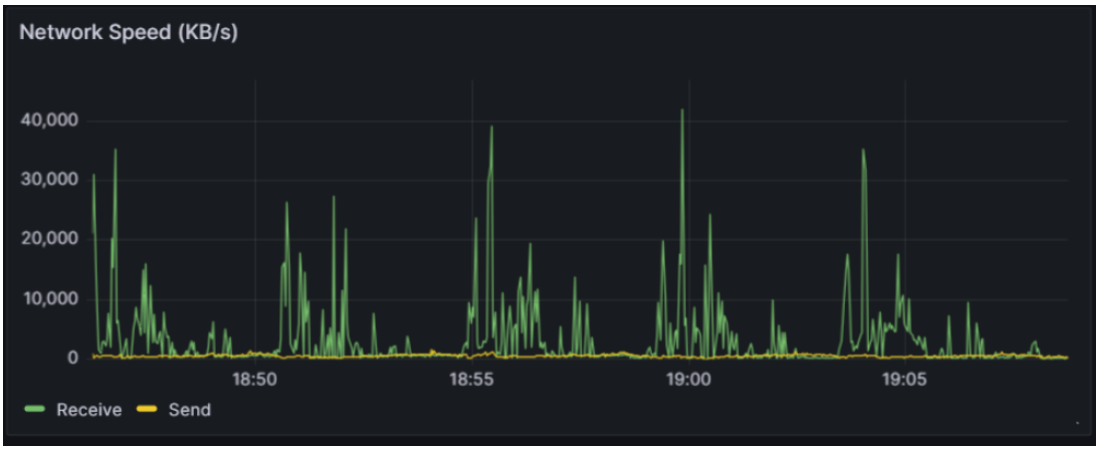


MD Core #2

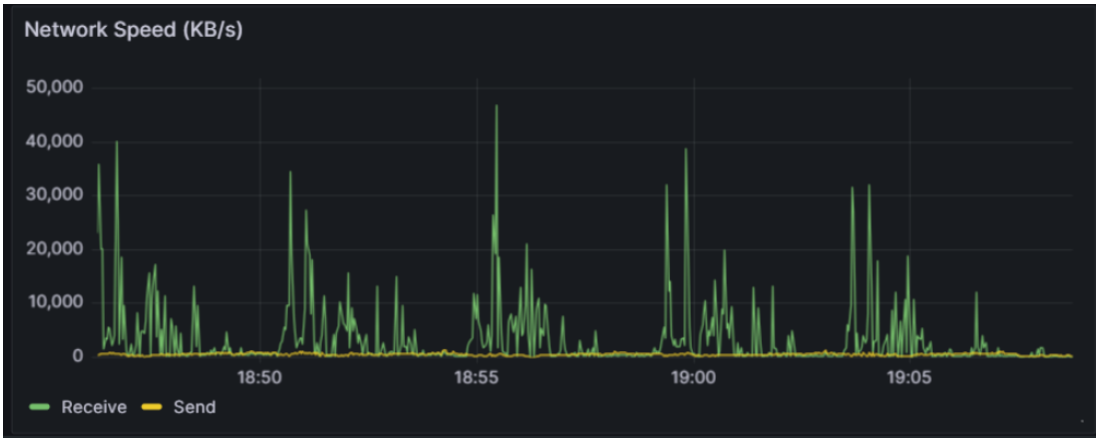


Network usage

MD Core #1



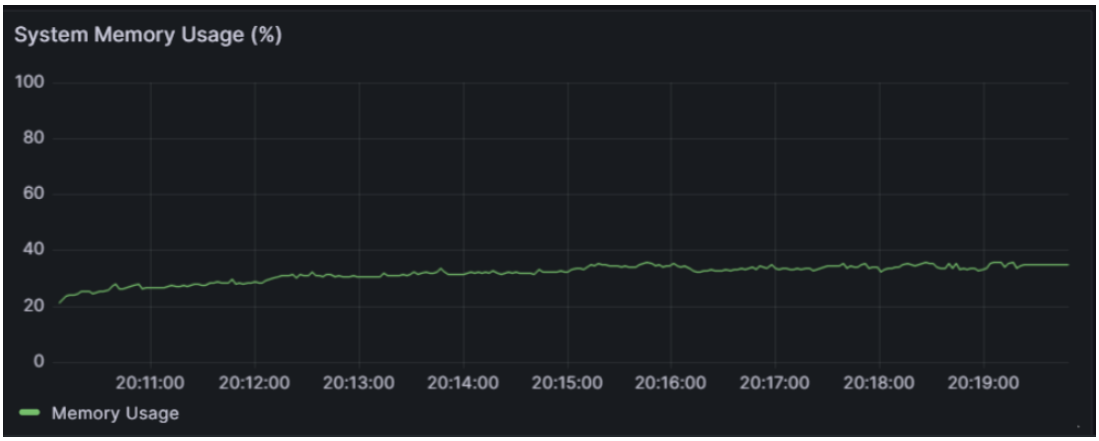
MD Core #2



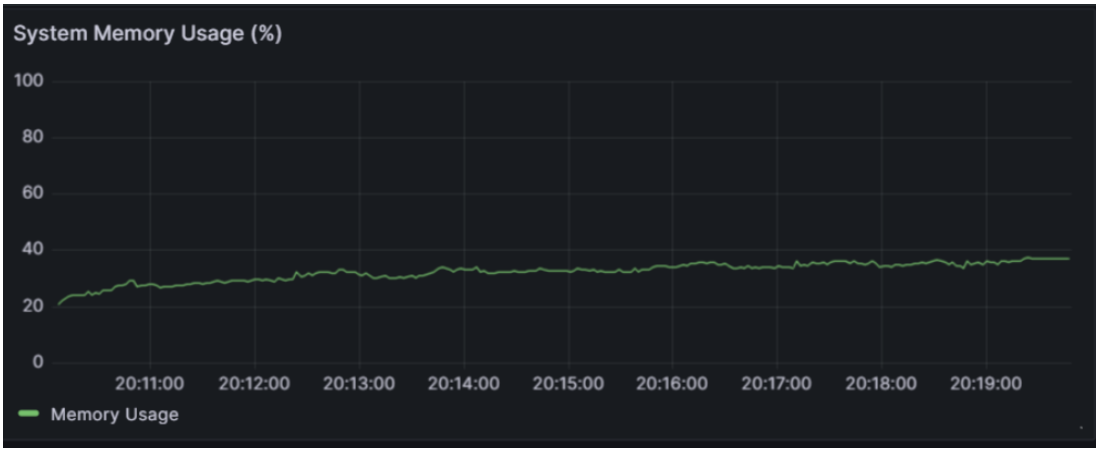
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1

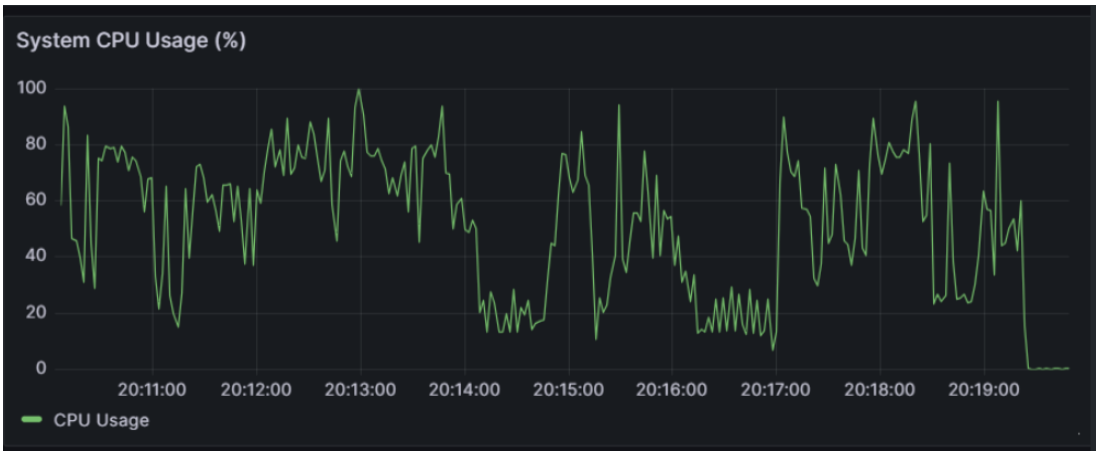


MD Core #2



CPU usage

MD Core #1

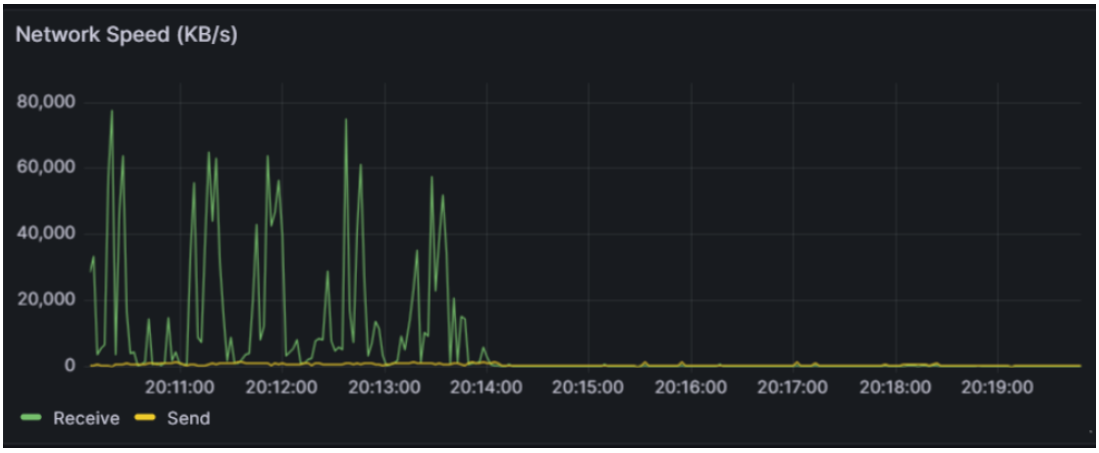


MD Core #2

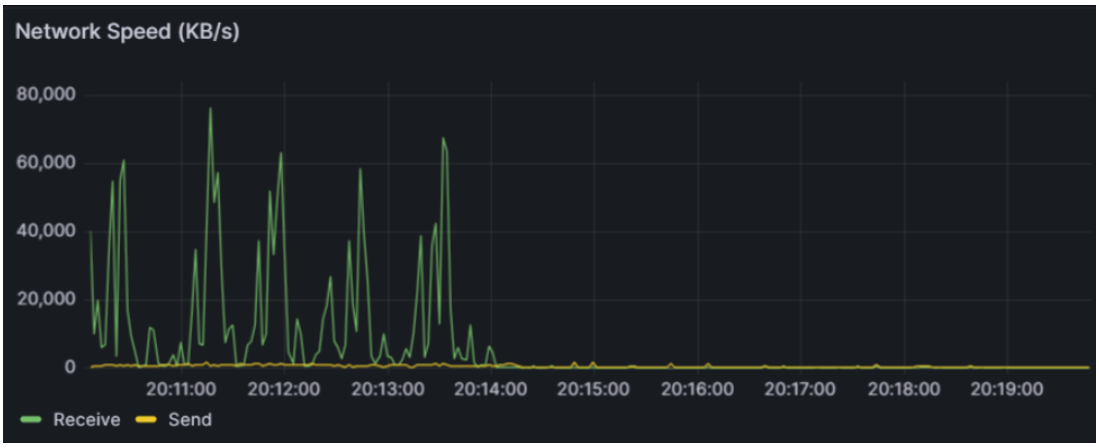


Network usage

MD Core #1



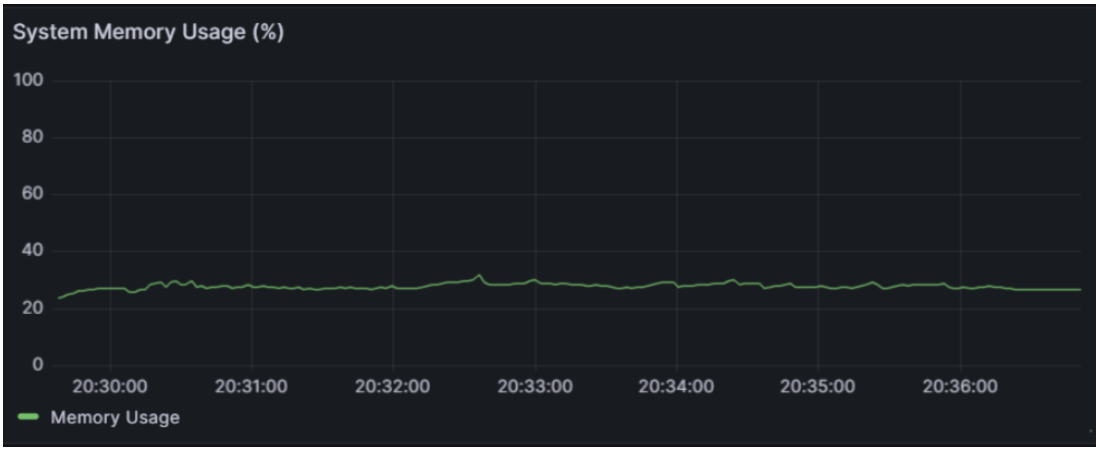
MD Core #2



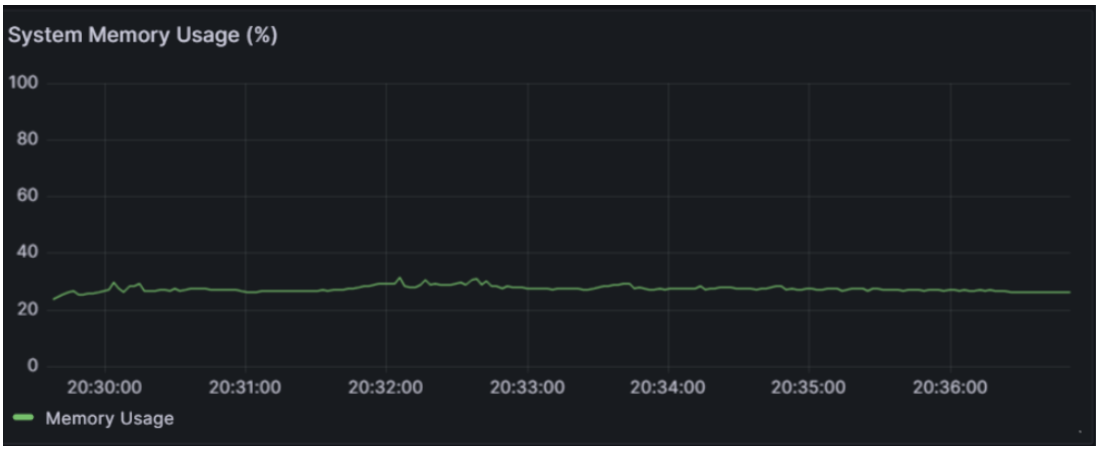
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1

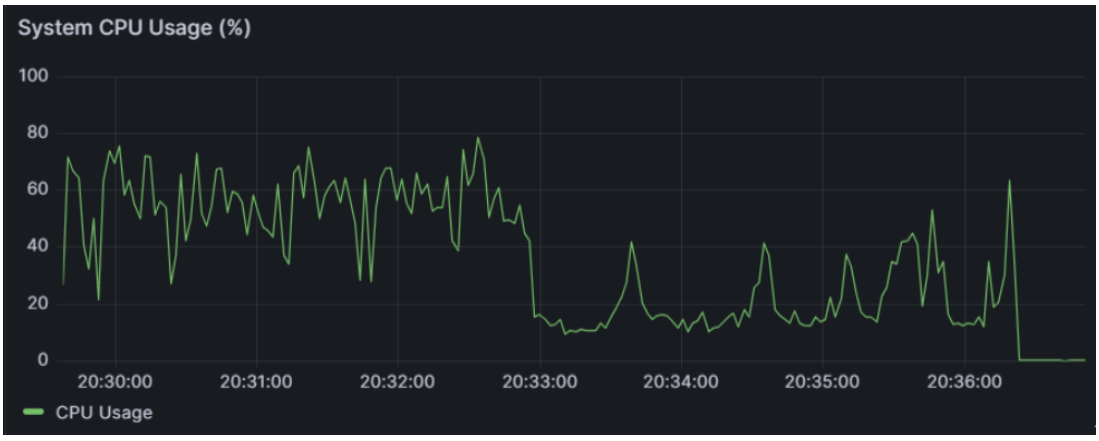


MD Core #2

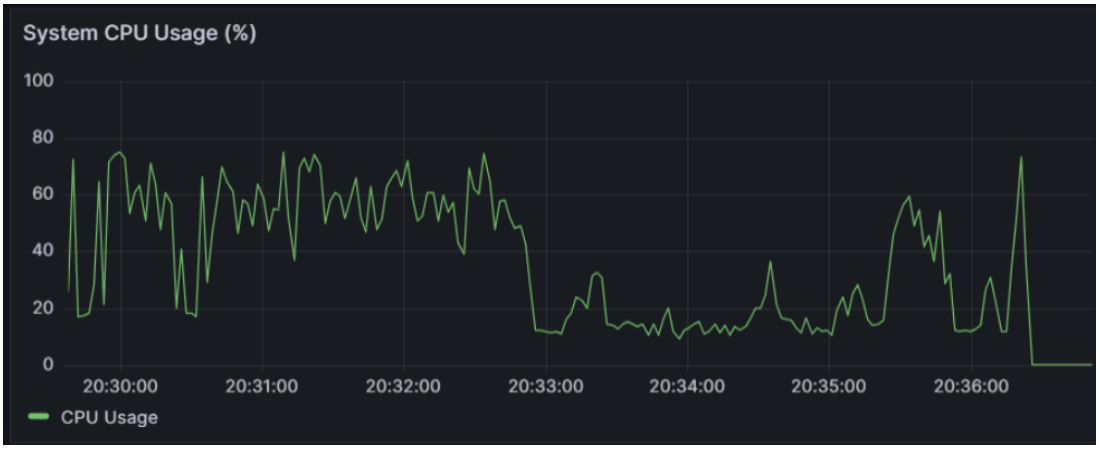


CPU usage

MD Core #1

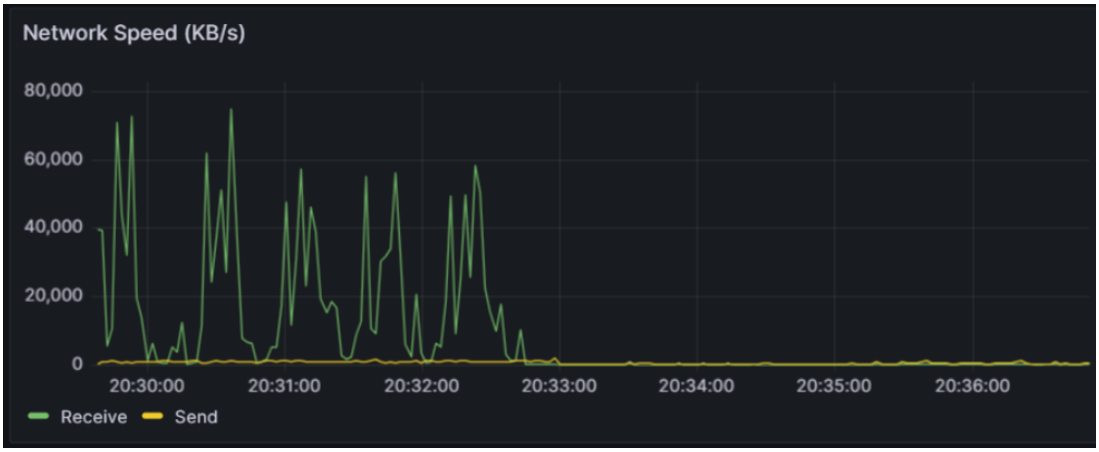


MD Core #2

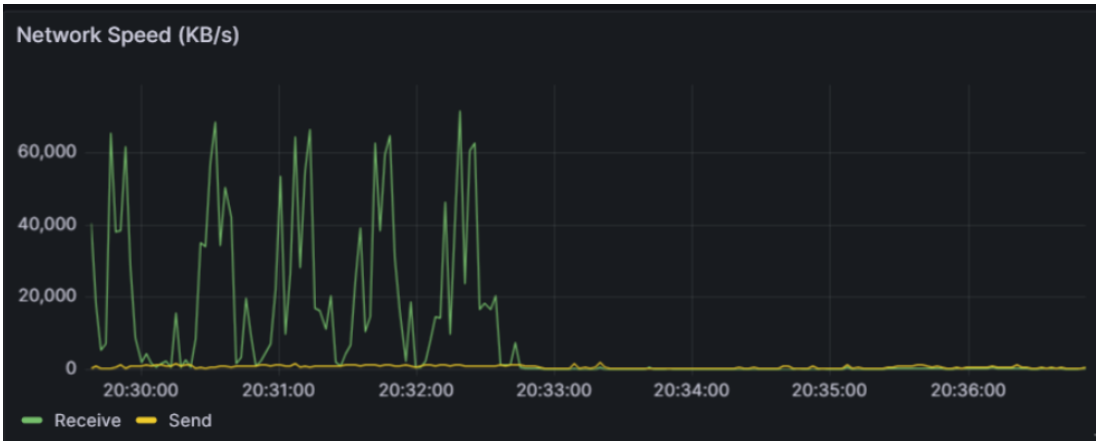


Network usage

MD Core #1



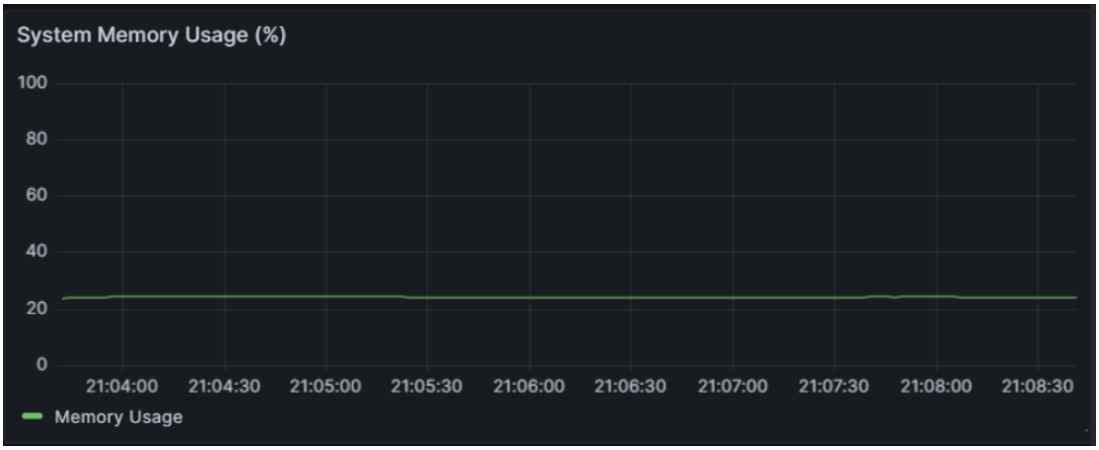
MD Core #2



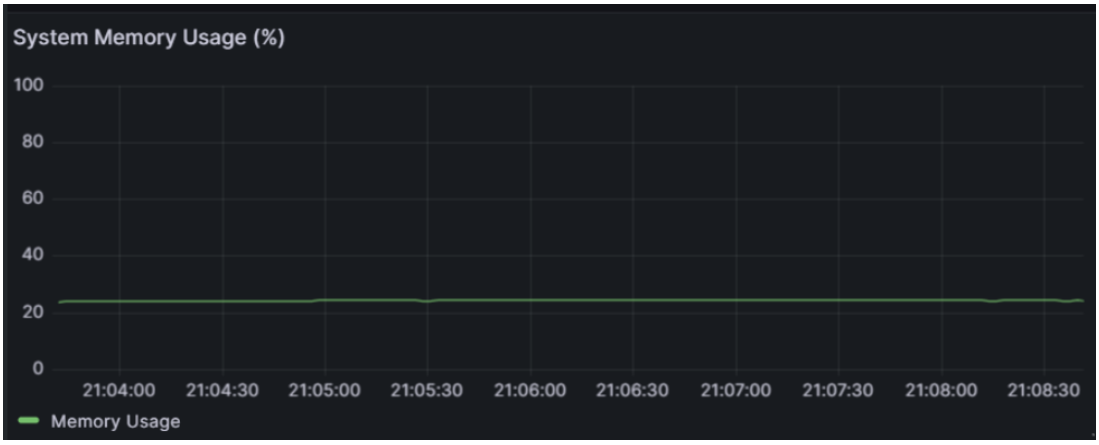
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

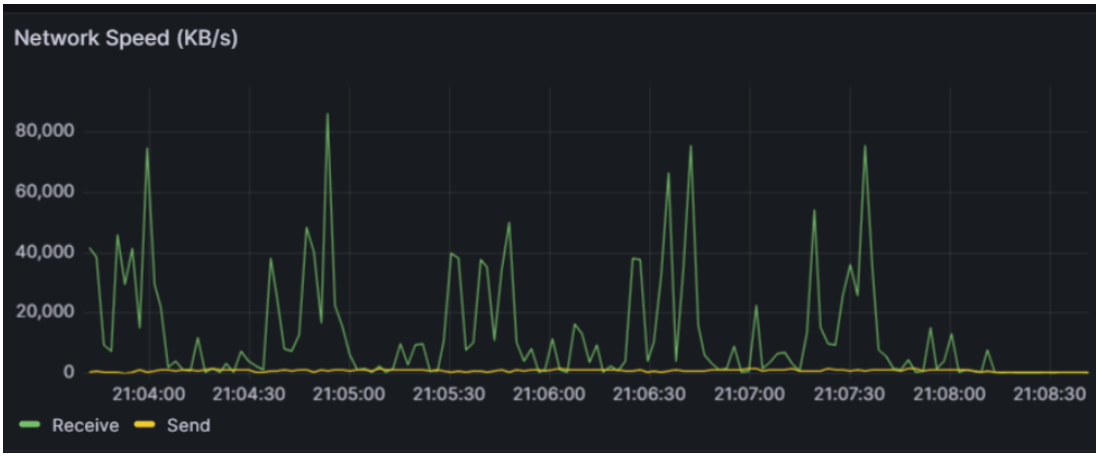


MD Core #2

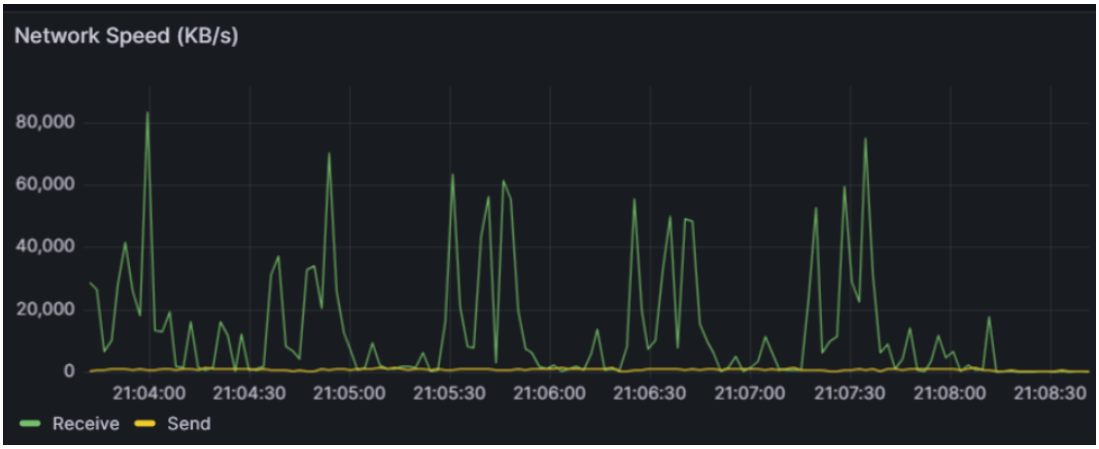


Network usage

MD Core #1



MD Core #2

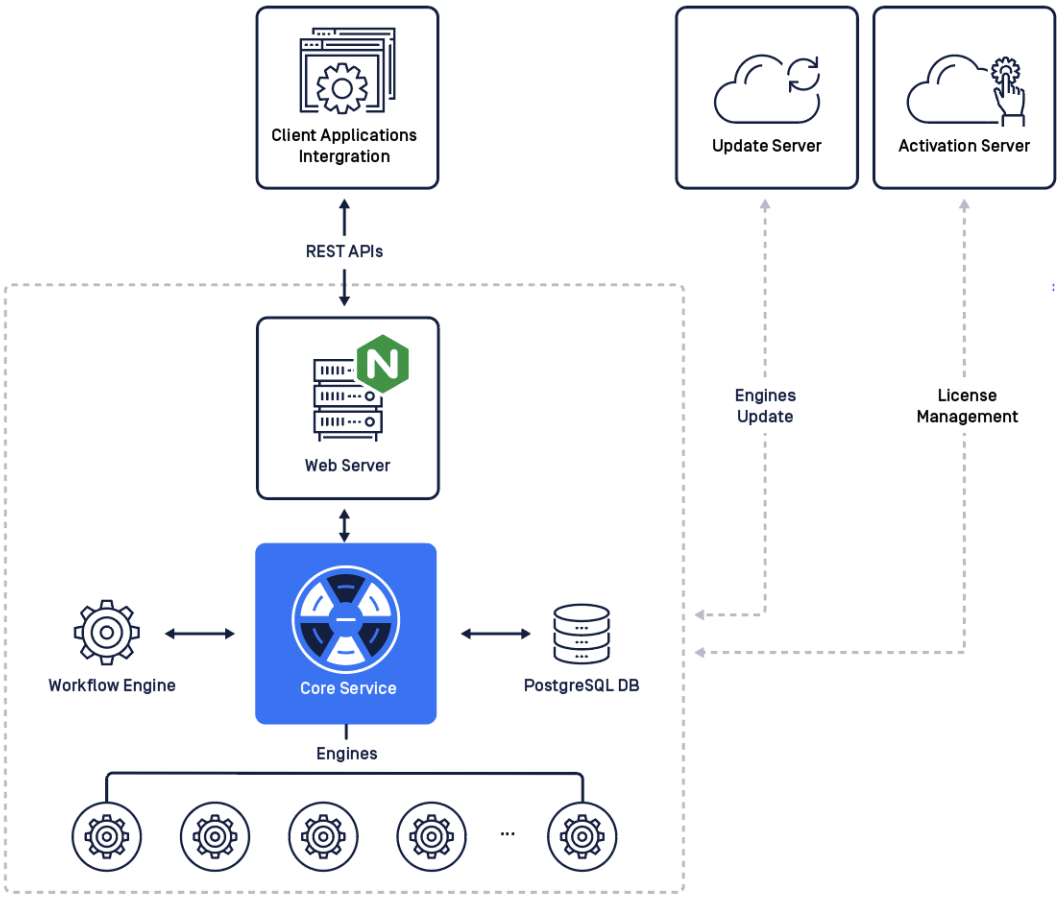

Network Speed (KB/s)
80,000
60,000
40,000
20,000
0
21:04:00
21:04:30
21:05:00
21:05:30
21:06:00
21:06:30
21:07:00
21:07:30
21:08:00
21:08:30
Receive
Send

MetaDefender Core Documentation

The users can consult this web page or, alternatively, they can download the manual in pdf format from the link below:

MetaDefender Core manual (SHA256:).

Recommended System Configuration



MetaDefender Core supports a wide range of deployment options, including air-gapped environments, providing the flexibility to choose the environment that best suits customers' requirements and infrastructure.

- Physical server.
- Virtualization platforms like VMware, Hyper-V and XenServer.
- Cloud Deployment.
- Container Deployment.

> **Note**
> Before installing MetaDefender Core, please refer to the **recommended minimum system configuration** listed below. The requirements are based on a standardized customer deployment and may require more or less resources to achieve the level of performance desired in your environment. Please note that the server specifications are built to allow a high volume daily processing. Please refer to Performance and Load Estimation for more details.

The recommendations below are for MetaDefender Core, API usage only. For any other use cases, please consult the user guide of the licensed products for accurate recommendations.

For certain use cases these might be adjusted and customized on specific needs and SLAs. We highly recommend to engage our ProServ team to assist in fine tuning MetaDefender and get the maximum performance out of your systems.

Microsoft Windows Deployments

[Windows] Supported Operating Systems

Only support 64-bit platforms.

- Windows 10, Windows 11 23H2, 11 IoT (22H2) only for End-User systems (e.g. Kiosk deployments)
  - Based on Microsoft License Terms, you're not allowed to use Windows Desktop as a server, therefore, MetaDefender Core can't be used on these systems for server use cases.
- Windows Server 2016, 2019, 2022

> **Disclaimer**
> End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

> **Plan for legacy OS support**
> OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:
>
> - Ubuntu 20.04 after **April 2025**.
> - Amazon Linux 2 after **June 2025**.
> - Windows 10 after **October 2025**.
>
> If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

[Windows] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 8	8	8 GB	16 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 12	16	16 GB	24 GB + 50 GB + 100 GB
MetaDefender Core 16	16	16 GB	32 GB + 50 GB + 100 GB
MetaDefender Core 20	32	16 GB	40 GB + 50 GB + 100 GB
MetaDefender Core MAX	32	32 GB	120 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core is installed.
- For a better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.
- SBOM processing requires more disk space, please allow at least 15GB free disk for the engine to work properly.

[Windows] Third Party Dependencies

- .NET framework 4.5 or above
- Microsoft Visual C++ Redistributable for Visual Studio 2015-2022 (version 14.38 or higher); x64 version.
- PostgreSQL: Only applicable when using a pre-installed PostgreSQL server running remotely.
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Some engines also have dependencies as described below:

Engine Name	Dependency
Deep CDR	Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 5.8.0 or above)
ESET	MetaDefender Core v4 temporary directory should have more than 200MB free disk space
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
Proactive DLP	• Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 2.0 or above) • .NET 6 dependencies • CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
Scrutiny	Microsoft Visual C++ 2015 - 2019 Redistributable x86 and x64
Systweak	.NET runtime 6.0, 7.0 or 8.0
Bkav Pro	Microsoft Visual C++ 2015 - 2022 Redistributable x64

[Windows] Installation Details

MetaDefender Core on Windows uses "C:\Program Files\OPSWAT" folder for storing resources or the installation directory.

MetaDefender will use its resources folder to store temp files as part of the analysis. **It's recommended to exclude this folder from real-time protection monitoring**. Knowledge base:

- How to set up exclusions on anti-malware software to prevent disruption
- Mitigating scan failure triggered by real-time protection service
- Can local AVs interrupt ongoing scans?

Linux Deployments

[Linux] Supported Operating Systems

Only support 64-bit platforms.

- Red Hat Enterprise Linux 8, 9
- Rocky Linux 9
- Debian 11, 12
- Ubuntu 20.04, 22.04
- Amazon Linux 2 is supported via Amazon Machine Image (AMI)

Plan for legacy OS support

OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:

- Ubuntu 20.04 after **April 2025**.
- Amazon Linux 2 after **June 2025**.
- Windows 10 after **October 2025**.

If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

For Hardended Linux OS

Some AV engines could be failed to initialize and run on MetaDefender Core (e.g. ESET, Kaspersky) due to execution permission required by them on `/var/tmp/ometascan` folder.

Please follow steps at Why have the ESET and Kaspersky scan engines failed to initialize on the hardened Linux OS? for remediation details.

Disclaimer

End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

[Linux] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 5	4	4 GB	10 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 10	8	8 GB	20 GB + 50 GB + 100 GB
MetaDefender Core MAX	16	16 GB	40 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core resides on.
- For better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.

[Linux] Third Party Dependencies

Dependencies list:

- openssl
- grep
- lib32stdc++6 (>= 4.5)
- libc6-i386 (>= 2.10)
- procps
- zlib1g
- libcurl3 (>= 7.19.7)
- libcurl4
- postgresql:
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Not all the above dependencies will need to be installed, it is dependent on different Unix distro and version

Some engines also have dependencies as described below:

Engine name	Dependency
Archive engine	• To extract password protected MS Office files, you need to install .NET 6 dependencies, libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • Rocky Linux 9.x/Red Hat 9.x: libnsl
Deep CDR	• Rocky Linux 9.x: libgomp, ncurses-compat-libs, libnsl • CentOS: libgomp, ncurses-compat-libs • Red Hat 8.x: libgomp, ncurses-compat-libs, libnsl • Red Hat 9.x: libgomp, ncurses-compat-libs, libnsl, compat-openssl11 • Ubuntu/Debian: libgomp1, libncurses5 • libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • .NET 6 dependencies
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
FileType engine	Rocky Linux 9.x/Red Hat 9.x: libnsl
Proactive DLP	libgdiplus, mscorefonts .NET 6 dependencies CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
RocketCyber	libgomp

Note: Some dependencies may need to be installed from external repositories, not from the OS default repositories.

[Linux] Installation Details

MetaDefender Core default installation path is using /var folder for storing resources:

- `/var/lib/ometascan` : installation folder with all its resources (database, DLP processed / CDR sanitized / quarantined file storage, engine package and definition updates).
- `/var/log/ometascan` : application logs.
- `/var/run/ometascan` : runtime folder for Core related processes.
- `/var/lib/ometascan` : all successful deployed engines data files.
- `/usr/lib/ometascan` : all shared libraries required by MetaDefender Core, bundled PostgreSQL server related binaries and libraries.
- `/var/tmp/ometascan/resources` : all temporary processing files for MetaDefender Core to process.

Storage Usage Information

Based on the configuration, MetaDefender Core will need additional disk space to store analysis/result data, quarantined files and temporary files:

	Data Storage
Analysis Reports (Processing History)	Approximately 4.5GB is required for every 1M analysis reports stored. • 100 GB is recommended as a baseline. Database storage requirements are affected by Data Retention settings, engine settings and scanning load.
Quarantined Files	Depends on the customers' dataset and configuration
Sanitized Files	Depends on the customers' dataset and configuration
Temporary processing files	Depends on the customers' dataset and configuration • A 1GB non-archive file requires 1GB free disk space at least. • A 1GB archive file requires (1 * 2)GB free disk space at least. • 50GB is a baseline and should be adjusted based on observation of the solution within the environment, with consideration to peak times and overall file makeup (e.g. complexity, size, archives).
Note	Options are available to control how large and/or complex files are handled to limit the amount of system resources required. See Workflow Configuration Template for more information.

Custom Engines

The recommendations above are specific for MetaDefender pre-packaged bundles.

However for additional Custom Engines, please review the Knowledge base to review additional requirements (if any) for the selected engine.

Browser Requirements for the MetaDefender Core Management Console

One of the following browsers is suggested to view the MetaDefender Core Management Console:

- Microsoft Edge
- Chrome
- Firefox
- Safari

Chrome, Firefox, Safari and Microsoft Edge browsers are tested with the latest available version at the time of release.

Network Bandwidth

In online environment where MetaDefender Core is configured to download engine update packages directly from OPSWAT update server source, then using gigabit network speed (125 MB/sec) is highly recommended to avoid potential network related issue while downloading files, or at least 100 Mbps (12.5 MB/sec) network speed.

Allowlisting URLs

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts':

- https://activation.dl.opswat.com - for license online activation
- https://update.dl.opswat.com - for engines and databases download

Note: *IP address-based allowlisting on your firewall might fail after some time since OPSWAT uses CDN (AWS Cloudfront) to faster delivery updates over the world, and IP address of edge servers might change over time.*

Depending on your location, connections with OPSWAT cloud-based servers be forwarded to specific AWS Cloudfront's edge locations, each is assigned with its own unique IP address ranges. You might also need to allow correct AWS Cloudfront's corresponding IP address ranges by referring to https://ip-ranges.amazonaws.com/ip-ranges.json

Warning

Even the OPSWAT update servers host updates for all of the available engines we support, sometimes custom engines might try to connect to their own cloud for updates, but this can be disabled in firewall and they will be updated just from OPSWAT cloud servers instead.

Differences Between MetaDefender Core v4 and v5

We are very excited to inform you about the release of version 5.0.0 of our MetaDefender Core!

The new version is designed to enhance the product performance, user experience, as well as to provide more flexibility, security, and scalability for our customers.

At the same time, we are also committed to keeping backward compatibility for whom running MetaDefender Core v4, and avoid the need for updating your client integration to move forward with MetaDefender Core v5.

v4 REST API Integration Supported
All public REST API endpoints available on MetaDefender Core v4 will be continously supported on v5.

v4 Product Major Features Supported
All product major features available on MetaDefender Core 4.21.2 (the latest one on v4) will be continously supported on v5.

Seamless Product Upgrade
Upgrading from MetaDefender Core v4 to v5 will be supported and covered with data migration without additional manual steps from users.

Licensing information will be also migrated.

Below are some highlights of the first v5 release, MetaDefender Core 5.0.0:

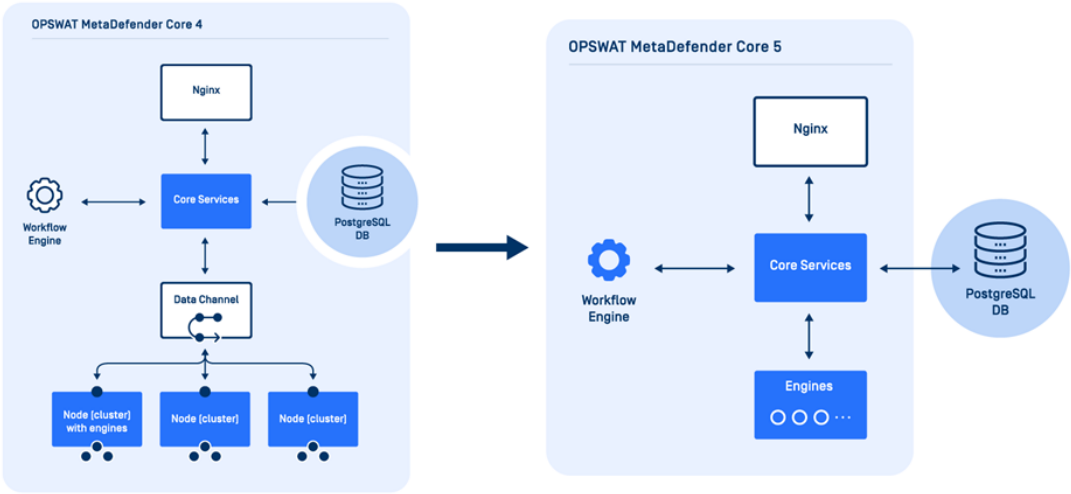
New product architecture

This version introduces a simpler product architecture that speeds up integration and automation, file processing flow, and immensely enhance the product performance and efficiency.

Earlier versions utilized MetaDefender Node concept, designed to support scaling of the scanning infrastructure by distributing scan requests among several different processing Nodes. Product licensing was based on number of Node instances. The benefit of having such a distributed infrastructure is that based on Node loads, Metadefender Core server can always choose the most appropriate Node to assign a new scan task to. However, having nodes in the v4 architecture caused more complexity and possible latency in the processing flow and entailed maintenance cost.

In this release, we retired multi-Node deployment and restructured MetaDefender Core's backend. This alteration significantly improves file processing speed and product efficiency, as well as eliminates extra cost for maintaining node services. Users now can simply install additional MetaDefender Core instances for scalability requirement,

as well as configure their own load balancer to coordinate tasks among multiple MetaDefender Core instances. Accordingly, product licensing is now based on number of the product instances.



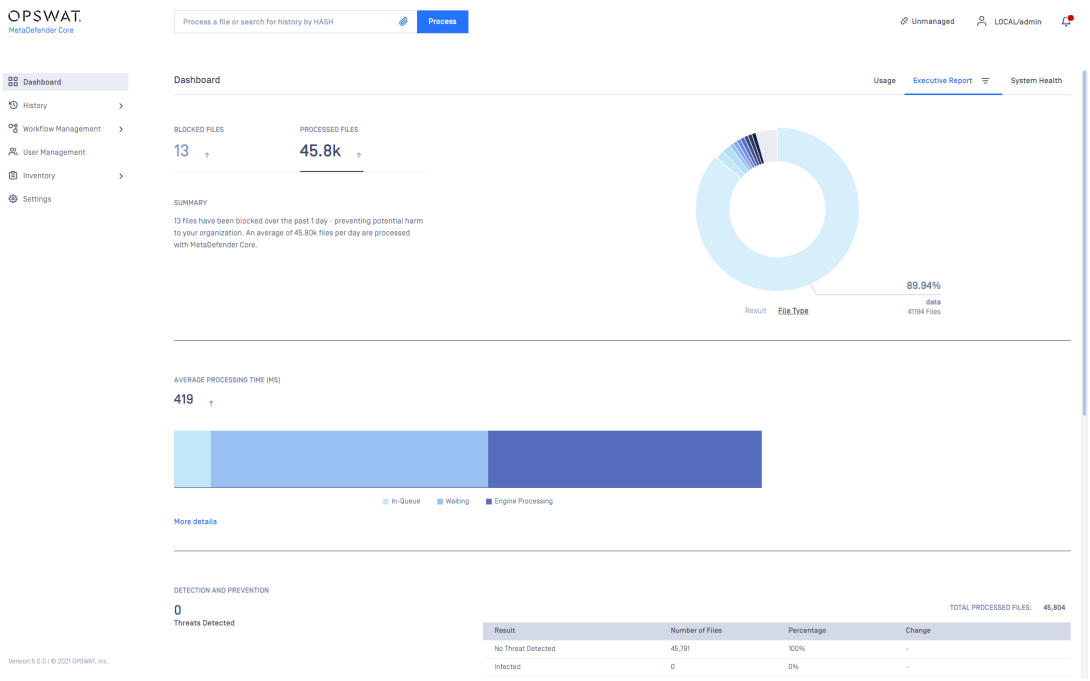
Please be informed that

- All existing administrative APIs related to "Node" on v4 will be kept as it is on v5 to ensure backward compatibility with existing client integrations.
- Applicable settings defined in Windows registry (Windows) or ometascan-node.conf config file (Linux) for Node will be migrated to Core. Details: MetaDefender Configuration

Brand-new product user interface

The second notable update in this version is a brand-new UI for the management console, which was designed with the focus on the quality and thoughtfulness of the user experience. The new administrative console UI provides an advanced set of Core features and observable processing status.

Comprehensive executive dashboard reporting offers dynamic, real-time and actionable threat insights. It also provides visibility of your security posture allowing you to paint a picture of security that aligns with the overall business. The report includes easy-to-read charts, graphs, and statistic explanation that help your security professionals reduce clutter, gather information quicker, spot threat activity early and act swiftly.



We're continuing to improve our product UI to provide users an excellent experience with the application and overall usability. Please expect more powerful UI features coming in next releases.

New compression archive engine for archive sanitization

In previous version, MetaDefender Core's archive engine implemented four functions including decryption, extraction, encryption and compression that with a high volume of files it takes time to process them all. For the purpose of enhancing archive handling performance, this release comes with a separated compression archive engine, which dedicatedly processes archive sanitization (compression and encryption). It is also created to facilitate maintenance and further feature development.

Archive sanitization feature now requires Deep CDR license

As to respect sanitization practice and ensure unknown threats prevention as designed, now starting on version 5, it's mandatory to have Deep CDR licensed for archive sanitization feature enablement on MetaDefender Core workflow configurations.

Archive handing related engines changed

The legacy "Archive" engine on MetaDefender Core v4 will be changed its name to "Archive Extraction" engine, even when you are still on MetaDefender Core v4.

A brand new "compression archive engine" is to dedicatedly handle archive sanitization in v5 (archive compression), that helps relevant archive handling performance improvement, easier for maintainance and further feature development.

Notes when you are still on MetaDefender Core v4 (not upgraded yet)

The legacy "Archive" engine (now name changed to "Archive Extraction" engine) on MetaDefender Core v4.x will still cover both extraction and compression archive tasks as usual.

Even though your MetaDefender Core v4.x might now have a new "Archive Compression" engine auto downloaded, but that engine cannot be deployed (marked as "permanently failed" in relevant engine status check API). This behavior is expected, and not causing any impact to MetaDefender Core processing flow, or its performance though.

- Just in case your client integration has a check for engine status before triggering scan request, then you might want to disable "Archive Compression" engine status on MetaDefender Core v4 using REST API. Detailed instruction: Archive Compression Engine Permanently Failed on MetaDefender Core v4
- When you later upgrade to MetaDefender Core v5 from v4, then you might need to manually re-enable "Archive Compression" engine to make it active and running on MetaDefender Core v5.

Notes for licensing

For offline deployment (MetaDefender Core is not connected to the Internet directly for online update), you will need to re-activate again on OPSWAT portal to retrieve latest engine list in .yml file including "Archive Compression" engine if you are currently licensed with Deep CDR.

Meanwhile, for online deployment, this manual procedure is not required since MetaDefender Core will cover that itself.

Import & export configurations

Not only workflow settings like in previous versions, but also all other general settings (update, security, data retention, server, hashes to skip) and most importantly, user settings, will be able to export and import on MetaDefender Core version 5.

Learn more at Import/Export configuration

Backward compatibility supported

The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

Windows application event logging disabled by default

Applicable only to Windows OS based MetaDefender Core.

Help avoid duplicated logging with separate file option, and improve overall processing performance. Of course, when being still valid to you, you are still supported to enable it back.

CSRF token header renamed

CSRF token originally created for security strengthening and being used by MetaDefender Core management console UI since version 4.21.0 via header `oms_csrf_token`

To get it compliant with some load balancers which do not accept underscore in header name by default, and save you from additional load balancer configuration step, we have now changed the header name to `oms-csrf-token`

This change does not impact to any REST API based client integration.

More comprehensive and accurate error messages on REST APIs

Impacting on following REST API endpoints:

- `PUT /admin/config/session`
- `PUT /admin/config/update`
- `PUT /admin/config/scan`
- `PUT /admin/config/skip`
- `PUT /admin/config/auditlog`
- `PUT /admin/config/webhook`
- `PUT /admin/config/rule`

Upload time and queue time logic updated in JSON scan response

Impacting on following REST API endpoints:

- `GET /file/{data_id}`
- `GET /hash/{md5/sha1/sha256}`
- `GET /archive/{data_id}`
- `POST /file/{async}`

Added new field:

- `receive_data_timestamp`: The timestamp when upload progress started (first byte received) (in milliseconds)
- `upload_time`: Total time elapsed for upload process (in milliseconds)

Modified existing field (new logic):

> **Backward compatibility supported**
> The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

- `upload_timestamp`: The timestamp when upload progress finished (all bytes received) (in milliseconds)
- `queue_time`: Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue time = start_time - upload_ timestamp) (in milliseconds)

Release version 5.0.0 also comes with variety of outstanding features and improvements.

For detailed information about our latest release please view our Release notes

Installation

Starting with MetaDefender Core 4.20.0, the centralized/shared database deployment model is supported (all MetaDefender Core instances share the same database), along with traditional standalone database mode (every MetaDefender Core instance has its own database to manage).

- Standalone mode
- Shared Database Mode

Installation

1. Download the package of your choice from My OPSWAT
2. Install the package on your computer via the Command Line or via the Wizard
3. Open a web browser and point to http://<server name or IP>:<port>
    - The default port is **8008**
    - In case of problem check Inaccessible Management Console page
4. Complete the required steps of the basic configuration wizard
5. You must activate this deployment to use its features

Installation notes

- If the MetaDefender Core package **dependencies are not installed** on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.
- During installation the databases might need to be upgraded. This could take noticeable time depending on database size (eg.: length of scan history).

Standalone mode

Wizard Installation

The Install Wizard is only for the Windows installer (.msi file).

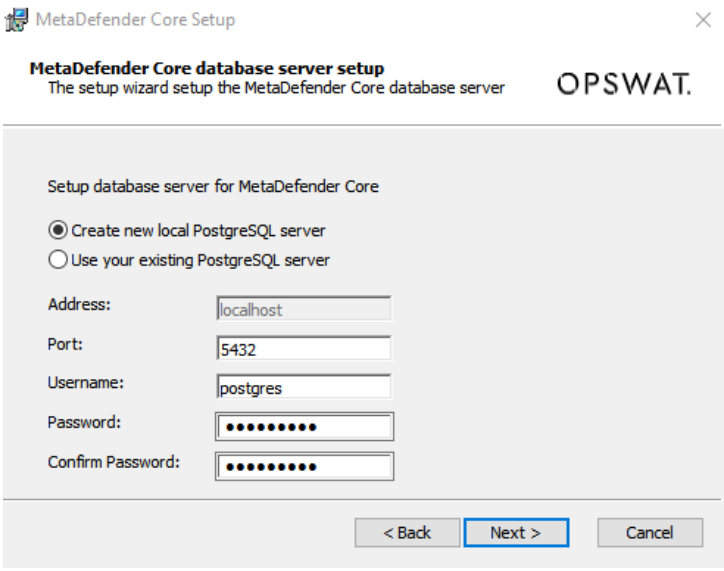
Since MetaDefender Core 4.19.0, we have switched to using PostgreSQL DBMS which needs a dedicated database server. MetaDefender Core supports two modes to setup PostgreSQL server:

Local (Bundle) PostgreSQL server

MetaDefender Core will install a new PostgreSQL server locally in the same box with the product.

> **PostgreSQL user credentials**
> Unicode characters are NOT supported for "Username" and "Password"
>
> "Password" does not accept # (hash) as starting character (e.g. #ABC)

> **The ignition file**
> If the ignition file exist in the designated location, then MetaDefender Core will use the details defined in the ignition file, rather than what is passed via the installation wizard.
>
> More info about the ignition file based installation: Command Line Installation

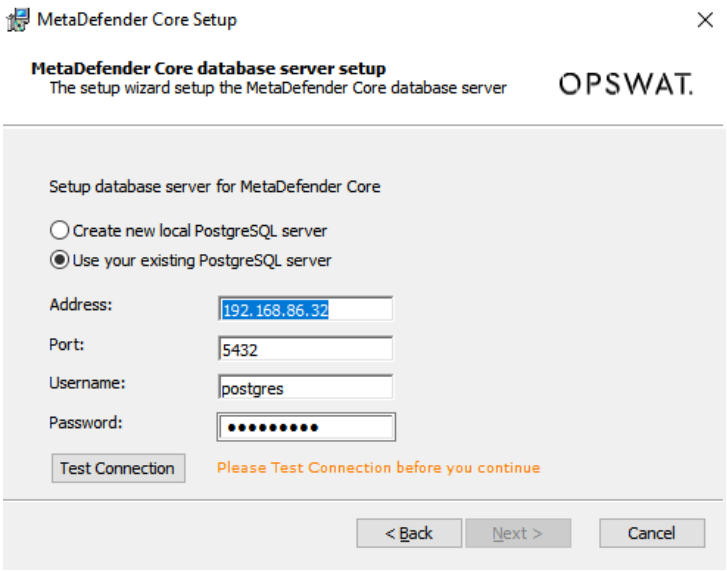


Remote (Existing) PostgreSQL server

MetaDefender Core will connect to remote pre-installed PostgreSQL running remotely.

> **PostgreSQL connection configuration**
> "Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

> **PostgreSQL user privilege requirements**
> MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.
>
> Details: PostgreSQL User Privilege Requirements



Command Line Installation

Ignition File

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Make sure to create the ignition file, if it doesn't exists before you go ahead and install MetaDefender Core via command line.

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

MetaDefender Core supports two modes to setup PostgreSQL server:

1) Local PostgreSQL server

- MetaDefender Core will install a new PostgreSQL server locally, next to the product.
  - A sample iginition file for PostgreSQL server information (***Only non-Unicode characters supported for "user" and "password"__***):

ometascan.conf bash

```
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
```

2) Remote PostgreSQL server

> **Info**
>
> **PostgreSQL version 12 (at least 12.3) or version 14 is required.**
>
> According to PostgreSQL notice, version 12 will stop receiving fixes on November 14, 2024, so we suggest that you use version 14.

MetaDefender Core will connect to a pre-installed PostgreSQL server running remotely and will setup its database.

Make sure to follow steps at Open Connection On PostgreSQL Server first, so that your remote existing PostgreSQL server to accept upcoming connection from MetaDefender Core.

A sample ignition file for PostgreSQL server information:

ometascan.conf bash

```
[dbserver]
type=remote
host=192.168.86.32
port=5432
user=existingserver_admin_user
password=existing_server_pass
```

After installing MetaDefender Core successfully, you may want to remove all credentials info created for [dbserver] section in ignition file for security reason.

> **Warning**
> If the MetaDefender Core package dependencies are not installed on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.

Debian package (.deb)

bash

```
sudo dpkg -i <filename> || sudo apt-get install -f
```

On Red Hat Enterprise 7.x Linux / CentOS package (.rpm)

bash

```
sudo yum install <filename>
```

For systems which enabled GPD check flag:

bash

```
sudo yum install --nogpgcheck <filename>
```

Red Hat Enterprise Linux 8.x package (.rpm)

Install the latest PostgreSQL repository RPM:

> **Note**
> Due to the shorter support cycle on Red Hat Enterprise Linux, all supported versions of PostgreSQL are not available on this platform. The PostgreSQL Yum Repository will integrate with the normal systems and patch management, and provide automatic updates for all supported versions of PostgreSQL throughout the supported lifetime of PostgreSQL. Reference: https://www.postgresql.org/download/linux/redhat/

bash

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

In case installing a remote PostgreSQL on either the same MetaDefender Core server or another server, you need to install the following additional package. PostgreSQL version 12.3 or newer is recommended.

bash

```
sudo dnf install -y postgresql12-contrib
```

Install MetaDefender Core:

bash

```
sudo dnf install -y <filename>
```

Windows package (.msi)

On Windows systems it is possible to install the product by running the corresponding .msi file.

From command line interface it is also possible to install the product by executing

bash

```
msiexec /qn /i <msi file name> <option key>=<option value>
```

where the possible keys and their default values are the following:

Key	Default Value	Description
INSTALLFOLDER	\Program Files\OPSWAT\MetaDefender Core	Customize installation folder for product Example: `INSTALLFOLDER="D:\Products"`
RESTADDRESS	*	REST interface binding IPv4 or IPv6 address ('*' means that service listens on all IPv4 and IPv6 interfaces)
RESTPORT	8008	REST interface binding port
REPORT_ENGINE_ISSUE	true	Enable reporting of engine issue count. (possible values: "true" or "false").

Note

Reporting of engine issue count

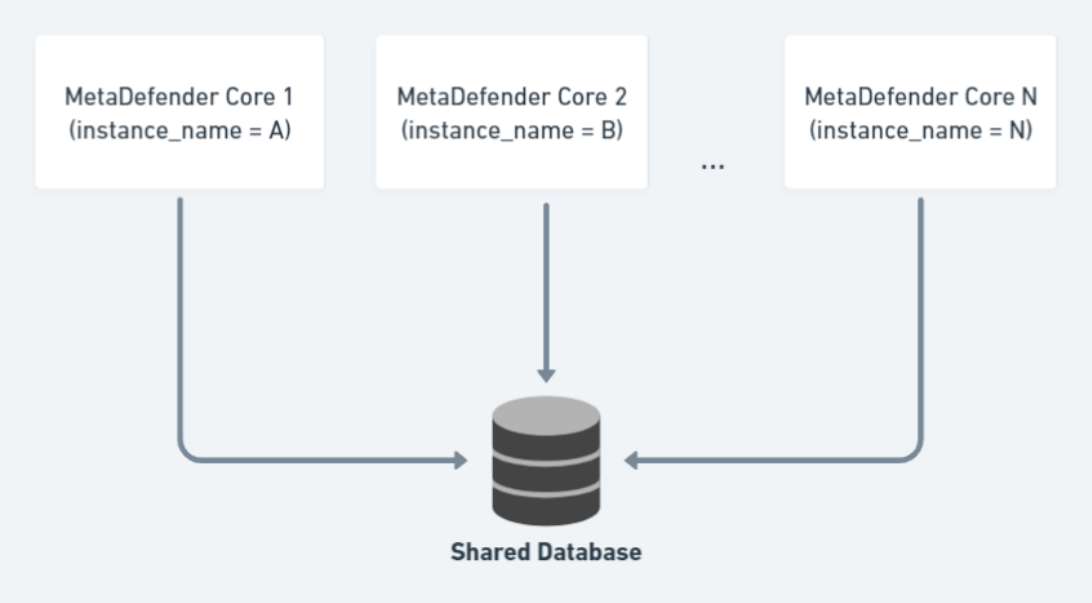
If reporting of engine issue count is enabled, MetaDefender Core server will send only the **number** of initialization errors and **number** of unexpected stops for the specific db/engine version. This information is sent over a HTTPS channel when the product downloads the latest package descriptors. This information is used for early detection of any specific 3rd party engine quality issues.

For details on using msiexec please consult Windows installer documentation.

Shared Database Mode

Starting with MetaDefender 4.20.0, you can use a centralized database deployment model, where all MetaDefender Core instances can share the same external database.

That helps auto-sync all product configurations (**except Single-Sign-On** that requires unique settings for each Core), processing history, live/on-going scan result query, audit, statistics data.



This article demonstrates how to install and setup MetaDefender Core 4.20.0 or newer with shared database mode.

Important notes & current limitations:

Currently not supported

- Does not support Central Management use-case with shared DB mode.
- Does not support to switch from shared database mode back to standalone database mode.
- Does not support to change instance_name of MetaDefender Core in shared database deployment model after installation.
- Does not support upgrading older MetaDefender Core from non-persistent mode to using the shared database mode.
- Does not support upgrading older MetaDefender Core from standalone mode to using the shared database mode.
- Does not support to upgrade multiple MetaDefender Core instances at once in shared database mode
    - Stop all Core instances' scanning service before performing the upgrade.
    - Upgrade each Core server one by one instead.

- For each Core, when the upgrade finished, then you can start using that Core for scanning without waiting for the other Core instances' upgrade to be done.

> **Disclaimer**
> MetaDefender Core does not support mode switching.

Requirements for shared database setup:

- The central remote PostgreSQL server (for shared database) could be utilized with a built-in (native) PostgreSQL database server installed with one of the MetaDefender Core instances in the network. Or using a dedicated existing PostgreSQL database server (see detailed requirements for PostgreSQL version at Recommended System Configuration).
- Before setting up, make sure to follow steps at Open Connection On PostgreSQL Server so that all MetaDefender Core instances could connect to the central remote PostgreSQL server.
- MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core, details: PostgreSQL User Privilege Requirements
- All MetaDefender Core instances must be installed on the same Operating System (Windows / Linux).
- The first MetaDefender Core instance connects to the shared database must be ready first (license activated, users/groups/roles created) before setting up other MetaDefender Core instances to connect and use that shared database.

Engine update:

- If engine updates is configured to be via Folder source, all MetaDefender Core instances must be able to access to the configured update folder, and the option DELETE FILES AFTER IMPORT will be unchecked and prevented for modification.

Authentication:

- Single-Sign-On (SSO) authentication settings (if using) cannot be shared among MetaDefender Core instances because SSO requires an unique login URL for every different Core.
- If an user is locked for login on a certain MetaDefender Core instance(e.g. exceeded the number of failed login times), that user is not prevented for logging in on any other instances.
- If an user is locked for sign-in on a certain MetaDefender Core instance (e.g. exceeded the number of failed login times), if the admin releases lockout on that user on that MetaDefender Core instance, that user is still not able to login on other MetaDefender Core instances.

Sanitized / DLP / Quarantined location changed after MetaDefender Core install:

Windows:

Shared storage path will be stored in 3 strings under key global in registry: quarantinepath, dlppath, sanitizepath as below

(Default)	REG_SZ	(value not set)
dbmode	REG_SZ	4
dlppath	REG_SZ	\\10.40.50.100\test
instance_name	REG_SZ	481rvj
quarantinepath	REG_SZ	\\10.40.50.100\test
restaddress	REG_SZ	*
restport	REG_SZ	8008
sanitizepath	REG_SZ	\\10.40.50.100\test

You can set either 1, 2 or 3 different paths for the 3 settings above (means that quarantined, DLP and sanitized files will be stored in same or different storages corresponding). In case shared storage is not set in installation, values of 3 settings above will be empty.

Restart MetaDefender Core service after changed.

Linux:

Open file `/etc/ometascan/ometascan.conf` and set shared storage path for 3 keys quarantinepath, dlppath, sanitizepath under section [global]

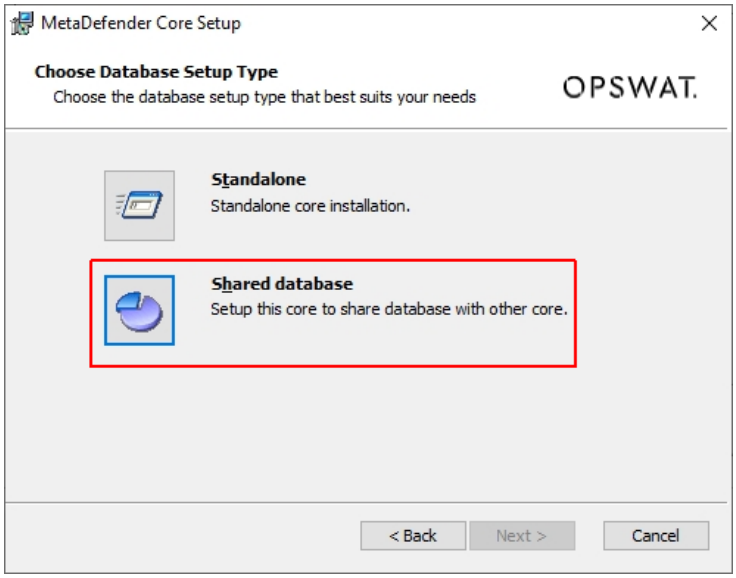
Restart ometascan service after changed.

Wizard Installation (Windows Only)

> **Info**
> Only applicable to MetaDefender Core 4.20.0 or newer using wizard on Windows OS.

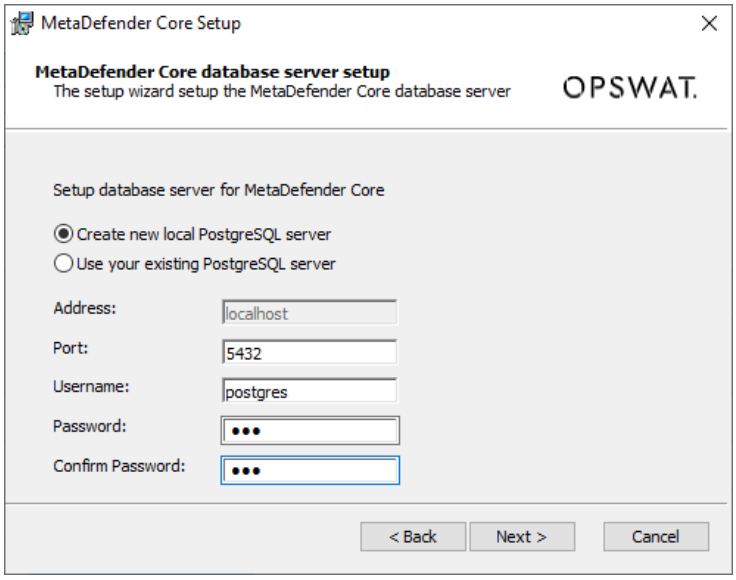
Clean install scenario

1. **Choose Shared database option:**

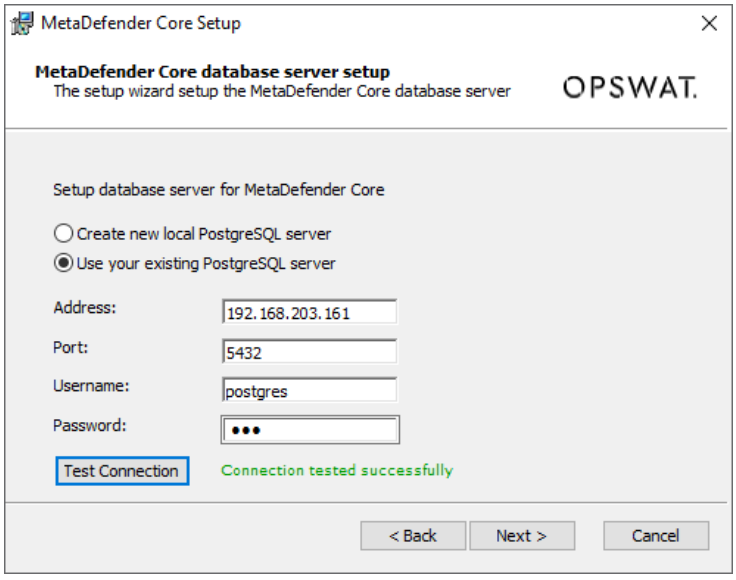


2. **Setup database server for MetaDefender Core:**

You can choose to create a native / built-in PostgreSQL server and setup the shared database on (then other MetaDefender Core instances will want to connect to that PostgreSQL server afterwards)



Or choose to connect to a pre-setup existing PostgreSQL server:



PostgreSQL connection configuration

"Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

PostgreSQL user privilege requirements

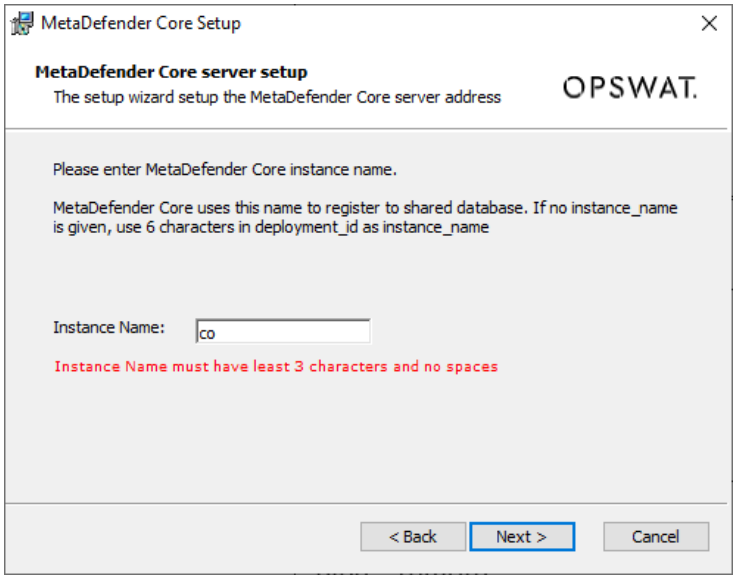
MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.

Details: PostgreSQL User Privilege Requirements

3. **Create an unique instance name for that MetaDefender Core (different from other MetaDefender Core instances):**

Instance name must contain at least 3 characters and no white space.

If you choose to leave it blank, which means it will use the default instance name with 6 characters extracted from that MetaDefender Core deployment ID.

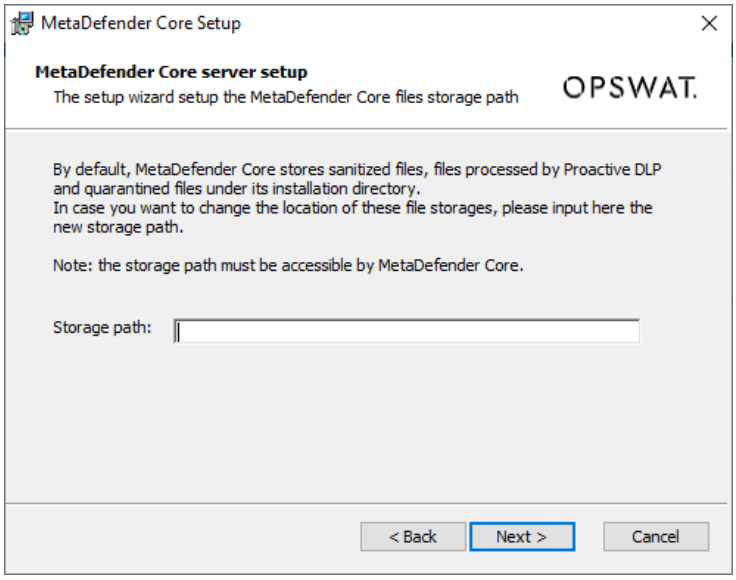


4. **Input the file storage path (sanitized, DLP, quarantined files)**

In the shared database model, all MetaDefender Core instances could use the same folder path to store all their own sanitized, DLP, quarantined files after processed.

The default location is: ***<MetaDefender Core installation folder>\data\ {sanitized|dlp|quarantine}***

If you wish to use that default location, just leave the input box blank and press Next.



Otherwise, you can specify a different location of sanitized/dlp/quarantined files. It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core.

For example:

- C:\new_folder
- \192.168.1.1\new_folder

Command Line Installation

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Clean install scenario

1. Create ignition file if not existed

- Windows: **C:\OPSWAT\ometascan.conf**
- Linux: ***/etc/opswat/ometascan.conf***

An example of ignition file:

ometascan.conf bash

```
eula=true

[global]
dbmode=4
instance_name=MetaDefenderCoreA
storage_path=D:\path\

[user]
name=admin
password=admin
email=admin@local

[dbserver]
type=remote
host=192.168.1.1
port=5432
user=postgres
password=your_own_password
```

Field	Required	Mapping to wizard UI	Possible values
global/dbmode	optional default is 1 (standalone)		1 - Standalone 2 - Central Hub deployment 3 - Non-persistent **4 - Shared**
global/instance_name	optional default is 6 characters of deployment ID		Unicode and number At least 3 characters No whitespace
global/storage_path	optional default is empty		Path to storages (DLP, quarantined, sanitized files) It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core. For example: • C:\folder • \192.168.1.1\folder

2. Run installers using command line to install

Please refer to Command Line Installation for installation command.

Deployment automation support

Note

An Ignition file could be utilized for either / both of following use-cases:

1. Pre-define PostgreSQL server information for MetaDefender Core to connect and setup database: only required on Linux installation, or Windows with command line installation. Learn more details at:
    a. Standalone Mode > Command Line Installation (applicable on version 4.19.0 or newer).
    b. Shared Database Mode > Command Line Installation (only applicable on version 4.20.0 or newer).
2. Automating to accept EULA, create a default local admin user and import MetaDefender Core configurations upon installed: Those steps could be automated using ignition file described in this page.

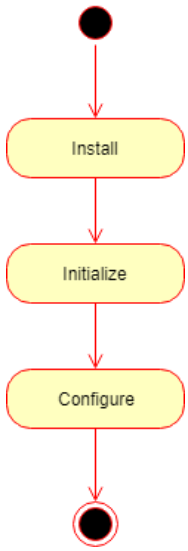
Ignition file location to create:

- **Windows**: `C:\OPSWAT\ometascan.conf`
- **Linux**: `/etc/opswat/ometascan.conf`

The product supports fully automated deployment. It means that it can be installed and configured with no human interaction.

The automated deployment can be split to three steps on a high level:

1. Installation
2. Initialization
3. Configuration



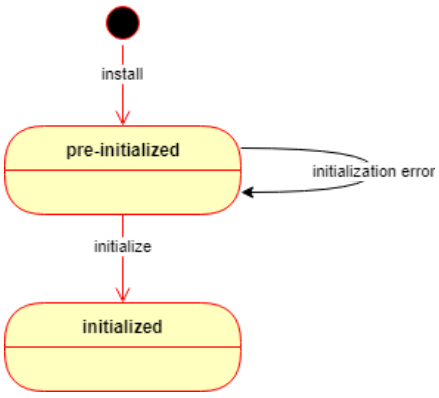
Installation

To automate the installation, install the product from the command line and provide the installation-time options as parameters to the installer. For further details see Command Line Installation

After the installation is complete, the product starts up and waits in a pre-initialized status. The product may be initialized in two ways:

1. Manually using the Wizard Setup or
2. Automatically using an *ignition file* (see below).

> **Warning**
> If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried after fixing the problem (e.g. placing the ignition file in its lookup location) and restarting the OPSWAT MetaDefender Core service.
>
> Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

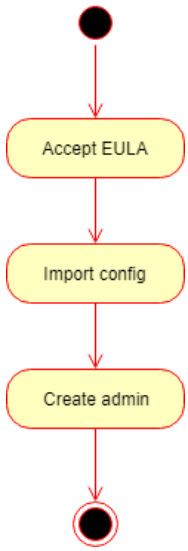


Initialization

Initialization is the process of bringing the product to an operable status.

Basically the initialization consists of the following steps:

1. Accept the End User License Agreement (EULA),
2. Import product configuration and
3. Create the first administrator user account.



Ignition file

The initialization process can be configured in a file called the *ignition file*.

> **Warning**
> In Linux, the ignition file and the folder /etc/opswat should have the permission 755

Ignition file location

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

The ignition file must be in conf format.

ometascan.conf bash

```
eula=true
[user]
name=admin
password=admin
email=admin@local
apikey=44c96cbb4e4e6c0a1d16bcb3c3da24def317
[config]
import=config_package.zip
import_password=123
[security]
session_cookie_samesite=Lax
[internal]
skip_startup_pages=true
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
private_username=my_internal_db_user
private_password=mypassword
[ocm]
ocm_url=https://my.opswat.com
reg_code=XXXXXXXX
```

Ignition file fields

The ignition file accepts the following fields:

Section	Key	Required	Accepted Values	Description
	eula	Optional (mandatory when having [user] section)	true false (default)	Whether to accept the End User License Agreement. This key must be set at the top of ignition file. This key must be set to "true" to accept the EULA. Any other value will cause the initialization to fail. When having [user] section defined, then this key must be set.
[user]		Optional		Initial administrator user account properties. The Administrator role is granted to the account.
	name	Optional	string	User name for the initial administrator user account.
	password	Optional	string	Password for the initial administrator user account. WARNING! Clear text password The password in this configuration file must be stored in its clear-text format and as so it may be visible for unauthorized parties.
	email	Optional	string	E-mail address for the initial administrator user account.
	apikey	Optional	string	API key for the initial administrator user account.
[config]		Optional		
	import	Optional	string	Path to a file in json/zip format that contains a previously exported configuration to be imported. Zip-format file contains a full configuration including user management settings, so it will overwrite [user] section.
	import_password	Optional	string	Password to decrypt .zip configuration package.
[security]		Optional		

Section	Key	Required	Accepted Values	Description
	session_cookie_samesite	Optional	Strict Lax (default) None	Declare if session cookie should be restricted to a first-party or same-site context.
[internal]		Optional		
	skip_startup_pages	Optional	true false (default)	Skip all welcome and other startup pages (e.g. upgrade result) when installing / upgrading MetaDefender, and going straight to the product index default page.
[dbserver]		Optional		
	type	Mandatory when having [dbserver] section	local remote	Where the PostgreSQL database server should physically locate. "remote" means PostgreSQL server and MetaDefender Core are not in the same machine.
	host	Mandatory when having [dbserver] section	string	IP address / domain name of the server where PostgreSQL server locates. "localhost" can be used when applicable.
	port	Mandatory when having [dbserver] section	number	Port of PostgreSQL server is listening for connections from clients (i.e. MetaDefender Core).
	user	Mandatory when having [dbserver] section	string	PostgreSQL server's user. SUPERUSER privilege is required for MetaDefender Core to setup its database and extensions for the first time. **Only non-Unicode characters supported.**

Section	Key	Required	Accepted Values	Description
	password	Mandatory when having [dbserver] section	string	PostgreSQL server's user credentials. **Only non-Unicode characters supported.**
	private_username	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
	private_password	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
[ocm]		Optional		
	ocm_url	Optional	string	URL to My OPSWAT server.
	reg_code	Optional	string	A registration code to enroll to My OPSWAT server.

Detailed initialization process

1) After the product has been started, it looks for the ignition file in the designated location.

Windows: C:\OPSWAT\ometascan.conf ** Linux: /etc/opswat/ometascan.conf

** 2) If an ignition file is found, then

2.1) It gets validated, and if it is valid, then based on the information found in the ignition file:

- The EULA is accepted,
- The configuration is imported,
- The administrator account is created.
- If any of the above steps fails, then the error is logged, and the initialization gets terminated.

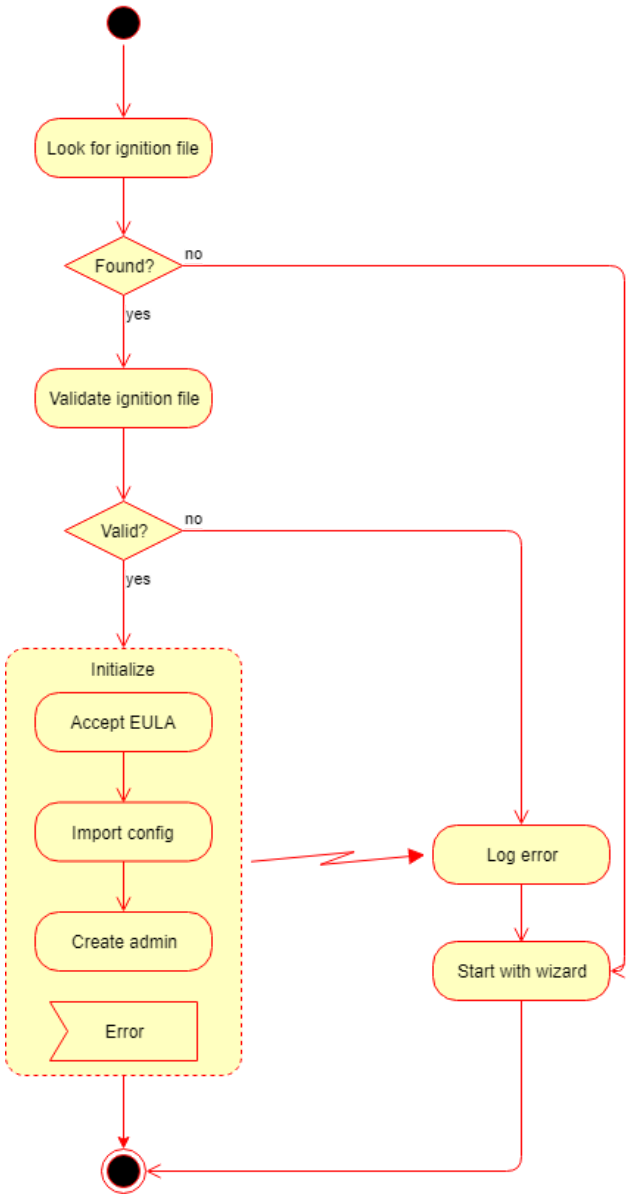
> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

2.2) If it is not valid, then the error is logged, and the initialization gets terminated.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

3) If there is no ignition file, then no initialization is performed.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.



Warning

If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried fixing the problem (e.g. placing the ignition file to its lookup location) and restarting the OPSWAT MetaDefender Core service.

Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

Configuration

After the initialization is complete, the product is ready with the default and the imported configuration.

This configuration can be later changed calling the configuration API functions. For further details about the API see Configuration related APIs API

CIS Level 1 Guidelines

Support Red Hat Enterprise Linux 9
For more details about Center for Internet Security (CIS) please refer to this document

Instruction steps

I. Install OpenSCAP

bash bash

```
yum install openscap-scanner scap-security-guide
```

II. Generate a result file and a HTML report using OpenSCAP scanner tool

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

III. Remediation of CIS Level 1 issues

Generate a remediation script based on the `ssg-rhel9-ds.xml` file:

bash

```
oscap xccdf generate fix --profile
xccdf_org.ssgproject.content_profile_cis_server_l1 --fix-type bash --output
remediations.sh /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

And execute remediation script.

bash

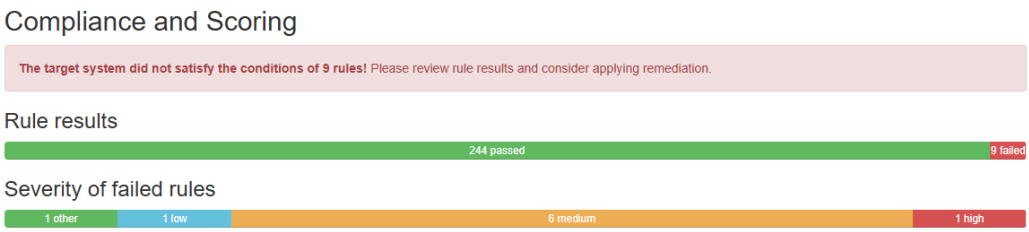
```
./remediations.sh
```

IV. Review the results after remediation

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

Example:



Notes

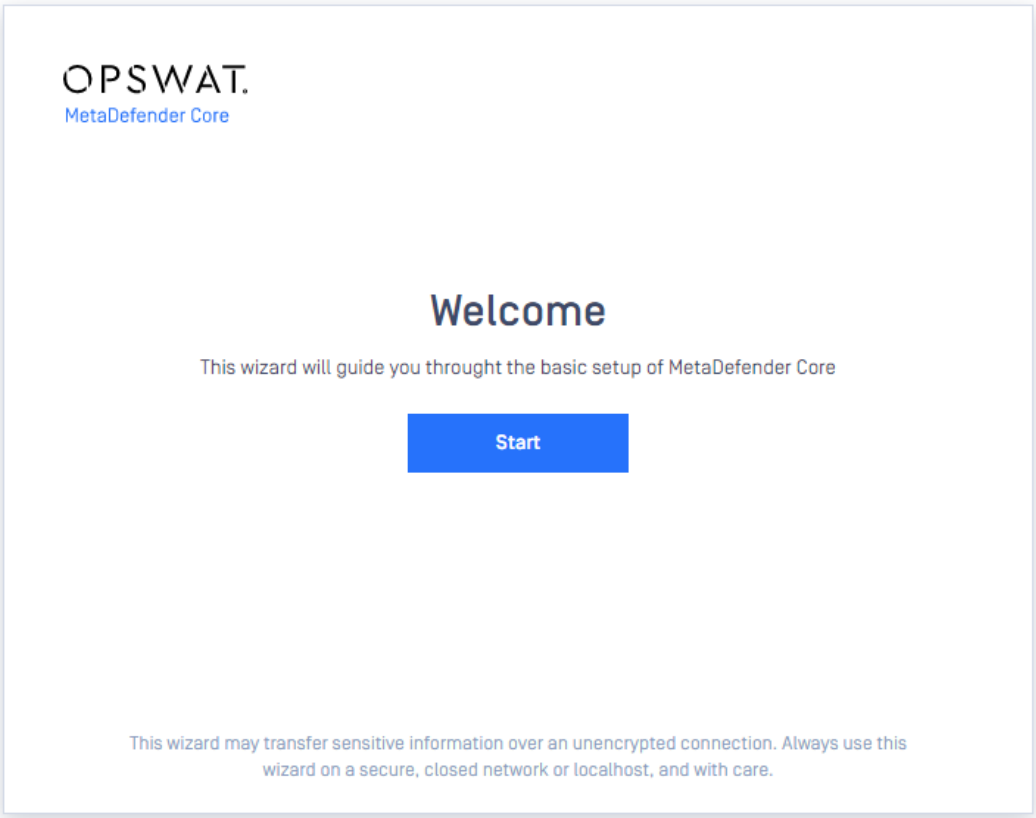
- CIS Level 1 requires `/tmp` folder to be mounted in a separate partition. Please ensure that that new partition have enough disk space for MetaDefender Core to run.
- CIS Level 1 requires that "Ensure No World-Writable Files Exist".
    - For now, when freshly installing MetaDefender Core, all its binary files meet the requirement.
    - When installing/updating engines, some engines might create additional files for its operation, and it might violate this requirement. In this case, you need to again execute the remediation script in the step III.

Wizard Setup

Introduction

Access the MetaDefender Core management console for the first time (i.e. http://localhost:8008), you are to complete a quick walkthrough for wizard setup in order to be able to use the product.

The administrative management console will be available only upon you have successfully finished this wizard stage.

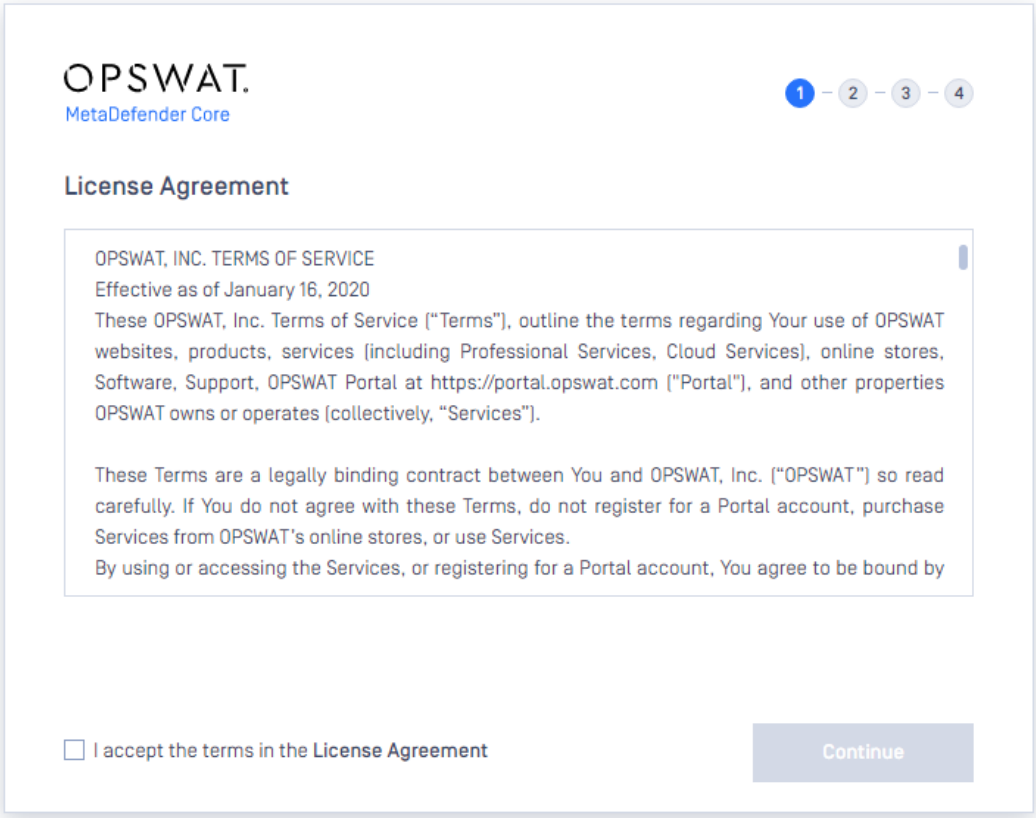


Warning

This wizard may transfer sensitive information over an unencrypted connection. Always use this wizard on a secure, closed network or localhost, and with care.

Basic configuration steps

End-User License Agreement



In the first page you can find the End-User License Agreement. You have to accept the terms before moving on.

Please read through the EULA (https://docs.opswat.com/policies/privacy-policy/portal-privacy-policy) carefully and if you agree with it, check `I accept the terms in the **License Agreement**` and click `Continue` to proceed.

Local Admin User Setup

The next step is to set up a default local administrator account.

This account will be the first one being able to access the administrative management console, and to create accounts for other users. You have to fill out all fields in this page to be able to move forward. When you are done, click NEXT to continue.

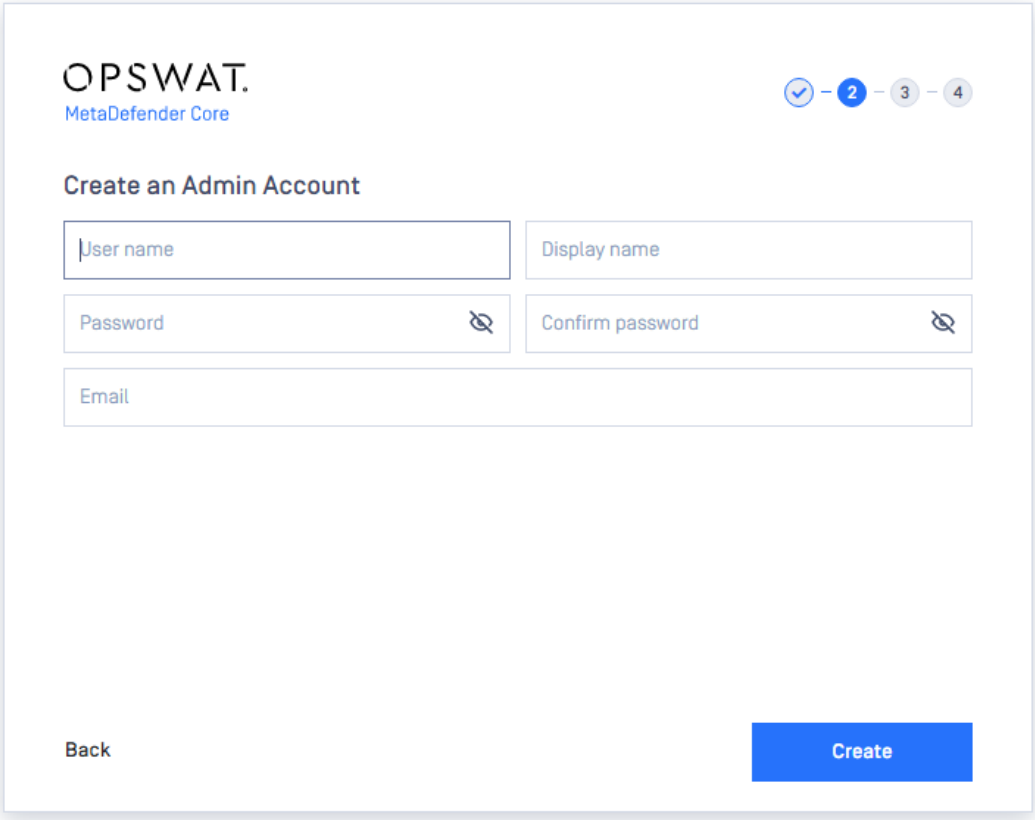
Info - User Directory

The administrator account, that is created via the basic configuration wizard, is always added to the LOCAL user directory as a member.

Learn more about user directory: User directories

The following information is required for the administrator account:

ACCOUNT NAME	The unique name of the account that is used at the time of login and in log messages for accountability.
ACCOUNT DISPLAY NAME	Name of the person bound to this account. This name (appended to the name of the account's user directory) is displayed in the top right corner of the administrative management console.
PASSWORD	Password of the user bound to this account that is used at the time of login. Passwords sent clear-text As long as TLS is not configured for the basic configuration wizard, passwords are sent clear-text over the network and may be disclosed to unauthorized parties. As a mitigation action: 1. Either use the wizard on *localhost* or on a direct network connection, or 2. Enabling HTTPS as soon as possible and change the password immediately if it has already been set.
EMAIL	Email address of the person bound to this account.



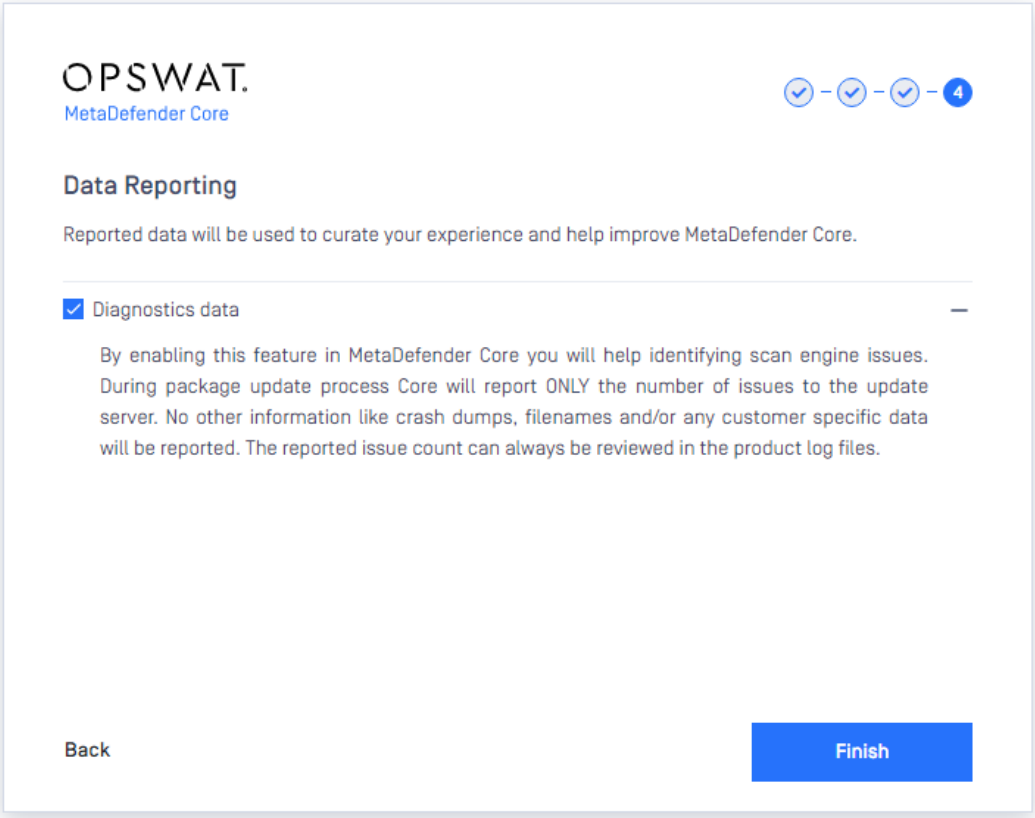
License activation

For license activation details see License Activation

Data Reporting

Enabling this option to allow MetaDefender Core collect its own engine issues and report back to OPSWAT for further analysis.

MetaDefender will not collect any other information.

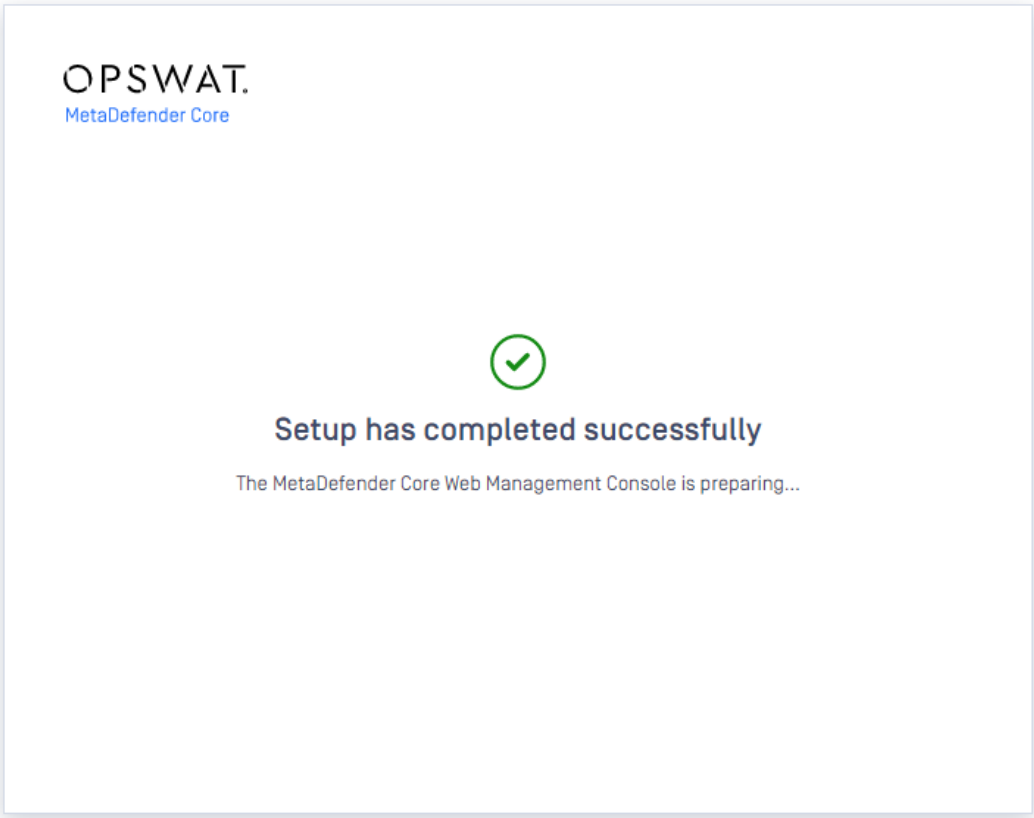


Wizard completion

After you have completed every steps you are ready to finish the wizard and start using the product. Click the `Fishish` button to complete the wizard.

> **Warning**
> The product's service will be restarted and the browser will be redirected to the administrative management console. This could take several seconds.

You can login to the Web Management Console with the administrator user that have just been created in the previous steps.



Setup HTTPS

Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network. Websites, like the Web Management Console, are able to use TLS to secure all communications between their servers and web browsers.

The TLS protocol aims primarily to provide confidentiality (privacy) and data integrity between two communicating computer applications.

Warning

No TLS for the wizard

By default, TLS is not enabled for the basic configuration wizard. As a consequence sessions between the wizard's backend and the browser may be insecure.

Info

Performing the same steps as for the Web Management Console, it is possible to set up TLS for the basic configuration wizard. Remember completing the TLS setup before launching the wizard.

For instructions to set up TLS see Enabling HTTPS

License Activation

##

To activate your installation go to the **Settings > License** menu in the Web Management Console. If you have no valid license, you will only see your installation's **deployment ID**. You will also see a warning in the Web Management Console header.

After successful activation, the product will start downloading the latest available scan engines and malware databases. You can follow the status of the scan engine installation on the **Inventory > Modules** page.

When all the licensed engines are installed, you can start using the MetaDefender Core to process files - File Processing.

Deployment ID

Each MetaDefender Core instance represented by an unique system fingerprint called `Deployment ID`

MetaDefender Core licensing is purely based on deployment ID activated. Therefore, everytime your MetaDefender Core is activated with a different deployment ID, then it will be captured as a different activation record accumulated into your total activation times allowed.

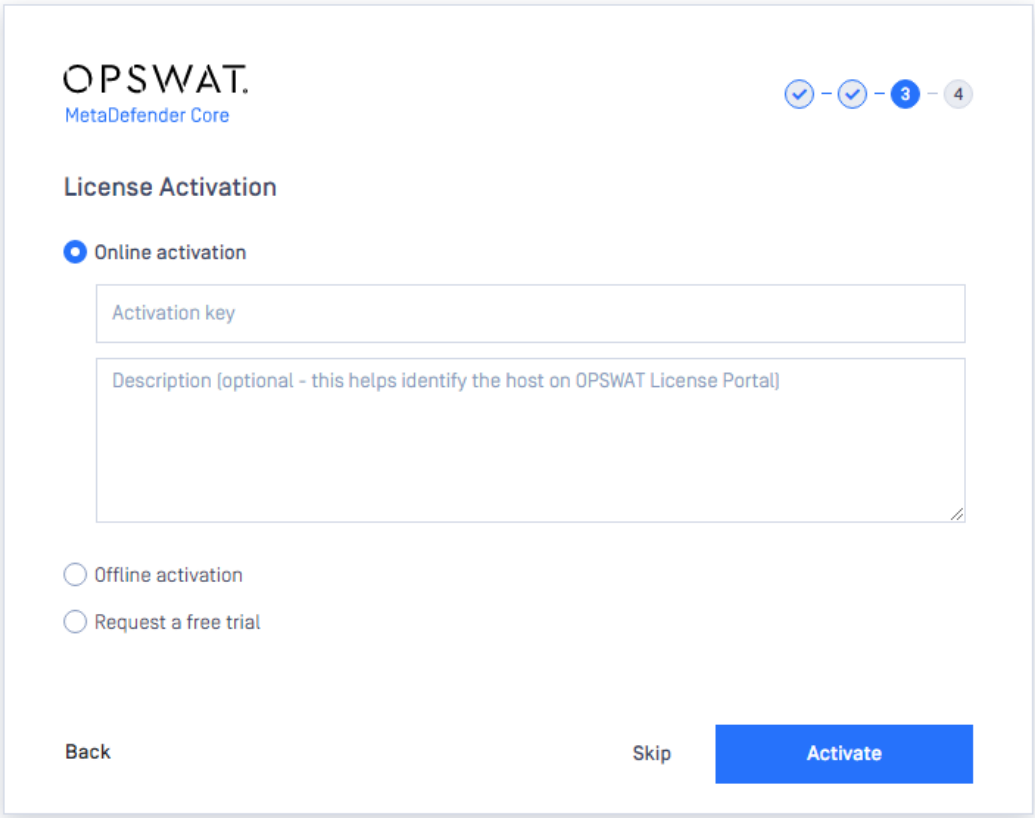
> **Notice**
> Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you alter the system where MetaDefender Core relies on (e.g. network card, OS hostname).
>
> Deployment ID is an UUID version 4 random based string. Learn more at: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you update the system where MetaDefender Core relies on.

Activation Modes

There are three different options for the product activation:

- Online activation: MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.
- Offline activation: Requiring you to upload a license file (.yml). That license file could be retrieved by activating your license via OPSWAT Portal. Follow the displayed instructions for details.
- Request trial key online: For evaluation purpose, you can receive an trial Activation key via email. Follow the displayed instructions for details.



Online activation

MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.

Required to input your Activation License key, and optional for description (but we recommend this for troubleshooting and auditing purpose if needed).

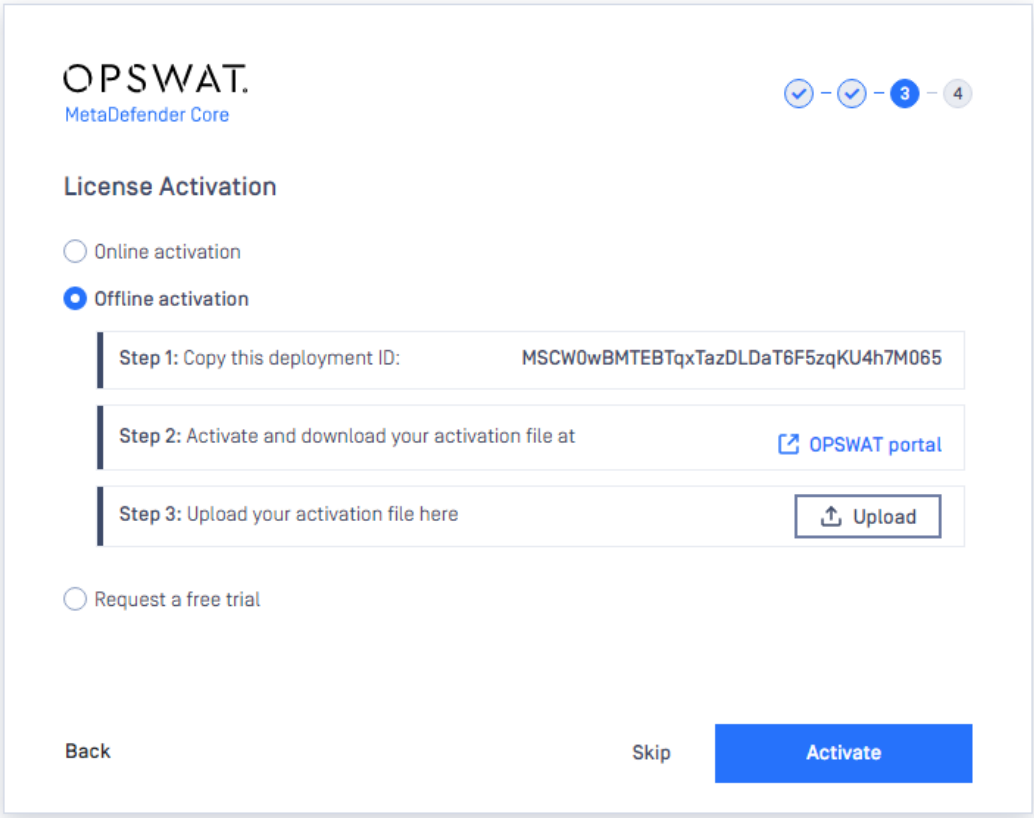
Warning

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts' for license activation:

- https://activation.dl.opswat.com Note: *IP address-based whitelisting on your firewall might fail after some time since OPSWAT uses CDN (Content Delivery Network) to faster delivery updates over the world, and IP address of edge servers might change over time.*

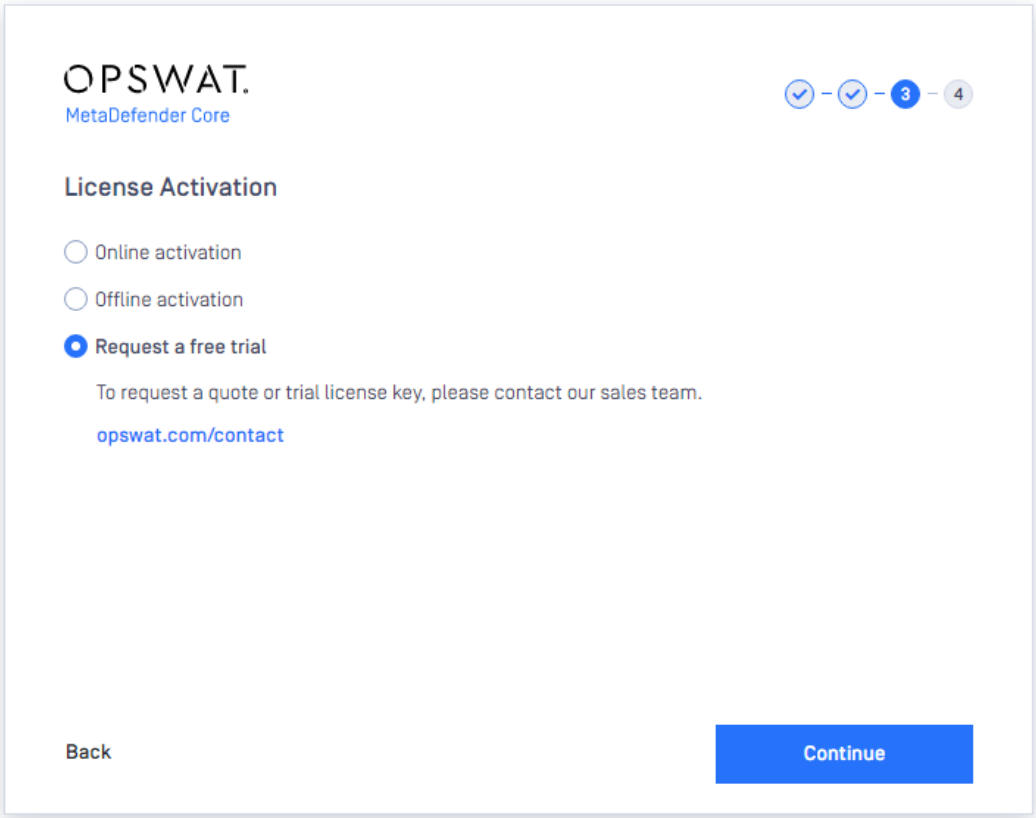
Offline activation

With no internet connection on the server the Metadefender v4 instance may be activated indirectly from a different machine, that has internet connection. The Deployment ID of the Metadefender v4 instance and the the Activation key received at the time of purchasing the product will be required. Follow the steps on the screen to activate the product offline.



Request trial key online

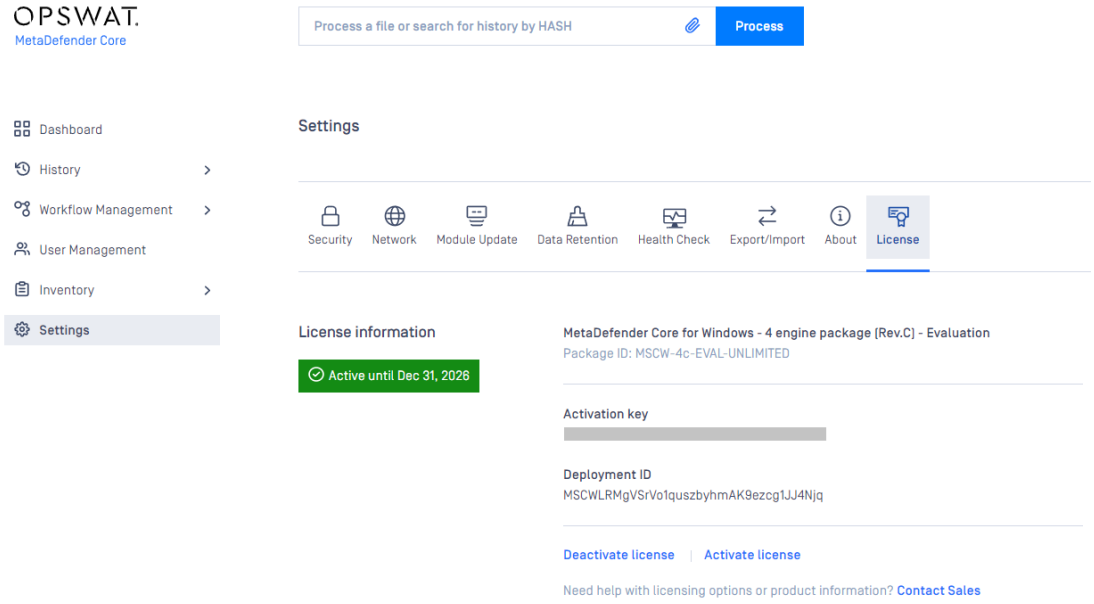
An evaluation license could be acquired by contacting our sales team.



Check Your License Details

For more license details and activating your MetaDefender Core, go to Settings > License menu tab on the Web Management Console:

- Product name: product name as on your order (e.g. MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation)
- Package ID: product identification as on your order (e.g. MSCW-4c-EVAL-UNLIMITED)
- Expiration: last day of license validity
- Activation key: key used for activating product
- Deployment ID: identification of this installation


OPSWAT.
MetaDefender Core
Process a file or search for history by HASH
Process
Dashboard
History
Workflow Management
User Management
Inventory
Settings
Settings
Security
Network
Module Update
Data Retention
Health Check
Export/Import
About
License
License information
Active until Dec 31, 2026
MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation
Package ID: MSCW-4c-EVAL-UNLIMITED
Activation key
Deployment ID
MSCWLRMgVSrVo1quszbyhmAK9ezcg1JJ4Njq
Deactivate license
Activate license
Need help with licensing options or product information? Contact Sales

Scripted license management

Using REST API calls there is an option for scripted activation and deactivation for MetaDefender Core servers with **no Internet connection or if your infrastructure is using automation to create/destroy MetaDefender Core instances**. In this chapter the steps of these two scenarios are described.

Requirements

- an installed MetaDefender Core instance without Internet connection
- another computer that has Internet connection and can run your activation scripts
- a manual or automated way to transfer data between the two computers

Activation steps

1. In order to activate MetaDefender Core, deployment ID and activation key are needed to generate the license file.
    a. Activation key should be purchased from OPSWAT.
    b. MetaDefender Core deployment ID can be queried by using REST API (for details see GET Get Current License Information API page)

> **Info**
> Save this deployment ID in your system, you might need this ID when the instance is unavailable at the time of deactivation.

2. Activate license and get the license file using the following URL, where <quantity> is 1.

activation call bash

```
curl -X GET https://activation.dl.opswat.com/activation?key=<activation-key>&deployment=<deployment unique-ID>&quantity=<quantity>
```

If the activation is successfully, a license file is downloaded. Save this file.

HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	license file	Activation was successfully.
200 Activation failed	error: '<user conform error message>'	Failed activation
200 Invalid parameter	error: 'Could not activate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not activate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
200 Invalid parameter	error: 'Could not activate your product because the quantity you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key, quantity or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Info
In case of any activation issue, contact OPSWAT support for help

3. Upload license file to MetaDefender Core: The license file should be uploaded to the MetaDefender Core v4 to activate the product. For details see page: POST Upload license key API

Deactivation steps

1. For deactivation, following infomation is required:
    a. Activation key
    b. MetaDefender Core deployment ID: can be queried by using REST API (for details see GET Get Current License Information API page)
2. Deactivate license using the following URL:

deactivation call bash

```
curl -X GET https://activation.dl.opswat.com/deactivation?key=<activation-
key>&deployment=<deployment unique-ID>
```

Response HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	result: ok	Successful deactivation
200 No active license found	error: 'Could not found any active license with the given parameters'	The license has not been activated yet or it has been deactivated already.
200 Invalid parameter	error: 'Could not deactivate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not deactivate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Important notes

Warning

Product activation is tied to several hardware, operating system and software parameters. In case of one or more major hardware or software parameter change the product might turn into deactivated status. Operating system updates, other software updates should not affecting the activation status.

If the product is online activated and has live Internet connection then in case of deployment ID change the product reactivates itself. If the product doesn't have Internet connection then the administrator is responsible to reactivate the product.

Because of the above administrator should consider licensing restrictions of the offline deployments if using any virtualization and/or containerization technologies.

File Processing

MetaDefender Core provides REST API interface for file processing business.

There are several ways to process files with MetaDefender Core:

- Analyze Files via Web Interface
- Process Files via REST API

Product Upgrading

> **Info**
> Simply run installer file, details could be found at Installation

> **Info**
> For command line mode, upgrading (e.g. on **Linux**), prior to running the installer file, please make sure to **create the ignition file** with PostgreSQL server information included. Details:
>
> - Standalone DB mode: Command Line Installation (Windows and Linux)
> - Shared DB mode: Command Line Installation

Recommendation for MetaDefender Core upgrade

Our goal is to help you execute the product upgrade continuously, efficiently, and correctly.

PostgreSQL service upgrade

- **Bundled PostgreSQL Service**: If you are using the bundled PostgreSQL server included in the installer, the PostgreSQL service will be automatically upgraded to the newer version (if available). No additional action is required for this process.
- **Remote PostgreSQL Service**: For remote PostgreSQL servers, customers are responsible for managing and upgrading the service independently. Upgrading MetaDefender Core does not update the remote PostgreSQL service.

Before the upgrade

Prepare a proper disaster recovery plan (DRP) including some common practices:

- Backup, or take a snapshot of your current working MetaDefender Core server.
  - This is to help quickly rollback for any upgrade failure might cause later for any reason.
- Optionally backup your PostgreSQL database, especially when using remote shared database mode.
  - Reference: https://www.postgresql.org/docs/current/backup.html
  - Please do it after stopping all linked MetaDefender Core instances to avoid potential data collision.
- Lastly, if backing up entire server is not feasible, then at least export MetaDefender Core configuration package Import/Export configuration just in case you'll need to start off a new server with current MetaDefender Core configuration to keep the scanning service running without any impact.

On the PostgreSQL server, double check server resource to ensure sufficient RAM memory and disk space to be preserved for the upgrade process.

- 8GB RAM available at least during the upgrade.

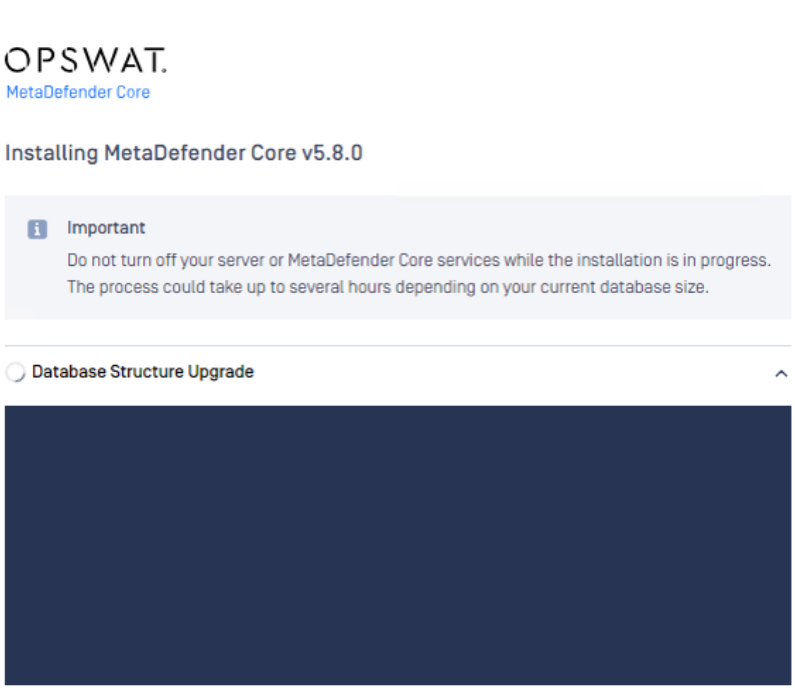
- Disk availability is greater than MetaDefender Core database size.

While doing the upgrade

Applicable for shared database deployment:

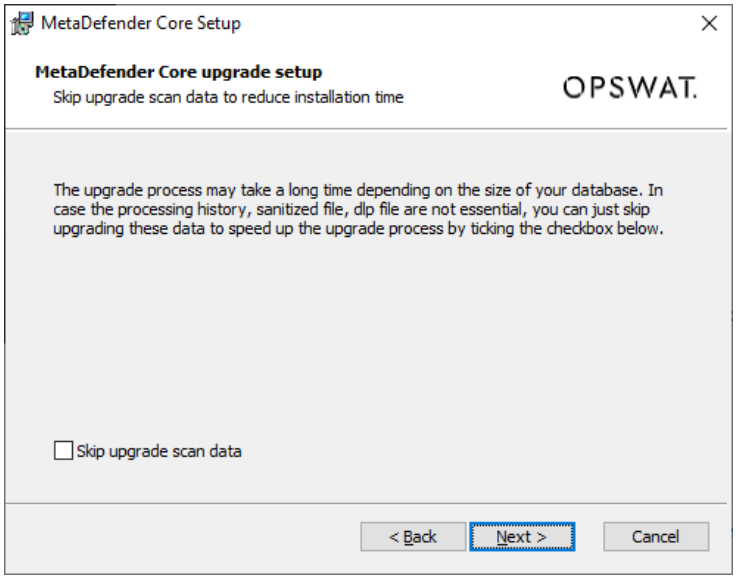
- Stop all MetaDefender Core instances' scanning service before performing the upgrade.
- Upgrade one (1) MetaDefender Core instance at a time.
- Proceed to upgrade MetaDefender Core instance where the bundle local PostgreSQL server is used for shared database setup.
- Otherwise is using remote PostgreSQL then the upgrade order does not matter.

For large size database, mostly for scan history, expecting long time to be spent on the upgrade process.



The product supports an option to skip data migration for scan history to expedite the process drastically and mitigate any failure risk. You can manually backup all data from your PostgreSQL server for offline usage if any.

Check "Skip upgrade scan data" to enable this feature while doing the upgrade.



or add following key in the ignition file:

none

```
[internal]
skip_upgrade_scan_data = 0
```

Troubleshooting

If the upgrade process ever runs into an error:

- Double check your network and services, applicable to remote PostgreSQL server.
- Optionally configure MetaDefender Core to skip migrating scan history.
  - Windows
    - Open Windows registry, and navigate to HKLM\Software\OPSWAT\Metascan
    - Add a new key `internal/skip_upgrade_scan_data` with value = 1
  - Linux
    - Open MetaDefender Core configuration file `/etc/ometascan/ometascan.conf`
    - Add new key into the file
      - [internal]
      - skip_upgrade_scan_data=1
  - Note: This key will be auto removed upon the upgrade finishes.
  - Then restart MetaDefender Core service to retry the upgrade.
    - Might retry this a few times until the upgrade is done.

Just in case the upgrade process still fails, please contact OPSWAT support with the support package - How to Create Support Package? This will help us analyze the root cause and figure out an optimal solution for your system failure.

Using external Load Balancer

Application Load Balancing (Layer 7)

This page provides information about recommended ways to use sessions with a Layer 7 load-balancer.

Most load-balancers have the ability to provide cookies for pinning subsequent traffic from a client to the appropriate server. This method is called sticky session, session persistence or session affinity. Using cookies efficiently requires the client to know when a cookie should be sent, should not be sent or should be deleted.

Sticky session load balancing

Each Metadefender Core instance has its own databases and application sessions that cannot be seen by other instances. Therefore, in order to get the related data/response to our queries we should ask the appropriate Metadefender Core v5 server. On the other hand, to keep the advantages of the used load balancing method, cookies should not be sent if it is not necessary.

Single file scanning

Step	Stage	Task	Cookie usage
1.	Sending file	Initiate processing a file on the client side. Send file through the load-balancer. (See POST File Analysis API)	Cookie should not be sent
2.	Sending file	Save the cookie and the data_id you got from the load-balancer	Save cookie
3.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Send cookie saved in step 2
4.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Send cookie saved in step 2

Batch scanning

Step	Stage	Task	Cookie usage
1.	Open batch	Initiate processing file(s) in batch. Request a batch ID through the load-balancer. (See POST Initiate Batch API)	Cookie should not be sent
2.	Open batch	Save the cookie and the batch_id you got from a Core server through the load-balancer.	Save cookie
3.	Sending files	Send file through the load-balancer. (See POST File Analysis (using batch header) API)	Send cookie saved in step 2
4.	Sending files	Save the data_id you got from the load-balancer.	-
	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Send cookie saved in step 2
5.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
6.	Getting batch status	See GET Batch Status API	Send cookie saved in step 2
7.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Send cookie saved in step 2
8.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API	Send cookie saved in step 2

If it does not matter which upstream server responds, the queries should be sent without cookie.

> **Info**
> It is recommended not to send cookies when it's not necessary to allow load-balancer to use its own method to share the load between Metadefender Core v5 servers.

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration through load-balancer

OPSWAT products that support HTTP load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	4.3.4	-
OPSWAT Client	Windows: 7.6.247.0 Mac: 10.4.243.0	-
MetaDefender Email Security	4.3.0	-
MetaDefender ICAP Server	4.3.0	-
MetaDefender Vault	1.3.0	-
MetaDefender for Secure Storage	2.0.7	-

DNS Load Balancing

Using this method is logically similar to Layer 7 load-balancing.

Briefly how it works

Client uses a domain name to send a query to a server. Client's DNS server has more "A" records for that name with different IPs. When a client resolves the server's domain name DNS server randomly chooses an IP for that name to send back. When a session is used on the application layer, client should know the IP address of the Core server that handles that specific session. In every other case, client should resolve the domain name with DNS query to let requests to be balanced between Metadefender Core servers.

Single file scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Sending file	Initiate processing a file on the client side. (See POST File Analysis API)	Use IP saved in step 2
4.	Sending file	Save the data_id got from the Core server	-
5.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Use IP saved in step 2
6.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Use IP saved in step 2

Batch scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Open batch	Initiate processing file(s) in batch. Request a batch ID. (See POST Initiate Batch API)	Use IP saved in step 2
4.	Sending files	Send file to the specific Core server with the basch ID saved in step 3. (See POST File Analysis (using batch header) API)	Use IP saved in step 2
5.	Sending files	Save the data_id you got from the Core server.	-
6.	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Use IP saved in step 2
7.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
8.	Getting batch status	See GET Batch Status API	Use IP saved in step 2
9.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Use IP saved in step 2
10.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API).	Use IP saved in step 2

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration via DNS load-balancing

OPSWAT products that support DNS load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	does not support yet	-
MetaDefender Client	does not support yet	-
MetaDefender Email Security	does not support yet	-
MetaDefender ICAP Server	does not support yet	-
MetaDefender Vault	does not support yet	-

SSL connection for PostgreSQL communication

> **Note**
> The SSL connection configuration is possible thanks to the support made by PostgreSQL. For reference: https://www.postgresql.org/docs/14/ssl-tcp.html
>
> MetaDefender Core is tested against this mode on particular version 4.21.1

Step 1: Generate SSL certificate for PostgreSQL server

> **Note**
> Following instructions are for self-sign certificate. For production environment, making sure to use the certificate signed by your trusted CA.

1) Generate a private key

bash

```
openssl genrsa -des3 -out server.key 1024
```

Sample output:

bash

```
Generating RSA private key, 1024 bit long modulus ........++++++
...............++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

Type your desired password / pass phrase to encrypt the private key

2) Remove the pass phrase to automatically start up the PostgreSQL server

bash

```
openssl rsa -in server.key -out server.key
```

> **Note**
> On Linux, make sure to set permission on the server.key file

bash

```
chmod og-rwx server.key
```

3) Create a self-signed certificate

bash

```
openssl req -new -key server.key -days 3650 -out server.crt -x509
```

You will be prompted to enter detailed information which is incorporated into your self-signed certificate request.

none

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:CA
Locality Name (eg, city) [Default City]:San Francisco
Organization Name (eg, company) [Default Company Ltd]:OPSWAT
Unit Name (eg, section) []:OPSWAT
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:dummy_email@opswat.com
```

For self-signed certificate, use the server ceriticate as the trusted root certificate:

bash

```
cp server.crt root.crt
```

Copy **server.key**, **server.crt**, **root.crt** to PostgreSQL data folder. For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data**
- Linux: **/var/lib/ometascan/pg_data/**

Step 2: Configure PostgreSQL server for SSL authentication connection

1) Create a custom config file (e.g. ssl.conf) for PostgreSQL.

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\postgres\ssl.conf**
- Linux: **/var/lib/ometascan/postgres/ssl.conf**

2) Content of ssl.conf:

none

```
ssl = on
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
```

3) Modify pg_hba.conf

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data\pg_hba.conf**
- Linux: **/var/lib/ometascan/pg_data/pg_hba.conf**

Add following (modify **::1/128** if needed to match with your environment requirement)

none

```
hostssl all all      ::1/128      scram-sha-256
```

After the modify, the content of pg_hba.conf should be like this (example):

> **Note**
> In a Linux environment, a local connection is required to upgrade from version 5.5.1 or earlier to version 5.6.0 or later.

none

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local         DATABASE  USER  METHOD  [OPTIONS]
# host          DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl     DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostgssenc    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnogssenc  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
local    all all                    scram-sha-256
hostssl all all      ::1/128      scram-sha-256

#More content.......
```

4) Reload the services

- Standalone DB mode: Restart **ometascan** service (MetaDefender Core service)
- Shared DB mode: Restart **ometascan-pg** / **ometascan-postgresql** service (MetaDefender Core PostgreSQL service) and then **ometascan** service (MetaDefender Core service)

Test your SSL database connection:

bash

```
psql.exe "sslmode=require dbname=postgres user=postgres host=localhost port=5432"
Password for user postgres:
psql (12.7)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
Type "help" for help.
```

This message indicate that you are now using SSL connection:

bash

```
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
```

Performance and Load Estimation

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Some factors that affect performance

- MetaDefender Core version
- MetaDefender Core engine package and configuration
  - set of engines (which and how many)
  - product configuration (e.g., thread pool size)
- System environment
  - server profile (CPU, RAM, hard disk)
  - client application location - remote or local
  - system caching and engine level caching
- Dataset
  - encrypted or decrypted
  - file types
    - different file types (e.g., document, image, executable)
    - archive file or compound document format files
  - file size
  - bad or unknown (assume to be clean)
- Performance tool

How test results are calculated

Performance (mainly scanning speed) is measured by throughput rather than unit speed. For example, if it takes 10 seconds to process 1 file, and it also takes 10 seconds to process 10 files, then performance is quantified as 1 second per file, rather than 10 seconds.

- total time / total number of files processed: 10 seconds / 10 files = 1 second / file.

Standalone DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.2.416.0

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

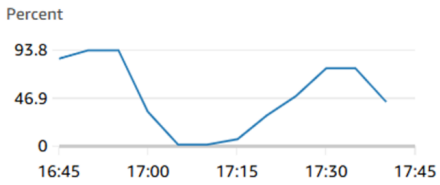
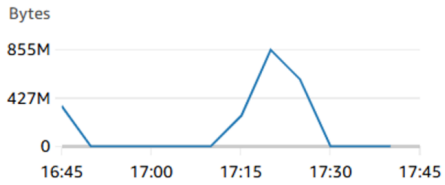
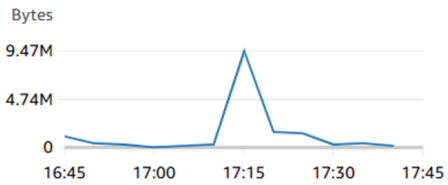
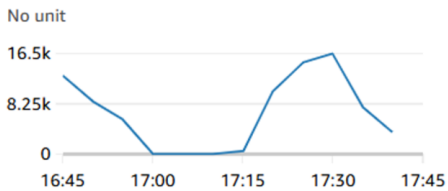
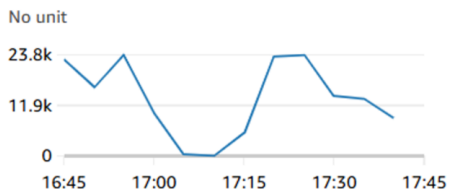
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV)	24.2	47.2 / 80	37.3 / 41.1		315,310	0.0114
Deep CDR	19.9	48.9 / 81.8	32.6 / 37		383,442	0.0094
Proactive DLP	17.1	36.8 / 58.2	30.4 / 33.6		446,228	0.0081
Vulnerability	10.1	59.2 / 85	29.3 / 33.3		755,495	0.0048

MetaDefender Core with common engine packages:

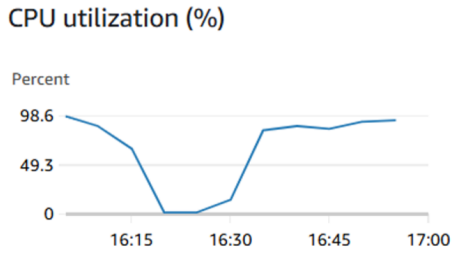
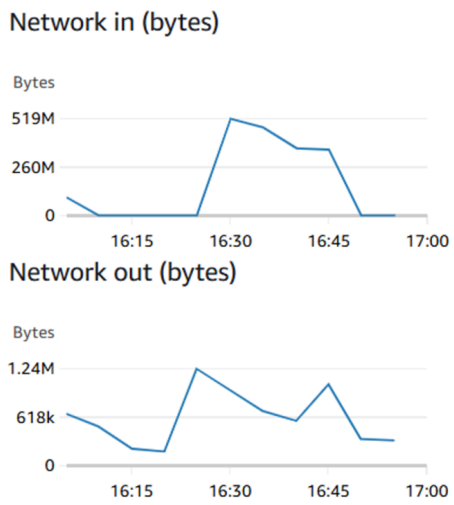
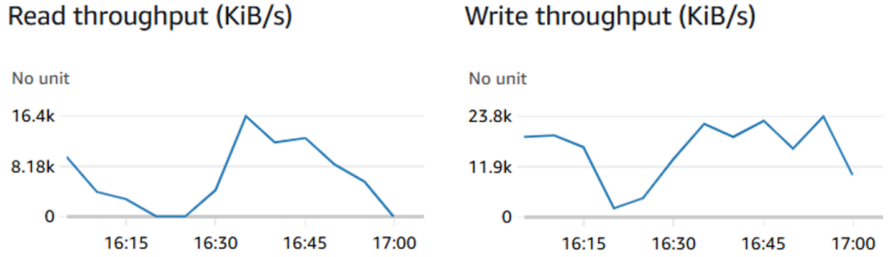
Use case	Scan duration	Avg./Max CPU usage	Avg./Max RAM usage	Avg./Max Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV) + Deep CDR	30.8	68.1 / 88	40.1 / 46.8		247,744	0.0145
Metascan (8 AV) + Deep CDR + Proactive DLP	31.2	74.2 / 98.6	41.5 / 51.1		244,567.3	0.0147
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	31.8	73.5 / 98	40.6 / 51.5		239,953	0.015

System resource utilization charts

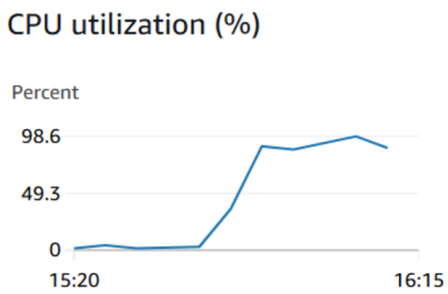
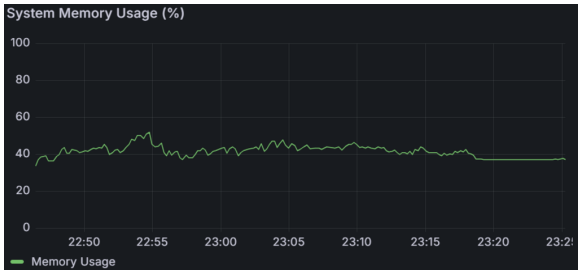
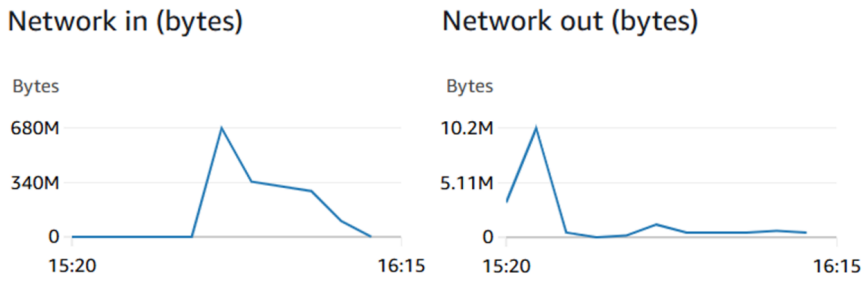
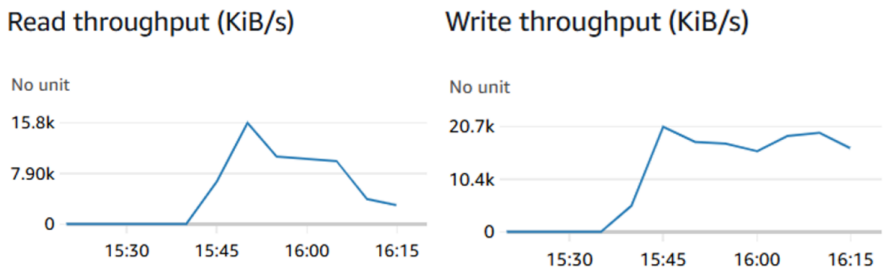
MetaDefender Core with Metascan (8 AV engines):

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

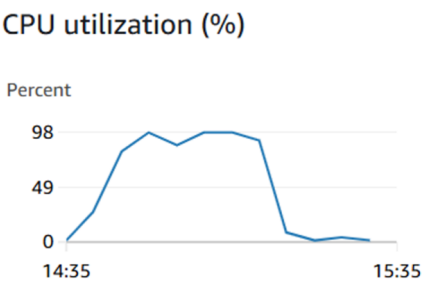
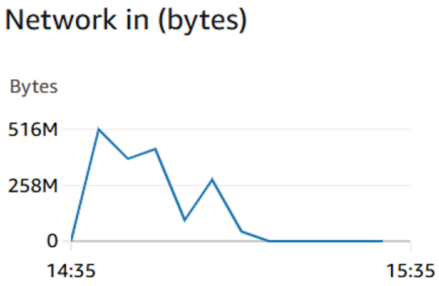
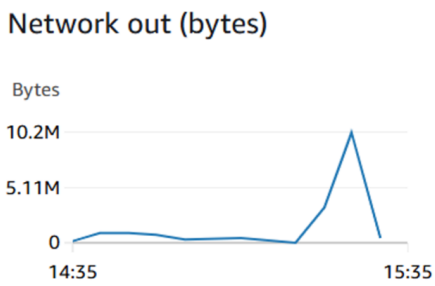
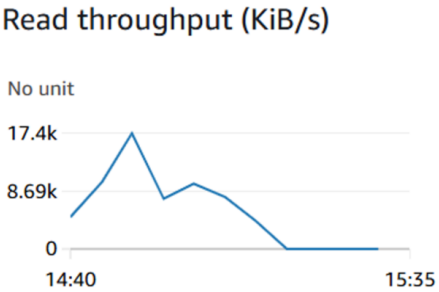
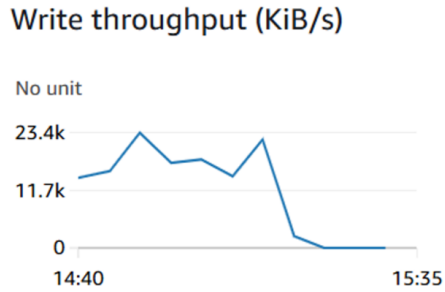
MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

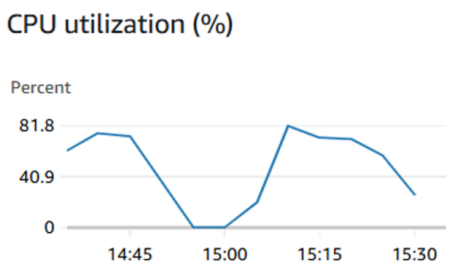
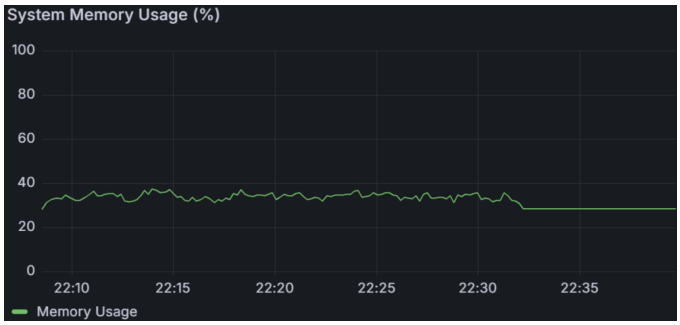
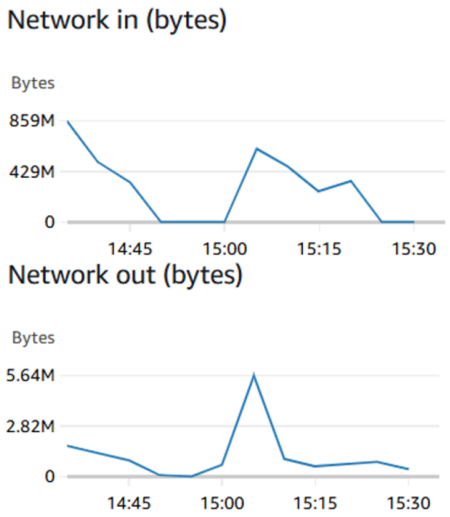
MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

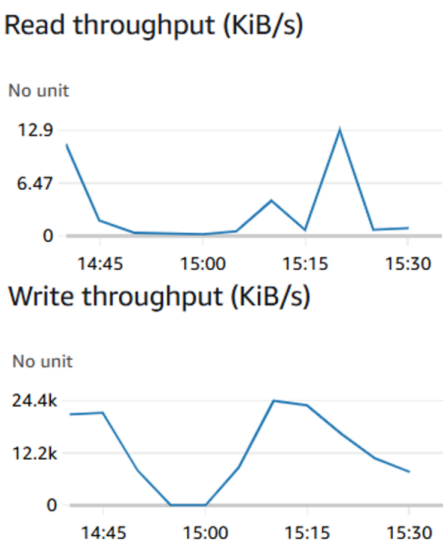
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

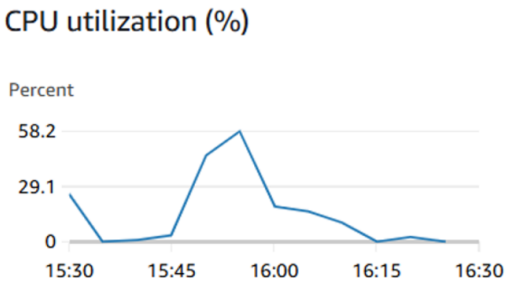
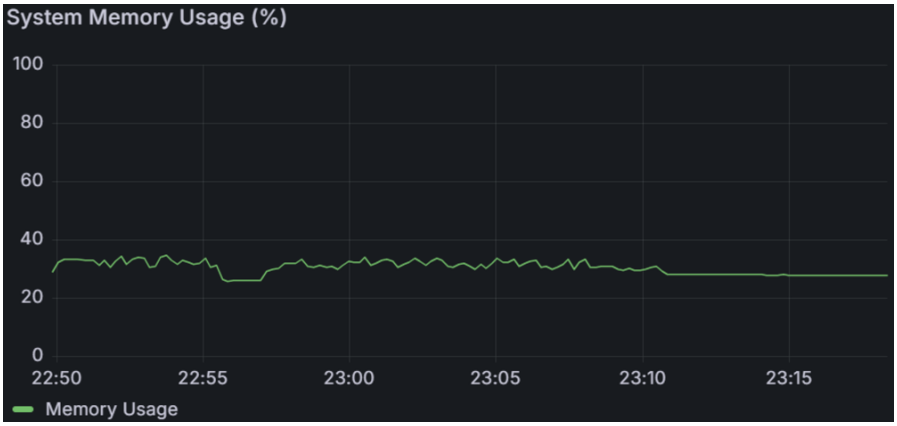
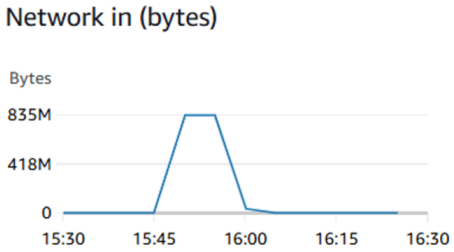
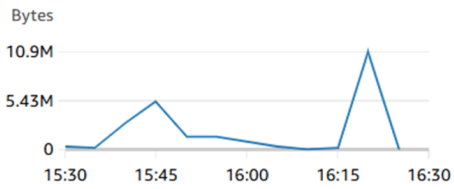
	Utilization charts
CPU usage	
Memory usage	
Network usage	 
Disk usage	 

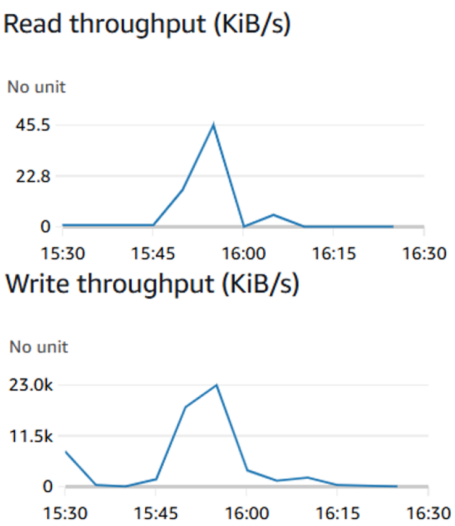
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

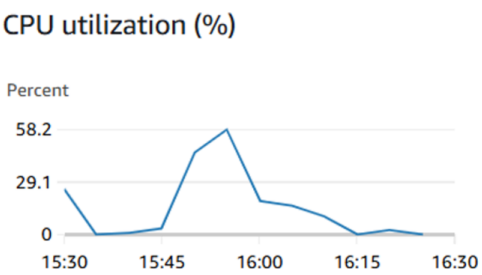
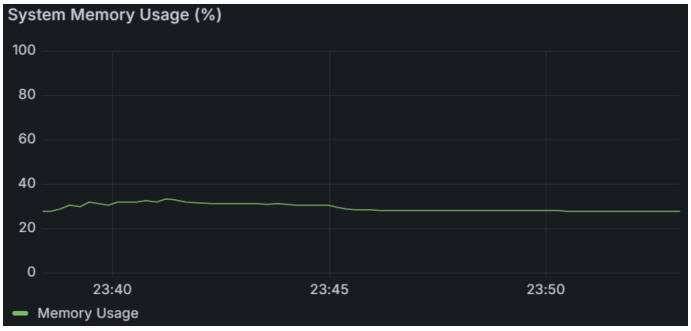
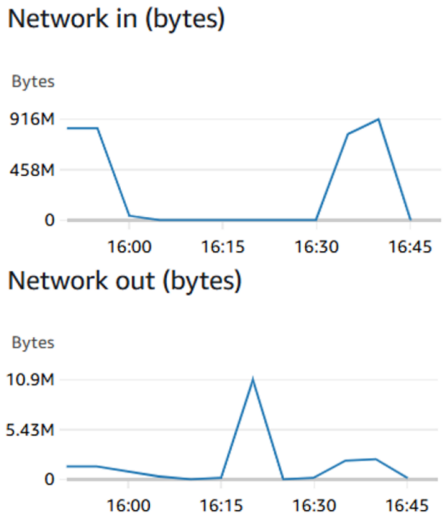
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	 

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability:

	Utilization charts
CPU usage	
Memory usage	
Network usage	

Utilization charts

Disk usage



Standalone DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Ubuntu 22.04	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.57-236

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

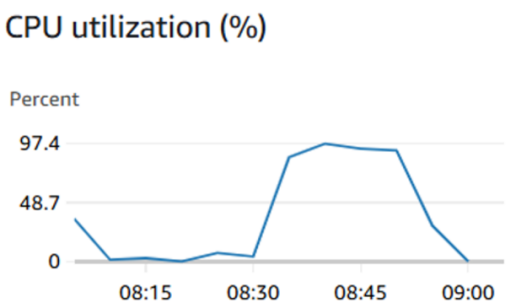
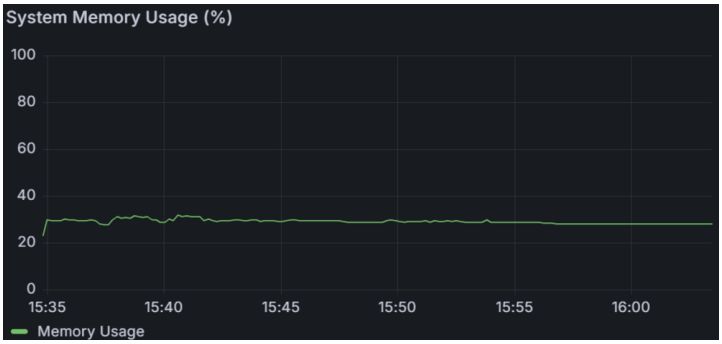
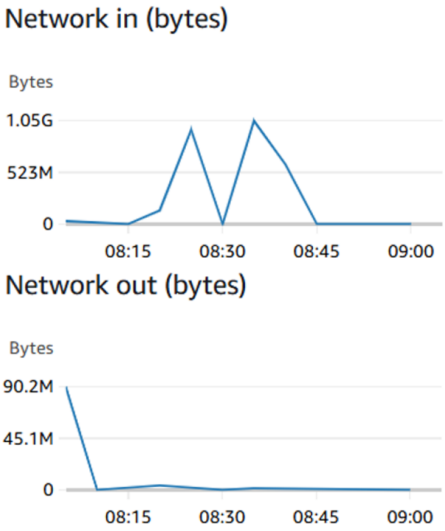
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV)	30.2	69 / 98	29 / 32		252,666	0.0142
Deep CDR	19.4	48 / 62	36 / 49		393,325	0.009
Proactive DLP	16.5	34 / 39	32 / 48		462,455	0.008
Vulnerability	12	50 / 65	46 / 46		635,875	0.006

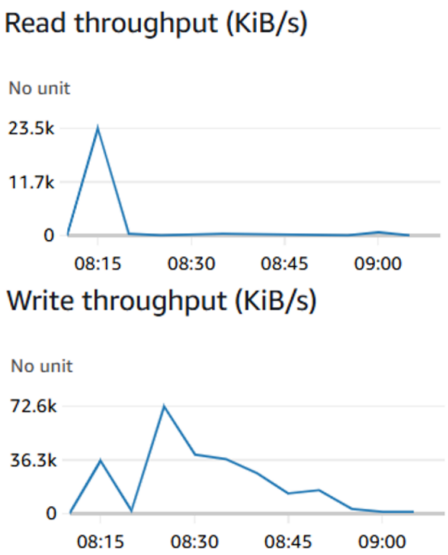
MetaDefender Core with common engine packages:

Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV) + Deep CDR	39.8	78 / 99	62 / 73		191,721	0.0188
Metascan (10 AV) + Deep CDR + Proactive DLP	43.5	61 / 84	47 / 67		175,414	0.0205
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	44.8	62 / 88	69 / 83		170,324	0.0211

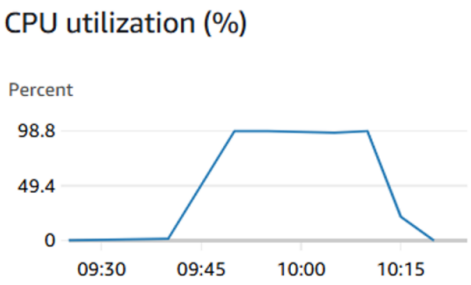
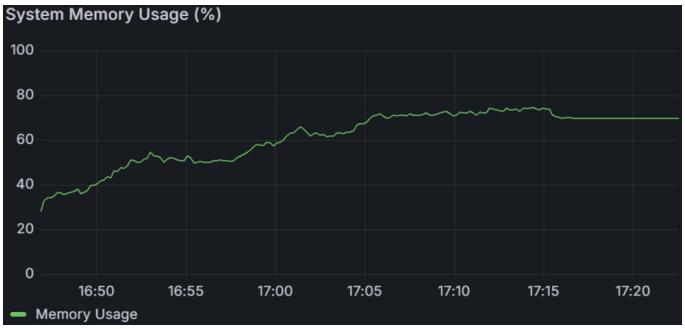
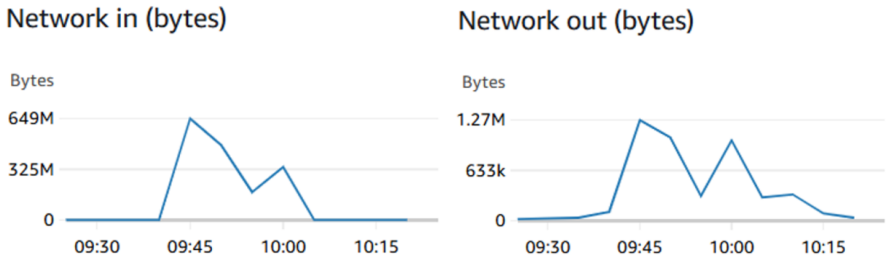
System resource utilization charts

MetaDefender Core with Metascan (10 AV engines) :

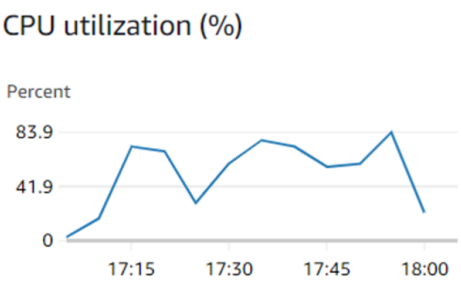
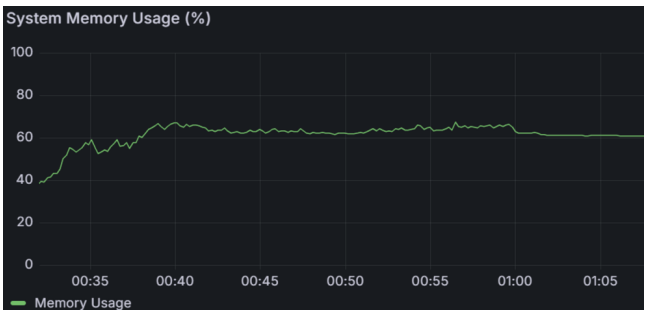
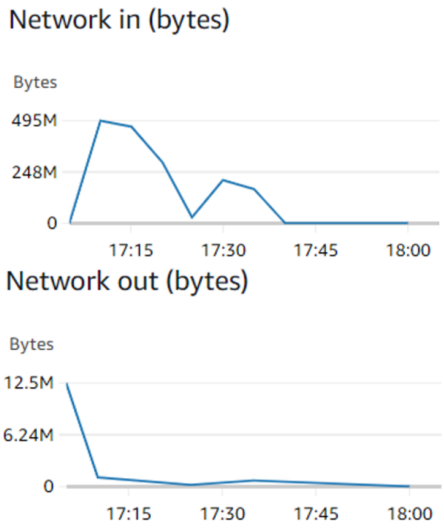
	Utilization charts
CPU usage	
Memory usage	
Network usage	

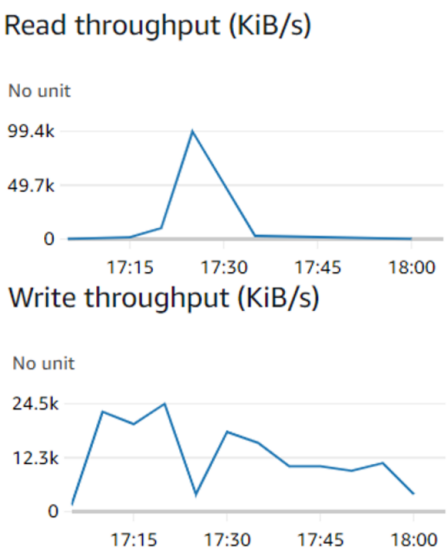
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR :

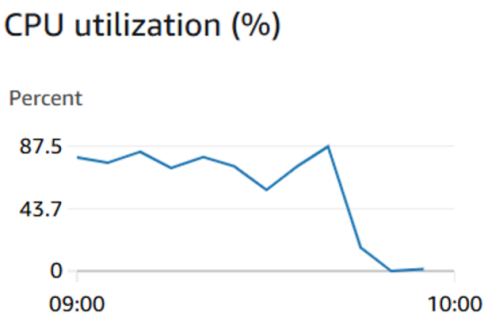
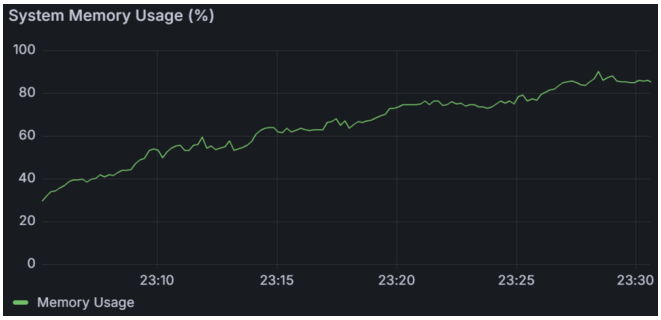
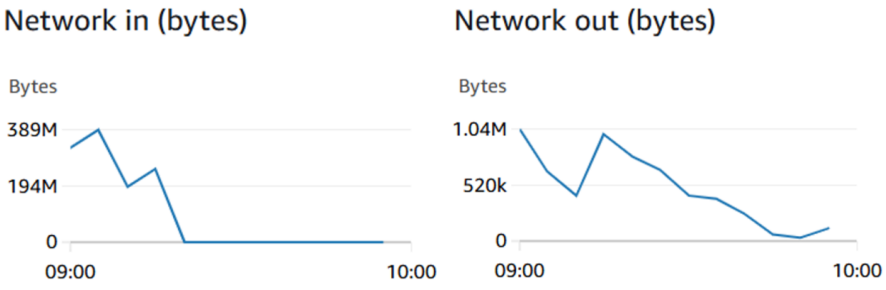
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP :

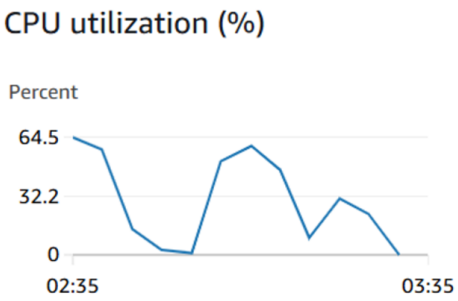
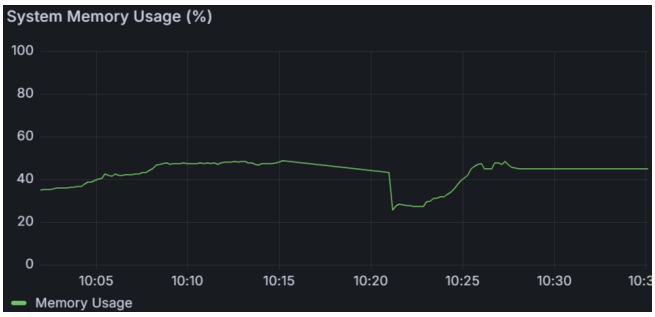
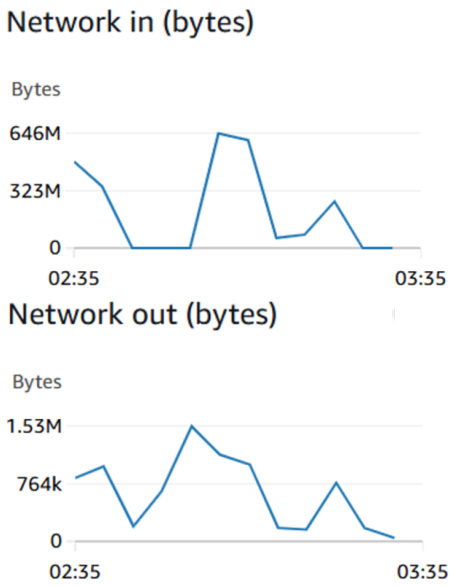
	Utilization charts
CPU usage	
Memory usage	
Network usage	

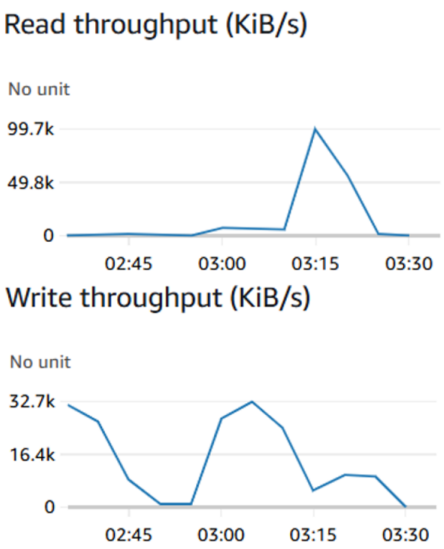
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP + Vulnerability :

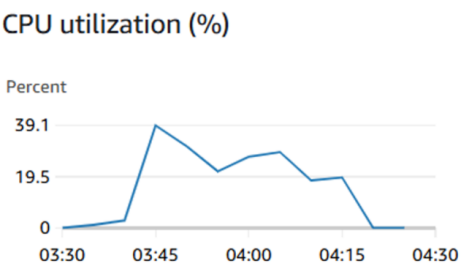
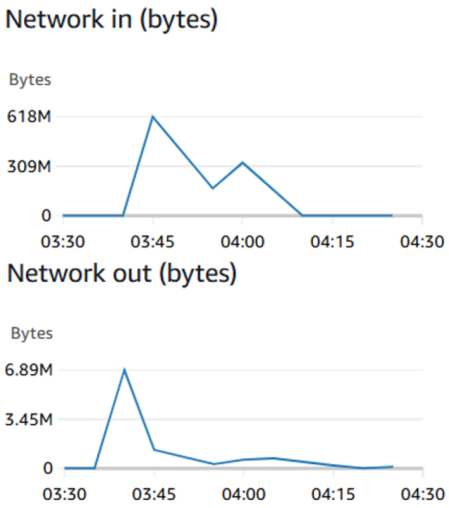
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

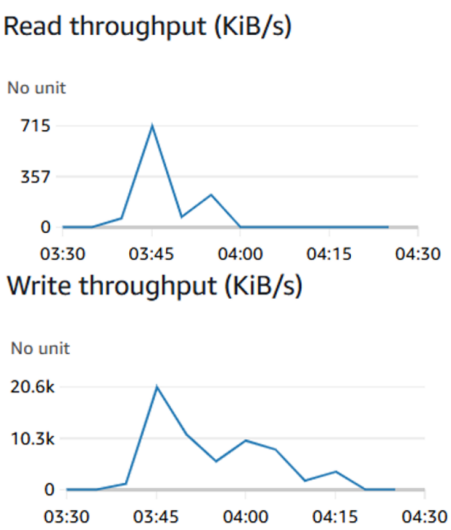
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

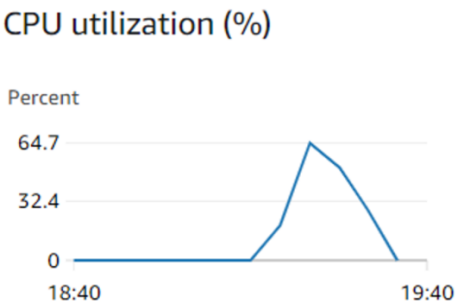
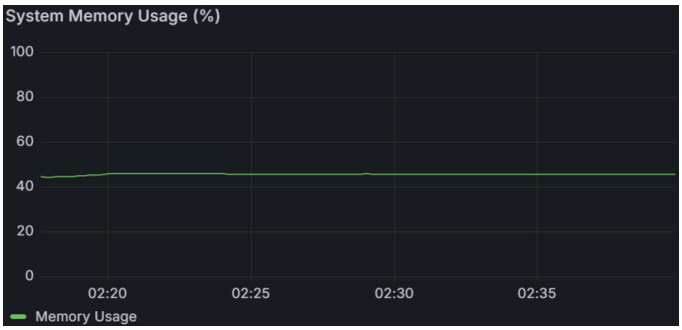
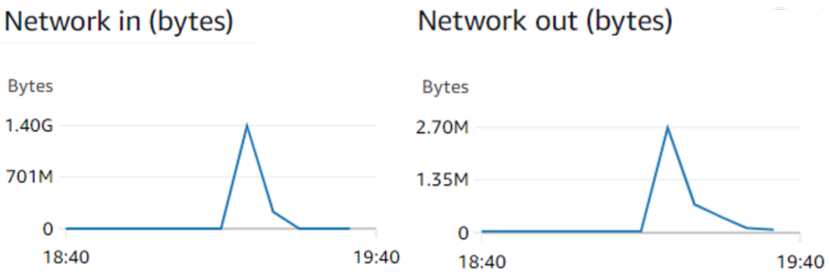
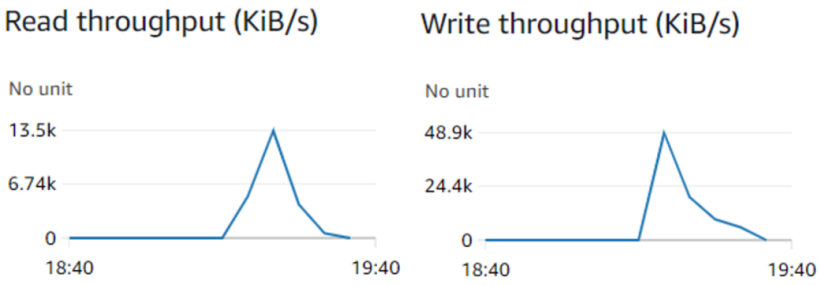
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability :

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

Shared DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

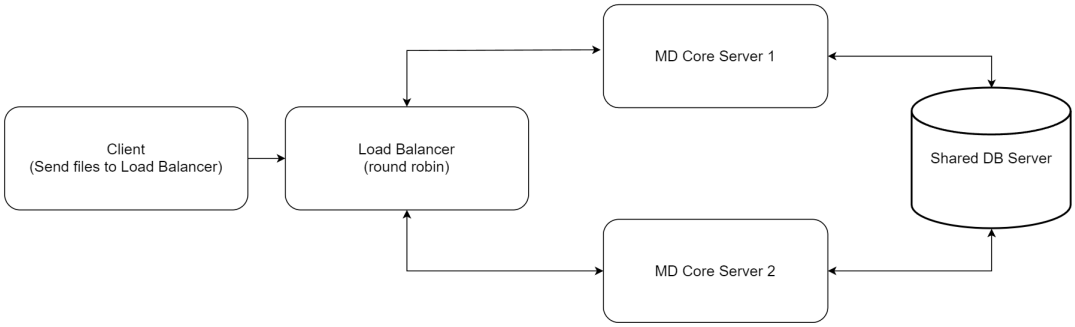
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV)	11.9	37.0	41.9	30.1	30.4	Receive: 6441.1 Send: 769.6	Receive: 6200.2 Send: 693.7	640,821.2	0.0
Deep CDR	10.5	60.1	66.7	30.3	32.2	Receive: 6379.8 Send: 594.8	Receive: 7146.1 Send: 609.6	729,368.5	0.0
Proactive DLP	7.8	38.7	37.5	29.7	31.0	Receive: 8842.6 Send: 791.8	Receive: 8635.6 Send: 729.1	965,455.7	0.0
Vulnerability	4.4	52.2	53.0	26.0	26.9	Receive: 16,215.5 Send: 1006.8	Receive: 15,785.7 Send: 997.2	1714,449.3	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV) + Deep CDR	15.4	72.4	75	34.3	35.4	Receive: 5436.6 Send: 611.3	Receive: 5589.3 Send: 607.2	495,697.4	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP	16.9	77.2	80.2	36.4	37.9	Receive: 5254 Send: 585.4	Receive: 5041.3 Send: 576.1	450,657.6	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	18.0	72.1	80.4	38.6	38.0	Receive: 4504.3 Send: 574.1	Receive: 4649.5 Send: 571.1	423,010	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

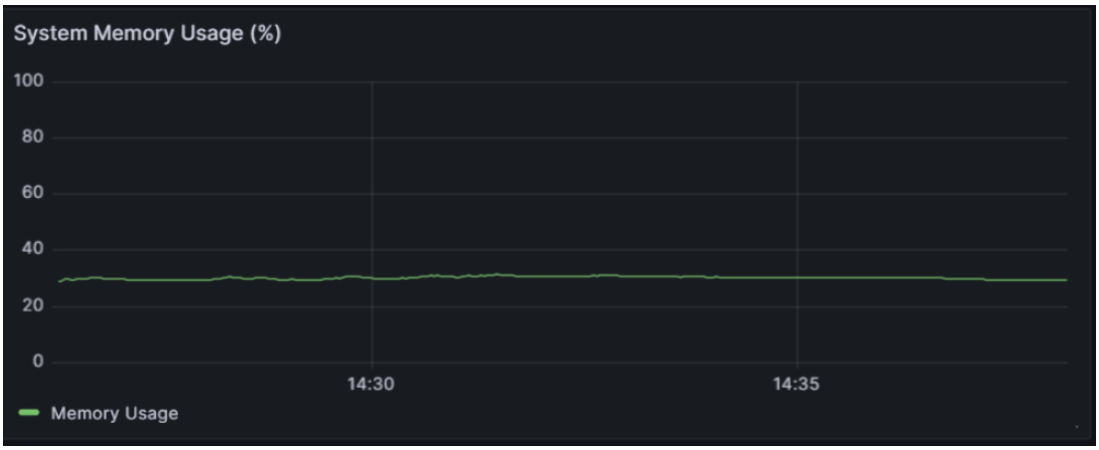
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2

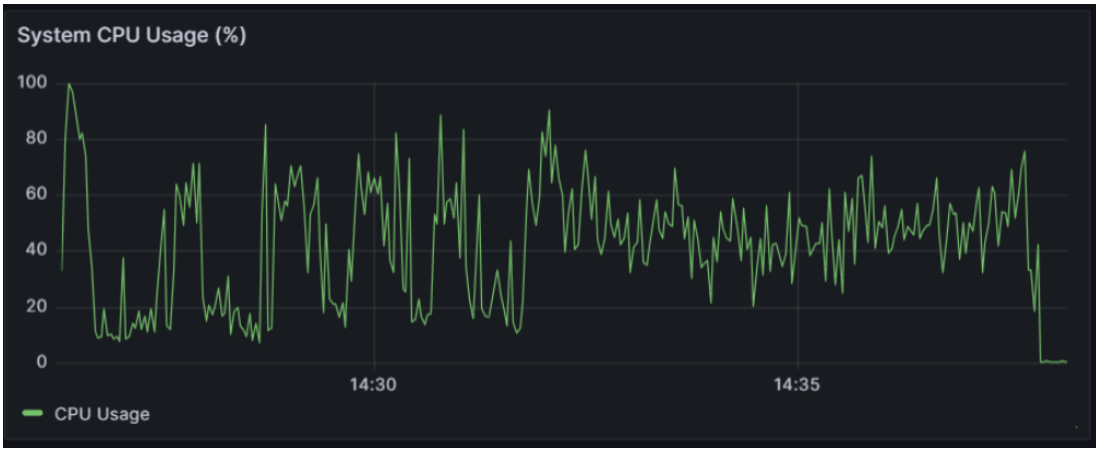


CPU usage

MD Core #1

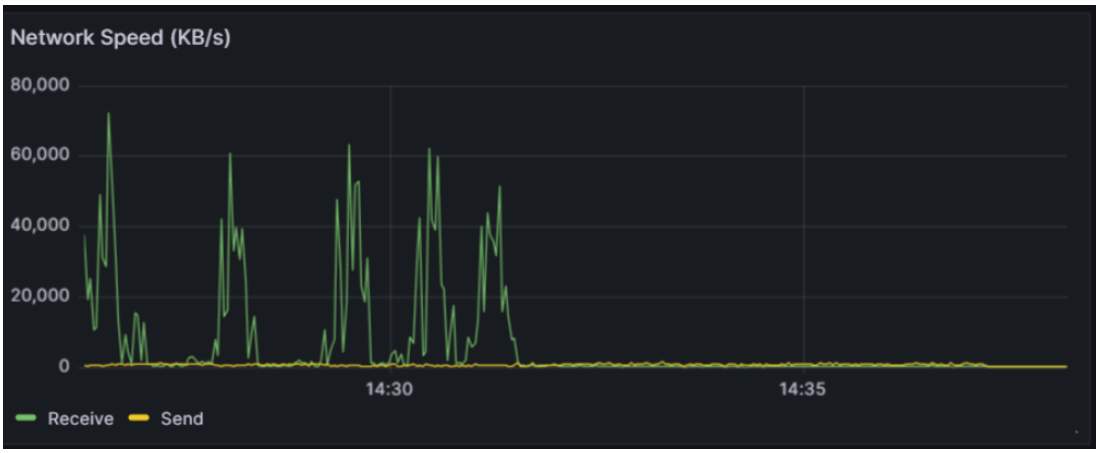


MD Core #2

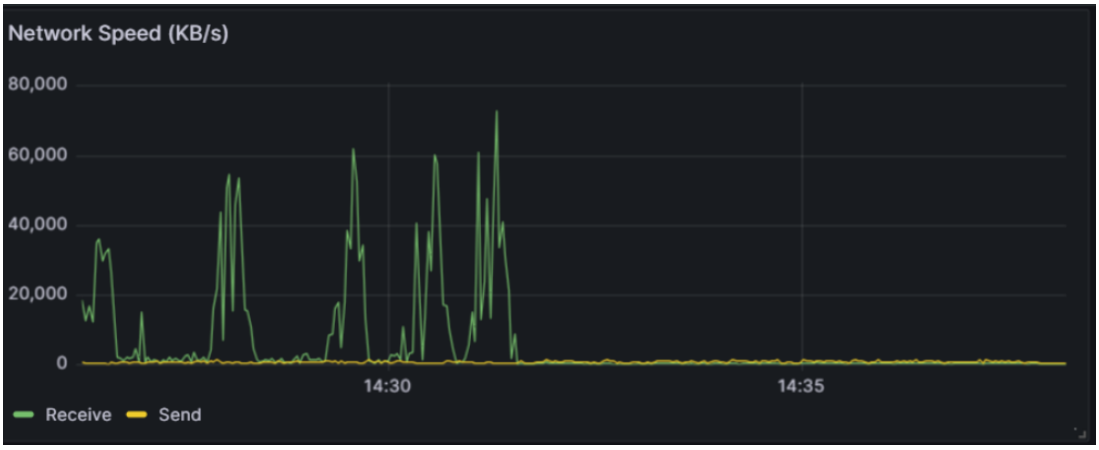


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2

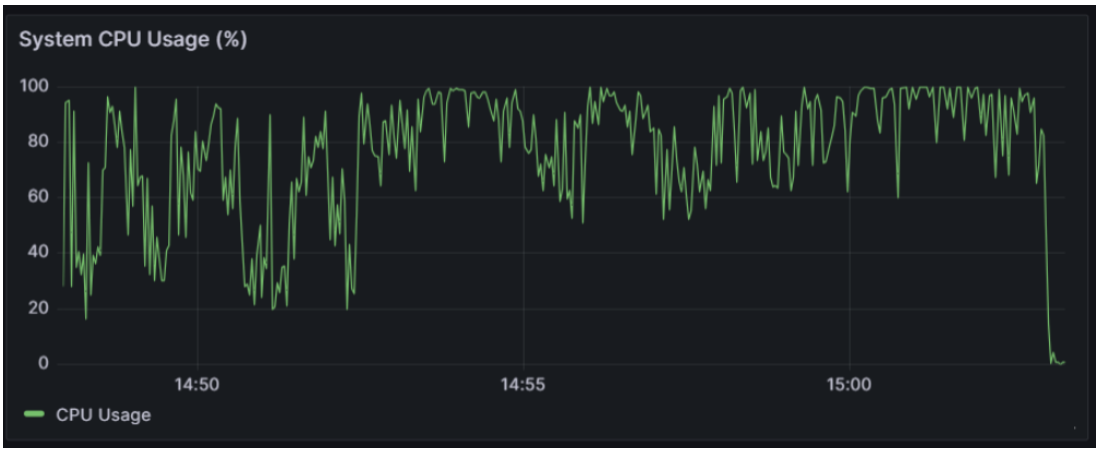


CPU usage

MD Core #1

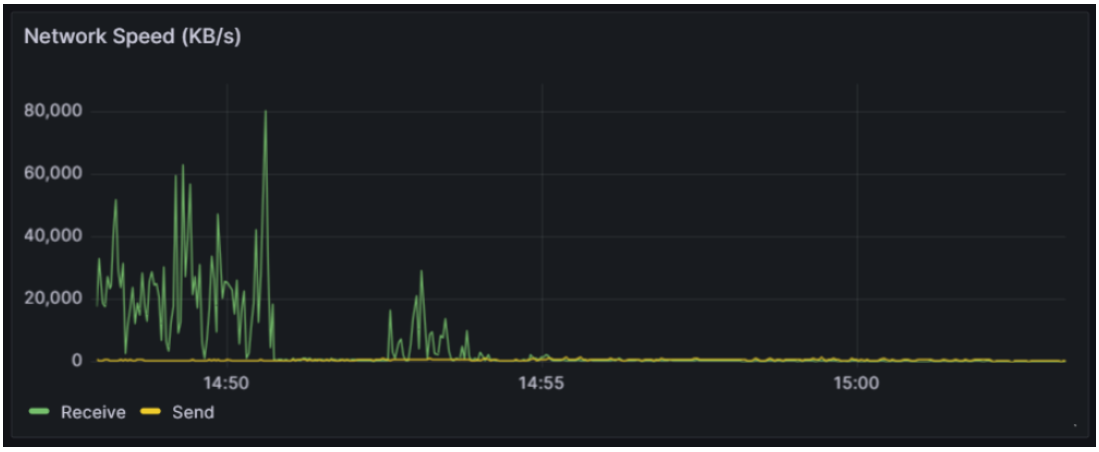


MD Core #2



Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

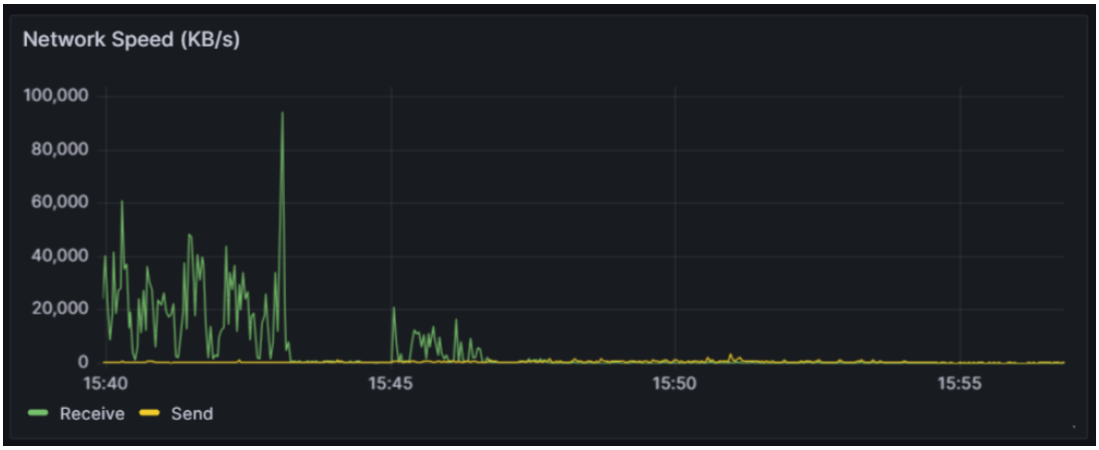


MD Core #2

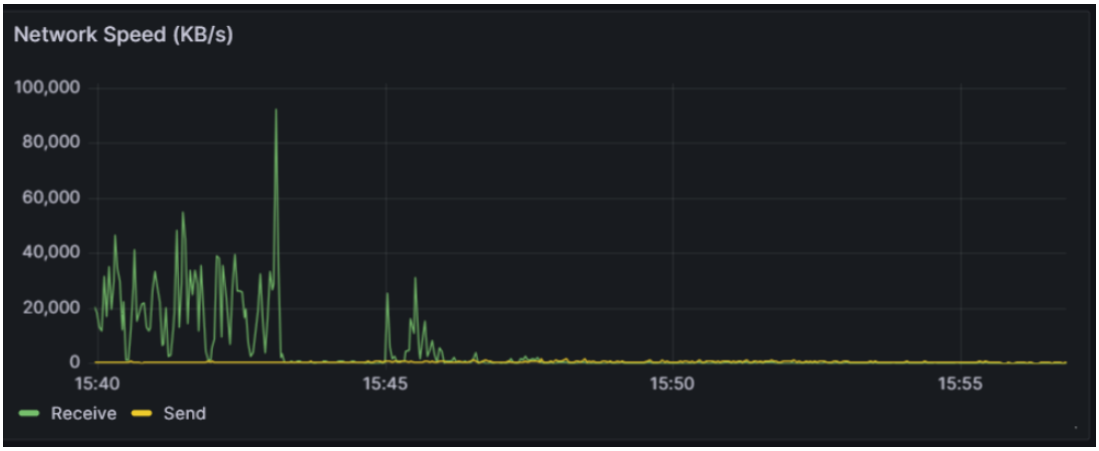


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1

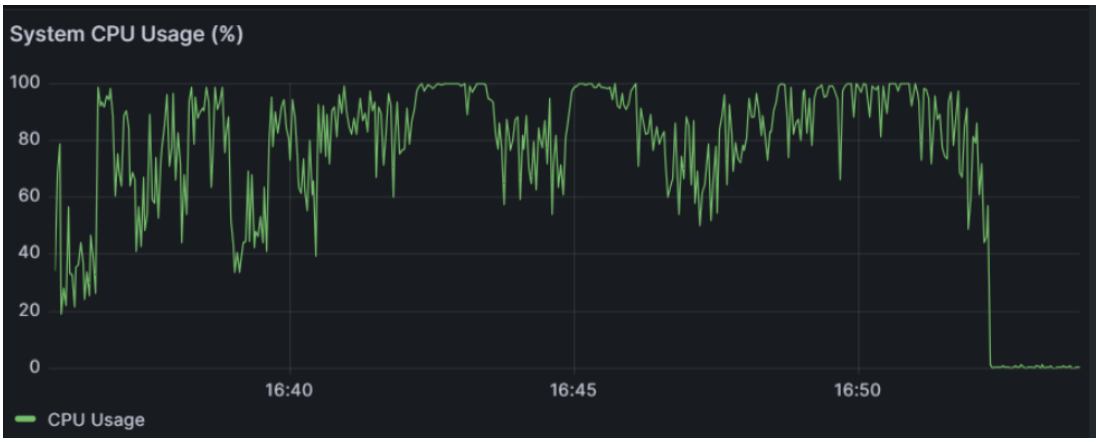


MD Core #2



CPU usage

MD Core #1

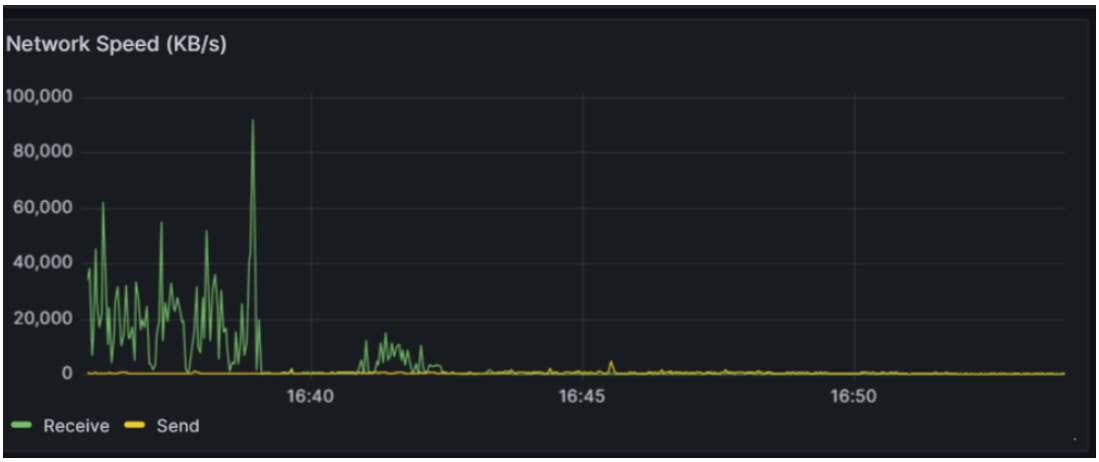


MD Core #2

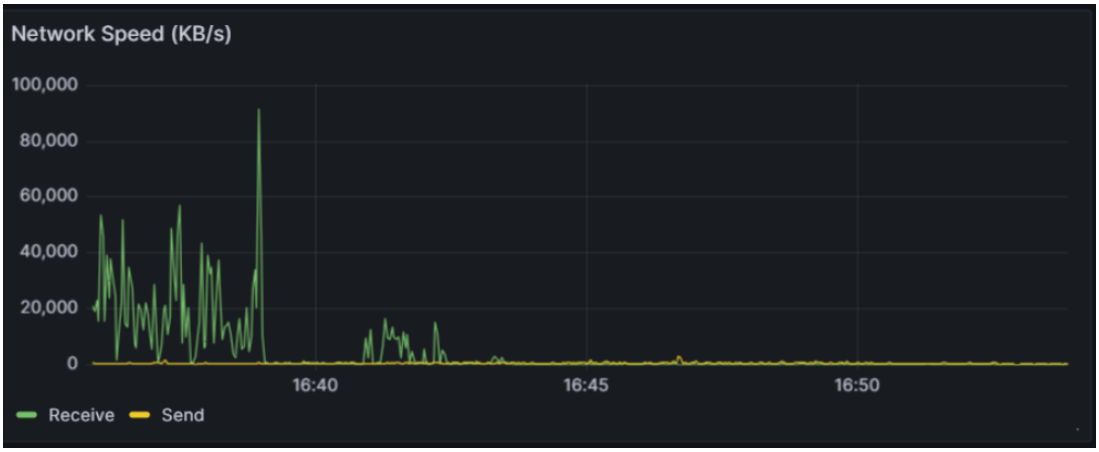


Network usage

MD Core #1



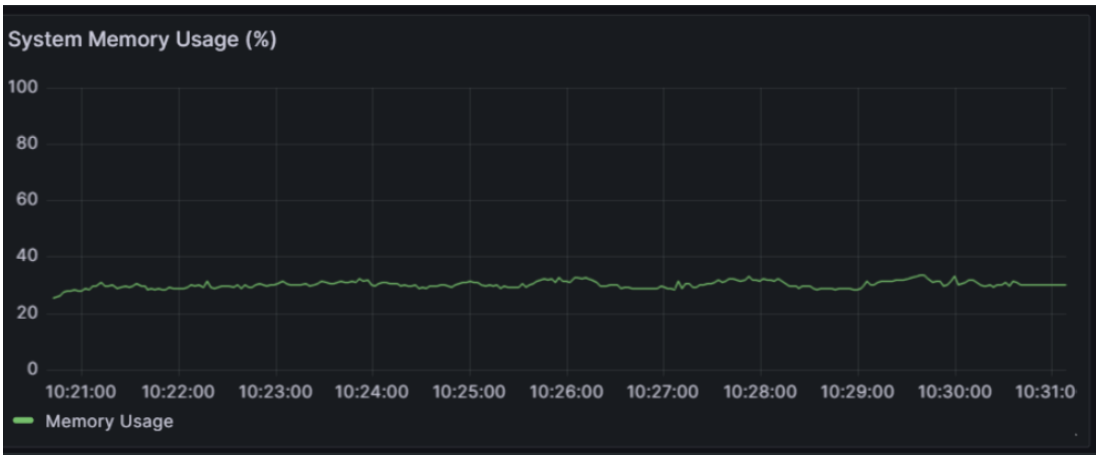
MD Core #2



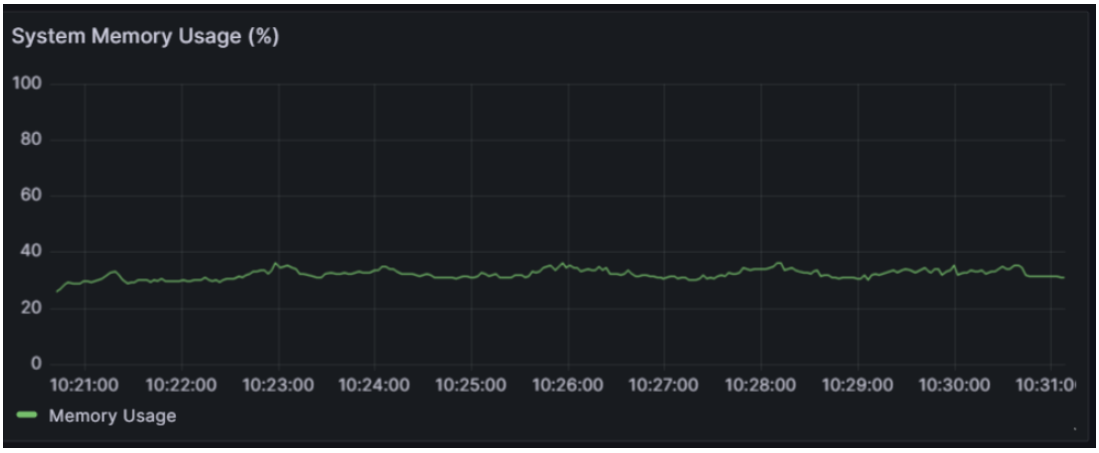
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

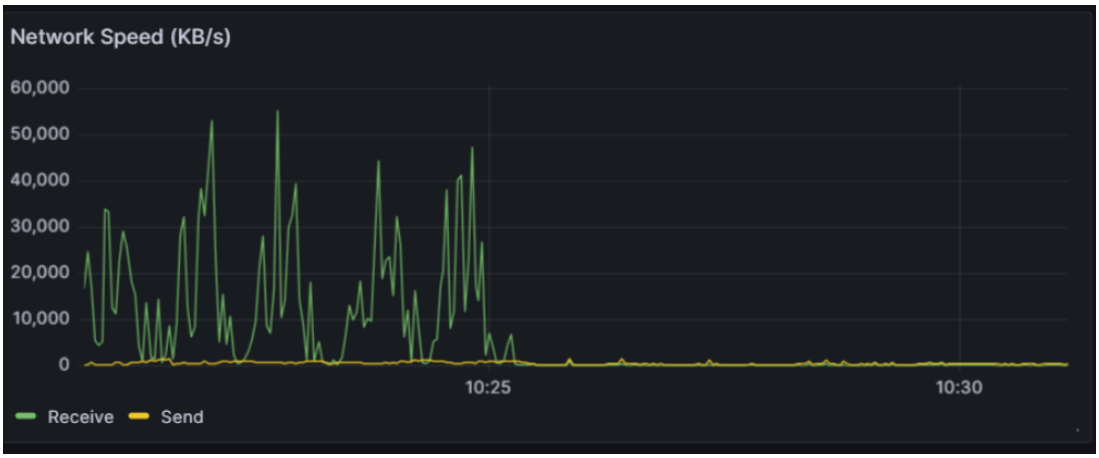


MD Core #2

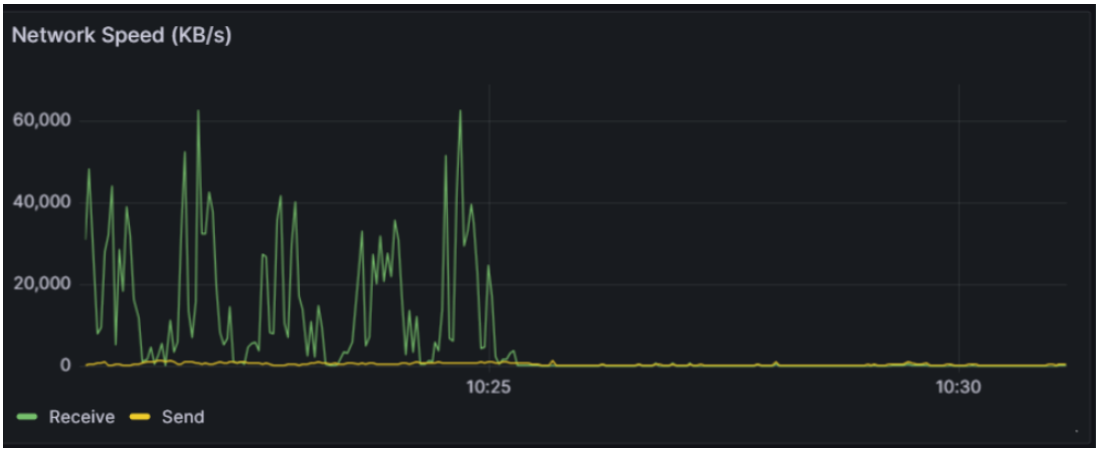


Network usage

MD Core #1



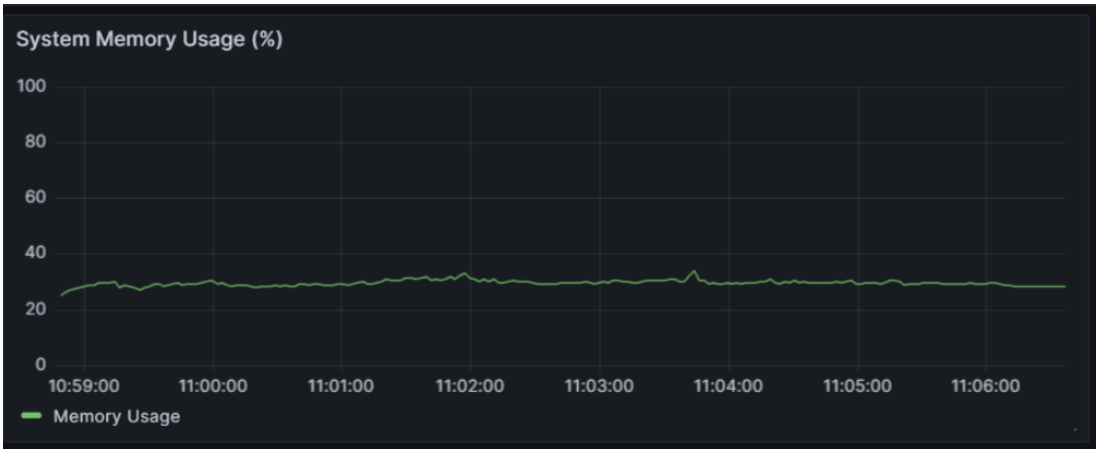
MD Core #2



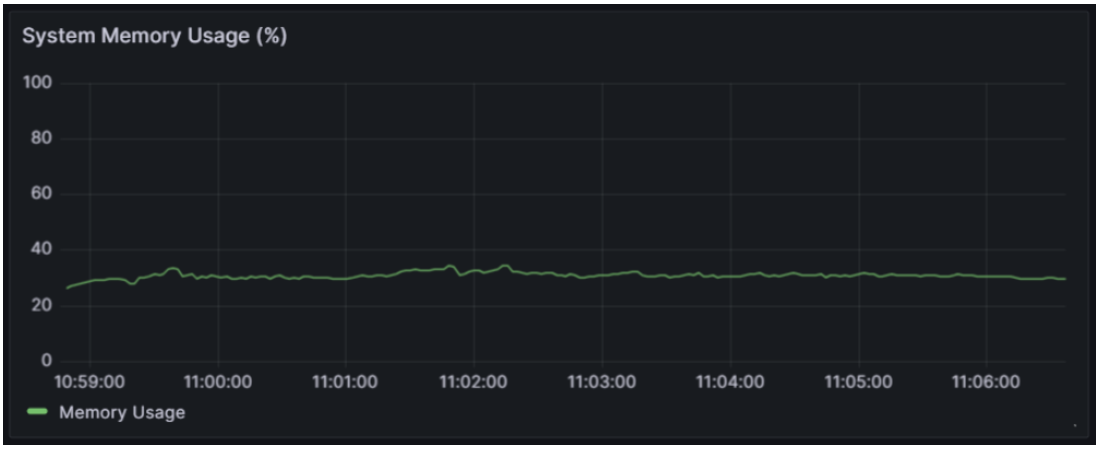
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

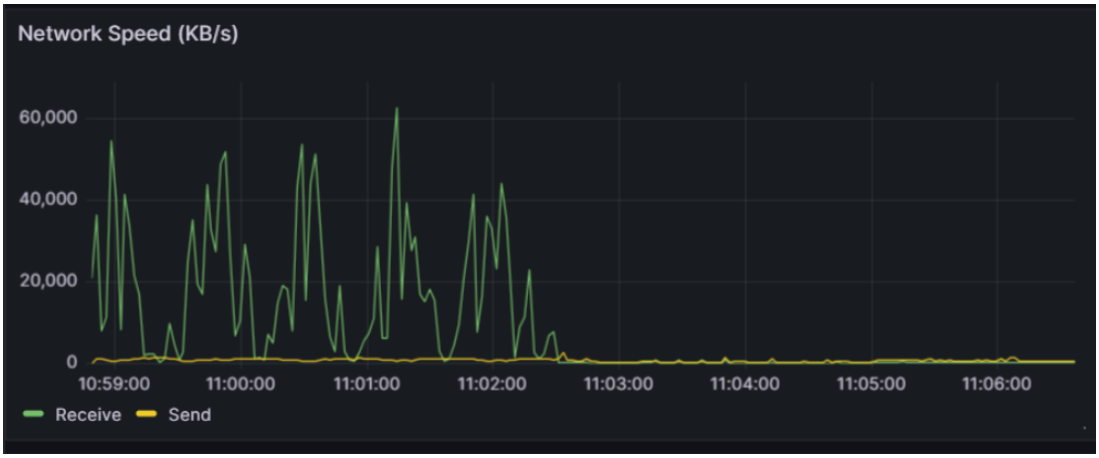


MD Core #2

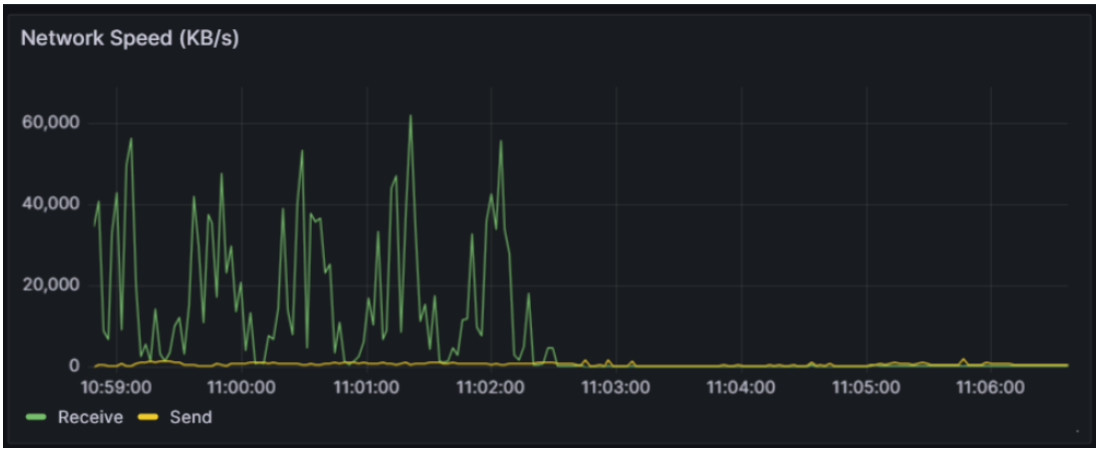


Network usage

MD Core #1



MD Core #2



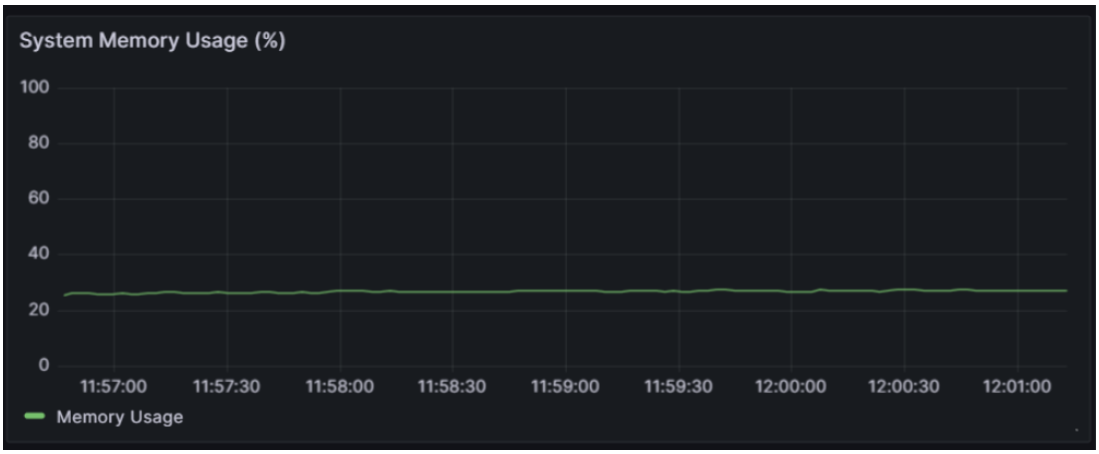
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

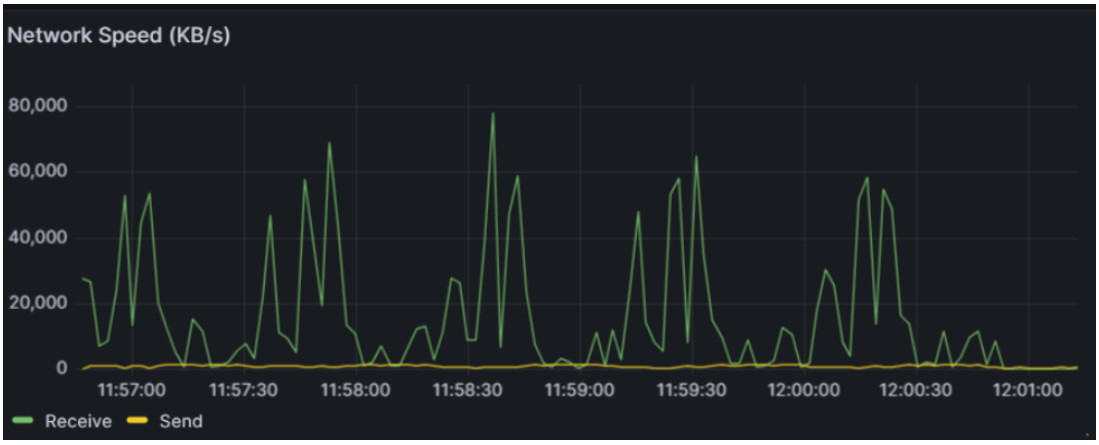


MD Core #2

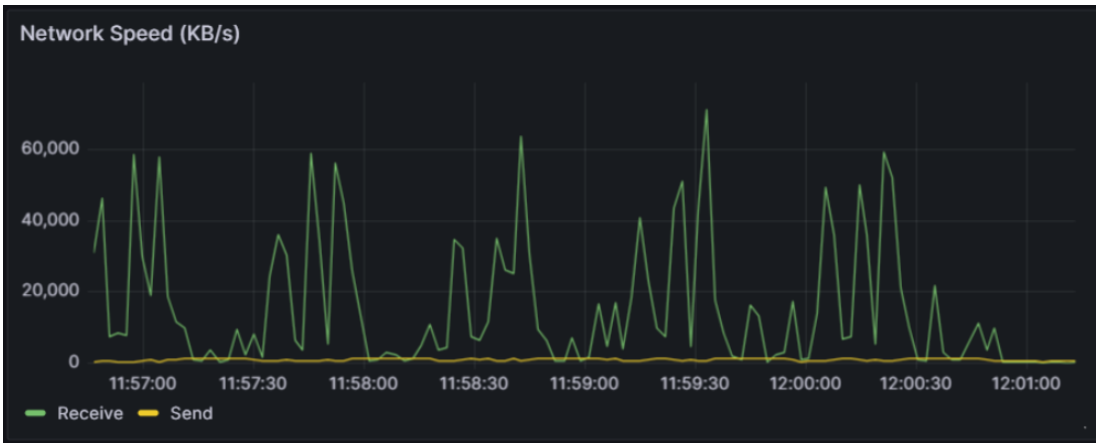


Network usage

MD Core #1



MD Core #2



Shared DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

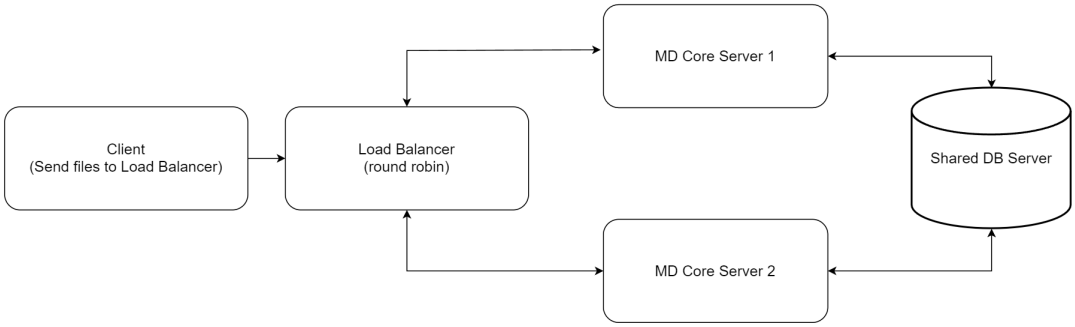
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV)	15.5	63.1	73.3	25.2	25.8	Receive: 4555.9 Send: 694.4	Receive: 4516.2 Send: 692.4	492,620.5	0.0
Deep CDR	9.70	50.7	51.2	31.9	32.5	Receive: 7062.9 Send: 623.2	Receive: 6653 Send: 654.1	786,189.6	0.0
Proactive DLP	7.2	35.2	35.0	27.9	27.5	Receive: 9047.4 Send: 802.4	Receive: 9859.8 Send: 787.5	1,055,110.8	0.0
Vulnerability	5.0	45.7	49.3	24.4	24.5	Receive: 13,585.7 Send: 966.7	Receive: 13,505.8 Send: 987.4	1,519,562.1	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV) + Deep CDR	19.9	79.1	77.6	34.0	34.2	Receive: 3785.9 Send: 573.8	Receive: 3663.1 Send: 577.7	382,716	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP	21.6	80.3	81.5	37.7	44.4	Receive: 3305.6 Send: 543.3	Receive: 3414.3 Send: 556.2	353,694.9	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	22.5	84.2	79.8	37.9	44.7	Receive: 3267.9 Send: 554.9	Receive: 3398.2 Send: 553.8	338,507.6	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

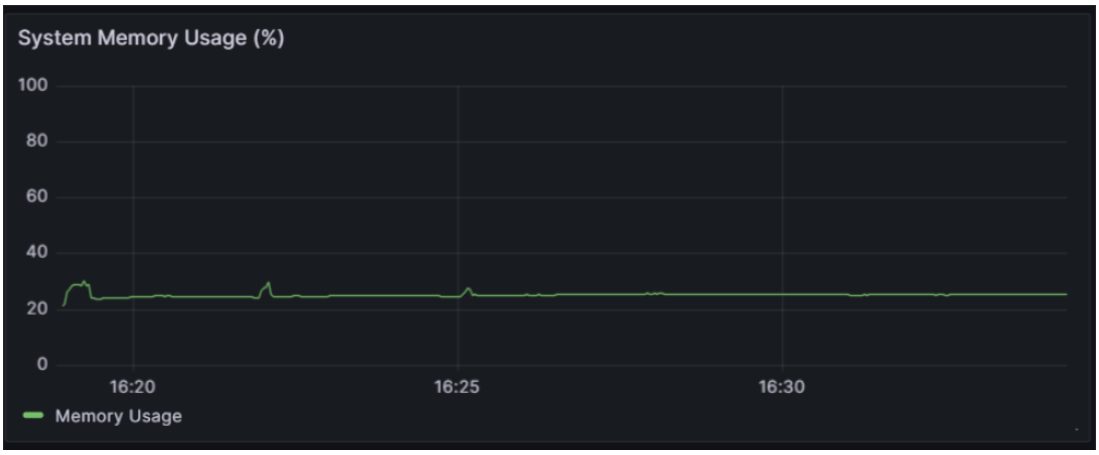
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

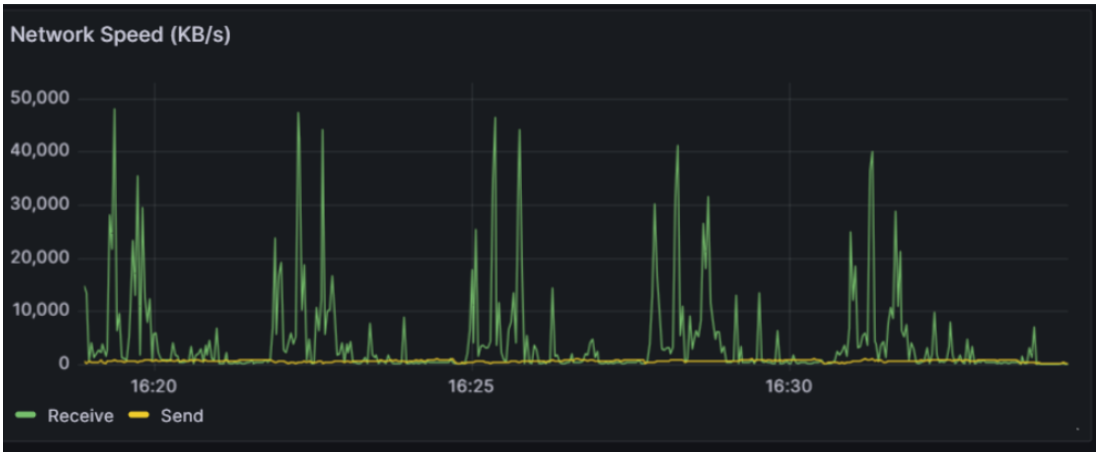


MD Core #2

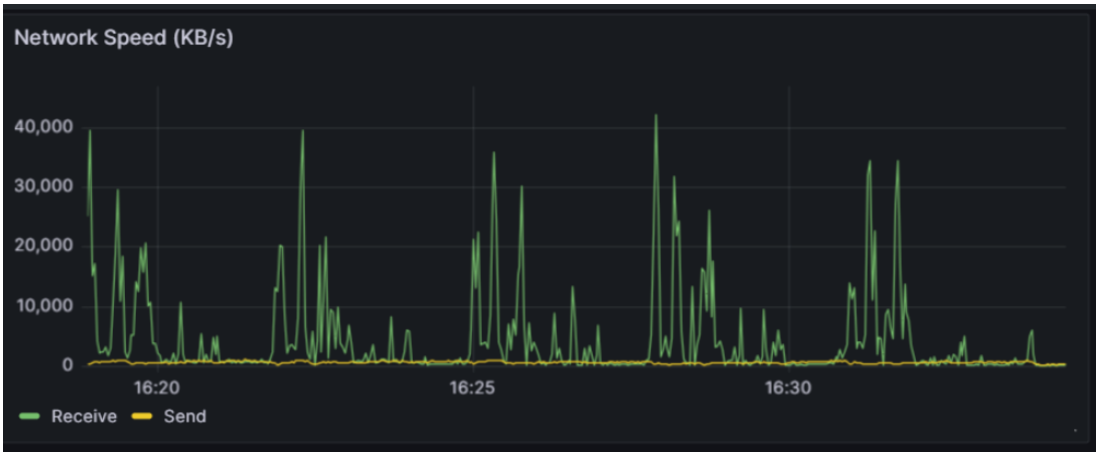


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

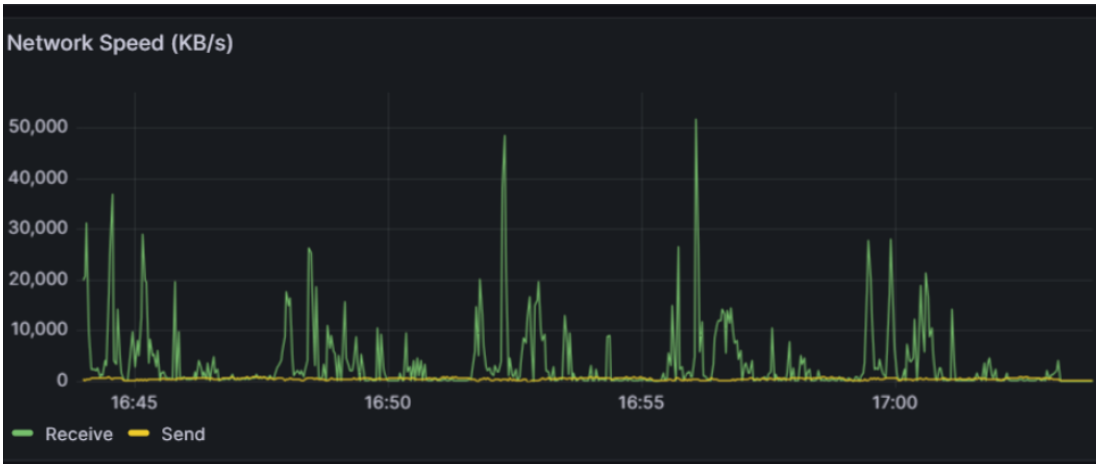


MD Core #2

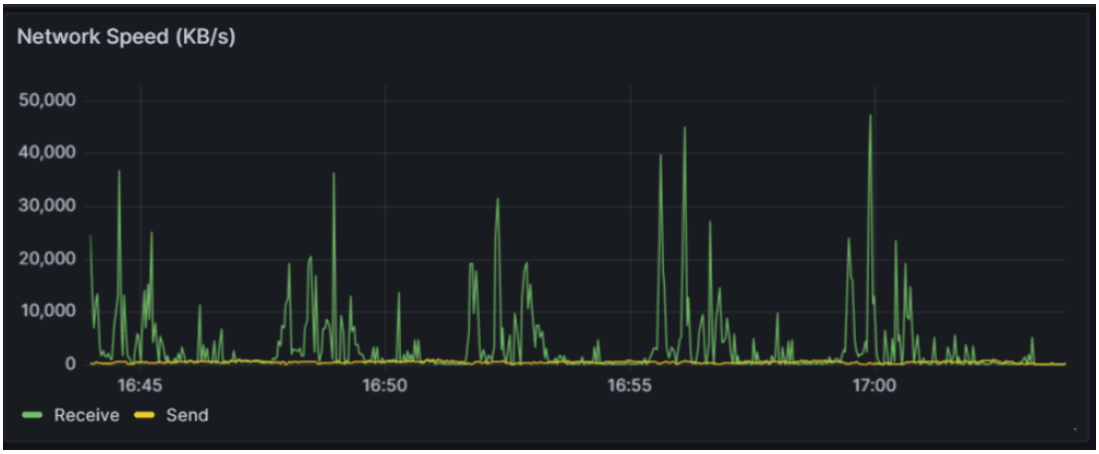


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1

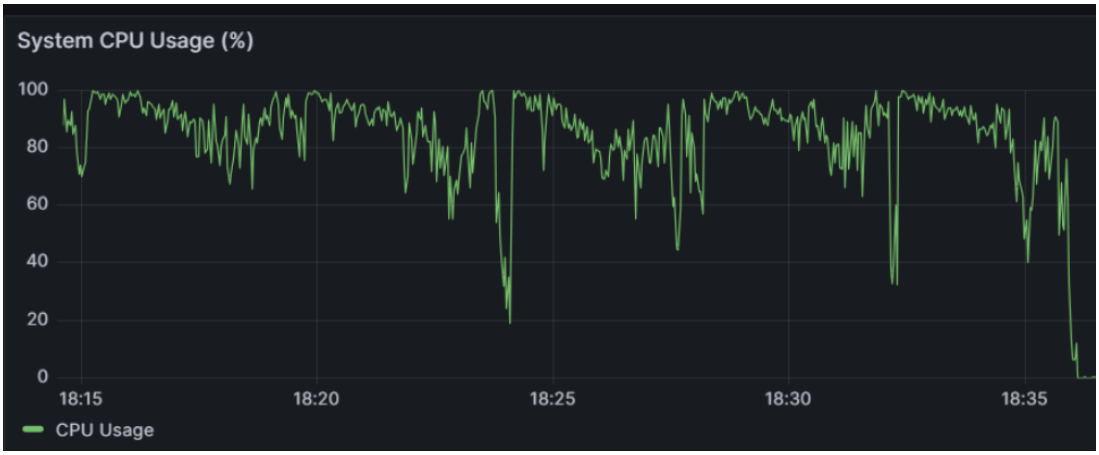


MD Core #2



CPU usage

MD Core #1

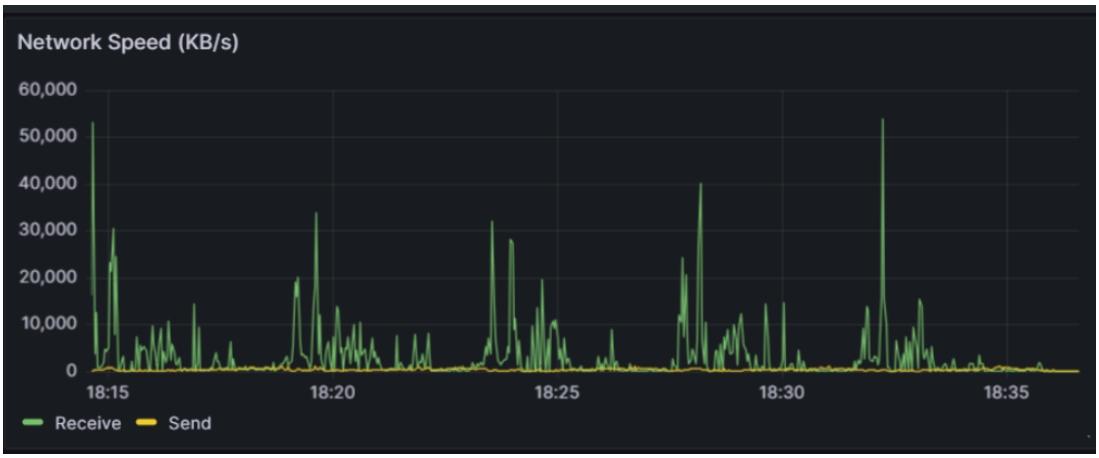


MD Core #2

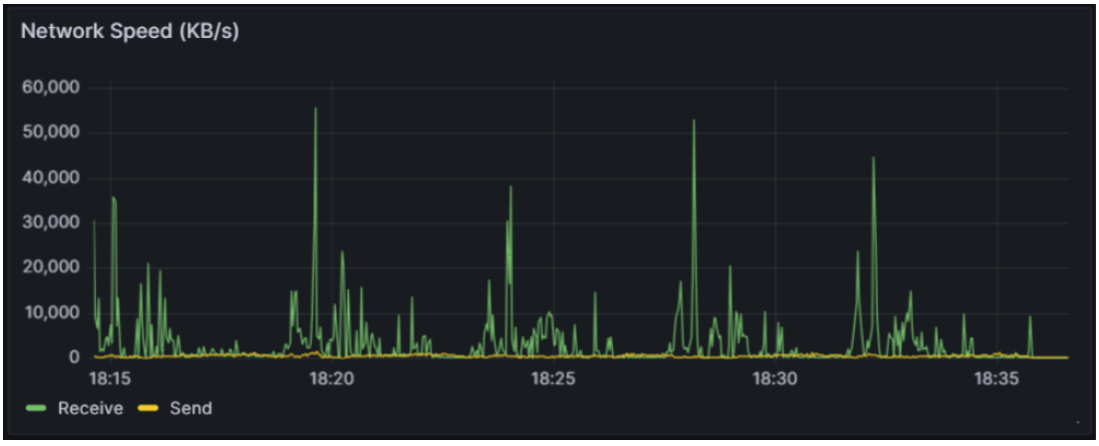


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

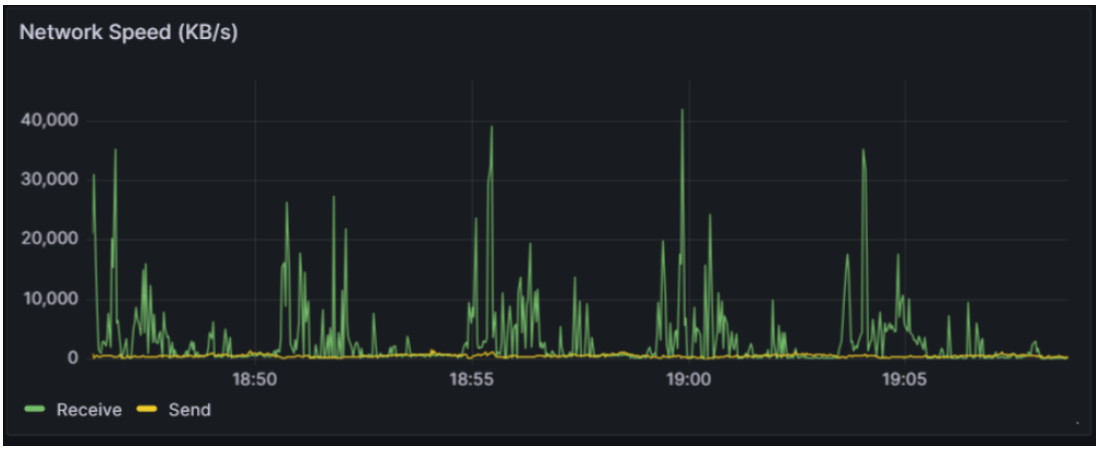


MD Core #2

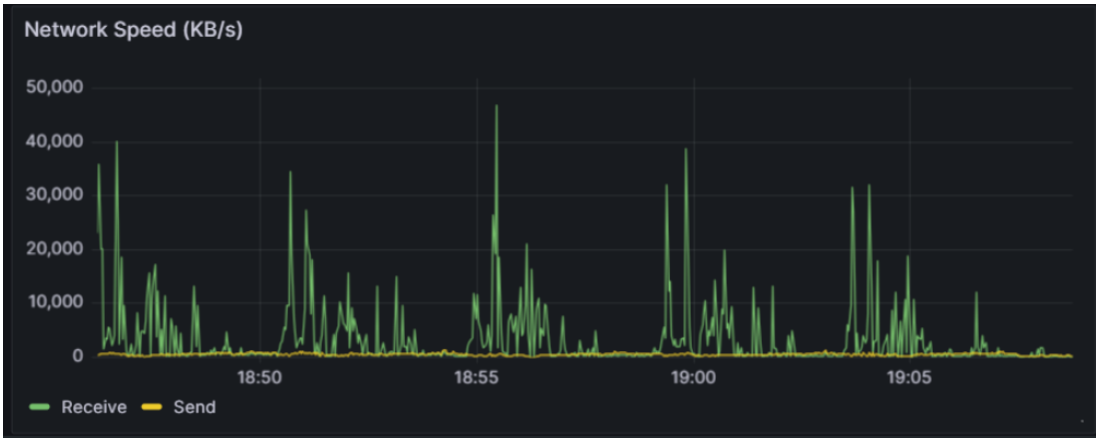


Network usage

MD Core #1



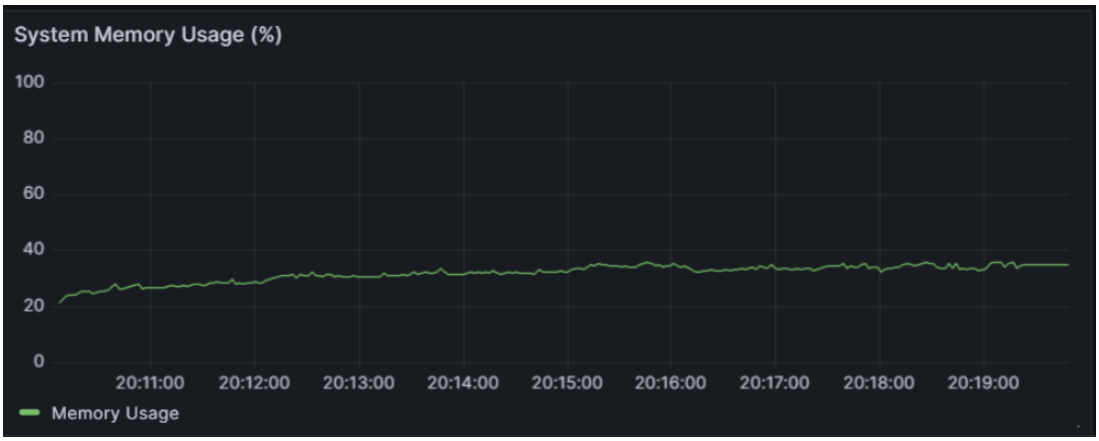
MD Core #2



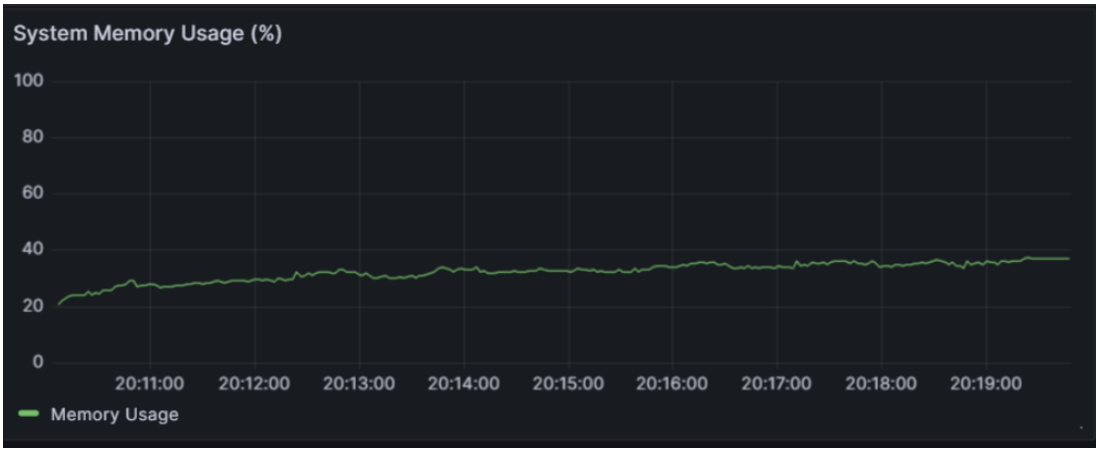
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1

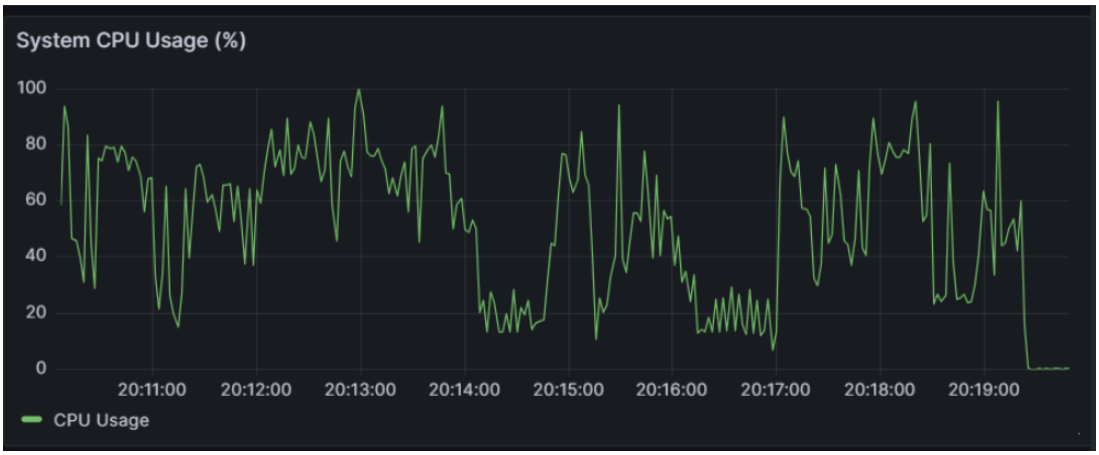


MD Core #2



CPU usage

MD Core #1

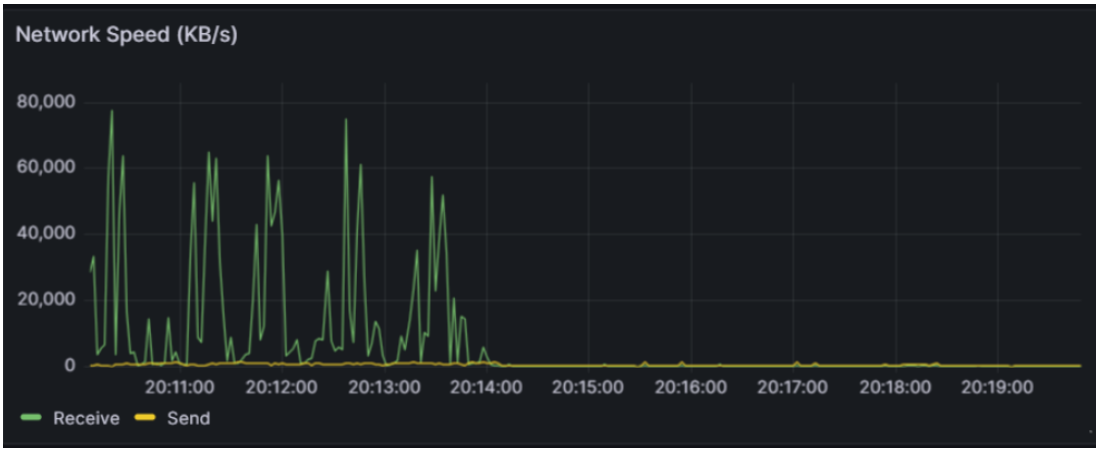


MD Core #2

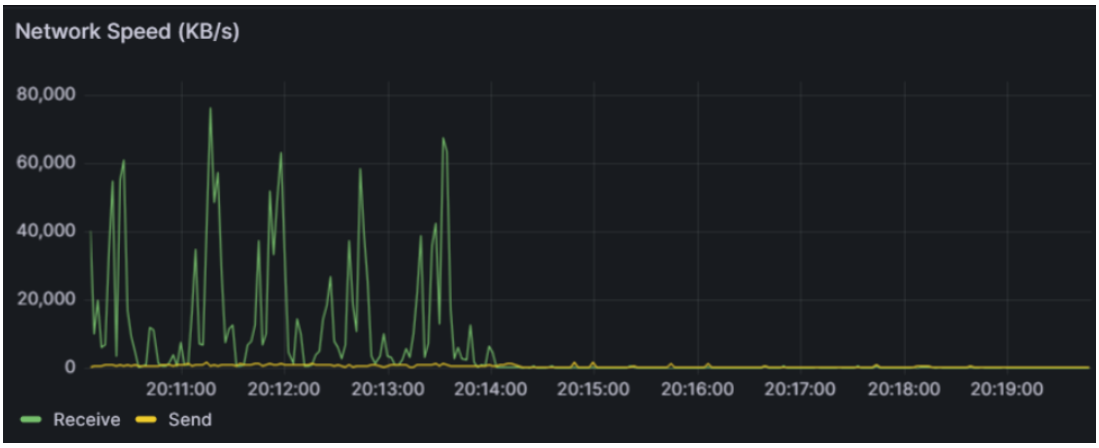


Network usage

MD Core #1



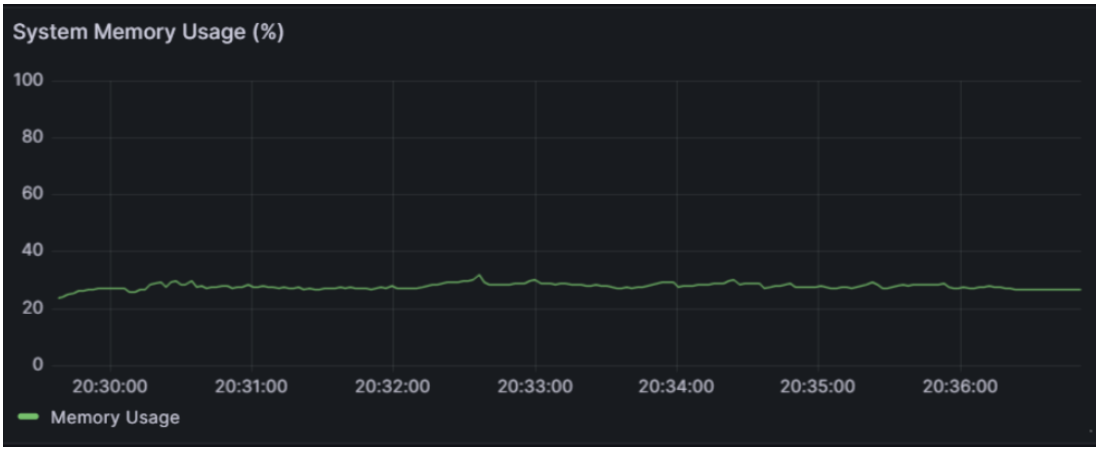
MD Core #2



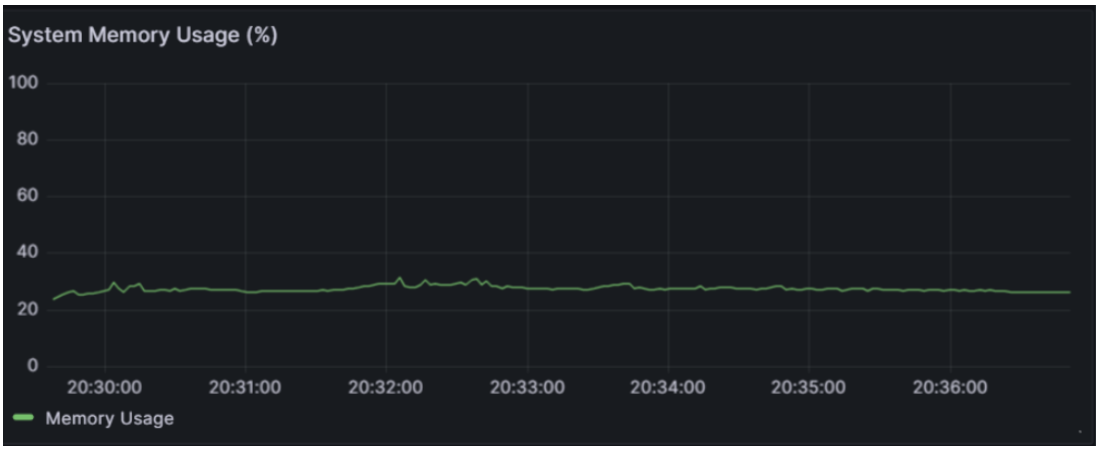
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1

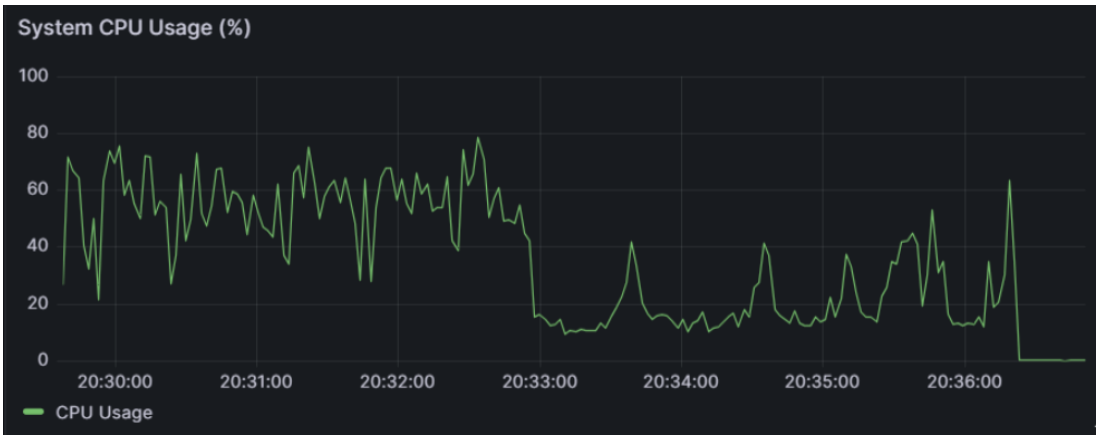


MD Core #2

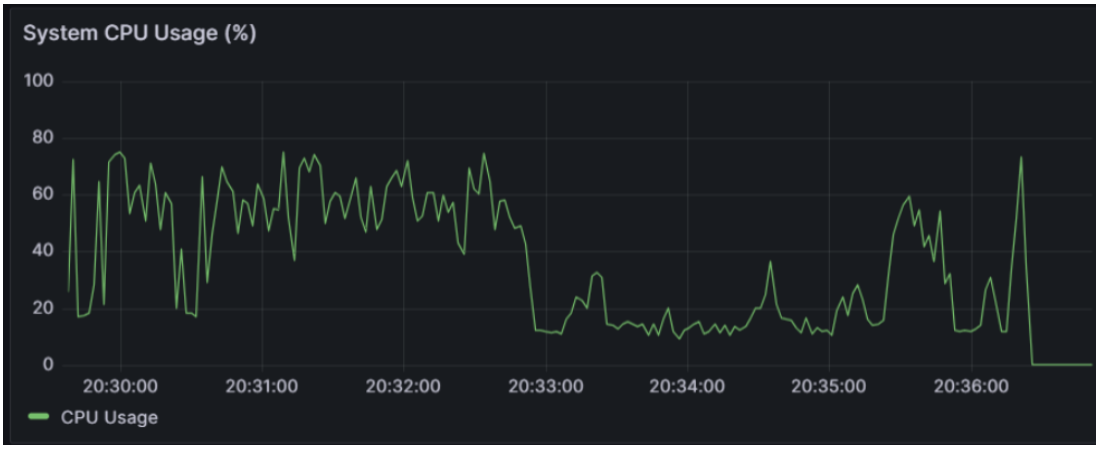


CPU usage

MD Core #1

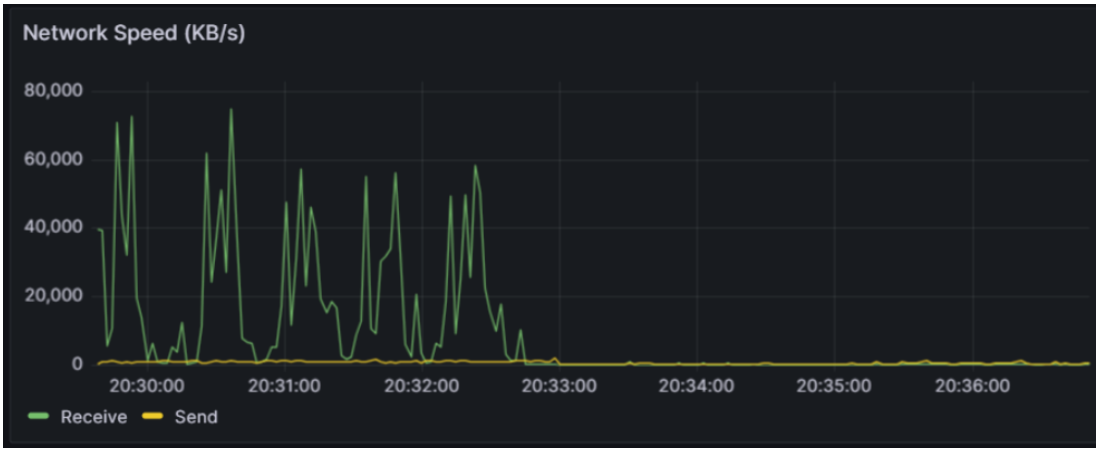


MD Core #2

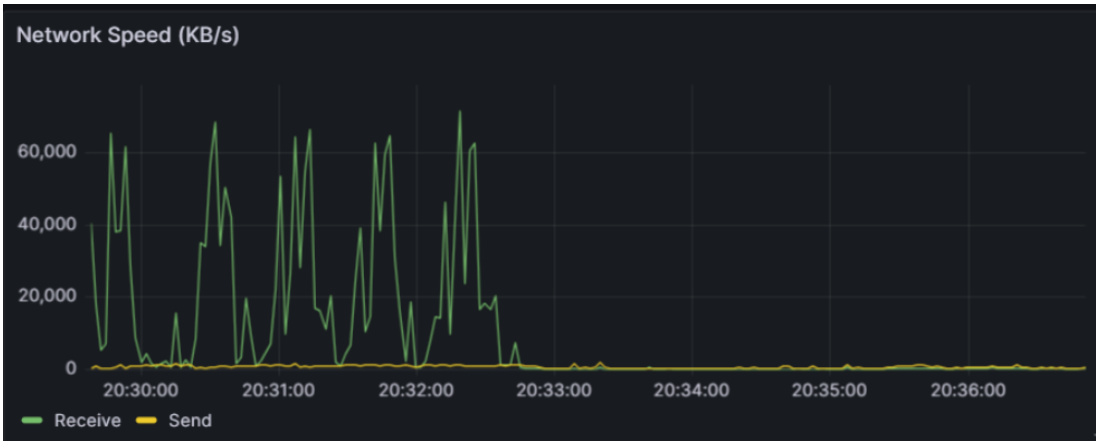


Network usage

MD Core #1



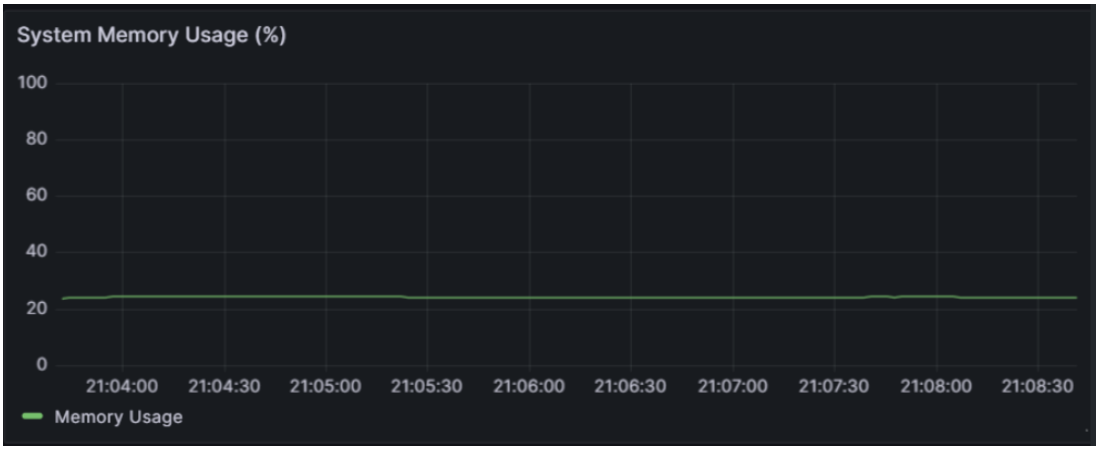
MD Core #2



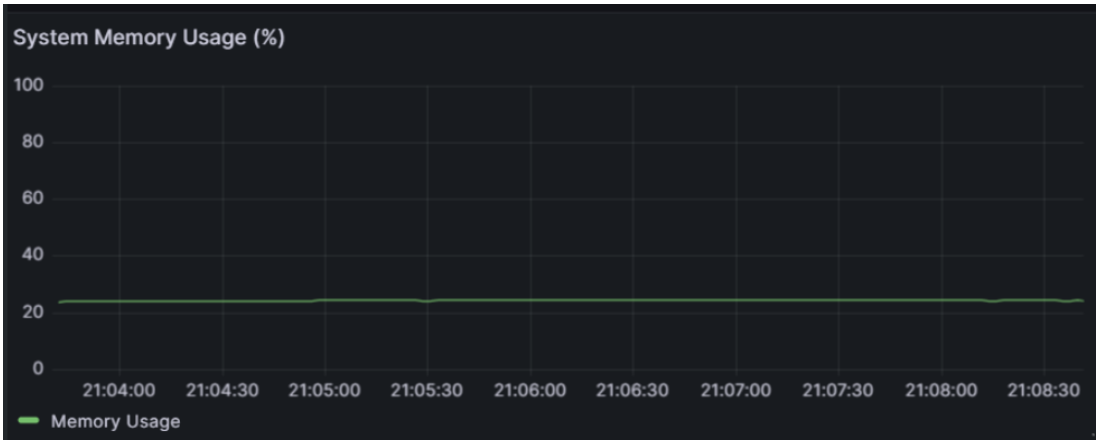
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

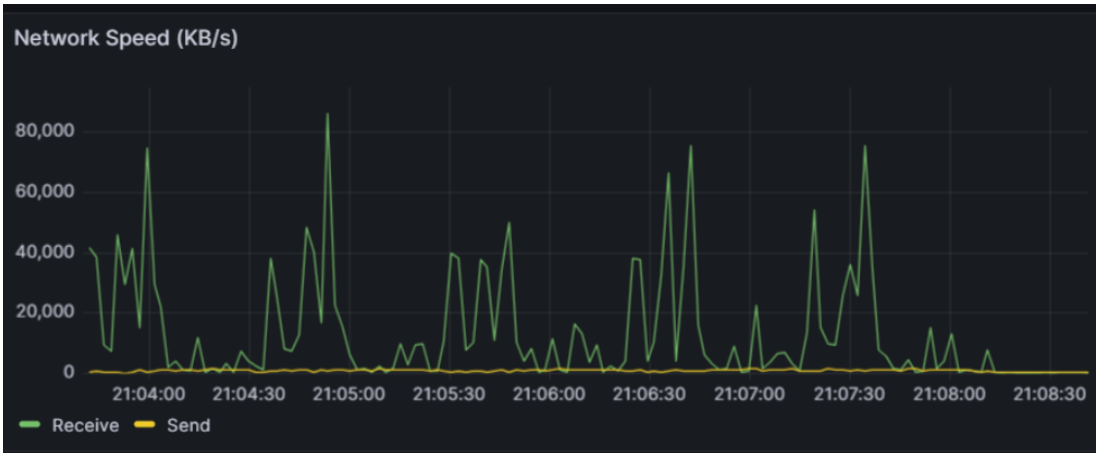


MD Core #2

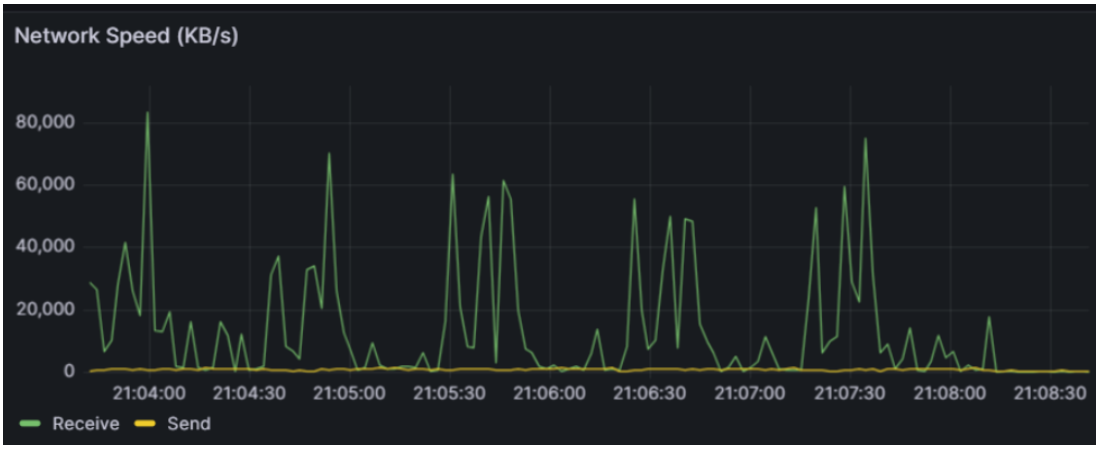


Network usage

MD Core #1



MD Core #2

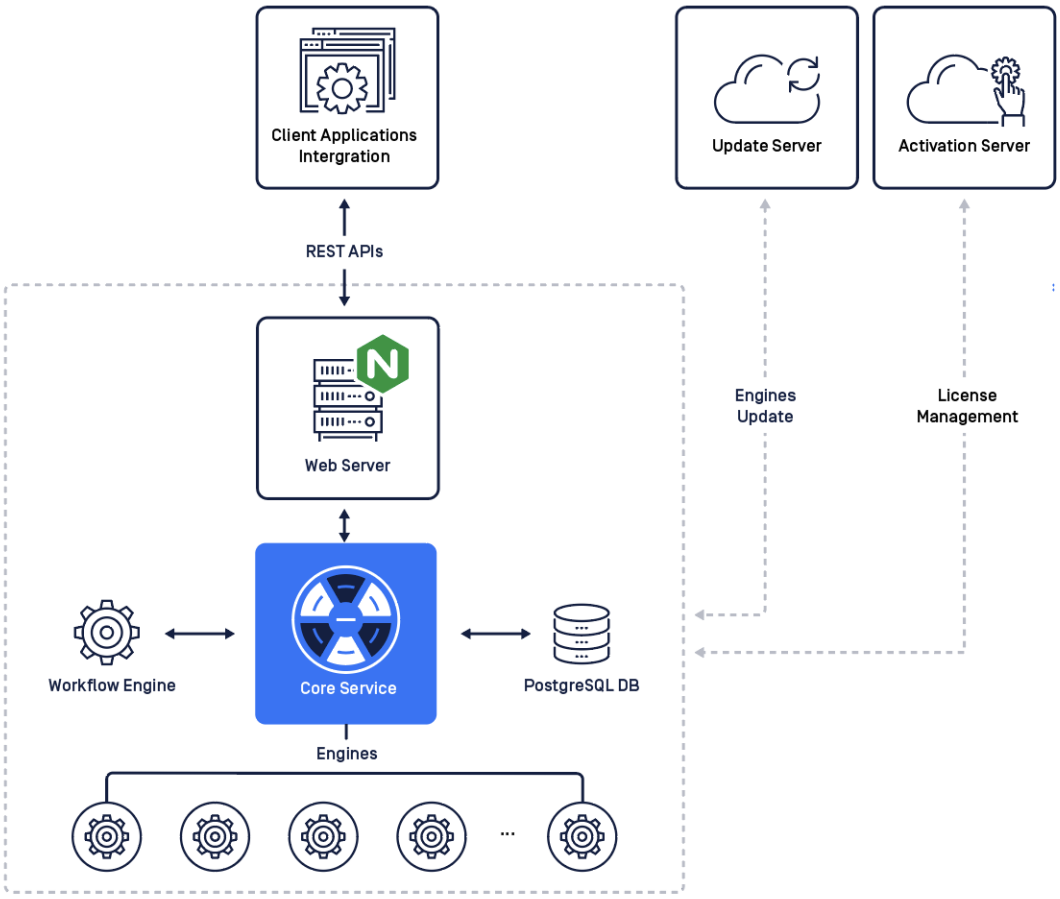

Network Speed (KB/s)
80,000
60,000
40,000
20,000
0
21:04:00
21:04:30
21:05:00
21:05:30
21:06:00
21:06:30
21:07:00
21:07:30
21:08:00
21:08:30
Receive
Send

MetaDefender Core Documentation

The users can consult this web page or, alternatively, they can download the manual in pdf format from the link below:

MetaDefender Core manual (SHA256:).

Recommended System Configuration



MetaDefender Core supports a wide range of deployment options, including air-gapped environments, providing the flexibility to choose the environment that best suits customers' requirements and infrastructure.

- Physical server.
- Virtualization platforms like VMware, Hyper-V and XenServer.
- Cloud Deployment.
- Container Deployment.

> **Note**
> Before installing MetaDefender Core, please refer to the **recommended minimum system configuration** listed below. The requirements are based on a standardized customer deployment and may require more or less resources to achieve the level of performance desired in your environment. Please note that the server specifications are built to allow a high volume daily processing. Please refer to Performance and Load Estimation for more details.

The recommendations below are for MetaDefender Core, API usage only. For any other use cases, please consult the user guide of the licensed products for accurate recommendations.

For certain use cases these might be adjusted and customized on specific needs and SLAs. We highly recommend to engage our ProServ team to assist in fine tuning MetaDefender and get the maximum performance out of your systems.

Microsoft Windows Deployments

[Windows] Supported Operating Systems

Only support 64-bit platforms.

- Windows 10, Windows 11 23H2, 11 IoT (22H2) only for End-User systems (e.g. Kiosk deployments)
  - Based on Microsoft License Terms, you're not allowed to use Windows Desktop as a server, therefore, MetaDefender Core can't be used on these systems for server use cases.
- Windows Server 2016, 2019, 2022

> **Disclaimer**
> End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

> **Plan for legacy OS support**
> OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:
>
> - Ubuntu 20.04 after **April 2025**.
> - Amazon Linux 2 after **June 2025**.
> - Windows 10 after **October 2025**.
>
> If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

[Windows] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 8	8	8 GB	16 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 12	16	16 GB	24 GB + 50 GB + 100 GB
MetaDefender Core 16	16	16 GB	32 GB + 50 GB + 100 GB
MetaDefender Core 20	32	16 GB	40 GB + 50 GB + 100 GB
MetaDefender Core MAX	32	32 GB	120 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core is installed.
- For a better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.
- SBOM processing requires more disk space, please allow at least 15GB free disk for the engine to work properly.

[Windows] Third Party Dependencies

- .NET framework 4.5 or above
- Microsoft Visual C++ Redistributable for Visual Studio 2015-2022 (version 14.38 or higher); x64 version.
- PostgreSQL: Only applicable when using a pre-installed PostgreSQL server running remotely.
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Some engines also have dependencies as described below:

Engine Name	Dependency
Deep CDR	Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 5.8.0 or above)
ESET	MetaDefender Core v4 temporary directory should have more than 200MB free disk space
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
Proactive DLP	• Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 2.0 or above) • .NET 6 dependencies • CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
Scrutiny	Microsoft Visual C++ 2015 - 2019 Redistributable x86 and x64
Systweak	.NET runtime 6.0, 7.0 or 8.0
Bkav Pro	Microsoft Visual C++ 2015 - 2022 Redistributable x64

[Windows] Installation Details

MetaDefender Core on Windows uses "C:\Program Files\OPSWAT" folder for storing resources or the installation directory.

MetaDefender will use its resources folder to store temp files as part of the analysis. **It's recommended to exclude this folder from real-time protection monitoring**. Knowledge base:

- How to set up exclusions on anti-malware software to prevent disruption
- Mitigating scan failure triggered by real-time protection service
- Can local AVs interrupt ongoing scans?

Linux Deployments

[Linux] Supported Operating Systems

Only support 64-bit platforms.

- Red Hat Enterprise Linux 8, 9
- Rocky Linux 9
- Debian 11, 12
- Ubuntu 20.04, 22.04
- Amazon Linux 2 is supported via Amazon Machine Image (AMI)

Plan for legacy OS support

OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:

- Ubuntu 20.04 after **April 2025**.
- Amazon Linux 2 after **June 2025**.
- Windows 10 after **October 2025**.

If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

For Hardended Linux OS

Some AV engines could be failed to initialize and run on MetaDefender Core (e.g. ESET, Kaspersky) due to execution permission required by them on `/var/tmp/ometascan` folder.

Please follow steps at Why have the ESET and Kaspersky scan engines failed to initialize on the hardened Linux OS? for remediation details.

Disclaimer

End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

[Linux] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 5	4	4 GB	10 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 10	8	8 GB	20 GB + 50 GB + 100 GB
MetaDefender Core MAX	16	16 GB	40 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core resides on.
- For better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.

[Linux] Third Party Dependencies

Dependencies list:

- openssl
- grep
- lib32stdc++6 (>= 4.5)
- libc6-i386 (>= 2.10)
- procps
- zlib1g
- libcurl3 (>= 7.19.7)
- libcurl4
- postgresql:
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Not all the above dependencies will need to be installed, it is dependent on different Unix distro and version

Some engines also have dependencies as described below:

Engine name	Dependency
Archive engine	• To extract password protected MS Office files, you need to install .NET 6 dependencies, libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • Rocky Linux 9.x/Red Hat 9.x: libnsl
Deep CDR	• Rocky Linux 9.x: libgomp, ncurses-compat-libs, libnsl • CentOS: libgomp, ncurses-compat-libs • Red Hat 8.x: libgomp, ncurses-compat-libs, libnsl • Red Hat 9.x: libgomp, ncurses-compat-libs, libnsl, compat-openssl11 • Ubuntu/Debian: libgomp1, libncurses5 • libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • .NET 6 dependencies
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
FileType engine	Rocky Linux 9.x/Red Hat 9.x: libnsl
Proactive DLP	libgdiplus, mscorefonts .NET 6 dependencies CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
RocketCyber	libgomp

Note: Some dependencies may need to be installed from external repositories, not from the OS default repositories.

[Linux] Installation Details

MetaDefender Core default installation path is using /var folder for storing resources:

- `/var/lib/ometascan` : installation folder with all its resources (database, DLP processed / CDR sanitized / quarantined file storage, engine package and definition updates).
- `/var/log/ometascan` : application logs.
- `/var/run/ometascan` : runtime folder for Core related processes.
- `/var/lib/ometascan` : all successful deployed engines data files.
- `/usr/lib/ometascan` : all shared libraries required by MetaDefender Core, bundled PostgreSQL server related binaries and libraries.
- `/var/tmp/ometascan/resources` : all temporary processing files for MetaDefender Core to process.

Storage Usage Information

Based on the configuration, MetaDefender Core will need additional disk space to store analysis/result data, quarantined files and temporary files:

	Data Storage
Analysis Reports (Processing History)	Approximately 4.5GB is required for every 1M analysis reports stored. • 100 GB is recommended as a baseline. Database storage requirements are affected by Data Retention settings, engine settings and scanning load.
Quarantined Files	Depends on the customers' dataset and configuration
Sanitized Files	Depends on the customers' dataset and configuration
Temporary processing files	Depends on the customers' dataset and configuration • A 1GB non-archive file requires 1GB free disk space at least. • A 1GB archive file requires (1 * 2)GB free disk space at least. • 50GB is a baseline and should be adjusted based on observation of the solution within the environment, with consideration to peak times and overall file makeup (e.g. complexity, size, archives).
Note	Options are available to control how large and/or complex files are handled to limit the amount of system resources required. See Workflow Configuration Template for more information.

Custom Engines

The recommendations above are specific for MetaDefender pre-packaged bundles.

However for additional Custom Engines, please review the Knowledge base to review additional requirements (if any) for the selected engine.

Browser Requirements for the MetaDefender Core Management Console

One of the following browsers is suggested to view the MetaDefender Core Management Console:

- Microsoft Edge
- Chrome
- Firefox
- Safari

Chrome, Firefox, Safari and Microsoft Edge browsers are tested with the latest available version at the time of release.

Network Bandwidth

In online environment where MetaDefender Core is configured to download engine update packages directly from OPSWAT update server source, then using gigabit network speed (125 MB/sec) is highly recommended to avoid potential network related issue while downloading files, or at least 100 Mbps (12.5 MB/sec) network speed.

Allowlisting URLs

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts':

- https://activation.dl.opswat.com - for license online activation
- https://update.dl.opswat.com - for engines and databases download

Note: *IP address-based allowlisting on your firewall might fail after some time since OPSWAT uses CDN (AWS Cloudfront) to faster delivery updates over the world, and IP address of edge servers might change over time.*

Depending on your location, connections with OPSWAT cloud-based servers be forwarded to specific AWS Cloudfront's edge locations, each is assigned with its own unique IP address ranges. You might also need to allow correct AWS Cloudfront's corresponding IP address ranges by referring to https://ip-ranges.amazonaws.com/ip-ranges.json

Warning

Even the OPSWAT update servers host updates for all of the available engines we support, sometimes custom engines might try to connect to their own cloud for updates, but this can be disabled in firewall and they will be updated just from OPSWAT cloud servers instead.

Differences Between MetaDefender Core v4 and v5

We are very excited to inform you about the release of version 5.0.0 of our MetaDefender Core!

The new version is designed to enhance the product performance, user experience, as well as to provide more flexibility, security, and scalability for our customers.

At the same time, we are also committed to keeping backward compatibility for whom running MetaDefender Core v4, and avoid the need for updating your client integration to move forward with MetaDefender Core v5.

v4 REST API Integration Supported
All public REST API endpoints available on MetaDefender Core v4 will be continously supported on v5.

v4 Product Major Features Supported
All product major features available on MetaDefender Core 4.21.2 (the latest one on v4) will be continously supported on v5.

Seamless Product Upgrade
Upgrading from MetaDefender Core v4 to v5 will be supported and covered with data migration without additional manual steps from users.

Licensing information will be also migrated.

Below are some highlights of the first v5 release, MetaDefender Core 5.0.0:

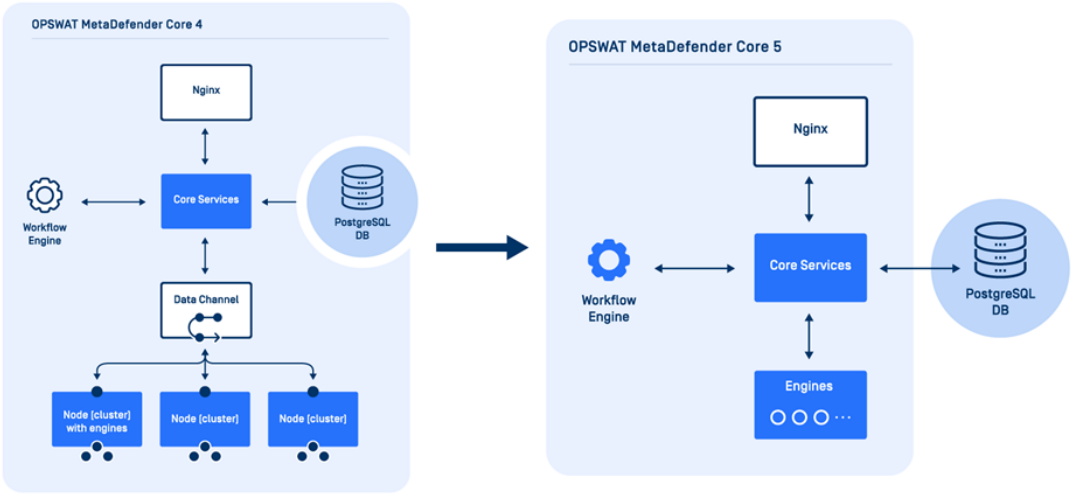
New product architecture

This version introduces a simpler product architecture that speeds up integration and automation, file processing flow, and immensely enhance the product performance and efficiency.

Earlier versions utilized MetaDefender Node concept, designed to support scaling of the scanning infrastructure by distributing scan requests among several different processing Nodes. Product licensing was based on number of Node instances. The benefit of having such a distributed infrastructure is that based on Node loads, Metadefender Core server can always choose the most appropriate Node to assign a new scan task to. However, having nodes in the v4 architecture caused more complexity and possible latency in the processing flow and entailed maintenance cost.

In this release, we retired multi-Node deployment and restructured MetaDefender Core's backend. This alteration significantly improves file processing speed and product efficiency, as well as eliminates extra cost for maintaining node services. Users now can simply install additional MetaDefender Core instances for scalability requirement,

as well as configure their own load balancer to coordinate tasks among multiple MetaDefender Core instances. Accordingly, product licensing is now based on number of the product instances.



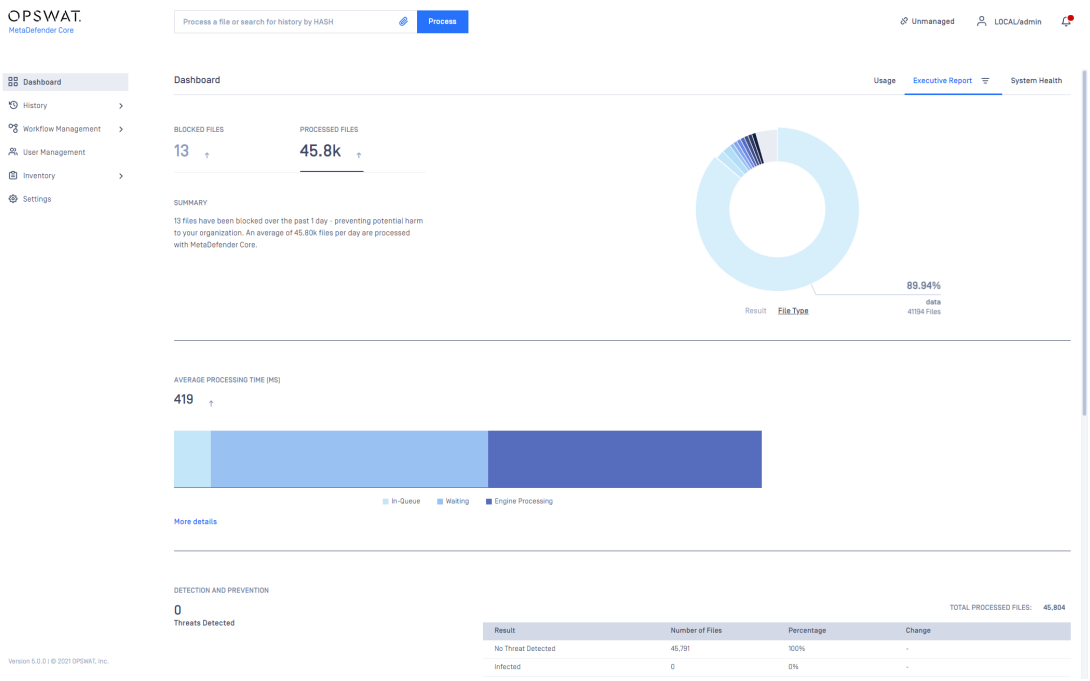
Please be informed that

- All existing administrative APIs related to "Node" on v4 will be kept as it is on v5 to ensure backward compatibility with existing client integrations.
- Applicable settings defined in Windows registry (Windows) or ometascan-node.conf config file (Linux) for Node will be migrated to Core. Details: MetaDefender Configuration

Brand-new product user interface

The second notable update in this version is a brand-new UI for the management console, which was designed with the focus on the quality and thoughtfulness of the user experience. The new administrative console UI provides an advanced set of Core features and observable processing status.

Comprehensive executive dashboard reporting offers dynamic, real-time and actionable threat insights. It also provides visibility of your security posture allowing you to paint a picture of security that aligns with the overall business. The report includes easy-to-read charts, graphs, and statistic explanation that help your security professionals reduce clutter, gather information quicker, spot threat activity early and act swiftly.



We're continuing to improve our product UI to provide users an excellent experience with the application and overall usability. Please expect more powerful UI features coming in next releases.

New compression archive engine for archive sanitization

In previous version, MetaDefender Core's archive engine implemented four functions including decryption, extraction, encryption and compression that with a high volume of files it takes time to process them all. For the purpose of enhancing archive handling performance, this release comes with a separated compression archive engine, which dedicatedly processes archive sanitization (compression and encryption). It is also created to facilitate maintenance and further feature development.

Archive sanitization feature now requires Deep CDR license

As to respect sanitization practice and ensure unknown threats prevention as designed, now starting on version 5, it's mandatory to have Deep CDR licensed for archive sanitization feature enablement on MetaDefender Core workflow configurations.

Archive handing related engines changed

The legacy "Archive" engine on MetaDefender Core v4 will be changed its name to "Archive Extraction" engine, even when you are still on MetaDefender Core v4.

A brand new "compression archive engine" is to dedicatedly handle archive sanitization in v5 (archive compression), that helps relevant archive handling performance improvement, easier for maintainance and further feature development.

Notes when you are still on MetaDefender Core v4 (not upgraded yet)

The legacy "Archive" engine (now name changed to "Archive Extraction" engine) on MetaDefender Core v4.x will still cover both extraction and compression archive tasks as usual.

Even though your MetaDefender Core v4.x might now have a new "Archive Compression" engine auto downloaded, but that engine cannot be deployed (marked as "permanently failed" in relevant engine status check API). This behavior is expected, and not causing any impact to MetaDefender Core processing flow, or its performance though.

- Just in case your client integration has a check for engine status before triggering scan request, then you might want to disable "Archive Compression" engine status on MetaDefender Core v4 using REST API. Detailed instruction: Archive Compression Engine Permanently Failed on MetaDefender Core v4
- When you later upgrade to MetaDefender Core v5 from v4, then you might need to manually re-enable "Archive Compression" engine to make it active and running on MetaDefender Core v5.

Notes for licensing

For offline deployment (MetaDefender Core is not connected to the Internet directly for online update), you will need to re-activate again on OPSWAT portal to retrieve latest engine list in .yml file including "Archive Compression" engine if you are currently licensed with Deep CDR.

Meanwhile, for online deployment, this manual procedure is not required since MetaDefender Core will cover that itself.

Import & export configurations

Not only workflow settings like in previous versions, but also all other general settings (update, security, data retention, server, hashes to skip) and most importantly, user settings, will be able to export and import on MetaDefender Core version 5.

Learn more at Import/Export configuration

Backward compatibility supported

The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

Windows application event logging disabled by default

Applicable only to Windows OS based MetaDefender Core.

Help avoid duplicated logging with separate file option, and improve overall processing performance. Of course, when being still valid to you, you are still supported to enable it back.

CSRF token header renamed

CSRF token originally created for security strengthening and being used by MetaDefender Core management console UI since version 4.21.0 via header `oms_csrf_token`

To get it compliant with some load balancers which do not accept underscore in header name by default, and save you from additional load balancer configuration step, we have now changed the header name to `oms-csrf-token`

This change does not impact to any REST API based client integration.

More comprehensive and accurate error messages on REST APIs

Impacting on following REST API endpoints:

- `PUT /admin/config/session`
- `PUT /admin/config/update`
- `PUT /admin/config/scan`
- `PUT /admin/config/skip`
- `PUT /admin/config/auditlog`
- `PUT /admin/config/webhook`
- `PUT /admin/config/rule`

Upload time and queue time logic updated in JSON scan response

Impacting on following REST API endpoints:

- `GET /file/{data_id}`
- `GET /hash/{md5/sha1/sha256}`
- `GET /archive/{data_id}`
- `POST /file/{async}`

Added new field:

- `receive_data_timestamp`: The timestamp when upload progress started (first byte received) (in milliseconds)
- `upload_time`: Total time elapsed for upload process (in milliseconds)

Modified existing field (new logic):

> **Backward compatibility supported**
> The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

- `upload_timestamp`: The timestamp when upload progress finished (all bytes received) (in milliseconds)
- `queue_time`: Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue time = start_time - upload_ timestamp) (in milliseconds)

Release version 5.0.0 also comes with variety of outstanding features and improvements.

For detailed information about our latest release please view our Release notes

Installation

Starting with MetaDefender Core 4.20.0, the centralized/shared database deployment model is supported (all MetaDefender Core instances share the same database), along with traditional standalone database mode (every MetaDefender Core instance has its own database to manage).

- Standalone mode
- Shared Database Mode

Installation

1. Download the package of your choice from My OPSWAT
2. Install the package on your computer via the Command Line or via the Wizard
3. Open a web browser and point to http://<server name or IP>:<port>
    - The default port is **8008**
    - In case of problem check Inaccessible Management Console page
4. Complete the required steps of the basic configuration wizard
5. You must activate this deployment to use its features

Installation notes

- If the MetaDefender Core package **dependencies are not installed** on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.
- During installation the databases might need to be upgraded. This could take noticeable time depending on database size (eg.: length of scan history).

Standalone mode

Wizard Installation

The Install Wizard is only for the Windows installer (.msi file).

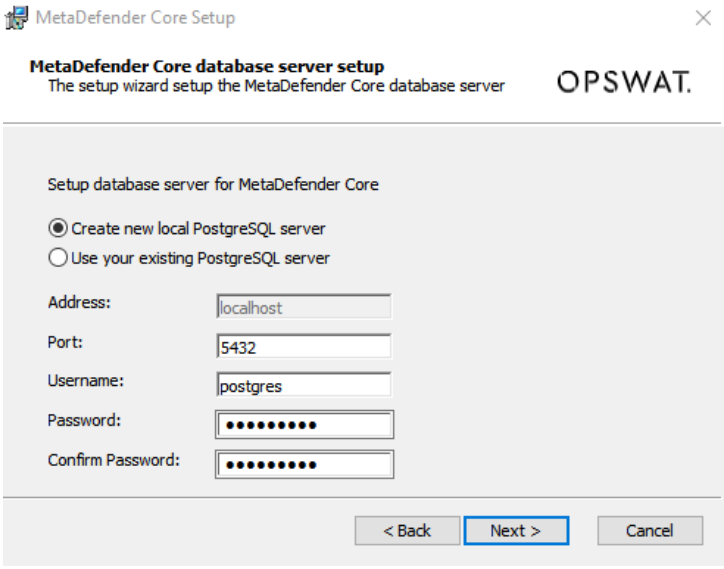
Since MetaDefender Core 4.19.0, we have switched to using PostgreSQL DBMS which needs a dedicated database server. MetaDefender Core supports two modes to setup PostgreSQL server:

Local (Bundle) PostgreSQL server

MetaDefender Core will install a new PostgreSQL server locally in the same box with the product.

> **PostgreSQL user credentials**
> Unicode characters are NOT supported for "Username" and "Password"
>
> "Password" does not accept # (hash) as starting character (e.g. #ABC)

> **The ignition file**
> If the ignition file exist in the designated location, then MetaDefender Core will use the details defined in the ignition file, rather than what is passed via the installation wizard.
>
> More info about the ignition file based installation: Command Line Installation

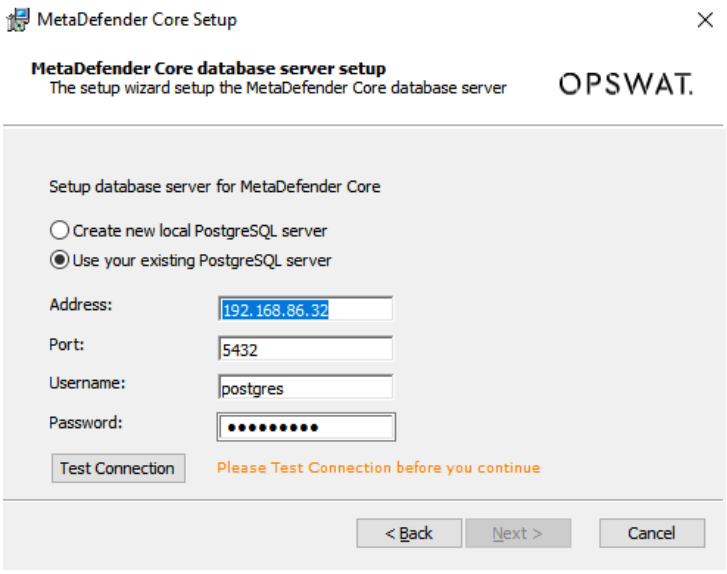


Remote (Existing) PostgreSQL server

MetaDefender Core will connect to remote pre-installed PostgreSQL running remotely.

> **PostgreSQL connection configuration**
> "Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

> **PostgreSQL user privilege requirements**
> MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.
>
> Details: PostgreSQL User Privilege Requirements



Command Line Installation

Ignition File

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Make sure to create the ignition file, if it doesn't exists before you go ahead and install MetaDefender Core via command line.

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

MetaDefender Core supports two modes to setup PostgreSQL server:

1) Local PostgreSQL server

- MetaDefender Core will install a new PostgreSQL server locally, next to the product.
  - A sample iginition file for PostgreSQL server information (***Only non-Unicode characters supported for "user" and "password"__***):

ometascan.conf bash

```
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
```

2) Remote PostgreSQL server

> **Info**
>
> **PostgreSQL version 12 (at least 12.3) or version 14 is required.**
>
> According to PostgreSQL notice, version 12 will stop receiving fixes on November 14, 2024, so we suggest that you use version 14.

MetaDefender Core will connect to a pre-installed PostgreSQL server running remotely and will setup its database.

Make sure to follow steps at Open Connection On PostgreSQL Server first, so that your remote existing PostgreSQL server to accept upcoming connection from MetaDefender Core.

A sample ignition file for PostgreSQL server information:

ometascan.conf bash

```
[dbserver]
type=remote
host=192.168.86.32
port=5432
user=existingserver_admin_user
password=existing_server_pass
```

After installing MetaDefender Core successfully, you may want to remove all credentials info created for [dbserver] section in ignition file for security reason.

> **Warning**
> If the MetaDefender Core package dependencies are not installed on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.

Debian package (.deb)

bash

```
sudo dpkg -i <filename> || sudo apt-get install -f
```

On Red Hat Enterprise 7.x Linux / CentOS package (.rpm)

bash

```
sudo yum install <filename>
```

For systems which enabled GPD check flag:

bash

```
sudo yum install --nogpgcheck <filename>
```

Red Hat Enterprise Linux 8.x package (.rpm)

Install the latest PostgreSQL repository RPM:

> **Note**
> Due to the shorter support cycle on Red Hat Enterprise Linux, all supported versions of PostgreSQL are not available on this platform. The PostgreSQL Yum Repository will integrate with the normal systems and patch management, and provide automatic updates for all supported versions of PostgreSQL throughout the supported lifetime of PostgreSQL. Reference: https://www.postgresql.org/download/linux/redhat/

bash

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

In case installing a remote PostgreSQL on either the same MetaDefender Core server or another server, you need to install the following additional package. PostgreSQL version 12.3 or newer is recommended.

bash

```
sudo dnf install -y postgresql12-contrib
```

Install MetaDefender Core:

bash

```
sudo dnf install -y <filename>
```

Windows package (.msi)

On Windows systems it is possible to install the product by running the corresponding .msi file.

From command line interface it is also possible to install the product by executing

bash

```
msiexec /qn /i <msi file name> <option key>=<option value>
```

where the possible keys and their default values are the following:

Key	Default Value	Description
INSTALLFOLDER	\Program Files\OPSWAT\MetaDefender Core	Customize installation folder for product Example: `INSTALLFOLDER="D:\Products"`
RESTADDRESS	*	REST interface binding IPv4 or IPv6 address ('*' means that service listens on all IPv4 and IPv6 interfaces)
RESTPORT	8008	REST interface binding port
REPORT_ENGINE_ISSUE	true	Enable reporting of engine issue count. (possible values: "true" or "false").

Note

Reporting of engine issue count

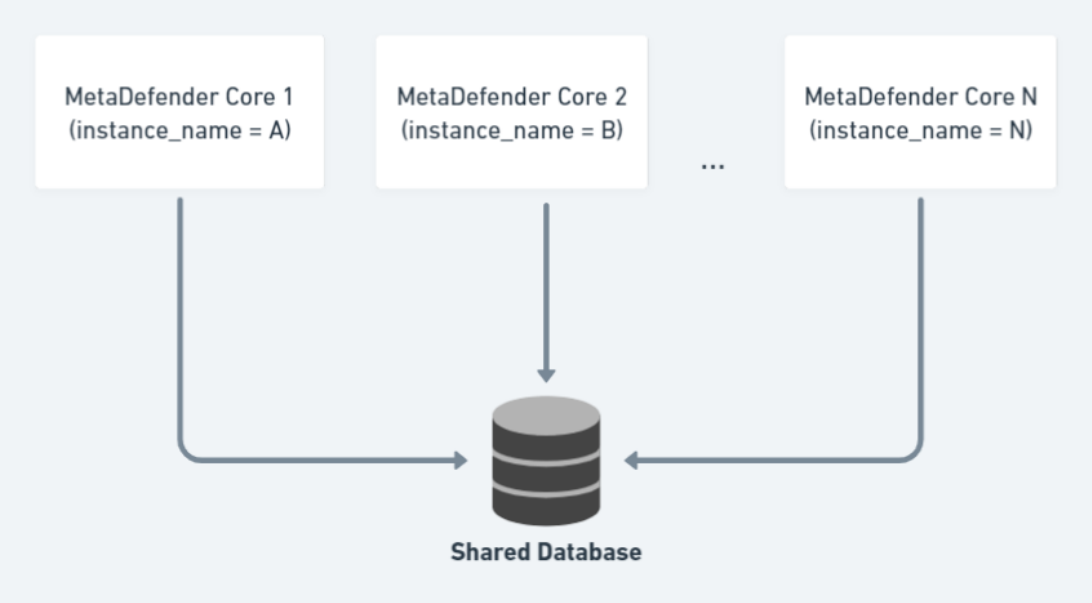
If reporting of engine issue count is enabled, MetaDefender Core server will send only the **number** of initialization errors and **number** of unexpected stops for the specific db/engine version. This information is sent over a HTTPS channel when the product downloads the latest package descriptors. This information is used for early detection of any specific 3rd party engine quality issues.

For details on using msiexec please consult Windows installer documentation.

Shared Database Mode

Starting with MetaDefender 4.20.0, you can use a centralized database deployment model, where all MetaDefender Core instances can share the same external database.

That helps auto-sync all product configurations (**except Single-Sign-On** that requires unique settings for each Core), processing history, live/on-going scan result query, audit, statistics data.



This article demonstrates how to install and setup MetaDefender Core 4.20.0 or newer with shared database mode.

Important notes & current limitations:

Currently not supported

- Does not support Central Management use-case with shared DB mode.
- Does not support to switch from shared database mode back to standalone database mode.
- Does not support to change instance_name of MetaDefender Core in shared database deployment model after installation.
- Does not support upgrading older MetaDefender Core from non-persistent mode to using the shared database mode.
- Does not support upgrading older MetaDefender Core from standalone mode to using the shared database mode.
- Does not support to upgrade multiple MetaDefender Core instances at once in shared database mode
    - Stop all Core instances' scanning service before performing the upgrade.
    - Upgrade each Core server one by one instead.

- For each Core, when the upgrade finished, then you can start using that Core for scanning without waiting for the other Core instances' upgrade to be done.

> **Disclaimer**
> MetaDefender Core does not support mode switching.

Requirements for shared database setup:

- The central remote PostgreSQL server (for shared database) could be utilized with a built-in (native) PostgreSQL database server installed with one of the MetaDefender Core instances in the network. Or using a dedicated existing PostgreSQL database server (see detailed requirements for PostgreSQL version at Recommended System Configuration).
- Before setting up, make sure to follow steps at Open Connection On PostgreSQL Server so that all MetaDefender Core instances could connect to the central remote PostgreSQL server.
- MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core, details: PostgreSQL User Privilege Requirements
- All MetaDefender Core instances must be installed on the same Operating System (Windows / Linux).
- The first MetaDefender Core instance connects to the shared database must be ready first (license activated, users/groups/roles created) before setting up other MetaDefender Core instances to connect and use that shared database.

Engine update:

- If engine updates is configured to be via Folder source, all MetaDefender Core instances must be able to access to the configured update folder, and the option DELETE FILES AFTER IMPORT will be unchecked and prevented for modification.

Authentication:

- Single-Sign-On (SSO) authentication settings (if using) cannot be shared among MetaDefender Core instances because SSO requires an unique login URL for every different Core.
- If an user is locked for login on a certain MetaDefender Core instance(e.g. exceeded the number of failed login times), that user is not prevented for logging in on any other instances.
- If an user is locked for sign-in on a certain MetaDefender Core instance (e.g. exceeded the number of failed login times), if the admin releases lockout on that user on that MetaDefender Core instance, that user is still not able to login on other MetaDefender Core instances.

Sanitized / DLP / Quarantined location changed after MetaDefender Core install:

Windows:

Shared storage path will be stored in 3 strings under key global in registry: quarantinepath, dlppath, sanitizepath as below

(Default)	REG_SZ	(value not set)
dbmode	REG_SZ	4
dlppath	REG_SZ	\\10.40.50.100\test
instance_name	REG_SZ	481rvj
quarantinepath	REG_SZ	\\10.40.50.100\test
restaddress	REG_SZ	*
restport	REG_SZ	8008
sanitizepath	REG_SZ	\\10.40.50.100\test

You can set either 1, 2 or 3 different paths for the 3 settings above (means that quarantined, DLP and sanitized files will be stored in same or different storages corresponding). In case shared storage is not set in installation, values of 3 settings above will be empty.

Restart MetaDefender Core service after changed.

Linux:

Open file `/etc/ometascan/ometascan.conf` and set shared storage path for 3 keys quarantinepath, dlppath, sanitizepath under section [global]

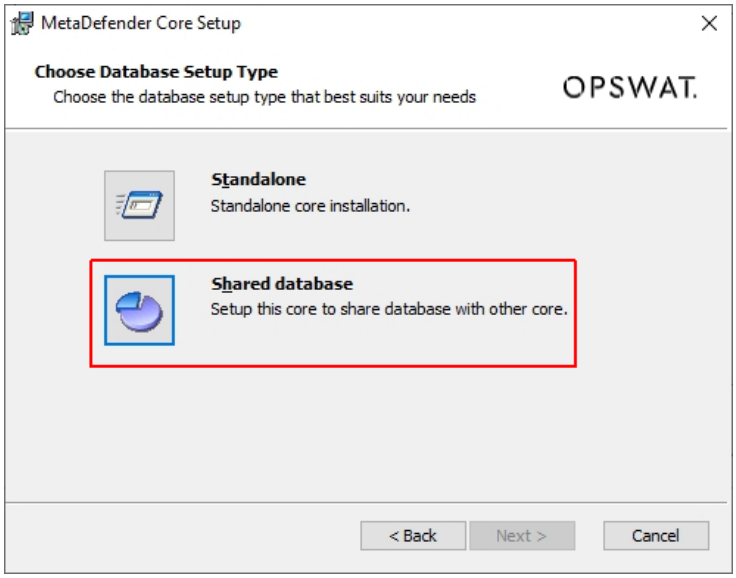
Restart ometascan service after changed.

Wizard Installation (Windows Only)

> **Info**
> Only applicable to MetaDefender Core 4.20.0 or newer using wizard on Windows OS.

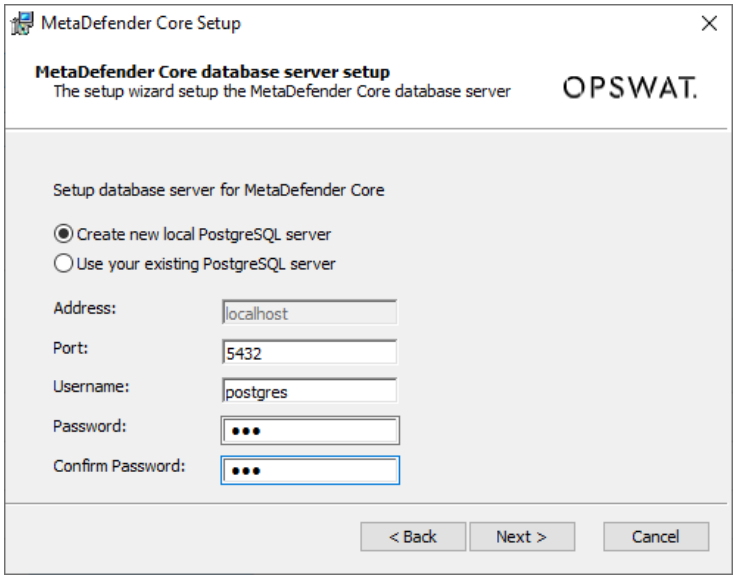
Clean install scenario

1. **Choose Shared database option:**

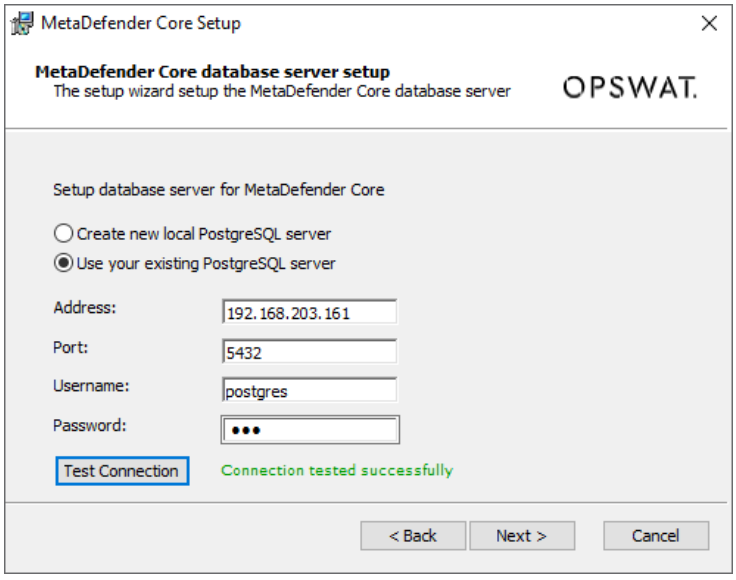


2. **Setup database server for MetaDefender Core:**

You can choose to create a native / built-in PostgreSQL server and setup the shared database on (then other MetaDefender Core instances will want to connect to that PostgreSQL server afterwards)



Or choose to connect to a pre-setup existing PostgreSQL server:



PostgreSQL connection configuration

"Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

PostgreSQL user privilege requirements

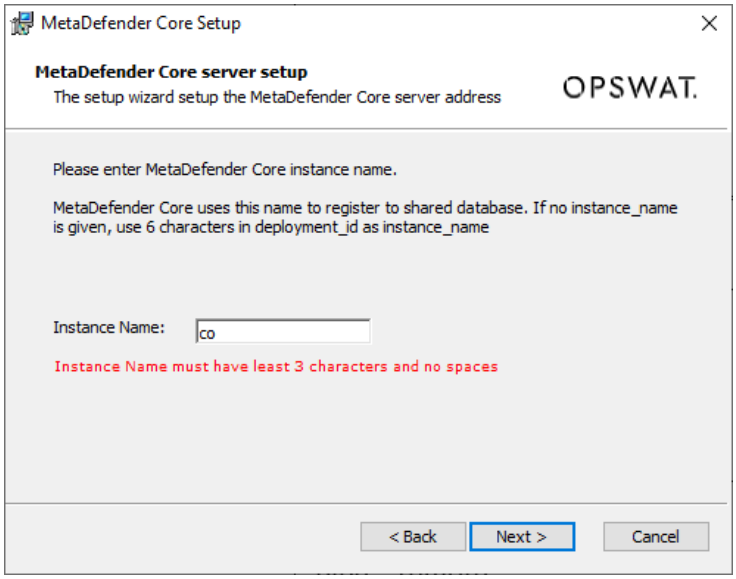
MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.

Details: PostgreSQL User Privilege Requirements

3. **Create an unique instance name for that MetaDefender Core (different from other MetaDefender Core instances):**

Instance name must contain at least 3 characters and no white space.

If you choose to leave it blank, which means it will use the default instance name with 6 characters extracted from that MetaDefender Core deployment ID.

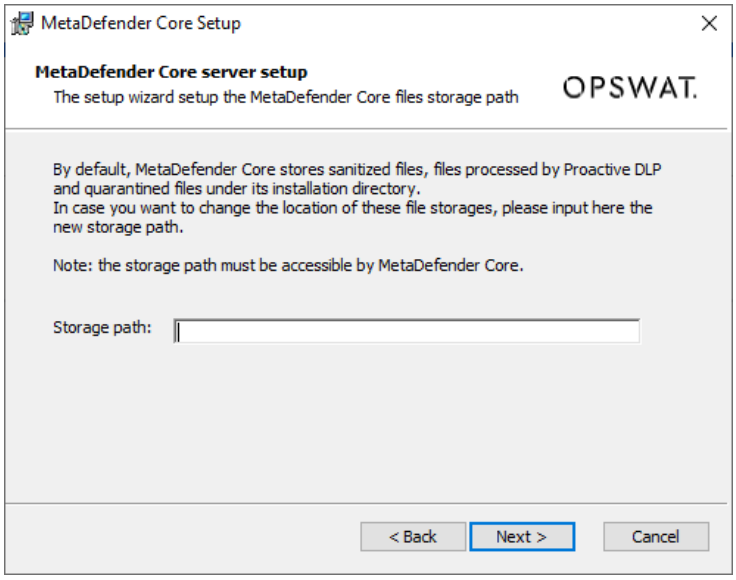


4. **Input the file storage path (sanitized, DLP, quarantined files)**

In the shared database model, all MetaDefender Core instances could use the same folder path to store all their own sanitized, DLP, quarantined files after processed.

The default location is: ***<MetaDefender Core installation folder>\data\ {sanitized|dlp|quarantine}***

If you wish to use that default location, just leave the input box blank and press Next.



Otherwise, you can specify a different location of sanitized/dlp/quarantined files. It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core.

For example:

- C:\new_folder
- \192.168.1.1\new_folder

Command Line Installation

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Clean install scenario

1. Create ignition file if not existed

- Windows: **C:\OPSWAT\ometascan.conf**
- Linux: ***/etc/opswat/ometascan.conf***

An example of ignition file:

ometascan.conf bash

```
eula=true

[global]
dbmode=4
instance_name=MetaDefenderCoreA
storage_path=D:\path\

[user]
name=admin
password=admin
email=admin@local

[dbserver]
type=remote
host=192.168.1.1
port=5432
user=postgres
password=your_own_password
```

Field	Required	Mapping to wizard UI	Possible values
global/dbmode	optional default is 1 (standalone)		1 - Standalone 2 - Central Hub deployment 3 - Non-persistent **4 - Shared**
global/instance_name	optional default is 6 characters of deployment ID		Unicode and number At least 3 characters No whitespace
global/storage_path	optional default is empty		Path to storages (DLP, quarantined, sanitized files) It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core. For example: • C:\folder • \192.168.1.1\folder

2. Run installers using command line to install

Please refer to Command Line Installation for installation command.

Deployment automation support

Note

An Ignition file could be utilized for either / both of following use-cases:

1. Pre-define PostgreSQL server information for MetaDefender Core to connect and setup database: only required on Linux installation, or Windows with command line installation. Learn more details at:
    a. Standalone Mode > Command Line Installation (applicable on version 4.19.0 or newer).
    b. Shared Database Mode > Command Line Installation (only applicable on version 4.20.0 or newer).
2. Automating to accept EULA, create a default local admin user and import MetaDefender Core configurations upon installed: Those steps could be automated using ignition file described in this page.

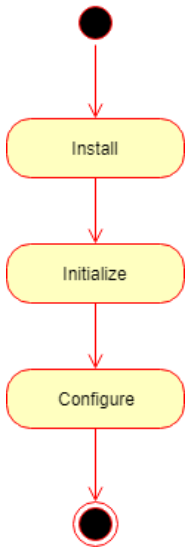
Ignition file location to create:

- **Windows**: `C:\OPSWAT\ometascan.conf`
- **Linux**: `/etc/opswat/ometascan.conf`

The product supports fully automated deployment. It means that it can be installed and configured with no human interaction.

The automated deployment can be split to three steps on a high level:

1. Installation
2. Initialization
3. Configuration



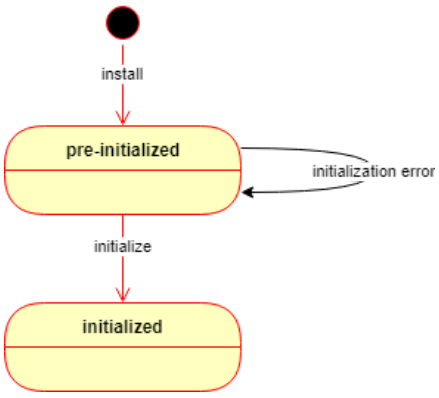
Installation

To automate the installation, install the product from the command line and provide the installation-time options as parameters to the installer. For further details see Command Line Installation

After the installation is complete, the product starts up and waits in a pre-initialized status. The product may be initialized in two ways:

1. Manually using the Wizard Setup or
2. Automatically using an *ignition file* (see below).

> **Warning**
> If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried after fixing the problem (e.g. placing the ignition file in its lookup location) and restarting the OPSWAT MetaDefender Core service.
>
> Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

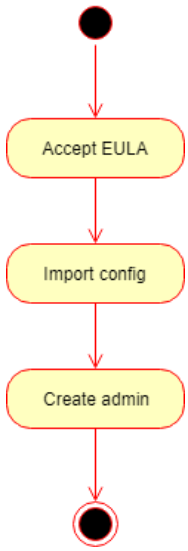


Initialization

Initialization is the process of bringing the product to an operable status.

Basically the initialization consists of the following steps:

1. Accept the End User License Agreement (EULA),
2. Import product configuration and
3. Create the first administrator user account.



Ignition file

The initialization process can be configured in a file called the *ignition file*.

> **Warning**
> In Linux, the ignition file and the folder /etc/opswat should have the permission 755

Ignition file location

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

The ignition file must be in conf format.

ometascan.conf bash

```
eula=true
[user]
name=admin
password=admin
email=admin@local
apikey=44c96cbb4e4e6c0a1d16bcb3c3da24def317
[config]
import=config_package.zip
import_password=123
[security]
session_cookie_samesite=Lax
[internal]
skip_startup_pages=true
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
private_username=my_internal_db_user
private_password=mypassword
[ocm]
ocm_url=https://my.opswat.com
reg_code=XXXXXXXX
```

Ignition file fields

The ignition file accepts the following fields:

Section	Key	Required	Accepted Values	Description
	eula	Optional (mandatory when having [user] section)	true false (default)	Whether to accept the End User License Agreement. This key must be set at the top of ignition file. This key must be set to "true" to accept the EULA. Any other value will cause the initialization to fail. When having [user] section defined, then this key must be set.
[user]		Optional		Initial administrator user account properties. The Administrator role is granted to the account.
	name	Optional	string	User name for the initial administrator user account.
	password	Optional	string	Password for the initial administrator user account. WARNING! Clear text password The password in this configuration file must be stored in its clear-text format and as so it may be visible for unauthorized parties.
	email	Optional	string	E-mail address for the initial administrator user account.
	apikey	Optional	string	API key for the initial administrator user account.
[config]		Optional		
	import	Optional	string	Path to a file in json/zip format that contains a previously exported configuration to be imported. Zip-format file contains a full configuration including user management settings, so it will overwrite [user] section.
	import_password	Optional	string	Password to decrypt .zip configuration package.
[security]		Optional		

Section	Key	Required	Accepted Values	Description
	session_cookie_samesite	Optional	Strict Lax (default) None	Declare if session cookie should be restricted to a first-party or same-site context.
[internal]		Optional		
	skip_startup_pages	Optional	true false (default)	Skip all welcome and other startup pages (e.g. upgrade result) when installing / upgrading MetaDefender, and going straight to the product index default page.
[dbserver]		Optional		
	type	Mandatory when having [dbserver] section	local remote	Where the PostgreSQL database server should physically locate. "remote" means PostgreSQL server and MetaDefender Core are not in the same machine.
	host	Mandatory when having [dbserver] section	string	IP address / domain name of the server where PostgreSQL server locates. "localhost" can be used when applicable.
	port	Mandatory when having [dbserver] section	number	Port of PostgreSQL server is listening for connections from clients (i.e. MetaDefender Core).
	user	Mandatory when having [dbserver] section	string	PostgreSQL server's user. SUPERUSER privilege is required for MetaDefender Core to setup its database and extensions for the first time. **Only non-Unicode characters supported.**

Section	Key	Required	Accepted Values	Description
	password	Mandatory when having [dbserver] section	string	PostgreSQL server's user credentials. **Only non-Unicode characters supported.**
	private_username	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
	private_password	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
[ocm]		Optional		
	ocm_url	Optional	string	URL to My OPSWAT server.
	reg_code	Optional	string	A registration code to enroll to My OPSWAT server.

Detailed initialization process

1) After the product has been started, it looks for the ignition file in the designated location.

Windows: C:\OPSWAT\ometascan.conf ** Linux: /etc/opswat/ometascan.conf

** 2) If an ignition file is found, then

2.1) It gets validated, and if it is valid, then based on the information found in the ignition file:

- The EULA is accepted,
- The configuration is imported,
- The administrator account is created.
- If any of the above steps fails, then the error is logged, and the initialization gets terminated.

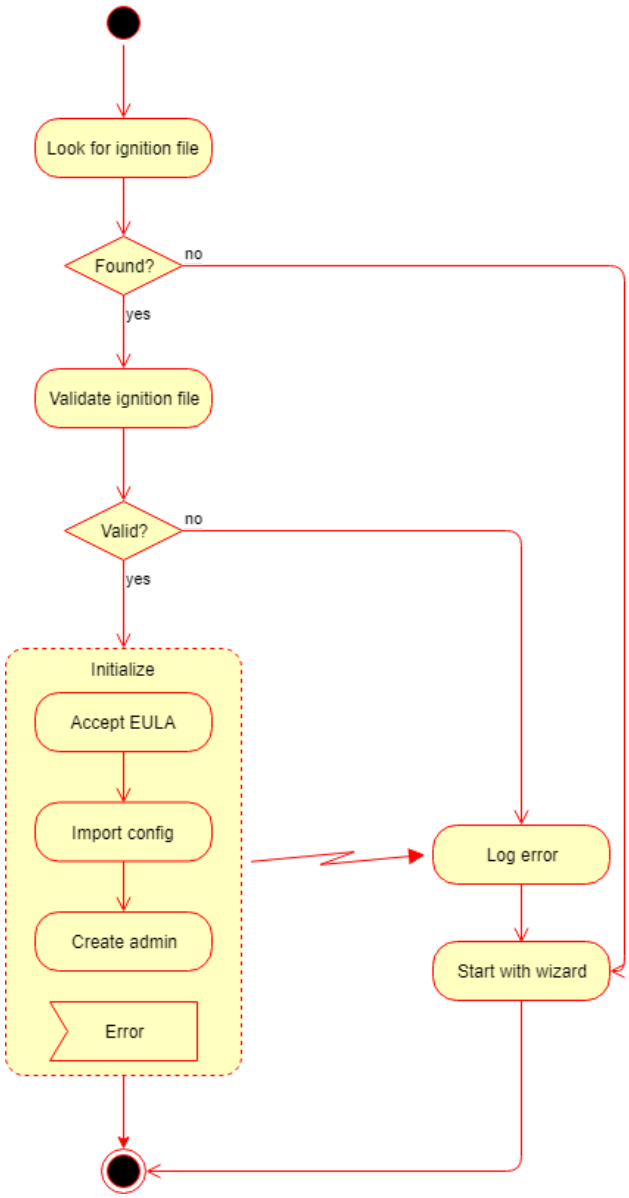
> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

2.2) If it is not valid, then the error is logged, and the initialization gets terminated.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

3) If there is no ignition file, then no initialization is performed.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.



Warning

If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried fixing the problem (e.g. placing the ignition file to its lookup location) and restarting the OPSWAT MetaDefender Core service.

Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

Configuration

After the initialization is complete, the product is ready with the default and the imported configuration.

This configuration can be later changed calling the configuration API functions. For further details about the API see Configuration related APIs API

CIS Level 1 Guidelines

Support Red Hat Enterprise Linux 9
For more details about Center for Internet Security (CIS) please refer to this document

Instruction steps

I. Install OpenSCAP

bash bash

```
yum install openscap-scanner scap-security-guide
```

II. Generate a result file and a HTML report using OpenSCAP scanner tool

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

III. Remediation of CIS Level 1 issues

Generate a remediation script based on the `ssg-rhel9-ds.xml` file:

bash

```
oscap xccdf generate fix --profile
xccdf_org.ssgproject.content_profile_cis_server_l1 --fix-type bash --output
remediations.sh /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

And execute remediation script.

bash

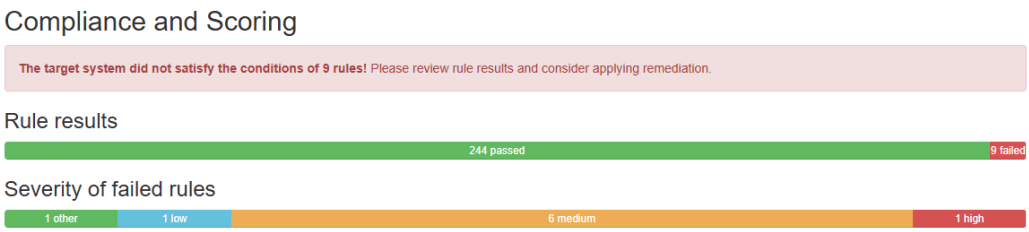
```
./remediations.sh
```

IV. Review the results after remediation

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

Example:



Notes

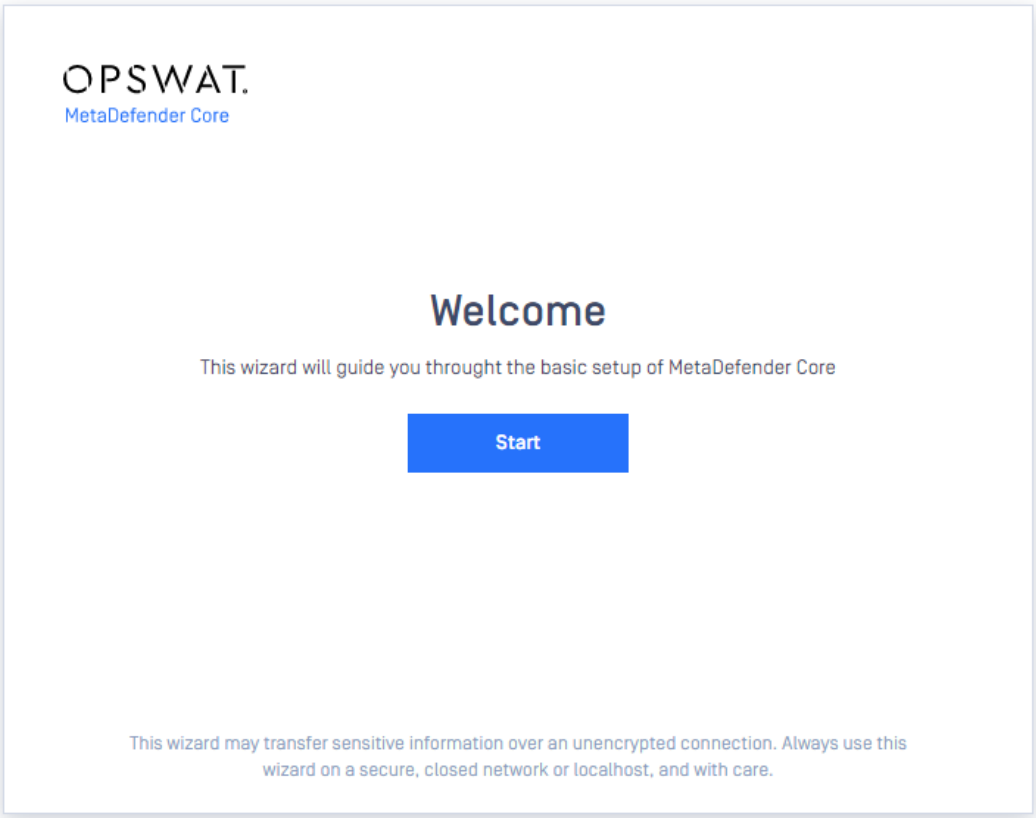
- CIS Level 1 requires `/tmp` folder to be mounted in a separate partition. Please ensure that that new partition have enough disk space for MetaDefender Core to run.
- CIS Level 1 requires that "Ensure No World-Writable Files Exist".
    - For now, when freshly installing MetaDefender Core, all its binary files meet the requirement.
    - When installing/updating engines, some engines might create additional files for its operation, and it might violate this requirement. In this case, you need to again execute the remediation script in the step III.

Wizard Setup

Introduction

Access the MetaDefender Core management console for the first time (i.e. http://localhost:8008), you are to complete a quick walkthrough for wizard setup in order to be able to use the product.

The administrative management console will be available only upon you have successfully finished this wizard stage.

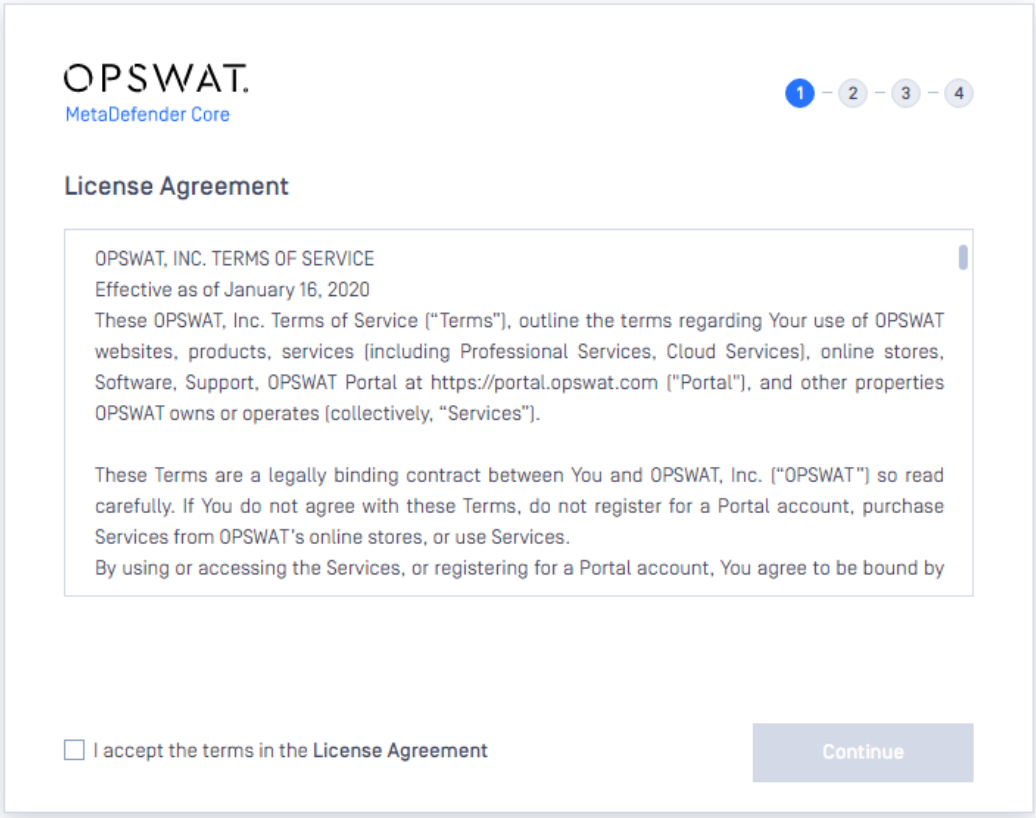


Warning

This wizard may transfer sensitive information over an unencrypted connection. Always use this wizard on a secure, closed network or localhost, and with care.

Basic configuration steps

End-User License Agreement



In the first page you can find the End-User License Agreement. You have to accept the terms before moving on.

Please read through the EULA (https://docs.opswat.com/policies/privacy-policy/portal-privacy-policy) carefully and if you agree with it, check `I accept the terms in the **License Agreement**` and click `Continue` to proceed.

Local Admin User Setup

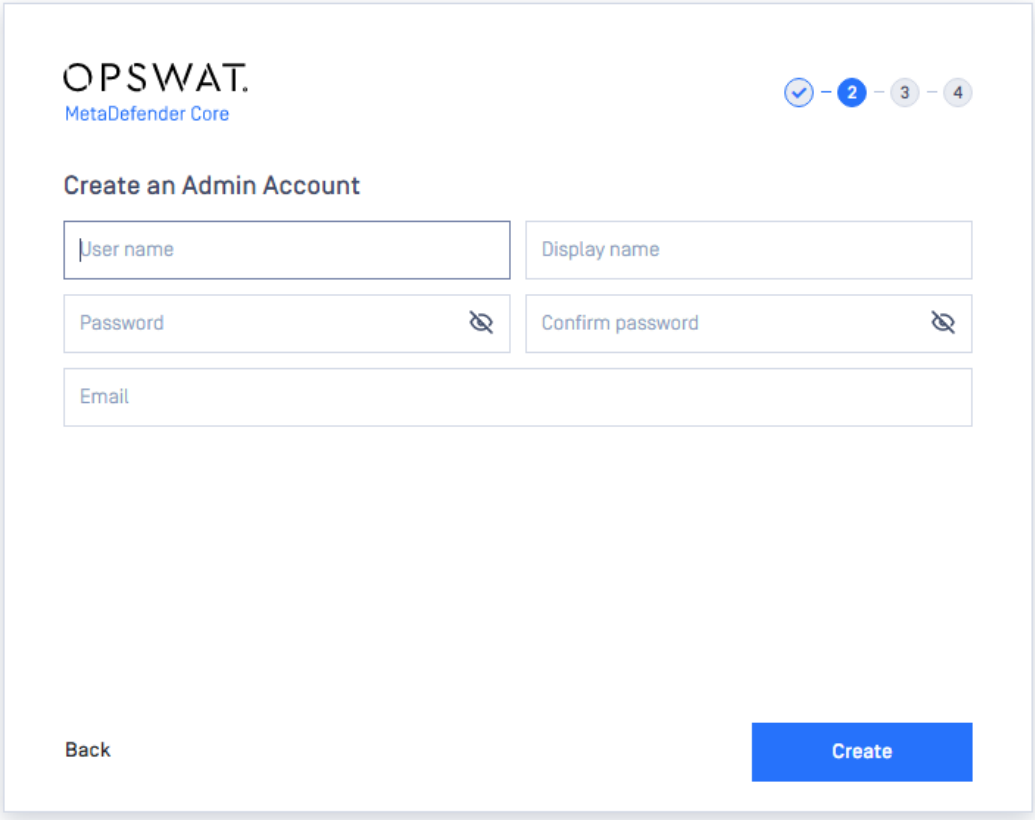
The next step is to set up a default local administrator account.

This account will be the first one being able to access the administrative management console, and to create accounts for other users. You have to fill out all fields in this page to be able to move forward. When you are done, click NEXT to continue.

> **Info - User Directory**
> The administrator account, that is created via the basic configuration wizard, is always added to the LOCAL user directory as a member.
>
> Learn more about user directory: User directories

The following information is required for the administrator account:

ACCOUNT NAME	The unique name of the account that is used at the time of login and in log messages for accountability.
ACCOUNT DISPLAY NAME	Name of the person bound to this account. This name (appended to the name of the account's user directory) is displayed in the top right corner of the administrative management console.
PASSWORD	Password of the user bound to this account that is used at the time of login. Passwords sent clear-text As long as TLS is not configured for the basic configuration wizard, passwords are sent clear-text over the network and may be disclosed to unauthorized parties. As a mitigation action: 1. Either use the wizard on *localhost* or on a direct network connection, or 2. Enabling HTTPS as soon as possible and change the password immediately if it has already been set.
EMAIL	Email address of the person bound to this account.



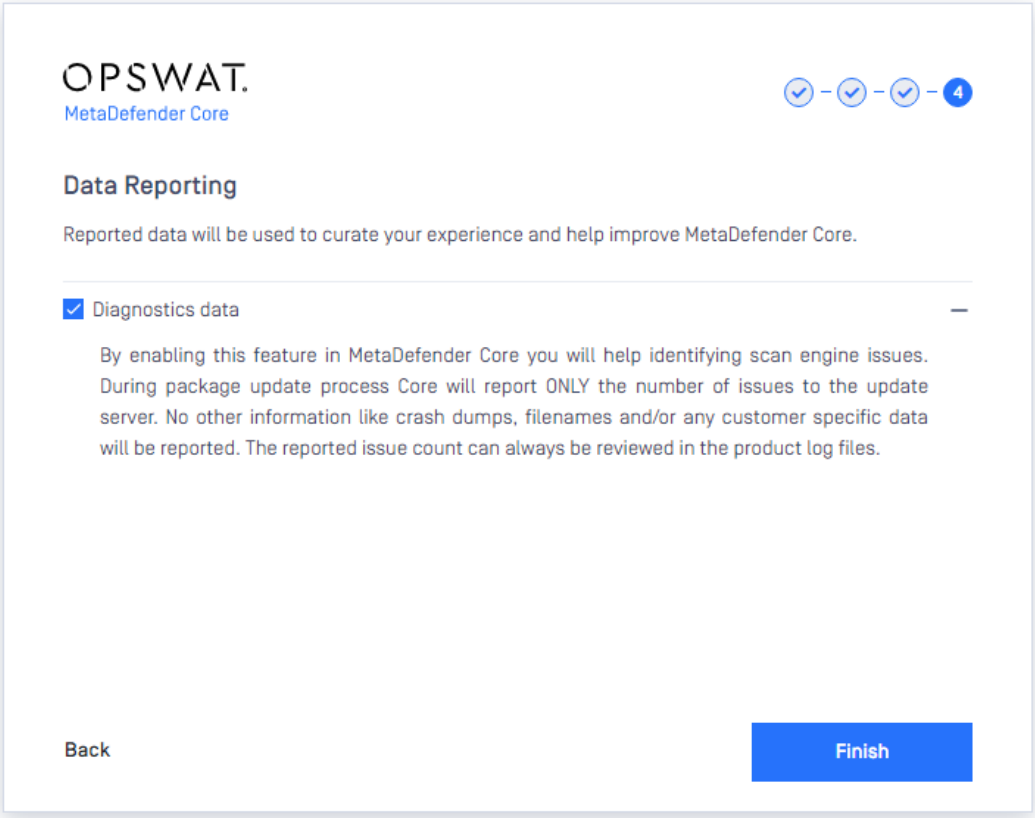
License activation

For license activation details see License Activation

Data Reporting

Enabling this option to allow MetaDefender Core collect its own engine issues and report back to OPSWAT for further analysis.

MetaDefender will not collect any other information.

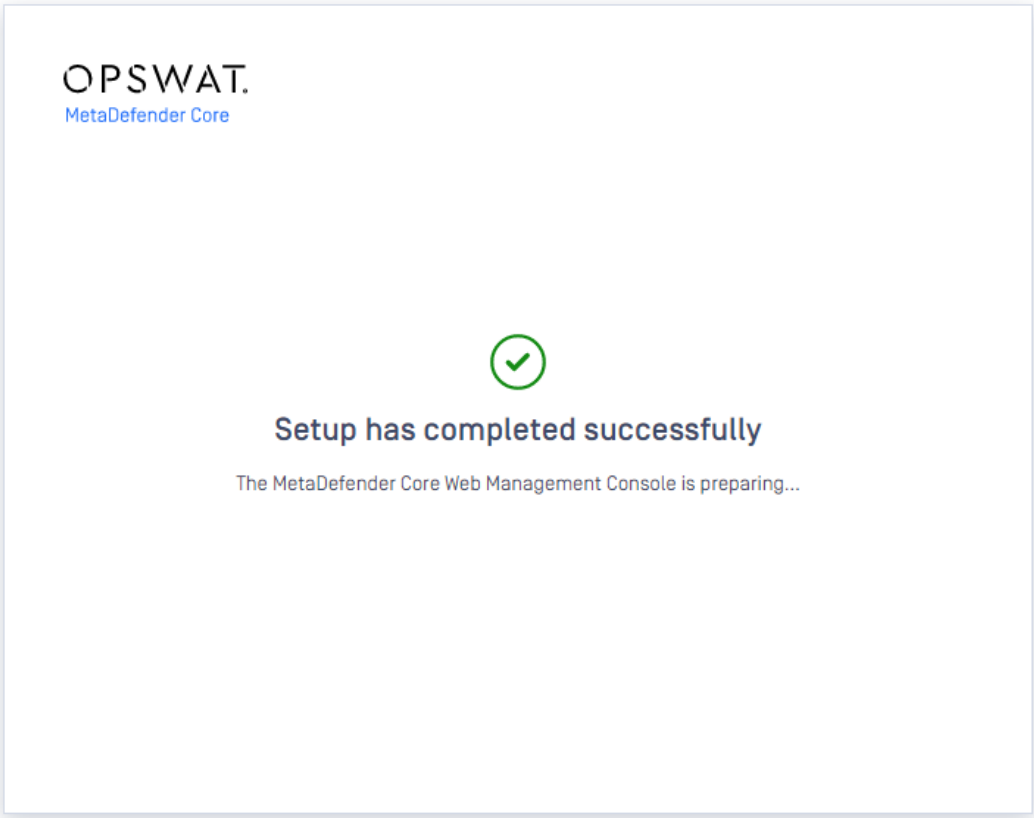


Wizard completion

After you have completed every steps you are ready to finish the wizard and start using the product. Click the `Fishish` button to complete the wizard.

> **Warning**
> The product's service will be restarted and the browser will be redirected to the administrative management console. This could take several seconds.

You can login to the Web Management Console with the administrator user that have just been created in the previous steps.



Setup HTTPS

Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network. Websites, like the Web Management Console, are able to use TLS to secure all communications between their servers and web browsers.

The TLS protocol aims primarily to provide confidentiality (privacy) and data integrity between two communicating computer applications.

Warning

No TLS for the wizard

By default, TLS is not enabled for the basic configuration wizard. As a consequence sessions between the wizard's backend and the browser may be insecure.

Info

Performing the same steps as for the Web Management Console, it is possible to set up TLS for the basic configuration wizard. Remember completing the TLS setup before launching the wizard.

For instructions to set up TLS see Enabling HTTPS

License Activation

##

To activate your installation go to the **Settings > License** menu in the Web Management Console. If you have no valid license, you will only see your installation's **deployment ID**. You will also see a warning in the Web Management Console header.

After successful activation, the product will start downloading the latest available scan engines and malware databases. You can follow the status of the scan engine installation on the **Inventory > Modules** page.

When all the licensed engines are installed, you can start using the MetaDefender Core to process files - File Processing.

Deployment ID

Each MetaDefender Core instance represented by an unique system fingerprint called `Deployment ID`

MetaDefender Core licensing is purely based on deployment ID activated. Therefore, everytime your MetaDefender Core is activated with a different deployment ID, then it will be captured as a different activation record accumulated into your total activation times allowed.

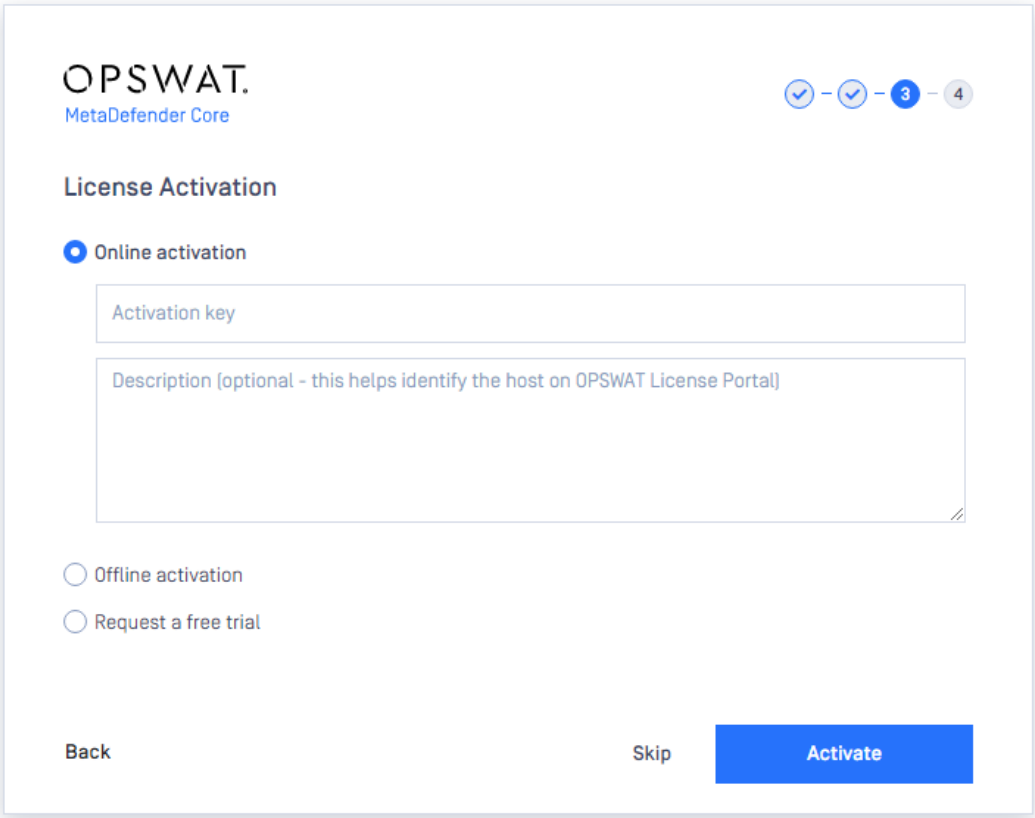
> **Notice**
> Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you alter the system where MetaDefender Core relies on (e.g. network card, OS hostname).
>
> Deployment ID is an UUID version 4 random based string. Learn more at: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you update the system where MetaDefender Core relies on.

Activation Modes

There are three different options for the product activation:

- Online activation: MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.
- Offline activation: Requiring you to upload a license file (.yml). That license file could be retrieved by activating your license via OPSWAT Portal. Follow the displayed instructions for details.
- Request trial key online: For evaluation purpose, you can receive an trial Activation key via email. Follow the displayed instructions for details.



Online activation

MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.

Required to input your Activation License key, and optional for description (but we recommend this for troubleshooting and auditing purpose if needed).

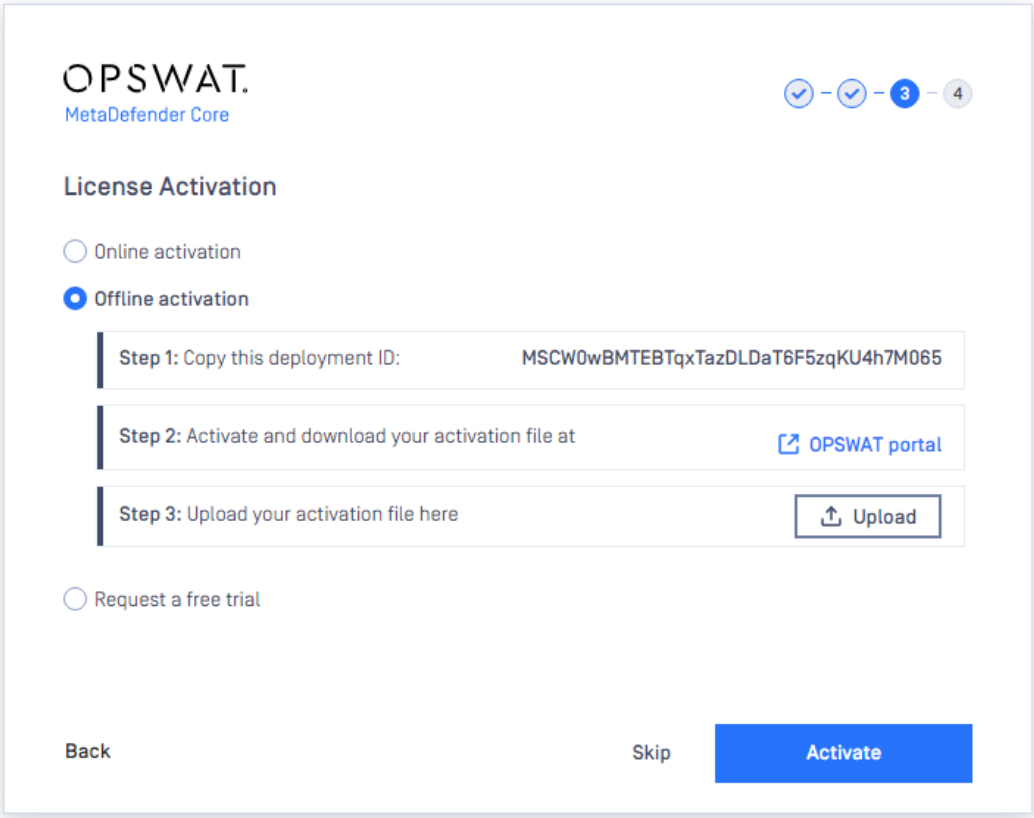
Warning

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts' for license activation:

- https://activation.dl.opswat.com Note: *IP address-based whitelisting on your firewall might fail after some time since OPSWAT uses CDN (Content Delivery Network) to faster delivery updates over the world, and IP address of edge servers might change over time.*

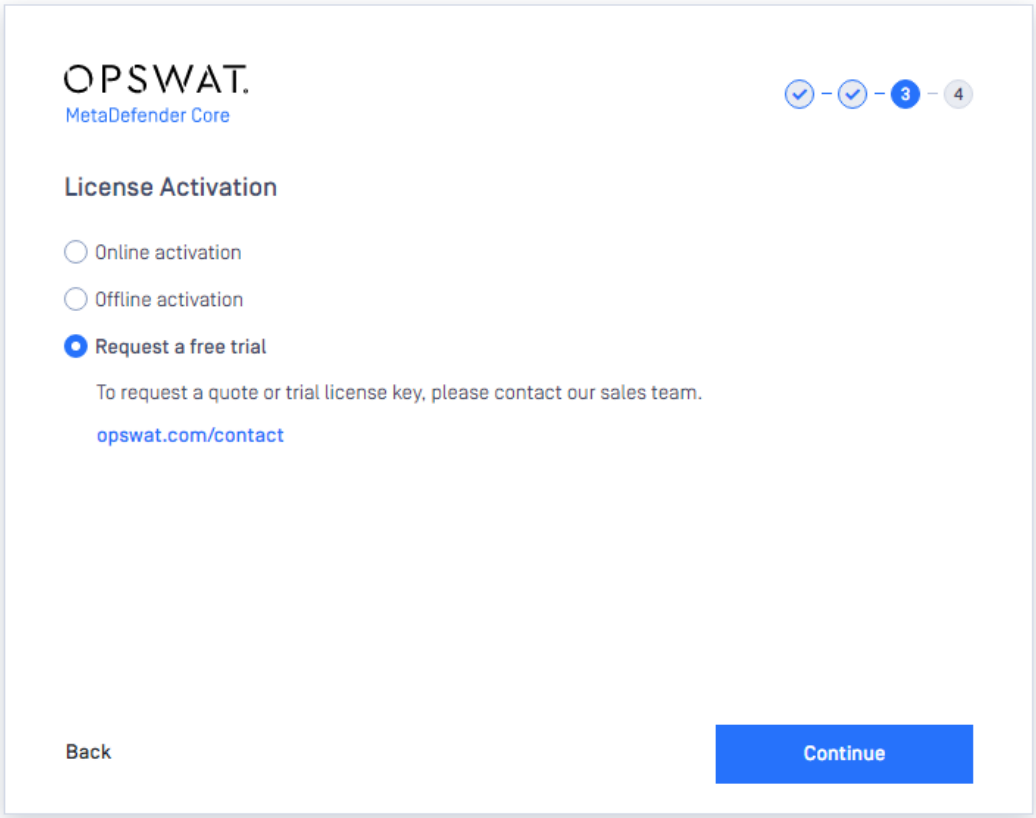
Offline activation

With no internet connection on the server the Metadefender v4 instance may be activated indirectly from a different machine, that has internet connection. The Deployment ID of the Metadefender v4 instance and the the Activation key received at the time of purchasing the product will be required. Follow the steps on the screen to activate the product offline.



Request trial key online

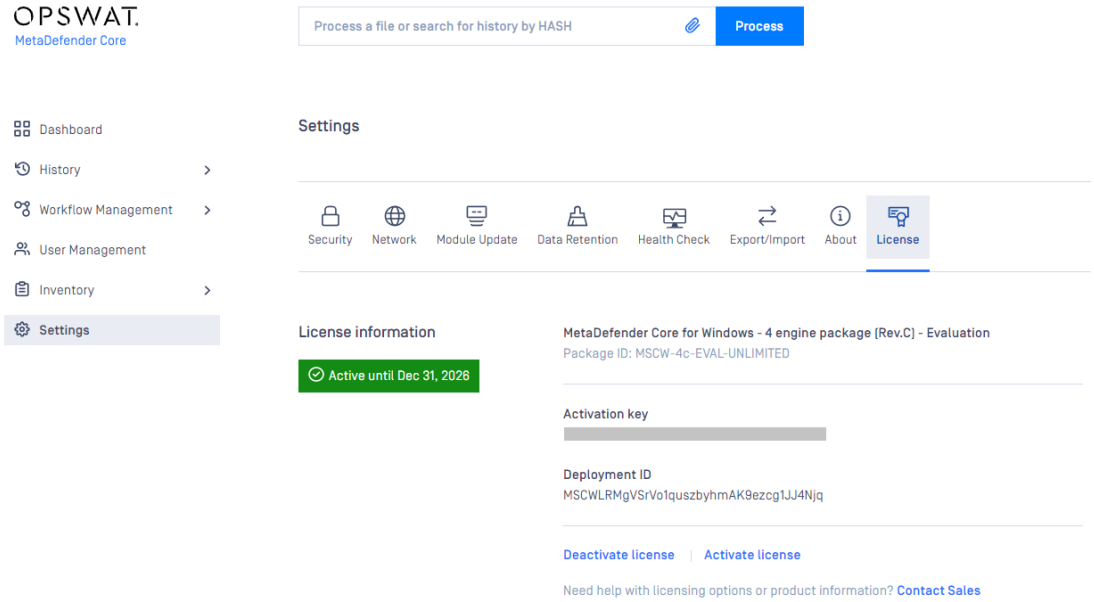
An evaluation license could be acquired by contacting our sales team.



Check Your License Details

For more license details and activating your MetaDefender Core, go to Settings > License menu tab on the Web Management Console:

- Product name: product name as on your order (e.g. MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation)
- Package ID: product identification as on your order (e.g. MSCW-4c-EVAL-UNLIMITED)
- Expiration: last day of license validity
- Activation key: key used for activating product
- Deployment ID: identification of this installation


OPSWAT.
MetaDefender Core
Process a file or search for history by HASH
Process
Dashboard
History
Workflow Management
User Management
Inventory
Settings
Settings
Security
Network
Module Update
Data Retention
Health Check
Export/Import
About
License
License information
Active until Dec 31, 2026
MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation
Package ID: MSCW-4c-EVAL-UNLIMITED
Activation key
Deployment ID
MSCWLRMgVSrVo1quszbyhmAK9ezcg1JJ4Njq
Deactivate license
Activate license
Need help with licensing options or product information? Contact Sales

Scripted license management

Using REST API calls there is an option for scripted activation and deactivation for MetaDefender Core servers with **no Internet connection or if your infrastructure is using automation to create/destroy MetaDefender Core instances**. In this chapter the steps of these two scenarios are described.

Requirements

- an installed MetaDefender Core instance without Internet connection
- another computer that has Internet connection and can run your activation scripts
- a manual or automated way to transfer data between the two computers

Activation steps

1. In order to activate MetaDefender Core, deployment ID and activation key are needed to generate the license file.
    a. Activation key should be purchased from OPSWAT.
    b. MetaDefender Core deployment ID can be queried by using REST API (for details see GET Get Current License Information API page)

> **Info**
> Save this deployment ID in your system, you might need this ID when the instance is unavailable at the time of deactivation.

2. Activate license and get the license file using the following URL, where <quantity> is 1.

activation call bash

```
curl -X GET https://activation.dl.opswat.com/activation?key=<activation-key>&deployment=<deployment unique-ID>&quantity=<quantity>
```

If the activation is successfully, a license file is downloaded. Save this file.

HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	license file	Activation was successfully.
200 Activation failed	error: '<user conform error message>'	Failed activation
200 Invalid parameter	error: 'Could not activate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not activate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
200 Invalid parameter	error: 'Could not activate your product because the quantity you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key, quantity or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Info
In case of any activation issue, contact OPSWAT support for help

3. Upload license file to MetaDefender Core: The license file should be uploaded to the MetaDefender Core v4 to activate the product. For details see page: POST Upload license key API

Deactivation steps

1. For deactivation, following infomation is required:
    a. Activation key
    b. MetaDefender Core deployment ID: can be queried by using REST API (for details see GET Get Current License Information API page)
2. Deactivate license using the following URL:

deactivation call bash

```
curl -X GET https://activation.dl.opswat.com/deactivation?key=<activation-
key>&deployment=<deployment unique-ID>
```

Response HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	result: ok	Successful deactivation
200 No active license found	error: 'Could not found any active license with the given parameters'	The license has not been activated yet or it has been deactivated already.
200 Invalid parameter	error: 'Could not deactivate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not deactivate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Important notes

Warning

Product activation is tied to several hardware, operating system and software parameters. In case of one or more major hardware or software parameter change the product might turn into deactivated status. Operating system updates, other software updates should not affecting the activation status.

If the product is online activated and has live Internet connection then in case of deployment ID change the product reactivates itself. If the product doesn't have Internet connection then the administrator is responsible to reactivate the product.

Because of the above administrator should consider licensing restrictions of the offline deployments if using any virtualization and/or containerization technologies.

File Processing

MetaDefender Core provides REST API interface for file processing business.

There are several ways to process files with MetaDefender Core:

- Analyze Files via Web Interface
- Process Files via REST API

Product Upgrading

> **Info**
> Simply run installer file, details could be found at Installation

> **Info**
> For command line mode, upgrading (e.g. on **Linux**), prior to running the installer file, please make sure to **create the ignition file** with PostgreSQL server information included. Details:
>
> - Standalone DB mode: Command Line Installation (Windows and Linux)
> - Shared DB mode: Command Line Installation

Recommendation for MetaDefender Core upgrade

Our goal is to help you execute the product upgrade continuously, efficiently, and correctly.

PostgreSQL service upgrade

- **Bundled PostgreSQL Service**: If you are using the bundled PostgreSQL server included in the installer, the PostgreSQL service will be automatically upgraded to the newer version (if available). No additional action is required for this process.
- **Remote PostgreSQL Service**: For remote PostgreSQL servers, customers are responsible for managing and upgrading the service independently. Upgrading MetaDefender Core does not update the remote PostgreSQL service.

Before the upgrade

Prepare a proper disaster recovery plan (DRP) including some common practices:

- Backup, or take a snapshot of your current working MetaDefender Core server.
  - This is to help quickly rollback for any upgrade failure might cause later for any reason.
- Optionally backup your PostgreSQL database, especially when using remote shared database mode.
  - Reference: https://www.postgresql.org/docs/current/backup.html
  - Please do it after stopping all linked MetaDefender Core instances to avoid potential data collision.
- Lastly, if backing up entire server is not feasible, then at least export MetaDefender Core configuration package Import/Export configuration just in case you'll need to start off a new server with current MetaDefender Core configuration to keep the scanning service running without any impact.

On the PostgreSQL server, double check server resource to ensure sufficient RAM memory and disk space to be preserved for the upgrade process.

- 8GB RAM available at least during the upgrade.

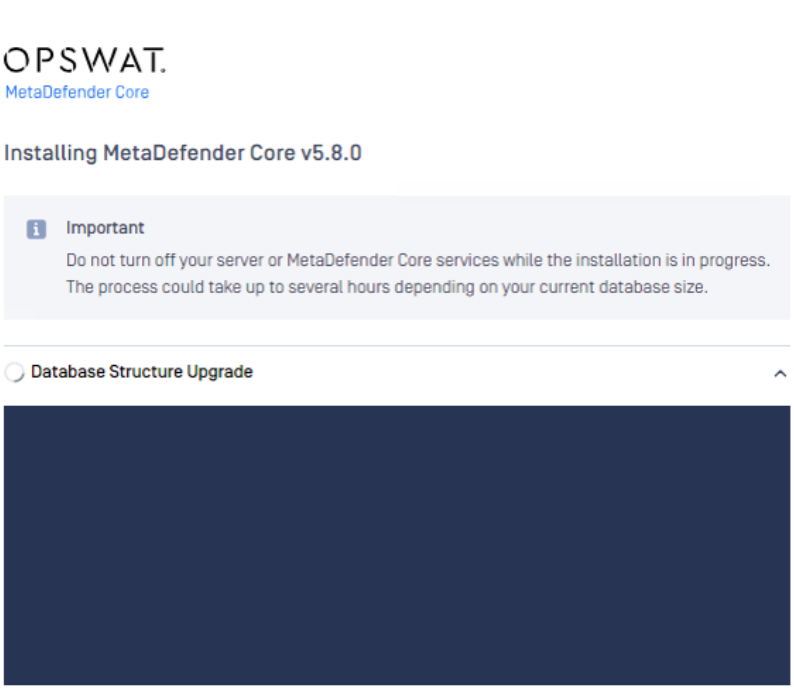
- Disk availability is greater than MetaDefender Core database size.

While doing the upgrade

Applicable for shared database deployment:

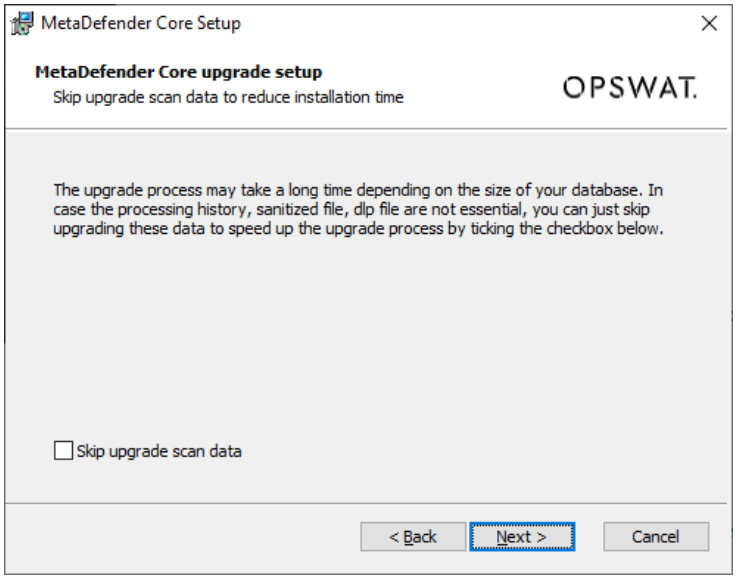
- Stop all MetaDefender Core instances' scanning service before performing the upgrade.
- Upgrade one (1) MetaDefender Core instance at a time.
- Proceed to upgrade MetaDefender Core instance where the bundle local PostgreSQL server is used for shared database setup.
- Otherwise is using remote PostgreSQL then the upgrade order does not matter.

For large size database, mostly for scan history, expecting long time to be spent on the upgrade process.



The product supports an option to skip data migration for scan history to expedite the process drastically and mitigate any failure risk. You can manually backup all data from your PostgreSQL server for offline usage if any.

Check "Skip upgrade scan data" to enable this feature while doing the upgrade.



or add following key in the ignition file:

none

```
[internal]
skip_upgrade_scan_data = 0
```

Troubleshooting

If the upgrade process ever runs into an error:

- Double check your network and services, applicable to remote PostgreSQL server.
- Optionally configure MetaDefender Core to skip migrating scan history.
  - Windows
    - Open Windows registry, and navigate to HKLM\Software\OPSWAT\Metascan
    - Add a new key `internal/skip_upgrade_scan_data` with value = 1
  - Linux
    - Open MetaDefender Core configuration file `/etc/ometascan/ometascan.conf`
    - Add new key into the file
      - [internal]
      - skip_upgrade_scan_data=1
  - Note: This key will be auto removed upon the upgrade finishes.
  - Then restart MetaDefender Core service to retry the upgrade.
    - Might retry this a few times until the upgrade is done.

Just in case the upgrade process still fails, please contact OPSWAT support with the support package - How to Create Support Package? This will help us analyze the root cause and figure out an optimal solution for your system failure.

Using external Load Balancer

Application Load Balancing (Layer 7)

This page provides information about recommended ways to use sessions with a Layer 7 load-balancer.

Most load-balancers have the ability to provide cookies for pinning subsequent traffic from a client to the appropriate server. This method is called sticky session, session persistence or session affinity. Using cookies efficiently requires the client to know when a cookie should be sent, should not be sent or should be deleted.

Sticky session load balancing

Each Metadefender Core instance has its own databases and application sessions that cannot be seen by other instances. Therefore, in order to get the related data/response to our queries we should ask the appropriate Metadefender Core v5 server. On the other hand, to keep the advantages of the used load balancing method, cookies should not be sent if it is not necessary.

Single file scanning

Step	Stage	Task	Cookie usage
1.	Sending file	Initiate processing a file on the client side. Send file through the load-balancer. (See POST File Analysis API)	Cookie should not be sent
2.	Sending file	Save the cookie and the data_id you got from the load-balancer	Save cookie
3.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Send cookie saved in step 2
4.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Send cookie saved in step 2

Batch scanning

Step	Stage	Task	Cookie usage
1.	Open batch	Initiate processing file(s) in batch. Request a batch ID through the load-balancer. (See POST Initiate Batch API)	Cookie should not be sent
2.	Open batch	Save the cookie and the batch_id you got from a Core server through the load-balancer.	Save cookie
3.	Sending files	Send file through the load-balancer. (See POST File Analysis (using batch header) API)	Send cookie saved in step 2
4.	Sending files	Save the data_id you got from the load-balancer.	-
	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Send cookie saved in step 2
5.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
6.	Getting batch status	See GET Batch Status API	Send cookie saved in step 2
7.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Send cookie saved in step 2
8.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API	Send cookie saved in step 2

If it does not matter which upstream server responds, the queries should be sent without cookie.

> **Info**
> It is recommended not to send cookies when it's not necessary to allow load-balancer to use its own method to share the load between Metadefender Core v5 servers.

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration through load-balancer

OPSWAT products that support HTTP load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	4.3.4	-
OPSWAT Client	Windows: 7.6.247.0 Mac: 10.4.243.0	-
MetaDefender Email Security	4.3.0	-
MetaDefender ICAP Server	4.3.0	-
MetaDefender Vault	1.3.0	-
MetaDefender for Secure Storage	2.0.7	-

DNS Load Balancing

Using this method is logically similar to Layer 7 load-balancing.

Briefly how it works

Client uses a domain name to send a query to a server. Client's DNS server has more "A" records for that name with different IPs. When a client resolves the server's domain name DNS server randomly chooses an IP for that name to send back. When a session is used on the application layer, client should know the IP address of the Core server that handles that specific session. In every other case, client should resolve the domain name with DNS query to let requests to be balanced between Metadefender Core servers.

Single file scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Sending file	Initiate processing a file on the client side. (See POST File Analysis API)	Use IP saved in step 2
4.	Sending file	Save the data_id got from the Core server	-
5.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Use IP saved in step 2
6.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Use IP saved in step 2

Batch scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Open batch	Initiate processing file(s) in batch. Request a batch ID. (See POST Initiate Batch API)	Use IP saved in step 2
4.	Sending files	Send file to the specific Core server with the basch ID saved in step 3. (See POST File Analysis (using batch header) API)	Use IP saved in step 2
5.	Sending files	Save the data_id you got from the Core server.	-
6.	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Use IP saved in step 2
7.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
8.	Getting batch status	See GET Batch Status API	Use IP saved in step 2
9.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Use IP saved in step 2
10.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API).	Use IP saved in step 2

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration via DNS load-balancing

OPSWAT products that support DNS load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	does not support yet	-
MetaDefender Client	does not support yet	-
MetaDefender Email Security	does not support yet	-
MetaDefender ICAP Server	does not support yet	-
MetaDefender Vault	does not support yet	-

SSL connection for PostgreSQL communication

> **Note**
> The SSL connection configuration is possible thanks to the support made by PostgreSQL. For reference: https://www.postgresql.org/docs/14/ssl-tcp.html
>
> MetaDefender Core is tested against this mode on particular version 4.21.1

Step 1: Generate SSL certificate for PostgreSQL server

> **Note**
> Following instructions are for self-sign certificate. For production environment, making sure to use the certificate signed by your trusted CA.

1) Generate a private key

bash

```
openssl genrsa -des3 -out server.key 1024
```

Sample output:

bash

```
Generating RSA private key, 1024 bit long modulus ........++++++
...............++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

Type your desired password / pass phrase to encrypt the private key

2) Remove the pass phrase to automatically start up the PostgreSQL server

bash

```
openssl rsa -in server.key -out server.key
```

> **Note**
> On Linux, make sure to set permission on the server.key file

bash

```
chmod og-rwx server.key
```

3) Create a self-signed certificate

bash

```
openssl req -new -key server.key -days 3650 -out server.crt -x509
```

You will be prompted to enter detailed information which is incorporated into your self-signed certificate request.

none

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:CA
Locality Name (eg, city) [Default City]:San Francisco
Organization Name (eg, company) [Default Company Ltd]:OPSWAT
Unit Name (eg, section) []:OPSWAT
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:dummy_email@opswat.com
```

For self-signed certificate, use the server ceriticate as the trusted root certificate:

bash

```
cp server.crt root.crt
```

Copy **server.key**, **server.crt**, **root.crt** to PostgreSQL data folder. For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data**
- Linux: **/var/lib/ometascan/pg_data/**

Step 2: Configure PostgreSQL server for SSL authentication connection

1) Create a custom config file (e.g. ssl.conf) for PostgreSQL.

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\postgres\ssl.conf**
- Linux: **/var/lib/ometascan/postgres/ssl.conf**

2) Content of ssl.conf:

none

```
ssl = on
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
```

3) Modify pg_hba.conf

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data\pg_hba.conf**
- Linux: **/var/lib/ometascan/pg_data/pg_hba.conf**

Add following (modify **::1/128** if needed to match with your environment requirement)

none

```
hostssl all all      ::1/128      scram-sha-256
```

After the modify, the content of pg_hba.conf should be like this (example):

> **Note**
> In a Linux environment, a local connection is required to upgrade from version 5.5.1 or earlier to version 5.6.0 or later.

none

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local         DATABASE  USER  METHOD  [OPTIONS]
# host          DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl     DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostgssenc    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnogssenc  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
local    all all                    scram-sha-256
hostssl all all      ::1/128      scram-sha-256

#More content.......
```

4) Reload the services

- Standalone DB mode: Restart **ometascan** service (MetaDefender Core service)
- Shared DB mode: Restart **ometascan-pg** / **ometascan-postgresql** service (MetaDefender Core PostgreSQL service) and then **ometascan** service (MetaDefender Core service)

Test your SSL database connection:

bash

```
psql.exe "sslmode=require dbname=postgres user=postgres host=localhost port=5432"
Password for user postgres:
psql (12.7)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
Type "help" for help.
```

This message indicate that you are now using SSL connection:

bash

```
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
```

Performance and Load Estimation

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Some factors that affect performance

- MetaDefender Core version
- MetaDefender Core engine package and configuration
  - set of engines (which and how many)
  - product configuration (e.g., thread pool size)
- System environment
  - server profile (CPU, RAM, hard disk)
  - client application location - remote or local
  - system caching and engine level caching
- Dataset
  - encrypted or decrypted
  - file types
    - different file types (e.g., document, image, executable)
    - archive file or compound document format files
  - file size
  - bad or unknown (assume to be clean)
- Performance tool

How test results are calculated

Performance (mainly scanning speed) is measured by throughput rather than unit speed. For example, if it takes 10 seconds to process 1 file, and it also takes 10 seconds to process 10 files, then performance is quantified as 1 second per file, rather than 10 seconds.

- total time / total number of files processed: 10 seconds / 10 files = 1 second / file.

Standalone DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.2.416.0

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

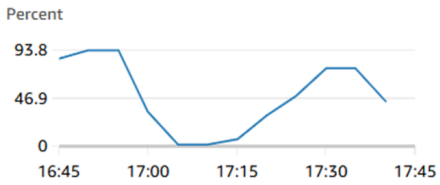
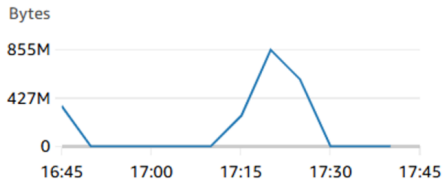
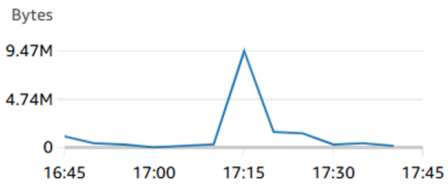
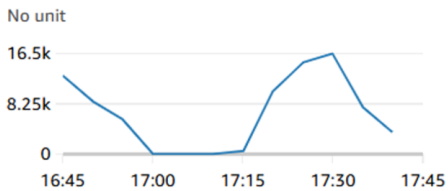
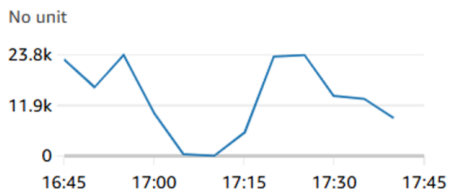
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV)	24.2	47.2 / 80	37.3 / 41.1		315,310	0.0114
Deep CDR	19.9	48.9 / 81.8	32.6 / 37		383,442	0.0094
Proactive DLP	17.1	36.8 / 58.2	30.4 / 33.6		446,228	0.0081
Vulnerability	10.1	59.2 / 85	29.3 / 33.3		755,495	0.0048

MetaDefender Core with common engine packages:

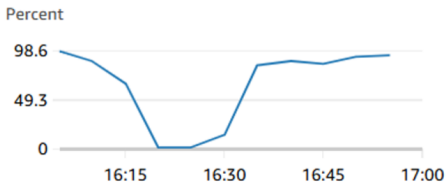
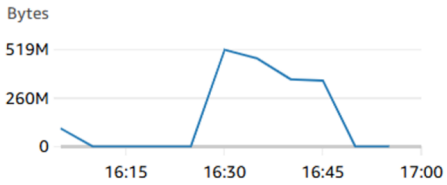
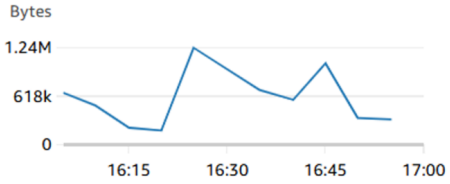
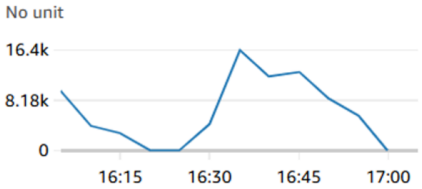
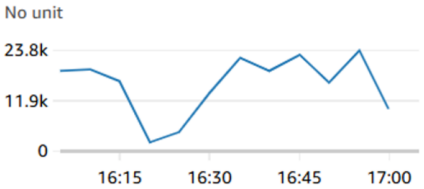
Use case	Scan duration	Avg./Max CPU usage	Avg./Max RAM usage	Avg./Max Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV) + Deep CDR	30.8	68.1 / 88	40.1 / 46.8		247,744	0.0145
Metascan (8 AV) + Deep CDR + Proactive DLP	31.2	74.2 / 98.6	41.5 / 51.1		244,567.3	0.0147
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	31.8	73.5 / 98	40.6 / 51.5		239,953	0.015

System resource utilization charts

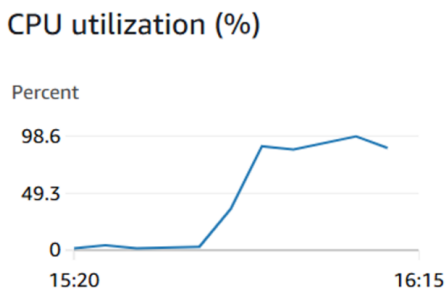
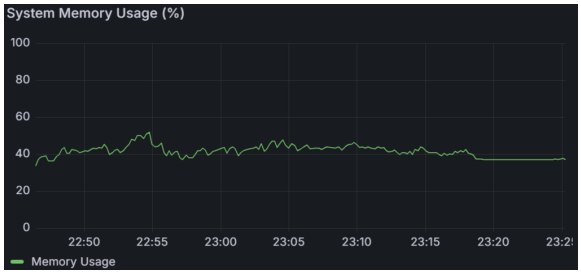
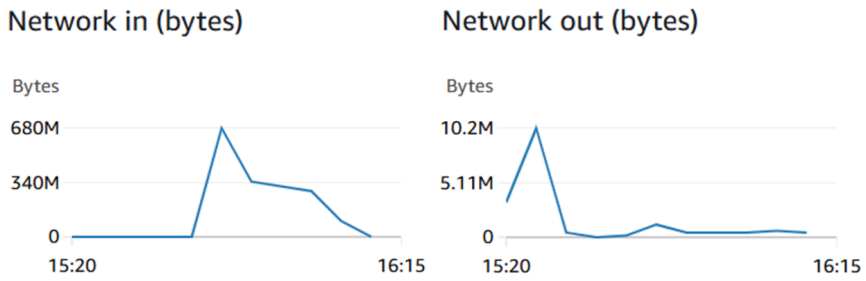
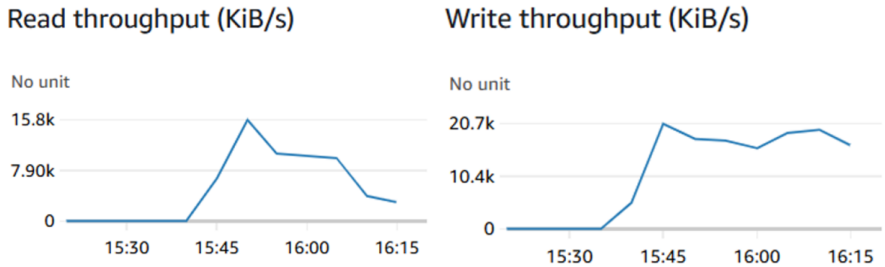
MetaDefender Core with Metascan (8 AV engines):

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

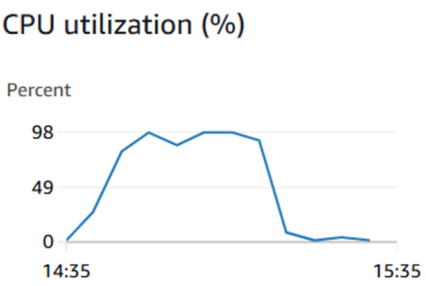
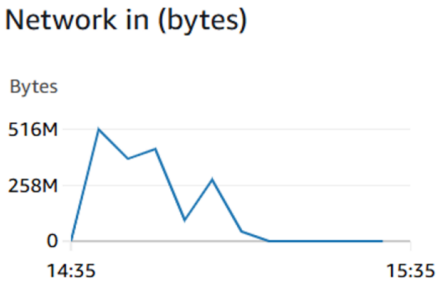
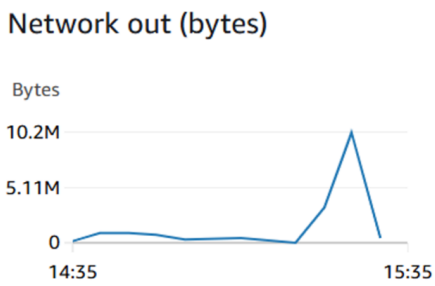
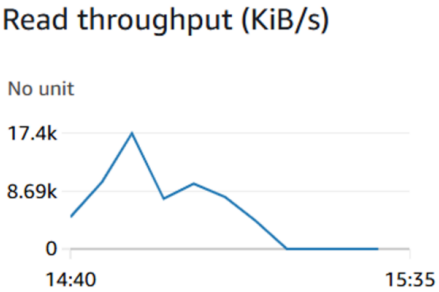
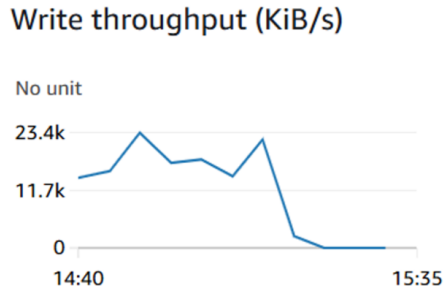
MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

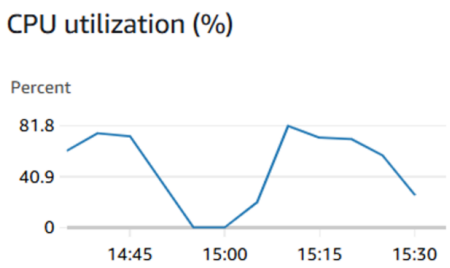
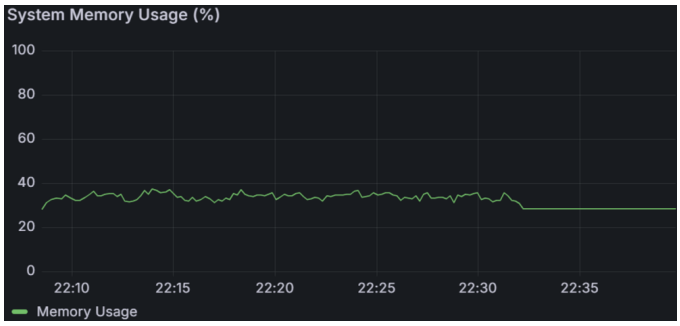
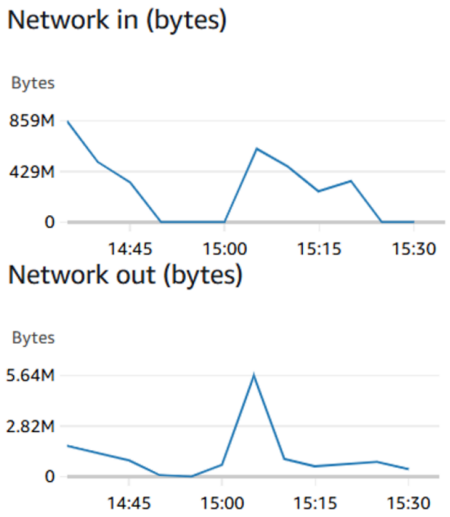
MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

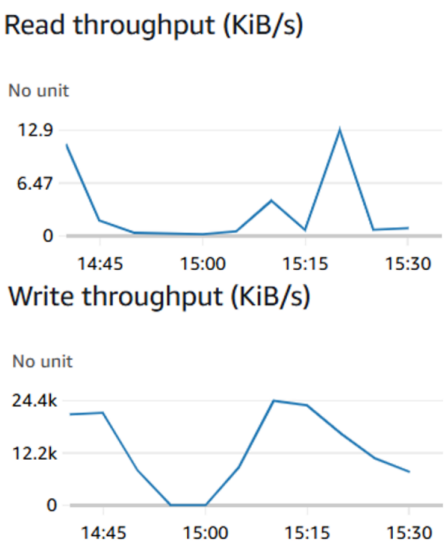
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

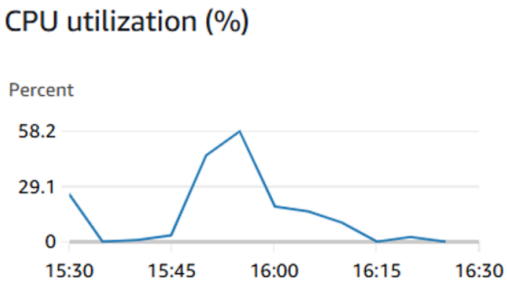
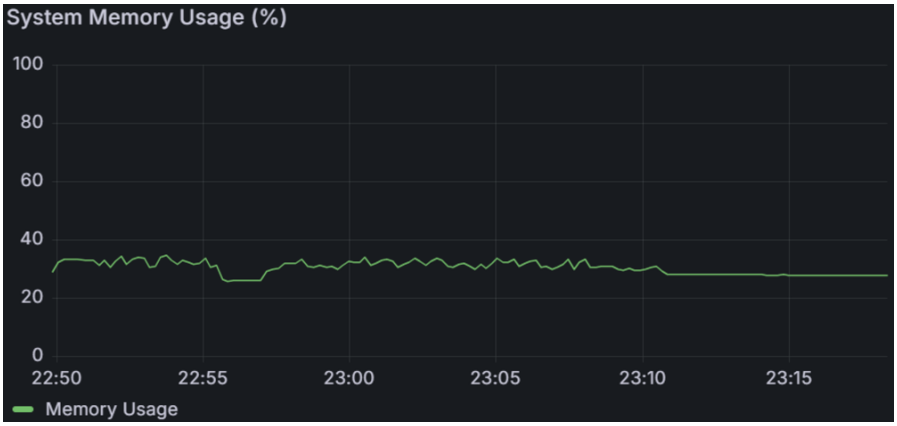
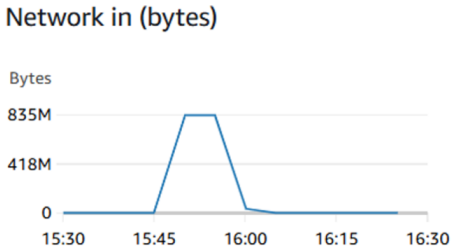
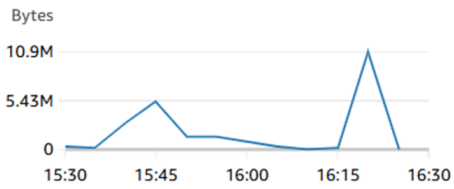
	Utilization charts
CPU usage	
Memory usage	
Network usage	 
Disk usage	 

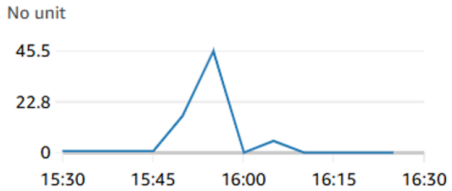
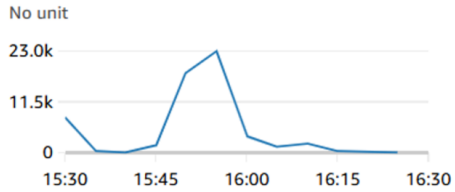
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

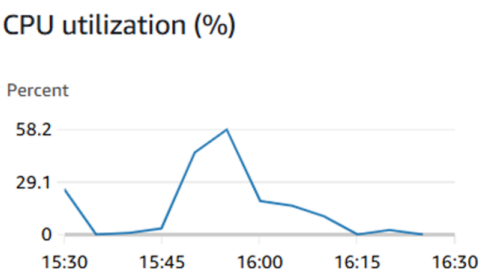
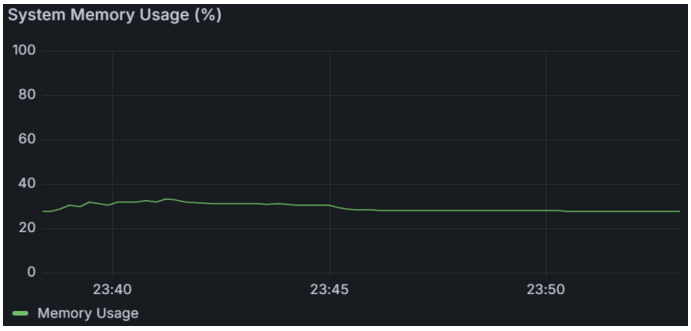
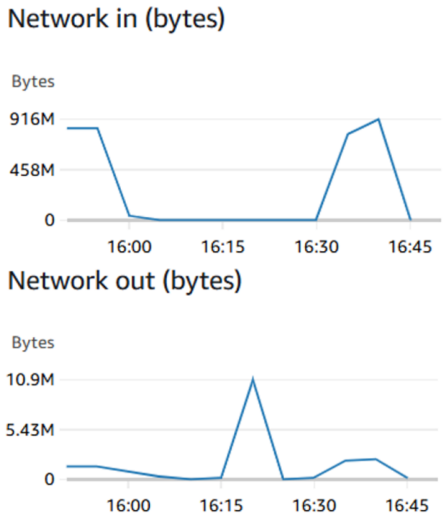
	Utilization charts
Disk usage	

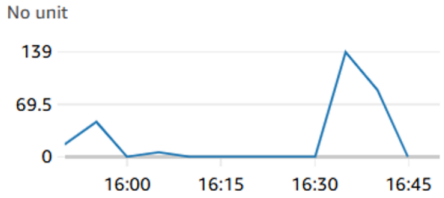
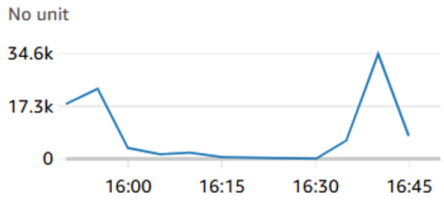
MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	 

	Utilization charts
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

MetaDefender Core with Vulnerability:

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

Standalone DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Ubuntu 22.04	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.57-236

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

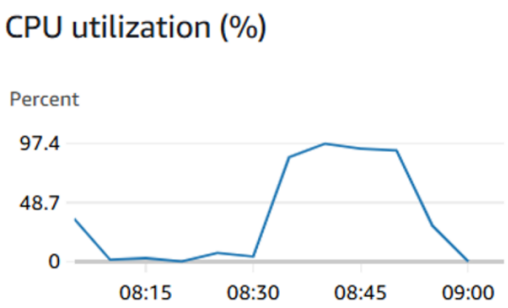
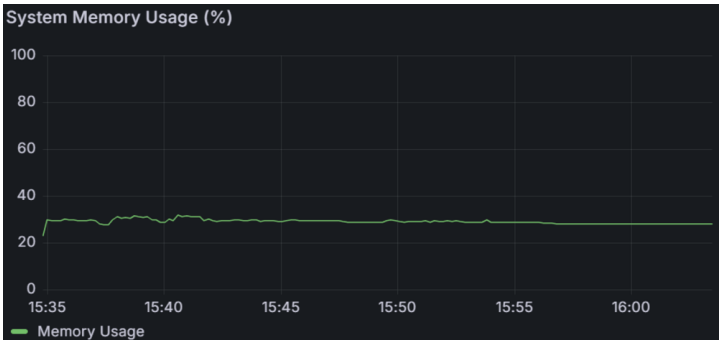
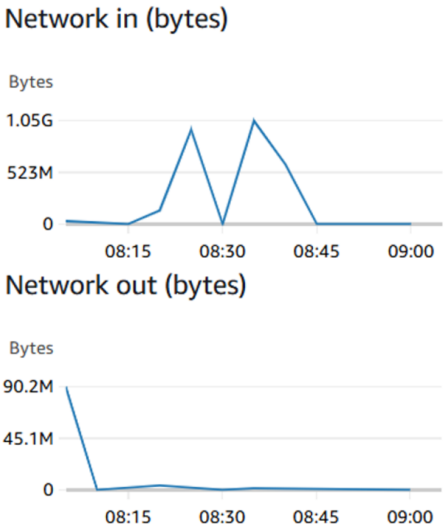
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV)	30.2	69 / 98	29 / 32		252,666	0.0142
Deep CDR	19.4	48 / 62	36 / 49		393,325	0.009
Proactive DLP	16.5	34 / 39	32 / 48		462,455	0.008
Vulnerability	12	50 / 65	46 / 46		635,875	0.006

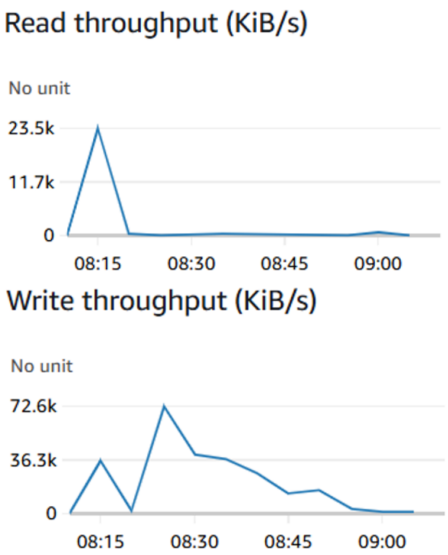
MetaDefender Core with common engine packages:

Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV) + Deep CDR	39.8	78 / 99	62 / 73		191,721	0.0188
Metascan (10 AV) + Deep CDR + Proactive DLP	43.5	61 / 84	47 / 67		175,414	0.0205
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	44.8	62 / 88	69 / 83		170,324	0.0211

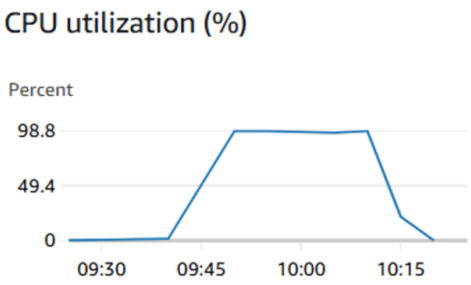
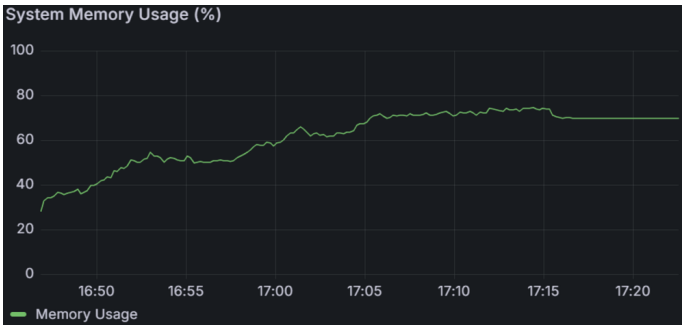
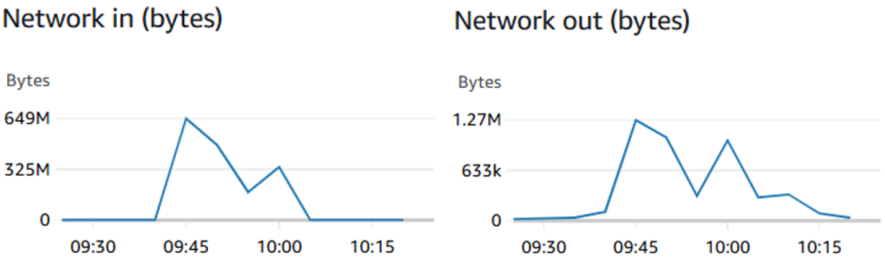
System resource utilization charts

MetaDefender Core with Metascan (10 AV engines) :

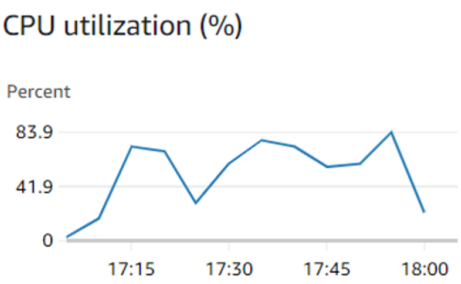
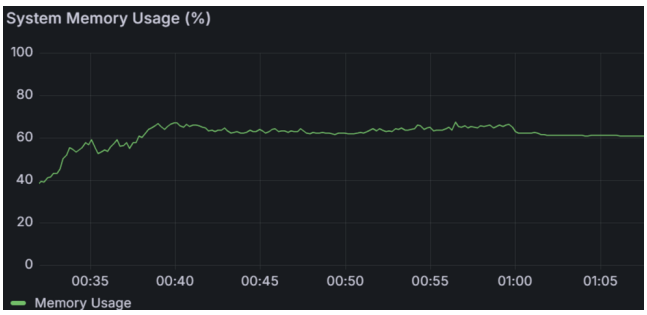
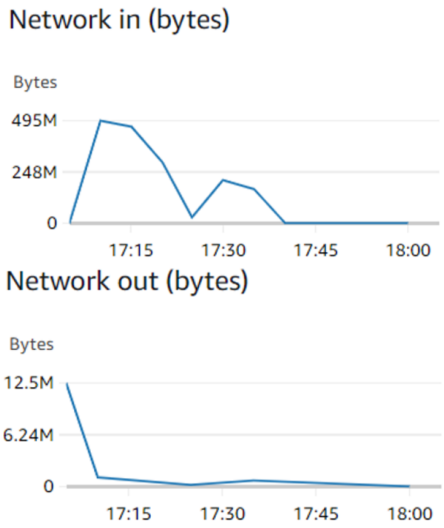
	Utilization charts
CPU usage	
Memory usage	
Network usage	

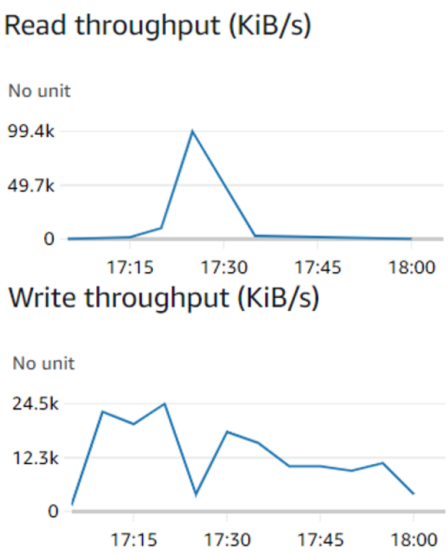
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR :

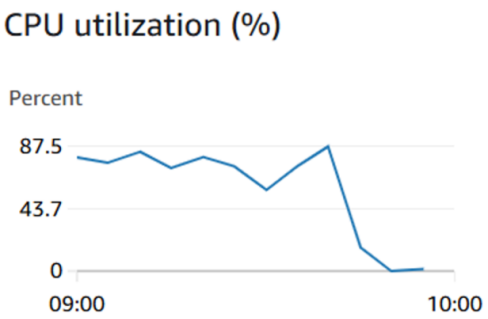
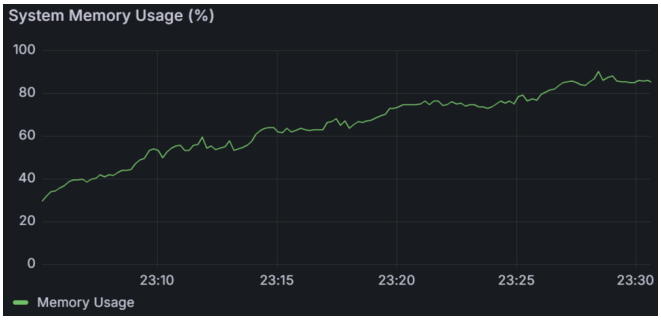
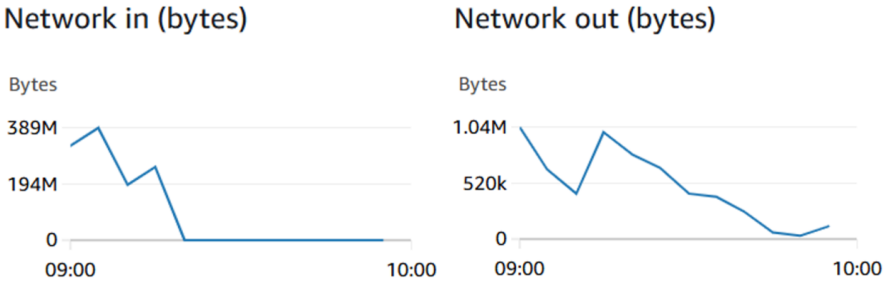
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP :

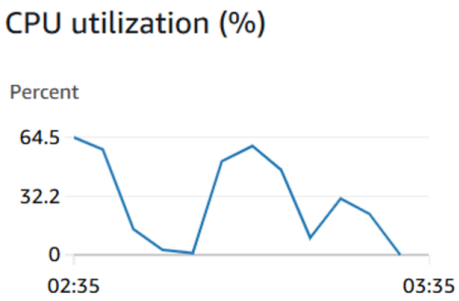
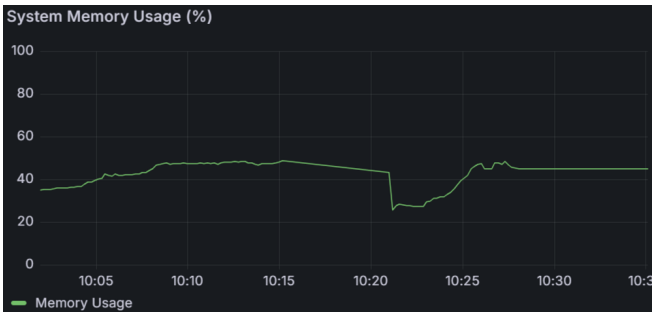
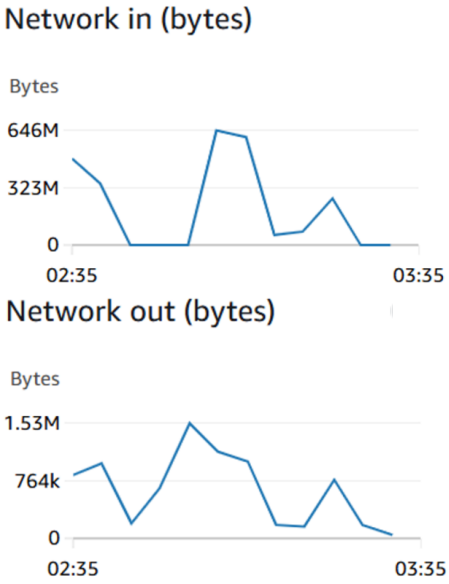
	Utilization charts
CPU usage	
Memory usage	
Network usage	

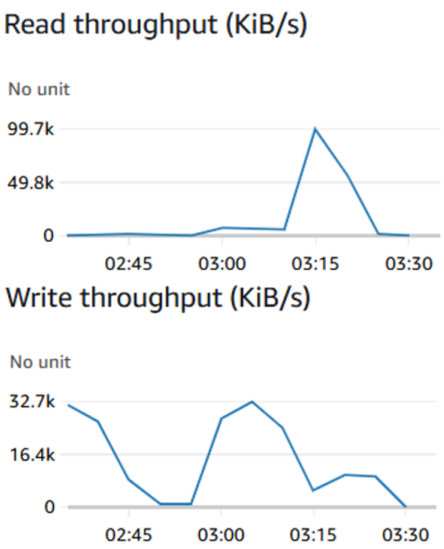
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP + Vulnerability :

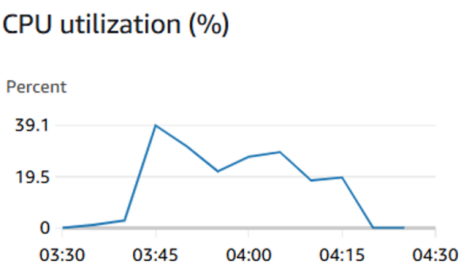
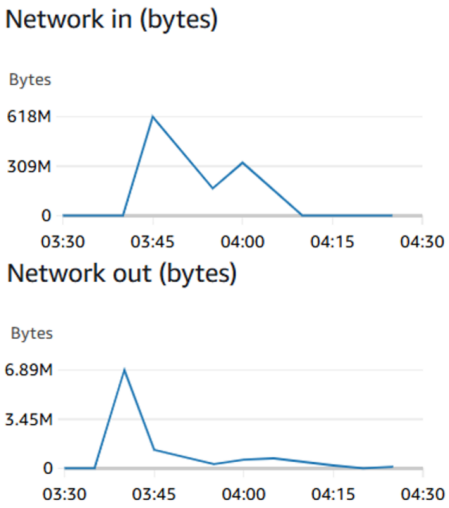
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

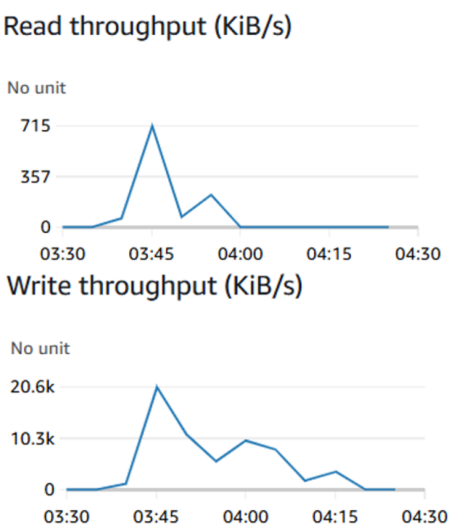
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

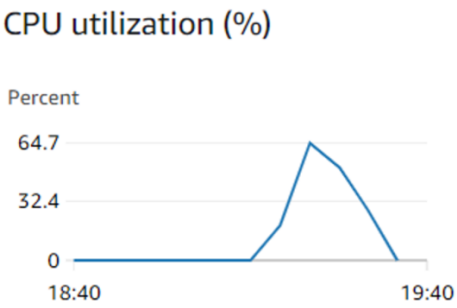
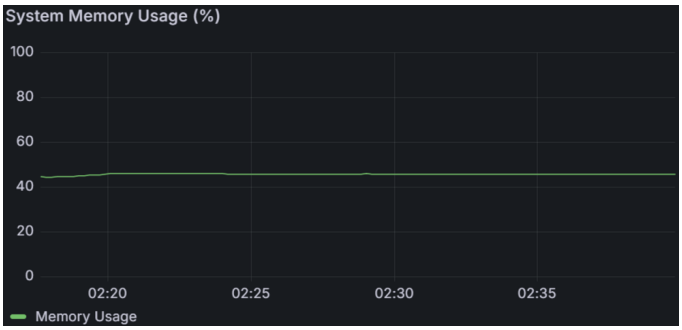
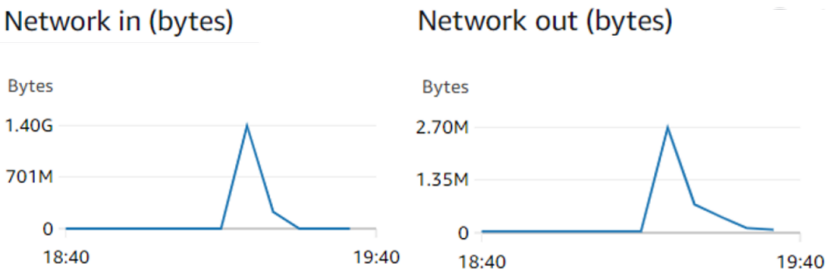
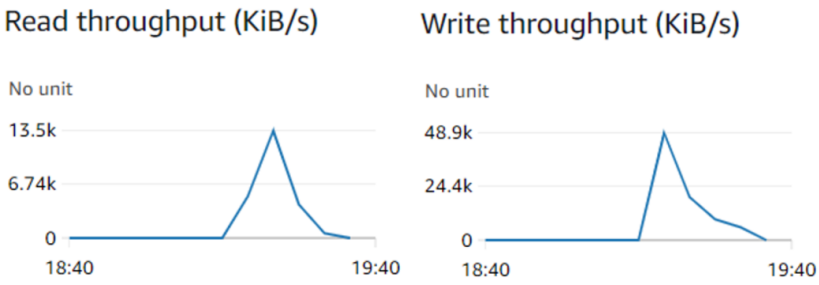
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability :

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

Shared DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

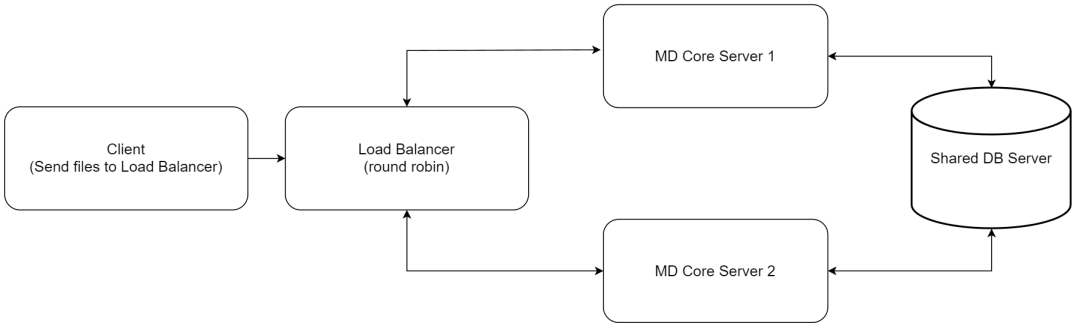
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV)	11.9	37.0	41.9	30.1	30.4	Receive: 6441.1 Send: 769.6	Receive: 6200.2 Send: 693.7	640,821.2	0.0
Deep CDR	10.5	60.1	66.7	30.3	32.2	Receive: 6379.8 Send: 594.8	Receive: 7146.1 Send: 609.6	729,368.5	0.0
Proactive DLP	7.8	38.7	37.5	29.7	31.0	Receive: 8842.6 Send: 791.8	Receive: 8635.6 Send: 729.1	965,455.7	0.0
Vulnerability	4.4	52.2	53.0	26.0	26.9	Receive: 16,215.5 Send: 1006.8	Receive: 15,785.7 Send: 997.2	1714,449.3	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV) + Deep CDR	15.4	72.4	75	34.3	35.4	Receive: 5436.6 Send: 611.3	Receive: 5589.3 Send: 607.2	495,697.4	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP	16.9	77.2	80.2	36.4	37.9	Receive: 5254 Send: 585.4	Receive: 5041.3 Send: 576.1	450,657.6	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	18.0	72.1	80.4	38.6	38.0	Receive: 4504.3 Send: 574.1	Receive: 4649.5 Send: 571.1	423,010	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

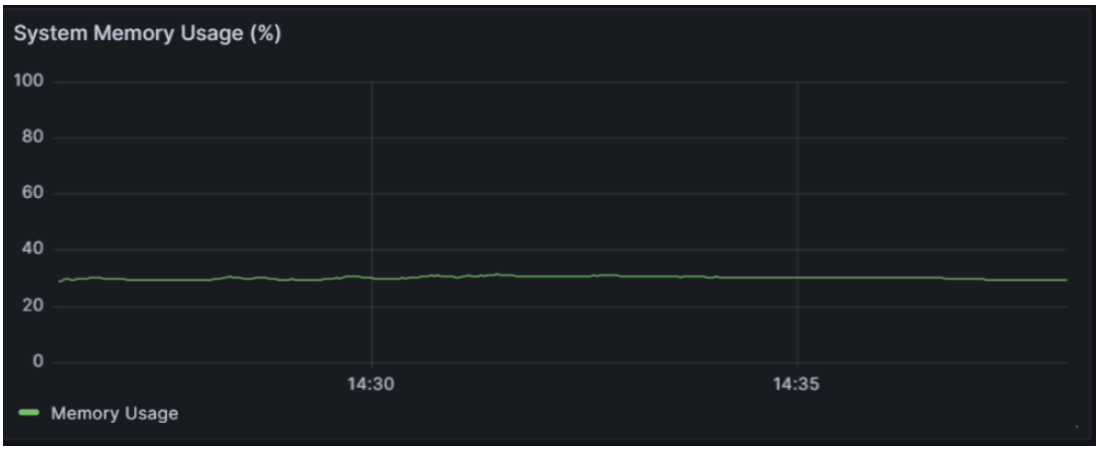
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2

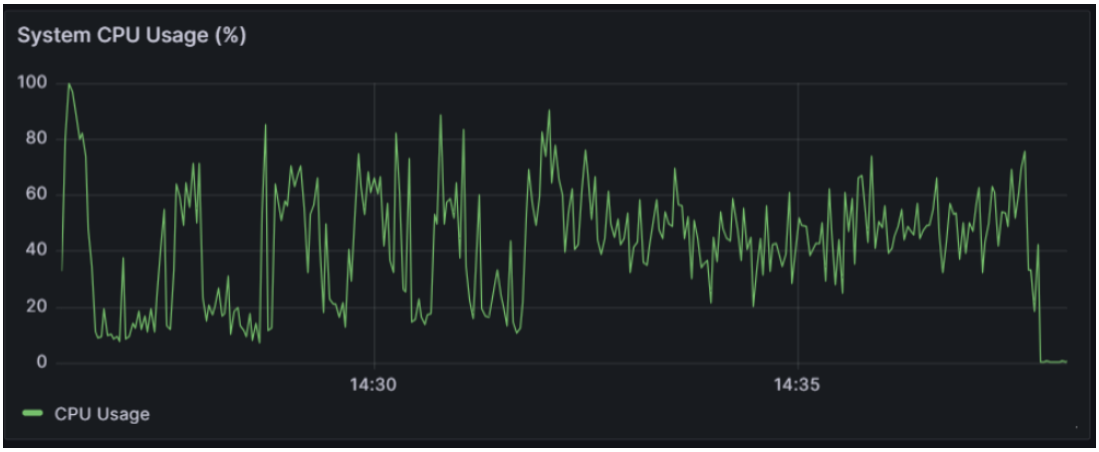


CPU usage

MD Core #1

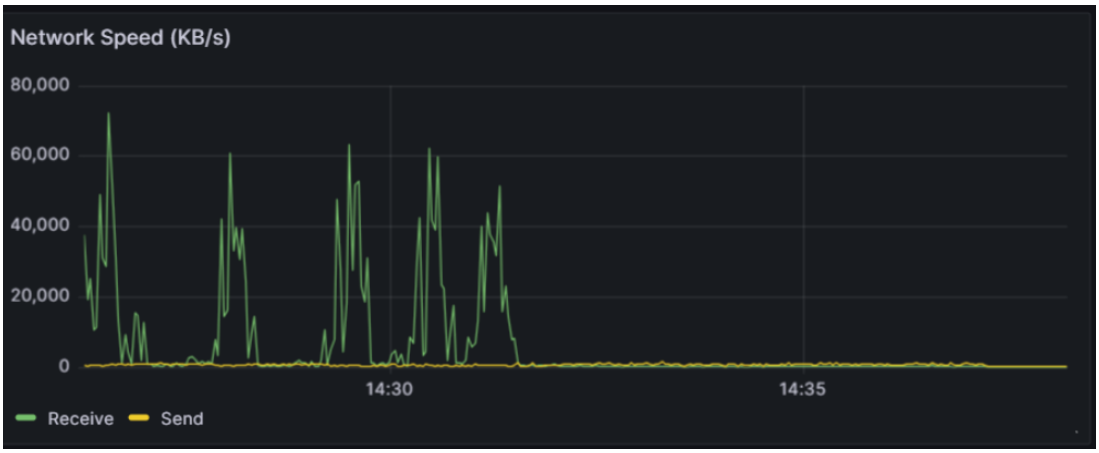


MD Core #2

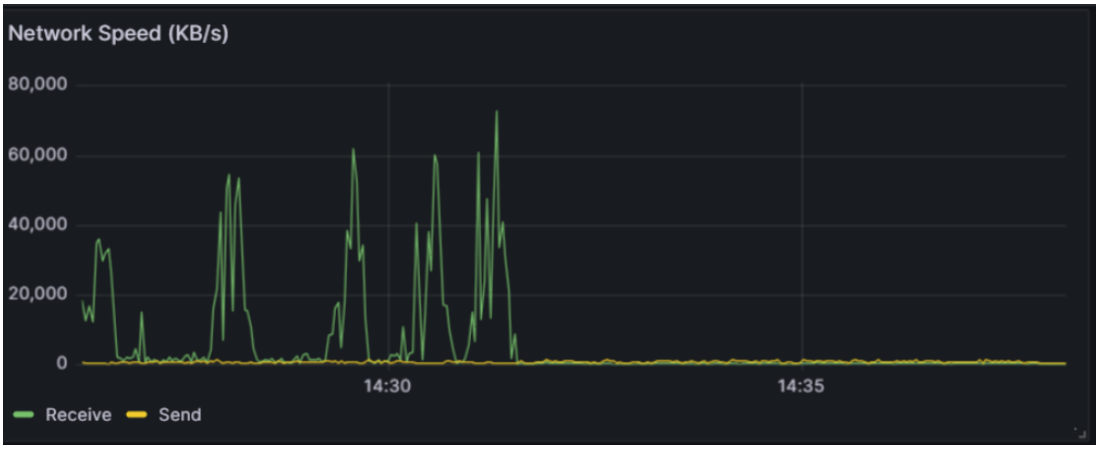


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2

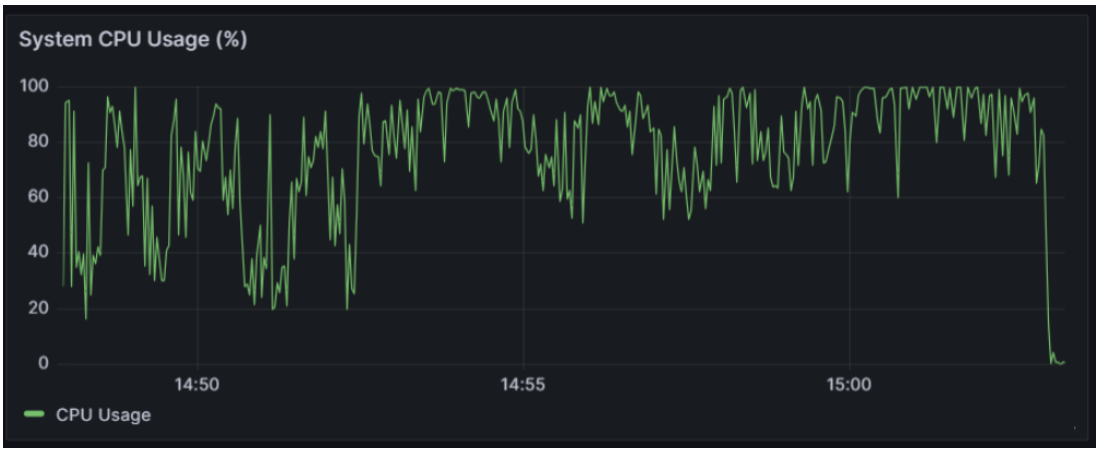


CPU usage

MD Core #1

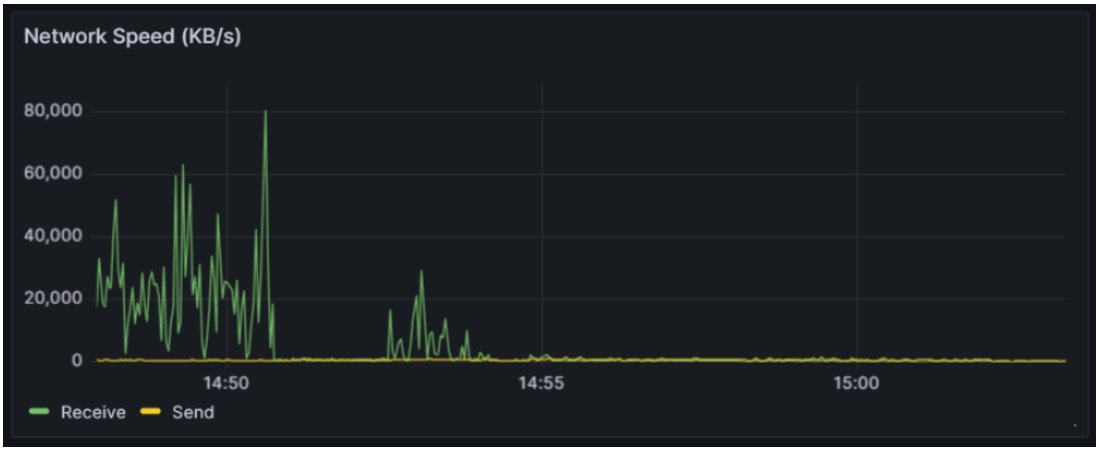


MD Core #2



Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

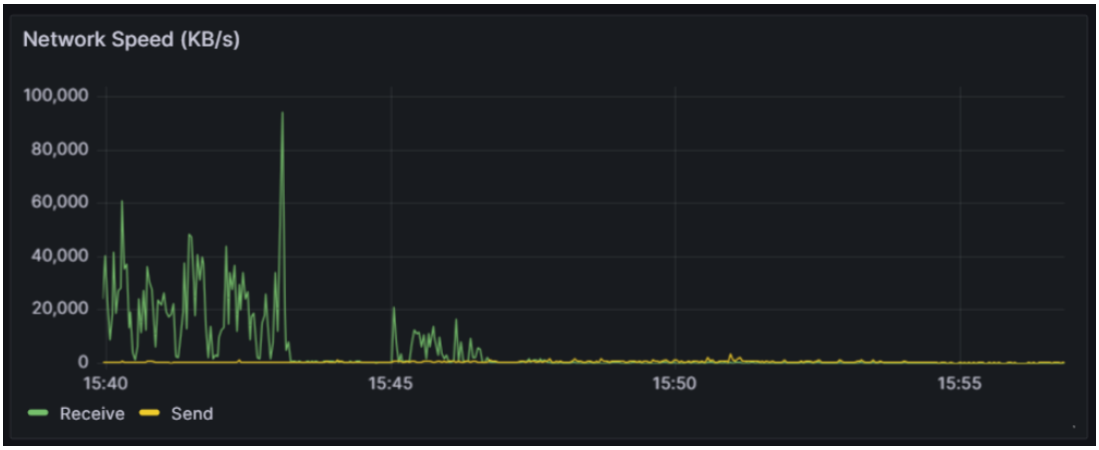


MD Core #2

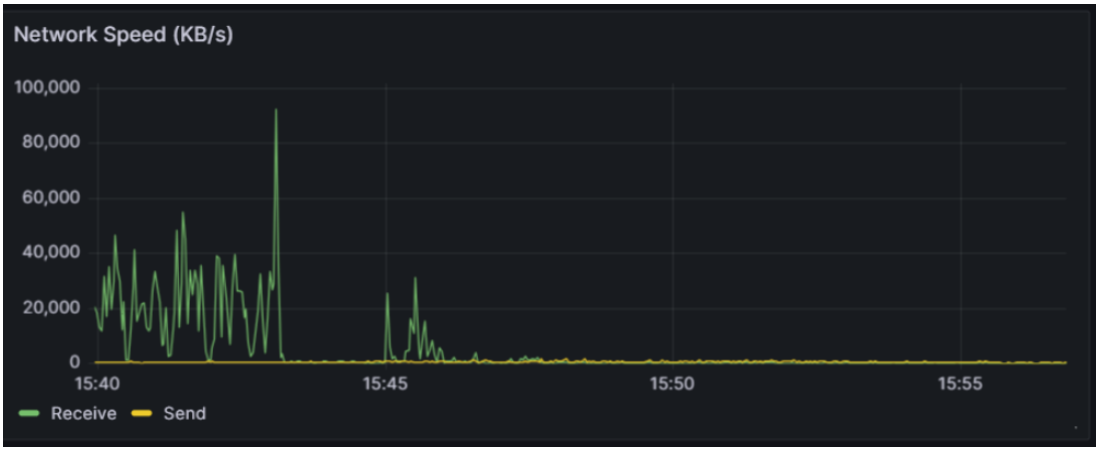


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1

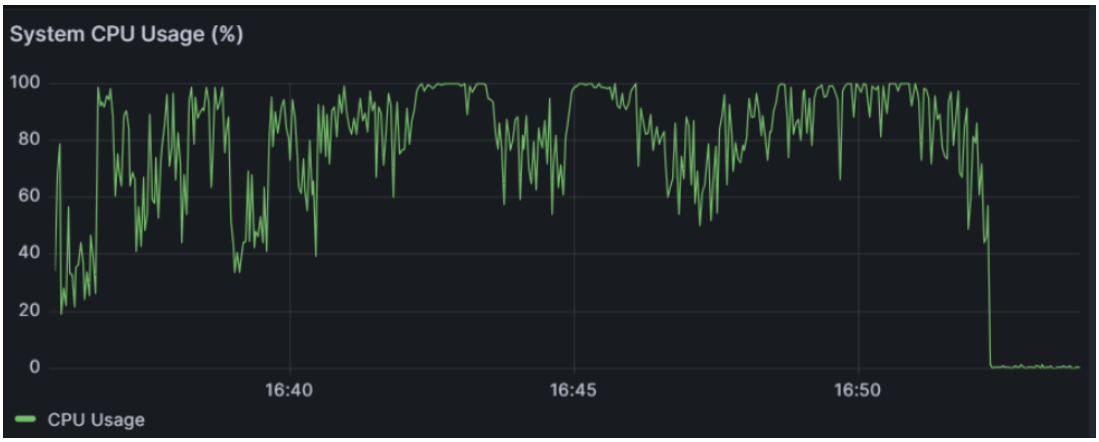


MD Core #2



CPU usage

MD Core #1

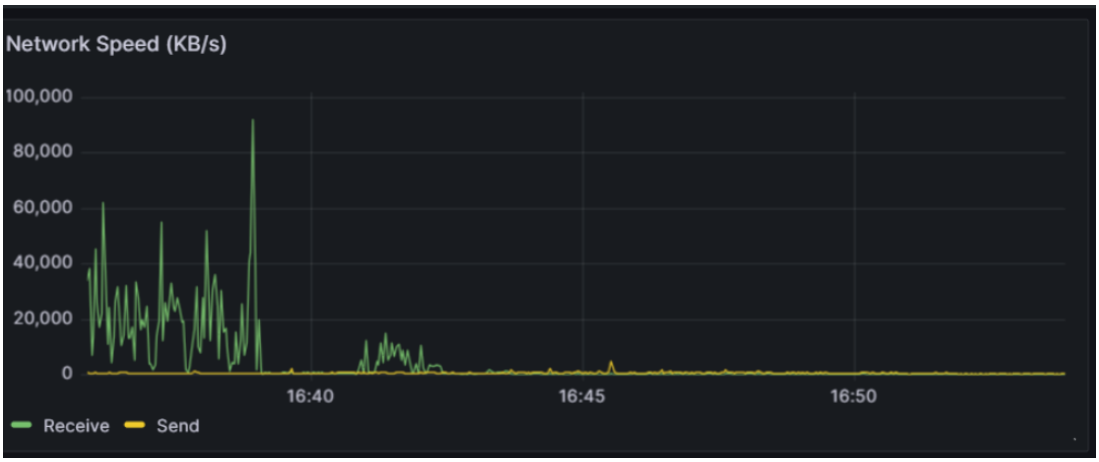


MD Core #2

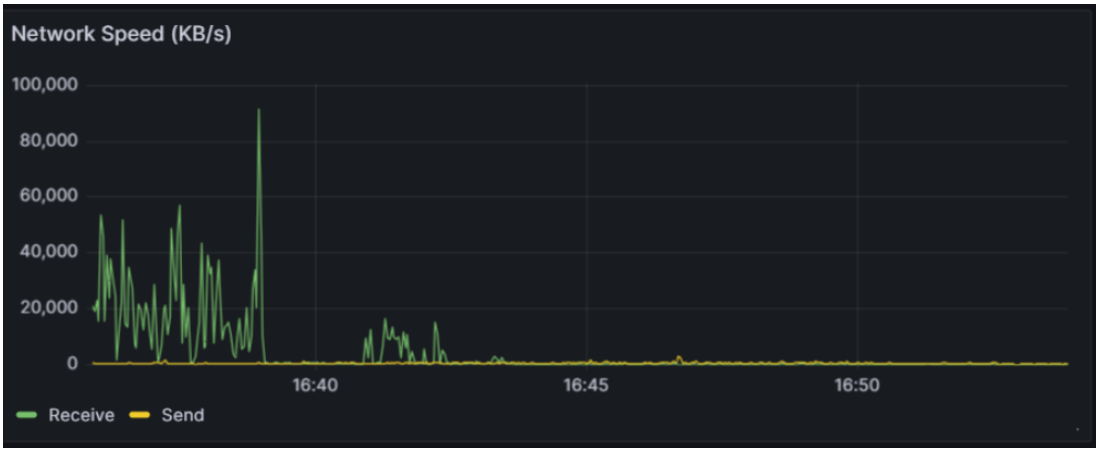


Network usage

MD Core #1



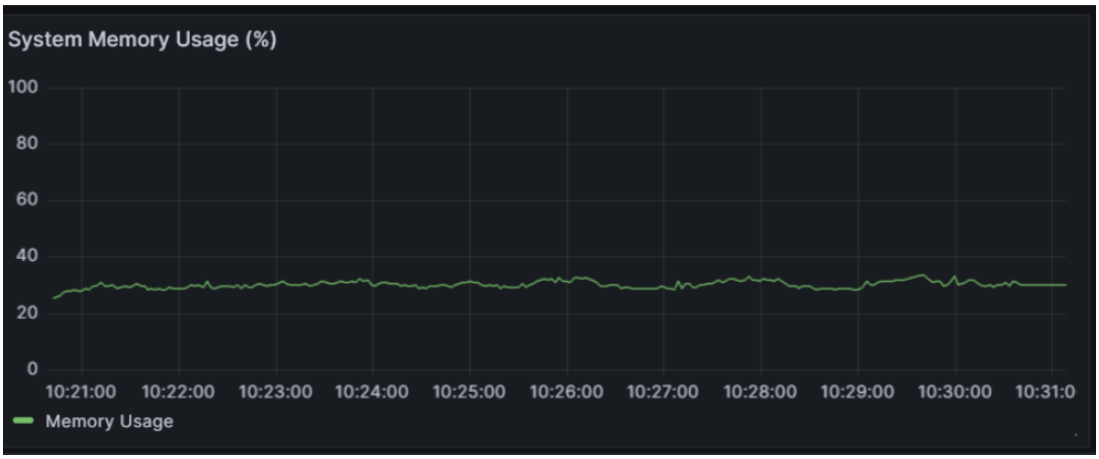
MD Core #2



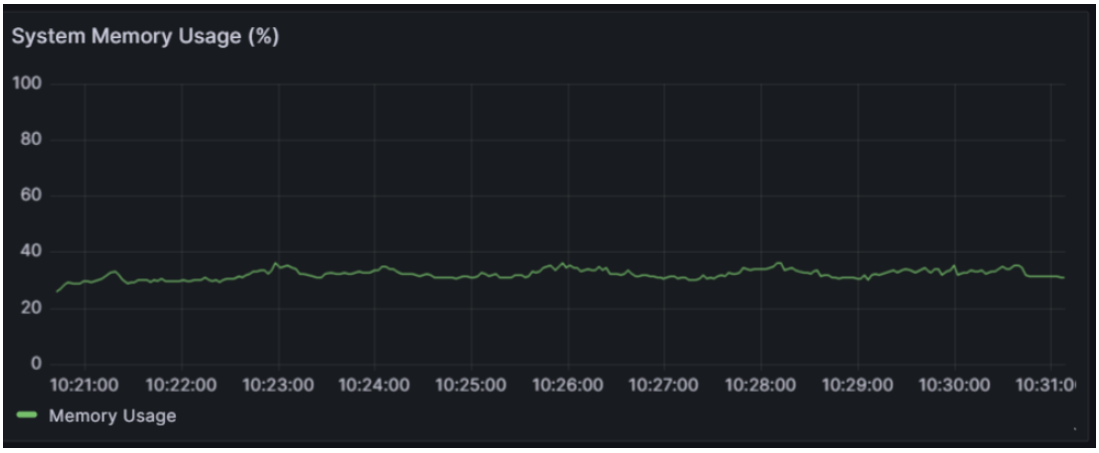
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

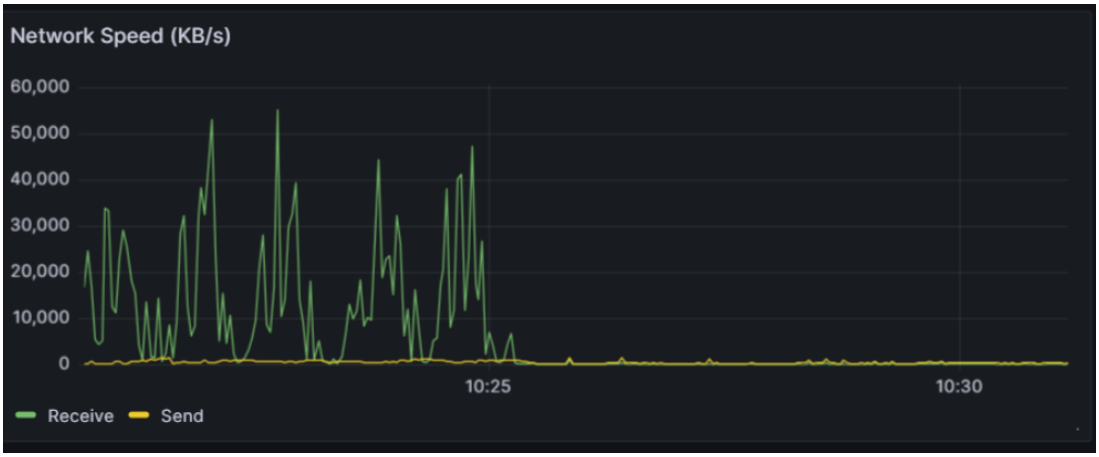


MD Core #2

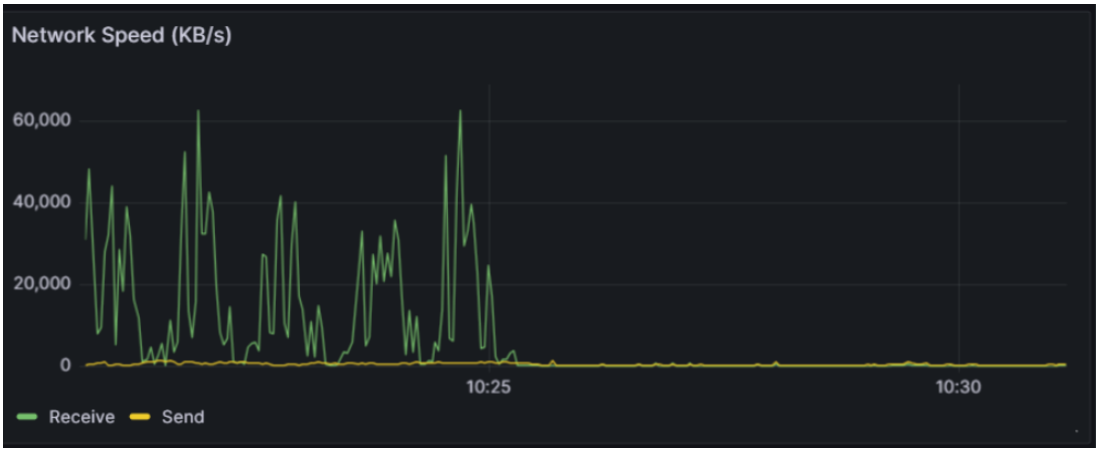


Network usage

MD Core #1



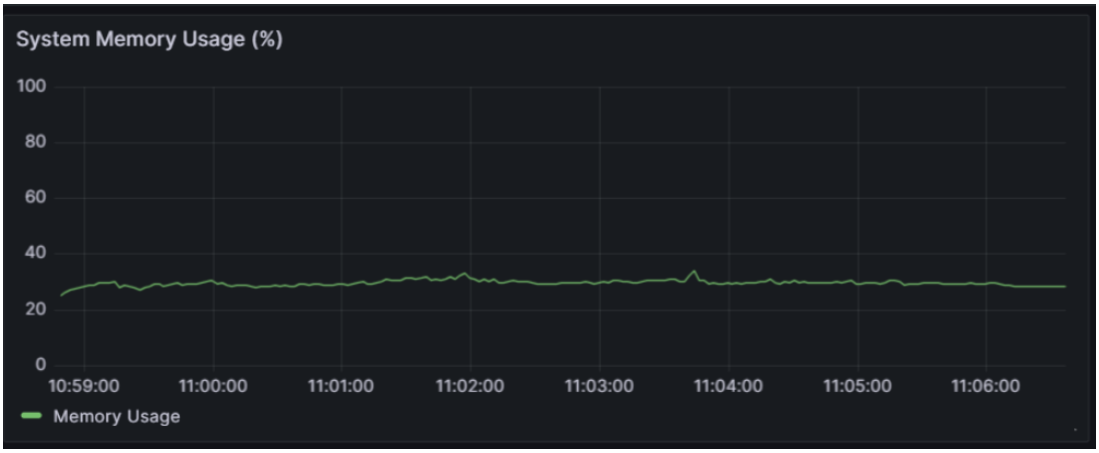
MD Core #2



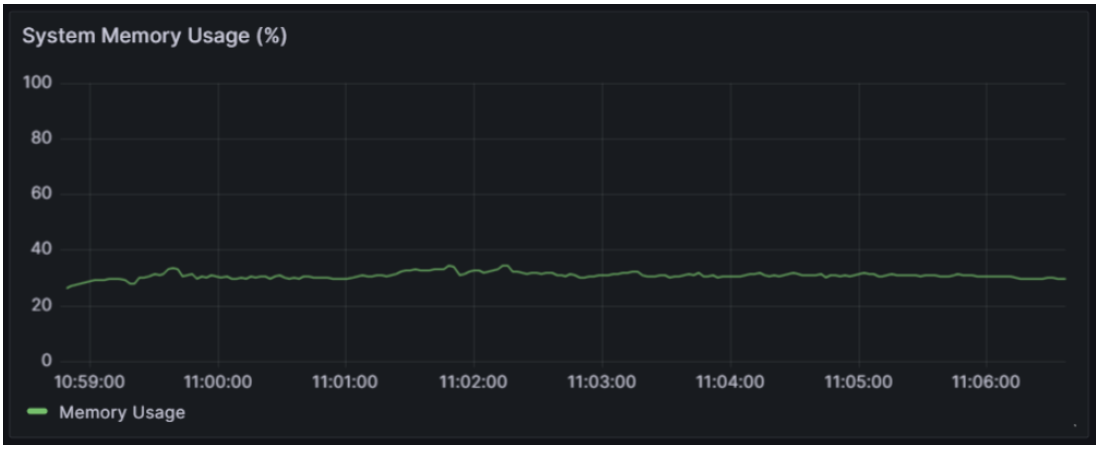
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

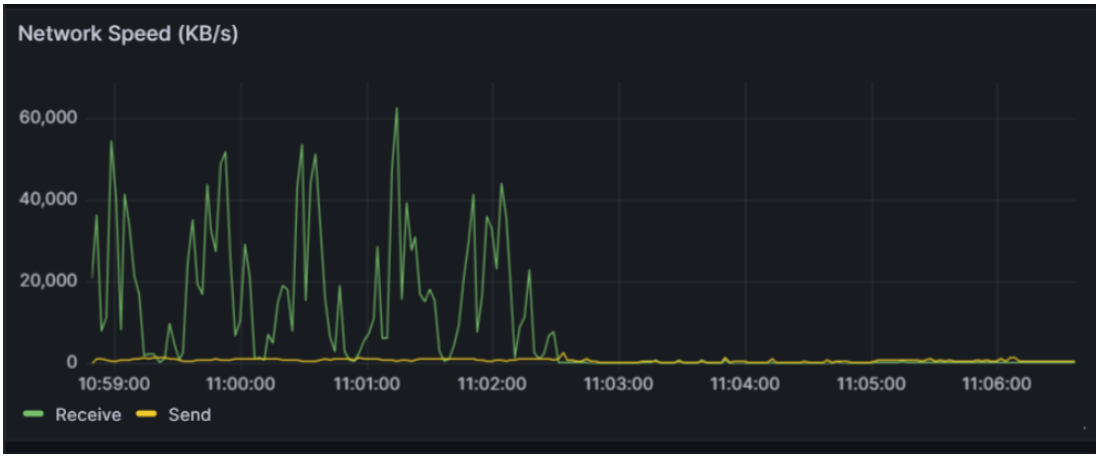


MD Core #2

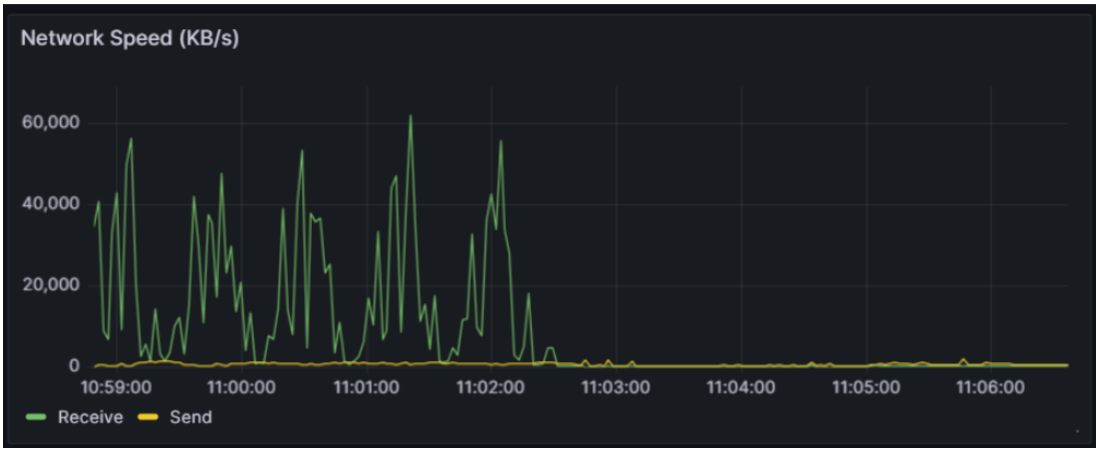


Network usage

MD Core #1



MD Core #2



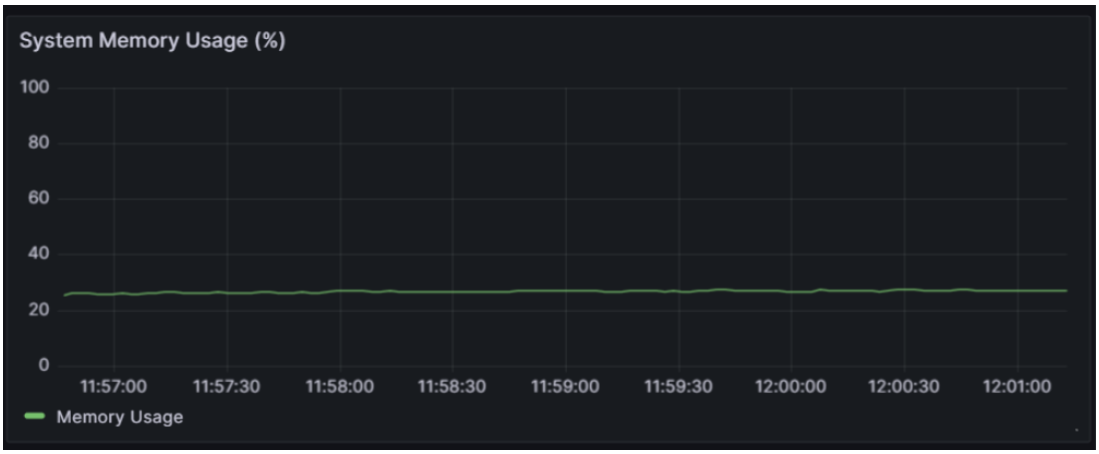
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

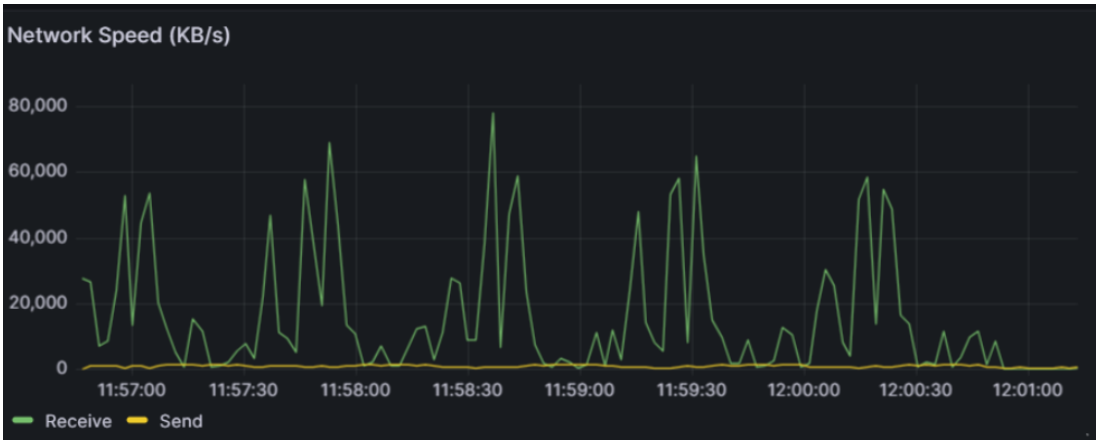


MD Core #2

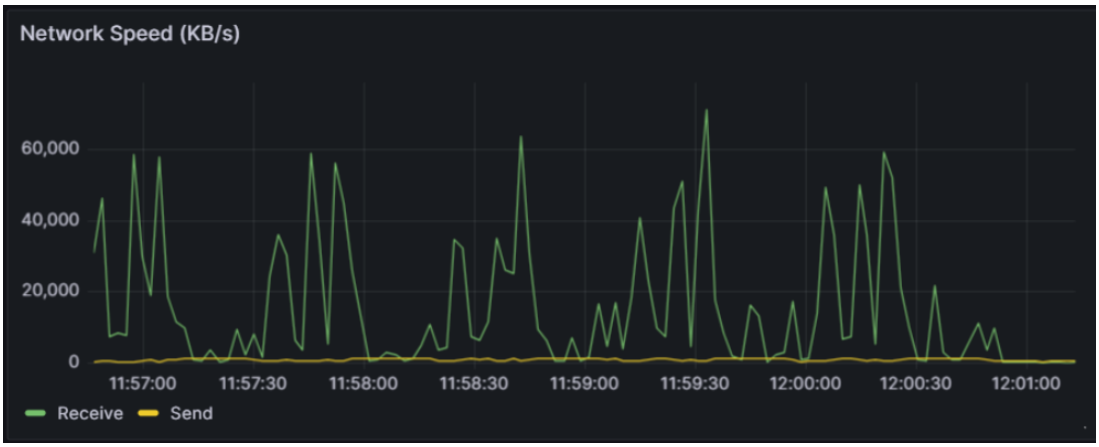


Network usage

MD Core #1



MD Core #2



Shared DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

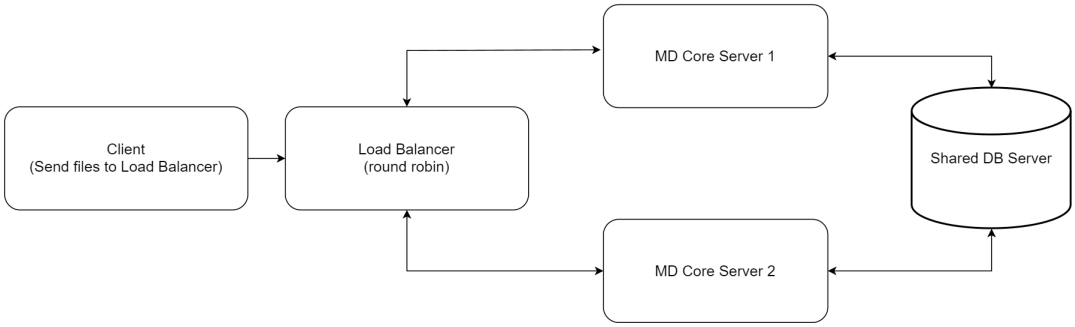
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV)	15.5	63.1	73.3	25.2	25.8	Receive: 4555.9 Send: 694.4	Receive: 4516.2 Send: 692.4	492,620.5	0.0
Deep CDR	9.70	50.7	51.2	31.9	32.5	Receive: 7062.9 Send: 623.2	Receive: 6653 Send: 654.1	786,189.6	0.0
Proactive DLP	7.2	35.2	35.0	27.9	27.5	Receive: 9047.4 Send: 802.4	Receive: 9859.8 Send: 787.5	1,055,110.8	0.0
Vulnerability	5.0	45.7	49.3	24.4	24.5	Receive: 13,585.7 Send: 966.7	Receive: 13,505.8 Send: 987.4	1,519,562.1	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV) + Deep CDR	19.9	79.1	77.6	34.0	34.2	Receive: 3785.9 Send: 573.8	Receive: 3663.1 Send: 577.7	382,716	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP	21.6	80.3	81.5	37.7	44.4	Receive: 3305.6 Send: 543.3	Receive: 3414.3 Send: 556.2	353,694.9	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	22.5	84.2	79.8	37.9	44.7	Receive: 3267.9 Send: 554.9	Receive: 3398.2 Send: 553.8	338,507.6	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

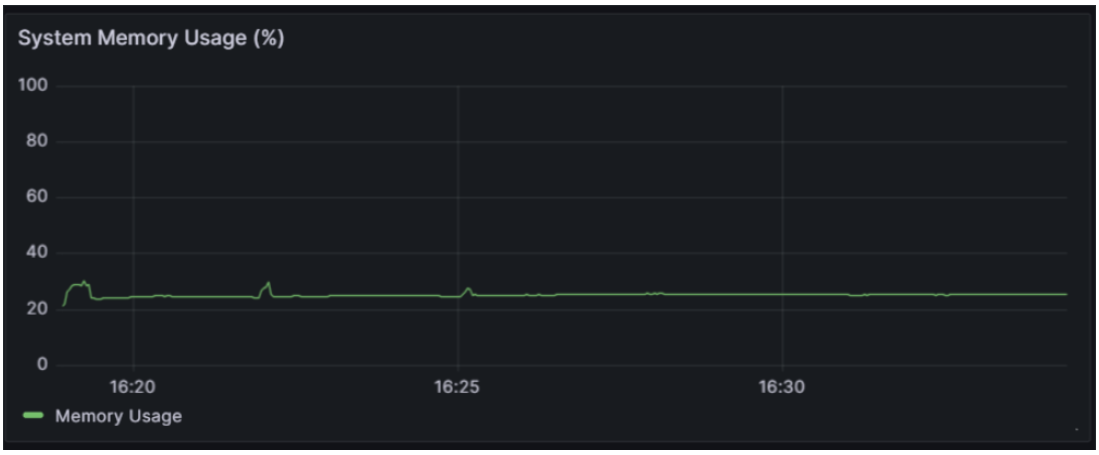
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

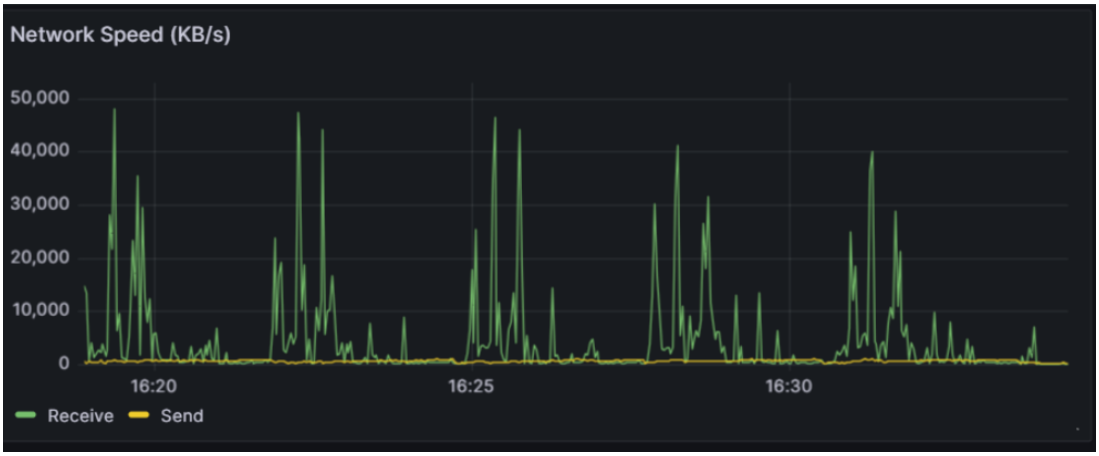


MD Core #2

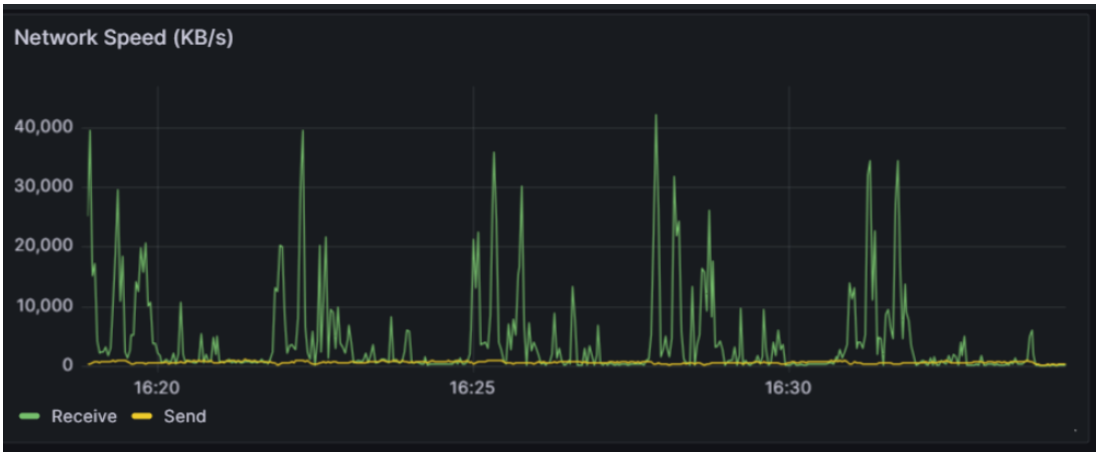


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

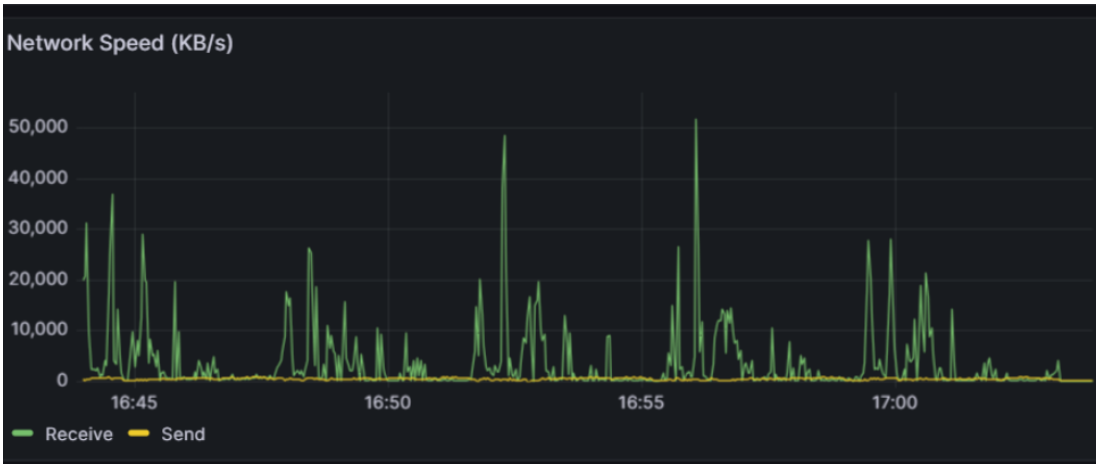


MD Core #2

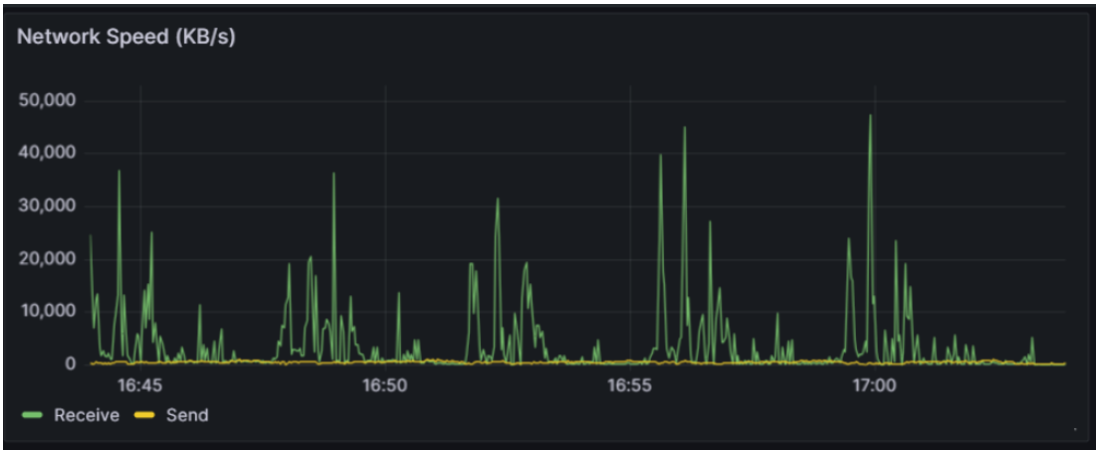


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1

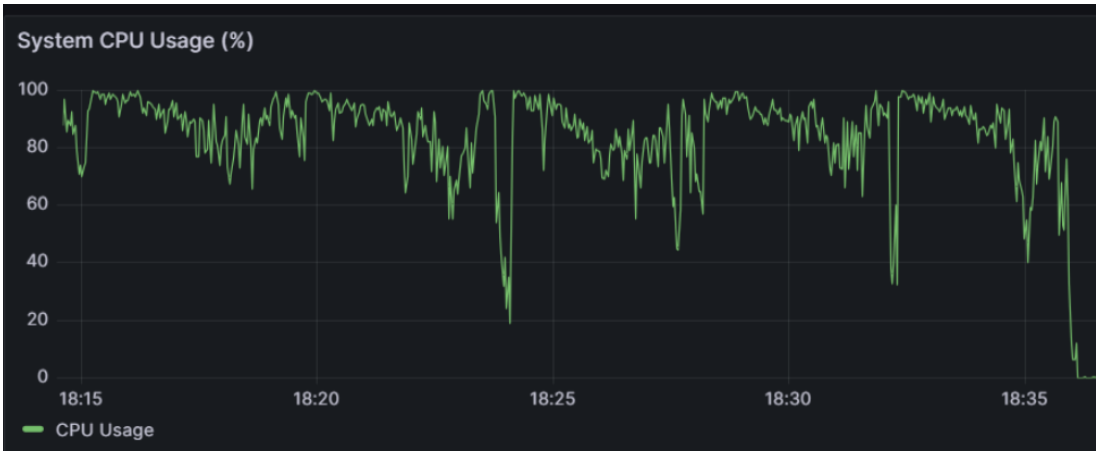


MD Core #2



CPU usage

MD Core #1

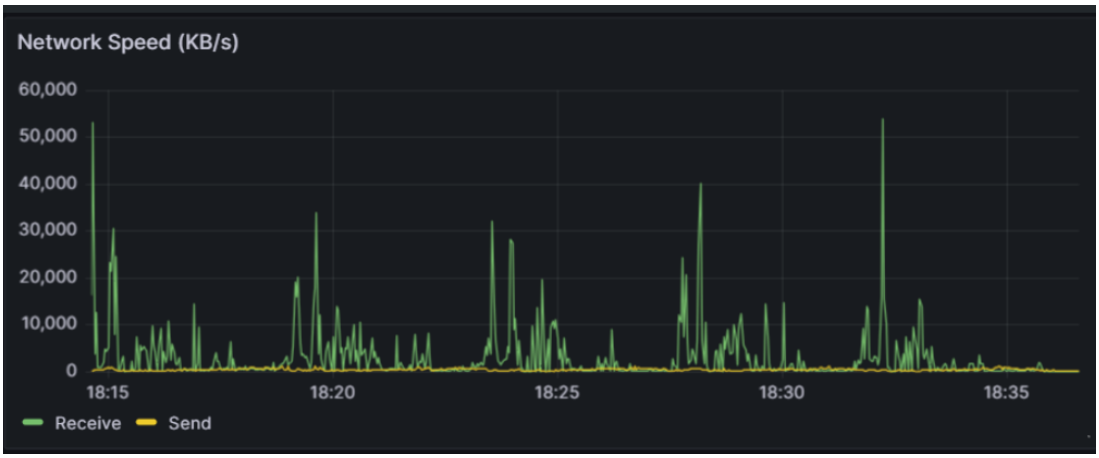


MD Core #2

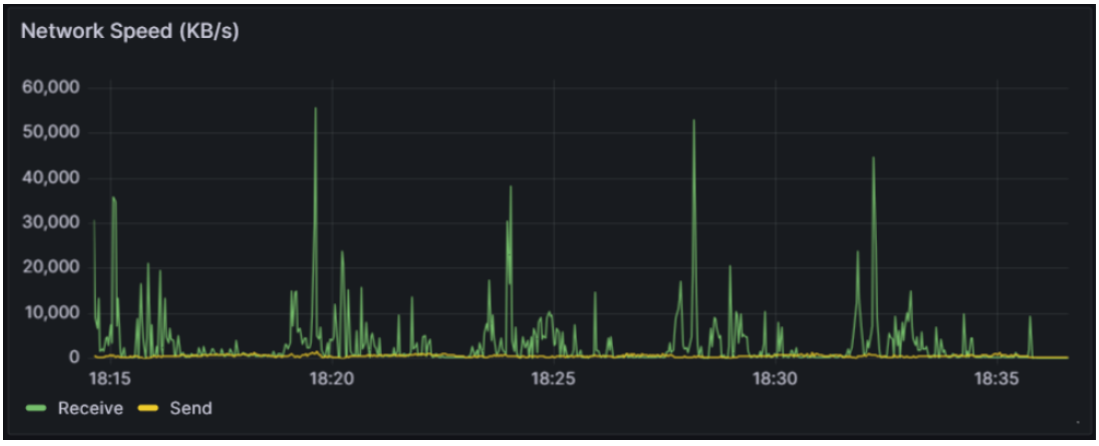


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

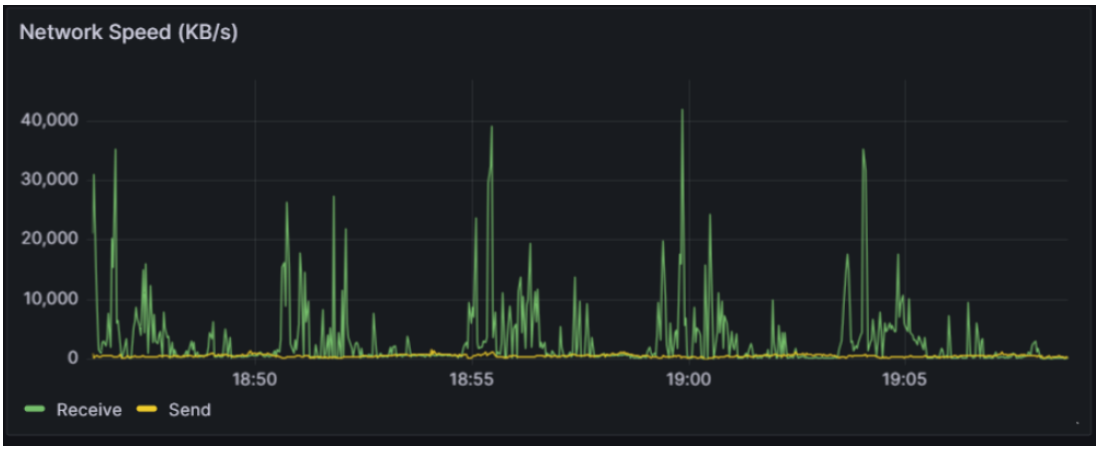


MD Core #2

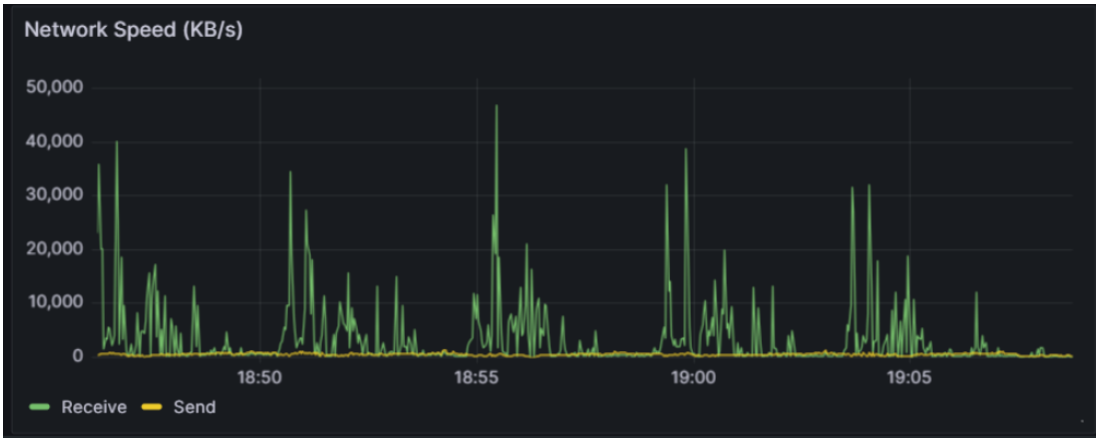


Network usage

MD Core #1



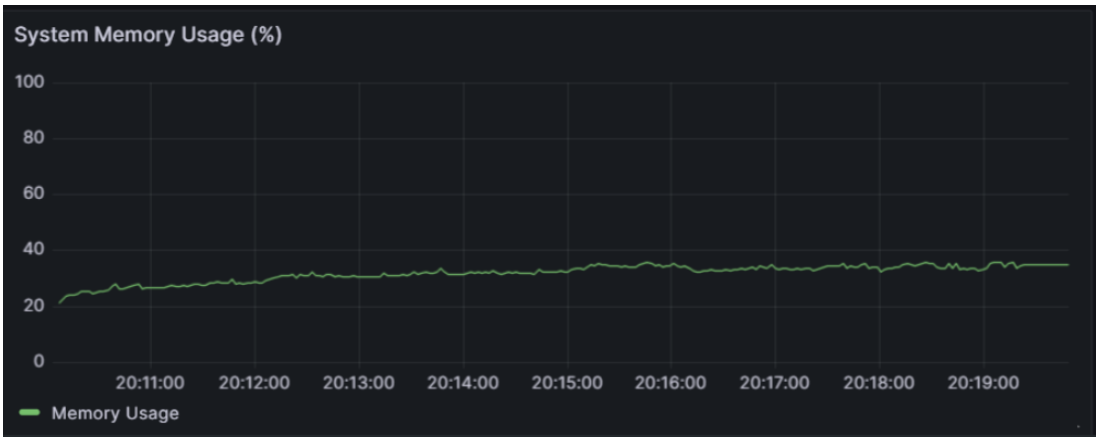
MD Core #2



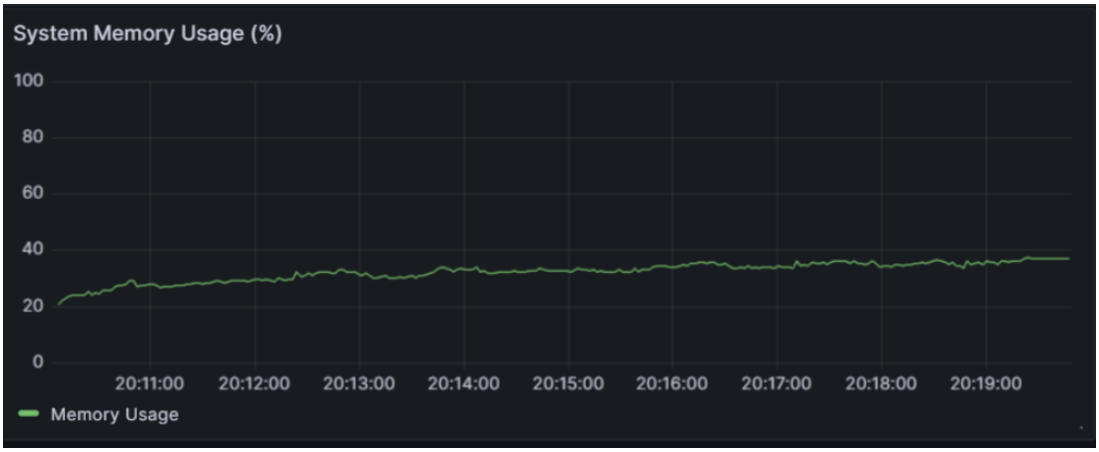
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1

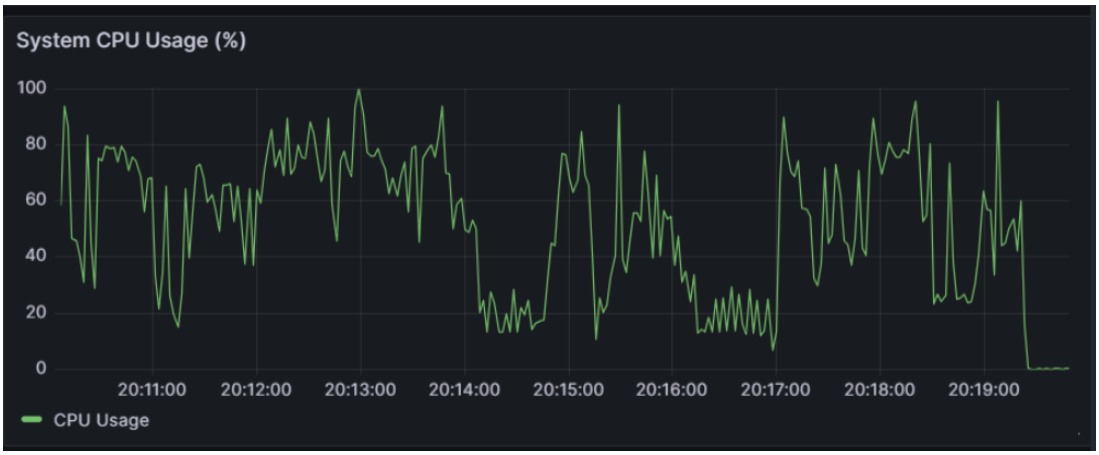


MD Core #2



CPU usage

MD Core #1

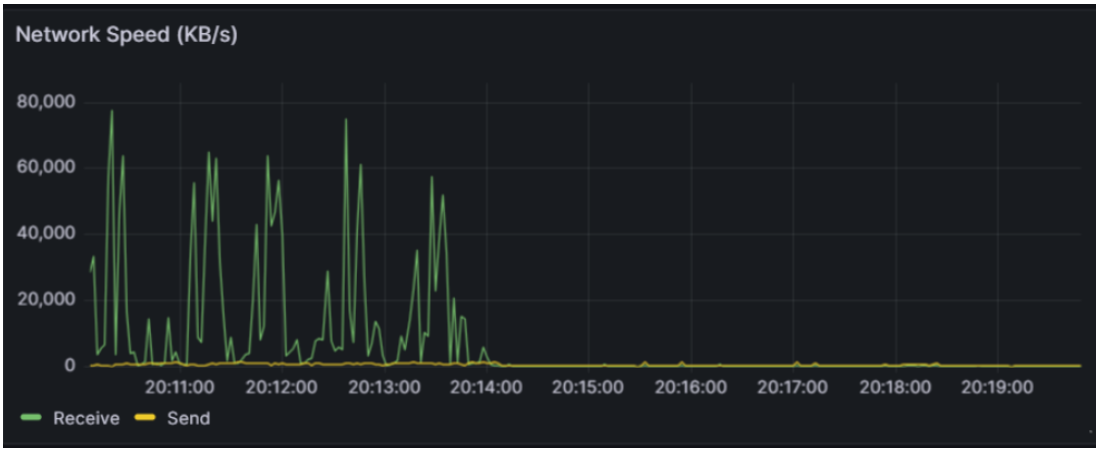


MD Core #2

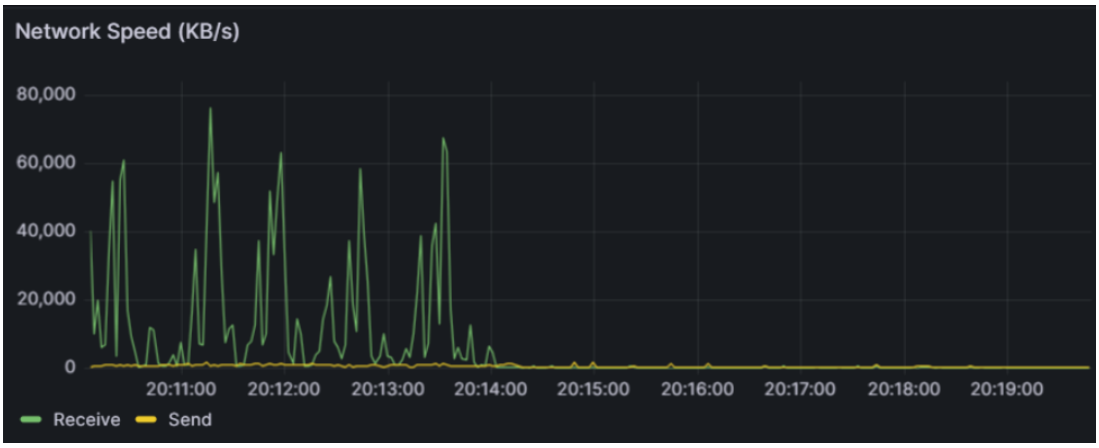


Network usage

MD Core #1



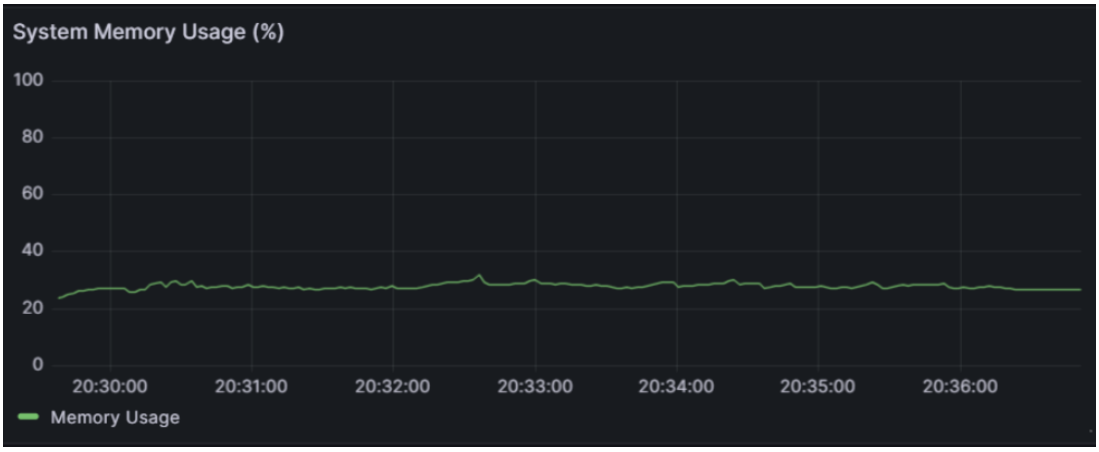
MD Core #2



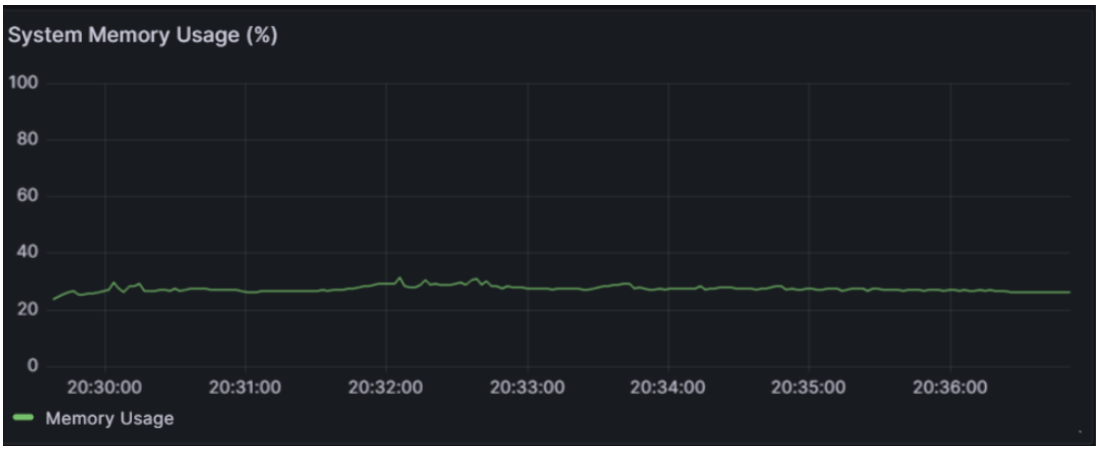
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1

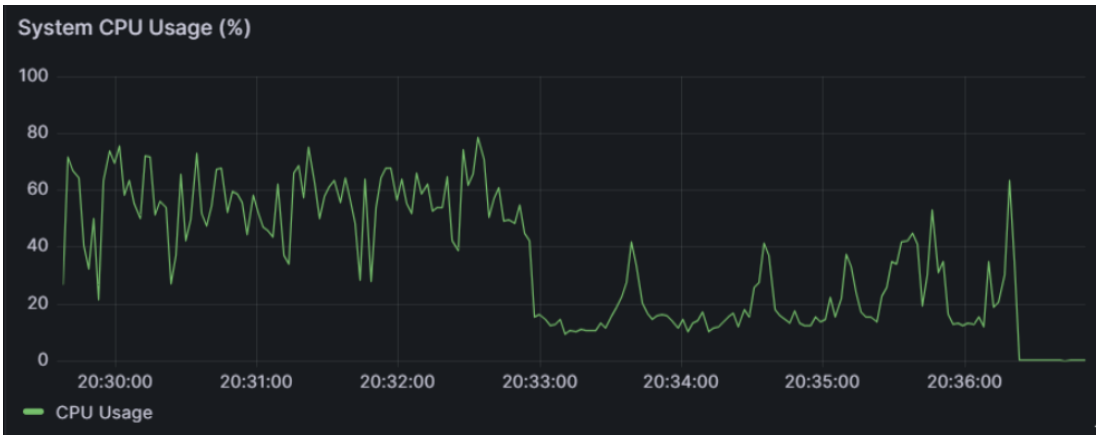


MD Core #2

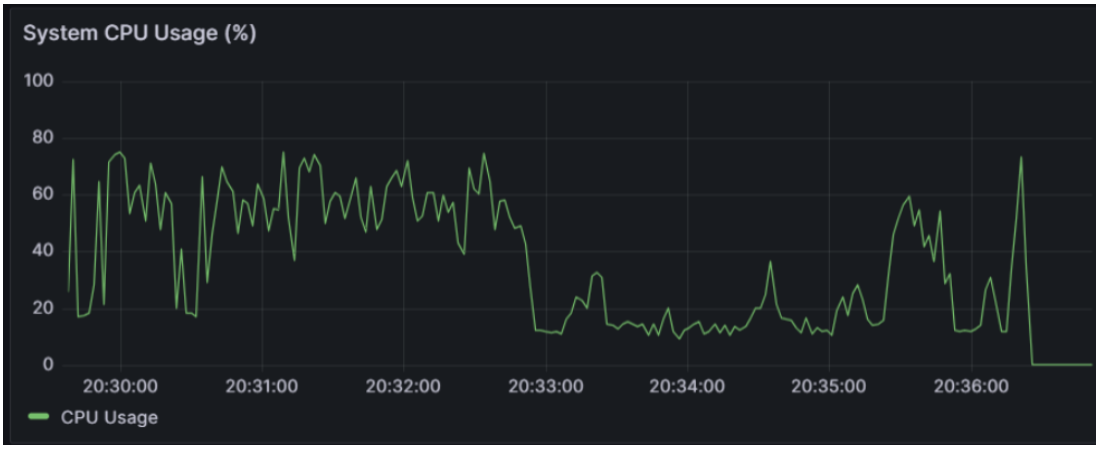


CPU usage

MD Core #1

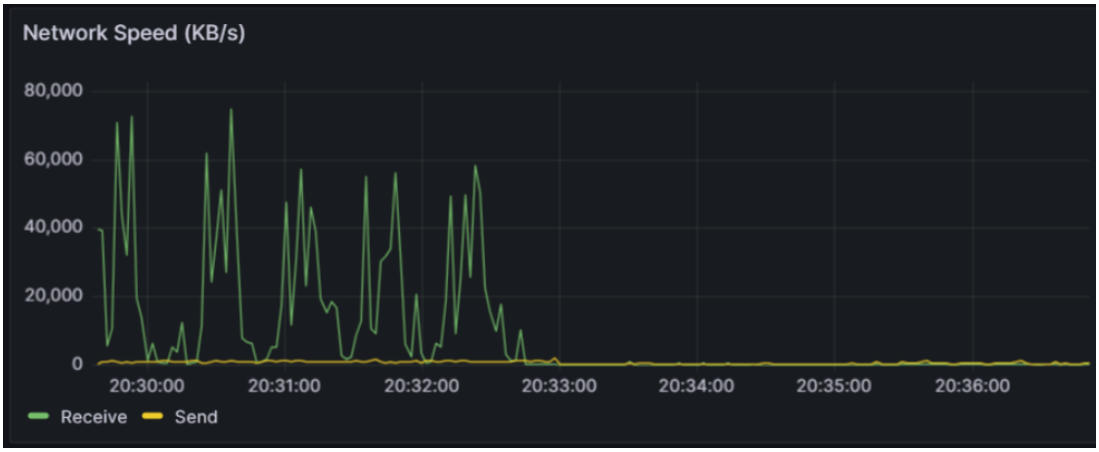


MD Core #2

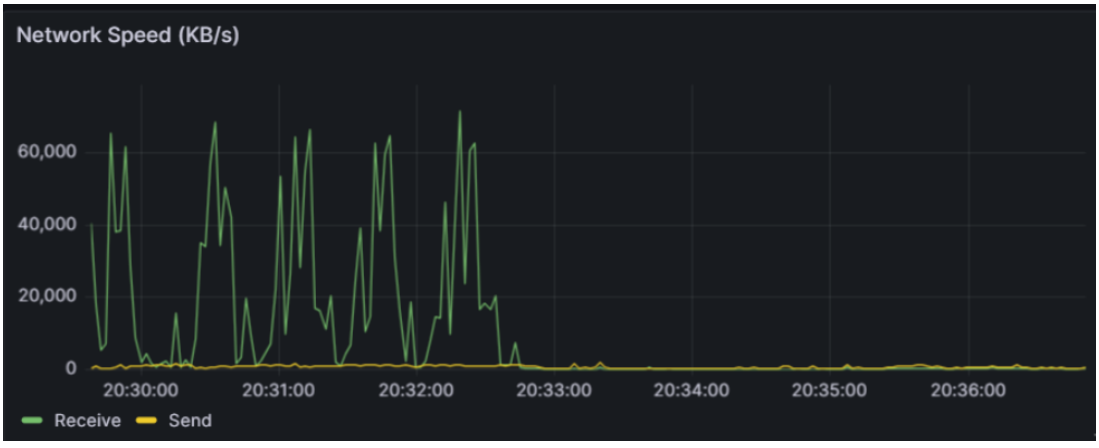


Network usage

MD Core #1



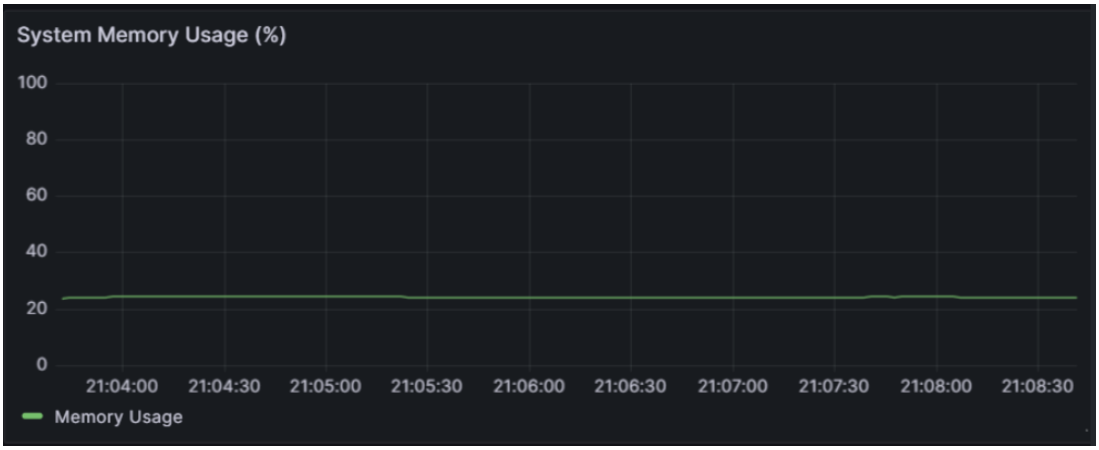
MD Core #2



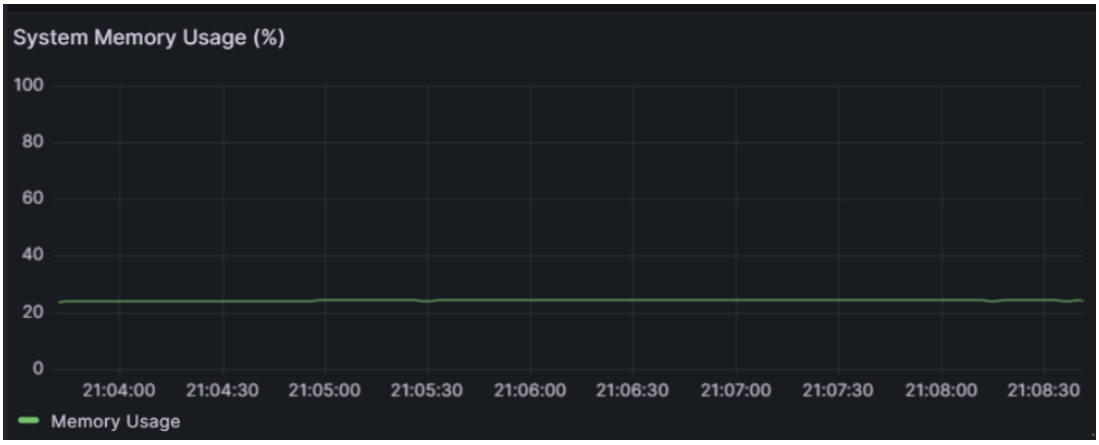
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

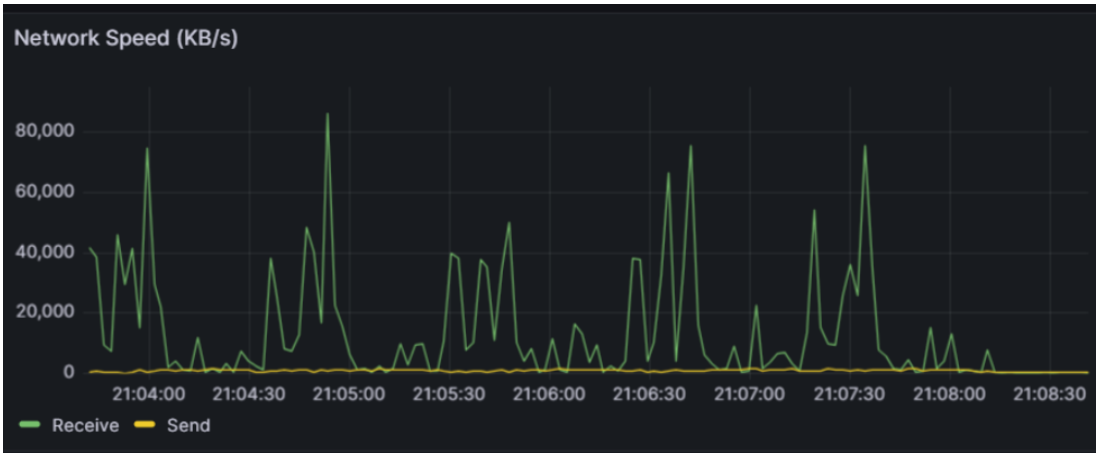


MD Core #2

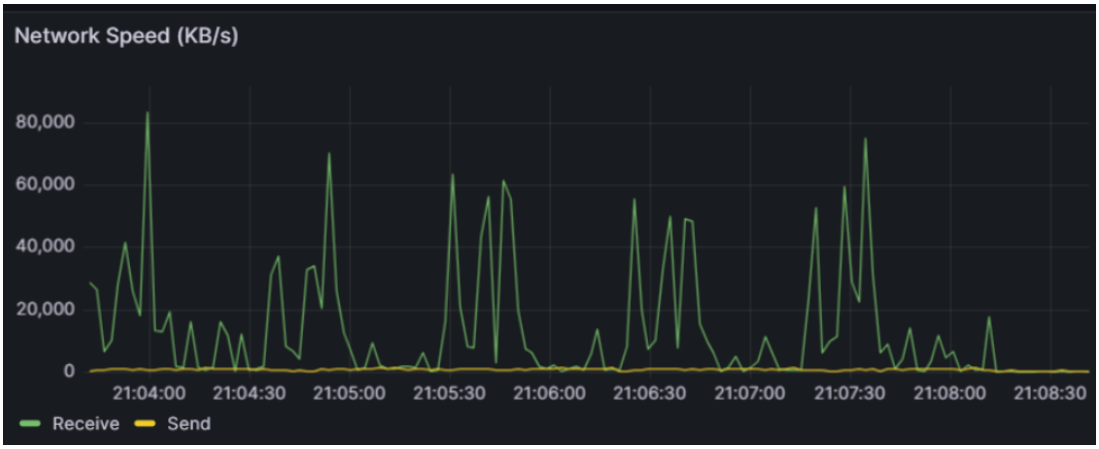


Network usage

MD Core #1



MD Core #2

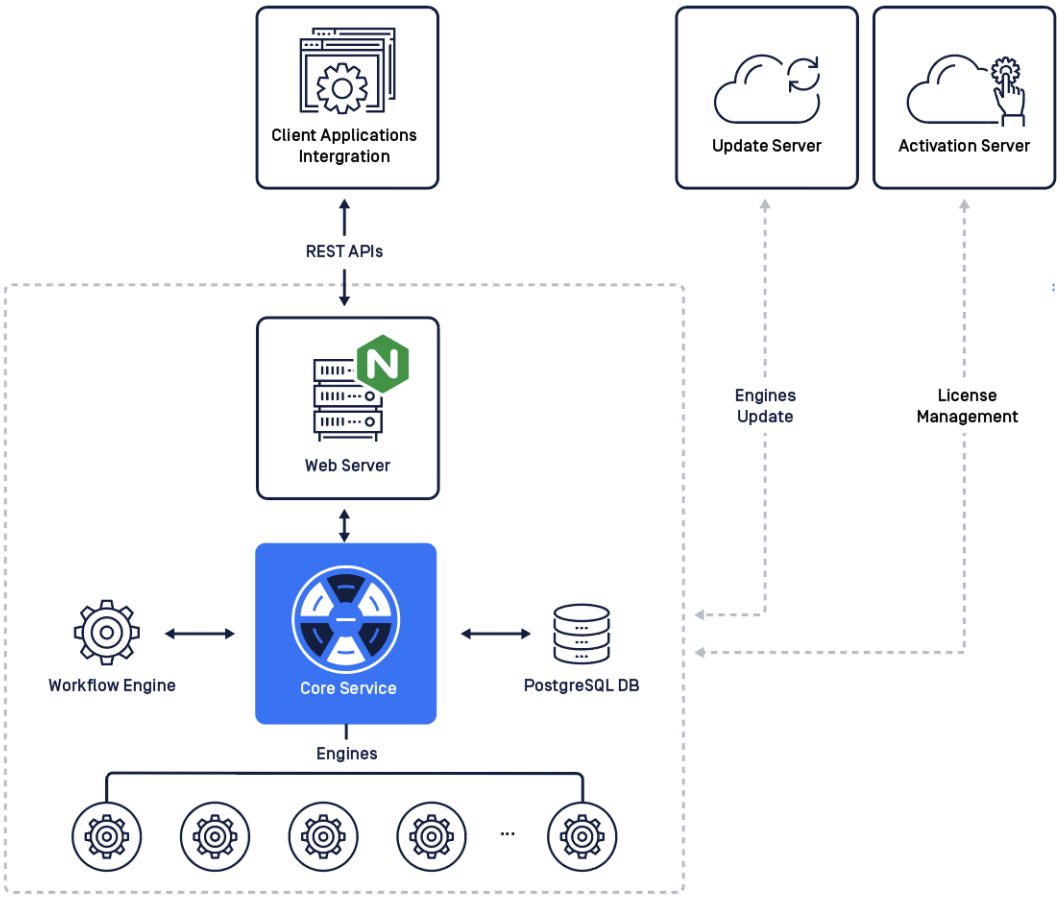

Network Speed (KB/s)
80,000
60,000
40,000
20,000
0
21:04:00
21:04:30
21:05:00
21:05:30
21:06:00
21:06:30
21:07:00
21:07:30
21:08:00
21:08:30
Receive
Send

MetaDefender Core Documentation

The users can consult this web page or, alternatively, they can download the manual in pdf format from the link below:

MetaDefender Core manual (SHA256:).

Recommended System Configuration



MetaDefender Core supports a wide range of deployment options, including air-gapped environments, providing the flexibility to choose the environment that best suits customers' requirements and infrastructure.

- Physical server.
- Virtualization platforms like VMware, Hyper-V and XenServer.
- Cloud Deployment.
- Container Deployment.

> **Note**
> Before installing MetaDefender Core, please refer to the **recommended minimum system configuration** listed below. The requirements are based on a standardized customer deployment and may require more or less resources to achieve the level of performance desired in your environment. Please note that the server specifications are built to allow a high volume daily processing. Please refer to Performance and Load Estimation for more details.

The recommendations below are for MetaDefender Core, API usage only. For any other use cases, please consult the user guide of the licensed products for accurate recommendations.

For certain use cases these might be adjusted and customized on specific needs and SLAs. We highly recommend to engage our ProServ team to assist in fine tuning MetaDefender and get the maximum performance out of your systems.

Microsoft Windows Deployments

[Windows] Supported Operating Systems

Only support 64-bit platforms.

- Windows 10, Windows 11 23H2, 11 IoT (22H2) only for End-User systems (e.g. Kiosk deployments)
  - Based on Microsoft License Terms, you're not allowed to use Windows Desktop as a server, therefore, MetaDefender Core can't be used on these systems for server use cases.
- Windows Server 2016, 2019, 2022

> **Disclaimer**
> End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

> **Plan for legacy OS support**
> OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:
>
> - Ubuntu 20.04 after **April 2025**.
> - Amazon Linux 2 after **June 2025**.
> - Windows 10 after **October 2025**.
>
> If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

[Windows] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 8	8	8 GB	16 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 12	16	16 GB	24 GB + 50 GB + 100 GB
MetaDefender Core 16	16	16 GB	32 GB + 50 GB + 100 GB
MetaDefender Core 20	32	16 GB	40 GB + 50 GB + 100 GB
MetaDefender Core MAX	32	32 GB	120 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core is installed.
- For a better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.
- SBOM processing requires more disk space, please allow at least 15GB free disk for the engine to work properly.

[Windows] Third Party Dependencies

- .NET framework 4.5 or above
- Microsoft Visual C++ Redistributable for Visual Studio 2015-2022 (version 14.38 or higher); x64 version.
- PostgreSQL: Only applicable when using a pre-installed PostgreSQL server running remotely.
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Some engines also have dependencies as described below:

Engine Name	Dependency
Deep CDR	Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 5.8.0 or above)
ESET	MetaDefender Core v4 temporary directory should have more than 200MB free disk space
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
Proactive DLP	• Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 2.0 or above) • .NET 6 dependencies • CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
Scrutiny	Microsoft Visual C++ 2015 - 2019 Redistributable x86 and x64
Systweak	.NET runtime 6.0, 7.0 or 8.0
Bkav Pro	Microsoft Visual C++ 2015 - 2022 Redistributable x64

[Windows] Installation Details

MetaDefender Core on Windows uses "C:\Program Files\OPSWAT" folder for storing resources or the installation directory.

MetaDefender will use its resources folder to store temp files as part of the analysis. **It's recommended to exclude this folder from real-time protection monitoring**. Knowledge base:

- How to set up exclusions on anti-malware software to prevent disruption
- Mitigating scan failure triggered by real-time protection service
- Can local AVs interrupt ongoing scans?

Linux Deployments

[Linux] Supported Operating Systems

Only support 64-bit platforms.

- Red Hat Enterprise Linux 8, 9
- Rocky Linux 9
- Debian 11, 12
- Ubuntu 20.04, 22.04
- Amazon Linux 2 is supported via Amazon Machine Image (AMI)

Plan for legacy OS support

OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:

- Ubuntu 20.04 after **April 2025**.
- Amazon Linux 2 after **June 2025**.
- Windows 10 after **October 2025**.

If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

For Hardended Linux OS

Some AV engines could be failed to initialize and run on MetaDefender Core (e.g. ESET, Kaspersky) due to execution permission required by them on `/var/tmp/ometascan` folder.

Please follow steps at Why have the ESET and Kaspersky scan engines failed to initialize on the hardened Linux OS? for remediation details.

Disclaimer

End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

[Linux] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 5	4	4 GB	10 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 10	8	8 GB	20 GB + 50 GB + 100 GB
MetaDefender Core MAX	16	16 GB	40 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core resides on.
- For better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.

[Linux] Third Party Dependencies

Dependencies list:

- openssl
- grep
- lib32stdc++6 (>= 4.5)
- libc6-i386 (>= 2.10)
- procps
- zlib1g
- libcurl3 (>= 7.19.7)
- libcurl4
- postgresql:
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Not all the above dependencies will need to be installed, it is dependent on different Unix distro and version

Some engines also have dependencies as described below:

Engine name	Dependency
Archive engine	• To extract password protected MS Office files, you need to install .NET 6 dependencies, libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • Rocky Linux 9.x/Red Hat 9.x: libnsl
Deep CDR	• Rocky Linux 9.x: libgomp, ncurses-compat-libs, libnsl • CentOS: libgomp, ncurses-compat-libs • Red Hat 8.x: libgomp, ncurses-compat-libs, libnsl • Red Hat 9.x: libgomp, ncurses-compat-libs, libnsl, compat-openssl11 • Ubuntu/Debian: libgomp1, libncurses5 • libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • .NET 6 dependencies
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
FileType engine	Rocky Linux 9.x/Red Hat 9.x: libnsl
Proactive DLP	libgdiplus, mscorefonts .NET 6 dependencies CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
RocketCyber	libgomp

Note: Some dependencies may need to be installed from external repositories, not from the OS default repositories.

[Linux] Installation Details

MetaDefender Core default installation path is using /var folder for storing resources:

- `/var/lib/ometascan` : installation folder with all its resources (database, DLP processed / CDR sanitized / quarantined file storage, engine package and definition updates).
- `/var/log/ometascan` : application logs.
- `/var/run/ometascan` : runtime folder for Core related processes.
- `/var/lib/ometascan` : all successful deployed engines data files.
- `/usr/lib/ometascan` : all shared libraries required by MetaDefender Core, bundled PostgreSQL server related binaries and libraries.
- `/var/tmp/ometascan/resources` : all temporary processing files for MetaDefender Core to process.

Storage Usage Information

Based on the configuration, MetaDefender Core will need additional disk space to store analysis/result data, quarantined files and temporary files:

	Data Storage
Analysis Reports (Processing History)	Approximately 4.5GB is required for every 1M analysis reports stored. • 100 GB is recommended as a baseline. Database storage requirements are affected by Data Retention settings, engine settings and scanning load.
Quarantined Files	Depends on the customers' dataset and configuration
Sanitized Files	Depends on the customers' dataset and configuration
Temporary processing files	Depends on the customers' dataset and configuration • A 1GB non-archive file requires 1GB free disk space at least. • A 1GB archive file requires (1 * 2)GB free disk space at least. • 50GB is a baseline and should be adjusted based on observation of the solution within the environment, with consideration to peak times and overall file makeup (e.g. complexity, size, archives).
Note	Options are available to control how large and/or complex files are handled to limit the amount of system resources required. See Workflow Configuration Template for more information.

Custom Engines

The recommendations above are specific for MetaDefender pre-packaged bundles.

However for additional Custom Engines, please review the Knowledge base to review additional requirements (if any) for the selected engine.

Browser Requirements for the MetaDefender Core Management Console

One of the following browsers is suggested to view the MetaDefender Core Management Console:

- Microsoft Edge
- Chrome
- Firefox
- Safari

Chrome, Firefox, Safari and Microsoft Edge browsers are tested with the latest available version at the time of release.

Network Bandwidth

In online environment where MetaDefender Core is configured to download engine update packages directly from OPSWAT update server source, then using gigabit network speed (125 MB/sec) is highly recommended to avoid potential network related issue while downloading files, or at least 100 Mbps (12.5 MB/sec) network speed.

Allowlisting URLs

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts':

- https://activation.dl.opswat.com - for license online activation
- https://update.dl.opswat.com - for engines and databases download

Note: *IP address-based allowlisting on your firewall might fail after some time since OPSWAT uses CDN (AWS Cloudfront) to faster delivery updates over the world, and IP address of edge servers might change over time.*

Depending on your location, connections with OPSWAT cloud-based servers be forwarded to specific AWS Cloudfront's edge locations, each is assigned with its own unique IP address ranges. You might also need to allow correct AWS Cloudfront's corresponding IP address ranges by referring to https://ip-ranges.amazonaws.com/ip-ranges.json

Warning

Even the OPSWAT update servers host updates for all of the available engines we support, sometimes custom engines might try to connect to their own cloud for updates, but this can be disabled in firewall and they will be updated just from OPSWAT cloud servers instead.

Differences Between MetaDefender Core v4 and v5

We are very excited to inform you about the release of version 5.0.0 of our MetaDefender Core!

The new version is designed to enhance the product performance, user experience, as well as to provide more flexibility, security, and scalability for our customers.

At the same time, we are also committed to keeping backward compatibility for whom running MetaDefender Core v4, and avoid the need for updating your client integration to move forward with MetaDefender Core v5.

v4 REST API Integration Supported
All public REST API endpoints available on MetaDefender Core v4 will be continously supported on v5.

v4 Product Major Features Supported
All product major features available on MetaDefender Core 4.21.2 (the latest one on v4) will be continously supported on v5.

Seamless Product Upgrade
Upgrading from MetaDefender Core v4 to v5 will be supported and covered with data migration without additional manual steps from users.

Licensing information will be also migrated.

Below are some highlights of the first v5 release, MetaDefender Core 5.0.0:

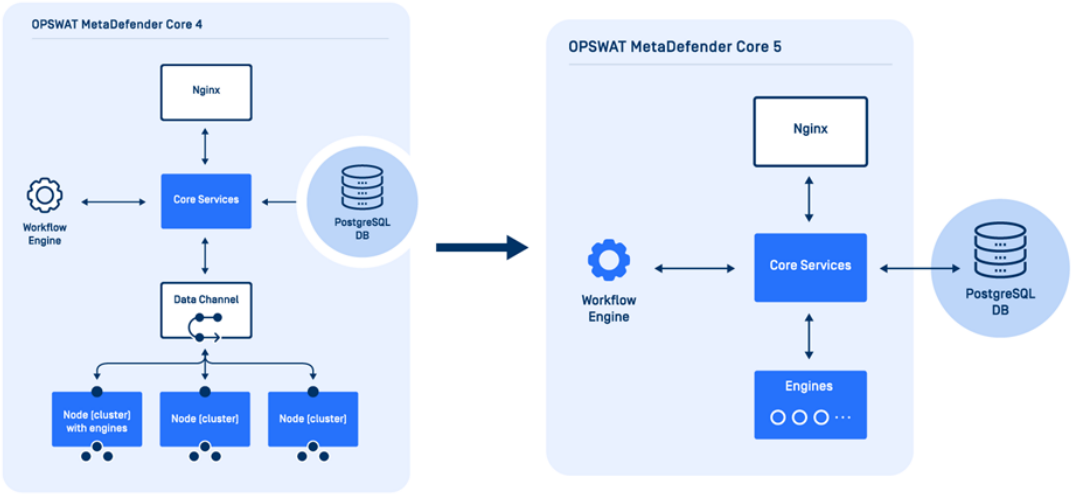
New product architecture

This version introduces a simpler product architecture that speeds up integration and automation, file processing flow, and immensely enhance the product performance and efficiency.

Earlier versions utilized MetaDefender Node concept, designed to support scaling of the scanning infrastructure by distributing scan requests among several different processing Nodes. Product licensing was based on number of Node instances. The benefit of having such a distributed infrastructure is that based on Node loads, Metadefender Core server can always choose the most appropriate Node to assign a new scan task to. However, having nodes in the v4 architecture caused more complexity and possible latency in the processing flow and entailed maintenance cost.

In this release, we retired multi-Node deployment and restructured MetaDefender Core's backend. This alteration significantly improves file processing speed and product efficiency, as well as eliminates extra cost for maintaining node services. Users now can simply install additional MetaDefender Core instances for scalability requirement,

as well as configure their own load balancer to coordinate tasks among multiple MetaDefender Core instances. Accordingly, product licensing is now based on number of the product instances.



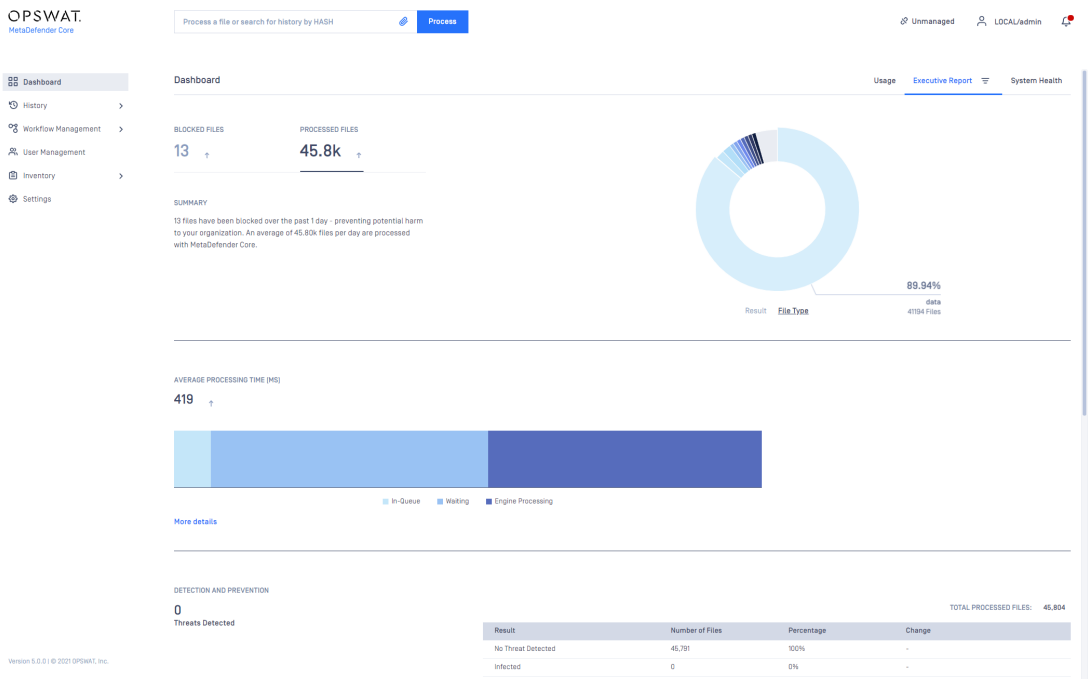
Please be informed that

- All existing administrative APIs related to "Node" on v4 will be kept as it is on v5 to ensure backward compatibility with existing client integrations.
- Applicable settings defined in Windows registry (Windows) or ometascan-node.conf config file (Linux) for Node will be migrated to Core. Details: MetaDefender Configuration

Brand-new product user interface

The second notable update in this version is a brand-new UI for the management console, which was designed with the focus on the quality and thoughtfulness of the user experience. The new administrative console UI provides an advanced set of Core features and observable processing status.

Comprehensive executive dashboard reporting offers dynamic, real-time and actionable threat insights. It also provides visibility of your security posture allowing you to paint a picture of security that aligns with the overall business. The report includes easy-to-read charts, graphs, and statistic explanation that help your security professionals reduce clutter, gather information quicker, spot threat activity early and act swiftly.



We're continuing to improve our product UI to provide users an excellent experience with the application and overall usability. Please expect more powerful UI features coming in next releases.

New compression archive engine for archive sanitization

In previous version, MetaDefender Core's archive engine implemented four functions including decryption, extraction, encryption and compression that with a high volume of files it takes time to process them all. For the purpose of enhancing archive handling performance, this release comes with a separated compression archive engine, which dedicatedly processes archive sanitization (compression and encryption). It is also created to facilitate maintenance and further feature development.

Archive sanitization feature now requires Deep CDR license

As to respect sanitization practice and ensure unknown threats prevention as designed, now starting on version 5, it's mandatory to have Deep CDR licensed for archive sanitization feature enablement on MetaDefender Core workflow configurations.

Archive handing related engines changed

The legacy "Archive" engine on MetaDefender Core v4 will be changed its name to "Archive Extraction" engine, even when you are still on MetaDefender Core v4.

A brand new "compression archive engine" is to dedicatedly handle archive sanitization in v5 (archive compression), that helps relevant archive handling performance improvement, easier for maintainance and further feature development.

Notes when you are still on MetaDefender Core v4 (not upgraded yet)

The legacy "Archive" engine (now name changed to "Archive Extraction" engine) on MetaDefender Core v4.x will still cover both extraction and compression archive tasks as usual.

Even though your MetaDefender Core v4.x might now have a new "Archive Compression" engine auto downloaded, but that engine cannot be deployed (marked as "permanently failed" in relevant engine status check API). This behavior is expected, and not causing any impact to MetaDefender Core processing flow, or its performance though.

- Just in case your client integration has a check for engine status before triggering scan request, then you might want to disable "Archive Compression" engine status on MetaDefender Core v4 using REST API. Detailed instruction: Archive Compression Engine Permanently Failed on MetaDefender Core v4
- When you later upgrade to MetaDefender Core v5 from v4, then you might need to manually re-enable "Archive Compression" engine to make it active and running on MetaDefender Core v5.

Notes for licensing

For offline deployment (MetaDefender Core is not connected to the Internet directly for online update), you will need to re-activate again on OPSWAT portal to retrieve latest engine list in .yml file including "Archive Compression" engine if you are currently licensed with Deep CDR.

Meanwhile, for online deployment, this manual procedure is not required since MetaDefender Core will cover that itself.

Import & export configurations

Not only workflow settings like in previous versions, but also all other general settings (update, security, data retention, server, hashes to skip) and most importantly, user settings, will be able to export and import on MetaDefender Core version 5.

Learn more at Import/Export configuration

Backward compatibility supported

The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

Windows application event logging disabled by default

Applicable only to Windows OS based MetaDefender Core.

Help avoid duplicated logging with separate file option, and improve overall processing performance. Of course, when being still valid to you, you are still supported to enable it back.

CSRF token header renamed

CSRF token originally created for security strengthening and being used by MetaDefender Core management console UI since version 4.21.0 via header `oms_csrf_token`

To get it compliant with some load balancers which do not accept underscore in header name by default, and save you from additional load balancer configuration step, we have now changed the header name to `oms-csrf-token`

This change does not impact to any REST API based client integration.

More comprehensive and accurate error messages on REST APIs

Impacting on following REST API endpoints:

- `PUT /admin/config/session`
- `PUT /admin/config/update`
- `PUT /admin/config/scan`
- `PUT /admin/config/skip`
- `PUT /admin/config/auditlog`
- `PUT /admin/config/webhook`
- `PUT /admin/config/rule`

Upload time and queue time logic updated in JSON scan response

Impacting on following REST API endpoints:

- `GET /file/{data_id}`
- `GET /hash/{md5/sha1/sha256}`
- `GET /archive/{data_id}`
- `POST /file/{async}`

Added new field:

- `receive_data_timestamp`: The timestamp when upload progress started (first byte received) (in milliseconds)
- `upload_time`: Total time elapsed for upload process (in milliseconds)

Modified existing field (new logic):

> **Backward compatibility supported**
> The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

- `upload_timestamp`: The timestamp when upload progress finished (all bytes received) (in milliseconds)
- `queue_time`: Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue time = start_time - upload_ timestamp) (in milliseconds)

Release version 5.0.0 also comes with variety of outstanding features and improvements.

For detailed information about our latest release please view our Release notes

Installation

Starting with MetaDefender Core 4.20.0, the centralized/shared database deployment model is supported (all MetaDefender Core instances share the same database), along with traditional standalone database mode (every MetaDefender Core instance has its own database to manage).

- Standalone mode
- Shared Database Mode

Installation

1. Download the package of your choice from My OPSWAT
2. Install the package on your computer via the Command Line or via the Wizard
3. Open a web browser and point to http://<server name or IP>:<port>
    - The default port is **8008**
    - In case of problem check Inaccessible Management Console page
4. Complete the required steps of the basic configuration wizard
5. You must activate this deployment to use its features

Installation notes

- If the MetaDefender Core package **dependencies are not installed** on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.
- During installation the databases might need to be upgraded. This could take noticeable time depending on database size (eg.: length of scan history).

Standalone mode

Wizard Installation

The Install Wizard is only for the Windows installer (.msi file).

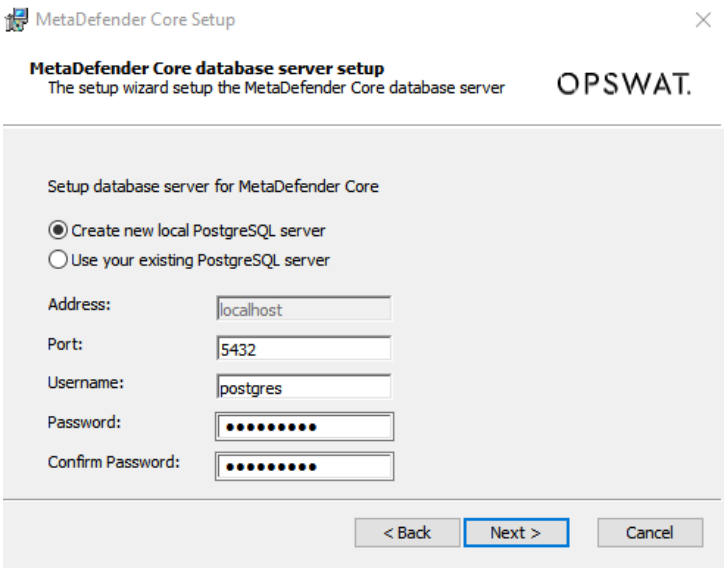
Since MetaDefender Core 4.19.0, we have switched to using PostgreSQL DBMS which needs a dedicated database server. MetaDefender Core supports two modes to setup PostgreSQL server:

Local (Bundle) PostgreSQL server

MetaDefender Core will install a new PostgreSQL server locally in the same box with the product.

> **PostgreSQL user credentials**
> Unicode characters are NOT supported for "Username" and "Password"
>
> "Password" does not accept # (hash) as starting character (e.g. #ABC)

> **The ignition file**
> If the ignition file exist in the designated location, then MetaDefender Core will use the details defined in the ignition file, rather than what is passed via the installation wizard.
>
> More info about the ignition file based installation: Command Line Installation

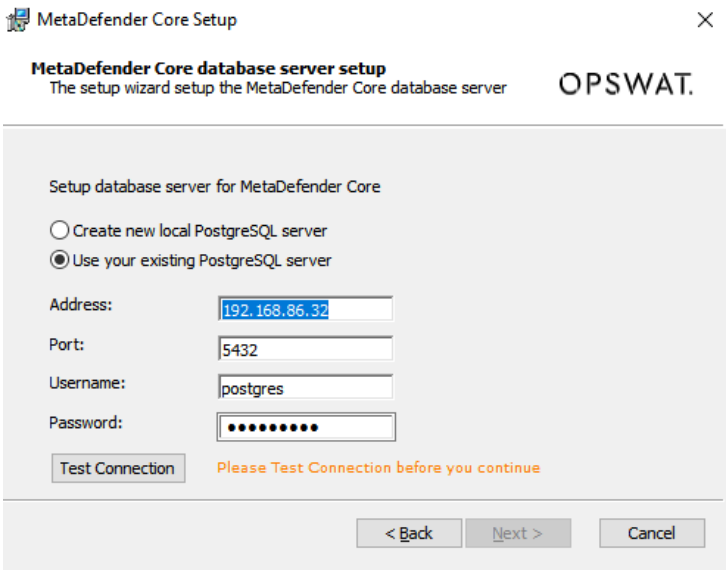


Remote (Existing) PostgreSQL server

MetaDefender Core will connect to remote pre-installed PostgreSQL running remotely.

> **PostgreSQL connection configuration**
> "Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

> **PostgreSQL user privilege requirements**
> MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.
>
> Details: PostgreSQL User Privilege Requirements



Command Line Installation

Ignition File

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Make sure to create the ignition file, if it doesn't exists before you go ahead and install MetaDefender Core via command line.

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

MetaDefender Core supports two modes to setup PostgreSQL server:

1) Local PostgreSQL server

- MetaDefender Core will install a new PostgreSQL server locally, next to the product.
  - A sample iginition file for PostgreSQL server information (***Only non-Unicode characters supported for "user" and "password"__***):

ometascan.conf bash

```
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
```

2) Remote PostgreSQL server

> **Info**
>
> **PostgreSQL version 12 (at least 12.3) or version 14 is required.**
>
> According to PostgreSQL notice, version 12 will stop receiving fixes on November 14, 2024, so we suggest that you use version 14.

MetaDefender Core will connect to a pre-installed PostgreSQL server running remotely and will setup its database.

Make sure to follow steps at Open Connection On PostgreSQL Server first, so that your remote existing PostgreSQL server to accept upcoming connection from MetaDefender Core.

A sample ignition file for PostgreSQL server information:

ometascan.conf bash

```
[dbserver]
type=remote
host=192.168.86.32
port=5432
user=existingserver_admin_user
password=existing_server_pass
```

After installing MetaDefender Core successfully, you may want to remove all credentials info created for [dbserver] section in ignition file for security reason.

> **Warning**
> If the MetaDefender Core package dependencies are not installed on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.

Debian package (.deb)

bash

```
sudo dpkg -i <filename> || sudo apt-get install -f
```

On Red Hat Enterprise 7.x Linux / CentOS package (.rpm)

bash

```
sudo yum install <filename>
```

For systems which enabled GPD check flag:

bash

```
sudo yum install --nogpgcheck <filename>
```

Red Hat Enterprise Linux 8.x package (.rpm)

Install the latest PostgreSQL repository RPM:

> **Note**
> Due to the shorter support cycle on Red Hat Enterprise Linux, all supported versions of PostgreSQL are not available on this platform. The PostgreSQL Yum Repository will integrate with the normal systems and patch management, and provide automatic updates for all supported versions of PostgreSQL throughout the supported lifetime of PostgreSQL. Reference: https://www.postgresql.org/download/linux/redhat/

bash

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

In case installing a remote PostgreSQL on either the same MetaDefender Core server or another server, you need to install the following additional package. PostgreSQL version 12.3 or newer is recommended.

bash

```
sudo dnf install -y postgresql12-contrib
```

Install MetaDefender Core:

bash

```
sudo dnf install -y <filename>
```

Windows package (.msi)

On Windows systems it is possible to install the product by running the corresponding .msi file.

From command line interface it is also possible to install the product by executing

bash

```
msiexec /qn /i <msi file name> <option key>=<option value>
```

where the possible keys and their default values are the following:

Key	Default Value	Description
INSTALLFOLDER	\Program Files\OPSWAT\MetaDefender Core	Customize installation folder for product Example: `INSTALLFOLDER="D:\Products"`
RESTADDRESS	*	REST interface binding IPv4 or IPv6 address ('*' means that service listens on all IPv4 and IPv6 interfaces)
RESTPORT	8008	REST interface binding port
REPORT_ENGINE_ISSUE	true	Enable reporting of engine issue count. (possible values: "true" or "false").

Note

Reporting of engine issue count

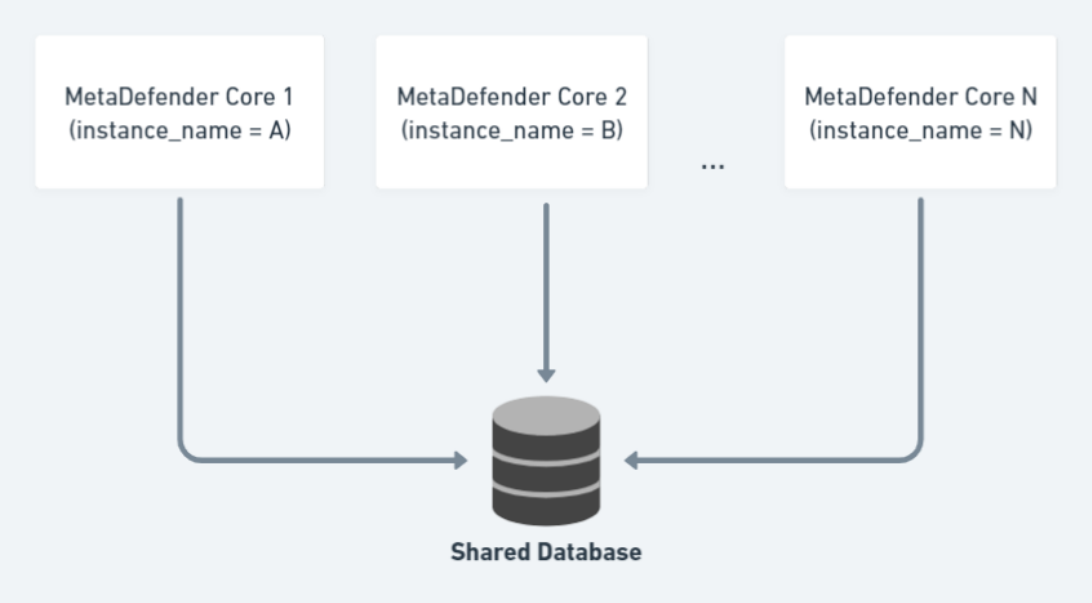
If reporting of engine issue count is enabled, MetaDefender Core server will send only the **number** of initialization errors and **number** of unexpected stops for the specific db/engine version. This information is sent over a HTTPS channel when the product downloads the latest package descriptors. This information is used for early detection of any specific 3rd party engine quality issues.

For details on using msiexec please consult Windows installer documentation.

Shared Database Mode

Starting with MetaDefender 4.20.0, you can use a centralized database deployment model, where all MetaDefender Core instances can share the same external database.

That helps auto-sync all product configurations (**except Single-Sign-On** that requires unique settings for each Core), processing history, live/on-going scan result query, audit, statistics data.



This article demonstrates how to install and setup MetaDefender Core 4.20.0 or newer with shared database mode.

Important notes & current limitations:

Currently not supported

- Does not support Central Management use-case with shared DB mode.
- Does not support to switch from shared database mode back to standalone database mode.
- Does not support to change instance_name of MetaDefender Core in shared database deployment model after installation.
- Does not support upgrading older MetaDefender Core from non-persistent mode to using the shared database mode.
- Does not support upgrading older MetaDefender Core from standalone mode to using the shared database mode.
- Does not support to upgrade multiple MetaDefender Core instances at once in shared database mode
    - Stop all Core instances' scanning service before performing the upgrade.
    - Upgrade each Core server one by one instead.

- For each Core, when the upgrade finished, then you can start using that Core for scanning without waiting for the other Core instances' upgrade to be done.

> **Disclaimer**
> MetaDefender Core does not support mode switching.

Requirements for shared database setup:

- The central remote PostgreSQL server (for shared database) could be utilized with a built-in (native) PostgreSQL database server installed with one of the MetaDefender Core instances in the network. Or using a dedicated existing PostgreSQL database server (see detailed requirements for PostgreSQL version at Recommended System Configuration).
- Before setting up, make sure to follow steps at Open Connection On PostgreSQL Server so that all MetaDefender Core instances could connect to the central remote PostgreSQL server.
- MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core, details: PostgreSQL User Privilege Requirements
- All MetaDefender Core instances must be installed on the same Operating System (Windows / Linux).
- The first MetaDefender Core instance connects to the shared database must be ready first (license activated, users/groups/roles created) before setting up other MetaDefender Core instances to connect and use that shared database.

Engine update:

- If engine updates is configured to be via Folder source, all MetaDefender Core instances must be able to access to the configured update folder, and the option DELETE FILES AFTER IMPORT will be unchecked and prevented for modification.

Authentication:

- Single-Sign-On (SSO) authentication settings (if using) cannot be shared among MetaDefender Core instances because SSO requires an unique login URL for every different Core.
- If an user is locked for login on a certain MetaDefender Core instance(e.g. exceeded the number of failed login times), that user is not prevented for logging in on any other instances.
- If an user is locked for sign-in on a certain MetaDefender Core instance (e.g. exceeded the number of failed login times), if the admin releases lockout on that user on that MetaDefender Core instance, that user is still not able to login on other MetaDefender Core instances.

Sanitized / DLP / Quarantined location changed after MetaDefender Core install:

Windows:

Shared storage path will be stored in 3 strings under key global in registry: quarantinepath, dlppath, sanitizepath as below

(Default)	REG_SZ	(value not set)
dbmode	REG_SZ	4
dlppath	REG_SZ	\\10.40.50.100\test
instance_name	REG_SZ	481rvj
quarantinepath	REG_SZ	\\10.40.50.100\test
restaddress	REG_SZ	*
restport	REG_SZ	8008
sanitizepath	REG_SZ	\\10.40.50.100\test

You can set either 1, 2 or 3 different paths for the 3 settings above (means that quarantined, DLP and sanitized files will be stored in same or different storages corresponding). In case shared storage is not set in installation, values of 3 settings above will be empty.

Restart MetaDefender Core service after changed.

Linux:

Open file `/etc/ometascan/ometascan.conf` and set shared storage path for 3 keys quarantinepath, dlppath, sanitizepath under section [global]

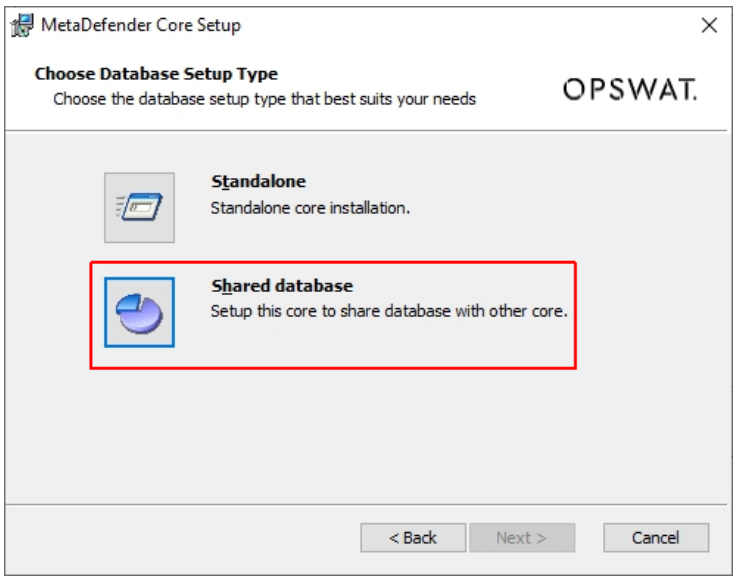
Restart ometascan service after changed.

Wizard Installation (Windows Only)

> **Info**
> Only applicable to MetaDefender Core 4.20.0 or newer using wizard on Windows OS.

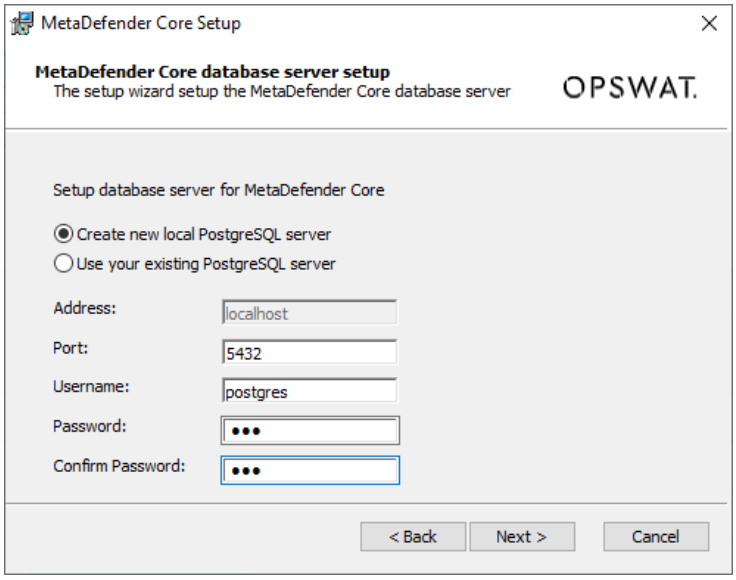
Clean install scenario

1. **Choose Shared database option:**

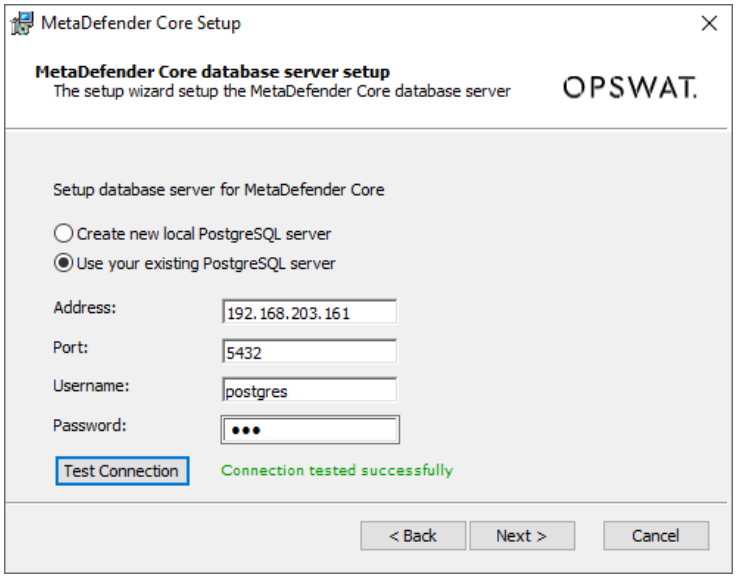


2. **Setup database server for MetaDefender Core:**

You can choose to create a native / built-in PostgreSQL server and setup the shared database on (then other MetaDefender Core instances will want to connect to that PostgreSQL server afterwards)



Or choose to connect to a pre-setup existing PostgreSQL server:



PostgreSQL connection configuration

"Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

PostgreSQL user privilege requirements

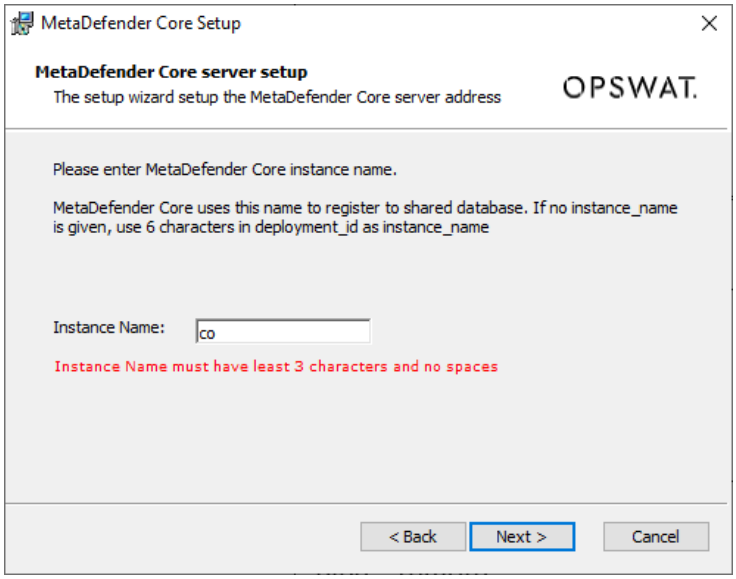
MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.

Details: PostgreSQL User Privilege Requirements

3. **Create an unique instance name for that MetaDefender Core (different from other MetaDefender Core instances):**

Instance name must contain at least 3 characters and no white space.

If you choose to leave it blank, which means it will use the default instance name with 6 characters extracted from that MetaDefender Core deployment ID.

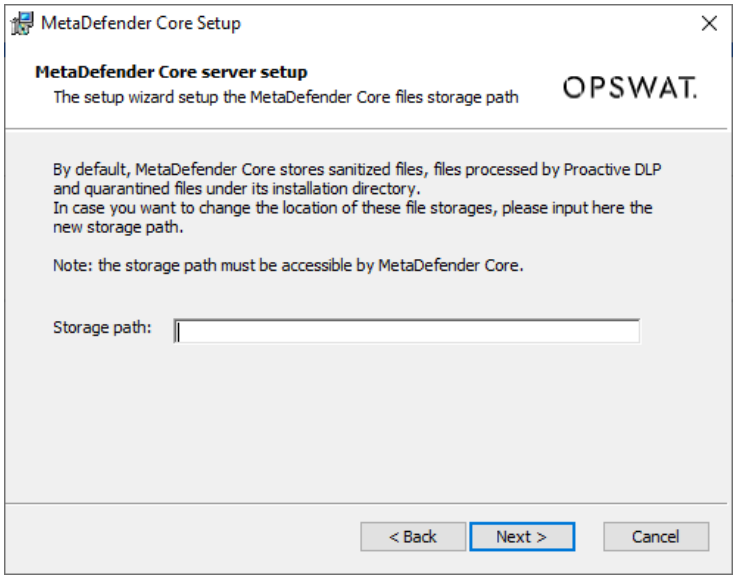


4. **Input the file storage path (sanitized, DLP, quarantined files)**

In the shared database model, all MetaDefender Core instances could use the same folder path to store all their own sanitized, DLP, quarantined files after processed.

The default location is: ***<MetaDefender Core installation folder>\data\ {sanitized|dlp|quarantine}***

If you wish to use that default location, just leave the input box blank and press Next.



Otherwise, you can specify a different location of sanitized/dlp/quarantined files. It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core.

For example:

- C:\new_folder
- \192.168.1.1\new_folder

Command Line Installation

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Clean install scenario

1. Create ignition file if not existed

- Windows: **C:\OPSWAT\ometascan.conf**
- Linux: ***/etc/opswat/ometascan.conf***

An example of ignition file:

ometascan.conf bash

```
eula=true

[global]
dbmode=4
instance_name=MetaDefenderCoreA
storage_path=D:\path\

[user]
name=admin
password=admin
email=admin@local

[dbserver]
type=remote
host=192.168.1.1
port=5432
user=postgres
password=your_own_password
```

Field	Required	Mapping to wizard UI	Possible values
global/dbmode	optional default is 1 (standalone)		1 - Standalone 2 - Central Hub deployment 3 - Non-persistent **4 - Shared**
global/instance_name	optional default is 6 characters of deployment ID		Unicode and number At least 3 characters No whitespace
global/storage_path	optional default is empty		Path to storages (DLP, quarantined, sanitized files) It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core. For example: • C:\folder • \192.168.1.1\folder

2. Run installers using command line to install

Please refer to Command Line Installation for installation command.

Deployment automation support

Note

An Ignition file could be utilized for either / both of following use-cases:

1. Pre-define PostgreSQL server information for MetaDefender Core to connect and setup database: only required on Linux installation, or Windows with command line installation. Learn more details at:
    a. Standalone Mode > Command Line Installation (applicable on version 4.19.0 or newer).
    b. Shared Database Mode > Command Line Installation (only applicable on version 4.20.0 or newer).
2. Automating to accept EULA, create a default local admin user and import MetaDefender Core configurations upon installed: Those steps could be automated using ignition file described in this page.

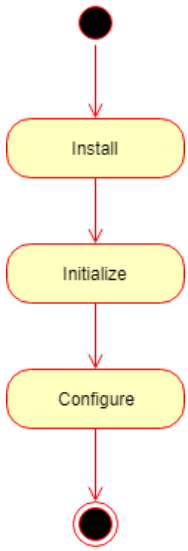
Ignition file location to create:

- **Windows**: `C:\OPSWAT\ometascan.conf`
- **Linux**: `/etc/opswat/ometascan.conf`

The product supports fully automated deployment. It means that it can be installed and configured with no human interaction.

The automated deployment can be split to three steps on a high level:

1. Installation
2. Initialization
3. Configuration



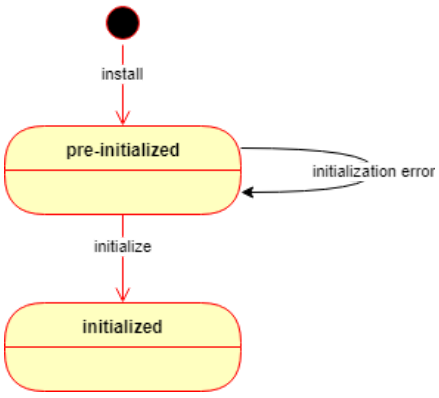
Installation

To automate the installation, install the product from the command line and provide the installation-time options as parameters to the installer. For further details see Command Line Installation

After the installation is complete, the product starts up and waits in a pre-initialized status. The product may be initialized in two ways:

1. Manually using the Wizard Setup or
2. Automatically using an *ignition file* (see below).

> **Warning**
> If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried after fixing the problem (e.g. placing the ignition file in its lookup location) and restarting the OPSWAT MetaDefender Core service.
>
> Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

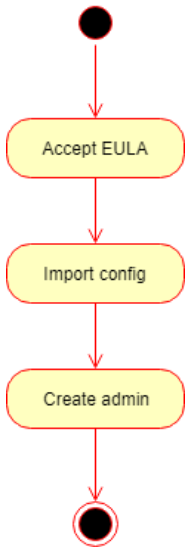


Initialization

Initialization is the process of bringing the product to an operable status.

Basically the initialization consists of the following steps:

1. Accept the End User License Agreement (EULA),
2. Import product configuration and
3. Create the first administrator user account.



Ignition file

The initialization process can be configured in a file called the *ignition file*.

> **Warning**
> In Linux, the ignition file and the folder /etc/opswat should have the permission 755

Ignition file location

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

The ignition file must be in conf format.

ometascan.conf bash

```
eula=true
[user]
name=admin
password=admin
email=admin@local
apikey=44c96cbb4e4e6c0a1d16bcb3c3da24def317
[config]
import=config_package.zip
import_password=123
[security]
session_cookie_samesite=Lax
[internal]
skip_startup_pages=true
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
private_username=my_internal_db_user
private_password=mypassword
[ocm]
ocm_url=https://my.opswat.com
reg_code=XXXXXXXX
```

Ignition file fields

The ignition file accepts the following fields:

Section	Key	Required	Accepted Values	Description
	eula	Optional (mandatory when having [user] section)	true false (default)	Whether to accept the End User License Agreement. This key must be set at the top of ignition file. This key must be set to "true" to accept the EULA. Any other value will cause the initialization to fail. When having [user] section defined, then this key must be set.
[user]		Optional		Initial administrator user account properties. The Administrator role is granted to the account.
	name	Optional	string	User name for the initial administrator user account.
	password	Optional	string	Password for the initial administrator user account. WARNING! Clear text password The password in this configuration file must be stored in its clear-text format and as so it may be visible for unauthorized parties.
	email	Optional	string	E-mail address for the initial administrator user account.
	apikey	Optional	string	API key for the initial administrator user account.
[config]		Optional		
	import	Optional	string	Path to a file in json/zip format that contains a previously exported configuration to be imported. Zip-format file contains a full configuration including user management settings, so it will overwrite [user] section.
	import_password	Optional	string	Password to decrypt .zip configuration package.
[security]		Optional		

Section	Key	Required	Accepted Values	Description
	session_cookie_samesite	Optional	Strict Lax (default) None	Declare if session cookie should be restricted to a first-party or same-site context.
[internal]		Optional		
	skip_startup_pages	Optional	true false (default)	Skip all welcome and other startup pages (e.g. upgrade result) when installing / upgrading MetaDefender, and going straight to the product index default page.
[dbserver]		Optional		
	type	Mandatory when having [dbserver] section	local remote	Where the PostgreSQL database server should physically locate. "remote" means PostgreSQL server and MetaDefender Core are not in the same machine.
	host	Mandatory when having [dbserver] section	string	IP address / domain name of the server where PostgreSQL server locates. "localhost" can be used when applicable.
	port	Mandatory when having [dbserver] section	number	Port of PostgreSQL server is listening for connections from clients (i.e. MetaDefender Core).
	user	Mandatory when having [dbserver] section	string	PostgreSQL server's user. SUPERUSER privilege is required for MetaDefender Core to setup its database and extensions for the first time. **Only non-Unicode characters supported.**

Section	Key	Required	Accepted Values	Description
	password	Mandatory when having [dbserver] section	string	PostgreSQL server's user credentials. **Only non-Unicode characters supported.**
	private_username	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
	private_password	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
[ocm]		Optional		
	ocm_url	Optional	string	URL to My OPSWAT server.
	reg_code	Optional	string	A registration code to enroll to My OPSWAT server.

Detailed initialization process

1) After the product has been started, it looks for the ignition file in the designated location.

Windows: C:\OPSWAT\ometascan.conf ** Linux: /etc/opswat/ometascan.conf

** 2) If an ignition file is found, then

2.1) It gets validated, and if it is valid, then based on the information found in the ignition file:

- The EULA is accepted,
- The configuration is imported,
- The administrator account is created.
- If any of the above steps fails, then the error is logged, and the initialization gets terminated.

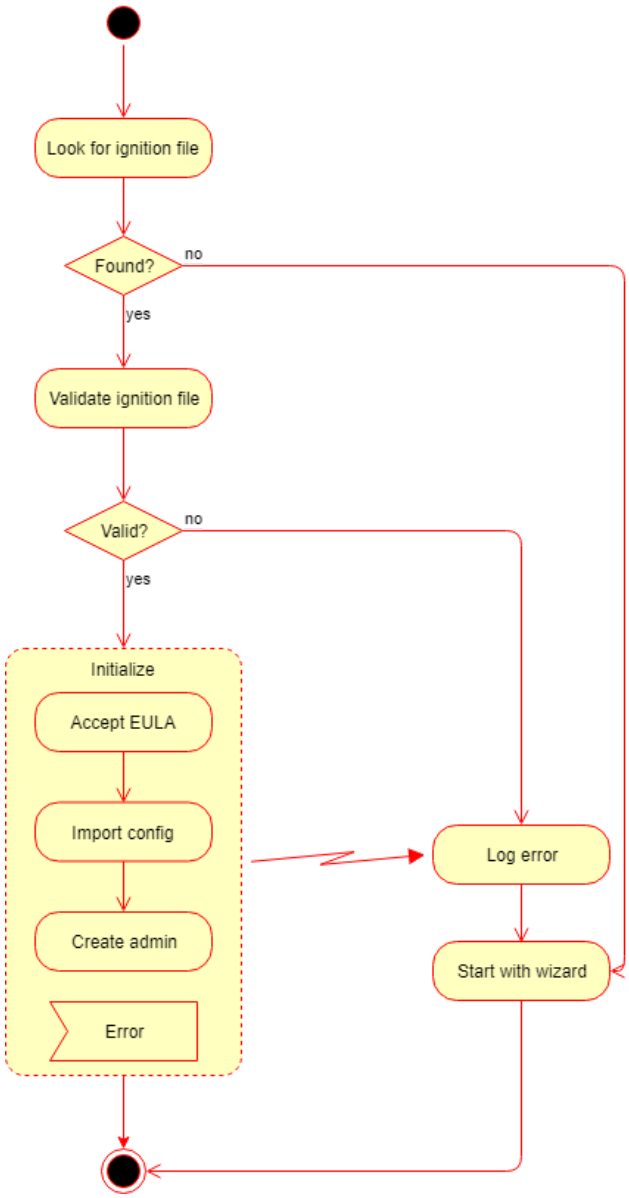
> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

2.2) If it is not valid, then the error is logged, and the initialization gets terminated.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

3) If there is no ignition file, then no initialization is performed.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.



Warning

If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried fixing the problem (e.g. placing the ignition file to its lookup location) and restarting the OPSWAT MetaDefender Core service.

Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

Configuration

After the initialization is complete, the product is ready with the default and the imported configuration.

This configuration can be later changed calling the configuration API functions. For further details about the API see Configuration related APIs API

CIS Level 1 Guidelines

Support Red Hat Enterprise Linux 9
For more details about Center for Internet Security (CIS) please refer to this document

Instruction steps

I. Install OpenSCAP

bash bash

```
yum install openscap-scanner scap-security-guide
```

II. Generate a result file and a HTML report using OpenSCAP scanner tool

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

III. Remediation of CIS Level 1 issues

Generate a remediation script based on the `ssg-rhel9-ds.xml` file:

bash

```
oscap xccdf generate fix --profile
xccdf_org.ssgproject.content_profile_cis_server_l1 --fix-type bash --output
remediations.sh /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

And execute remediation script.

bash

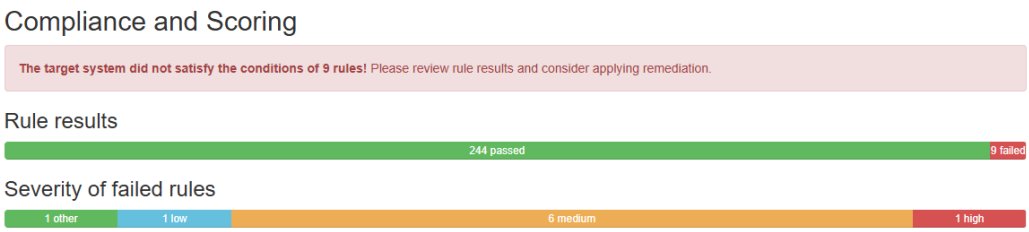
```
./remediations.sh
```

IV. Review the results after remediation

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

Example:



Notes

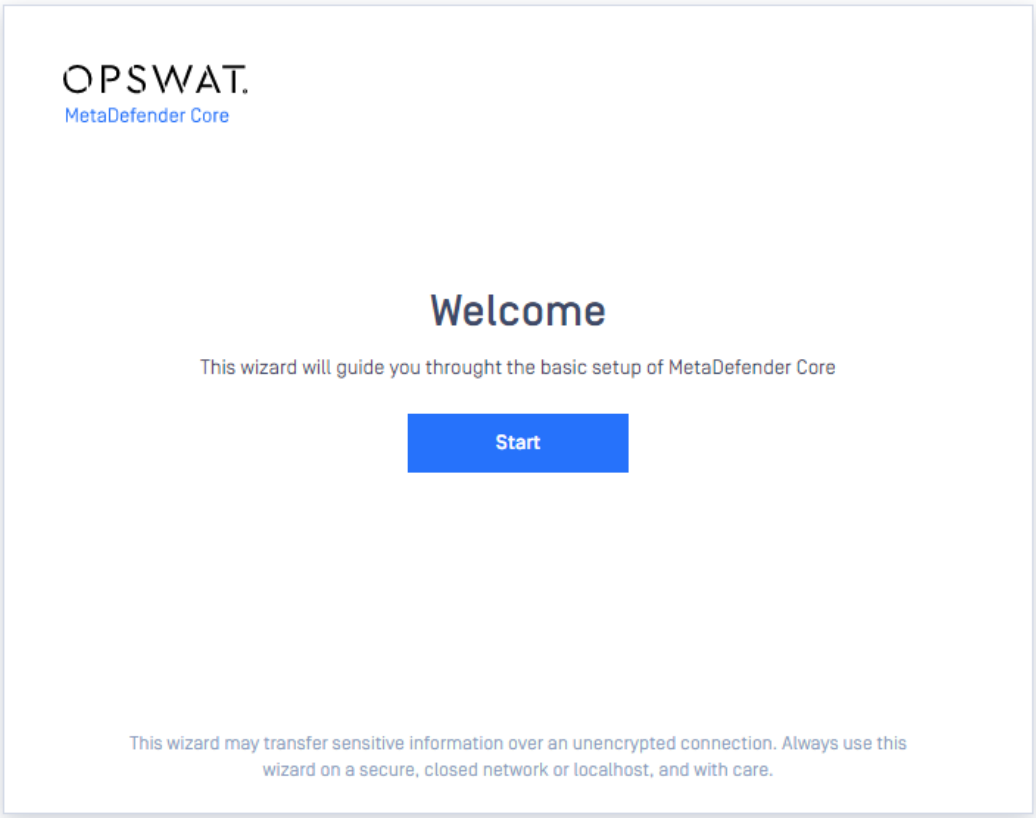
- CIS Level 1 requires `/tmp` folder to be mounted in a separate partition. Please ensure that that new partition have enough disk space for MetaDefender Core to run.
- CIS Level 1 requires that "Ensure No World-Writable Files Exist".
    - For now, when freshly installing MetaDefender Core, all its binary files meet the requirement.
    - When installing/updating engines, some engines might create additional files for its operation, and it might violate this requirement. In this case, you need to again execute the remediation script in the step III.

Wizard Setup

Introduction

Access the MetaDefender Core management console for the first time (i.e. http://localhost:8008), you are to complete a quick walkthrough for wizard setup in order to be able to use the product.

The administrative management console will be available only upon you have successfully finished this wizard stage.

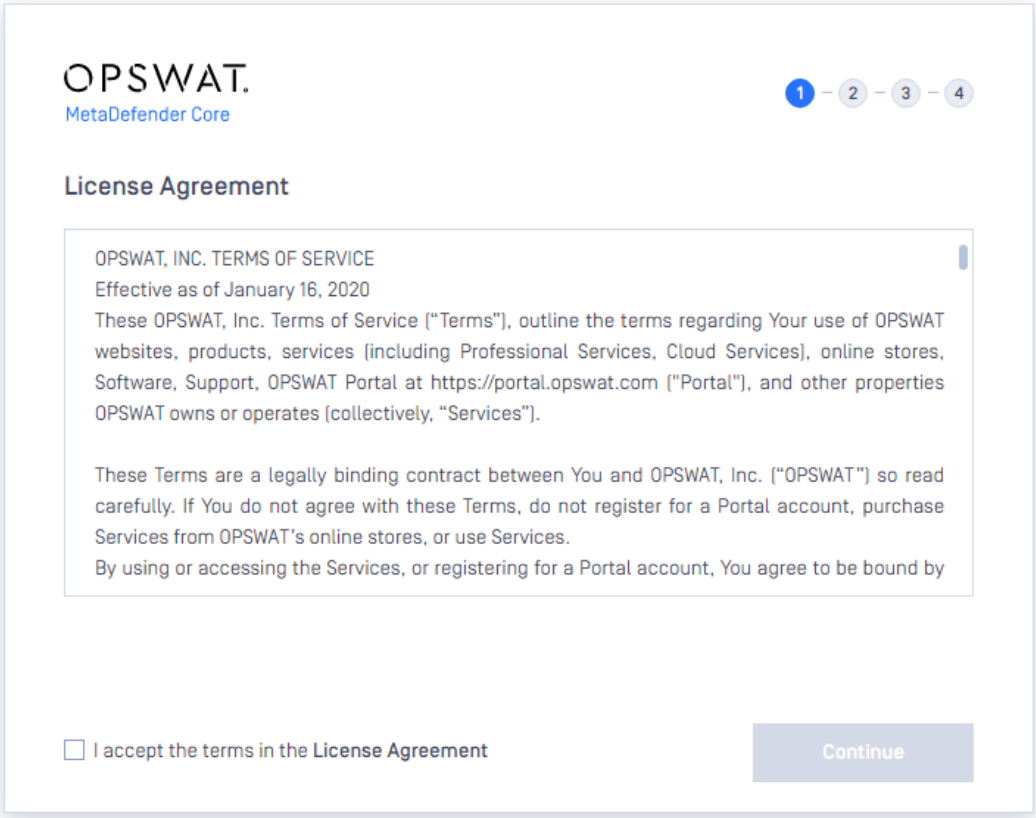


Warning

This wizard may transfer sensitive information over an unencrypted connection. Always use this wizard on a secure, closed network or localhost, and with care.

Basic configuration steps

End-User License Agreement



In the first page you can find the End-User License Agreement. You have to accept the terms before moving on.

Please read through the EULA (https://docs.opswat.com/policies/privacy-policy/portal-privacy-policy) carefully and if you agree with it, check `I accept the terms in the **License Agreement**` and click `Continue` to proceed.

Local Admin User Setup

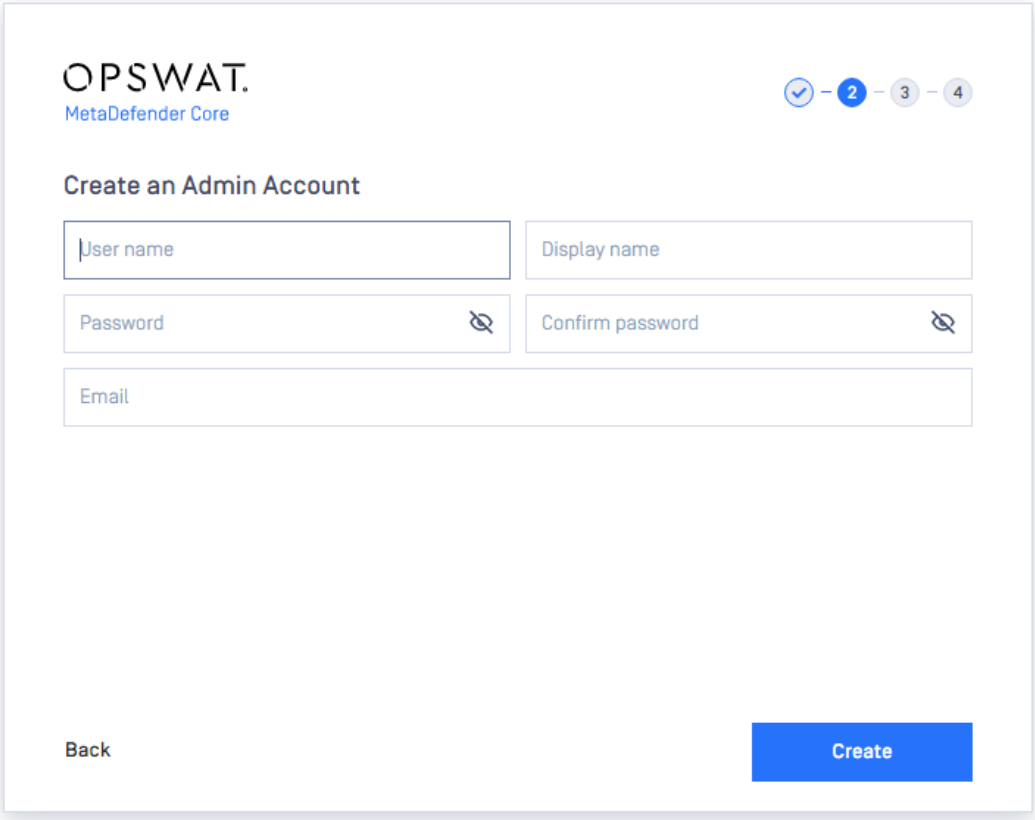
The next step is to set up a default local administrator account.

This account will be the first one being able to access the administrative management console, and to create accounts for other users. You have to fill out all fields in this page to be able to move forward. When you are done, click NEXT to continue.

> **Info - User Directory**
> The administrator account, that is created via the basic configuration wizard, is always added to the LOCAL user directory as a member.
>
> Learn more about user directory: User directories

The following information is required for the administrator account:

ACCOUNT NAME	The unique name of the account that is used at the time of login and in log messages for accountability.
ACCOUNT DISPLAY NAME	Name of the person bound to this account. This name (appended to the name of the account's user directory) is displayed in the top right corner of the administrative management console.
PASSWORD	Password of the user bound to this account that is used at the time of login. Passwords sent clear-text As long as TLS is not configured for the basic configuration wizard, passwords are sent clear-text over the network and may be disclosed to unauthorized parties. As a mitigation action: 1. Either use the wizard on *localhost* or on a direct network connection, or 2. Enabling HTTPS as soon as possible and change the password immediately if it has already been set.
EMAIL	Email address of the person bound to this account.



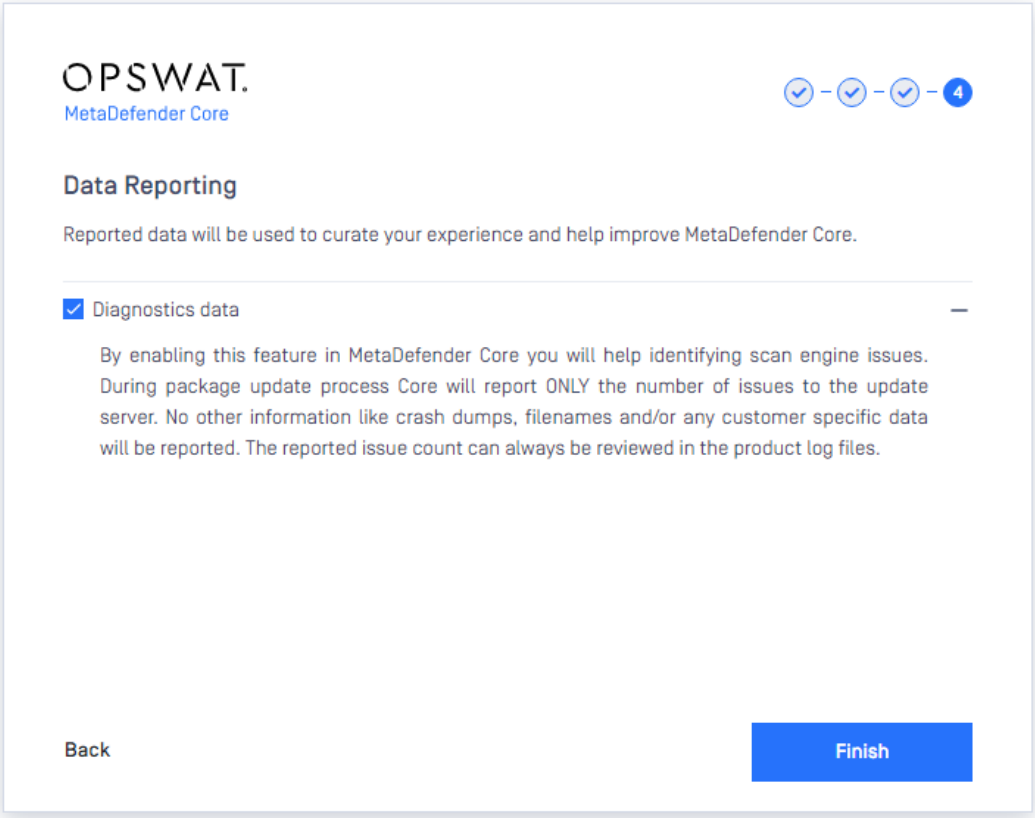
License activation

For license activation details see License Activation

Data Reporting

Enabling this option to allow MetaDefender Core collect its own engine issues and report back to OPSWAT for further analysis.

MetaDefender will not collect any other information.

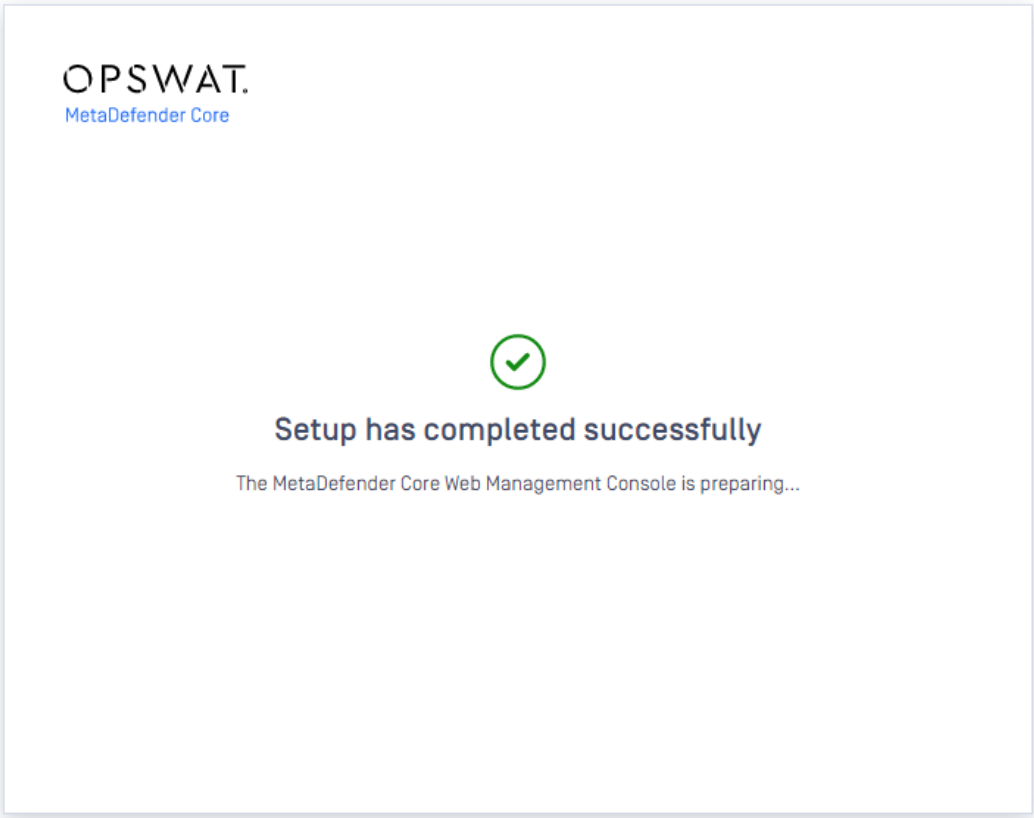


Wizard completion

After you have completed every steps you are ready to finish the wizard and start using the product. Click the `Fishish` button to complete the wizard.

> **Warning**
> The product's service will be restarted and the browser will be redirected to the administrative management console. This could take several seconds.

You can login to the Web Management Console with the administrator user that have just been created in the previous steps.



Setup HTTPS

Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network. Websites, like the Web Management Console, are able to use TLS to secure all communications between their servers and web browsers.

The TLS protocol aims primarily to provide confidentiality (privacy) and data integrity between two communicating computer applications.

Warning

No TLS for the wizard

By default, TLS is not enabled for the basic configuration wizard. As a consequence sessions between the wizard's backend and the browser may be insecure.

Info

Performing the same steps as for the Web Management Console, it is possible to set up TLS for the basic configuration wizard. Remember completing the TLS setup before launching the wizard.

For instructions to set up TLS see Enabling HTTPS

License Activation

##

To activate your installation go to the **Settings > License** menu in the Web Management Console. If you have no valid license, you will only see your installation's **deployment ID**. You will also see a warning in the Web Management Console header.

After successful activation, the product will start downloading the latest available scan engines and malware databases. You can follow the status of the scan engine installation on the **Inventory > Modules** page.

When all the licensed engines are installed, you can start using the MetaDefender Core to process files - File Processing.

Deployment ID

Each MetaDefender Core instance represented by an unique system fingerprint called `Deployment ID`

MetaDefender Core licensing is purely based on deployment ID activated. Therefore, everytime your MetaDefender Core is activated with a different deployment ID, then it will be captured as a different activation record accumulated into your total activation times allowed.

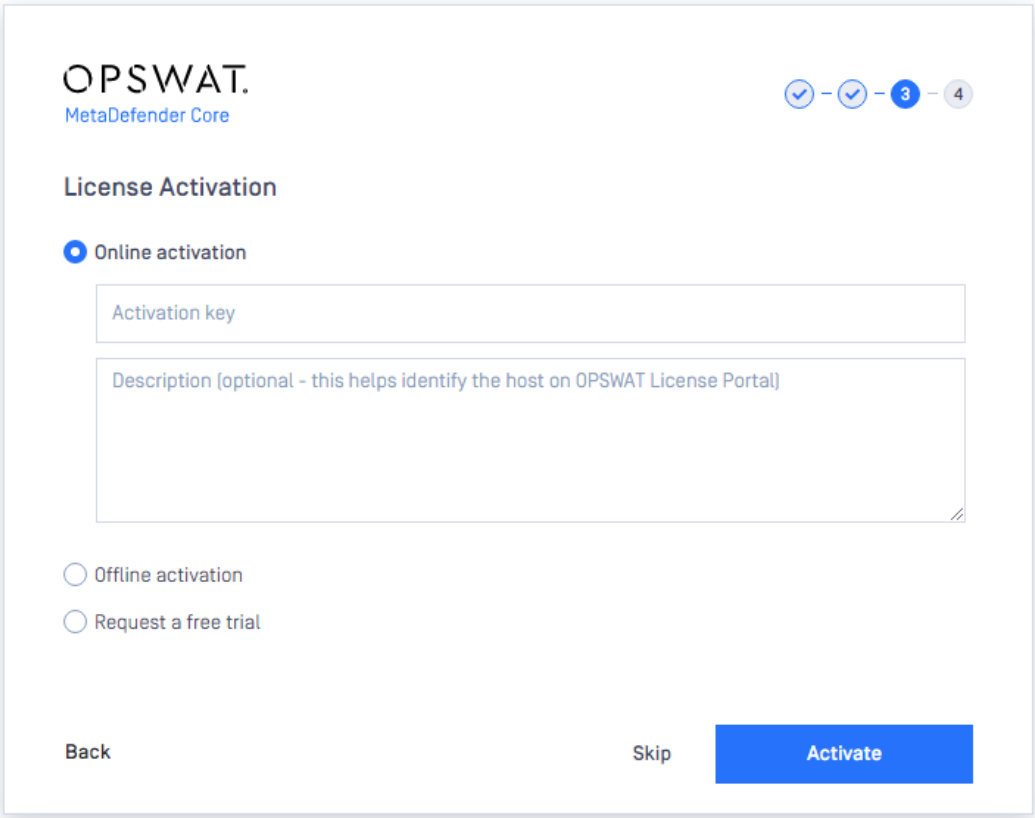
> **Notice**
> Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you alter the system where MetaDefender Core relies on (e.g. network card, OS hostname).
>
> Deployment ID is an UUID version 4 random based string. Learn more at: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you update the system where MetaDefender Core relies on.

Activation Modes

There are three different options for the product activation:

- Online activation: MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.
- Offline activation: Requiring you to upload a license file (.yml). That license file could be retrieved by activating your license via OPSWAT Portal. Follow the displayed instructions for details.
- Request trial key online: For evaluation purpose, you can receive an trial Activation key via email. Follow the displayed instructions for details.



Online activation

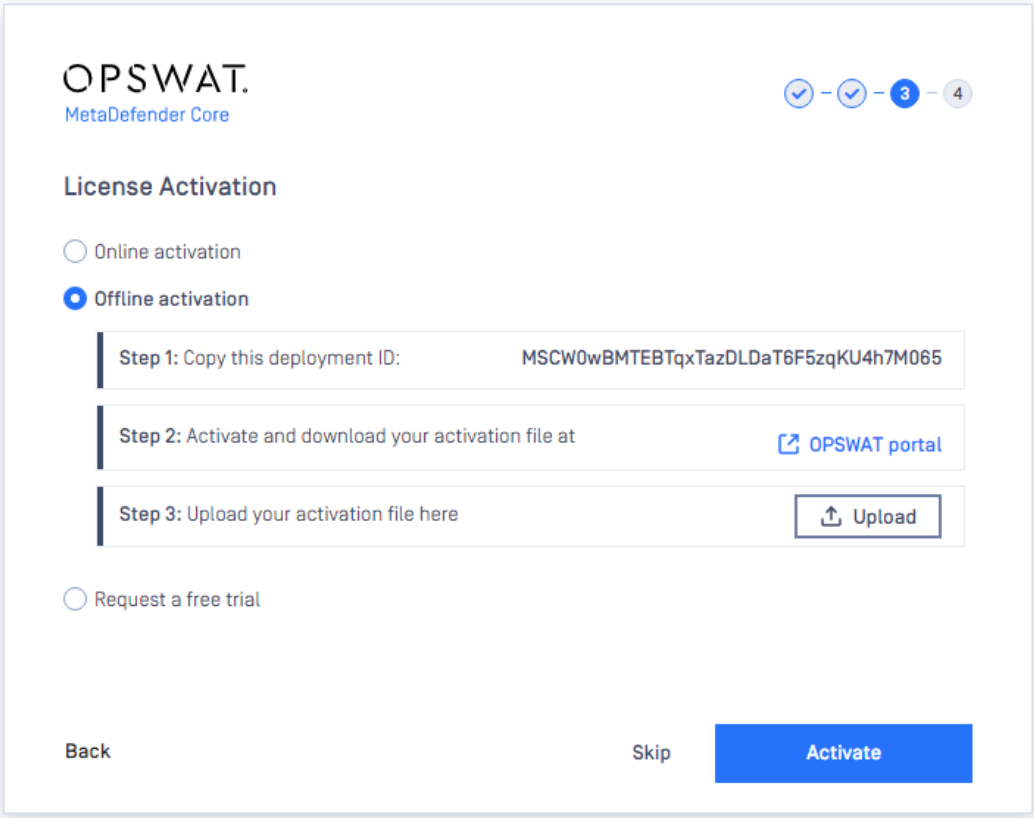
MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.

Required to input your Activation License key, and optional for description (but we recommend this for troubleshooting and auditing purpose if needed).

> **Warning**
> If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts' for license activation:
>
> - https://activation.dl.opswat.com Note: *IP address-based whitelisting on your firewall might fail after some time since OPSWAT uses CDN (Content Delivery Network) to faster delivery updates over the world, and IP address of edge servers might change over time.*

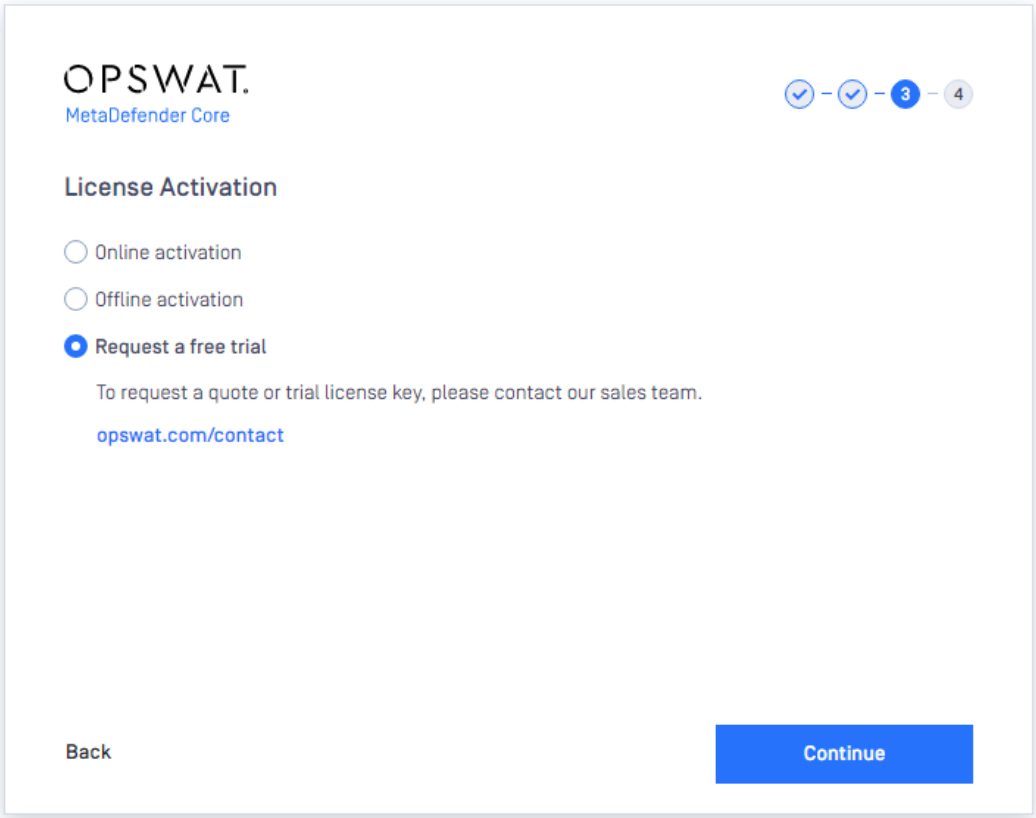
Offline activation

With no internet connection on the server the Metadefender v4 instance may be activated indirectly from a different machine, that has internet connection. The Deployment ID of the Metadefender v4 instance and the the Activation key received at the time of purchasing the product will be required. Follow the steps on the screen to activate the product offline.



Request trial key online

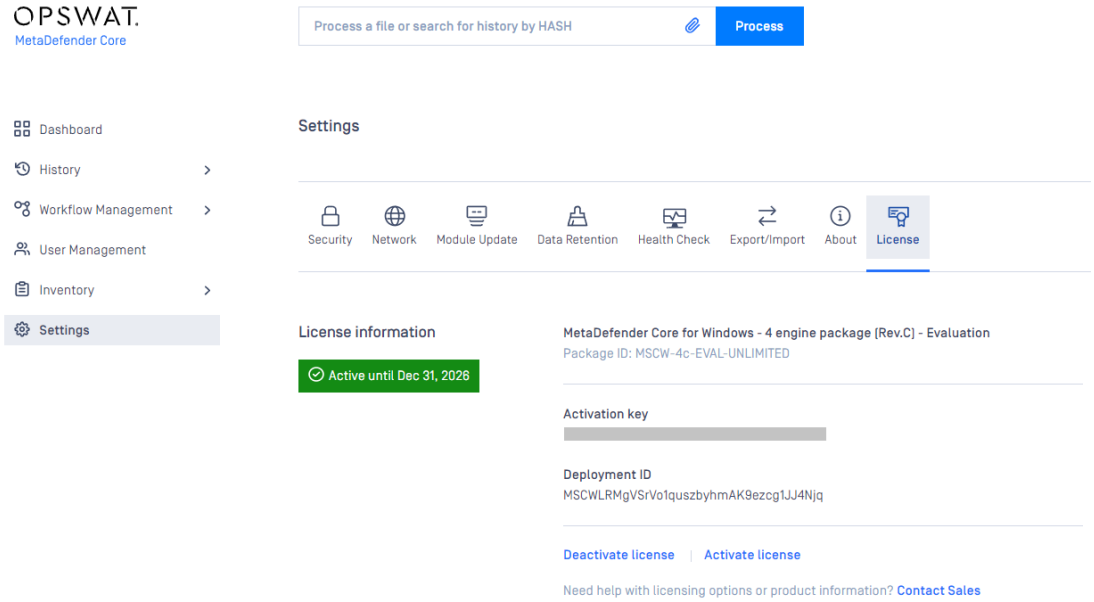
An evaluation license could be acquired by contacting our sales team.



Check Your License Details

For more license details and activating your MetaDefender Core, go to Settings > License menu tab on the Web Management Console:

- Product name: product name as on your order (e.g. MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation)
- Package ID: product identification as on your order (e.g. MSCW-4c-EVAL-UNLIMITED)
- Expiration: last day of license validity
- Activation key: key used for activating product
- Deployment ID: identification of this installation


OPSWAT.
MetaDefender Core
Process a file or search for history by HASH
Process
Dashboard
History
Workflow Management
User Management
Inventory
Settings
Settings
Security
Network
Module Update
Data Retention
Health Check
Export/Import
About
License
License information
Active until Dec 31, 2026
MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation
Package ID: MSCW-4c-EVAL-UNLIMITED
Activation key
Deployment ID
MSCWLRMgVSrVo1quszbyhmAK9ezcg1JJ4Njq
Deactivate license
Activate license
Need help with licensing options or product information? Contact Sales

Scripted license management

Using REST API calls there is an option for scripted activation and deactivation for MetaDefender Core servers with **no Internet connection or if your infrastructure is using automation to create/destroy MetaDefender Core instances**. In this chapter the steps of these two scenarios are described.

Requirements

- an installed MetaDefender Core instance without Internet connection
- another computer that has Internet connection and can run your activation scripts
- a manual or automated way to transfer data between the two computers

Activation steps

1. In order to activate MetaDefender Core, deployment ID and activation key are needed to generate the license file.
    a. Activation key should be purchased from OPSWAT.
    b. MetaDefender Core deployment ID can be queried by using REST API (for details see GET Get Current License Information API page)

> **Info**
> Save this deployment ID in your system, you might need this ID when the instance is unavailable at the time of deactivation.

2. Activate license and get the license file using the following URL, where <quantity> is 1.

activation call bash

```
curl -X GET https://activation.dl.opswat.com/activation?key=<activation-key>&deployment=<deployment unique-ID>&quantity=<quantity>
```

If the activation is successfully, a license file is downloaded. Save this file.

HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	license file	Activation was successfully.
200 Activation failed	error: '<user conform error message>'	Failed activation
200 Invalid parameter	error: 'Could not activate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not activate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
200 Invalid parameter	error: 'Could not activate your product because the quantity you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key, quantity or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Info
In case of any activation issue, contact OPSWAT support for help

3. Upload license file to MetaDefender Core: The license file should be uploaded to the MetaDefender Core v4 to activate the product. For details see page: POST Upload license key API

Deactivation steps

1. For deactivation, following infomation is required:
    a. Activation key
    b. MetaDefender Core deployment ID: can be queried by using REST API (for details see GET Get Current License Information API page)
2. Deactivate license using the following URL:

deactivation call bash

```
curl -X GET https://activation.dl.opswat.com/deactivation?key=<activation-
key>&deployment=<deployment unique-ID>
```

Response HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	result: ok	Successful deactivation
200 No active license found	error: 'Could not found any active license with the given parameters'	The license has not been activated yet or it has been deactivated already.
200 Invalid parameter	error: 'Could not deactivate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not deactivate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Important notes

Warning

Product activation is tied to several hardware, operating system and software parameters. In case of one or more major hardware or software parameter change the product might turn into deactivated status. Operating system updates, other software updates should not affecting the activation status.

If the product is online activated and has live Internet connection then in case of deployment ID change the product reactivates itself. If the product doesn't have Internet connection then the administrator is responsible to reactivate the product.

Because of the above administrator should consider licensing restrictions of the offline deployments if using any virtualization and/or containerization technologies.

File Processing

MetaDefender Core provides REST API interface for file processing business.

There are several ways to process files with MetaDefender Core:

- Analyze Files via Web Interface
- Process Files via REST API

Product Upgrading

> **Info**
> Simply run installer file, details could be found at Installation

> **Info**
> For command line mode, upgrading (e.g. on **Linux**), prior to running the installer file, please make sure to **create the ignition file** with PostgreSQL server information included. Details:
>
> - Standalone DB mode: Command Line Installation (Windows and Linux)
> - Shared DB mode: Command Line Installation

Recommendation for MetaDefender Core upgrade

Our goal is to help you execute the product upgrade continuously, efficiently, and correctly.

PostgreSQL service upgrade

- **Bundled PostgreSQL Service**: If you are using the bundled PostgreSQL server included in the installer, the PostgreSQL service will be automatically upgraded to the newer version (if available). No additional action is required for this process.
- **Remote PostgreSQL Service**: For remote PostgreSQL servers, customers are responsible for managing and upgrading the service independently. Upgrading MetaDefender Core does not update the remote PostgreSQL service.

Before the upgrade

Prepare a proper disaster recovery plan (DRP) including some common practices:

- Backup, or take a snapshot of your current working MetaDefender Core server.
  - This is to help quickly rollback for any upgrade failure might cause later for any reason.
- Optionally backup your PostgreSQL database, especially when using remote shared database mode.
  - Reference: https://www.postgresql.org/docs/current/backup.html
  - Please do it after stopping all linked MetaDefender Core instances to avoid potential data collision.
- Lastly, if backing up entire server is not feasible, then at least export MetaDefender Core configuration package Import/Export configuration just in case you'll need to start off a new server with current MetaDefender Core configuration to keep the scanning service running without any impact.

On the PostgreSQL server, double check server resource to ensure sufficient RAM memory and disk space to be preserved for the upgrade process.

- 8GB RAM available at least during the upgrade.

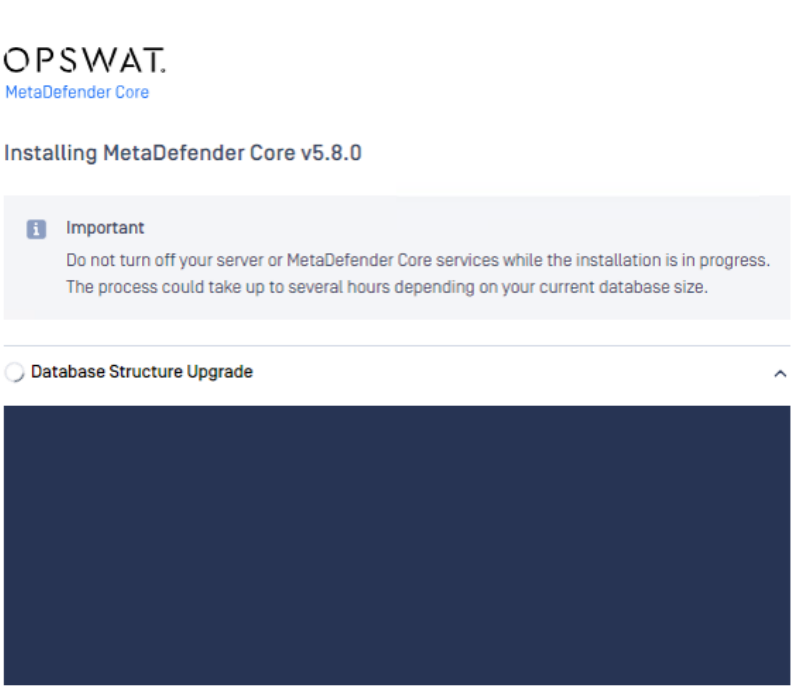
- Disk availability is greater than MetaDefender Core database size.

While doing the upgrade

Applicable for shared database deployment:

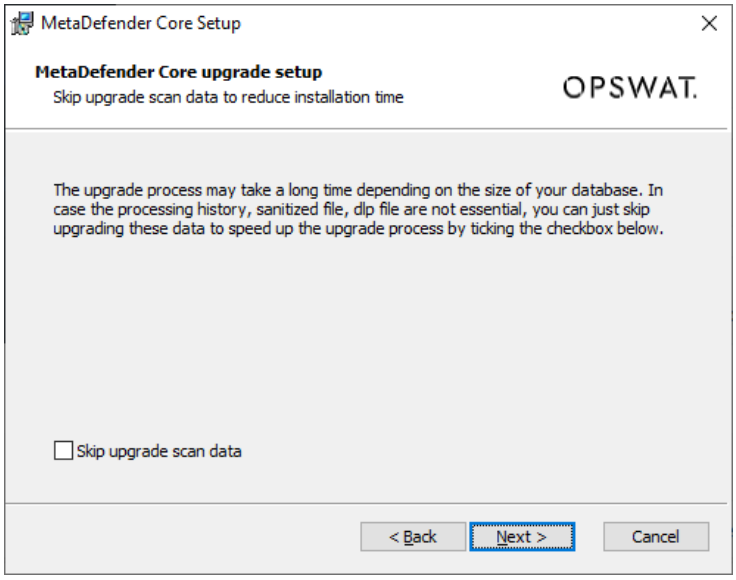
- Stop all MetaDefender Core instances' scanning service before performing the upgrade.
- Upgrade one (1) MetaDefender Core instance at a time.
- Proceed to upgrade MetaDefender Core instance where the bundle local PostgreSQL server is used for shared database setup.
- Otherwise is using remote PostgreSQL then the upgrade order does not matter.

For large size database, mostly for scan history, expecting long time to be spent on the upgrade process.



The product supports an option to skip data migration for scan history to expedite the process drastically and mitigate any failure risk. You can manually backup all data from your PostgreSQL server for offline usage if any.

Check "Skip upgrade scan data" to enable this feature while doing the upgrade.



or add following key in the ignition file:

none

```
[internal]
skip_upgrade_scan_data = 0
```

Troubleshooting

If the upgrade process ever runs into an error:

- Double check your network and services, applicable to remote PostgreSQL server.
- Optionally configure MetaDefender Core to skip migrating scan history.
  - Windows
    - Open Windows registry, and navigate to HKLM\Software\OPSWAT\Metascan
    - Add a new key `internal/skip_upgrade_scan_data` with value = 1
  - Linux
    - Open MetaDefender Core configuration file `/etc/ometascan/ometascan.conf`
    - Add new key into the file
      - [internal]
      - skip_upgrade_scan_data=1
  - Note: This key will be auto removed upon the upgrade finishes.
  - Then restart MetaDefender Core service to retry the upgrade.
    - Might retry this a few times until the upgrade is done.

Just in case the upgrade process still fails, please contact OPSWAT support with the support package - How to Create Support Package? This will help us analyze the root cause and figure out an optimal solution for your system failure.

Using external Load Balancer

Application Load Balancing (Layer 7)

This page provides information about recommended ways to use sessions with a Layer 7 load-balancer.

Most load-balancers have the ability to provide cookies for pinning subsequent traffic from a client to the appropriate server. This method is called sticky session, session persistence or session affinity. Using cookies efficiently requires the client to know when a cookie should be sent, should not be sent or should be deleted.

Sticky session load balancing

Each Metadefender Core instance has its own databases and application sessions that cannot be seen by other instances. Therefore, in order to get the related data/response to our queries we should ask the appropriate Metadefender Core v5 server. On the other hand, to keep the advantages of the used load balancing method, cookies should not be sent if it is not necessary.

Single file scanning

Step	Stage	Task	Cookie usage
1.	Sending file	Initiate processing a file on the client side. Send file through the load-balancer. (See POST File Analysis API)	Cookie should not be sent
2.	Sending file	Save the cookie and the data_id you got from the load-balancer	Save cookie
3.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Send cookie saved in step 2
4.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Send cookie saved in step 2

Batch scanning

Step	Stage	Task	Cookie usage
1.	Open batch	Initiate processing file(s) in batch. Request a batch ID through the load-balancer. (See POST Initiate Batch API)	Cookie should not be sent
2.	Open batch	Save the cookie and the batch_id you got from a Core server through the load-balancer.	Save cookie
3.	Sending files	Send file through the load-balancer. (See POST File Analysis (using batch header) API)	Send cookie saved in step 2
4.	Sending files	Save the data_id you got from the load-balancer.	-
	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Send cookie saved in step 2
5.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
6.	Getting batch status	See GET Batch Status API	Send cookie saved in step 2
7.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Send cookie saved in step 2
8.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API	Send cookie saved in step 2

If it does not matter which upstream server responds, the queries should be sent without cookie.

> **Info**
> It is recommended not to send cookies when it's not necessary to allow load-balancer to use its own method to share the load between Metadefender Core v5 servers.

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration through load-balancer

OPSWAT products that support HTTP load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	4.3.4	-
OPSWAT Client	Windows: 7.6.247.0 Mac: 10.4.243.0	-
MetaDefender Email Security	4.3.0	-
MetaDefender ICAP Server	4.3.0	-
MetaDefender Vault	1.3.0	-
MetaDefender for Secure Storage	2.0.7	-

DNS Load Balancing

Using this method is logically similar to Layer 7 load-balancing.

Briefly how it works

Client uses a domain name to send a query to a server. Client's DNS server has more "A" records for that name with different IPs. When a client resolves the server's domain name DNS server randomly chooses an IP for that name to send back. When a session is used on the application layer, client should know the IP address of the Core server that handles that specific session. In every other case, client should resolve the domain name with DNS query to let requests to be balanced between Metadefender Core servers.

Single file scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Sending file	Initiate processing a file on the client side. (See POST File Analysis API)	Use IP saved in step 2
4.	Sending file	Save the data_id got from the Core server	-
5.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Use IP saved in step 2
6.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Use IP saved in step 2

Batch scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Open batch	Initiate processing file(s) in batch. Request a batch ID. (See POST Initiate Batch API)	Use IP saved in step 2
4.	Sending files	Send file to the specific Core server with the basch ID saved in step 3. (See POST File Analysis (using batch header) API)	Use IP saved in step 2
5.	Sending files	Save the data_id you got from the Core server.	-
6.	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Use IP saved in step 2
7.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
8.	Getting batch status	See GET Batch Status API	Use IP saved in step 2
9.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Use IP saved in step 2
10.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API).	Use IP saved in step 2

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration via DNS load-balancing

OPSWAT products that support DNS load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	does not support yet	-
MetaDefender Client	does not support yet	-
MetaDefender Email Security	does not support yet	-
MetaDefender ICAP Server	does not support yet	-
MetaDefender Vault	does not support yet	-

SSL connection for PostgreSQL communication

> **Note**
> The SSL connection configuration is possible thanks to the support made by PostgreSQL. For reference: https://www.postgresql.org/docs/14/ssl-tcp.html
>
> MetaDefender Core is tested against this mode on particular version 4.21.1

Step 1: Generate SSL certificate for PostgreSQL server

> **Note**
> Following instructions are for self-sign certificate. For production environment, making sure to use the certificate signed by your trusted CA.

1) Generate a private key

bash

```
openssl genrsa -des3 -out server.key 1024
```

Sample output:

bash

```
Generating RSA private key, 1024 bit long modulus ........++++++
...............++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

Type your desired password / pass phrase to encrypt the private key

2) Remove the pass phrase to automatically start up the PostgreSQL server

bash

```
openssl rsa -in server.key -out server.key
```

> **Note**
> On Linux, make sure to set permission on the server.key file

bash

```
chmod og-rwx server.key
```

3) Create a self-signed certificate

bash

```
openssl req -new -key server.key -days 3650 -out server.crt -x509
```

You will be prompted to enter detailed information which is incorporated into your self-signed certificate request.

none

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:CA
Locality Name (eg, city) [Default City]:San Francisco
Organization Name (eg, company) [Default Company Ltd]:OPSWAT
Unit Name (eg, section) []:OPSWAT
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:dummy_email@opswat.com
```

For self-signed certificate, use the server ceriticate as the trusted root certificate:

bash

```
cp server.crt root.crt
```

Copy **server.key**, **server.crt**, **root.crt** to PostgreSQL data folder. For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data**
- Linux: **/var/lib/ometascan/pg_data/**

Step 2: Configure PostgreSQL server for SSL authentication connection

1) Create a custom config file (e.g. ssl.conf) for PostgreSQL.

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\postgres\ssl.conf**
- Linux: **/var/lib/ometascan/postgres/ssl.conf**

2) Content of ssl.conf:

none

```
ssl = on
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
```

3) Modify pg_hba.conf

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data\pg_hba.conf**
- Linux: **/var/lib/ometascan/pg_data/pg_hba.conf**

Add following (modify **::1/128** if needed to match with your environment requirement)

none

```
hostssl all all      ::1/128      scram-sha-256
```

After the modify, the content of pg_hba.conf should be like this (example):

> **Note**
> In a Linux environment, a local connection is required to upgrade from version 5.5.1 or earlier to version 5.6.0 or later.

none

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local         DATABASE  USER  METHOD  [OPTIONS]
# host          DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl     DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostgssenc    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnogssenc  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
local    all all                    scram-sha-256
hostssl all all      ::1/128      scram-sha-256

#More content.......
```

4) Reload the services

- Standalone DB mode: Restart **ometascan** service (MetaDefender Core service)
- Shared DB mode: Restart **ometascan-pg** / **ometascan-postgresql** service (MetaDefender Core PostgreSQL service) and then **ometascan** service (MetaDefender Core service)

Test your SSL database connection:

bash

```
psql.exe "sslmode=require dbname=postgres user=postgres host=localhost port=5432"
Password for user postgres:
psql (12.7)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
Type "help" for help.
```

This message indicate that you are now using SSL connection:

bash

```
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
```

Performance and Load Estimation

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Some factors that affect performance

- MetaDefender Core version
- MetaDefender Core engine package and configuration
  - set of engines (which and how many)
  - product configuration (e.g., thread pool size)
- System environment
  - server profile (CPU, RAM, hard disk)
  - client application location - remote or local
  - system caching and engine level caching
- Dataset
  - encrypted or decrypted
  - file types
    - different file types (e.g., document, image, executable)
    - archive file or compound document format files
  - file size
  - bad or unknown (assume to be clean)
- Performance tool

How test results are calculated

Performance (mainly scanning speed) is measured by throughput rather than unit speed. For example, if it takes 10 seconds to process 1 file, and it also takes 10 seconds to process 10 files, then performance is quantified as 1 second per file, rather than 10 seconds.

- total time / total number of files processed: 10 seconds / 10 files = 1 second / file.

Standalone DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.2.416.0

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

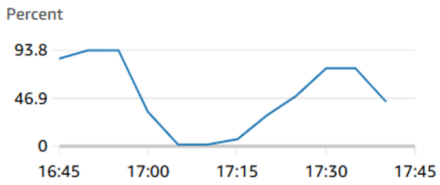
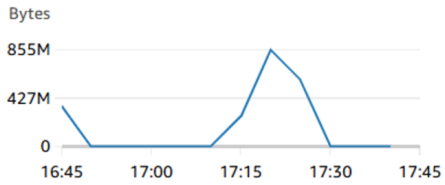
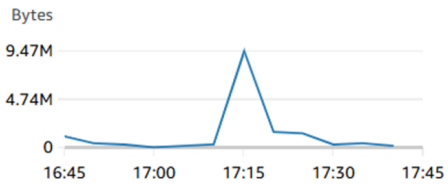
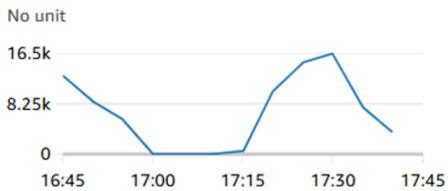
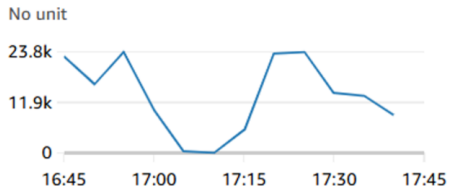
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV)	24.2	47.2 / 80	37.3 / 41.1		315,310	0.0114
Deep CDR	19.9	48.9 / 81.8	32.6 / 37		383,442	0.0094
Proactive DLP	17.1	36.8 / 58.2	30.4 / 33.6		446,228	0.0081
Vulnerability	10.1	59.2 / 85	29.3 / 33.3		755,495	0.0048

MetaDefender Core with common engine packages:

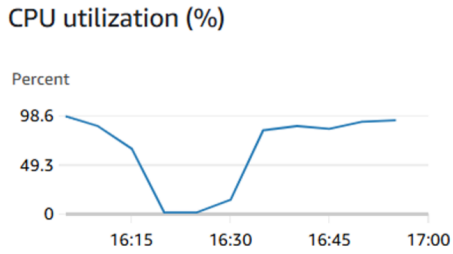
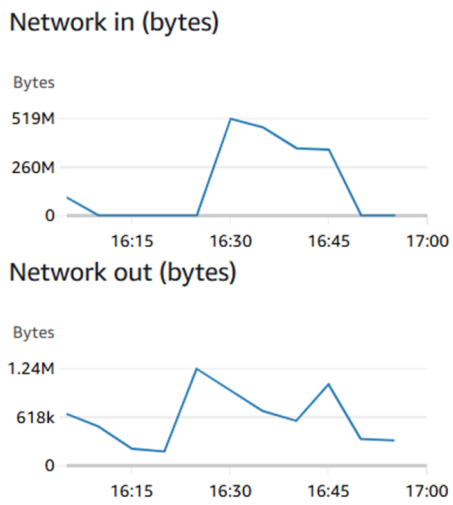
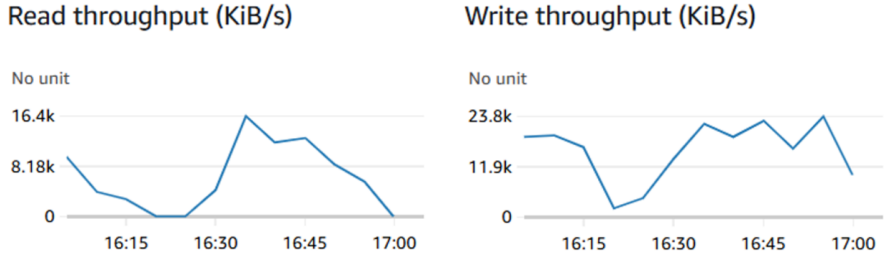
Use case	Scan duration	Avg./Max CPU usage	Avg./Max RAM usage	Avg./Max Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV) + Deep CDR	30.8	68.1 / 88	40.1 / 46.8		247,744	0.0145
Metascan (8 AV) + Deep CDR + Proactive DLP	31.2	74.2 / 98.6	41.5 / 51.1		244,567.3	0.0147
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	31.8	73.5 / 98	40.6 / 51.5		239,953	0.015

System resource utilization charts

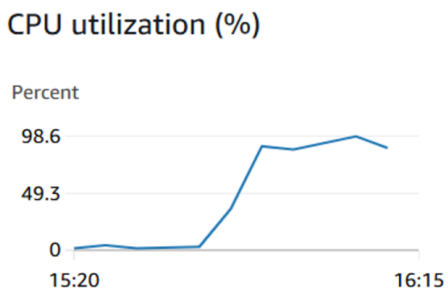
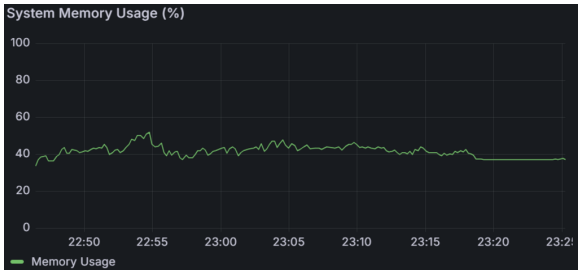
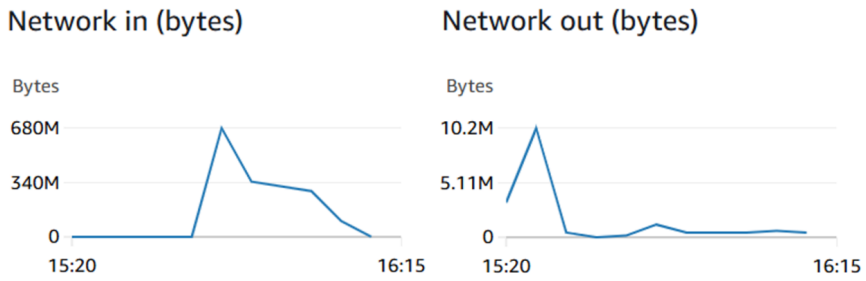
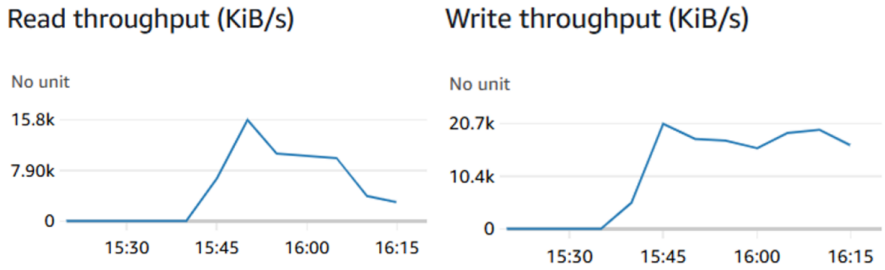
MetaDefender Core with Metascan (8 AV engines):

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

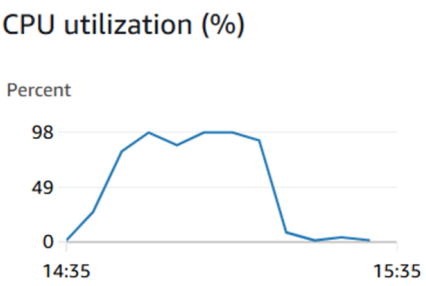
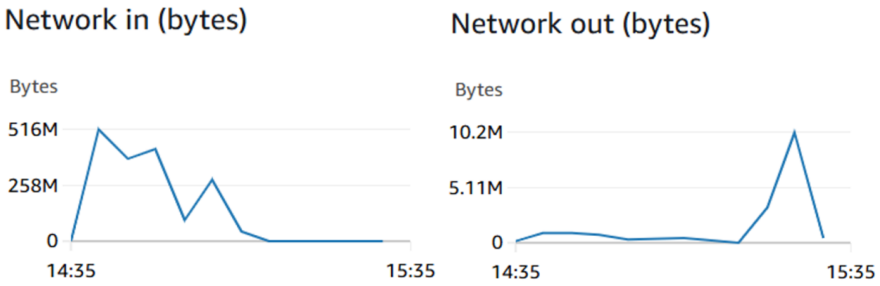
MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

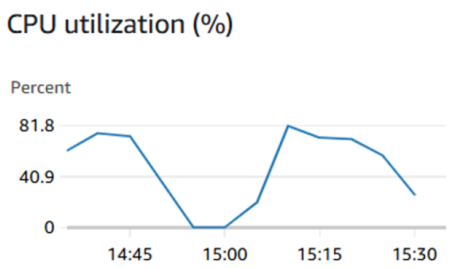
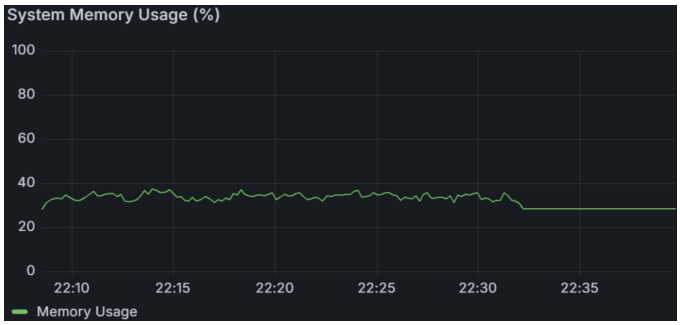
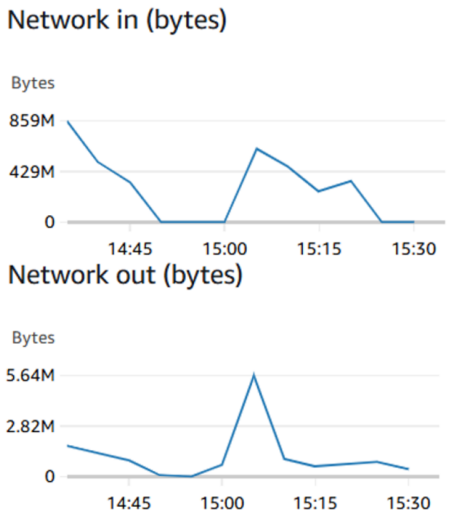
MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

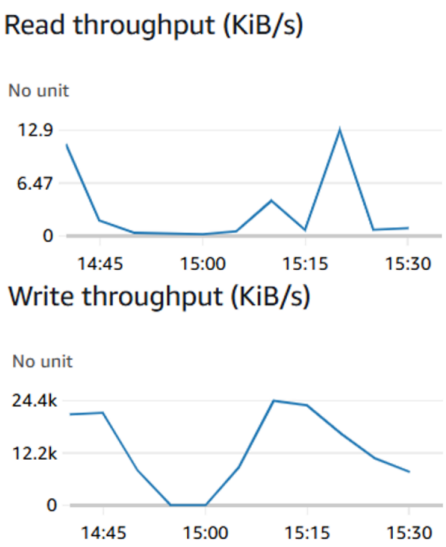
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

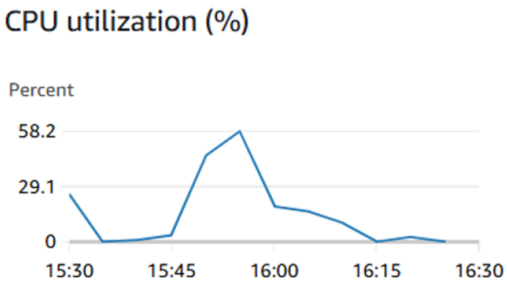
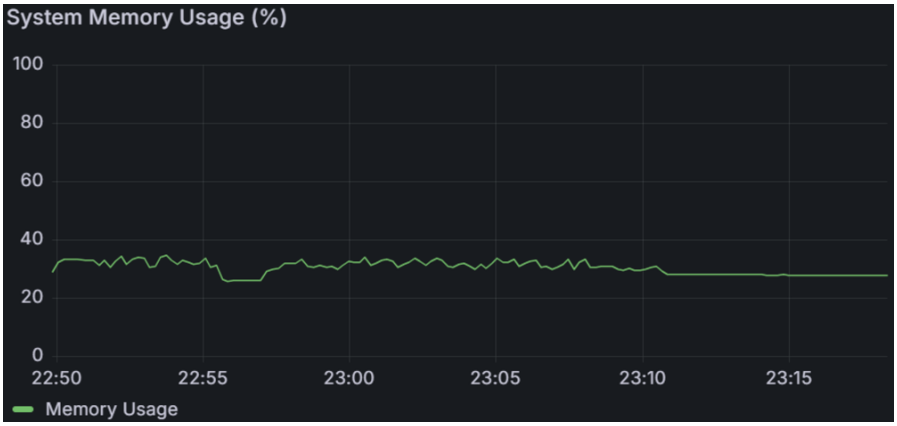
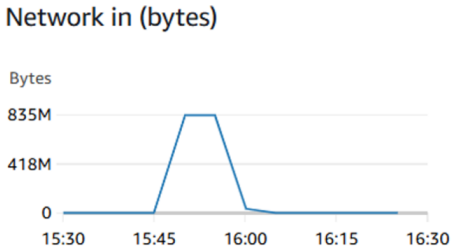
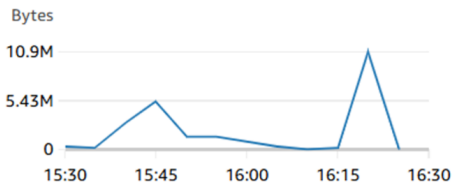
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

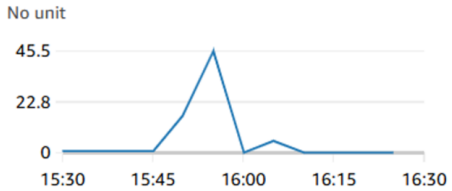
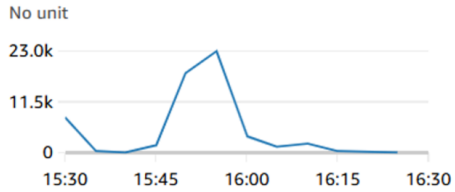
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

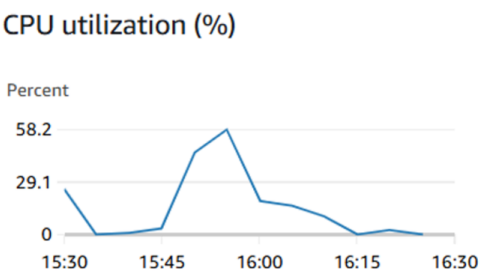
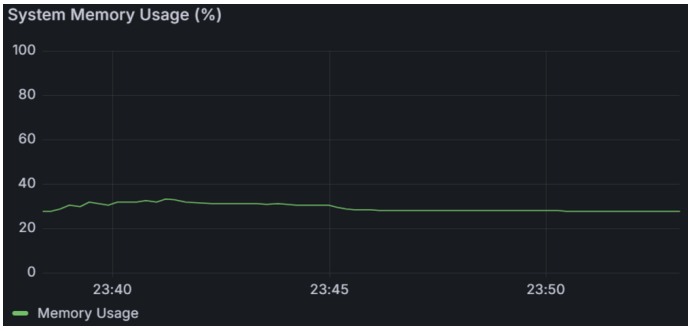
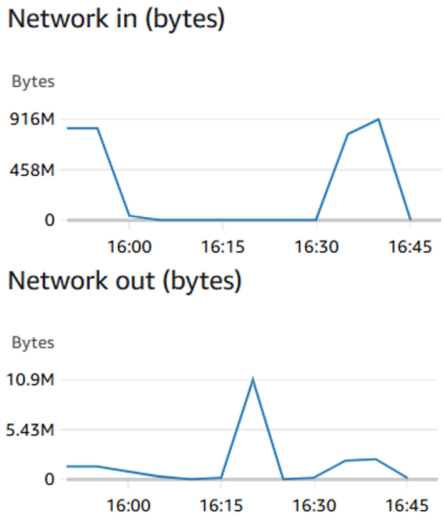
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	 

	Utilization charts
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

MetaDefender Core with Vulnerability:

	Utilization charts
CPU usage	
Memory usage	
Network usage	

Utilization charts

Disk usage



Standalone DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Ubuntu 22.04	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.57-236

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

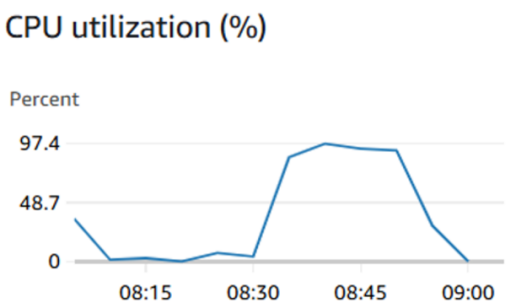
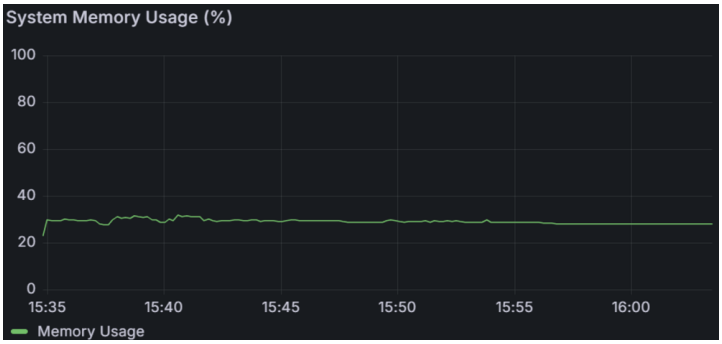
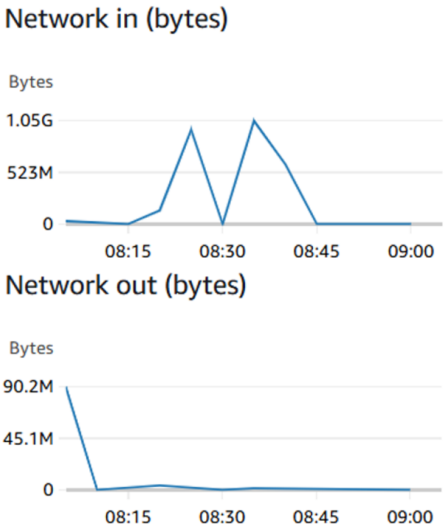
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV)	30.2	69 / 98	29 / 32		252,666	0.0142
Deep CDR	19.4	48 / 62	36 / 49		393,325	0.009
Proactive DLP	16.5	34 / 39	32 / 48		462,455	0.008
Vulnerability	12	50 / 65	46 / 46		635,875	0.006

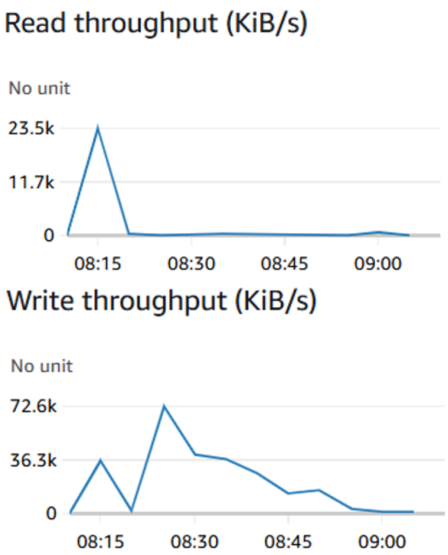
MetaDefender Core with common engine packages:

Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV) + Deep CDR	39.8	78 / 99	62 / 73		191,721	0.0188
Metascan (10 AV) + Deep CDR + Proactive DLP	43.5	61 / 84	47 / 67		175,414	0.0205
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	44.8	62 / 88	69 / 83		170,324	0.0211

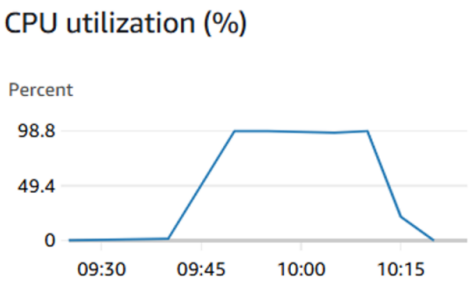
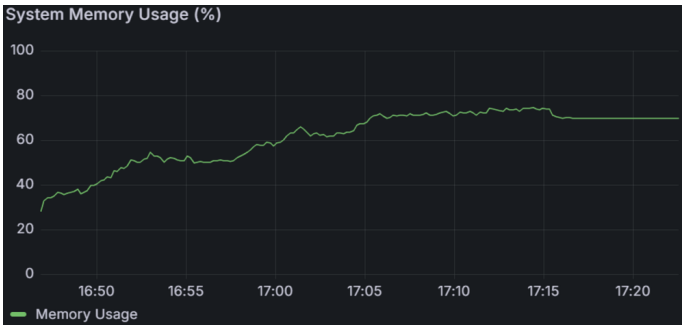
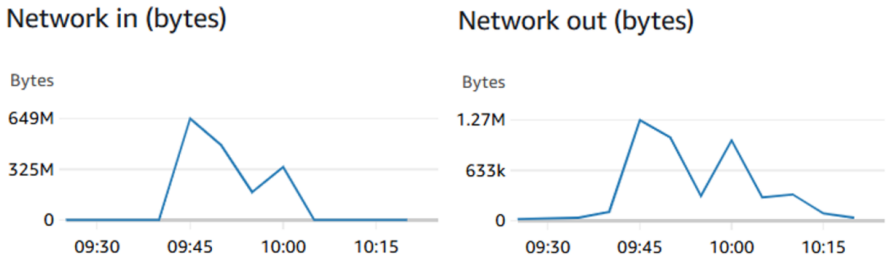
System resource utilization charts

MetaDefender Core with Metascan (10 AV engines) :

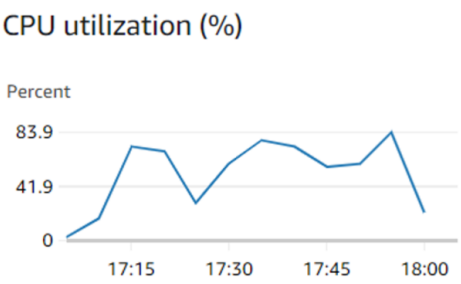
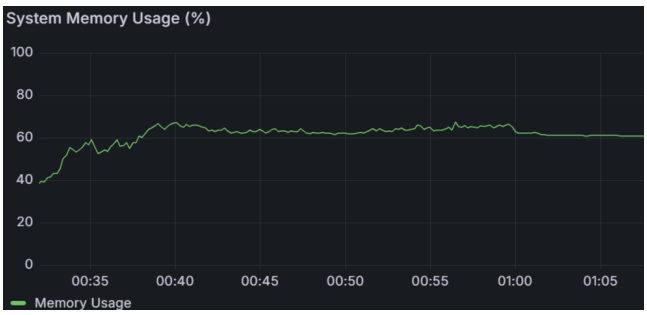
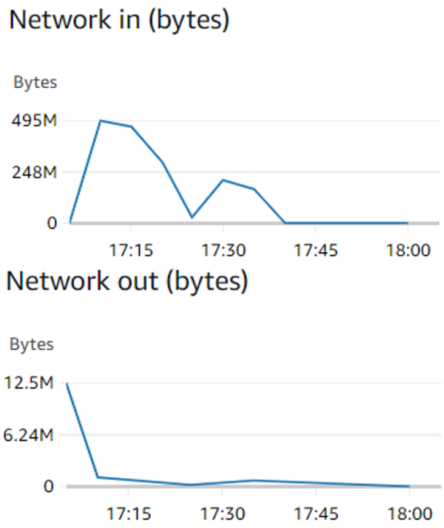
	Utilization charts
CPU usage	
Memory usage	
Network usage	

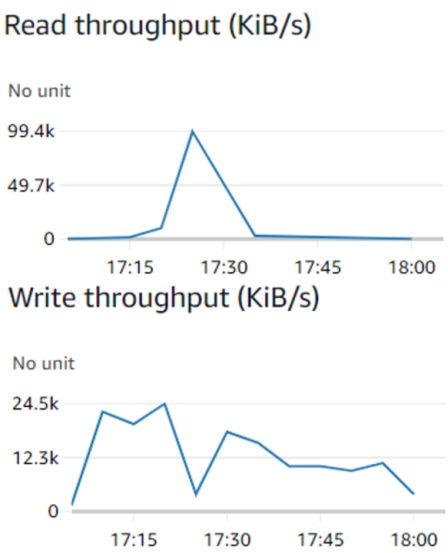
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR :

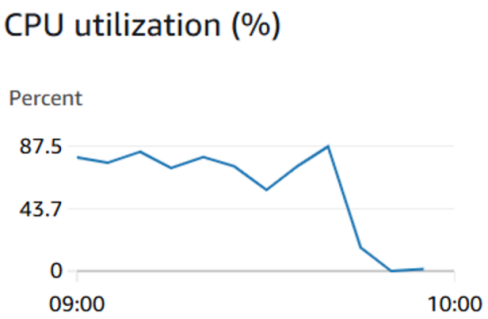
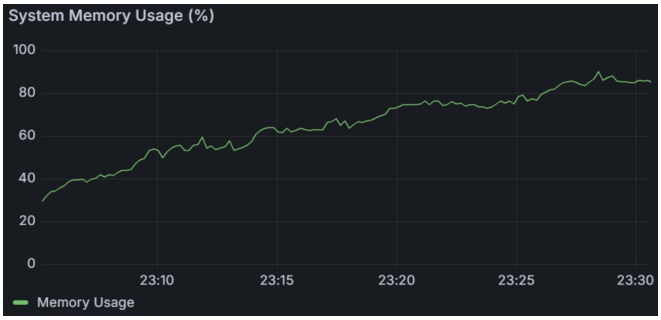
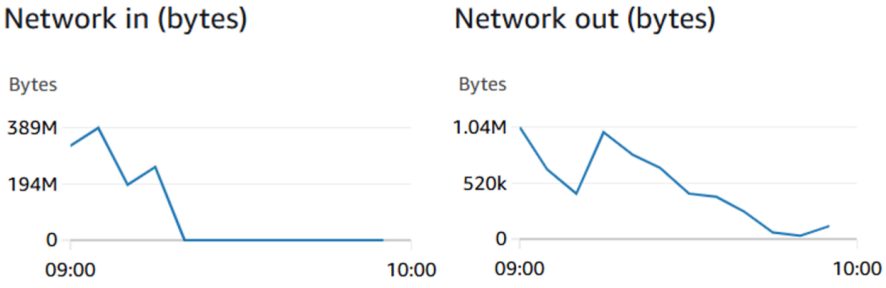
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP :

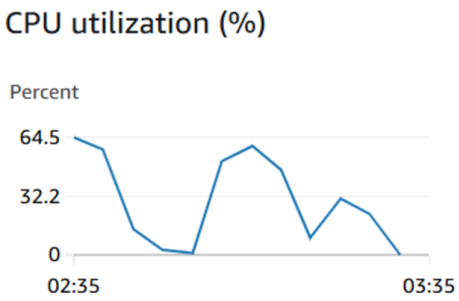
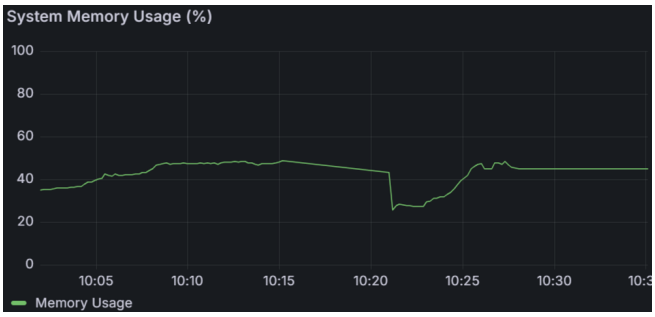
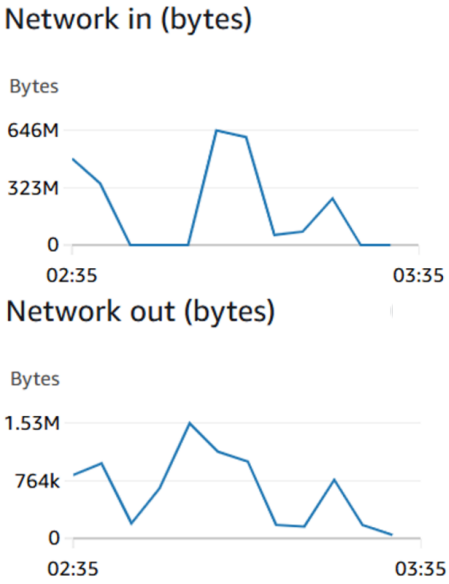
	Utilization charts
CPU usage	
Memory usage	
Network usage	

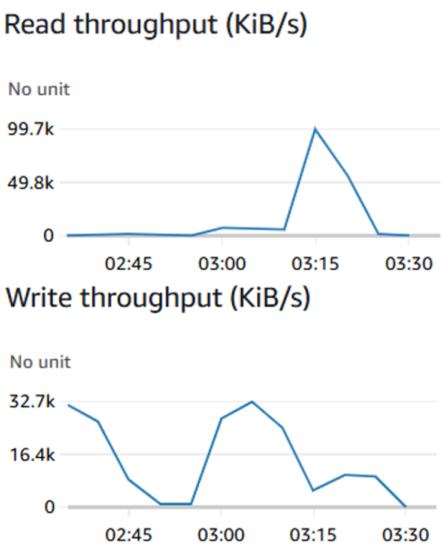
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP + Vulnerability :

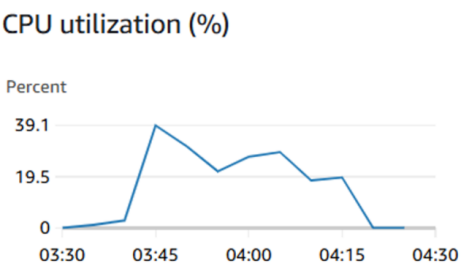
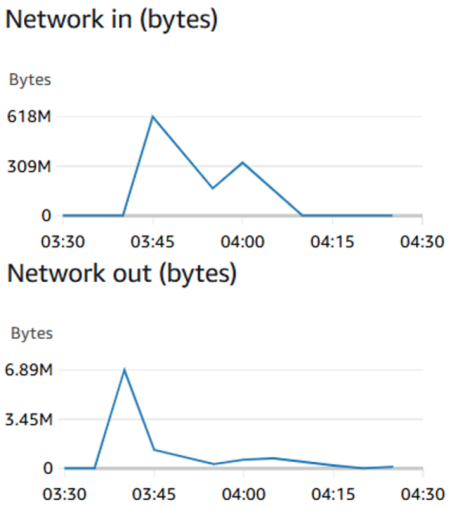
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

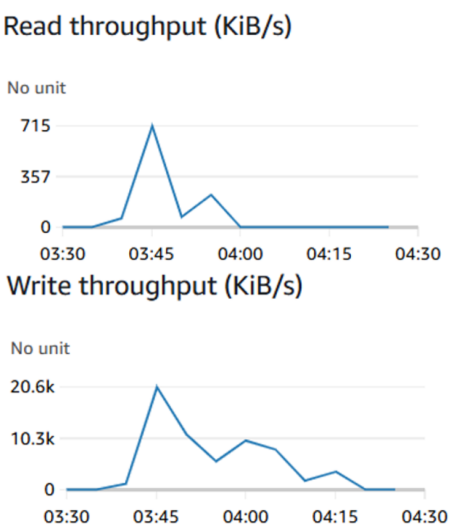
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

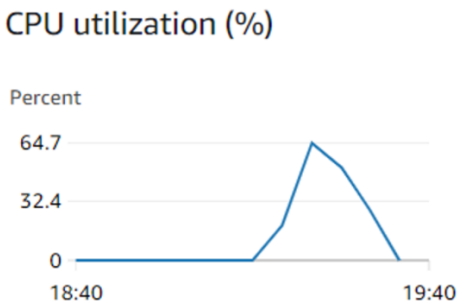
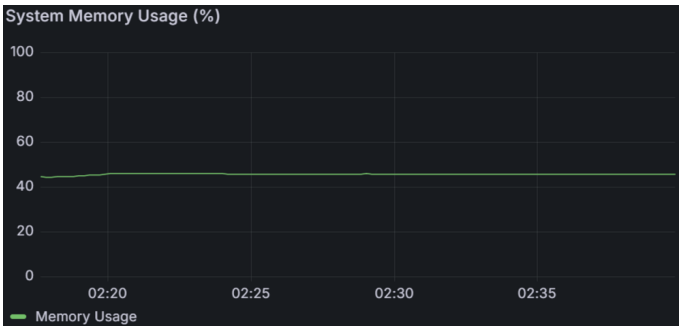
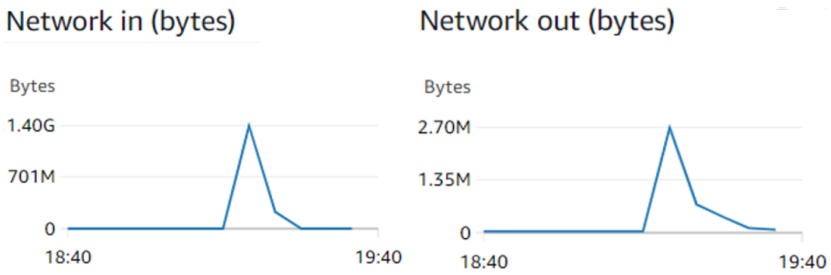
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability :

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

Shared DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

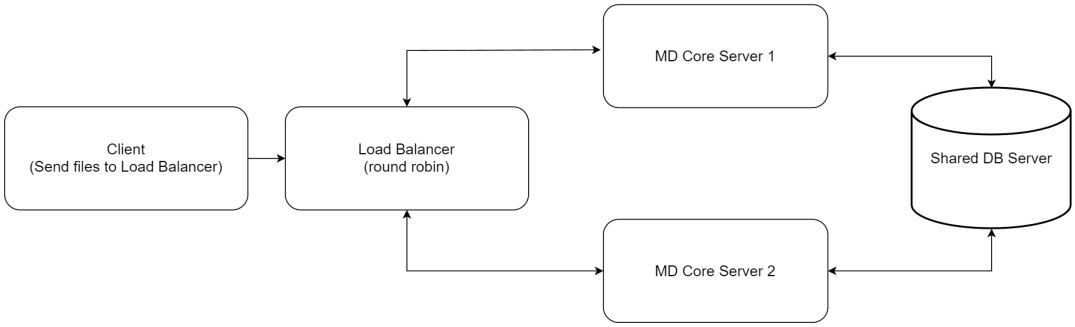
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV)	11.9	37.0	41.9	30.1	30.4	Receive: 6441.1 Send: 769.6	Receive: 6200.2 Send: 693.7	640,821.2	0.0
Deep CDR	10.5	60.1	66.7	30.3	32.2	Receive: 6379.8 Send: 594.8	Receive: 7146.1 Send: 609.6	729,368.5	0.0
Proactive DLP	7.8	38.7	37.5	29.7	31.0	Receive: 8842.6 Send: 791.8	Receive: 8635.6 Send: 729.1	965,455.7	0.0
Vulnerability	4.4	52.2	53.0	26.0	26.9	Receive: 16,215.5 Send: 1006.8	Receive: 15,785.7 Send: 997.2	1714,449.3	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV) + Deep CDR	15.4	72.4	75	34.3	35.4	Receive: 5436.6 Send: 611.3	Receive: 5589.3 Send: 607.2	495,697.4	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP	16.9	77.2	80.2	36.4	37.9	Receive: 5254 Send: 585.4	Receive: 5041.3 Send: 576.1	450,657.6	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	18.0	72.1	80.4	38.6	38.0	Receive: 4504.3 Send: 574.1	Receive: 4649.5 Send: 571.1	423,010	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

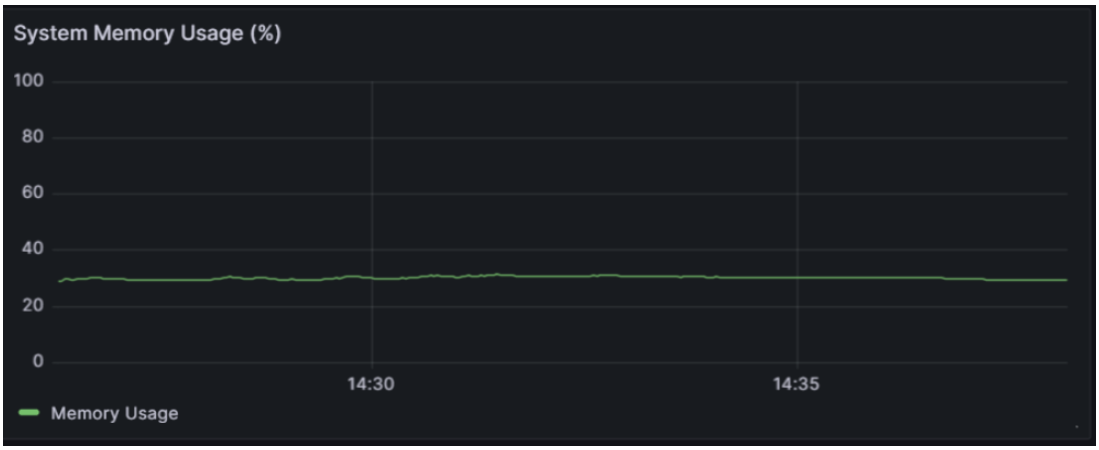
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2

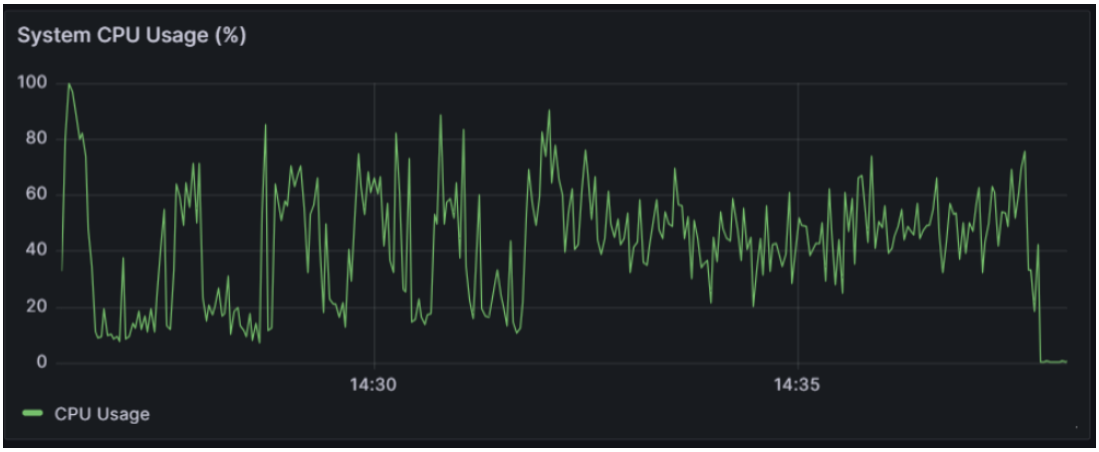


CPU usage

MD Core #1

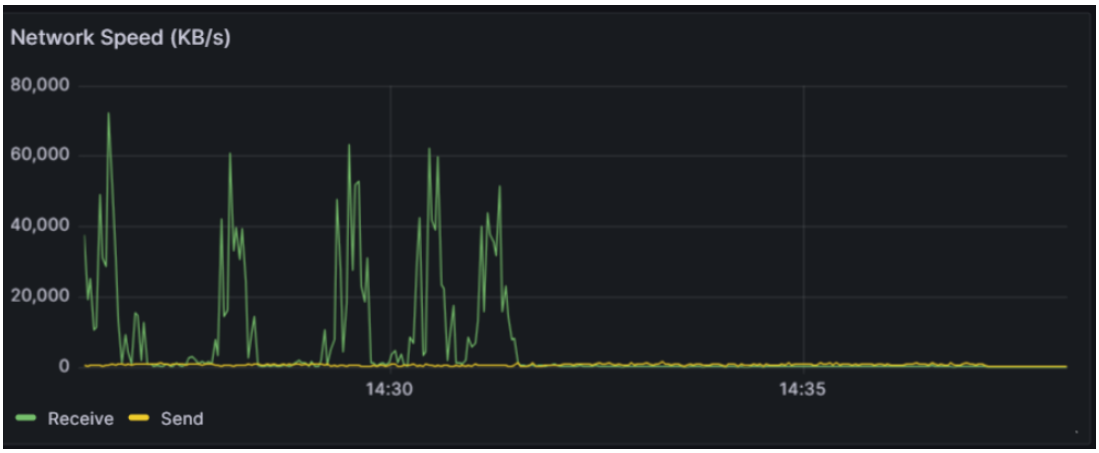


MD Core #2

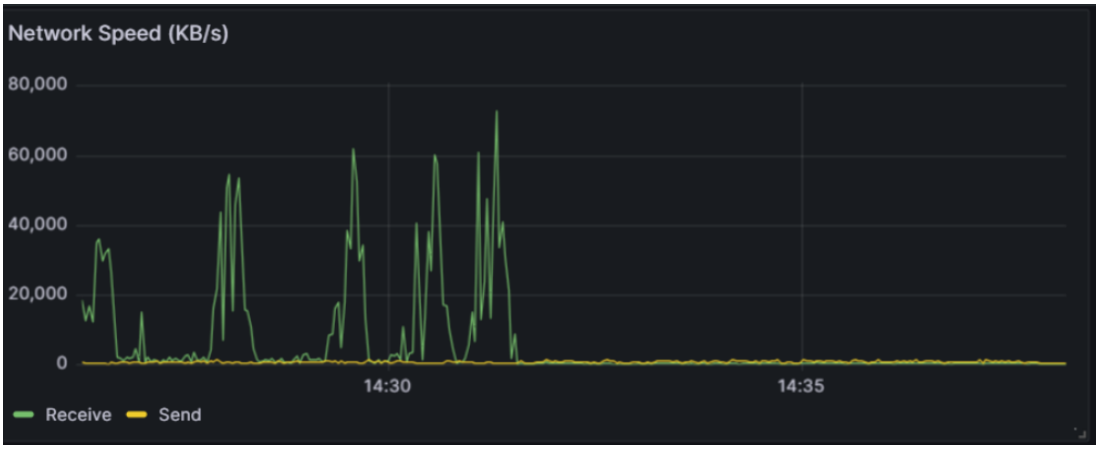


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2

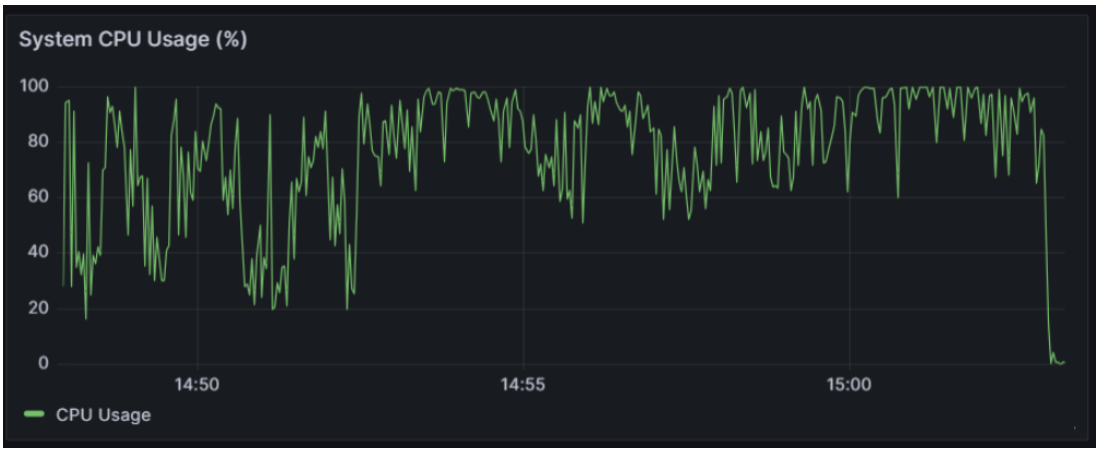


CPU usage

MD Core #1

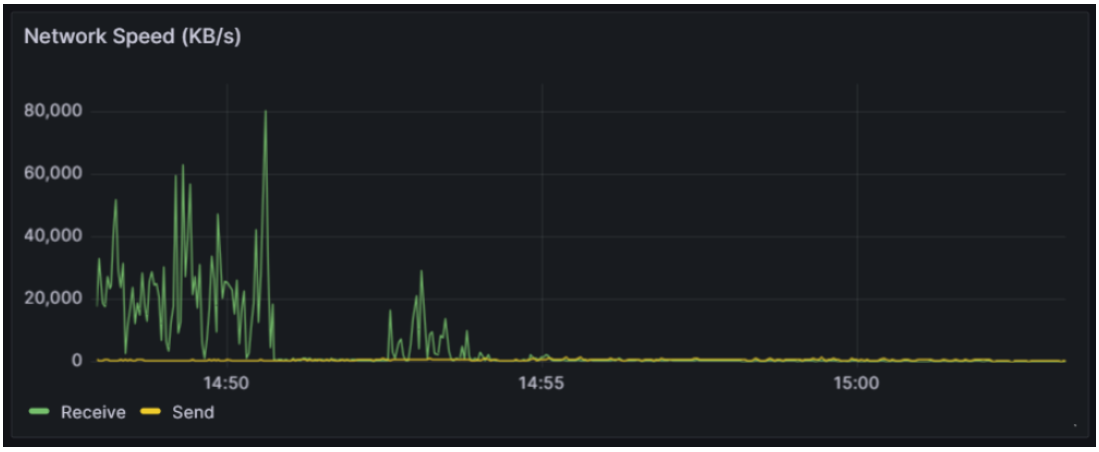


MD Core #2



Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

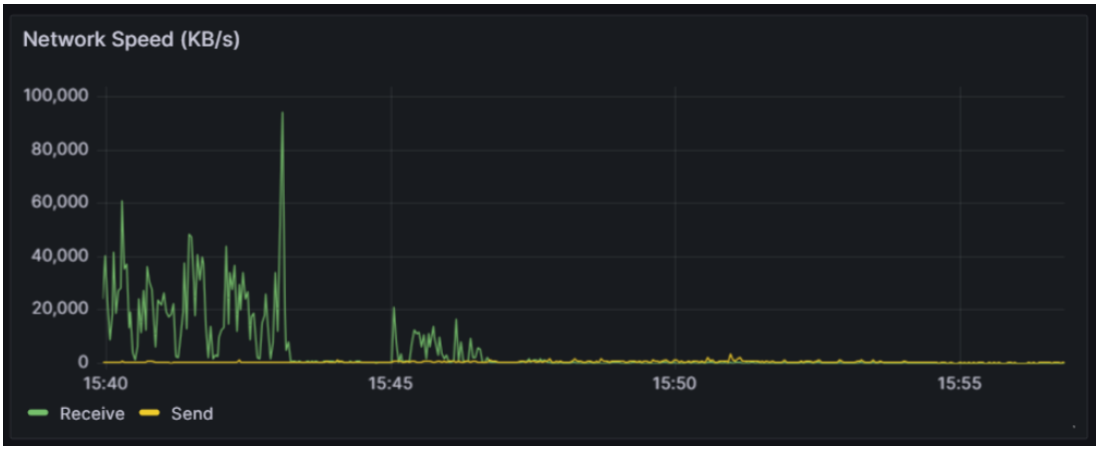


MD Core #2

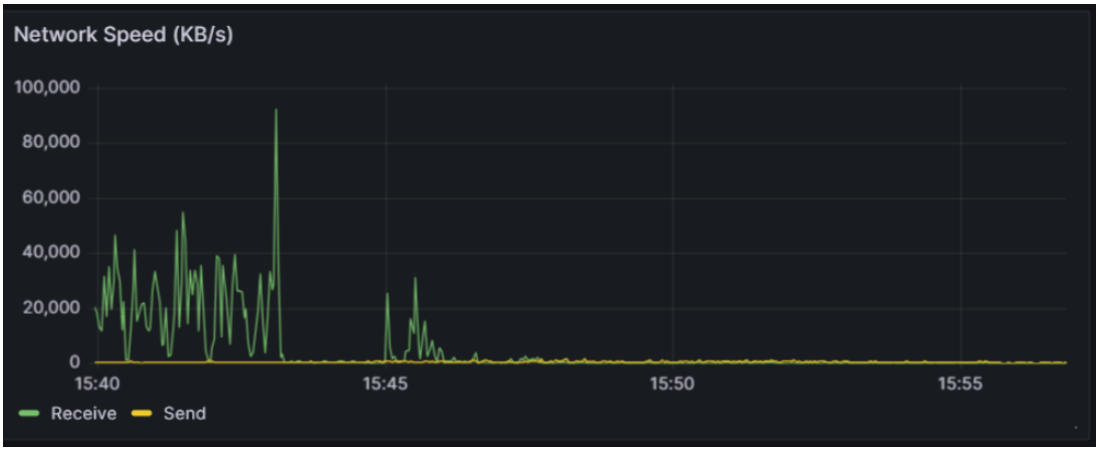


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1

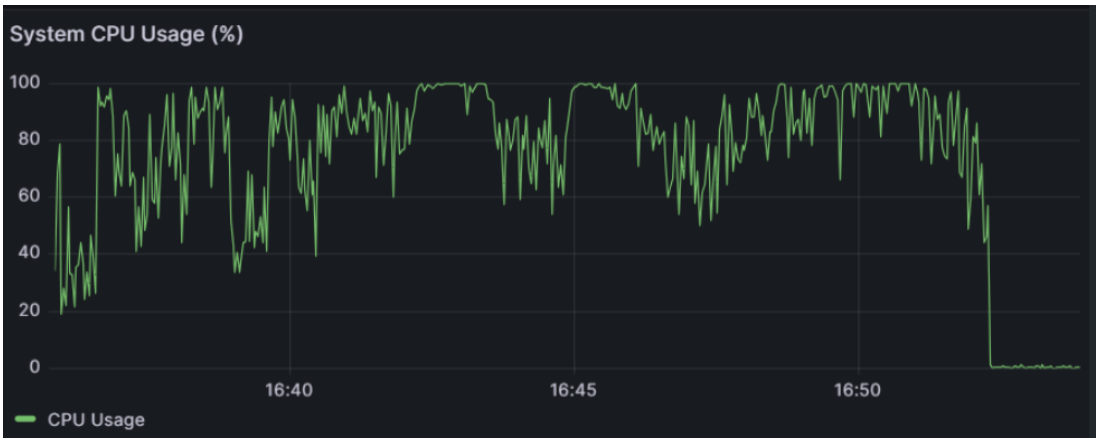


MD Core #2



CPU usage

MD Core #1

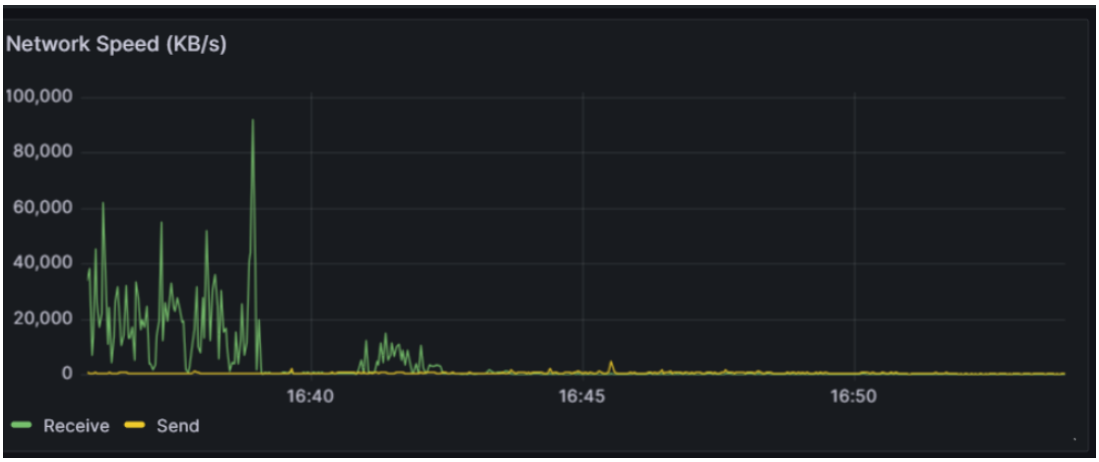


MD Core #2

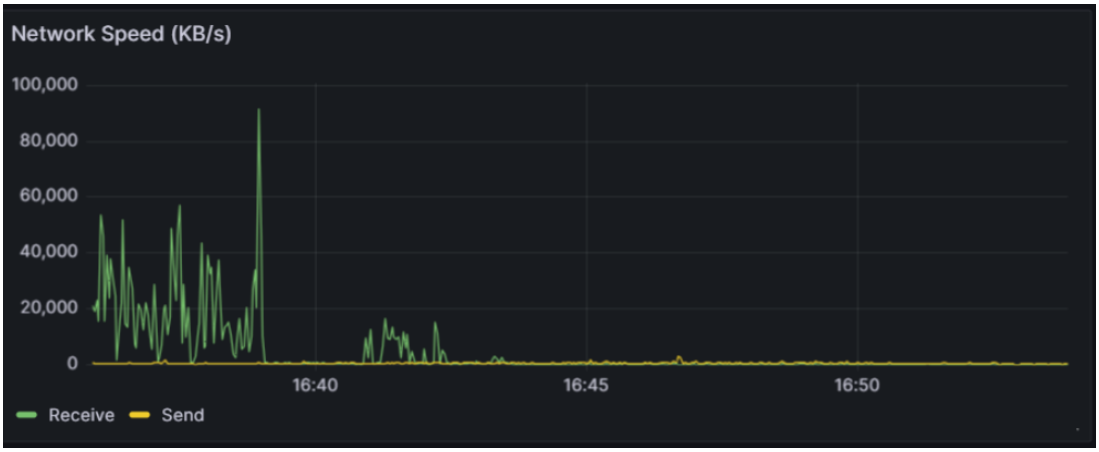


Network usage

MD Core #1



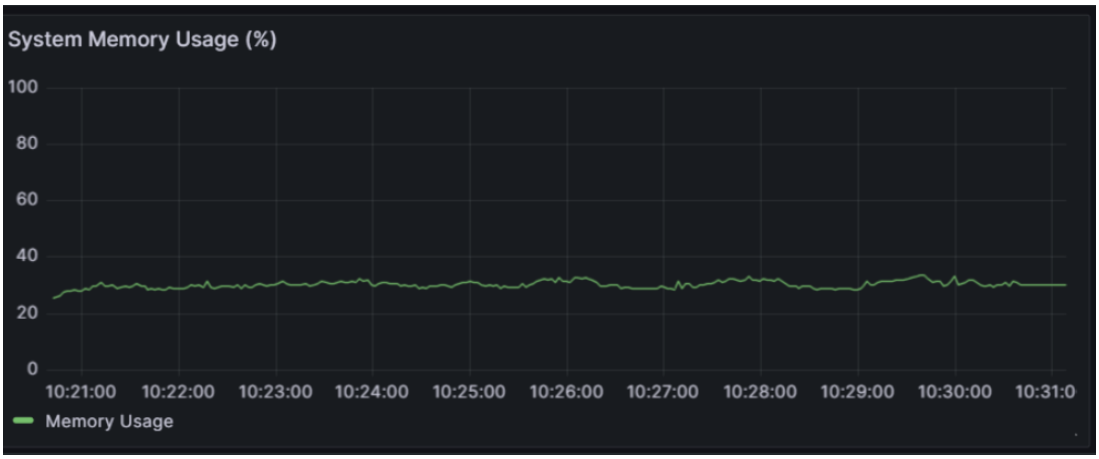
MD Core #2



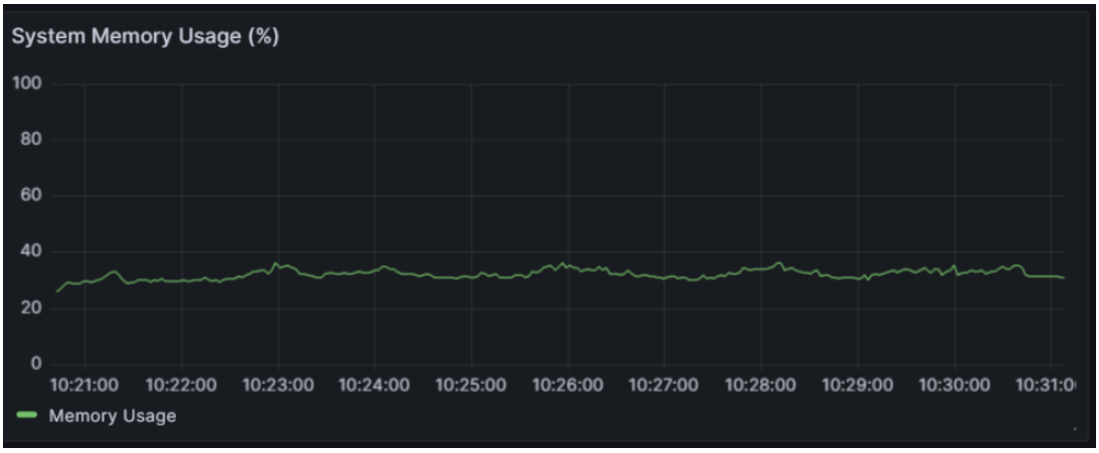
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

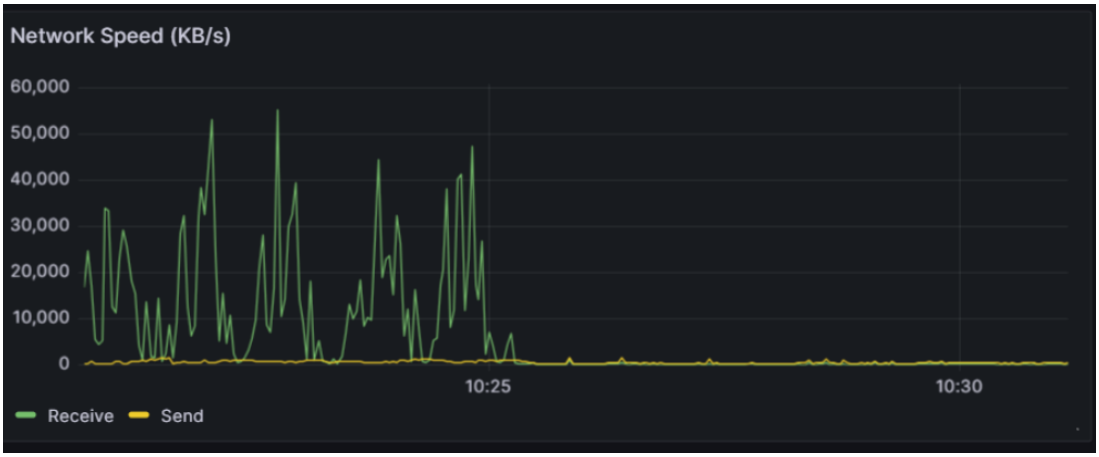


MD Core #2

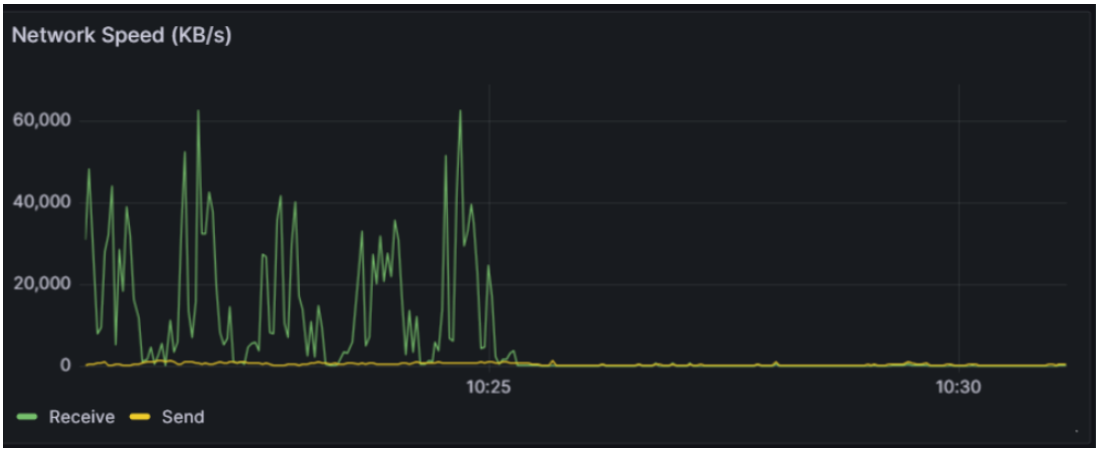


Network usage

MD Core #1



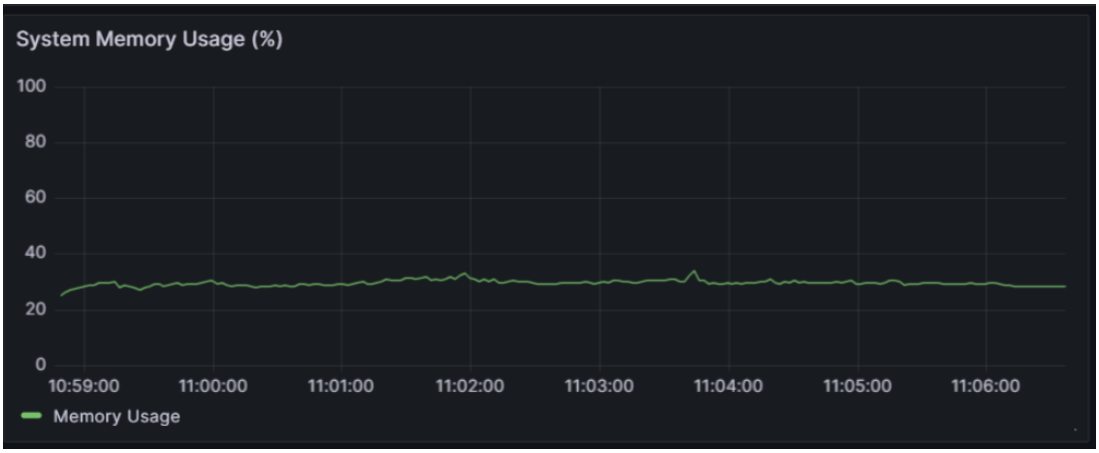
MD Core #2



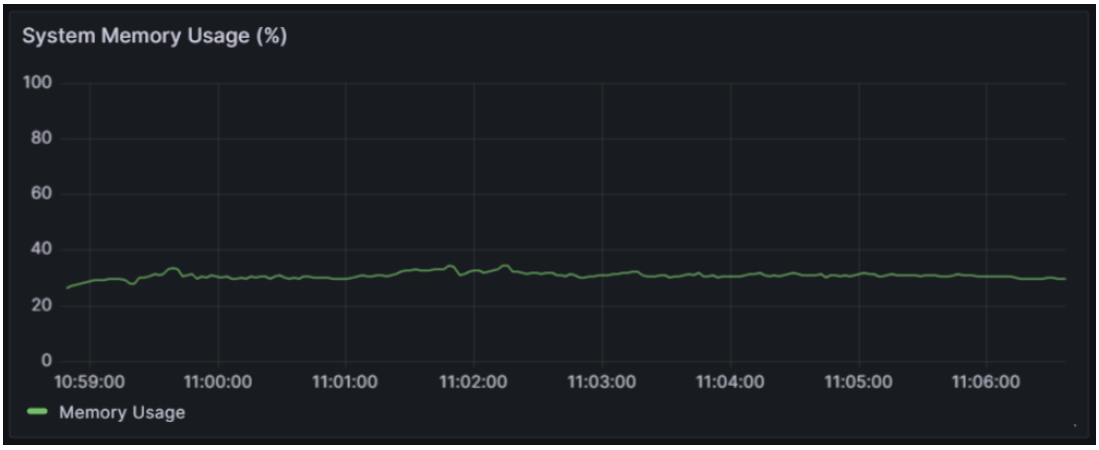
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

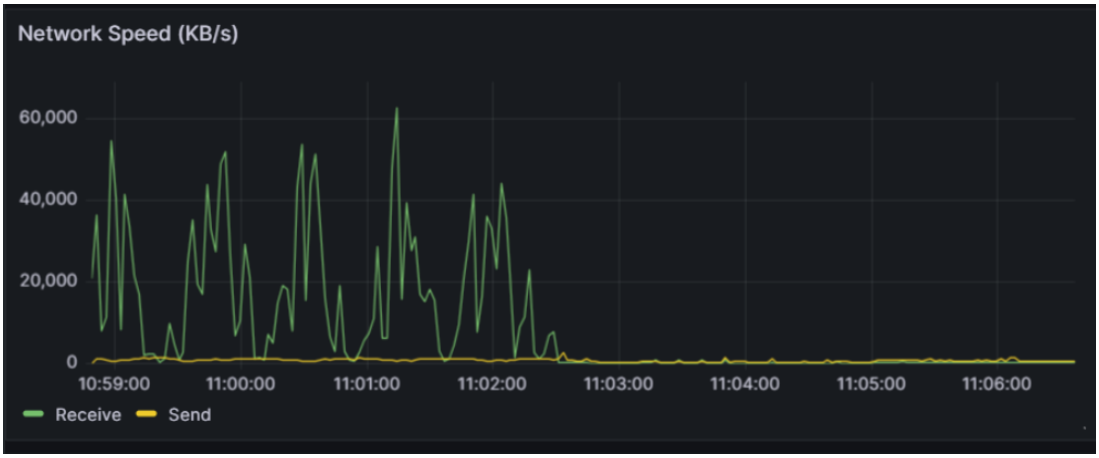


MD Core #2

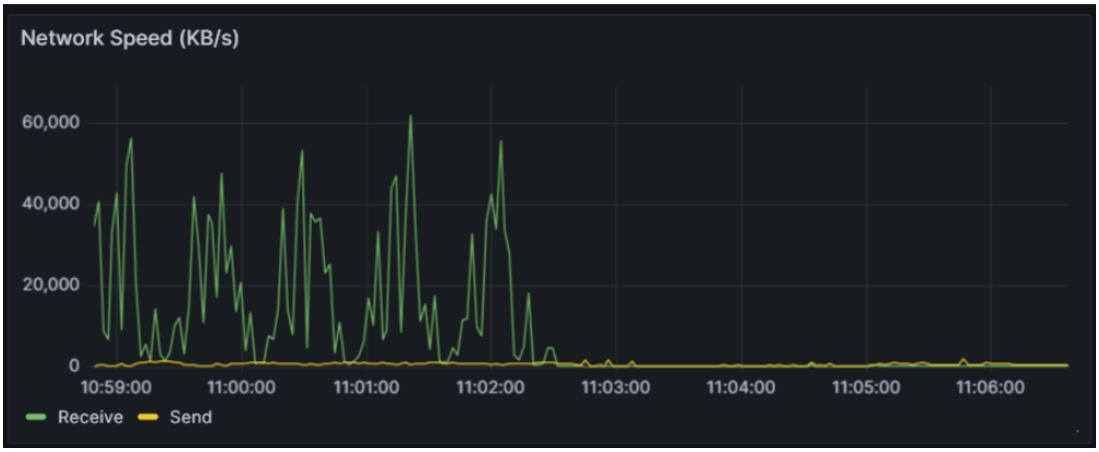


Network usage

MD Core #1



MD Core #2



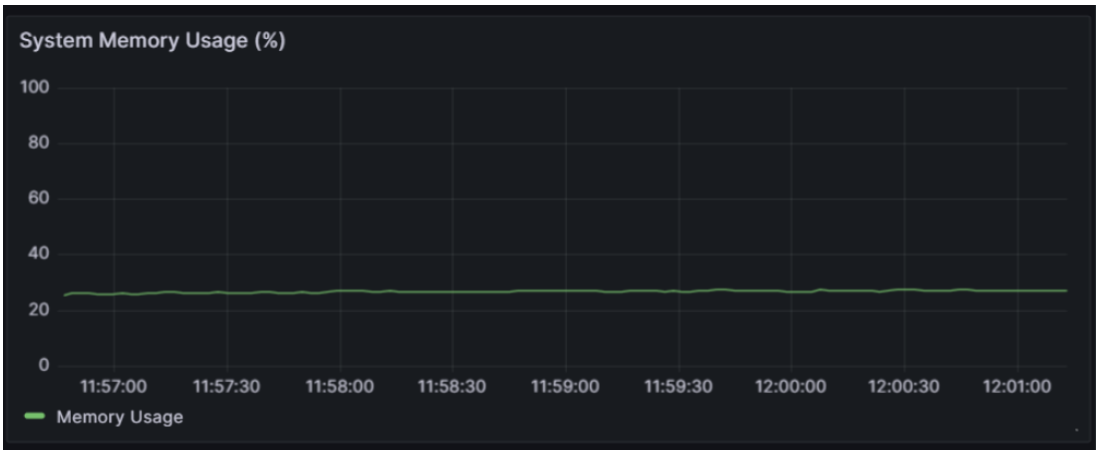
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

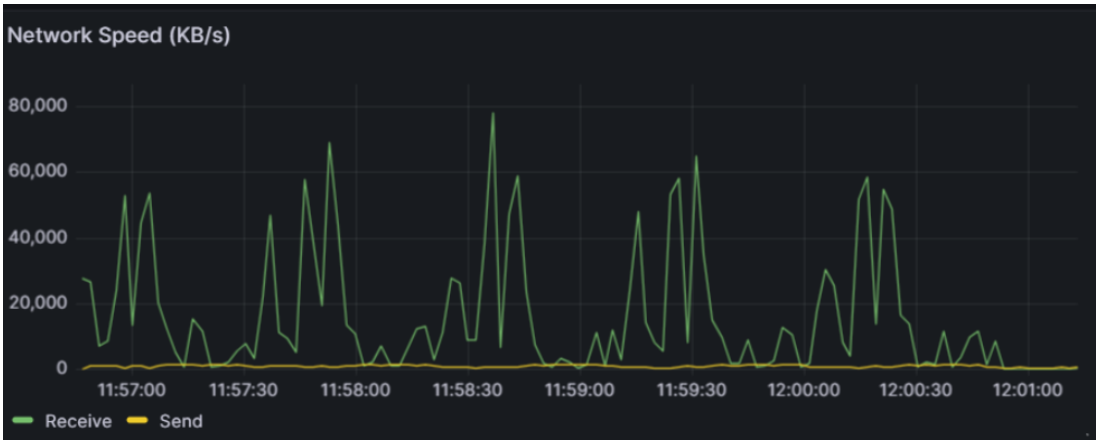


MD Core #2

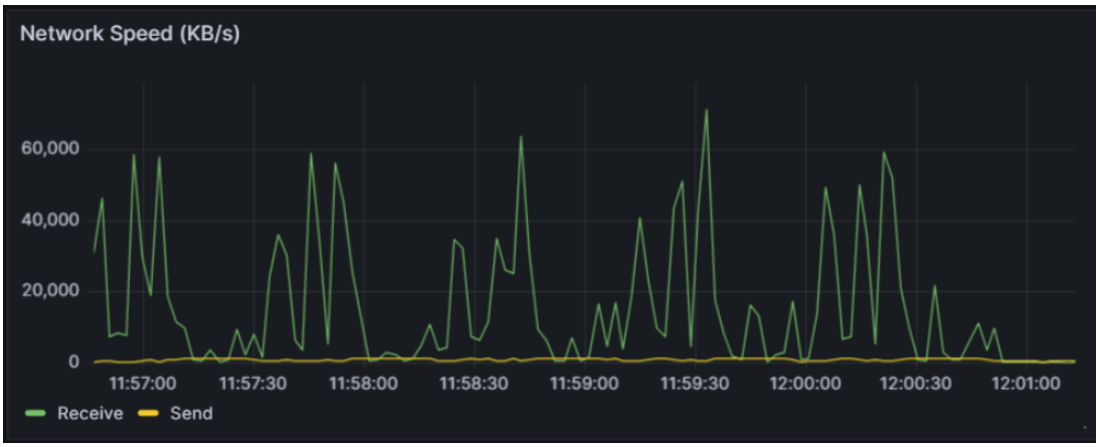


Network usage

MD Core #1



MD Core #2



Shared DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

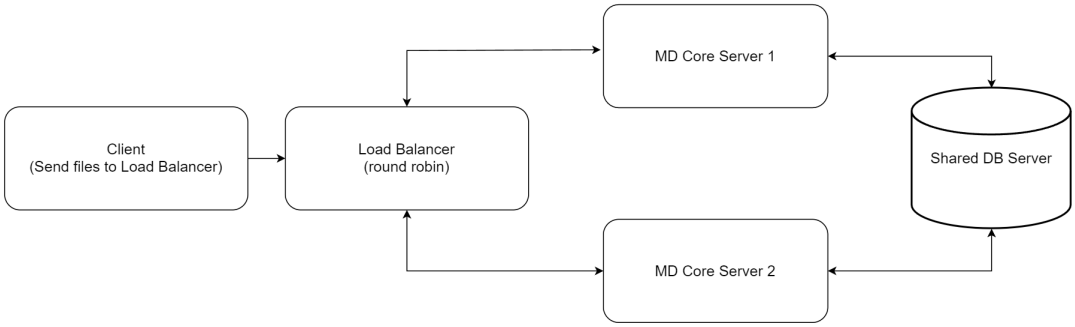
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV)	15.5	63.1	73.3	25.2	25.8	Receive: 4555.9 Send: 694.4	Receive: 4516.2 Send: 692.4	492,620.5	0.0
Deep CDR	9.70	50.7	51.2	31.9	32.5	Receive: 7062.9 Send: 623.2	Receive: 6653 Send: 654.1	786,189.6	0.0
Proactive DLP	7.2	35.2	35.0	27.9	27.5	Receive: 9047.4 Send: 802.4	Receive: 9859.8 Send: 787.5	1,055,110.8	0.0
Vulnerability	5.0	45.7	49.3	24.4	24.5	Receive: 13,585.7 Send: 966.7	Receive: 13,505.8 Send: 987.4	1,519,562.1	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV) + Deep CDR	19.9	79.1	77.6	34.0	34.2	Receive: 3785.9 Send: 573.8	Receive: 3663.1 Send: 577.7	382,716	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP	21.6	80.3	81.5	37.7	44.4	Receive: 3305.6 Send: 543.3	Receive: 3414.3 Send: 556.2	353,694.9	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	22.5	84.2	79.8	37.9	44.7	Receive: 3267.9 Send: 554.9	Receive: 3398.2 Send: 553.8	338,507.6	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

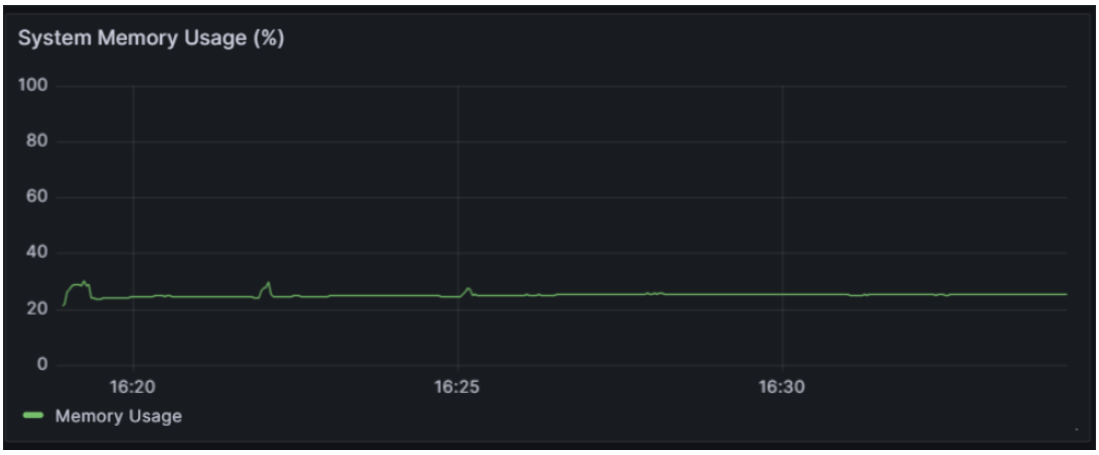
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

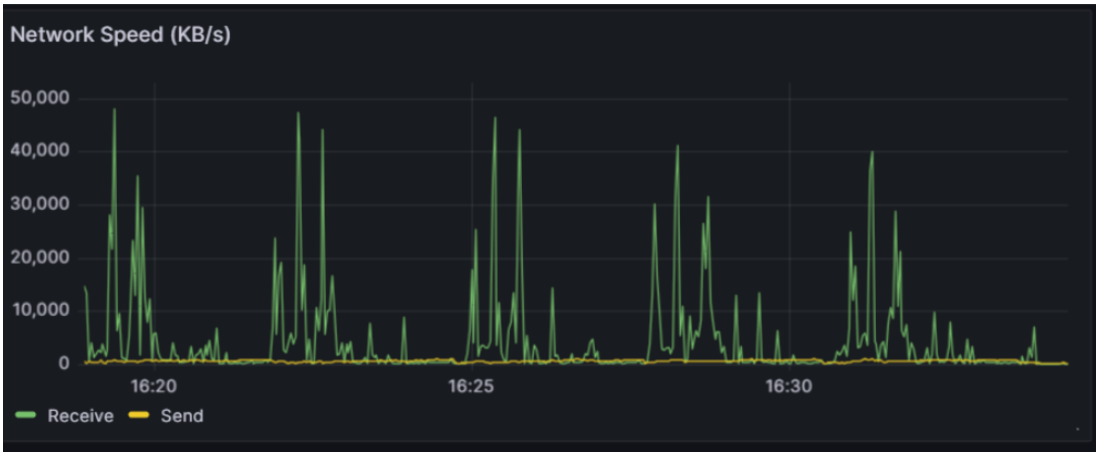


MD Core #2

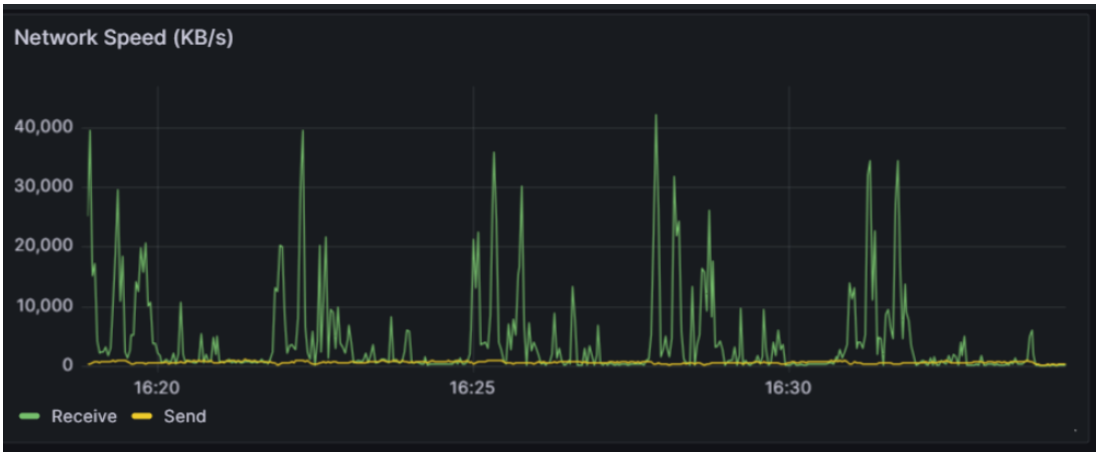


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

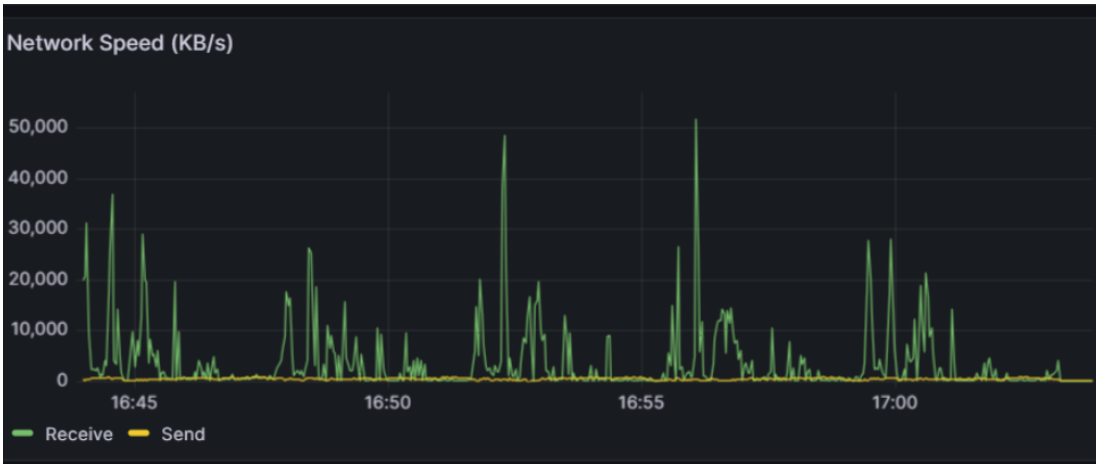


MD Core #2

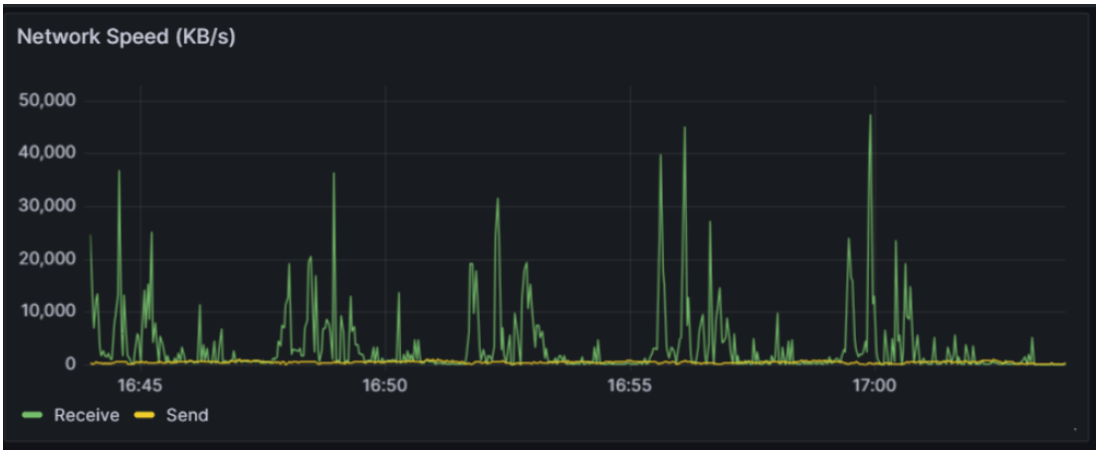


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

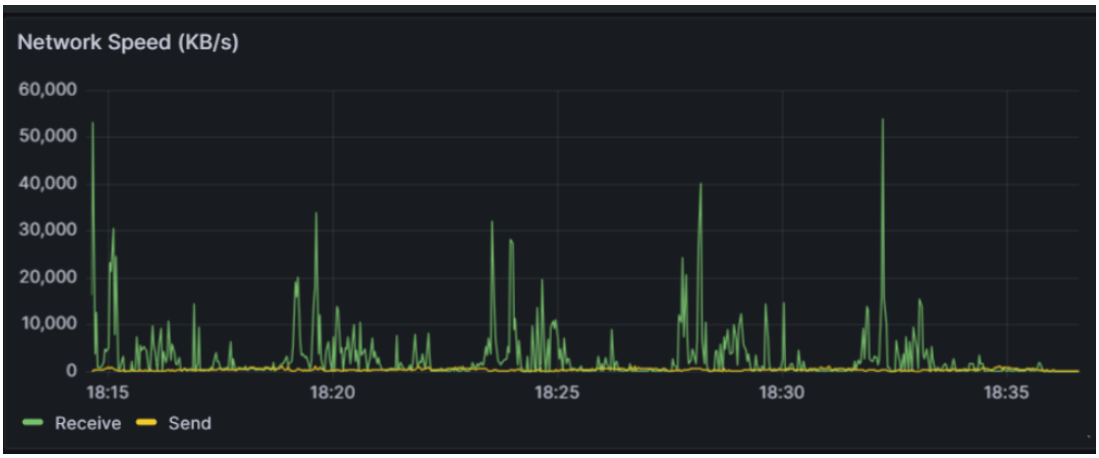


MD Core #2

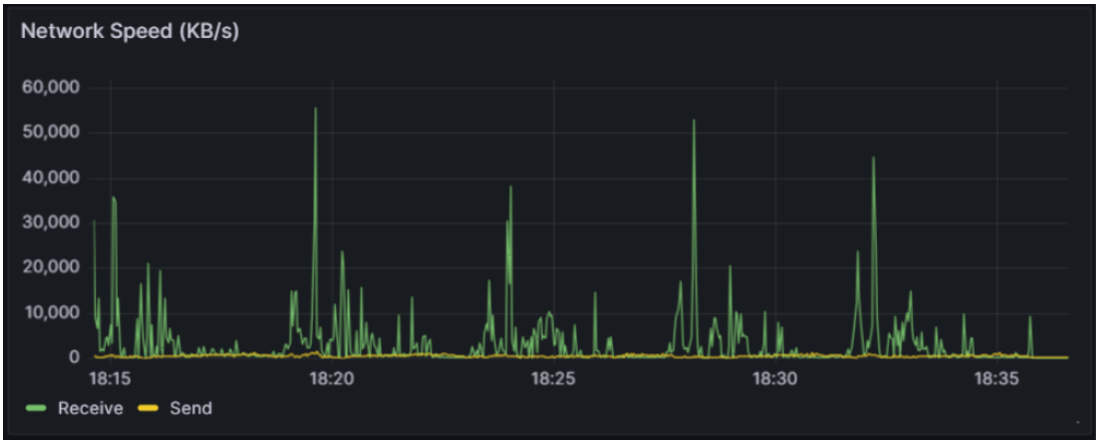


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

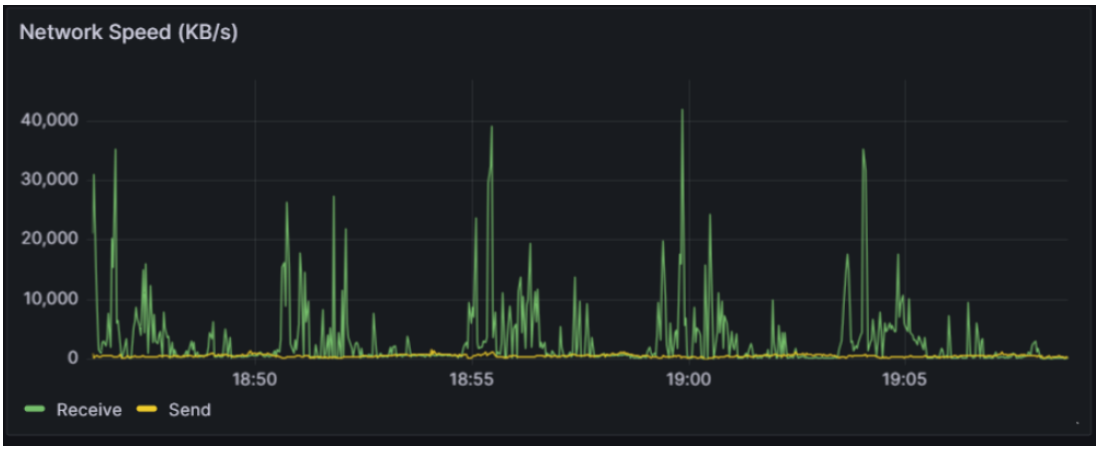


MD Core #2

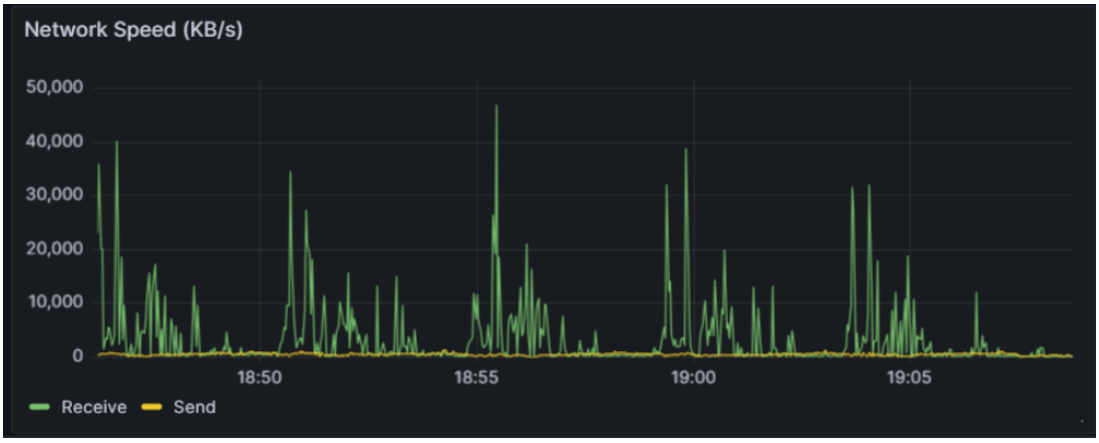


Network usage

MD Core #1



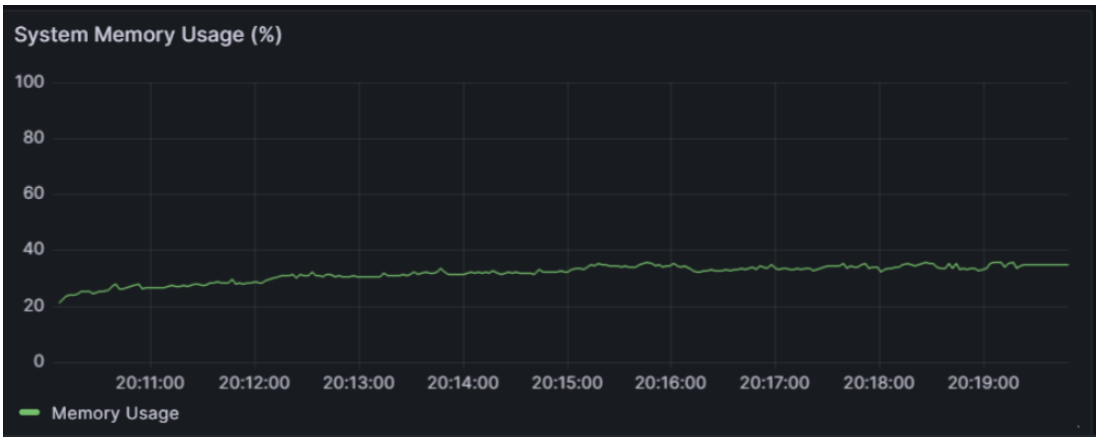
MD Core #2



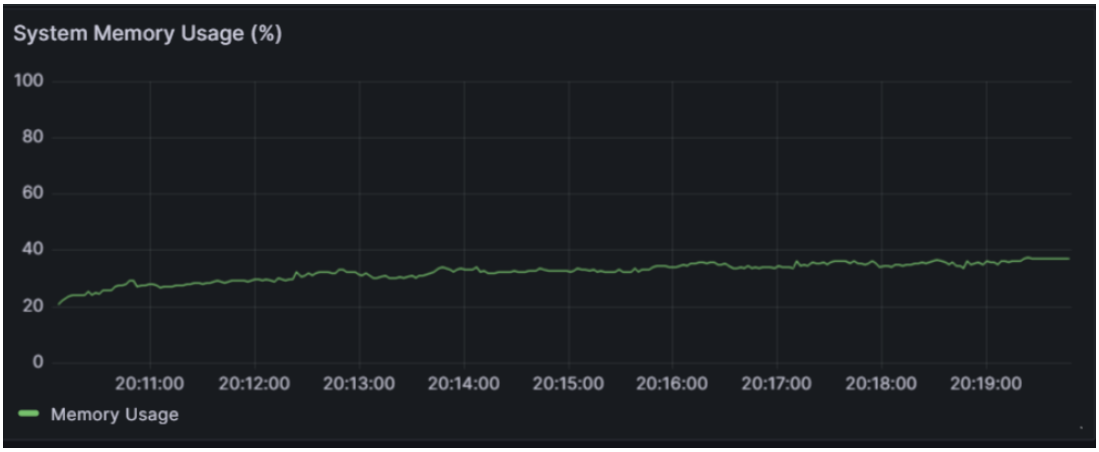
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1

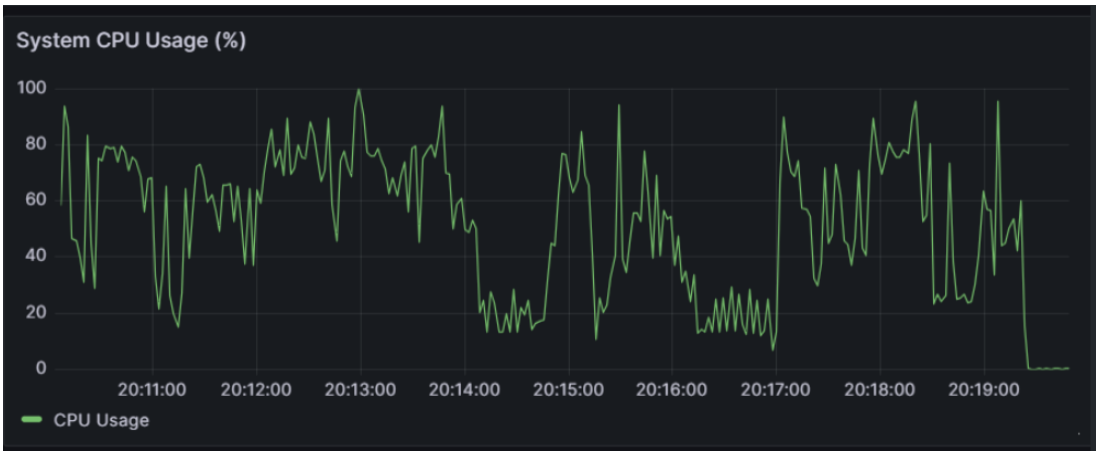


MD Core #2



CPU usage

MD Core #1

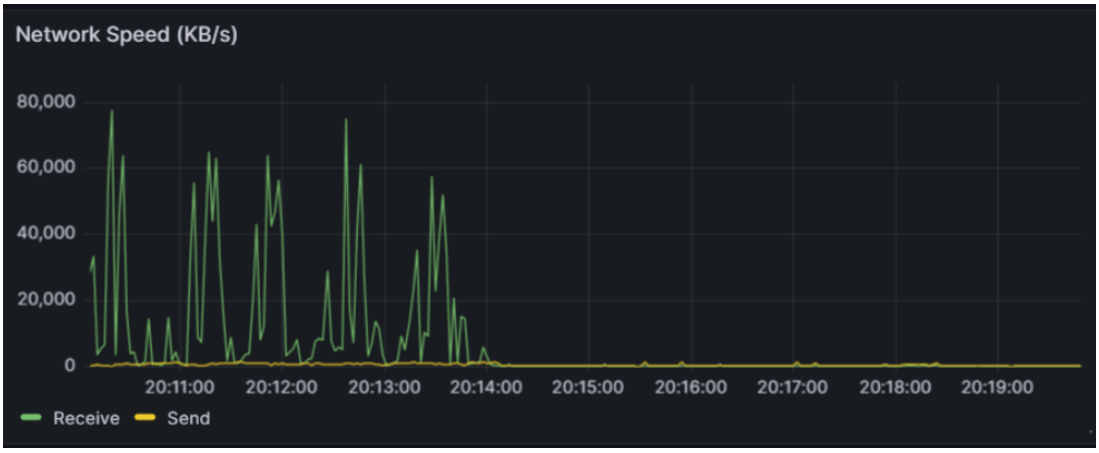


MD Core #2

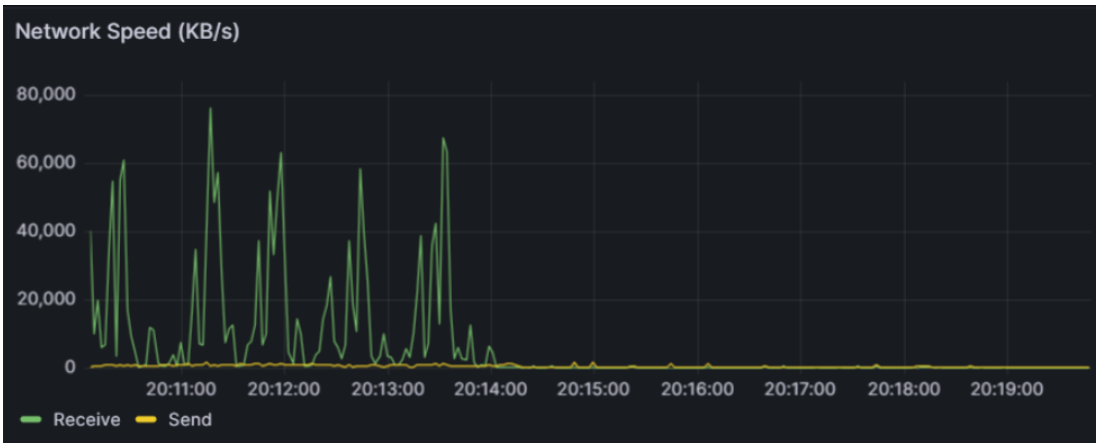


Network usage

MD Core #1



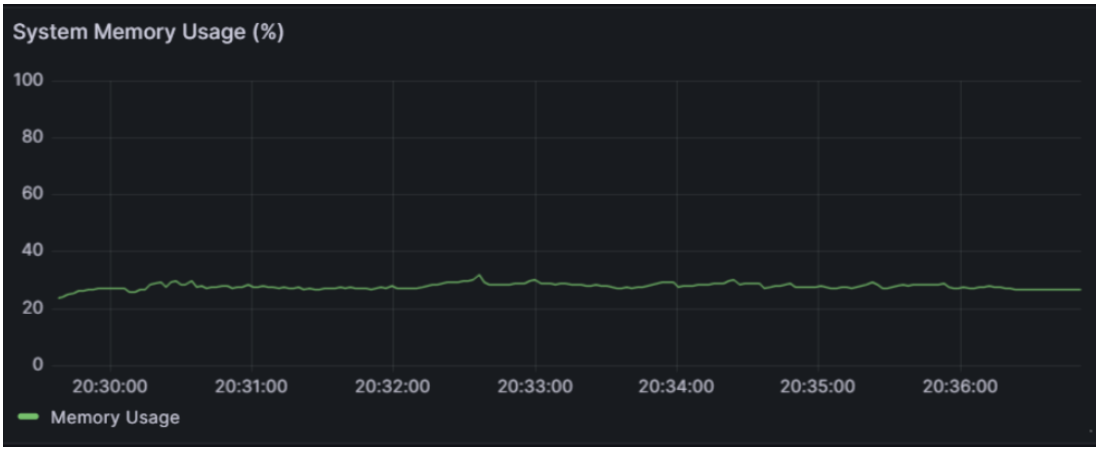
MD Core #2



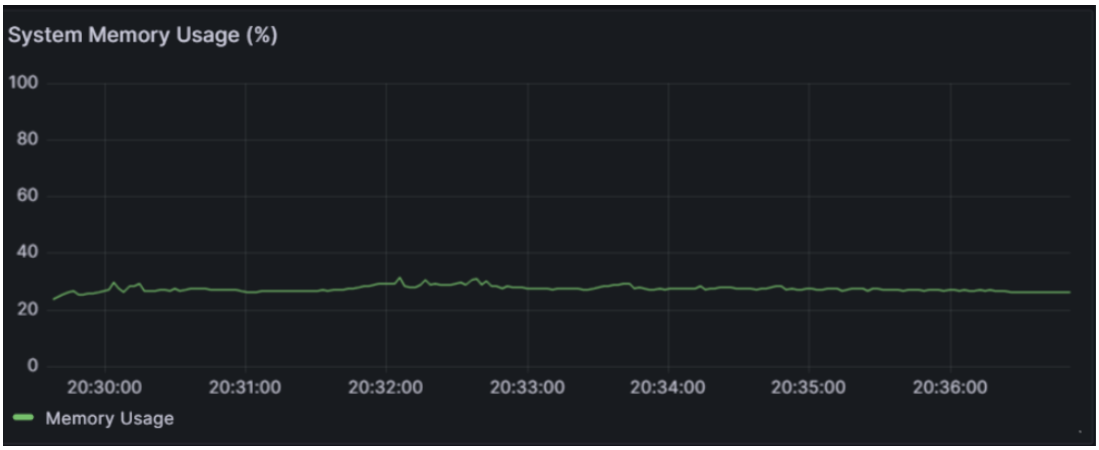
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1

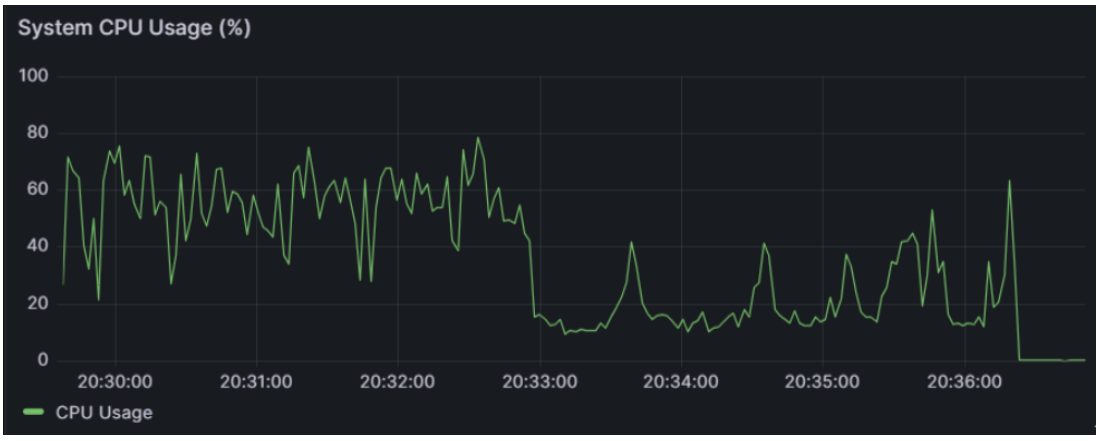


MD Core #2

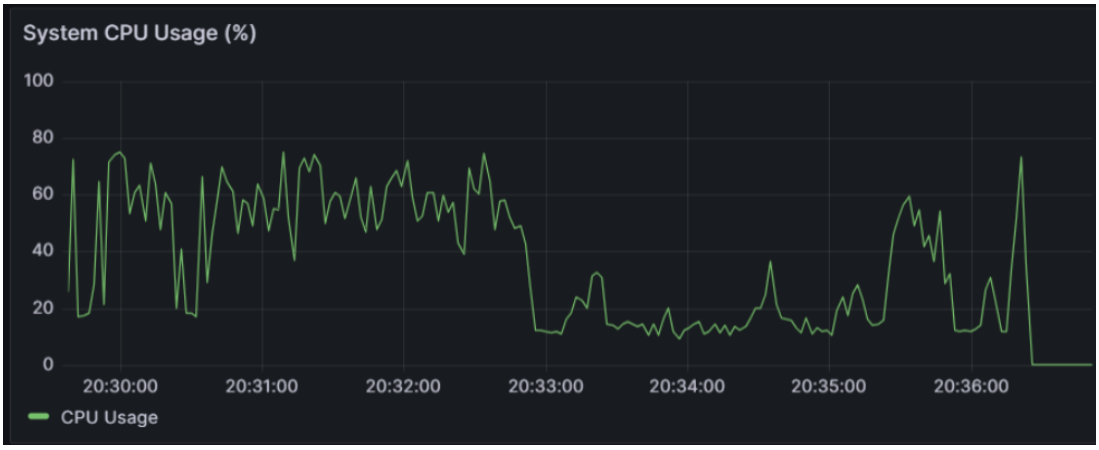


CPU usage

MD Core #1

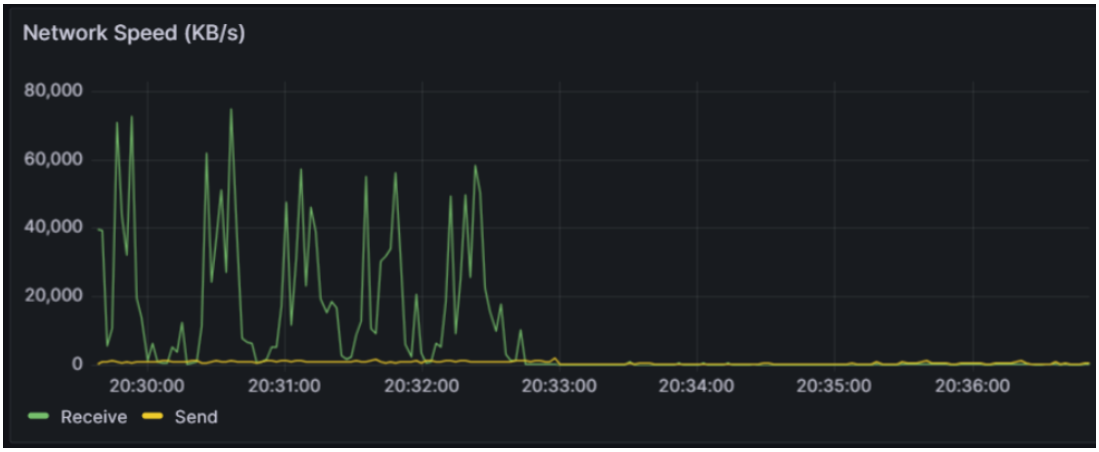


MD Core #2

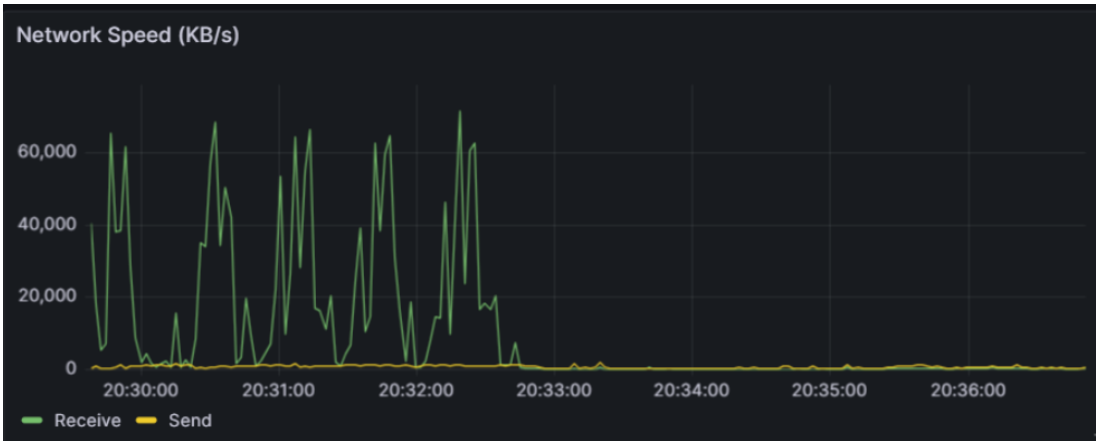


Network usage

MD Core #1



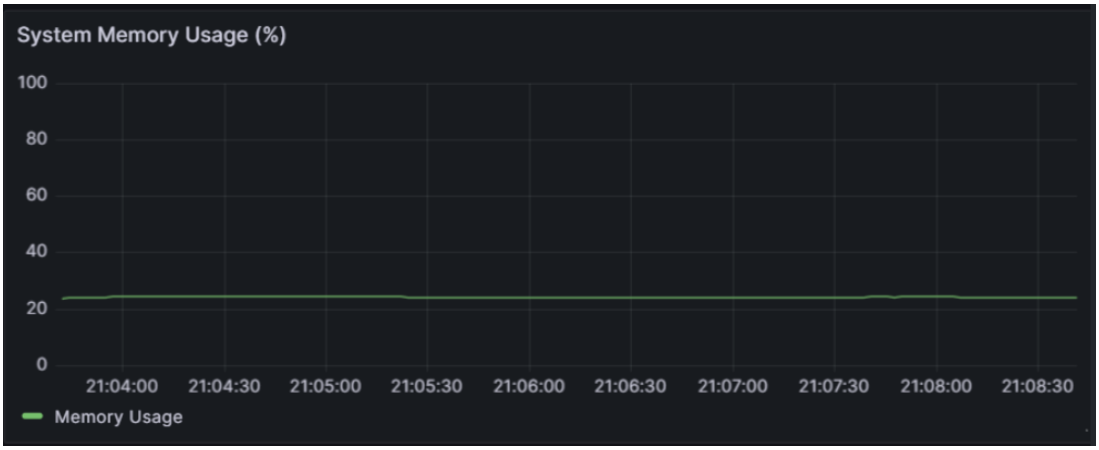
MD Core #2



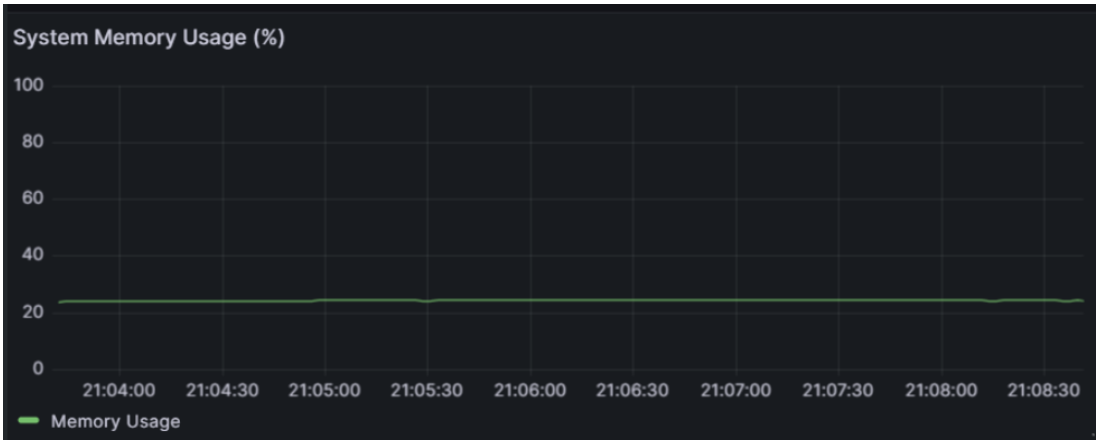
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

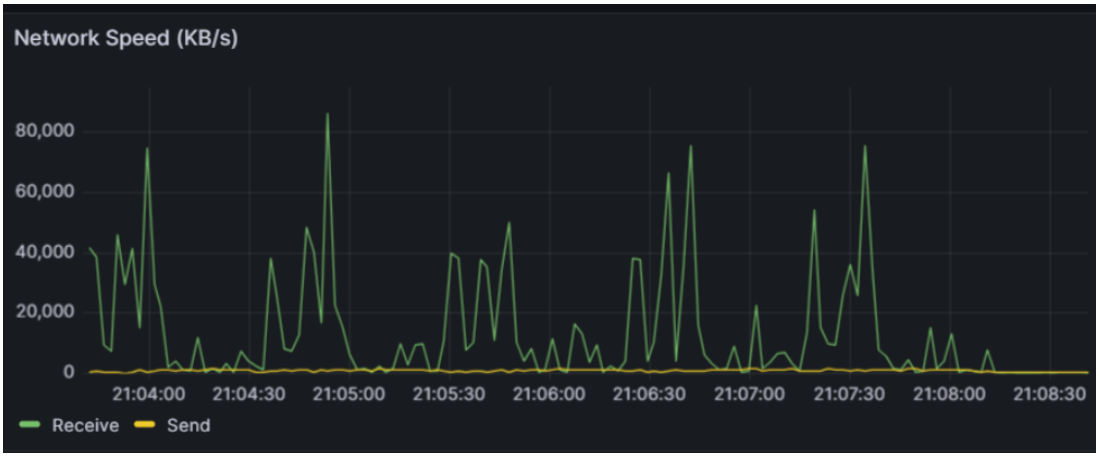


MD Core #2

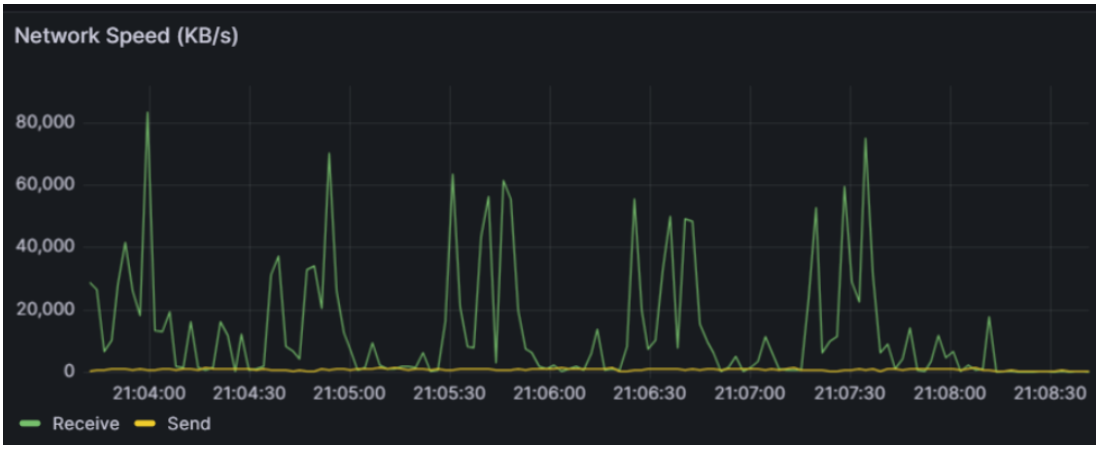


Network usage

MD Core #1



MD Core #2

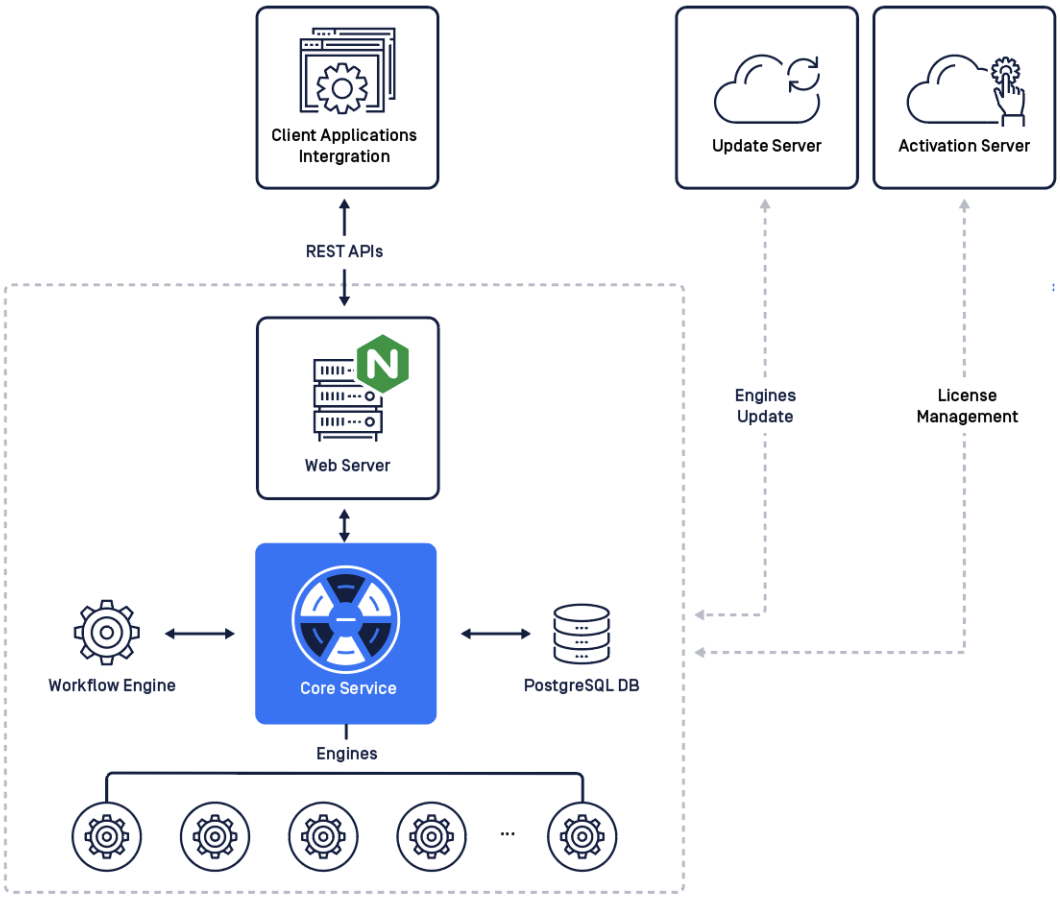

Network Speed (KB/s)
80,000
60,000
40,000
20,000
0
21:04:00
21:04:30
21:05:00
21:05:30
21:06:00
21:06:30
21:07:00
21:07:30
21:08:00
21:08:30
Receive
Send

MetaDefender Core Documentation

The users can consult this web page or, alternatively, they can download the manual in pdf format from the link below:

MetaDefender Core manual (SHA256:).

Recommended System Configuration



MetaDefender Core supports a wide range of deployment options, including air-gapped environments, providing the flexibility to choose the environment that best suits customers' requirements and infrastructure.

- Physical server.
- Virtualization platforms like VMware, Hyper-V and XenServer.
- Cloud Deployment.
- Container Deployment.

> **Note**
> Before installing MetaDefender Core, please refer to the **recommended minimum system configuration** listed below. The requirements are based on a standardized customer deployment and may require more or less resources to achieve the level of performance desired in your environment. Please note that the server specifications are built to allow a high volume daily processing. Please refer to Performance and Load Estimation for more details.

The recommendations below are for MetaDefender Core, API usage only. For any other use cases, please consult the user guide of the licensed products for accurate recommendations.

For certain use cases these might be adjusted and customized on specific needs and SLAs. We highly recommend to engage our ProServ team to assist in fine tuning MetaDefender and get the maximum performance out of your systems.

Microsoft Windows Deployments

[Windows] Supported Operating Systems

Only support 64-bit platforms.

- Windows 10, Windows 11 23H2, 11 IoT (22H2) only for End-User systems (e.g. Kiosk deployments)
  - Based on Microsoft License Terms, you're not allowed to use Windows Desktop as a server, therefore, MetaDefender Core can't be used on these systems for server use cases.
- Windows Server 2016, 2019, 2022

> **Disclaimer**
> End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

> **Plan for legacy OS support**
> OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:
>
> - Ubuntu 20.04 after **April 2025**.
> - Amazon Linux 2 after **June 2025**.
> - Windows 10 after **October 2025**.
>
> If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

[Windows] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 8	8	8 GB	16 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 12	16	16 GB	24 GB + 50 GB + 100 GB
MetaDefender Core 16	16	16 GB	32 GB + 50 GB + 100 GB
MetaDefender Core 20	32	16 GB	40 GB + 50 GB + 100 GB
MetaDefender Core MAX	32	32 GB	120 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core is installed.
- For a better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.
- SBOM processing requires more disk space, please allow at least 15GB free disk for the engine to work properly.

[Windows] Third Party Dependencies

- .NET framework 4.5 or above
- Microsoft Visual C++ Redistributable for Visual Studio 2015-2022 (version 14.38 or higher); x64 version.
- PostgreSQL: Only applicable when using a pre-installed PostgreSQL server running remotely.
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Some engines also have dependencies as described below:

Engine Name	Dependency
Deep CDR	Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 5.8.0 or above)
ESET	MetaDefender Core v4 temporary directory should have more than 200MB free disk space
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
Proactive DLP	• Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 2.0 or above) • .NET 6 dependencies • CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
Scrutiny	Microsoft Visual C++ 2015 - 2019 Redistributable x86 and x64
Systweak	.NET runtime 6.0, 7.0 or 8.0
Bkav Pro	Microsoft Visual C++ 2015 - 2022 Redistributable x64

[Windows] Installation Details

MetaDefender Core on Windows uses "C:\Program Files\OPSWAT" folder for storing resources or the installation directory.

MetaDefender will use its resources folder to store temp files as part of the analysis. **It's recommended to exclude this folder from real-time protection monitoring**. Knowledge base:

- How to set up exclusions on anti-malware software to prevent disruption
- Mitigating scan failure triggered by real-time protection service
- Can local AVs interrupt ongoing scans?

Linux Deployments

[Linux] Supported Operating Systems

Only support 64-bit platforms.

- Red Hat Enterprise Linux 8, 9
- Rocky Linux 9
- Debian 11, 12
- Ubuntu 20.04, 22.04
- Amazon Linux 2 is supported via Amazon Machine Image (AMI)

Plan for legacy OS support

OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:

- Ubuntu 20.04 after **April 2025**.
- Amazon Linux 2 after **June 2025**.
- Windows 10 after **October 2025**.

If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

For Hardended Linux OS

Some AV engines could be failed to initialize and run on MetaDefender Core (e.g. ESET, Kaspersky) due to execution permission required by them on `/var/tmp/ometascan` folder.

Please follow steps at Why have the ESET and Kaspersky scan engines failed to initialize on the hardened Linux OS? for remediation details.

Disclaimer

End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

[Linux] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 5	4	4 GB	10 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 10	8	8 GB	20 GB + 50 GB + 100 GB
MetaDefender Core MAX	16	16 GB	40 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core resides on.
- For better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.

[Linux] Third Party Dependencies

Dependencies list:

- openssl
- grep
- lib32stdc++6 (>= 4.5)
- libc6-i386 (>= 2.10)
- procps
- zlib1g
- libcurl3 (>= 7.19.7)
- libcurl4
- postgresql:
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Not all the above dependencies will need to be installed, it is dependent on different Unix distro and version

Some engines also have dependencies as described below:

Engine name	Dependency
Archive engine	• To extract password protected MS Office files, you need to install .NET 6 dependencies, libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • Rocky Linux 9.x/Red Hat 9.x: libnsl
Deep CDR	• Rocky Linux 9.x: libgomp, ncurses-compat-libs, libnsl • CentOS: libgomp, ncurses-compat-libs • Red Hat 8.x: libgomp, ncurses-compat-libs, libnsl • Red Hat 9.x: libgomp, ncurses-compat-libs, libnsl, compat-openssl11 • Ubuntu/Debian: libgomp1, libncurses5 • libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • .NET 6 dependencies
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
FileType engine	Rocky Linux 9.x/Red Hat 9.x: libnsl
Proactive DLP	libgdiplus, mscorefonts .NET 6 dependencies CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
RocketCyber	libgomp

Note: Some dependencies may need to be installed from external repositories, not from the OS default repositories.

[Linux] Installation Details

MetaDefender Core default installation path is using /var folder for storing resources:

- `/var/lib/ometascan` : installation folder with all its resources (database, DLP processed / CDR sanitized / quarantined file storage, engine package and definition updates).
- `/var/log/ometascan` : application logs.
- `/var/run/ometascan` : runtime folder for Core related processes.
- `/var/lib/ometascan` : all successful deployed engines data files.
- `/usr/lib/ometascan` : all shared libraries required by MetaDefender Core, bundled PostgreSQL server related binaries and libraries.
- `/var/tmp/ometascan/resources` : all temporary processing files for MetaDefender Core to process.

Storage Usage Information

Based on the configuration, MetaDefender Core will need additional disk space to store analysis/result data, quarantined files and temporary files:

	Data Storage
Analysis Reports (Processing History)	Approximately 4.5GB is required for every 1M analysis reports stored. • 100 GB is recommended as a baseline. Database storage requirements are affected by Data Retention settings, engine settings and scanning load.
Quarantined Files	Depends on the customers' dataset and configuration
Sanitized Files	Depends on the customers' dataset and configuration
Temporary processing files	Depends on the customers' dataset and configuration • A 1GB non-archive file requires 1GB free disk space at least. • A 1GB archive file requires (1 * 2)GB free disk space at least. • 50GB is a baseline and should be adjusted based on observation of the solution within the environment, with consideration to peak times and overall file makeup (e.g. complexity, size, archives).
Note	Options are available to control how large and/or complex files are handled to limit the amount of system resources required. See Workflow Configuration Template for more information.

Custom Engines

The recommendations above are specific for MetaDefender pre-packaged bundles.

However for additional Custom Engines, please review the Knowledge base to review additional requirements (if any) for the selected engine.

Browser Requirements for the MetaDefender Core Management Console

One of the following browsers is suggested to view the MetaDefender Core Management Console:

- Microsoft Edge
- Chrome
- Firefox
- Safari

Chrome, Firefox, Safari and Microsoft Edge browsers are tested with the latest available version at the time of release.

Network Bandwidth

In online environment where MetaDefender Core is configured to download engine update packages directly from OPSWAT update server source, then using gigabit network speed (125 MB/sec) is highly recommended to avoid potential network related issue while downloading files, or at least 100 Mbps (12.5 MB/sec) network speed.

Allowlisting URLs

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts':

- https://activation.dl.opswat.com - for license online activation
- https://update.dl.opswat.com - for engines and databases download

Note: *IP address-based allowlisting on your firewall might fail after some time since OPSWAT uses CDN (AWS Cloudfront) to faster delivery updates over the world, and IP address of edge servers might change over time.*

Depending on your location, connections with OPSWAT cloud-based servers be forwarded to specific AWS Cloudfront's edge locations, each is assigned with its own unique IP address ranges. You might also need to allow correct AWS Cloudfront's corresponding IP address ranges by referring to https://ip-ranges.amazonaws.com/ip-ranges.json

Warning

Even the OPSWAT update servers host updates for all of the available engines we support, sometimes custom engines might try to connect to their own cloud for updates, but this can be disabled in firewall and they will be updated just from OPSWAT cloud servers instead.

Differences Between MetaDefender Core v4 and v5

We are very excited to inform you about the release of version 5.0.0 of our MetaDefender Core!

The new version is designed to enhance the product performance, user experience, as well as to provide more flexibility, security, and scalability for our customers.

At the same time, we are also committed to keeping backward compatibility for whom running MetaDefender Core v4, and avoid the need for updating your client integration to move forward with MetaDefender Core v5.

v4 REST API Integration Supported
All public REST API endpoints available on MetaDefender Core v4 will be continously supported on v5.

v4 Product Major Features Supported
All product major features available on MetaDefender Core 4.21.2 (the latest one on v4) will be continously supported on v5.

Seamless Product Upgrade
Upgrading from MetaDefender Core v4 to v5 will be supported and covered with data migration without additional manual steps from users.

Licensing information will be also migrated.

Below are some highlights of the first v5 release, MetaDefender Core 5.0.0:

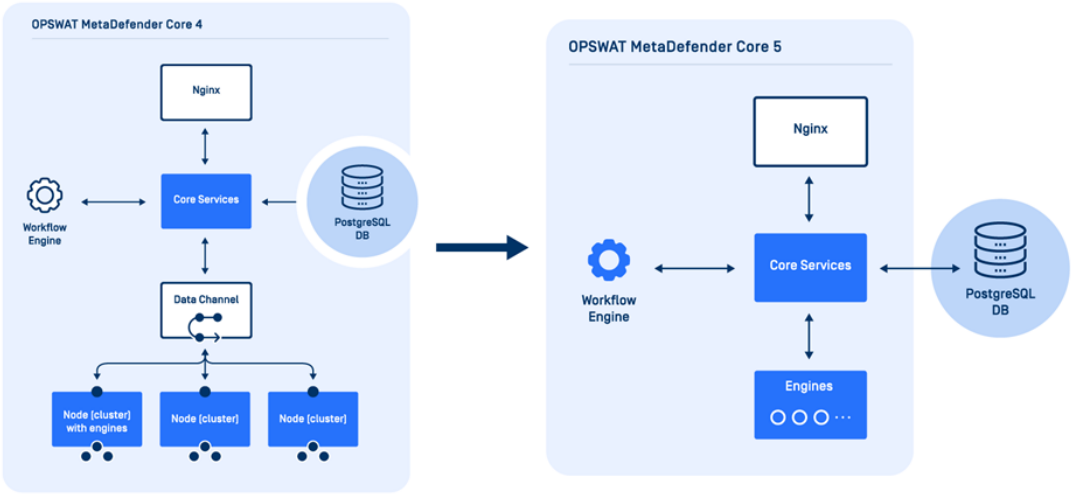
New product architecture

This version introduces a simpler product architecture that speeds up integration and automation, file processing flow, and immensely enhance the product performance and efficiency.

Earlier versions utilized MetaDefender Node concept, designed to support scaling of the scanning infrastructure by distributing scan requests among several different processing Nodes. Product licensing was based on number of Node instances. The benefit of having such a distributed infrastructure is that based on Node loads, Metadefender Core server can always choose the most appropriate Node to assign a new scan task to. However, having nodes in the v4 architecture caused more complexity and possible latency in the processing flow and entailed maintenance cost.

In this release, we retired multi-Node deployment and restructured MetaDefender Core's backend. This alteration significantly improves file processing speed and product efficiency, as well as eliminates extra cost for maintaining node services. Users now can simply install additional MetaDefender Core instances for scalability requirement,

as well as configure their own load balancer to coordinate tasks among multiple MetaDefender Core instances. Accordingly, product licensing is now based on number of the product instances.



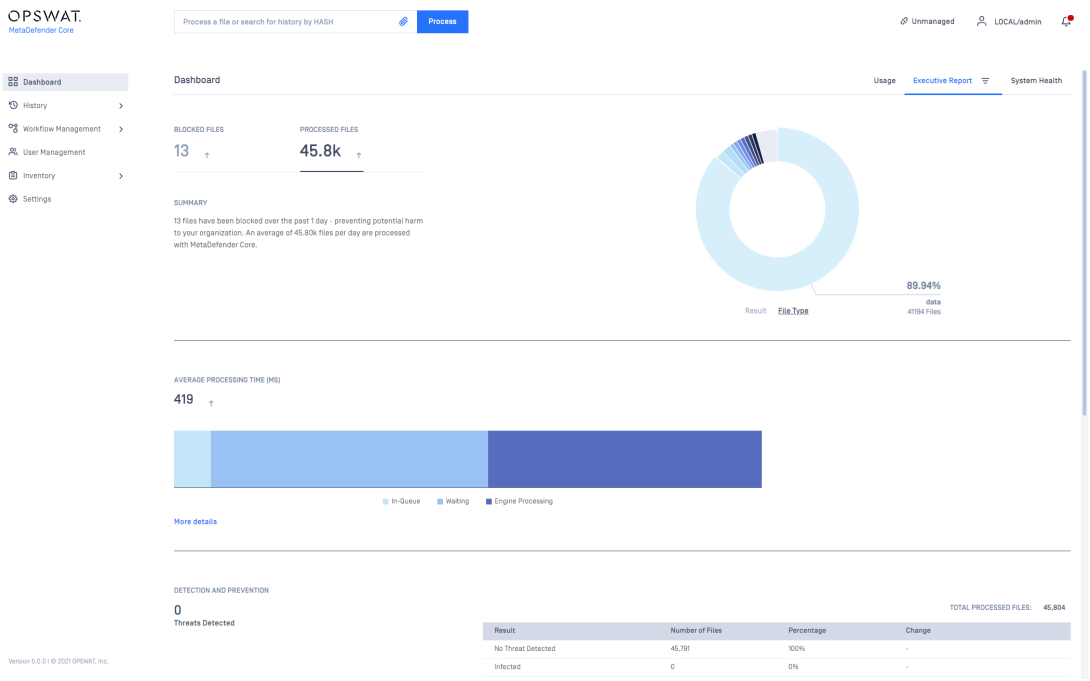
Please be informed that

- All existing administrative APIs related to "Node" on v4 will be kept as it is on v5 to ensure backward compatibility with existing client integrations.
- Applicable settings defined in Windows registry (Windows) or ometascan-node.conf config file (Linux) for Node will be migrated to Core. Details: MetaDefender Configuration

Brand-new product user interface

The second notable update in this version is a brand-new UI for the management console, which was designed with the focus on the quality and thoughtfulness of the user experience. The new administrative console UI provides an advanced set of Core features and observable processing status.

Comprehensive executive dashboard reporting offers dynamic, real-time and actionable threat insights. It also provides visibility of your security posture allowing you to paint a picture of security that aligns with the overall business. The report includes easy-to-read charts, graphs, and statistic explanation that help your security professionals reduce clutter, gather information quicker, spot threat activity early and act swiftly.



We're continuing to improve our product UI to provide users an excellent experience with the application and overall usability. Please expect more powerful UI features coming in next releases.

New compression archive engine for archive sanitization

In previous version, MetaDefender Core's archive engine implemented four functions including decryption, extraction, encryption and compression that with a high volume of files it takes time to process them all. For the purpose of enhancing archive handling performance, this release comes with a separated compression archive engine, which dedicatedly processes archive sanitization (compression and encryption). It is also created to facilitate maintenance and further feature development.

Archive sanitization feature now requires Deep CDR license

As to respect sanitization practice and ensure unknown threats prevention as designed, now starting on version 5, it's mandatory to have Deep CDR licensed for archive sanitization feature enablement on MetaDefender Core workflow configurations.

Archive handing related engines changed

The legacy "Archive" engine on MetaDefender Core v4 will be changed its name to "Archive Extraction" engine, even when you are still on MetaDefender Core v4.

A brand new "compression archive engine" is to dedicatedly handle archive sanitization in v5 (archive compression), that helps relevant archive handling performance improvement, easier for maintainance and further feature development.

Notes when you are still on MetaDefender Core v4 (not upgraded yet)

The legacy "Archive" engine (now name changed to "Archive Extraction" engine) on MetaDefender Core v4.x will still cover both extraction and compression archive tasks as usual.

Even though your MetaDefender Core v4.x might now have a new "Archive Compression" engine auto downloaded, but that engine cannot be deployed (marked as "permanently failed" in relevant engine status check API). This behavior is expected, and not causing any impact to MetaDefender Core processing flow, or its performance though.

- Just in case your client integration has a check for engine status before triggering scan request, then you might want to disable "Archive Compression" engine status on MetaDefender Core v4 using REST API. Detailed instruction: Archive Compression Engine Permanently Failed on MetaDefender Core v4
- When you later upgrade to MetaDefender Core v5 from v4, then you might need to manually re-enable "Archive Compression" engine to make it active and running on MetaDefender Core v5.

Notes for licensing

For offline deployment (MetaDefender Core is not connected to the Internet directly for online update), you will need to re-activate again on OPSWAT portal to retrieve latest engine list in .yml file including "Archive Compression" engine if you are currently licensed with Deep CDR.

Meanwhile, for online deployment, this manual procedure is not required since MetaDefender Core will cover that itself.

Import & export configurations

Not only workflow settings like in previous versions, but also all other general settings (update, security, data retention, server, hashes to skip) and most importantly, user settings, will be able to export and import on MetaDefender Core version 5.

Learn more at Import/Export configuration

Backward compatibility supported

The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

Windows application event logging disabled by default

Applicable only to Windows OS based MetaDefender Core.

Help avoid duplicated logging with separate file option, and improve overall processing performance. Of course, when being still valid to you, you are still supported to enable it back.

CSRF token header renamed

CSRF token originally created for security strengthening and being used by MetaDefender Core management console UI since version 4.21.0 via header `oms_csrf_token`

To get it compliant with some load balancers which do not accept underscore in header name by default, and save you from additional load balancer configuration step, we have now changed the header name to `oms-csrf-token`

This change does not impact to any REST API based client integration.

More comprehensive and accurate error messages on REST APIs

Impacting on following REST API endpoints:

- `PUT /admin/config/session`
- `PUT /admin/config/update`
- `PUT /admin/config/scan`
- `PUT /admin/config/skip`
- `PUT /admin/config/auditlog`
- `PUT /admin/config/webhook`
- `PUT /admin/config/rule`

Upload time and queue time logic updated in JSON scan response

Impacting on following REST API endpoints:

- `GET /file/{data_id}`
- `GET /hash/{md5/sha1/sha256}`
- `GET /archive/{data_id}`
- `POST /file/{async}`

Added new field:

- `receive_data_timestamp`: The timestamp when upload progress started (first byte received) (in milliseconds)
- `upload_time`: Total time elapsed for upload process (in milliseconds)

Modified existing field (new logic):

> **Backward compatibility supported**
> The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

- `upload_timestamp`: The timestamp when upload progress finished (all bytes received) (in milliseconds)
- `queue_time`: Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue time = start_time - upload_ timestamp) (in milliseconds)

Release version 5.0.0 also comes with variety of outstanding features and improvements.

For detailed information about our latest release please view our Release notes

Installation

Starting with MetaDefender Core 4.20.0, the centralized/shared database deployment model is supported (all MetaDefender Core instances share the same database), along with traditional standalone database mode (every MetaDefender Core instance has its own database to manage).

- Standalone mode
- Shared Database Mode

Installation

1. Download the package of your choice from My OPSWAT
2. Install the package on your computer via the Command Line or via the Wizard
3. Open a web browser and point to http://<server name or IP>:<port>
    - The default port is **8008**
    - In case of problem check Inaccessible Management Console page
4. Complete the required steps of the basic configuration wizard
5. You must activate this deployment to use its features

Installation notes

- If the MetaDefender Core package **dependencies are not installed** on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.
- During installation the databases might need to be upgraded. This could take noticeable time depending on database size (eg.: length of scan history).

Standalone mode

Wizard Installation

The Install Wizard is only for the Windows installer (.msi file).

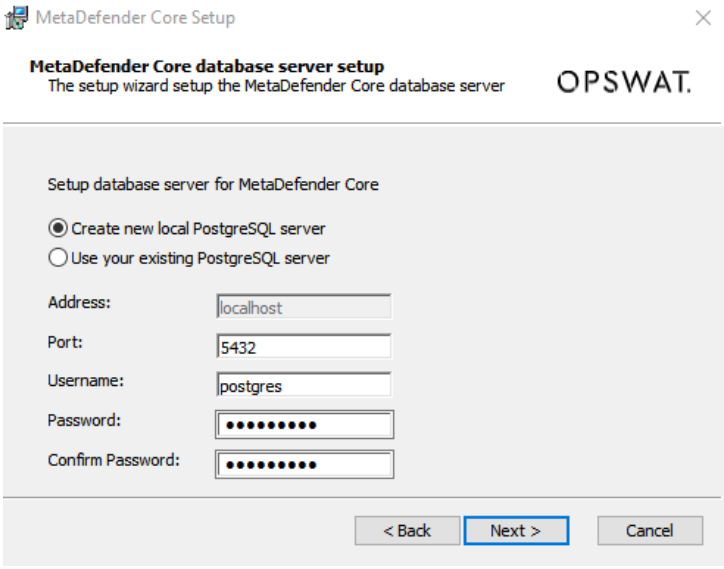
Since MetaDefender Core 4.19.0, we have switched to using PostgreSQL DBMS which needs a dedicated database server. MetaDefender Core supports two modes to setup PostgreSQL server:

Local (Bundle) PostgreSQL server

MetaDefender Core will install a new PostgreSQL server locally in the same box with the product.

> **PostgreSQL user credentials**
> Unicode characters are NOT supported for "Username" and "Password"
>
> "Password" does not accept # (hash) as starting character (e.g. #ABC)

> **The ignition file**
> If the ignition file exist in the designated location, then MetaDefender Core will use the details defined in the ignition file, rather than what is passed via the installation wizard.
>
> More info about the ignition file based installation: Command Line Installation

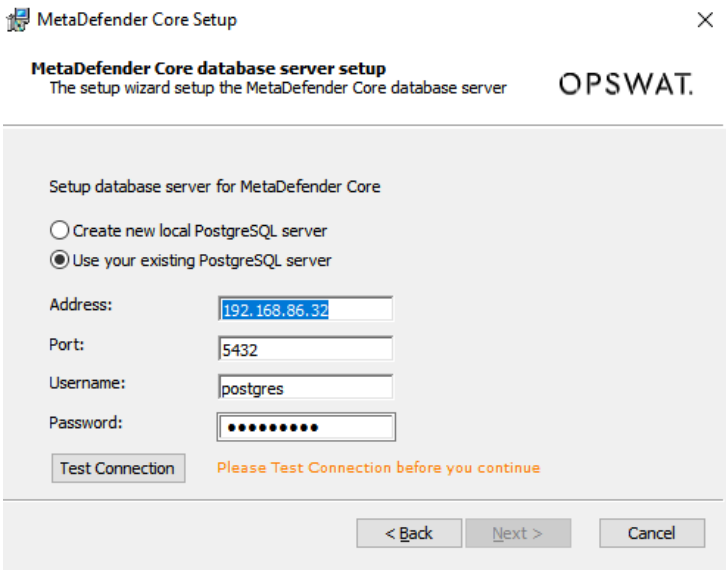


Remote (Existing) PostgreSQL server

MetaDefender Core will connect to remote pre-installed PostgreSQL running remotely.

> **PostgreSQL connection configuration**
> "Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

> **PostgreSQL user privilege requirements**
> MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.
>
> Details: PostgreSQL User Privilege Requirements



Command Line Installation

Ignition File

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Make sure to create the ignition file, if it doesn't exists before you go ahead and install MetaDefender Core via command line.

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

MetaDefender Core supports two modes to setup PostgreSQL server:

1) Local PostgreSQL server

- MetaDefender Core will install a new PostgreSQL server locally, next to the product.
  - A sample iginition file for PostgreSQL server information (***Only non-Unicode characters supported for "user" and "password"__***):

ometascan.conf bash

```
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
```

2) Remote PostgreSQL server

> **Info**
>
> **PostgreSQL version 12 (at least 12.3) or version 14 is required.**
>
> According to PostgreSQL notice, version 12 will stop receiving fixes on November 14, 2024, so we suggest that you use version 14.

MetaDefender Core will connect to a pre-installed PostgreSQL server running remotely and will setup its database.

Make sure to follow steps at Open Connection On PostgreSQL Server first, so that your remote existing PostgreSQL server to accept upcoming connection from MetaDefender Core.

A sample ignition file for PostgreSQL server information:

ometascan.conf bash

```
[dbserver]
type=remote
host=192.168.86.32
port=5432
user=existingserver_admin_user
password=existing_server_pass
```

After installing MetaDefender Core successfully, you may want to remove all credentials info created for [dbserver] section in ignition file for security reason.

> **Warning**
> If the MetaDefender Core package dependencies are not installed on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.

Debian package (.deb)

bash

```
sudo dpkg -i <filename> || sudo apt-get install -f
```

On Red Hat Enterprise 7.x Linux / CentOS package (.rpm)

bash

```
sudo yum install <filename>
```

For systems which enabled GPD check flag:

bash

```
sudo yum install --nogpgcheck <filename>
```

Red Hat Enterprise Linux 8.x package (.rpm)

Install the latest PostgreSQL repository RPM:

> **Note**
> Due to the shorter support cycle on Red Hat Enterprise Linux, all supported versions of PostgreSQL are not available on this platform. The PostgreSQL Yum Repository will integrate with the normal systems and patch management, and provide automatic updates for all supported versions of PostgreSQL throughout the supported lifetime of PostgreSQL. Reference: https://www.postgresql.org/download/linux/redhat/

bash

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

In case installing a remote PostgreSQL on either the same MetaDefender Core server or another server, you need to install the following additional package. PostgreSQL version 12.3 or newer is recommended.

bash

```
sudo dnf install -y postgresql12-contrib
```

Install MetaDefender Core:

bash

```
sudo dnf install -y <filename>
```

Windows package (.msi)

On Windows systems it is possible to install the product by running the corresponding .msi file.

From command line interface it is also possible to install the product by executing

bash

```
msiexec /qn /i <msi file name> <option key>=<option value>
```

where the possible keys and their default values are the following:

Key	Default Value	Description
INSTALLFOLDER	\Program Files\OPSWAT\MetaDefender Core	Customize installation folder for product Example: `INSTALLFOLDER="D:\Products"`
RESTADDRESS	*	REST interface binding IPv4 or IPv6 address ('*' means that service listens on all IPv4 and IPv6 interfaces)
RESTPORT	8008	REST interface binding port
REPORT_ENGINE_ISSUE	true	Enable reporting of engine issue count. (possible values: "true" or "false").

Note

Reporting of engine issue count

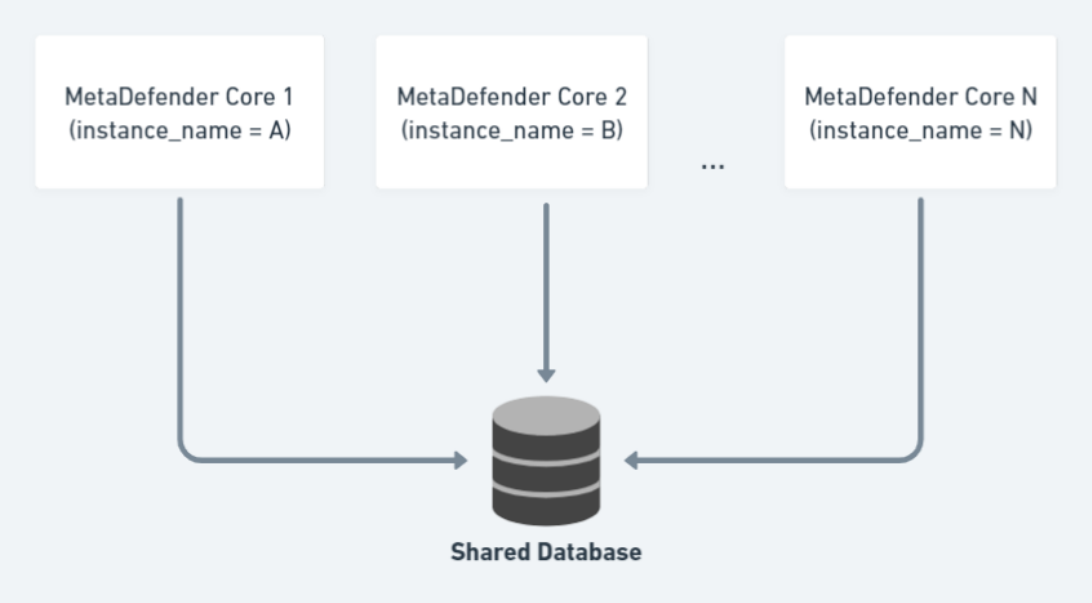
If reporting of engine issue count is enabled, MetaDefender Core server will send only the **number** of initialization errors and **number** of unexpected stops for the specific db/engine version. This information is sent over a HTTPS channel when the product downloads the latest package descriptors. This information is used for early detection of any specific 3rd party engine quality issues.

For details on using msiexec please consult Windows installer documentation.

Shared Database Mode

Starting with MetaDefender 4.20.0, you can use a centralized database deployment model, where all MetaDefender Core instances can share the same external database.

That helps auto-sync all product configurations (**except Single-Sign-On** that requires unique settings for each Core), processing history, live/on-going scan result query, audit, statistics data.



This article demonstrates how to install and setup MetaDefender Core 4.20.0 or newer with shared database mode.

Important notes & current limitations:

Currently not supported

- Does not support Central Management use-case with shared DB mode.
- Does not support to switch from shared database mode back to standalone database mode.
- Does not support to change instance_name of MetaDefender Core in shared database deployment model after installation.
- Does not support upgrading older MetaDefender Core from non-persistent mode to using the shared database mode.
- Does not support upgrading older MetaDefender Core from standalone mode to using the shared database mode.
- Does not support to upgrade multiple MetaDefender Core instances at once in shared database mode
    - Stop all Core instances' scanning service before performing the upgrade.
    - Upgrade each Core server one by one instead.

- For each Core, when the upgrade finished, then you can start using that Core for scanning without waiting for the other Core instances' upgrade to be done.

> **Disclaimer**
> MetaDefender Core does not support mode switching.

Requirements for shared database setup:

- The central remote PostgreSQL server (for shared database) could be utilized with a built-in (native) PostgreSQL database server installed with one of the MetaDefender Core instances in the network. Or using a dedicated existing PostgreSQL database server (see detailed requirements for PostgreSQL version at Recommended System Configuration).
- Before setting up, make sure to follow steps at Open Connection On PostgreSQL Server so that all MetaDefender Core instances could connect to the central remote PostgreSQL server.
- MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core, details: PostgreSQL User Privilege Requirements
- All MetaDefender Core instances must be installed on the same Operating System (Windows / Linux).
- The first MetaDefender Core instance connects to the shared database must be ready first (license activated, users/groups/roles created) before setting up other MetaDefender Core instances to connect and use that shared database.

Engine update:

- If engine updates is configured to be via Folder source, all MetaDefender Core instances must be able to access to the configured update folder, and the option DELETE FILES AFTER IMPORT will be unchecked and prevented for modification.

Authentication:

- Single-Sign-On (SSO) authentication settings (if using) cannot be shared among MetaDefender Core instances because SSO requires an unique login URL for every different Core.
- If an user is locked for login on a certain MetaDefender Core instance(e.g. exceeded the number of failed login times), that user is not prevented for logging in on any other instances.
- If an user is locked for sign-in on a certain MetaDefender Core instance (e.g. exceeded the number of failed login times), if the admin releases lockout on that user on that MetaDefender Core instance, that user is still not able to login on other MetaDefender Core instances.

Sanitized / DLP / Quarantined location changed after MetaDefender Core install:

Windows:

Shared storage path will be stored in 3 strings under key global in registry: quarantinepath, dlppath, sanitizepath as below

(Default)	REG_SZ	(value not set)
dbmode	REG_SZ	4
dlppath	REG_SZ	\\10.40.50.100\test
instance_name	REG_SZ	481rvj
quarantinepath	REG_SZ	\\10.40.50.100\test
restaddress	REG_SZ	*
restport	REG_SZ	8008
sanitizepath	REG_SZ	\\10.40.50.100\test

You can set either 1, 2 or 3 different paths for the 3 settings above (means that quarantined, DLP and sanitized files will be stored in same or different storages corresponding). In case shared storage is not set in installation, values of 3 settings above will be empty.

Restart MetaDefender Core service after changed.

Linux:

Open file `/etc/ometascan/ometascan.conf` and set shared storage path for 3 keys quarantinepath, dlppath, sanitizepath under section [global]

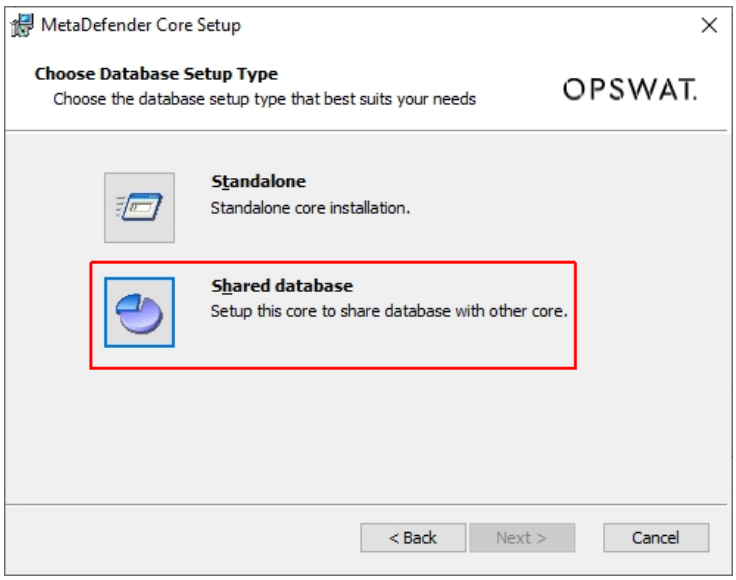
Restart ometascan service after changed.

Wizard Installation (Windows Only)

> **Info**
> Only applicable to MetaDefender Core 4.20.0 or newer using wizard on Windows OS.

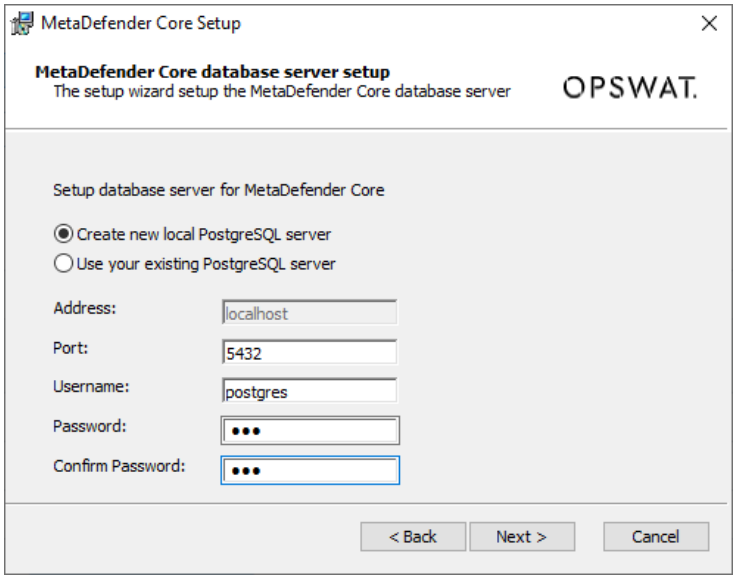
Clean install scenario

1. **Choose Shared database option:**

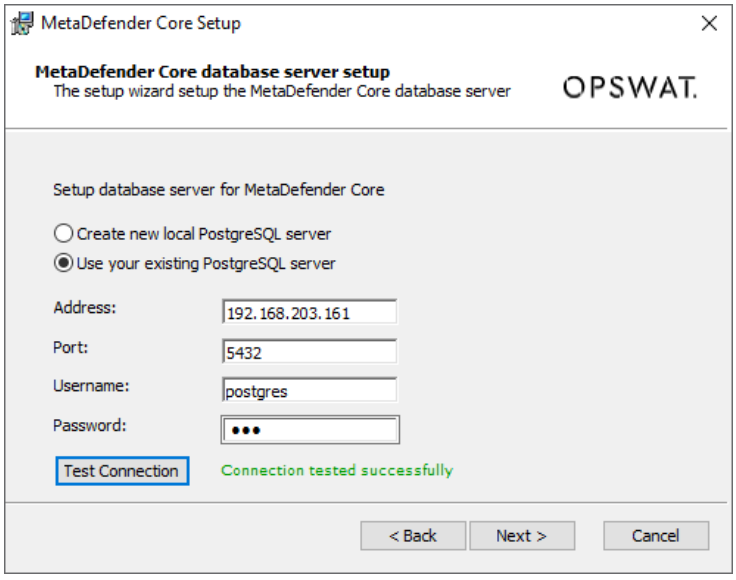


2. **Setup database server for MetaDefender Core:**

You can choose to create a native / built-in PostgreSQL server and setup the shared database on (then other MetaDefender Core instances will want to connect to that PostgreSQL server afterwards)



Or choose to connect to a pre-setup existing PostgreSQL server:



PostgreSQL connection configuration

"Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

PostgreSQL user privilege requirements

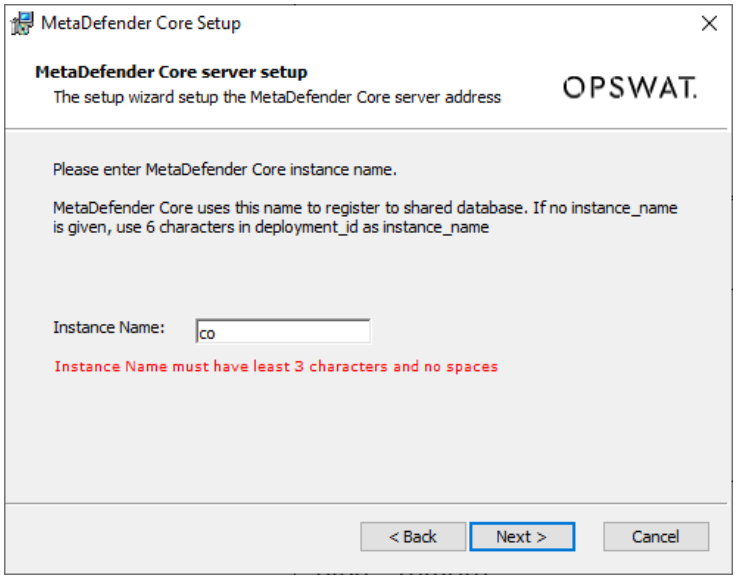
MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.

Details: PostgreSQL User Privilege Requirements

3. **Create an unique instance name for that MetaDefender Core (different from other MetaDefender Core instances):**

Instance name must contain at least 3 characters and no white space.

If you choose to leave it blank, which means it will use the default instance name with 6 characters extracted from that MetaDefender Core deployment ID.

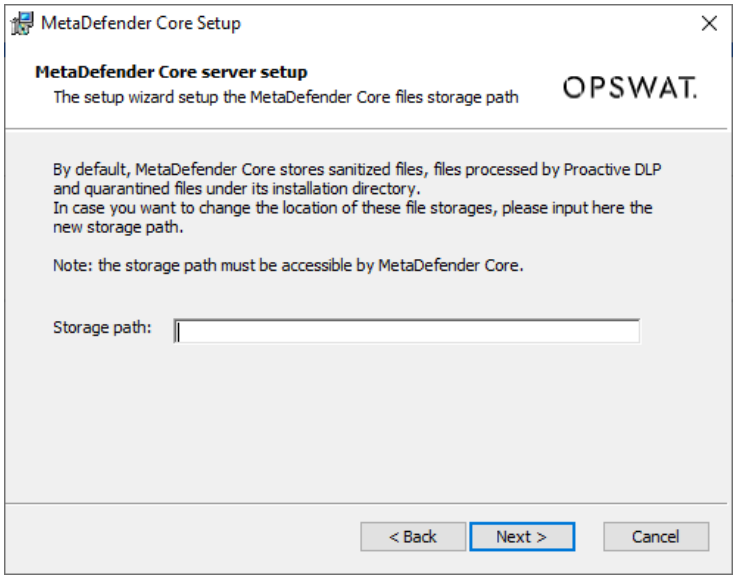


4. **Input the file storage path (sanitized, DLP, quarantined files)**

In the shared database model, all MetaDefender Core instances could use the same folder path to store all their own sanitized, DLP, quarantined files after processed.

The default location is: ***<MetaDefender Core installation folder>\data\ {sanitized|dlp|quarantine}***

If you wish to use that default location, just leave the input box blank and press Next.



Otherwise, you can specify a different location of sanitized/dlp/quarantined files. It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core.

For example:

- C:\new_folder
- \192.168.1.1\new_folder

Command Line Installation

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Clean install scenario

1. Create ignition file if not existed

- Windows: **C:\OPSWAT\ometascan.conf**
- Linux: ***/etc/opswat/ometascan.conf***

An example of ignition file:

ometascan.conf bash

```
eula=true

[global]
dbmode=4
instance_name=MetaDefenderCoreA
storage_path=D:\path\

[user]
name=admin
password=admin
email=admin@local

[dbserver]
type=remote
host=192.168.1.1
port=5432
user=postgres
password=your_own_password
```

Field	Required	Mapping to wizard UI	Possible values
global/dbmode	optional default is 1 (standalone)		1 - Standalone 2 - Central Hub deployment 3 - Non-persistent **4 - Shared**
global/instance_name	optional default is 6 characters of deployment ID		Unicode and number At least 3 characters No whitespace
global/storage_path	optional default is empty		Path to storages (DLP, quarantined, sanitized files) It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core. For example: • C:\folder • \192.168.1.1\folder

2. Run installers using command line to install

Please refer to Command Line Installation for installation command.

Deployment automation support

Note

An Ignition file could be utilized for either / both of following use-cases:

1. Pre-define PostgreSQL server information for MetaDefender Core to connect and setup database: only required on Linux installation, or Windows with command line installation. Learn more details at:
    a. Standalone Mode > Command Line Installation (applicable on version 4.19.0 or newer).
    b. Shared Database Mode > Command Line Installation (only applicable on version 4.20.0 or newer).
2. Automating to accept EULA, create a default local admin user and import MetaDefender Core configurations upon installed: Those steps could be automated using ignition file described in this page.

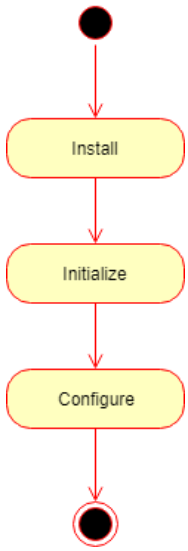
Ignition file location to create:

- **Windows**: `C:\OPSWAT\ometascan.conf`
- **Linux**: `/etc/opswat/ometascan.conf`

The product supports fully automated deployment. It means that it can be installed and configured with no human interaction.

The automated deployment can be split to three steps on a high level:

1. Installation
2. Initialization
3. Configuration



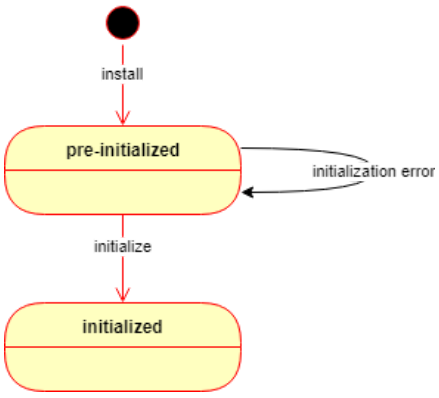
Installation

To automate the installation, install the product from the command line and provide the installation-time options as parameters to the installer. For further details see Command Line Installation

After the installation is complete, the product starts up and waits in a pre-initialized status. The product may be initialized in two ways:

1. Manually using the Wizard Setup or
2. Automatically using an *ignition file* (see below).

> **Warning**
> If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried after fixing the problem (e.g. placing the ignition file in its lookup location) and restarting the OPSWAT MetaDefender Core service.
>
> Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

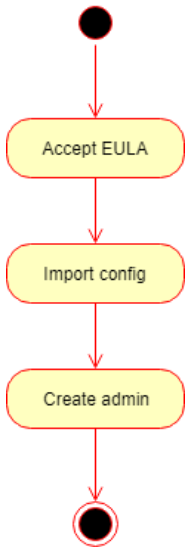


Initialization

Initialization is the process of bringing the product to an operable status.

Basically the initialization consists of the following steps:

1. Accept the End User License Agreement (EULA),
2. Import product configuration and
3. Create the first administrator user account.



Ignition file

The initialization process can be configured in a file called the *ignition file*.

> **Warning**
> In Linux, the ignition file and the folder /etc/opswat should have the permission 755

Ignition file location

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

The ignition file must be in conf format.

ometascan.conf bash

```
eula=true
[user]
name=admin
password=admin
email=admin@local
apikey=44c96cbb4e4e6c0a1d16bcb3c3da24def317
[config]
import=config_package.zip
import_password=123
[security]
session_cookie_samesite=Lax
[internal]
skip_startup_pages=true
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
private_username=my_internal_db_user
private_password=mypassword
[ocm]
ocm_url=https://my.opswat.com
reg_code=XXXXXXXX
```

Ignition file fields

The ignition file accepts the following fields:

Section	Key	Required	Accepted Values	Description
	eula	Optional (mandatory when having [user] section)	true false (default)	Whether to accept the End User License Agreement. This key must be set at the top of ignition file. This key must be set to "true" to accept the EULA. Any other value will cause the initialization to fail. When having [user] section defined, then this key must be set.
[user]		Optional		Initial administrator user account properties. The Administrator role is granted to the account.
	name	Optional	string	User name for the initial administrator user account.
	password	Optional	string	Password for the initial administrator user account. WARNING! Clear text password The password in this configuration file must be stored in its clear-text format and as so it may be visible for unauthorized parties.
	email	Optional	string	E-mail address for the initial administrator user account.
	apikey	Optional	string	API key for the initial administrator user account.
[config]		Optional		
	import	Optional	string	Path to a file in json/zip format that contains a previously exported configuration to be imported. Zip-format file contains a full configuration including user management settings, so it will overwrite [user] section.
	import_password	Optional	string	Password to decrypt .zip configuration package.
[security]		Optional		

Section	Key	Required	Accepted Values	Description
	session_cookie_samesite	Optional	Strict Lax (default) None	Declare if session cookie should be restricted to a first-party or same-site context.
[internal]		Optional		
	skip_startup_pages	Optional	true false (default)	Skip all welcome and other startup pages (e.g. upgrade result) when installing / upgrading MetaDefender, and going straight to the product index default page.
[dbserver]		Optional		
	type	Mandatory when having [dbserver] section	local remote	Where the PostgreSQL database server should physically locate. "remote" means PostgreSQL server and MetaDefender Core are not in the same machine.
	host	Mandatory when having [dbserver] section	string	IP address / domain name of the server where PostgreSQL server locates. "localhost" can be used when applicable.
	port	Mandatory when having [dbserver] section	number	Port of PostgreSQL server is listening for connections from clients (i.e. MetaDefender Core).
	user	Mandatory when having [dbserver] section	string	PostgreSQL server's user. SUPERUSER privilege is required for MetaDefender Core to setup its database and extensions for the first time. **Only non-Unicode characters supported.**

Section	Key	Required	Accepted Values	Description
	password	Mandatory when having [dbserver] section	string	PostgreSQL server's user credentials. **Only non-Unicode characters supported.**
	private_username	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
	private_password	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
[ocm]		Optional		
	ocm_url	Optional	string	URL to My OPSWAT server.
	reg_code	Optional	string	A registration code to enroll to My OPSWAT server.

Detailed initialization process

1) After the product has been started, it looks for the ignition file in the designated location.

Windows: C:\OPSWAT\ometascan.conf ** Linux: /etc/opswat/ometascan.conf

** 2) If an ignition file is found, then

2.1) It gets validated, and if it is valid, then based on the information found in the ignition file:

- The EULA is accepted,
- The configuration is imported,
- The administrator account is created.
- If any of the above steps fails, then the error is logged, and the initialization gets terminated.

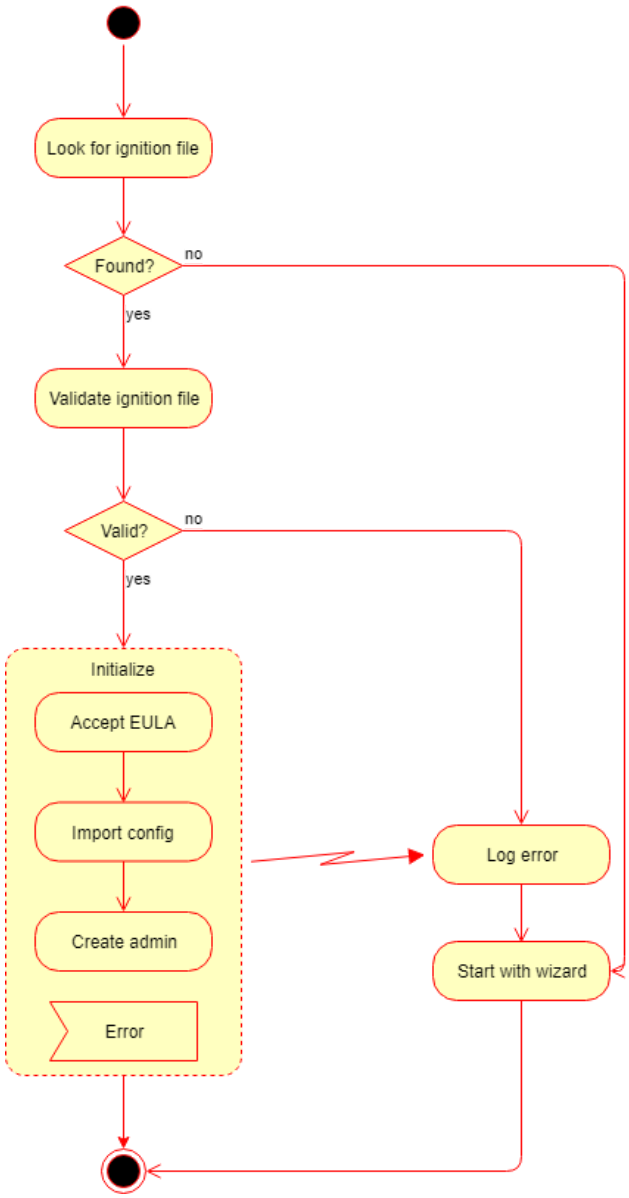
> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

2.2) If it is not valid, then the error is logged, and the initialization gets terminated.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

3) If there is no ignition file, then no initialization is performed.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.



Warning

If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried fixing the problem (e.g. placing the ignition file to its lookup location) and restarting the OPSWAT MetaDefender Core service.

Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

Configuration

After the initialization is complete, the product is ready with the default and the imported configuration.

This configuration can be later changed calling the configuration API functions. For further details about the API see Configuration related APIs API

CIS Level 1 Guidelines

Support Red Hat Enterprise Linux 9
For more details about Center for Internet Security (CIS) please refer to this document

Instruction steps

I. Install OpenSCAP

bash bash

```
yum install openscap-scanner scap-security-guide
```

II. Generate a result file and a HTML report using OpenSCAP scanner tool

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

III. Remediation of CIS Level 1 issues

Generate a remediation script based on the `ssg-rhel9-ds.xml` file:

bash

```
oscap xccdf generate fix --profile
xccdf_org.ssgproject.content_profile_cis_server_l1 --fix-type bash --output
remediations.sh /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

And execute remediation script.

bash

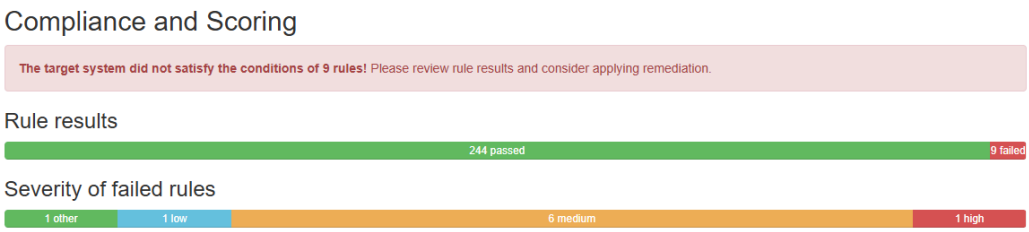
```
./remediations.sh
```

IV. Review the results after remediation

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

Example:



Notes

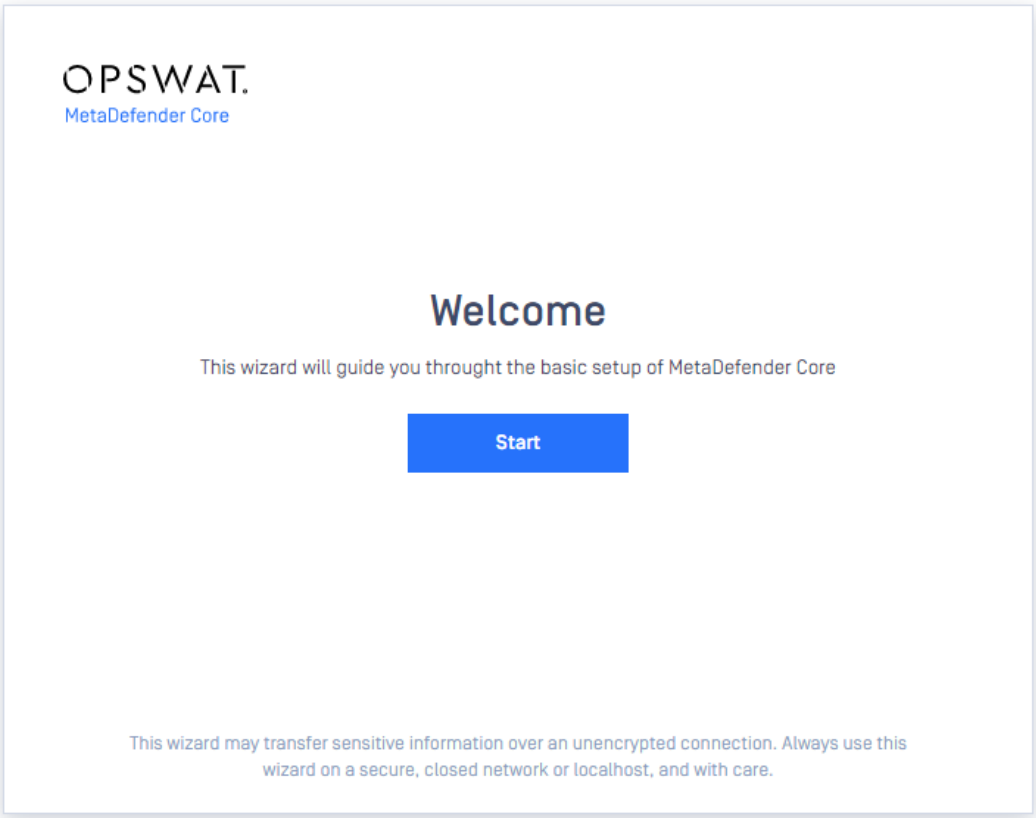
- CIS Level 1 requires `/tmp` folder to be mounted in a separate partition. Please ensure that that new partition have enough disk space for MetaDefender Core to run.
- CIS Level 1 requires that "Ensure No World-Writable Files Exist".
    - For now, when freshly installing MetaDefender Core, all its binary files meet the requirement.
    - When installing/updating engines, some engines might create additional files for its operation, and it might violate this requirement. In this case, you need to again execute the remediation script in the step III.

Wizard Setup

Introduction

Access the MetaDefender Core management console for the first time (i.e. http://localhost:8008), you are to complete a quick walkthrough for wizard setup in order to be able to use the product.

The administrative management console will be available only upon you have successfully finished this wizard stage.

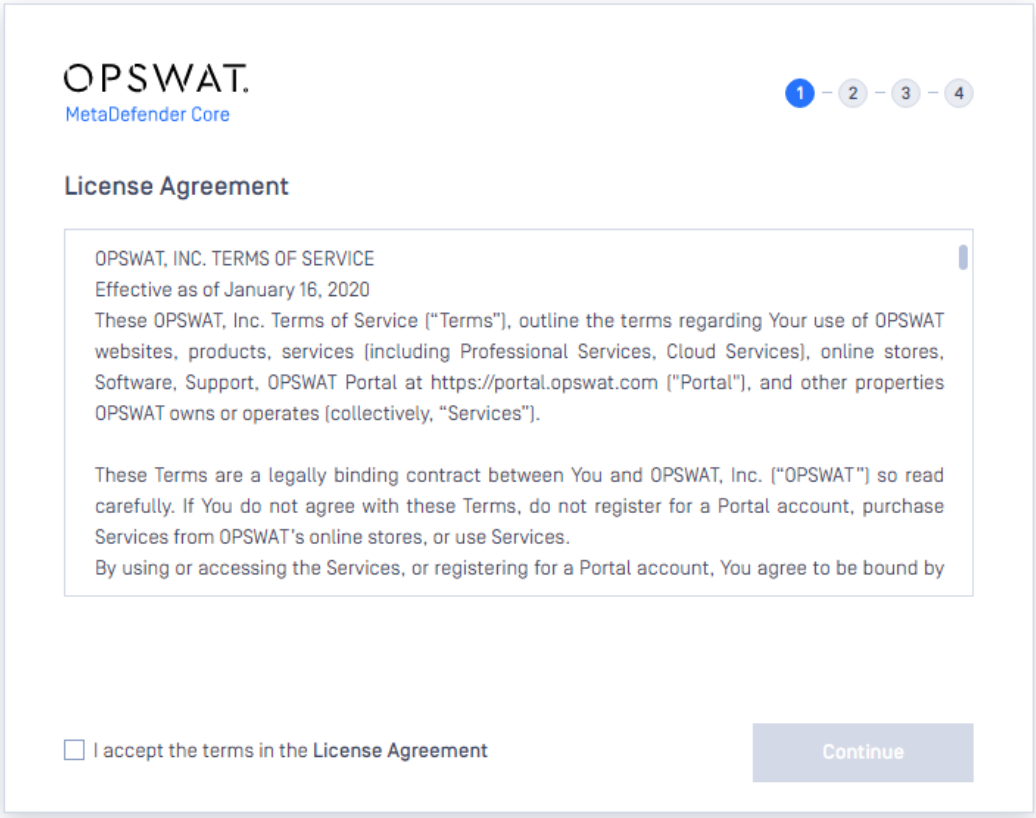


Warning

This wizard may transfer sensitive information over an unencrypted connection. Always use this wizard on a secure, closed network or localhost, and with care.

Basic configuration steps

End-User License Agreement



In the first page you can find the End-User License Agreement. You have to accept the terms before moving on.

Please read through the EULA (https://docs.opswat.com/policies/privacy-policy/portal-privacy-policy) carefully and if you agree with it, check `I accept the terms in the **License Agreement**` and click `Continue` to proceed.

Local Admin User Setup

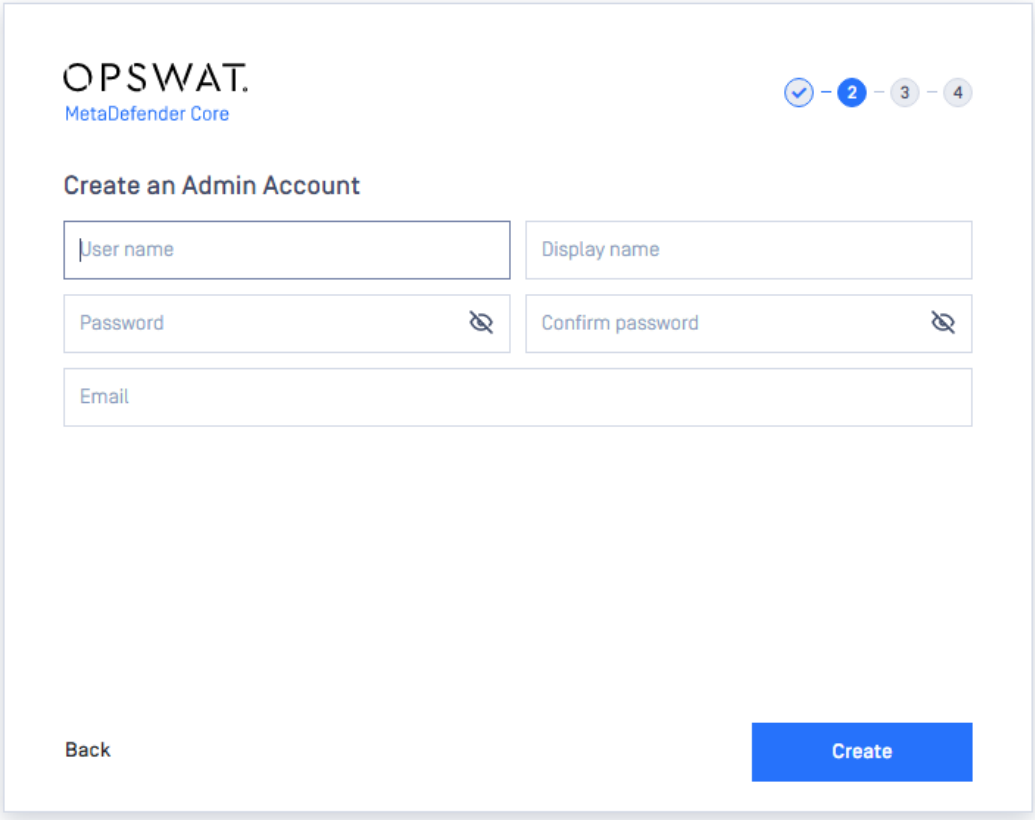
The next step is to set up a default local administrator account.

This account will be the first one being able to access the administrative management console, and to create accounts for other users. You have to fill out all fields in this page to be able to move forward. When you are done, click NEXT to continue.

> **Info - User Directory**
> The administrator account, that is created via the basic configuration wizard, is always added to the LOCAL user directory as a member.
>
> Learn more about user directory: User directories

The following information is required for the administrator account:

ACCOUNT NAME	The unique name of the account that is used at the time of login and in log messages for accountability.
ACCOUNT DISPLAY NAME	Name of the person bound to this account. This name (appended to the name of the account's user directory) is displayed in the top right corner of the administrative management console.
PASSWORD	Password of the user bound to this account that is used at the time of login. Passwords sent clear-text As long as TLS is not configured for the basic configuration wizard, passwords are sent clear-text over the network and may be disclosed to unauthorized parties. As a mitigation action: 1. Either use the wizard on *localhost* or on a direct network connection, or 2. Enabling HTTPS as soon as possible and change the password immediately if it has already been set.
EMAIL	Email address of the person bound to this account.



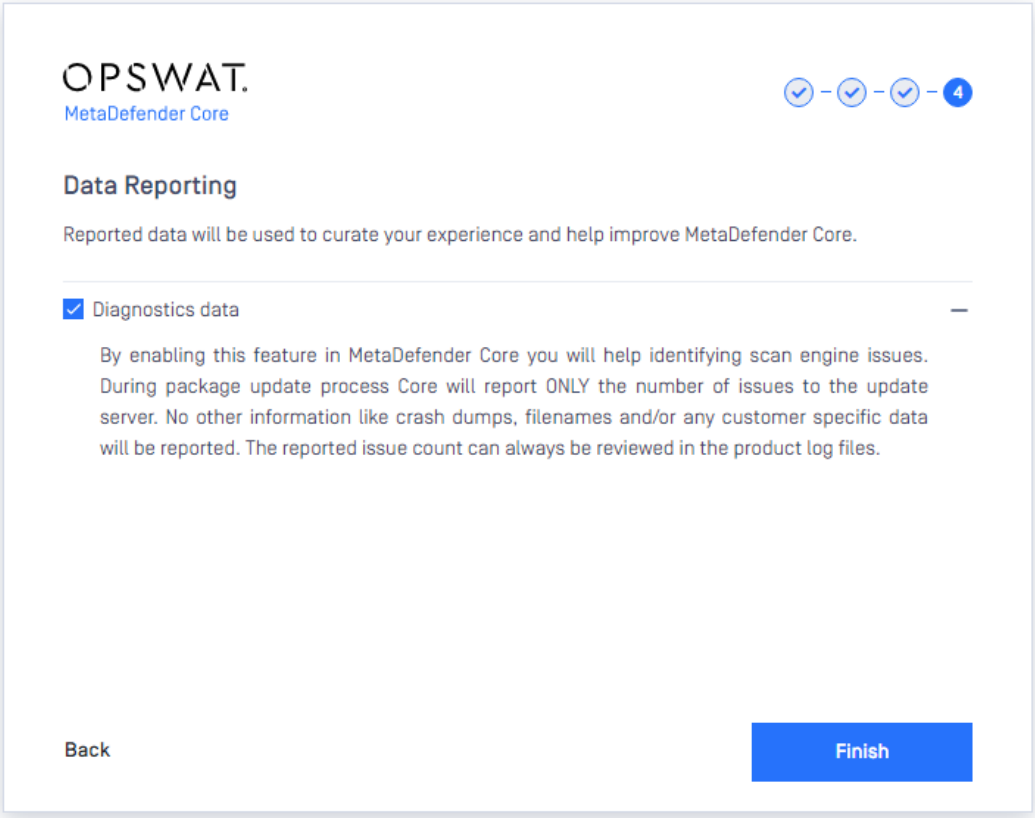
License activation

For license activation details see License Activation

Data Reporting

Enabling this option to allow MetaDefender Core collect its own engine issues and report back to OPSWAT for further analysis.

MetaDefender will not collect any other information.

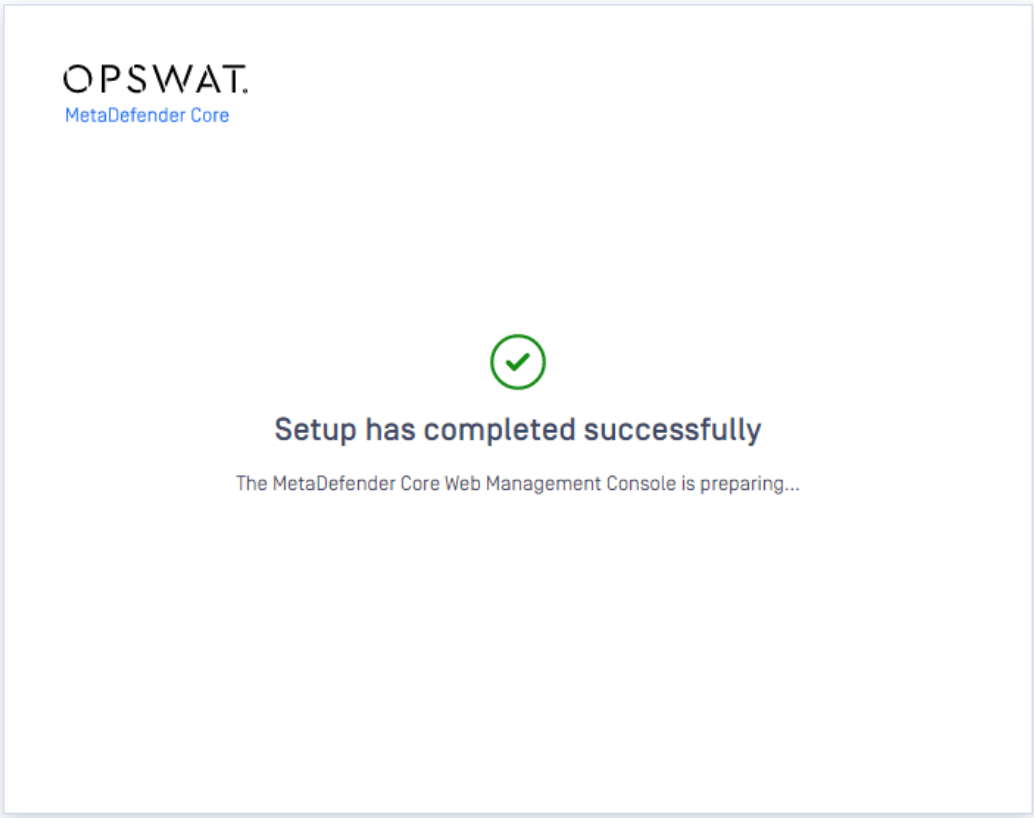


Wizard completion

After you have completed every steps you are ready to finish the wizard and start using the product. Click the `Fishish` button to complete the wizard.

> **Warning**
> The product's service will be restarted and the browser will be redirected to the administrative management console. This could take several seconds.

You can login to the Web Management Console with the administrator user that have just been created in the previous steps.



Setup HTTPS

Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network. Websites, like the Web Management Console, are able to use TLS to secure all communications between their servers and web browsers.

The TLS protocol aims primarily to provide confidentiality (privacy) and data integrity between two communicating computer applications.

Warning

No TLS for the wizard

By default, TLS is not enabled for the basic configuration wizard. As a consequence sessions between the wizard's backend and the browser may be insecure.

Info

Performing the same steps as for the Web Management Console, it is possible to set up TLS for the basic configuration wizard. Remember completing the TLS setup before launching the wizard.

For instructions to set up TLS see Enabling HTTPS

License Activation

##

To activate your installation go to the **Settings > License** menu in the Web Management Console. If you have no valid license, you will only see your installation's **deployment ID**. You will also see a warning in the Web Management Console header.

After successful activation, the product will start downloading the latest available scan engines and malware databases. You can follow the status of the scan engine installation on the **Inventory > Modules** page.

When all the licensed engines are installed, you can start using the MetaDefender Core to process files - File Processing.

Deployment ID

Each MetaDefender Core instance represented by an unique system fingerprint called `Deployment ID`

MetaDefender Core licensing is purely based on deployment ID activated. Therefore, everytime your MetaDefender Core is activated with a different deployment ID, then it will be captured as a different activation record accumulated into your total activation times allowed.

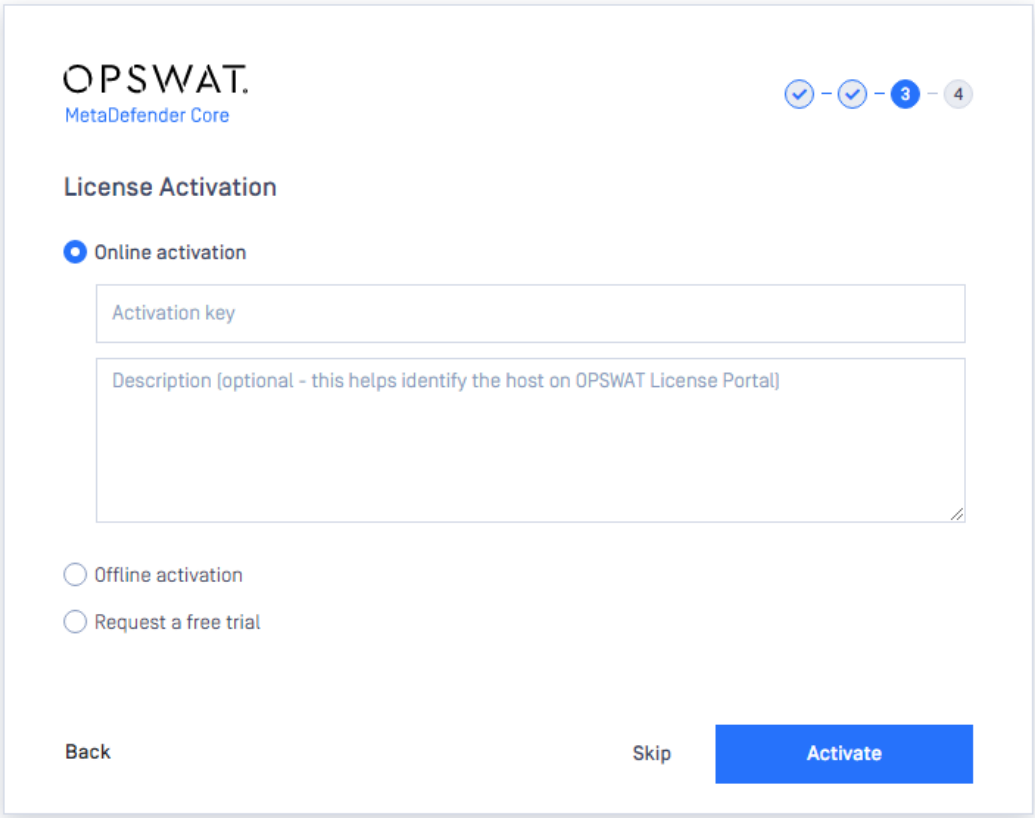
> **Notice**
> Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you alter the system where MetaDefender Core relies on (e.g. network card, OS hostname).
>
> Deployment ID is an UUID version 4 random based string. Learn more at: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you update the system where MetaDefender Core relies on.

Activation Modes

There are three different options for the product activation:

- Online activation: MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.
- Offline activation: Requiring you to upload a license file (.yml). That license file could be retrieved by activating your license via OPSWAT Portal. Follow the displayed instructions for details.
- Request trial key online: For evaluation purpose, you can receive an trial Activation key via email. Follow the displayed instructions for details.



Online activation

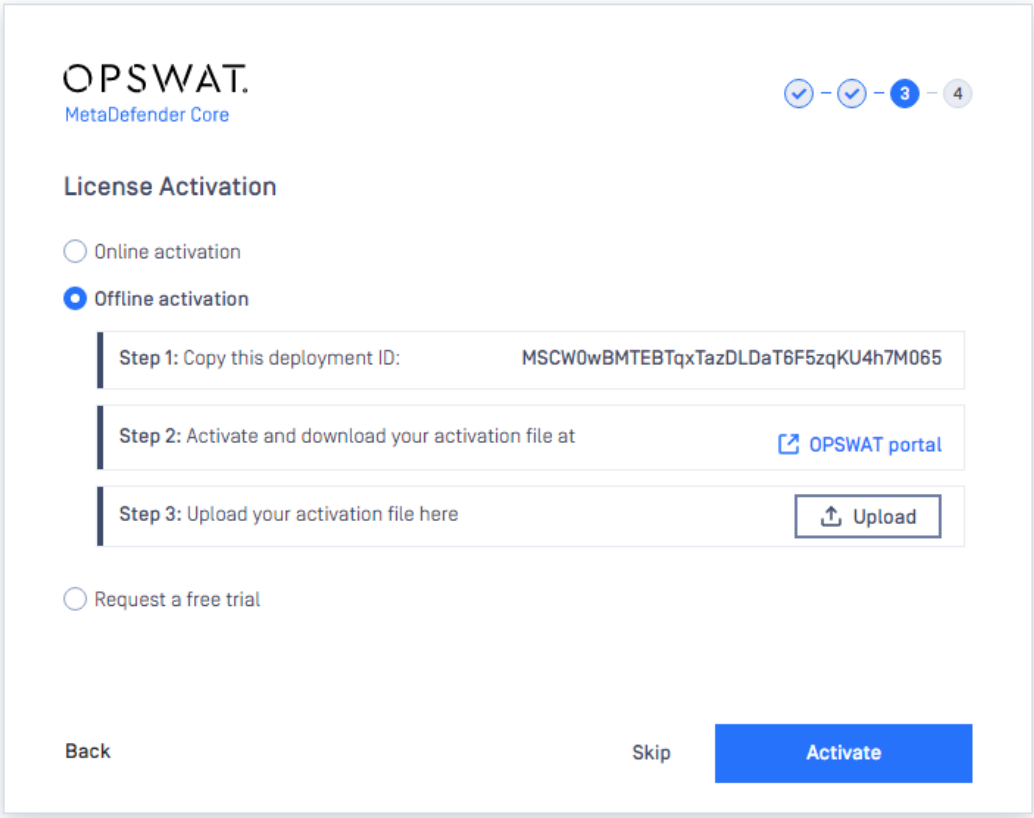
MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.

Required to input your Activation License key, and optional for description (but we recommend this for troubleshooting and auditing purpose if needed).

> **Warning**
> If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts' for license activation:
>
> - https://activation.dl.opswat.com Note: *IP address-based whitelisting on your firewall might fail after some time since OPSWAT uses CDN (Content Delivery Network) to faster delivery updates over the world, and IP address of edge servers might change over time.*

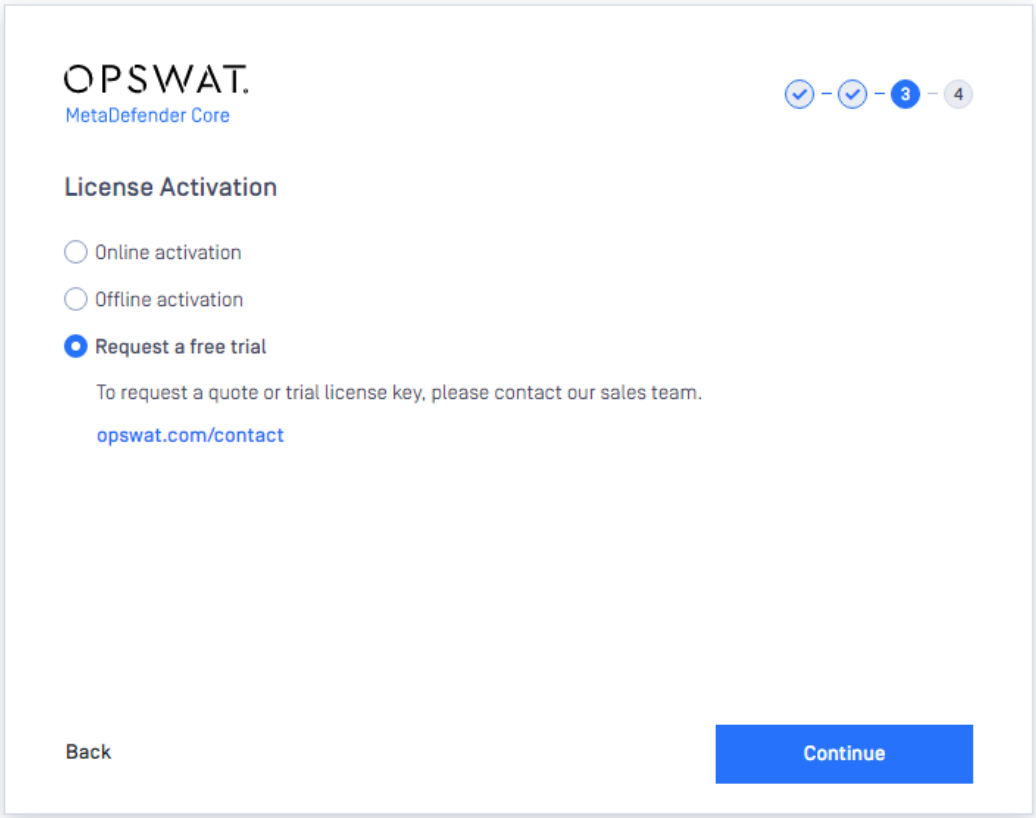
Offline activation

With no internet connection on the server the Metadefender v4 instance may be activated indirectly from a different machine, that has internet connection. The Deployment ID of the Metadefender v4 instance and the the Activation key received at the time of purchasing the product will be required. Follow the steps on the screen to activate the product offline.



Request trial key online

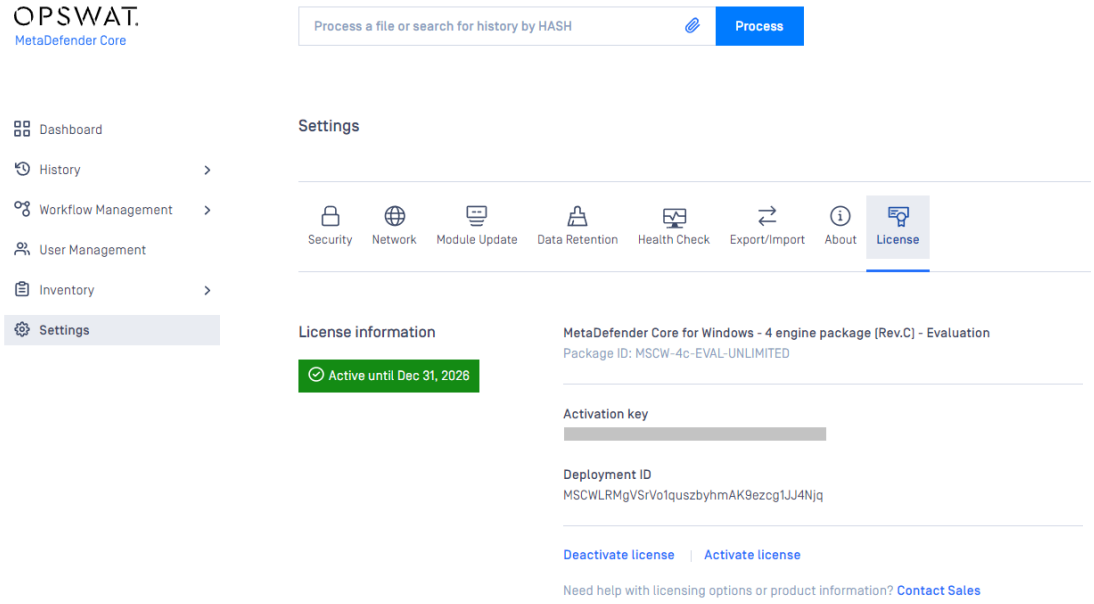
An evaluation license could be acquired by contacting our sales team.



Check Your License Details

For more license details and activating your MetaDefender Core, go to Settings > License menu tab on the Web Management Console:

- Product name: product name as on your order (e.g. MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation)
- Package ID: product identification as on your order (e.g. MSCW-4c-EVAL-UNLIMITED)
- Expiration: last day of license validity
- Activation key: key used for activating product
- Deployment ID: identification of this installation


OPSWAT.
MetaDefender Core
Process a file or search for history by HASH
Process
Dashboard
History
Workflow Management
User Management
Inventory
Settings
Settings
Security
Network
Module Update
Data Retention
Health Check
Export/Import
About
License
License information
Active until Dec 31, 2026
MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation
Package ID: MSCW-4c-EVAL-UNLIMITED
Activation key
Deployment ID
MSCWLRMgVSrVo1quszbyhmAK9ezcg1JJ4Njq
Deactivate license
Activate license
Need help with licensing options or product information? Contact Sales

Scripted license management

Using REST API calls there is an option for scripted activation and deactivation for MetaDefender Core servers with **no Internet connection or if your infrastructure is using automation to create/destroy MetaDefender Core instances**. In this chapter the steps of these two scenarios are described.

Requirements

- an installed MetaDefender Core instance without Internet connection
- another computer that has Internet connection and can run your activation scripts
- a manual or automated way to transfer data between the two computers

Activation steps

1. In order to activate MetaDefender Core, deployment ID and activation key are needed to generate the license file.
    a. Activation key should be purchased from OPSWAT.
    b. MetaDefender Core deployment ID can be queried by using REST API (for details see GET Get Current License Information API page)

> **Info**
> Save this deployment ID in your system, you might need this ID when the instance is unavailable at the time of deactivation.

2. Activate license and get the license file using the following URL, where <quantity> is 1.

activation call bash

```
curl -X GET https://activation.dl.opswat.com/activation?key=<activation-key>&deployment=<deployment unique-ID>&quantity=<quantity>
```

If the activation is successfully, a license file is downloaded. Save this file.

HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	license file	Activation was successfully.
200 Activation failed	error: '<user conform error message>'	Failed activation
200 Invalid parameter	error: 'Could not activate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not activate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
200 Invalid parameter	error: 'Could not activate your product because the quantity you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key, quantity or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Info
In case of any activation issue, contact OPSWAT support for help

3. Upload license file to MetaDefender Core: The license file should be uploaded to the MetaDefender Core v4 to activate the product. For details see page: POST Upload license key API

Deactivation steps

1. For deactivation, following infomation is required:
    a. Activation key
    b. MetaDefender Core deployment ID: can be queried by using REST API (for details see GET Get Current License Information API page)
2. Deactivate license using the following URL:

deactivation call bash

```
curl -X GET https://activation.dl.opswat.com/deactivation?key=<activation-
key>&deployment=<deployment unique-ID>
```

Response HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	result: ok	Successful deactivation
200 No active license found	error: 'Could not found any active license with the given parameters'	The license has not been activated yet or it has been deactivated already.
200 Invalid parameter	error: 'Could not deactivate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not deactivate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Important notes

Warning

Product activation is tied to several hardware, operating system and software parameters. In case of one or more major hardware or software parameter change the product might turn into deactivated status. Operating system updates, other software updates should not affecting the activation status.

If the product is online activated and has live Internet connection then in case of deployment ID change the product reactivates itself. If the product doesn't have Internet connection then the administrator is responsible to reactivate the product.

Because of the above administrator should consider licensing restrictions of the offline deployments if using any virtualization and/or containerization technologies.

File Processing

MetaDefender Core provides REST API interface for file processing business.

There are several ways to process files with MetaDefender Core:

- Analyze Files via Web Interface
- Process Files via REST API

Product Upgrading

> **Info**
> Simply run installer file, details could be found at Installation

> **Info**
> For command line mode, upgrading (e.g. on **Linux**), prior to running the installer file, please make sure to **create the ignition file** with PostgreSQL server information included. Details:
>
> - Standalone DB mode: Command Line Installation (Windows and Linux)
> - Shared DB mode: Command Line Installation

Recommendation for MetaDefender Core upgrade

Our goal is to help you execute the product upgrade continuously, efficiently, and correctly.

PostgreSQL service upgrade

- **Bundled PostgreSQL Service**: If you are using the bundled PostgreSQL server included in the installer, the PostgreSQL service will be automatically upgraded to the newer version (if available). No additional action is required for this process.
- **Remote PostgreSQL Service**: For remote PostgreSQL servers, customers are responsible for managing and upgrading the service independently. Upgrading MetaDefender Core does not update the remote PostgreSQL service.

Before the upgrade

Prepare a proper disaster recovery plan (DRP) including some common practices:

- Backup, or take a snapshot of your current working MetaDefender Core server.
  - This is to help quickly rollback for any upgrade failure might cause later for any reason.
- Optionally backup your PostgreSQL database, especially when using remote shared database mode.
  - Reference: https://www.postgresql.org/docs/current/backup.html
  - Please do it after stopping all linked MetaDefender Core instances to avoid potential data collision.
- Lastly, if backing up entire server is not feasible, then at least export MetaDefender Core configuration package Import/Export configuration just in case you'll need to start off a new server with current MetaDefender Core configuration to keep the scanning service running without any impact.

On the PostgreSQL server, double check server resource to ensure sufficient RAM memory and disk space to be preserved for the upgrade process.

- 8GB RAM available at least during the upgrade.

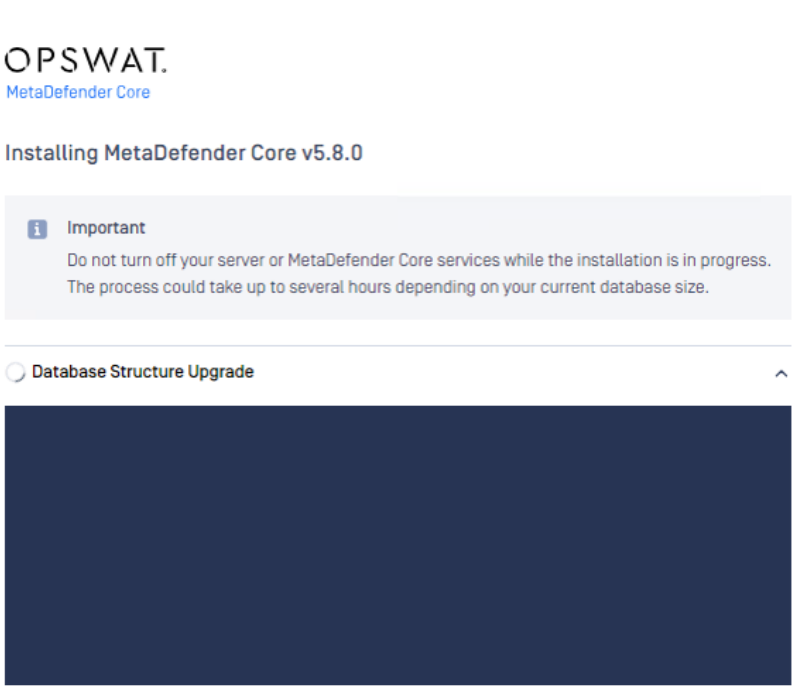
- Disk availability is greater than MetaDefender Core database size.

While doing the upgrade

Applicable for shared database deployment:

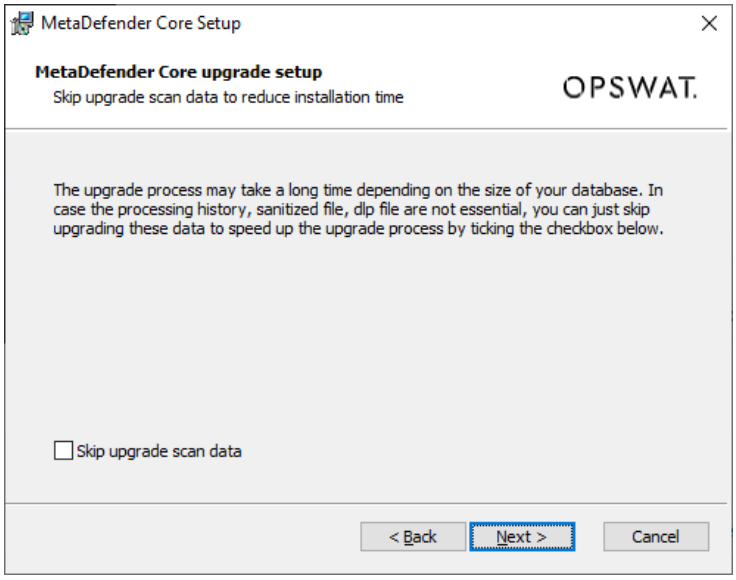
- Stop all MetaDefender Core instances' scanning service before performing the upgrade.
- Upgrade one (1) MetaDefender Core instance at a time.
- Proceed to upgrade MetaDefender Core instance where the bundle local PostgreSQL server is used for shared database setup.
- Otherwise is using remote PostgreSQL then the upgrade order does not matter.

For large size database, mostly for scan history, expecting long time to be spent on the upgrade process.



The product supports an option to skip data migration for scan history to expedite the process drastically and mitigate any failure risk. You can manually backup all data from your PostgreSQL server for offline usage if any.

Check "Skip upgrade scan data" to enable this feature while doing the upgrade.



or add following key in the ignition file:

none

```
[internal]
skip_upgrade_scan_data = 0
```

Troubleshooting

If the upgrade process ever runs into an error:

- Double check your network and services, applicable to remote PostgreSQL server.
- Optionally configure MetaDefender Core to skip migrating scan history.
  - Windows
    - Open Windows registry, and navigate to HKLM\Software\OPSWAT\Metascan
    - Add a new key `internal/skip_upgrade_scan_data` with value = 1
  - Linux
    - Open MetaDefender Core configuration file `/etc/ometascan/ometascan.conf`
    - Add new key into the file
      - [internal]
      - skip_upgrade_scan_data=1
  - Note: This key will be auto removed upon the upgrade finishes.
  - Then restart MetaDefender Core service to retry the upgrade.
    - Might retry this a few times until the upgrade is done.

Just in case the upgrade process still fails, please contact OPSWAT support with the support package - How to Create Support Package? This will help us analyze the root cause and figure out an optimal solution for your system failure.

Using external Load Balancer

Application Load Balancing (Layer 7)

This page provides information about recommended ways to use sessions with a Layer 7 load-balancer.

Most load-balancers have the ability to provide cookies for pinning subsequent traffic from a client to the appropriate server. This method is called sticky session, session persistence or session affinity. Using cookies efficiently requires the client to know when a cookie should be sent, should not be sent or should be deleted.

Sticky session load balancing

Each Metadefender Core instance has its own databases and application sessions that cannot be seen by other instances. Therefore, in order to get the related data/response to our queries we should ask the appropriate Metadefender Core v5 server. On the other hand, to keep the advantages of the used load balancing method, cookies should not be sent if it is not necessary.

Single file scanning

Step	Stage	Task	Cookie usage
1.	Sending file	Initiate processing a file on the client side. Send file through the load-balancer. (See POST File Analysis API)	Cookie should not be sent
2.	Sending file	Save the cookie and the data_id you got from the load-balancer	Save cookie
3.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Send cookie saved in step 2
4.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Send cookie saved in step 2

Batch scanning

Step	Stage	Task	Cookie usage
1.	Open batch	Initiate processing file(s) in batch. Request a batch ID through the load-balancer. (See POST Initiate Batch API)	Cookie should not be sent
2.	Open batch	Save the cookie and the batch_id you got from a Core server through the load-balancer.	Save cookie
3.	Sending files	Send file through the load-balancer. (See POST File Analysis (using batch header) API)	Send cookie saved in step 2
4.	Sending files	Save the data_id you got from the load-balancer.	-
	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Send cookie saved in step 2
5.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
6.	Getting batch status	See GET Batch Status API	Send cookie saved in step 2
7.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Send cookie saved in step 2
8.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API	Send cookie saved in step 2

If it does not matter which upstream server responds, the queries should be sent without cookie.

> **Info**
> It is recommended not to send cookies when it's not necessary to allow load-balancer to use its own method to share the load between Metadefender Core v5 servers.

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration through load-balancer

OPSWAT products that support HTTP load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	4.3.4	-
OPSWAT Client	Windows: 7.6.247.0 Mac: 10.4.243.0	-
MetaDefender Email Security	4.3.0	-
MetaDefender ICAP Server	4.3.0	-
MetaDefender Vault	1.3.0	-
MetaDefender for Secure Storage	2.0.7	-

DNS Load Balancing

Using this method is logically similar to Layer 7 load-balancing.

Briefly how it works

Client uses a domain name to send a query to a server. Client's DNS server has more "A" records for that name with different IPs. When a client resolves the server's domain name DNS server randomly chooses an IP for that name to send back. When a session is used on the application layer, client should know the IP address of the Core server that handles that specific session. In every other case, client should resolve the domain name with DNS query to let requests to be balanced between Metadefender Core servers.

Single file scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Sending file	Initiate processing a file on the client side. (See POST File Analysis API)	Use IP saved in step 2
4.	Sending file	Save the data_id got from the Core server	-
5.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Use IP saved in step 2
6.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Use IP saved in step 2

Batch scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Open batch	Initiate processing file(s) in batch. Request a batch ID. (See POST Initiate Batch API)	Use IP saved in step 2
4.	Sending files	Send file to the specific Core server with the basch ID saved in step 3. (See POST File Analysis (using batch header) API)	Use IP saved in step 2
5.	Sending files	Save the data_id you got from the Core server.	-
6.	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Use IP saved in step 2
7.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
8.	Getting batch status	See GET Batch Status API	Use IP saved in step 2
9.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Use IP saved in step 2
10.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API).	Use IP saved in step 2

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration via DNS load-balancing

OPSWAT products that support DNS load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	does not support yet	-
MetaDefender Client	does not support yet	-
MetaDefender Email Security	does not support yet	-
MetaDefender ICAP Server	does not support yet	-
MetaDefender Vault	does not support yet	-

SSL connection for PostgreSQL communication

Note
The SSL connection configuration is possible thanks to the support made by PostgreSQL. For reference: https://www.postgresql.org/docs/14/ssl-tcp.html

MetaDefender Core is tested against this mode on particular version 4.21.1

Step 1: Generate SSL certificate for PostgreSQL server

Note
Following instructions are for self-sign certificate. For production environment, making sure to use the certificate signed by your trusted CA.

1) Generate a private key

bash

```
openssl genrsa -des3 -out server.key 1024
```

Sample output:

bash

```
Generating RSA private key, 1024 bit long modulus ........++++++
...............++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

Type your desired password / pass phrase to encrypt the private key

2) Remove the pass phrase to automatically start up the PostgreSQL server

bash

```
openssl rsa -in server.key -out server.key
```

> **Note**
> On Linux, make sure to set permission on the server.key file

bash

```
chmod og-rwx server.key
```

3) Create a self-signed certificate

bash

```
openssl req -new -key server.key -days 3650 -out server.crt -x509
```

You will be prompted to enter detailed information which is incorporated into your self-signed certificate request.

none

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:CA
Locality Name (eg, city) [Default City]:San Francisco
Organization Name (eg, company) [Default Company Ltd]:OPSWAT
Unit Name (eg, section) []:OPSWAT
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:dummy_email@opswat.com
```

For self-signed certificate, use the server ceriticate as the trusted root certificate:

bash

```
cp server.crt root.crt
```

Copy **server.key**, **server.crt**, **root.crt** to PostgreSQL data folder. For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data**
- Linux: **/var/lib/ometascan/pg_data/**

Step 2: Configure PostgreSQL server for SSL authentication connection

1) Create a custom config file (e.g. ssl.conf) for PostgreSQL.

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\postgres\ssl.conf**
- Linux: **/var/lib/ometascan/postgres/ssl.conf**

2) Content of ssl.conf:

none

```
ssl = on
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
```

3) Modify pg_hba.conf

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data\pg_hba.conf**
- Linux: **/var/lib/ometascan/pg_data/pg_hba.conf**

Add following (modify **::1/128** if needed to match with your environment requirement)

none

```
hostssl all all      ::1/128      scram-sha-256
```

After the modify, the content of pg_hba.conf should be like this (example):

> **Note**
> In a Linux environment, a local connection is required to upgrade from version 5.5.1 or earlier to version 5.6.0 or later.

none

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local         DATABASE  USER  METHOD  [OPTIONS]
# host          DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl     DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostgssenc    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnogssenc  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
local    all all                    scram-sha-256
hostssl all all      ::1/128      scram-sha-256

#More content.......
```

4) Reload the services

- Standalone DB mode: Restart **ometascan** service (MetaDefender Core service)
- Shared DB mode: Restart **ometascan-pg** / **ometascan-postgresql** service (MetaDefender Core PostgreSQL service) and then **ometascan** service (MetaDefender Core service)

Test your SSL database connection:

bash

```
psql.exe "sslmode=require dbname=postgres user=postgres host=localhost port=5432"
Password for user postgres:
psql (12.7)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
Type "help" for help.
```

This message indicate that you are now using SSL connection:

bash

```
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
```

Performance and Load Estimation

> **Disclaimer**
> These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Some factors that affect performance

- MetaDefender Core version
- MetaDefender Core engine package and configuration
  - set of engines (which and how many)
  - product configuration (e.g., thread pool size)
- System environment
  - server profile (CPU, RAM, hard disk)
  - client application location - remote or local
  - system caching and engine level caching
- Dataset
  - encrypted or decrypted
  - file types
    - different file types (e.g., document, image, executable)
    - archive file or compound document format files
  - file size
  - bad or unknown (assume to be clean)
- Performance tool

How test results are calculated

Performance (mainly scanning speed) is measured by throughput rather than unit speed. For example, if it takes 10 seconds to process 1 file, and it also takes 10 seconds to process 10 files, then performance is quantified as 1 second per file, rather than 10 seconds.

- total time / total number of files processed: 10 seconds / 10 files = 1 second / file.

Standalone DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.2.416.0

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

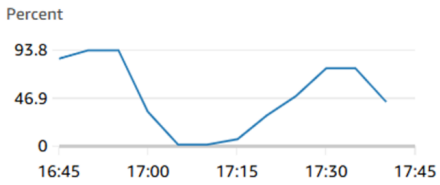
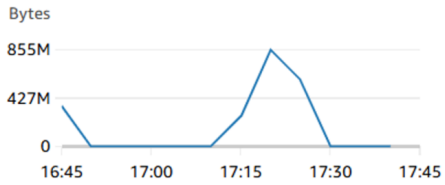
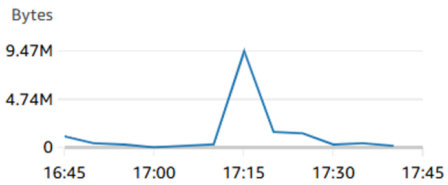
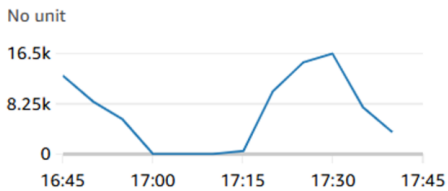
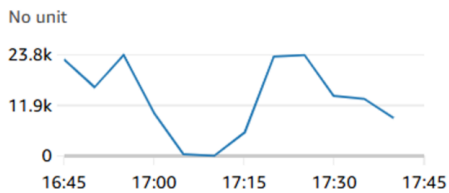
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV)	24.2	47.2 / 80	37.3 / 41.1		315,310	0.0114
Deep CDR	19.9	48.9 / 81.8	32.6 / 37		383,442	0.0094
Proactive DLP	17.1	36.8 / 58.2	30.4 / 33.6		446,228	0.0081
Vulnerability	10.1	59.2 / 85	29.3 / 33.3		755,495	0.0048

MetaDefender Core with common engine packages:

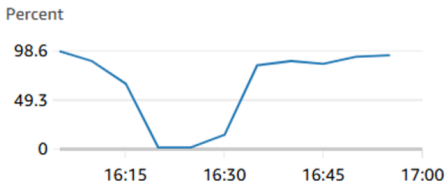
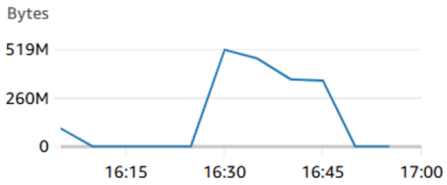
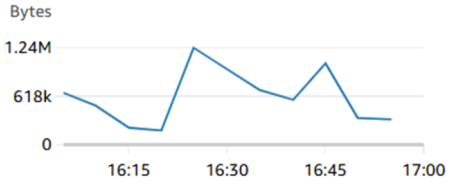
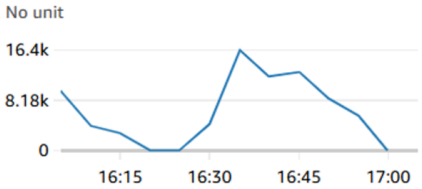
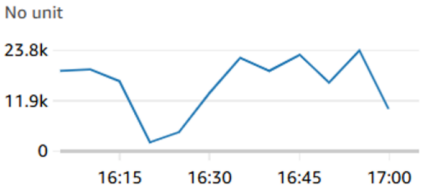
Use case	Scan duration	Avg./Max CPU usage	Avg./Max RAM usage	Avg./Max Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV) + Deep CDR	30.8	68.1 / 88	40.1 / 46.8		247,744	0.0145
Metascan (8 AV) + Deep CDR + Proactive DLP	31.2	74.2 / 98.6	41.5 / 51.1		244,567.3	0.0147
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	31.8	73.5 / 98	40.6 / 51.5		239,953	0.015

System resource utilization charts

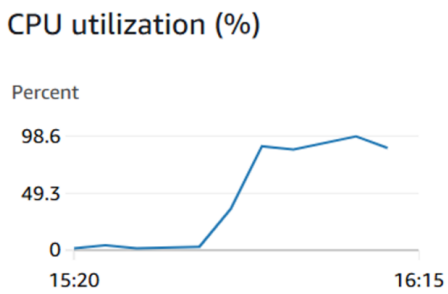
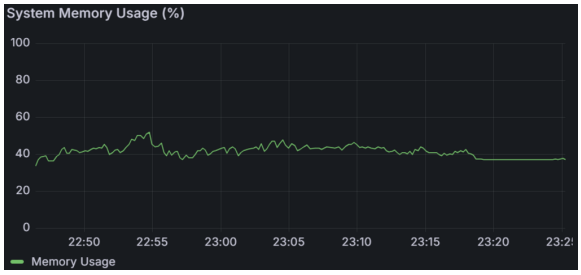
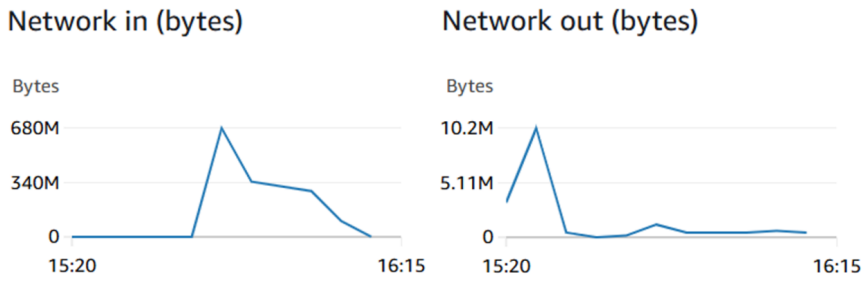
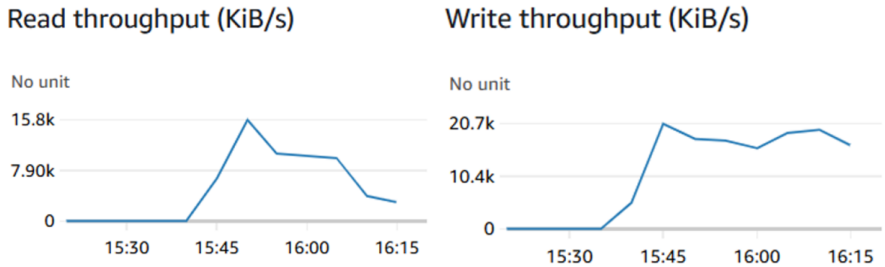
MetaDefender Core with Metascan (8 AV engines):

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

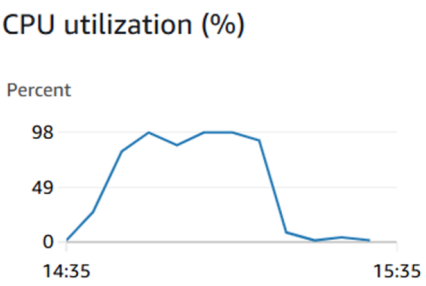
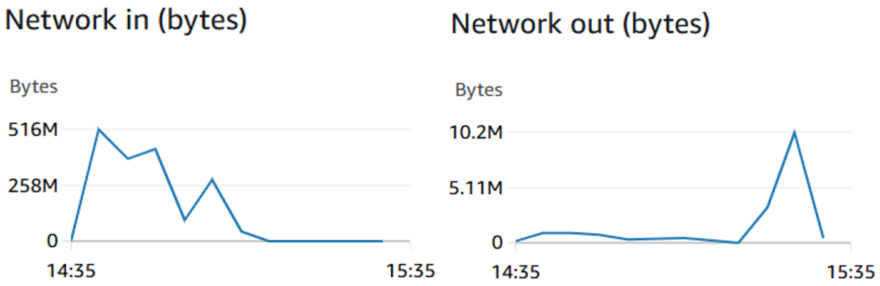
MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

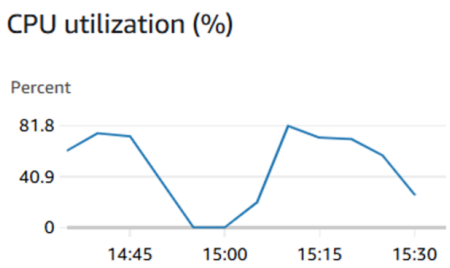
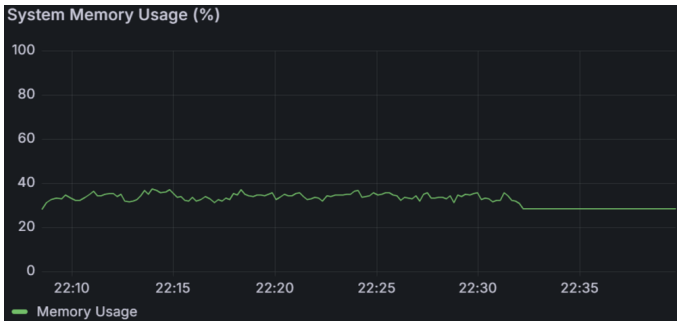
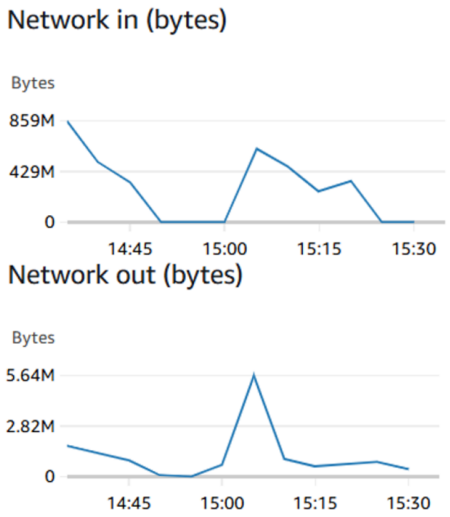
MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

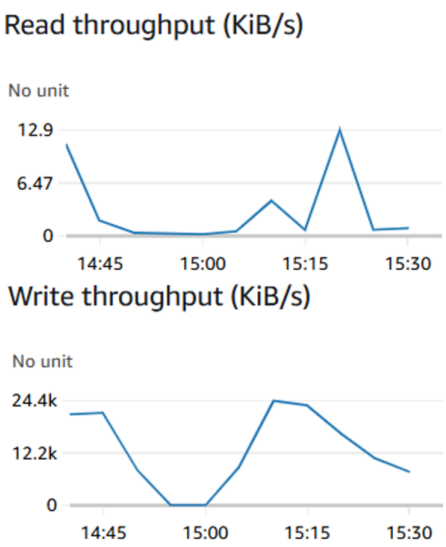
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

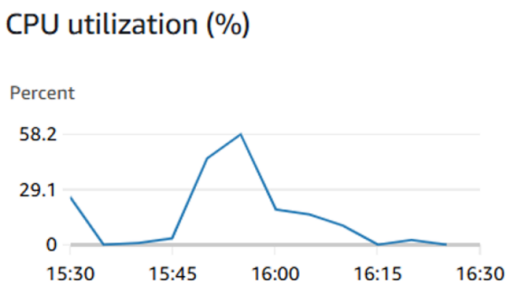
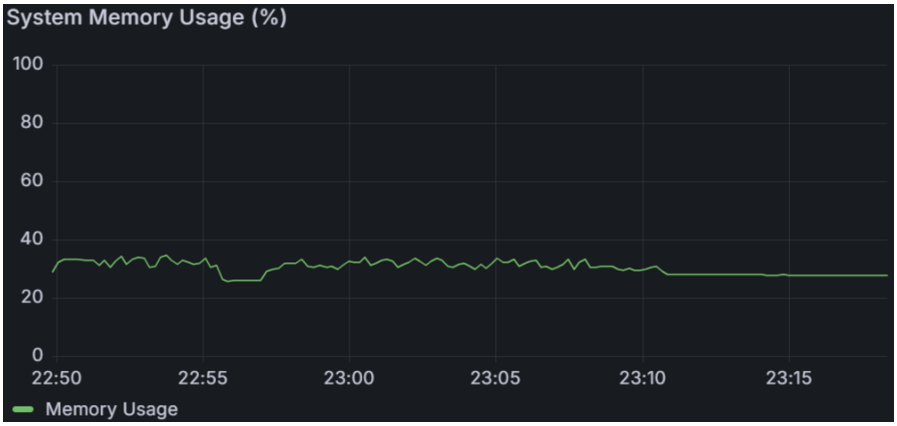
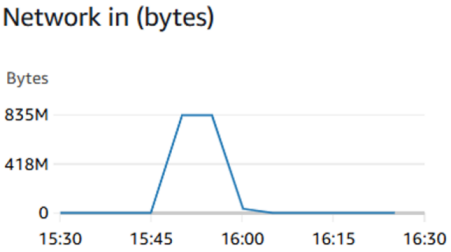
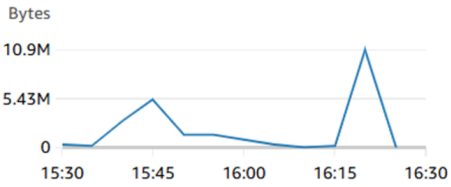
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

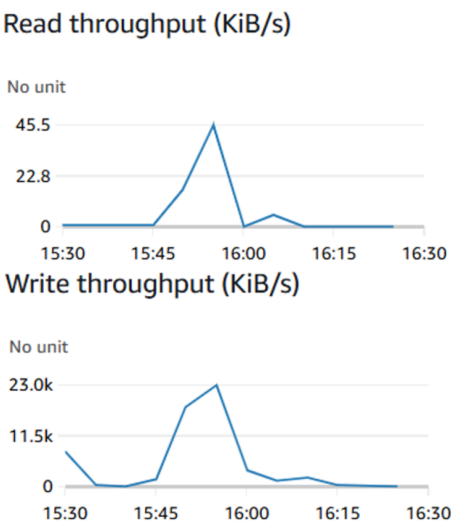
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

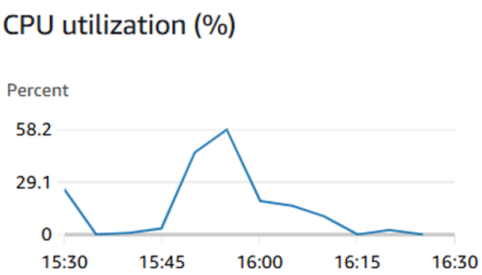
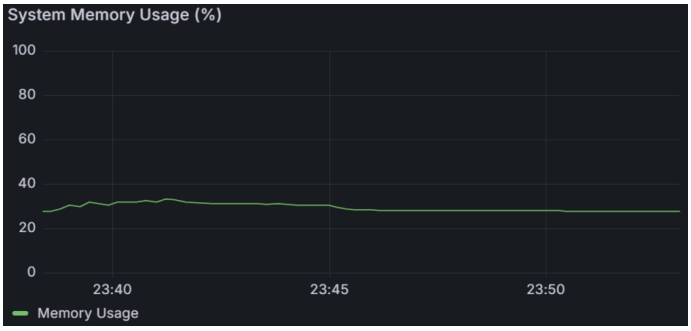
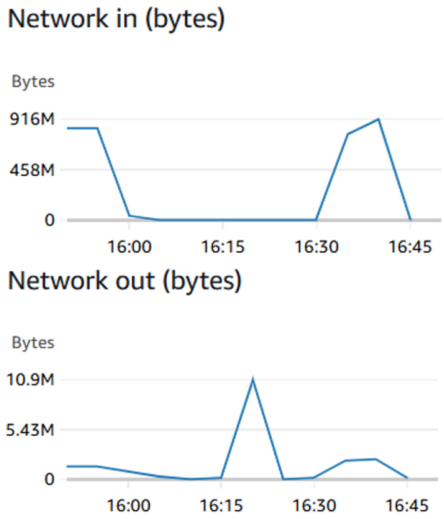
	Utilization charts
Disk usage	

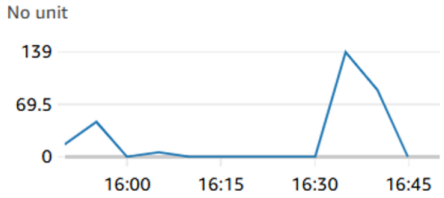
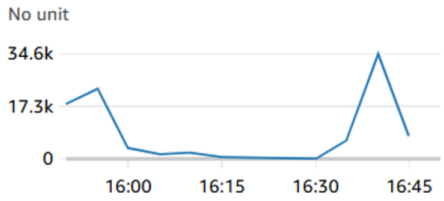
MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	 

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability:

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

Standalone DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Ubuntu 22.04	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.57-236

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

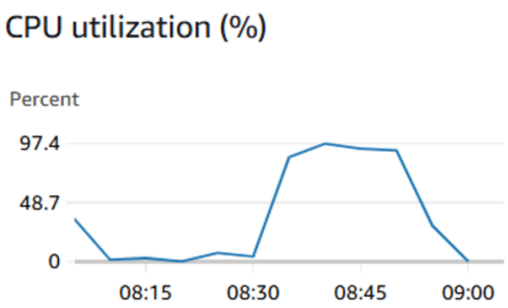
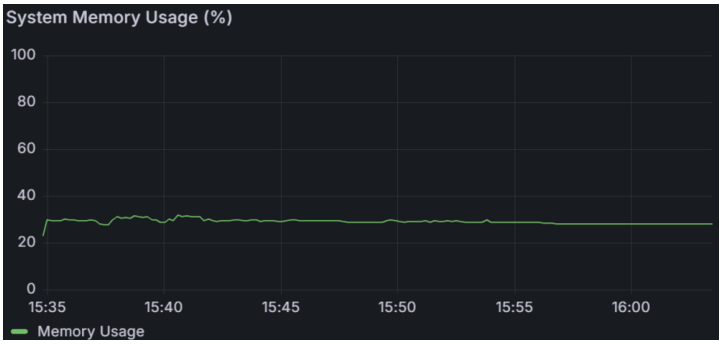
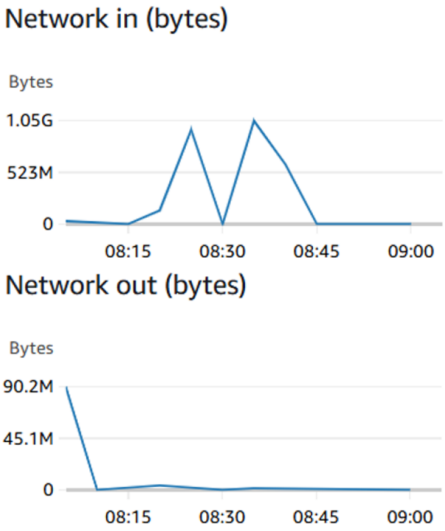
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV)	30.2	69 / 98	29 / 32		252,666	0.0142
Deep CDR	19.4	48 / 62	36 / 49		393,325	0.009
Proactive DLP	16.5	34 / 39	32 / 48		462,455	0.008
Vulnerability	12	50 / 65	46 / 46		635,875	0.006

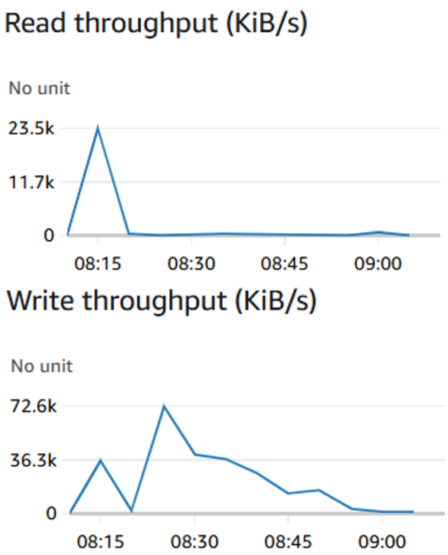
MetaDefender Core with common engine packages:

Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV) + Deep CDR	39.8	78 / 99	62 / 73		191,721	0.0188
Metascan (10 AV) + Deep CDR + Proactive DLP	43.5	61 / 84	47 / 67		175,414	0.0205
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	44.8	62 / 88	69 / 83		170,324	0.0211

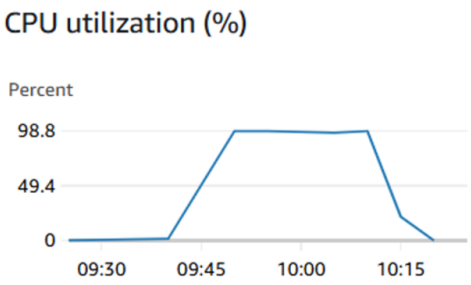
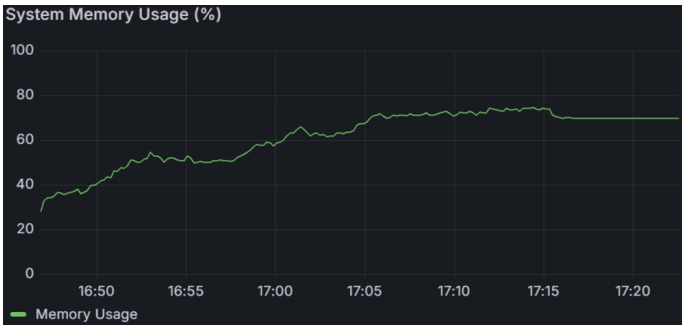
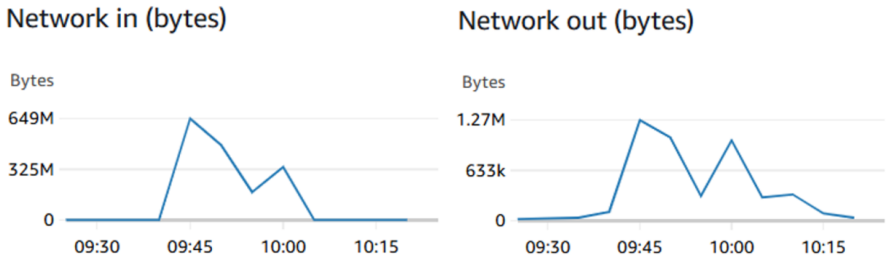
System resource utilization charts

MetaDefender Core with Metascan (10 AV engines) :

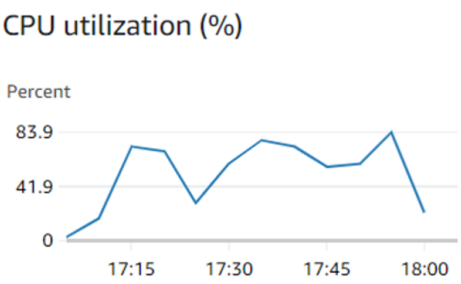
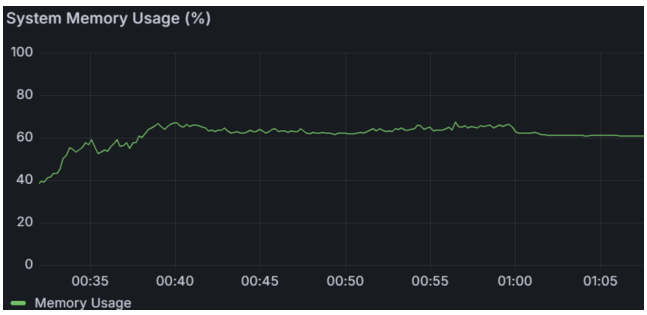
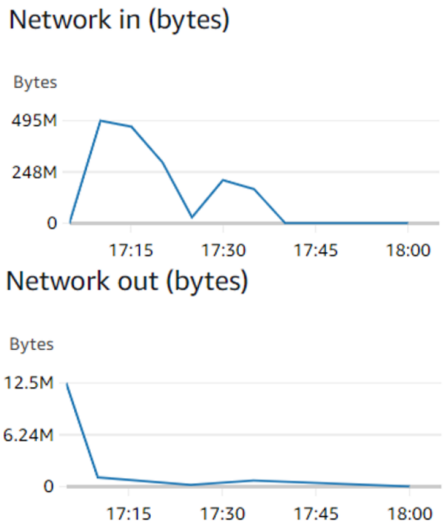
	Utilization charts
CPU usage	
Memory usage	
Network usage	

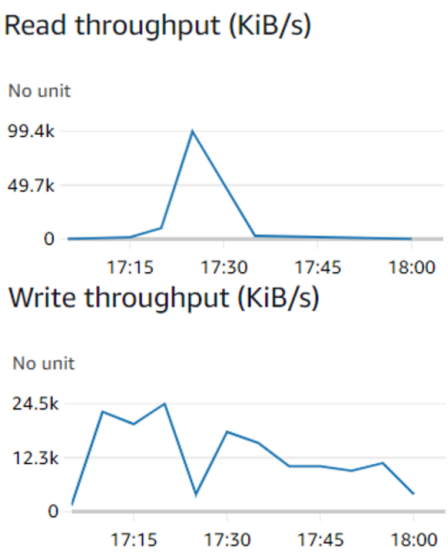
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR :

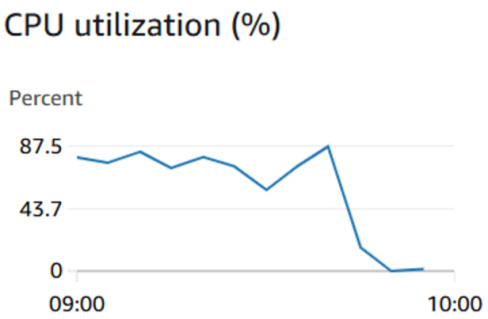
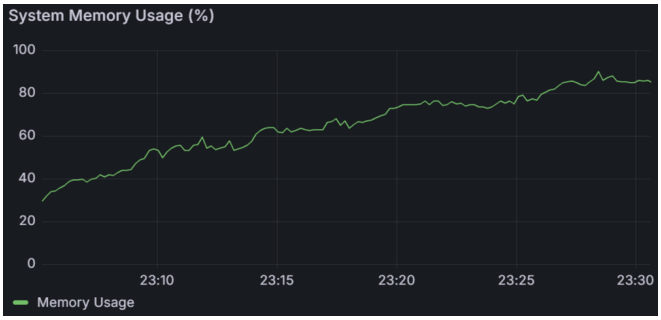
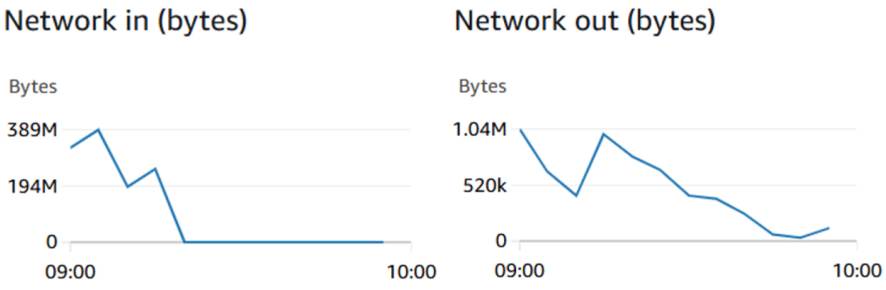
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP :

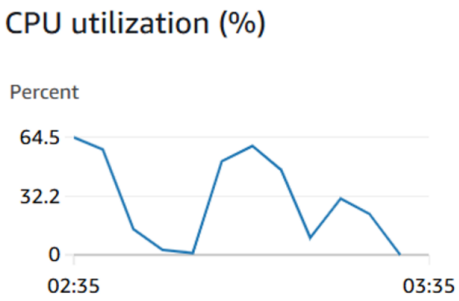
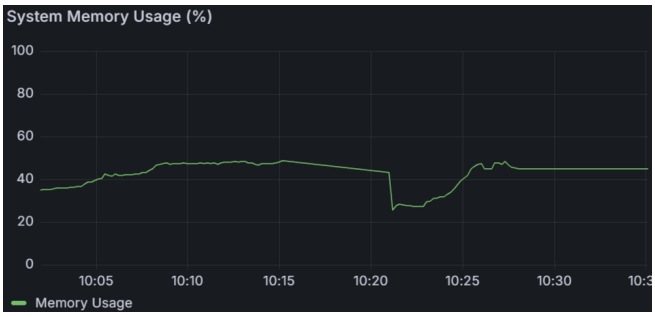
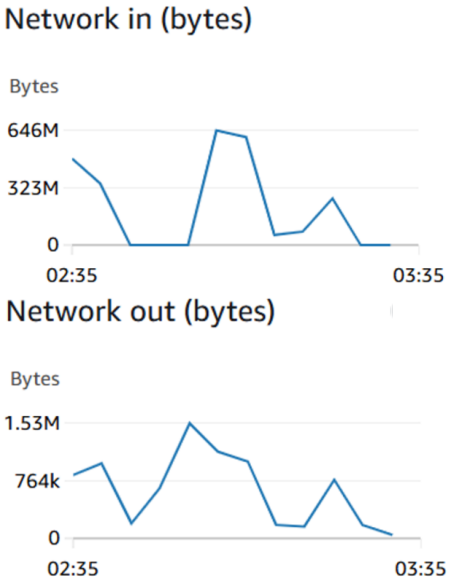
	Utilization charts
CPU usage	
Memory usage	
Network usage	

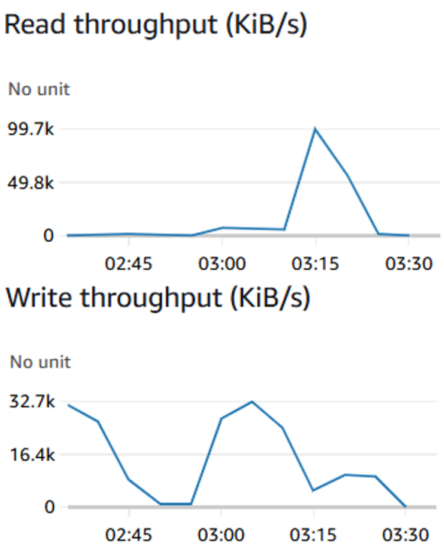
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP + Vulnerability :

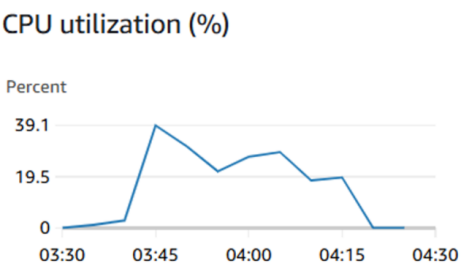
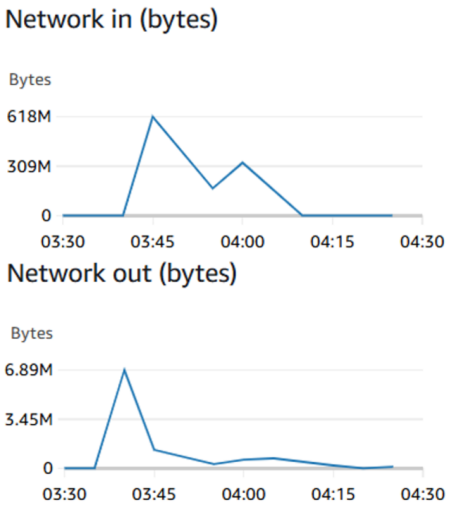
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

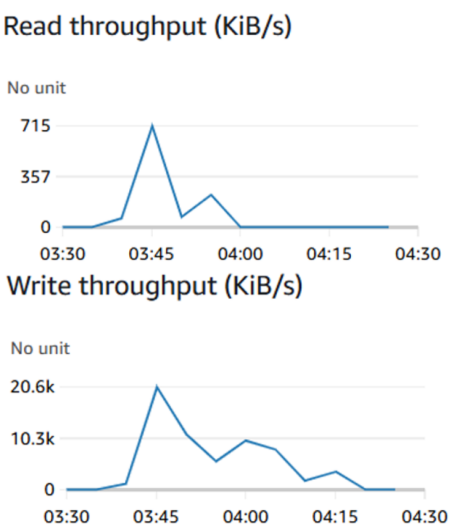
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

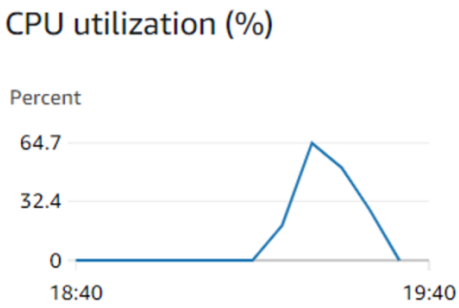
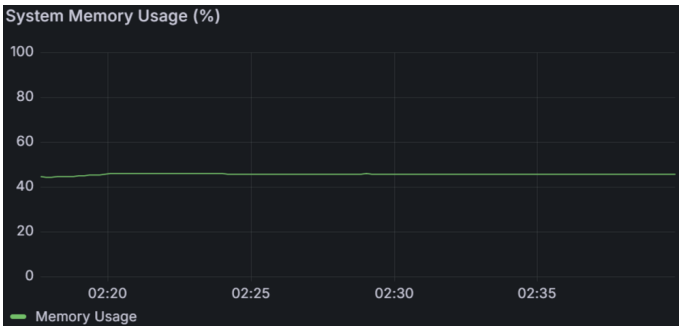
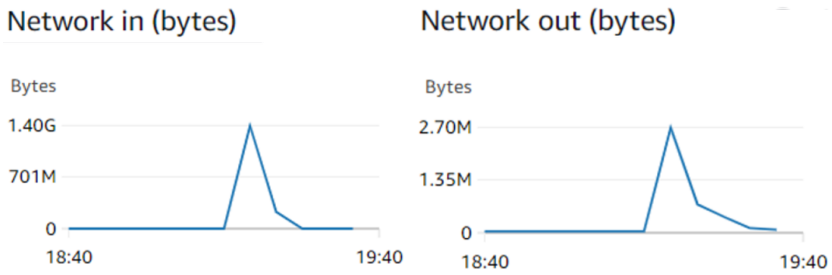
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability :

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

Shared DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

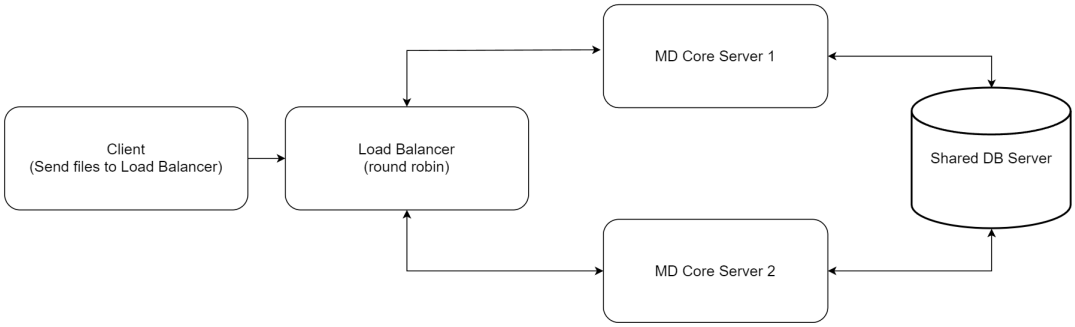
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV)	11.9	37.0	41.9	30.1	30.4	Receive: 6441.1 Send: 769.6	Receive: 6200.2 Send: 693.7	640,821.2	0.0
Deep CDR	10.5	60.1	66.7	30.3	32.2	Receive: 6379.8 Send: 594.8	Receive: 7146.1 Send: 609.6	729,368.5	0.0
Proactive DLP	7.8	38.7	37.5	29.7	31.0	Receive: 8842.6 Send: 791.8	Receive: 8635.6 Send: 729.1	965,455.7	0.0
Vulnerability	4.4	52.2	53.0	26.0	26.9	Receive: 16,215.5 Send: 1006.8	Receive: 15,785.7 Send: 997.2	1714,449.3	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV) + Deep CDR	15.4	72.4	75	34.3	35.4	Receive: 5436.6 Send: 611.3	Receive: 5589.3 Send: 607.2	495,697.4	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP	16.9	77.2	80.2	36.4	37.9	Receive: 5254 Send: 585.4	Receive: 5041.3 Send: 576.1	450,657.6	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	18.0	72.1	80.4	38.6	38.0	Receive: 4504.3 Send: 574.1	Receive: 4649.5 Send: 571.1	423,010	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

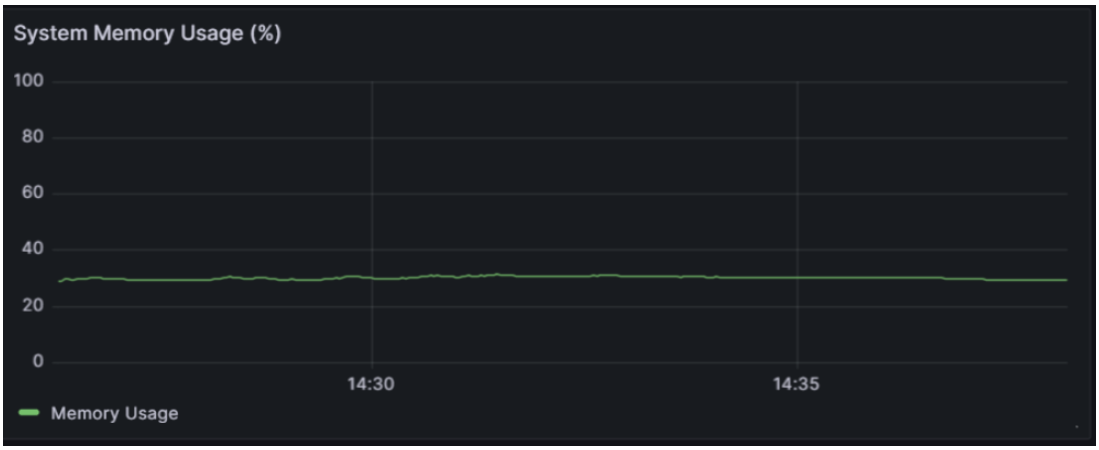
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2

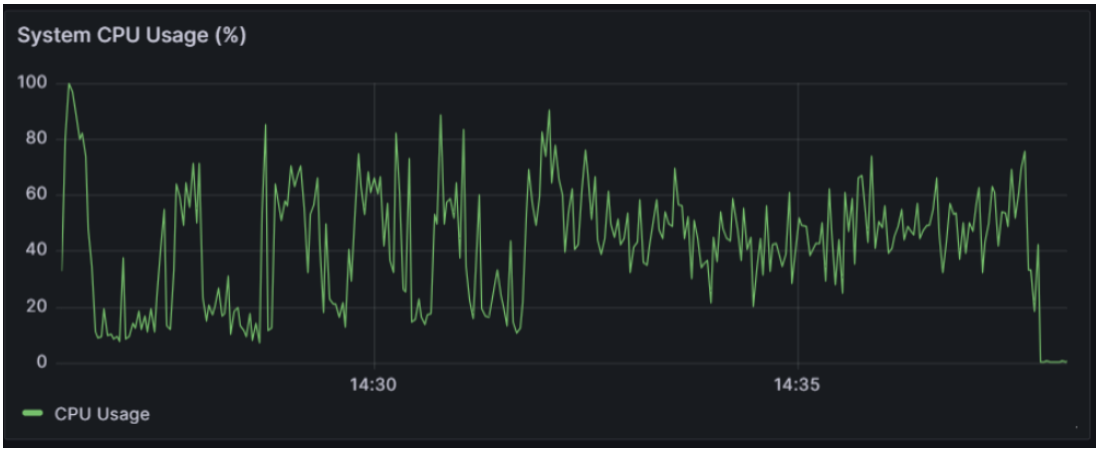


CPU usage

MD Core #1

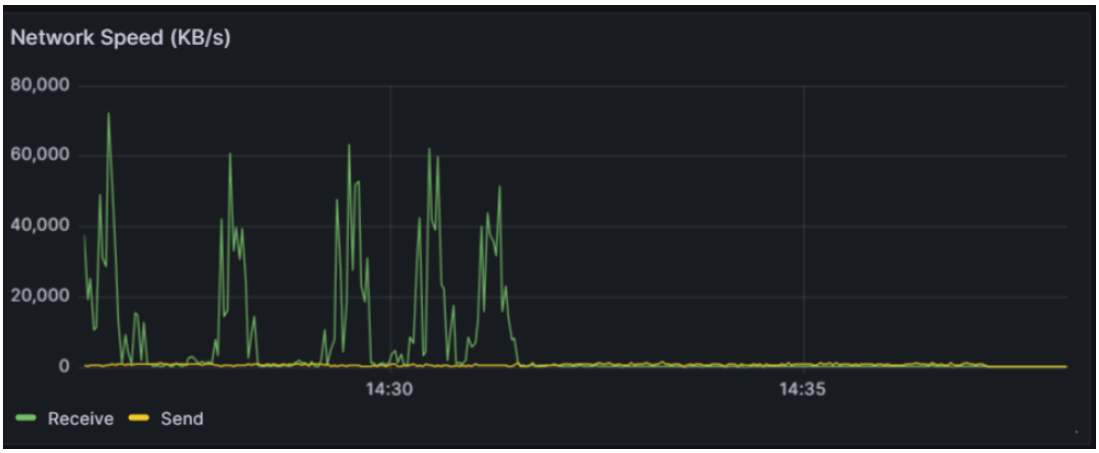


MD Core #2

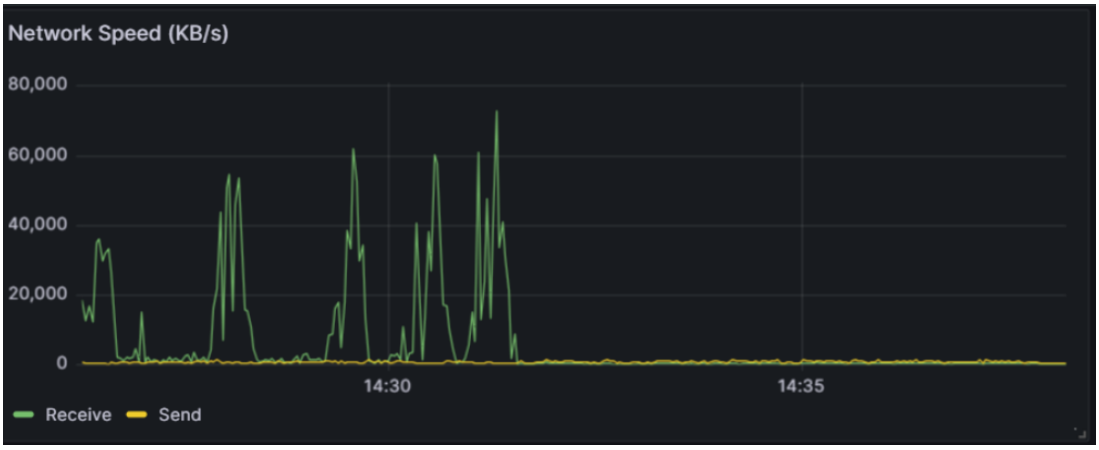


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2

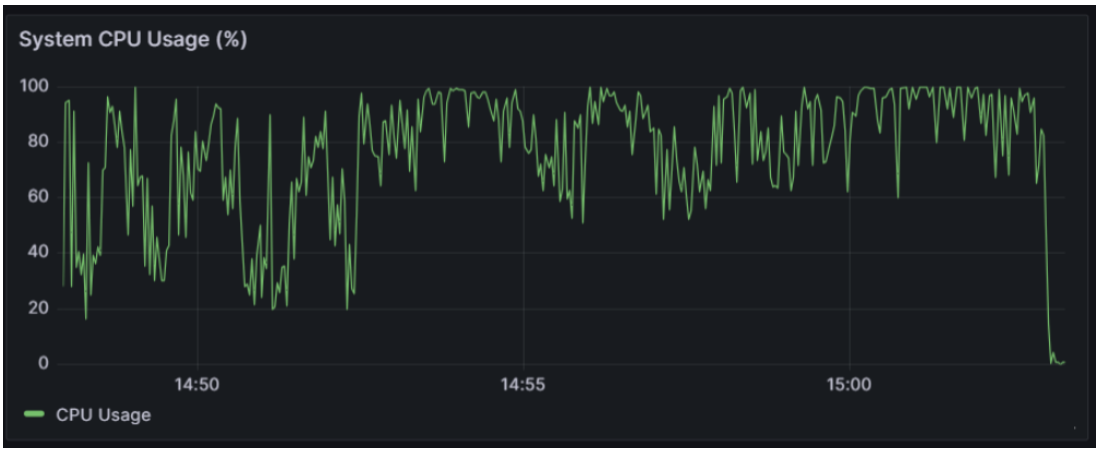


CPU usage

MD Core #1

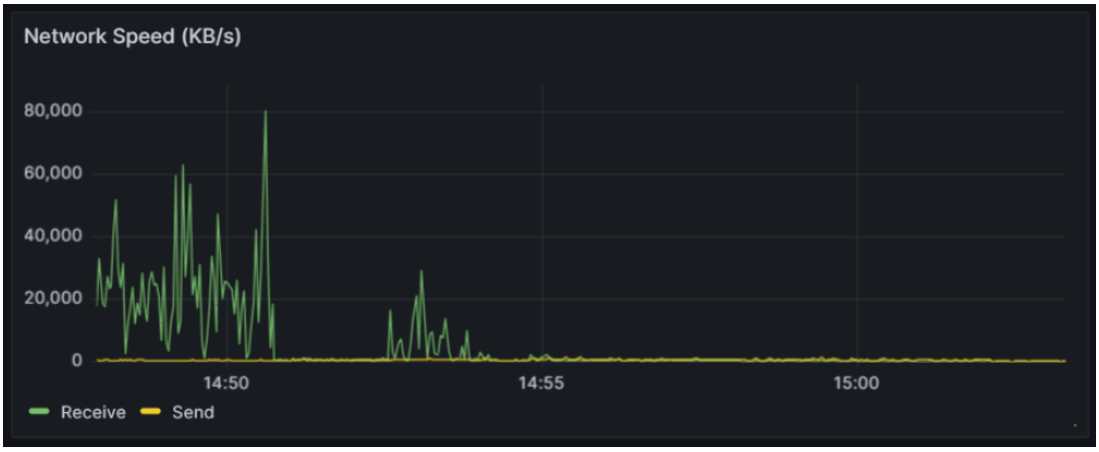


MD Core #2



Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

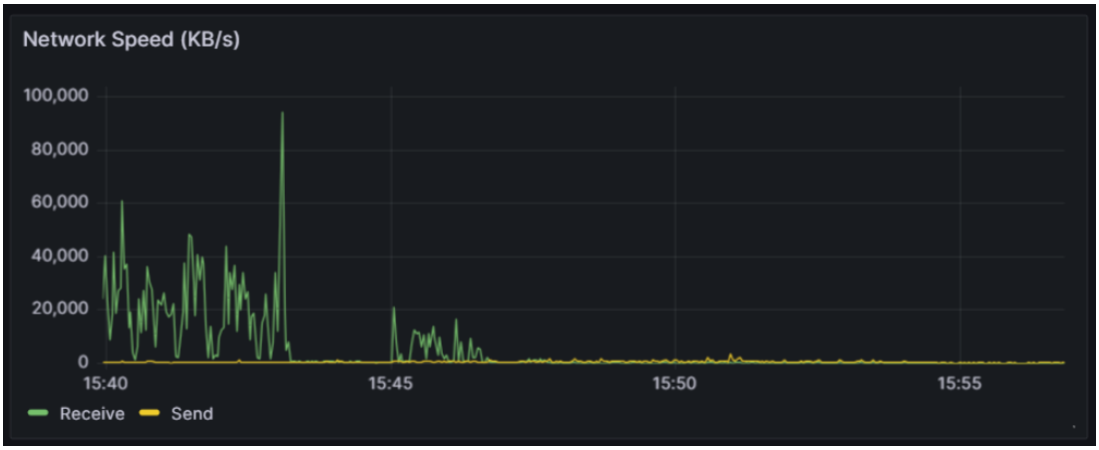


MD Core #2

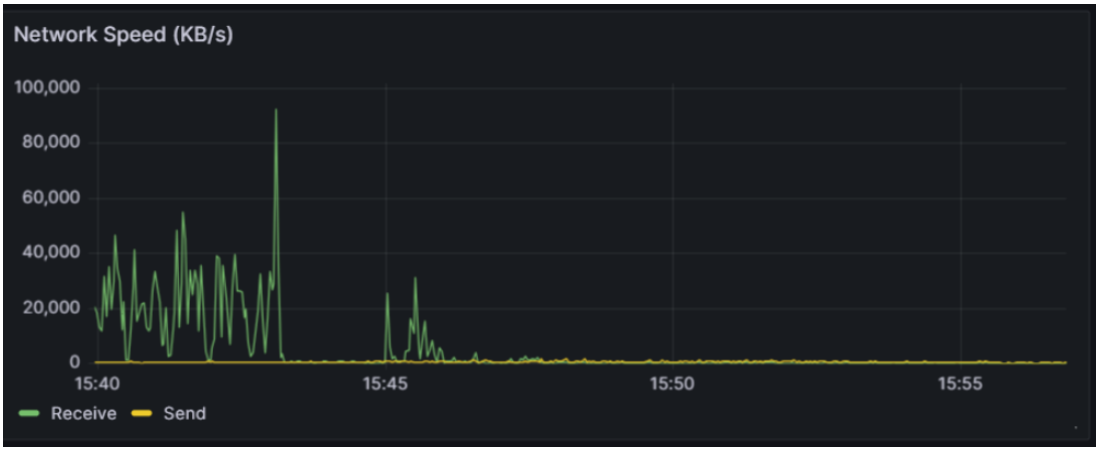


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1

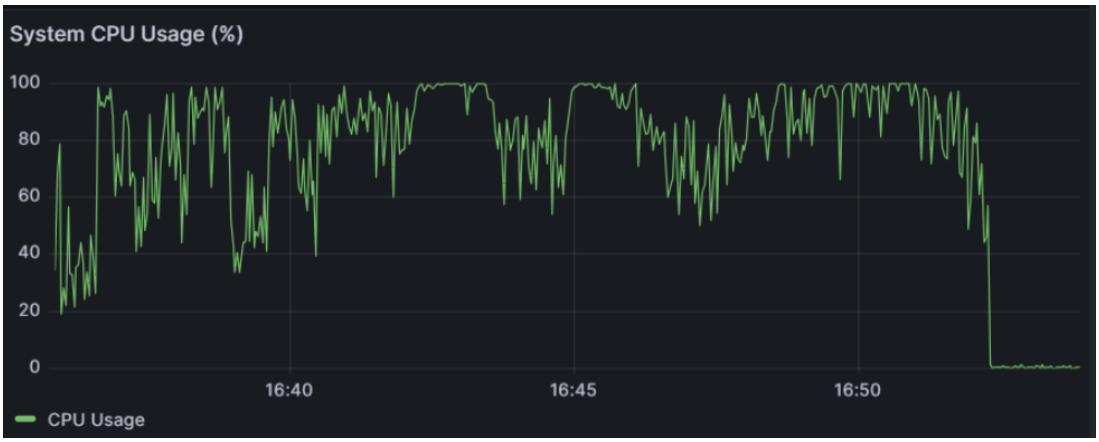


MD Core #2



CPU usage

MD Core #1

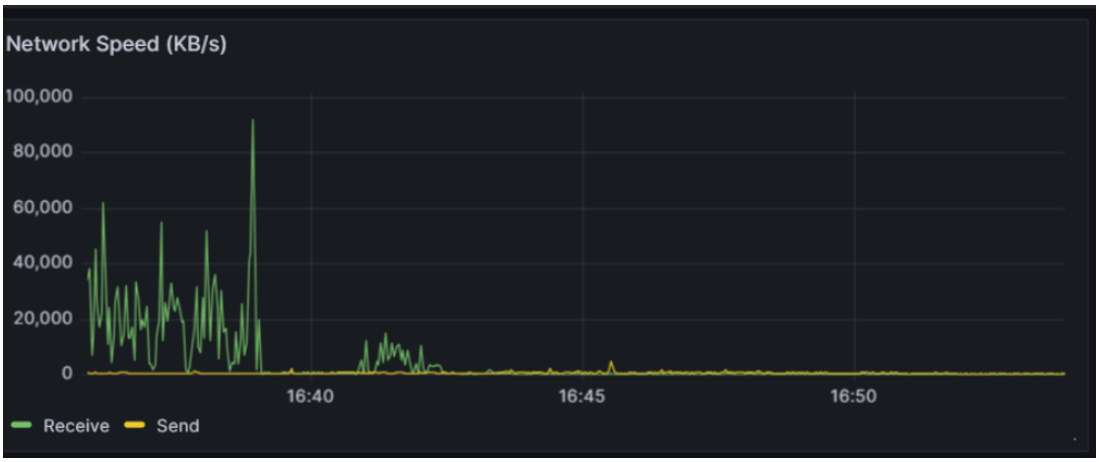


MD Core #2

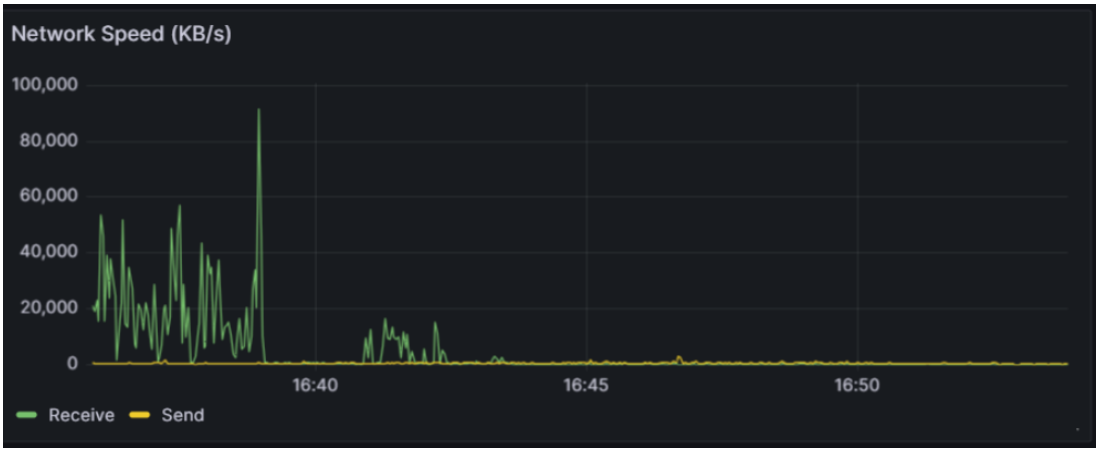


Network usage

MD Core #1



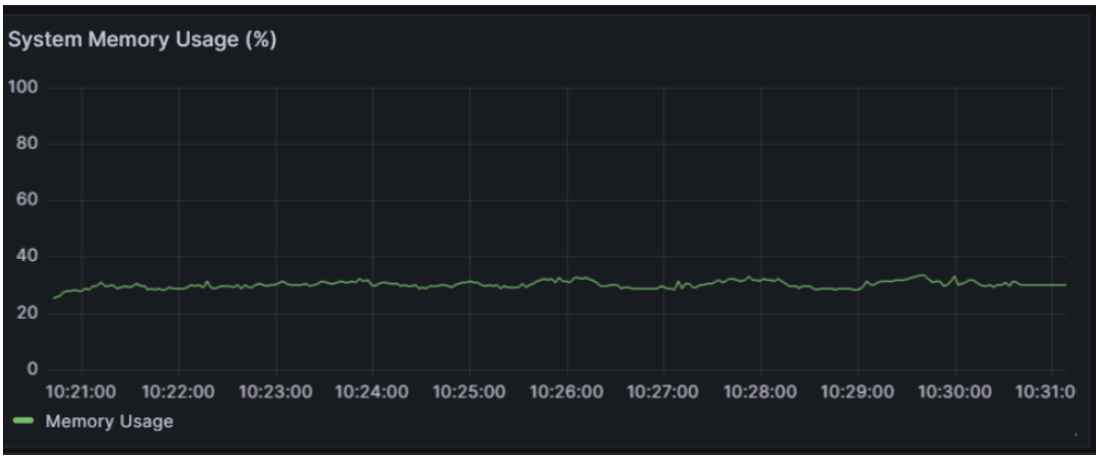
MD Core #2



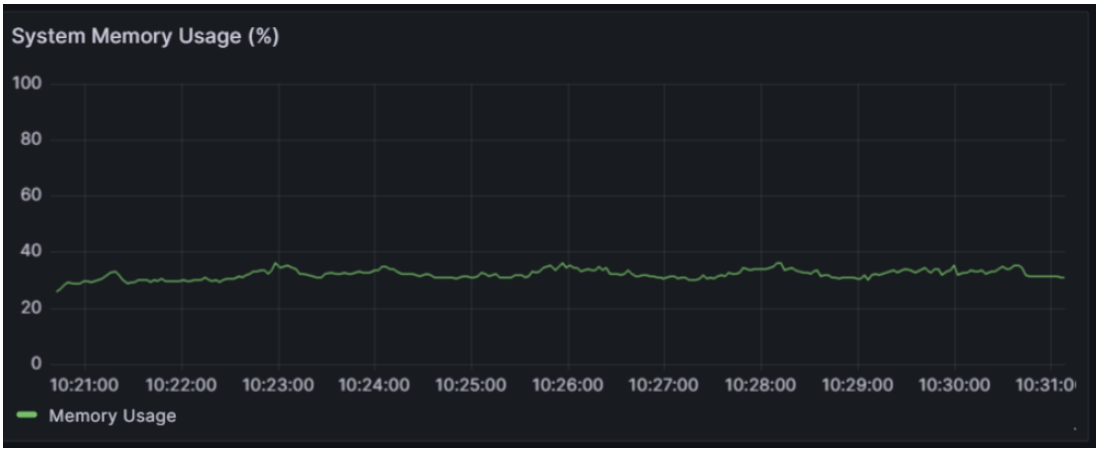
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

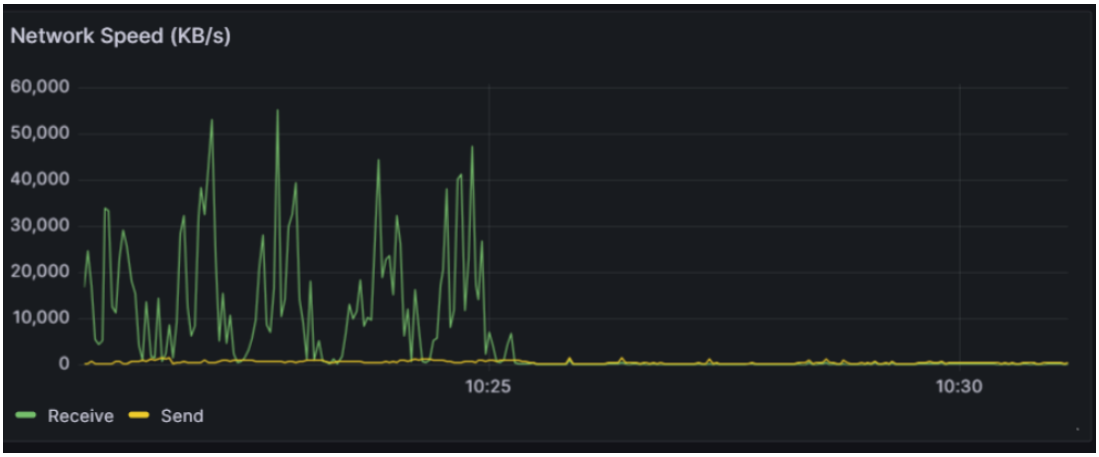


MD Core #2

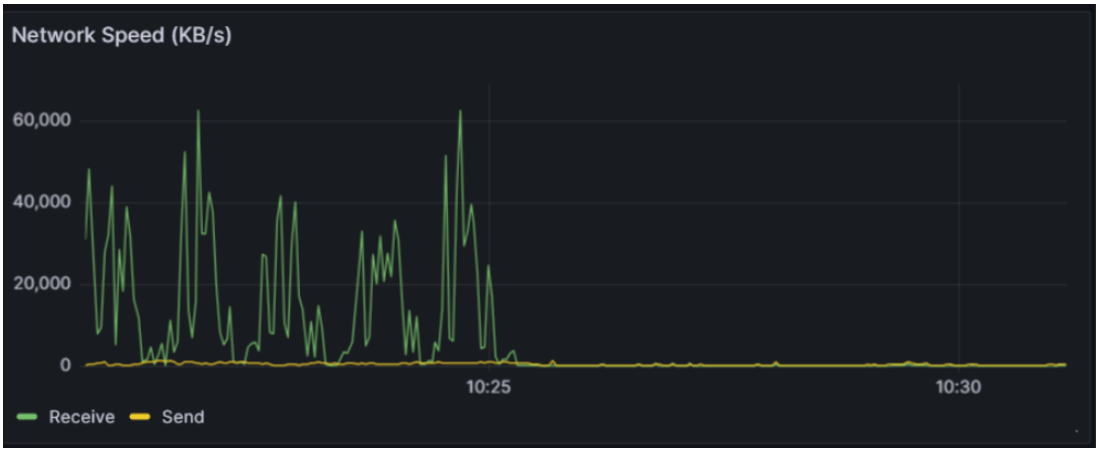


Network usage

MD Core #1



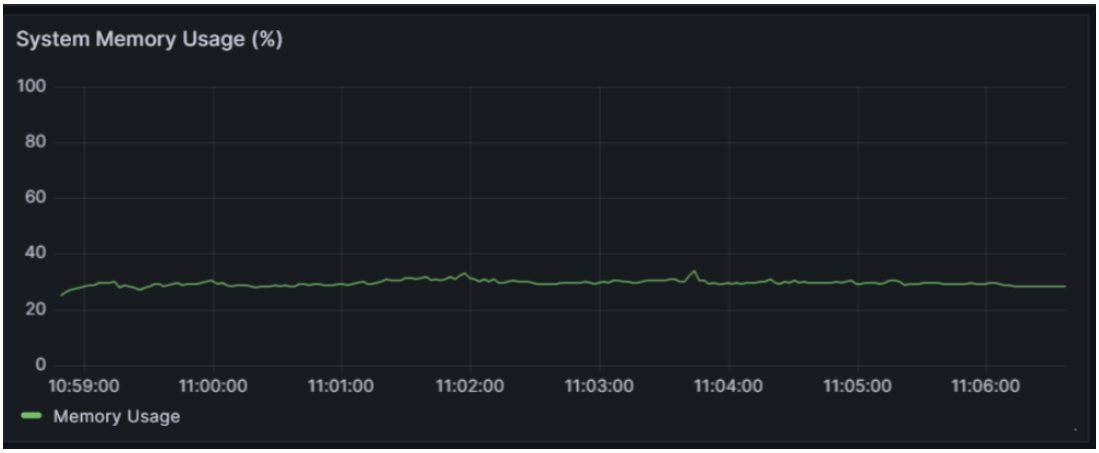
MD Core #2



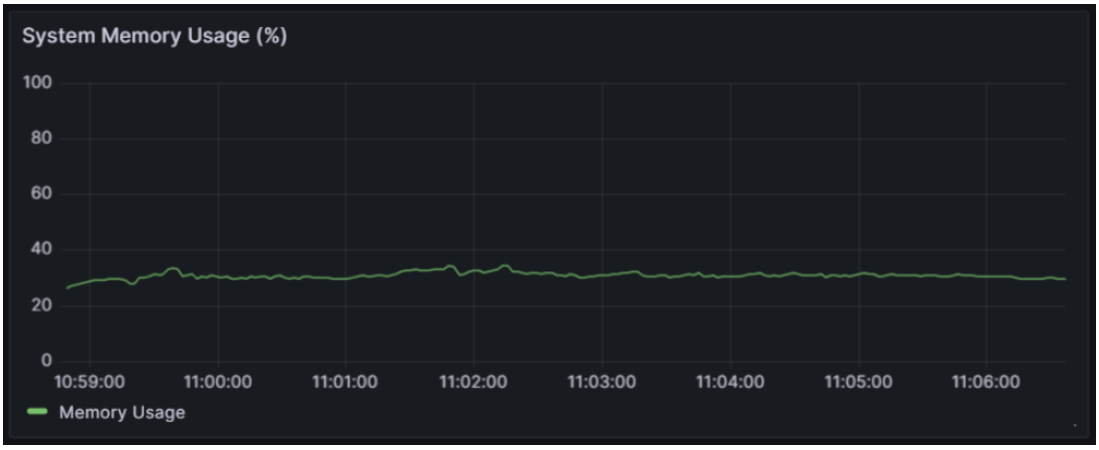
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

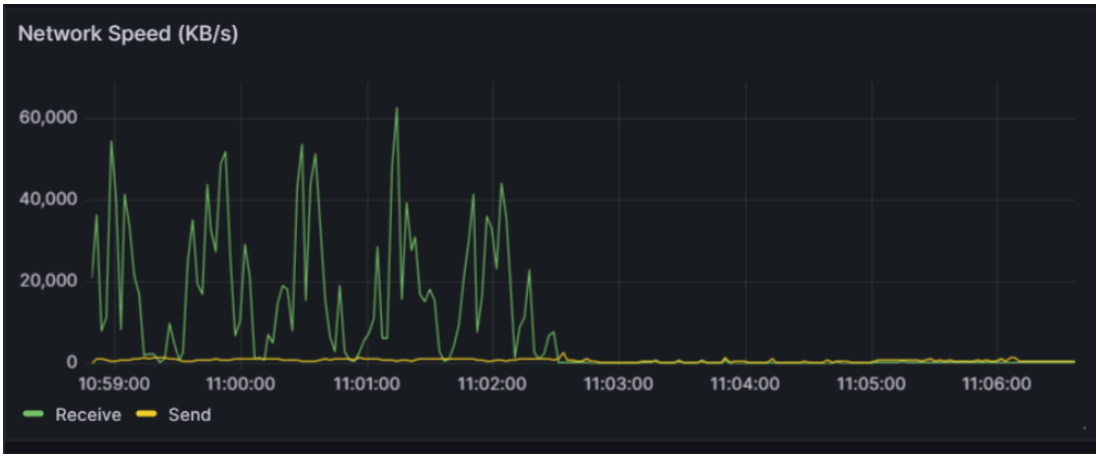


MD Core #2

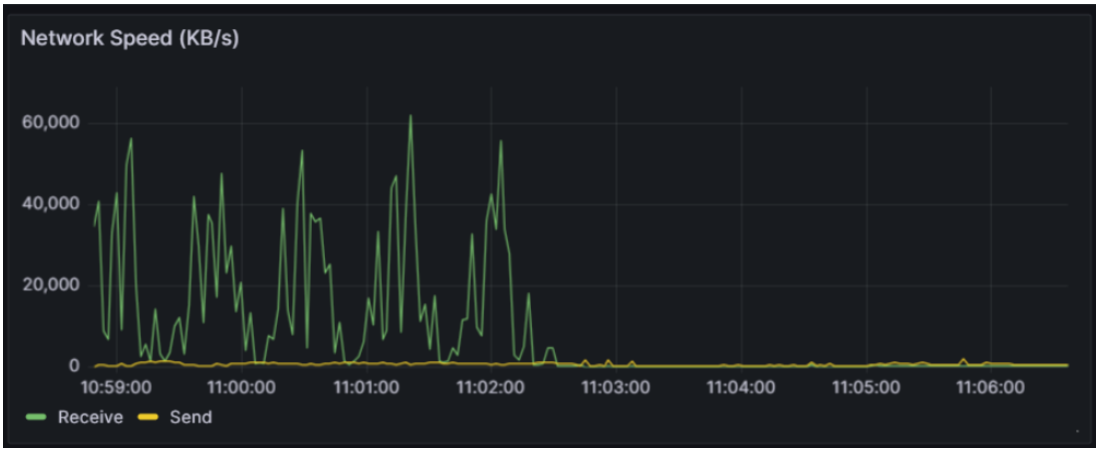


Network usage

MD Core #1



MD Core #2



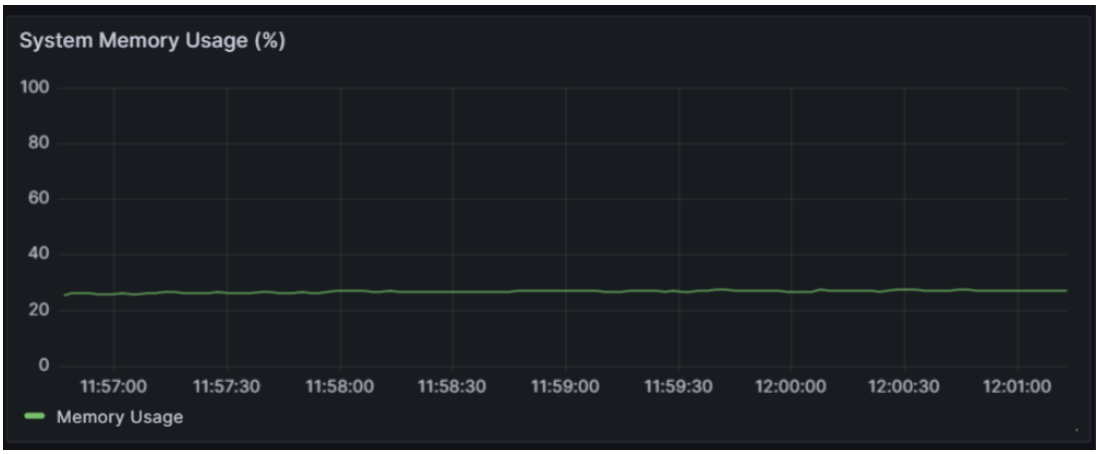
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

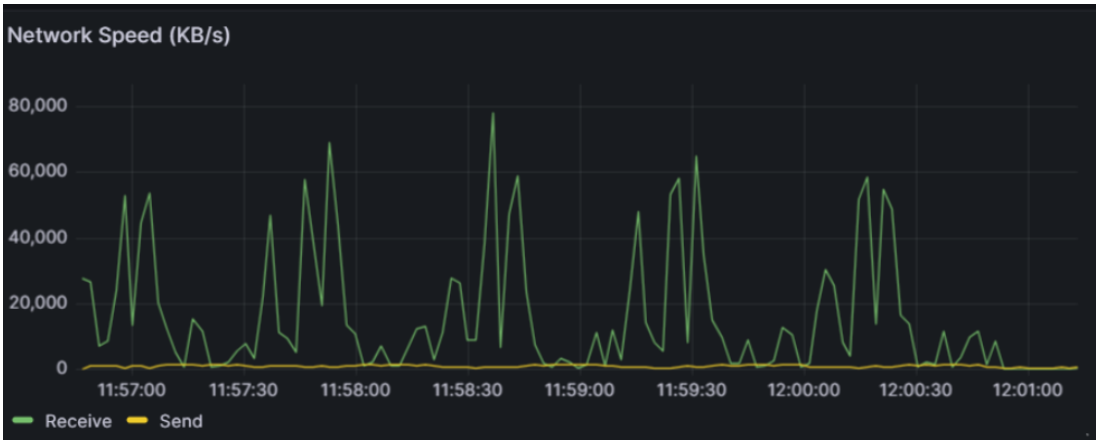


MD Core #2

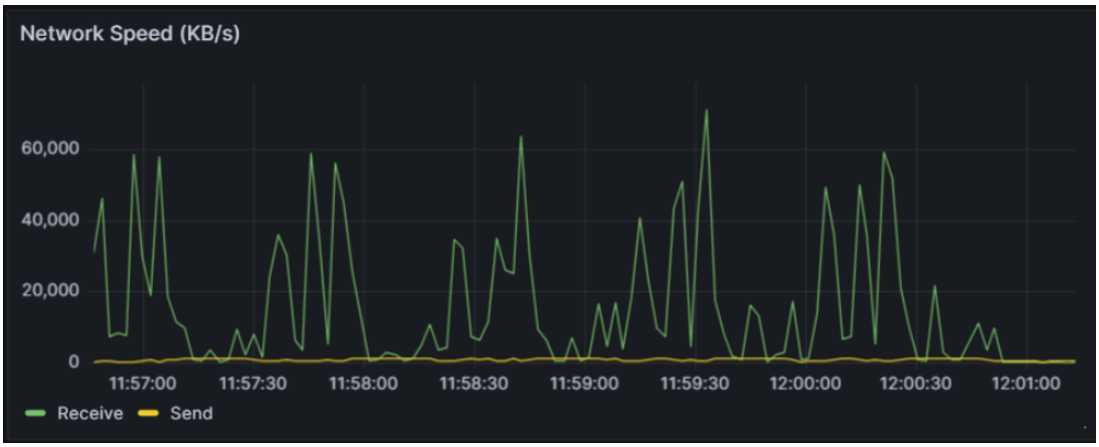


Network usage

MD Core #1



MD Core #2



Shared DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

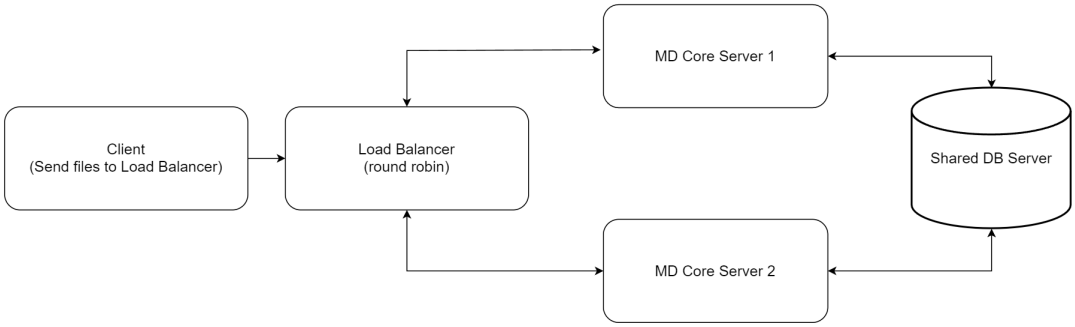
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV)	15.5	63.1	73.3	25.2	25.8	Receive: 4555.9 Send: 694.4	Receive: 4516.2 Send: 692.4	492,620.5	0.0
Deep CDR	9.70	50.7	51.2	31.9	32.5	Receive: 7062.9 Send: 623.2	Receive: 6653 Send: 654.1	786,189.6	0.0
Proactive DLP	7.2	35.2	35.0	27.9	27.5	Receive: 9047.4 Send: 802.4	Receive: 9859.8 Send: 787.5	1,055,110.8	0.0
Vulnerability	5.0	45.7	49.3	24.4	24.5	Receive: 13,585.7 Send: 966.7	Receive: 13,505.8 Send: 987.4	1,519,562.1	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV) + Deep CDR	19.9	79.1	77.6	34.0	34.2	Receive: 3785.9 Send: 573.8	Receive: 3663.1 Send: 577.7	382,716	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP	21.6	80.3	81.5	37.7	44.4	Receive: 3305.6 Send: 543.3	Receive: 3414.3 Send: 556.2	353,694.9	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	22.5	84.2	79.8	37.9	44.7	Receive: 3267.9 Send: 554.9	Receive: 3398.2 Send: 553.8	338,507.6	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

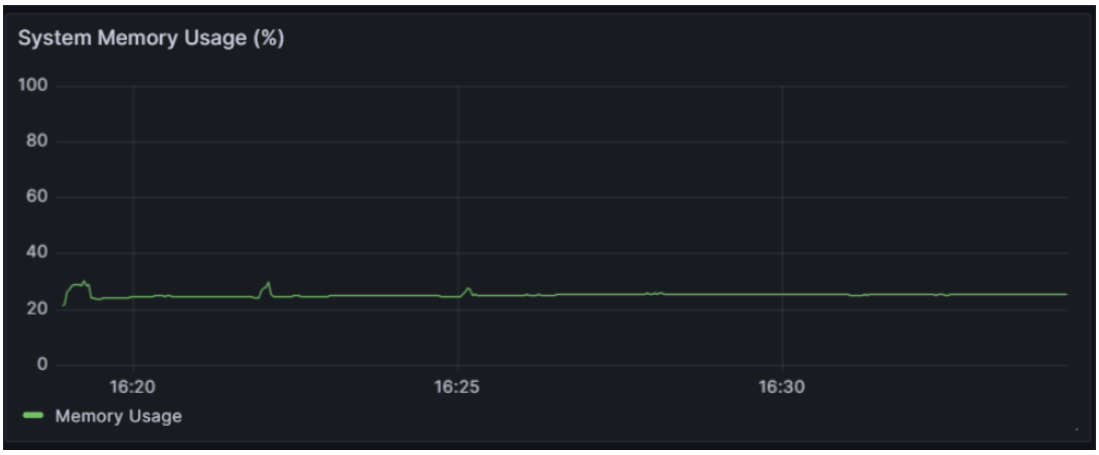
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

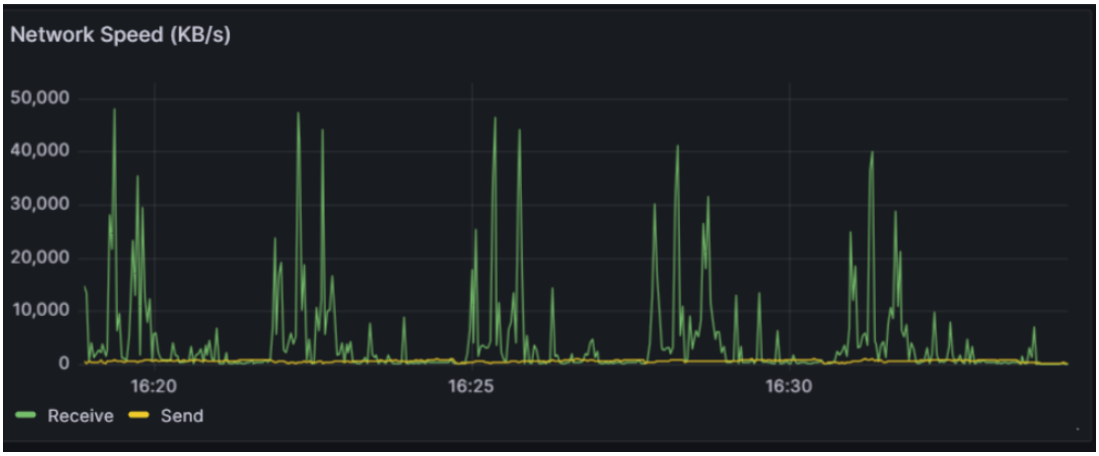


MD Core #2

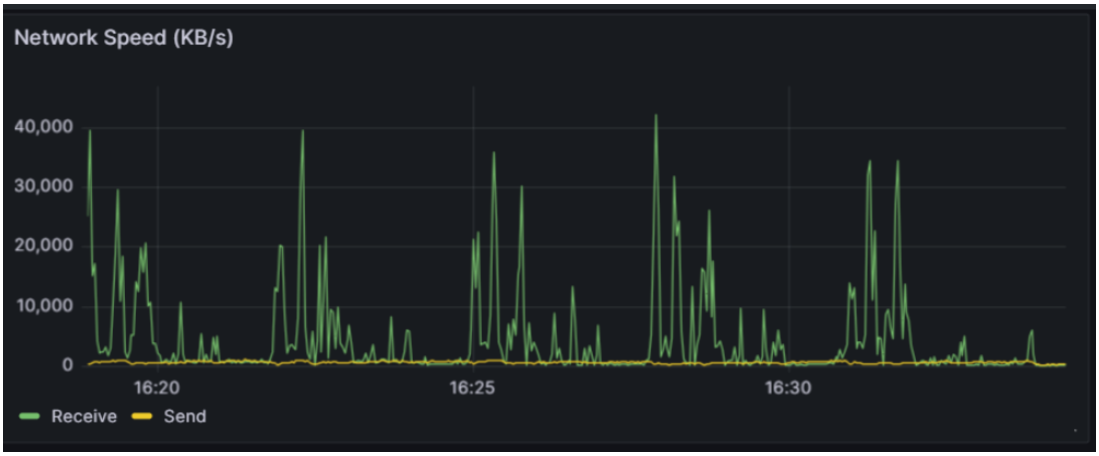


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

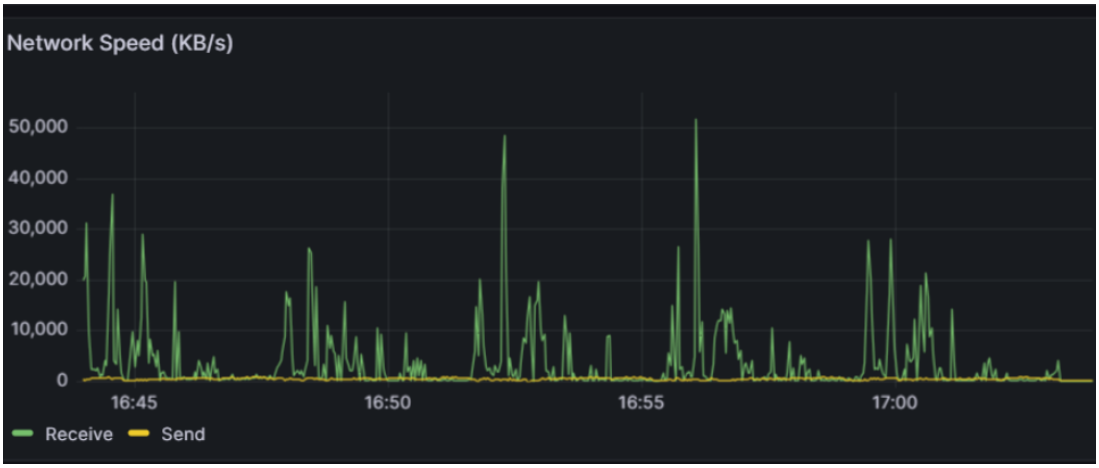


MD Core #2

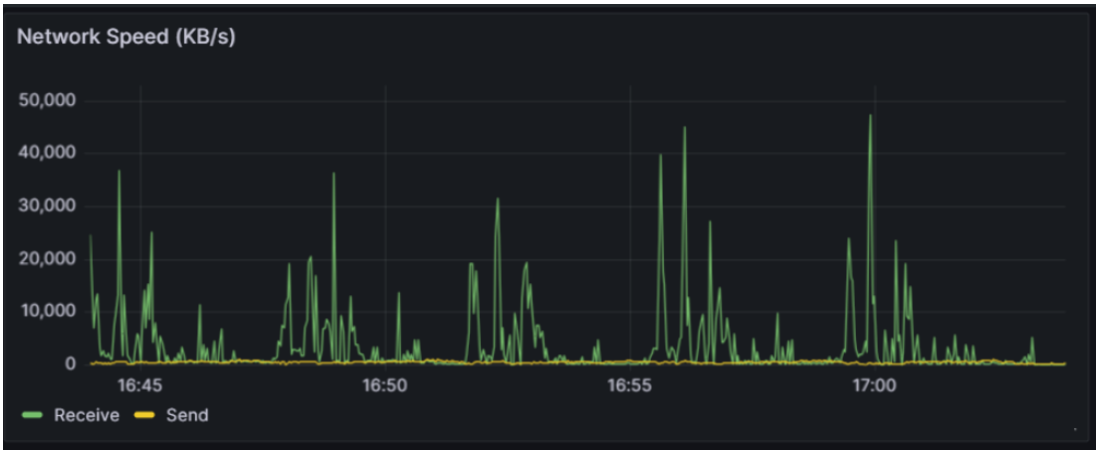


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1

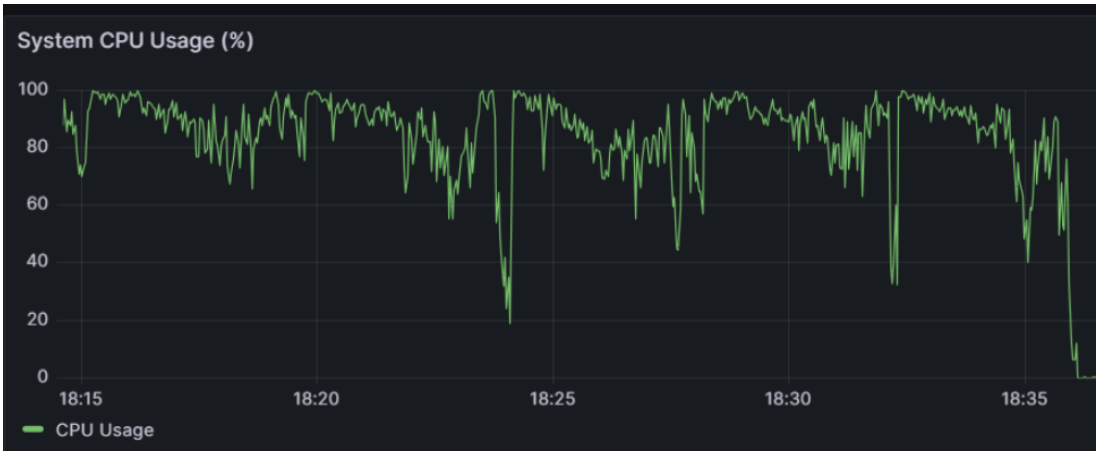


MD Core #2



CPU usage

MD Core #1

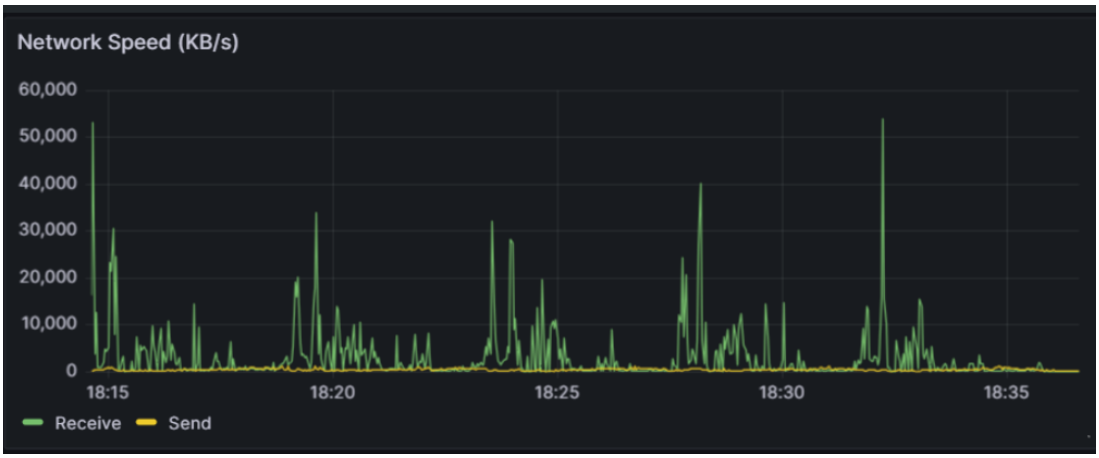


MD Core #2

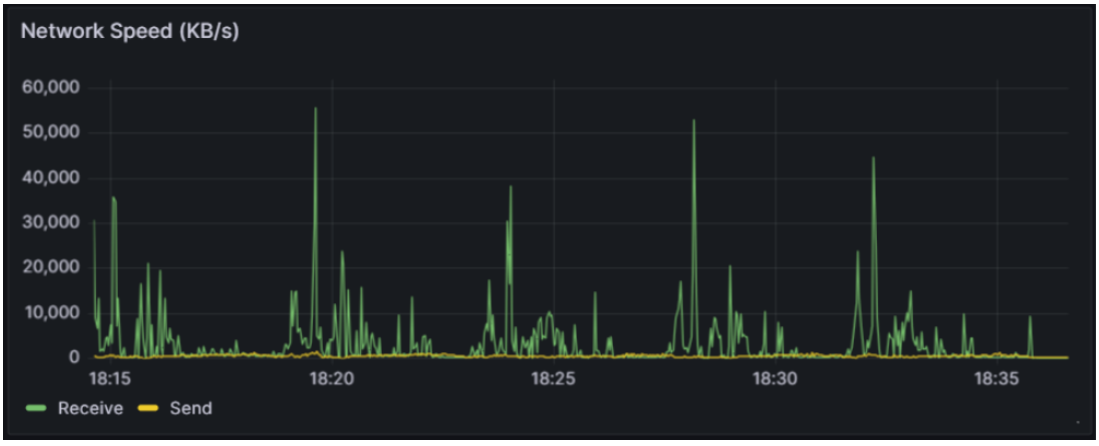


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

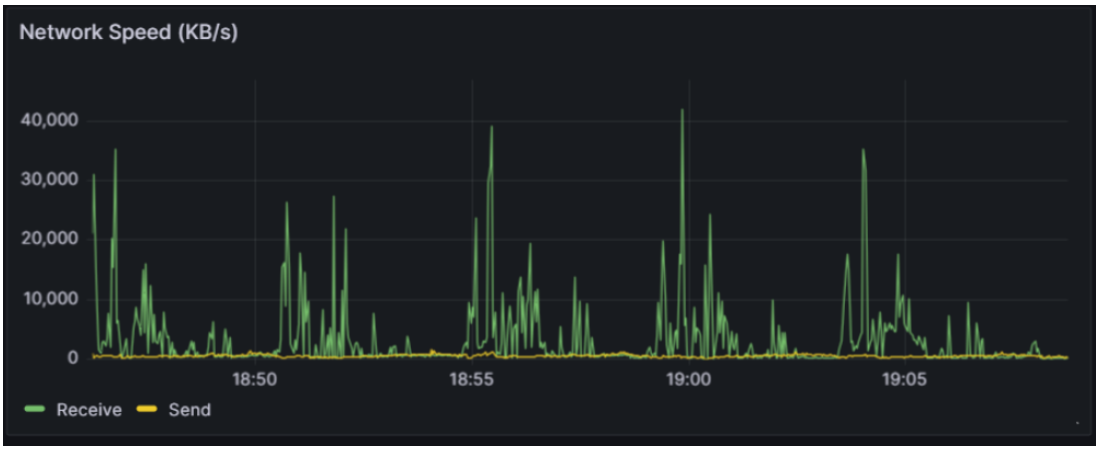


MD Core #2

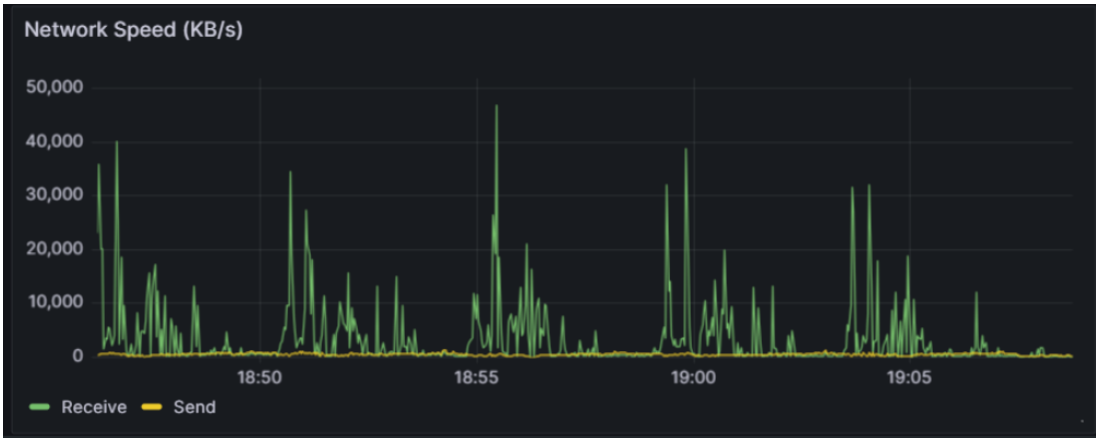


Network usage

MD Core #1



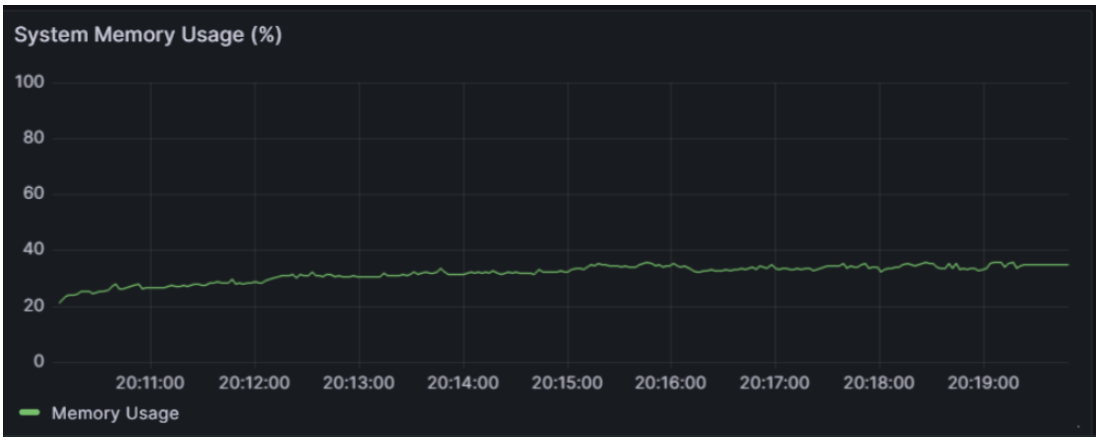
MD Core #2



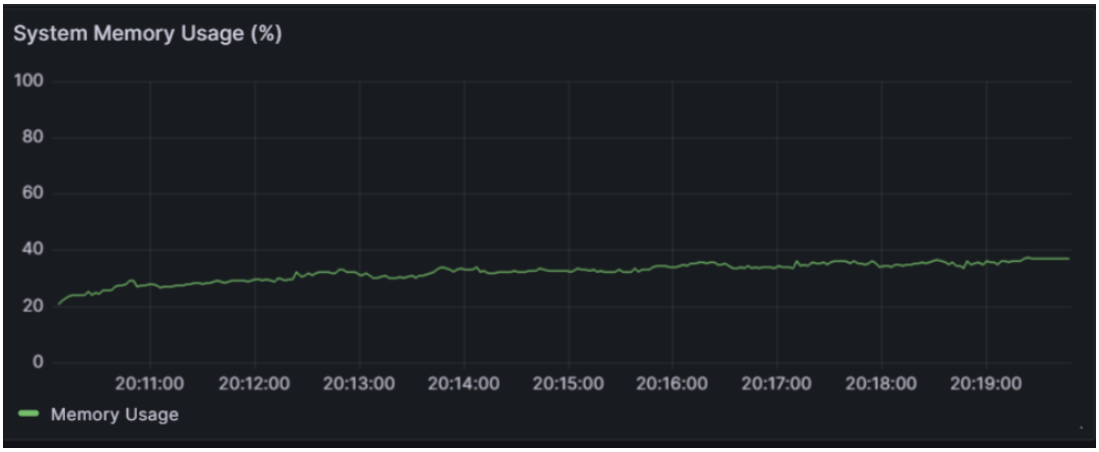
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1

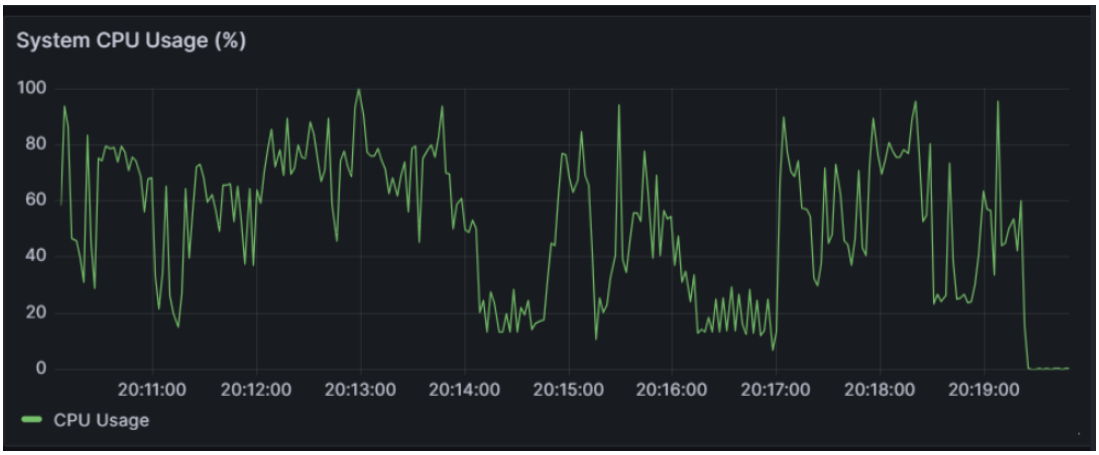


MD Core #2



CPU usage

MD Core #1

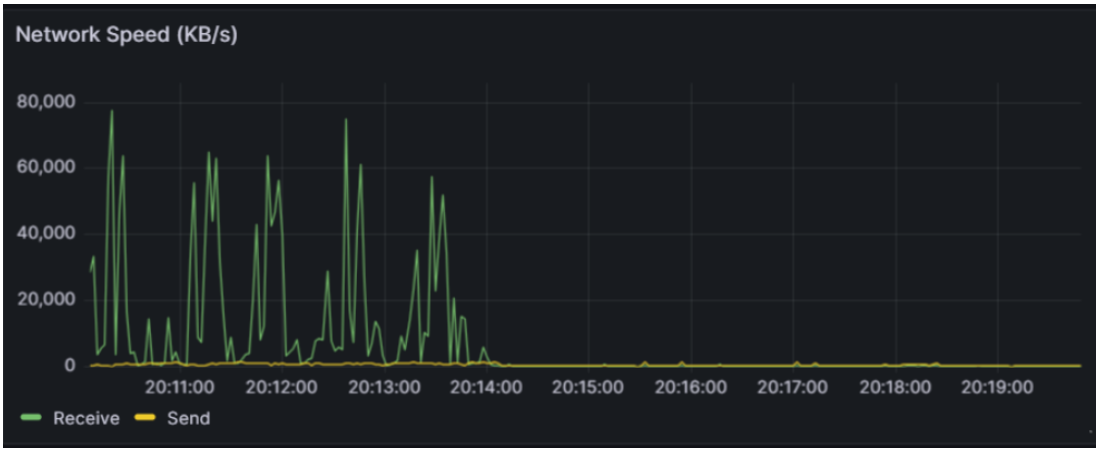


MD Core #2

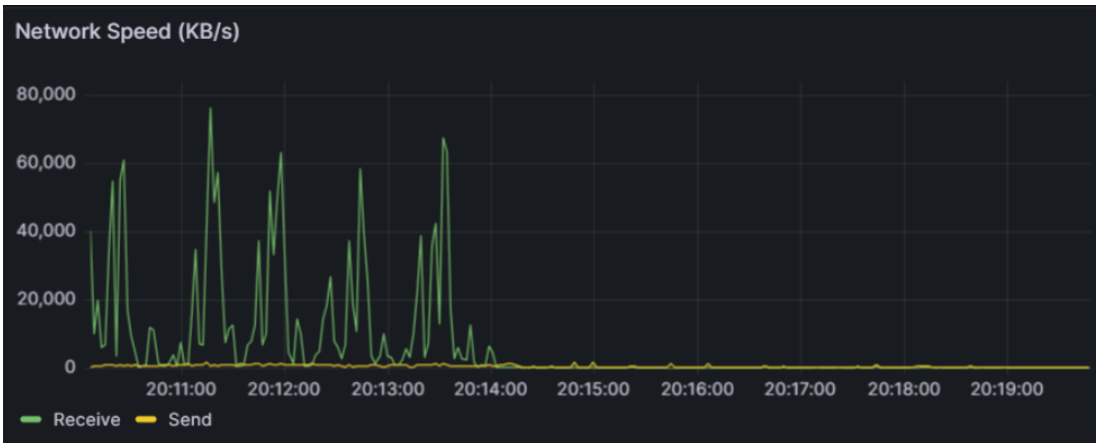


Network usage

MD Core #1



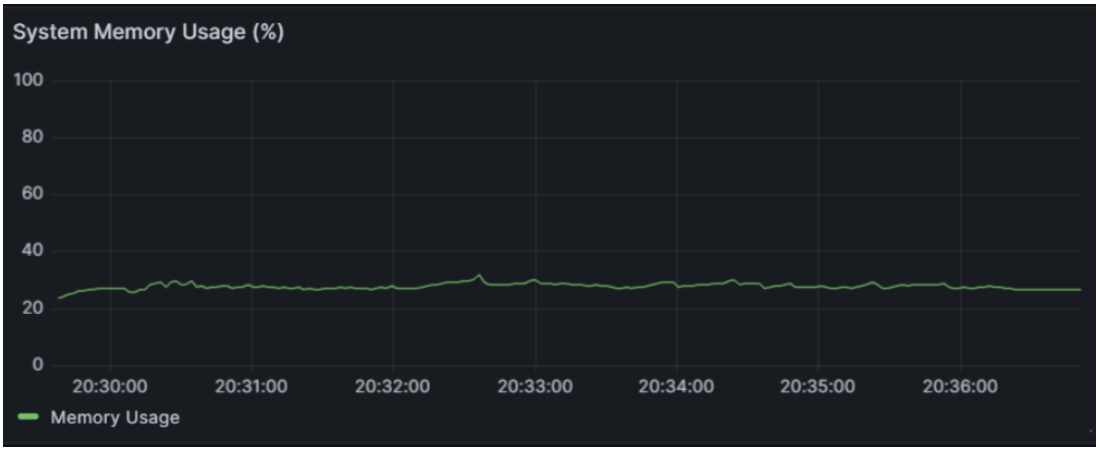
MD Core #2



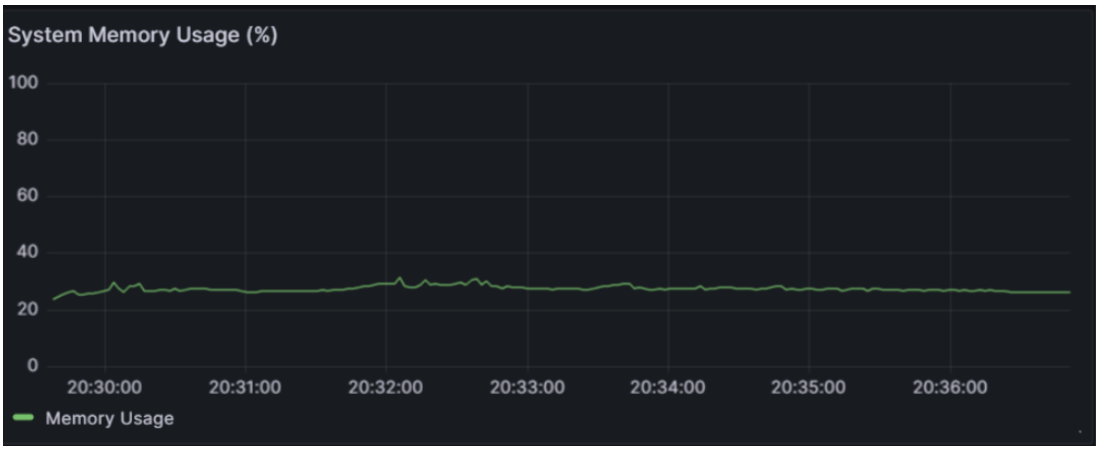
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1

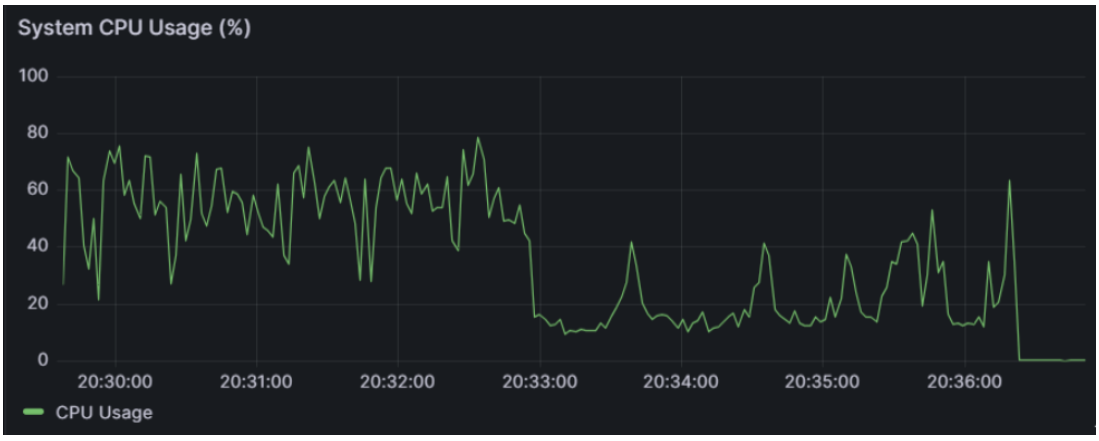


MD Core #2

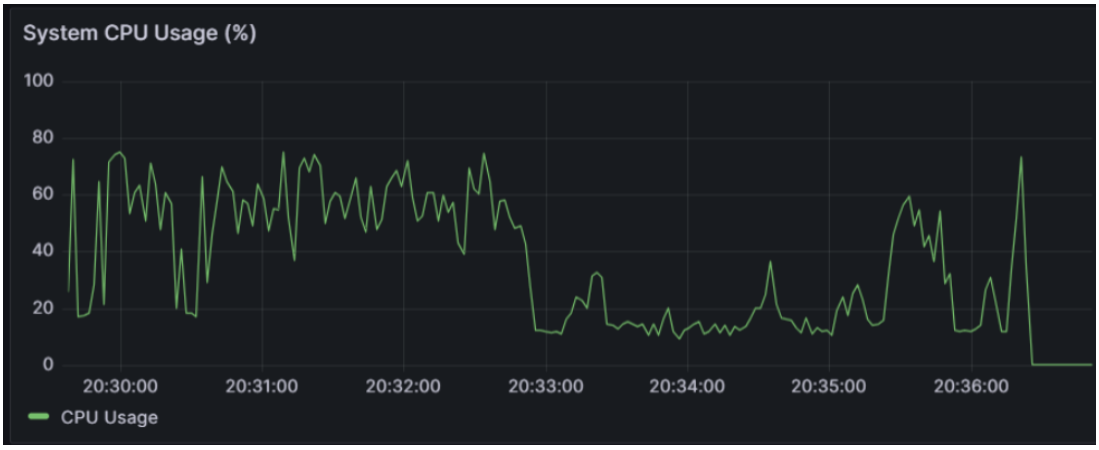


CPU usage

MD Core #1

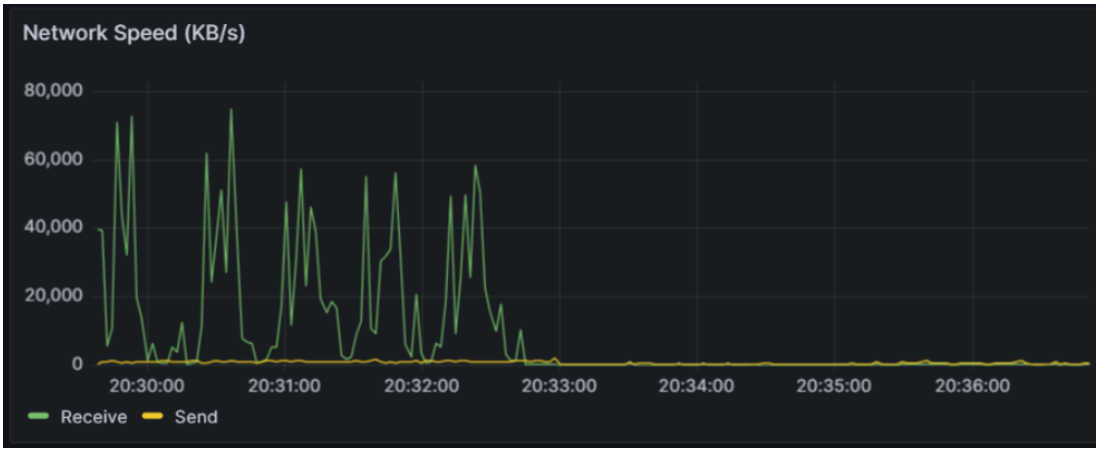


MD Core #2

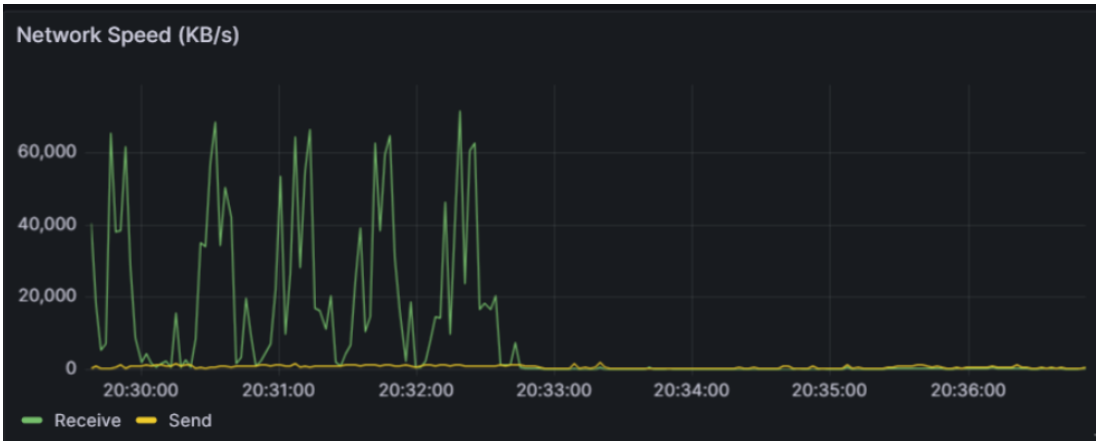


Network usage

MD Core #1



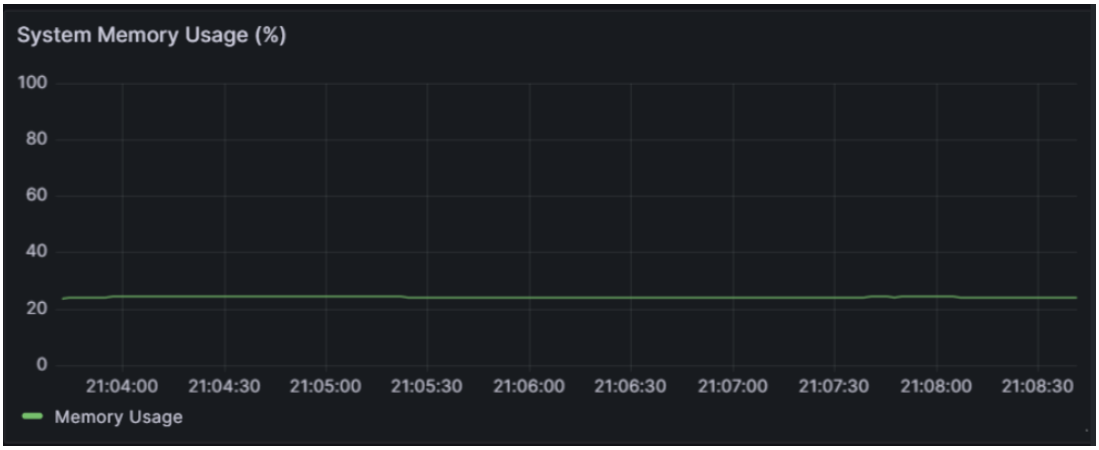
MD Core #2



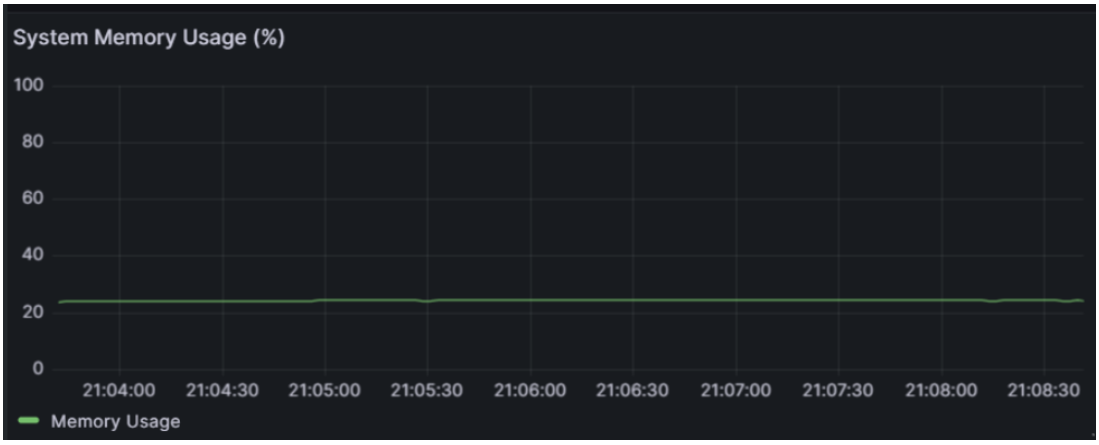
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

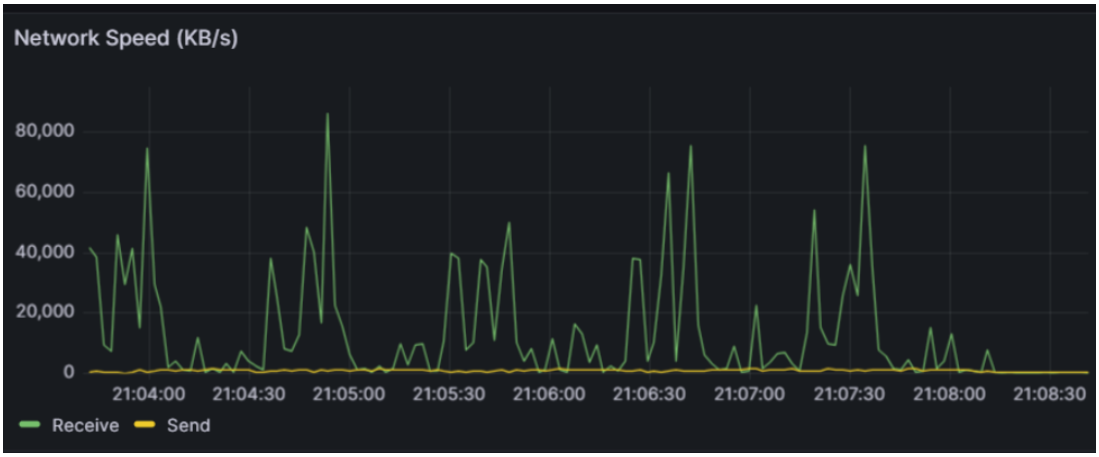


MD Core #2

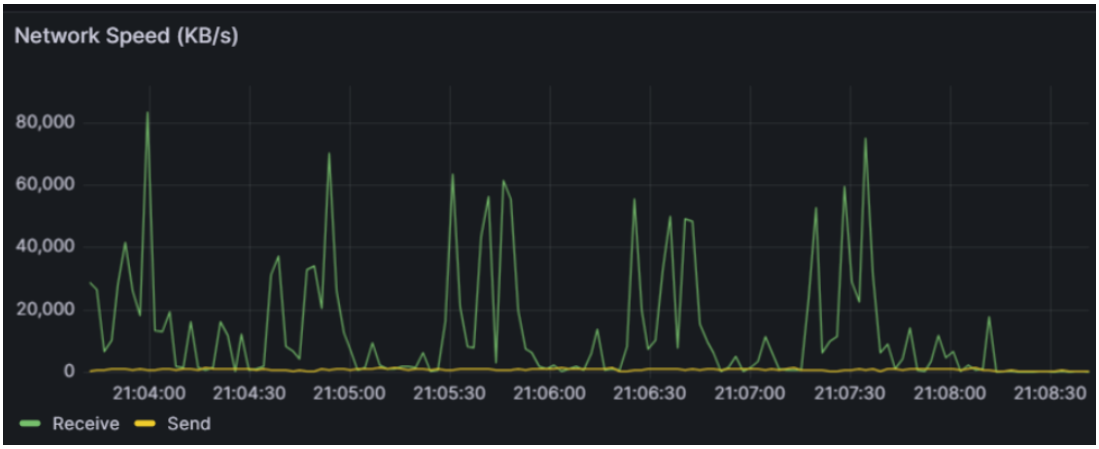


Network usage

MD Core #1



MD Core #2

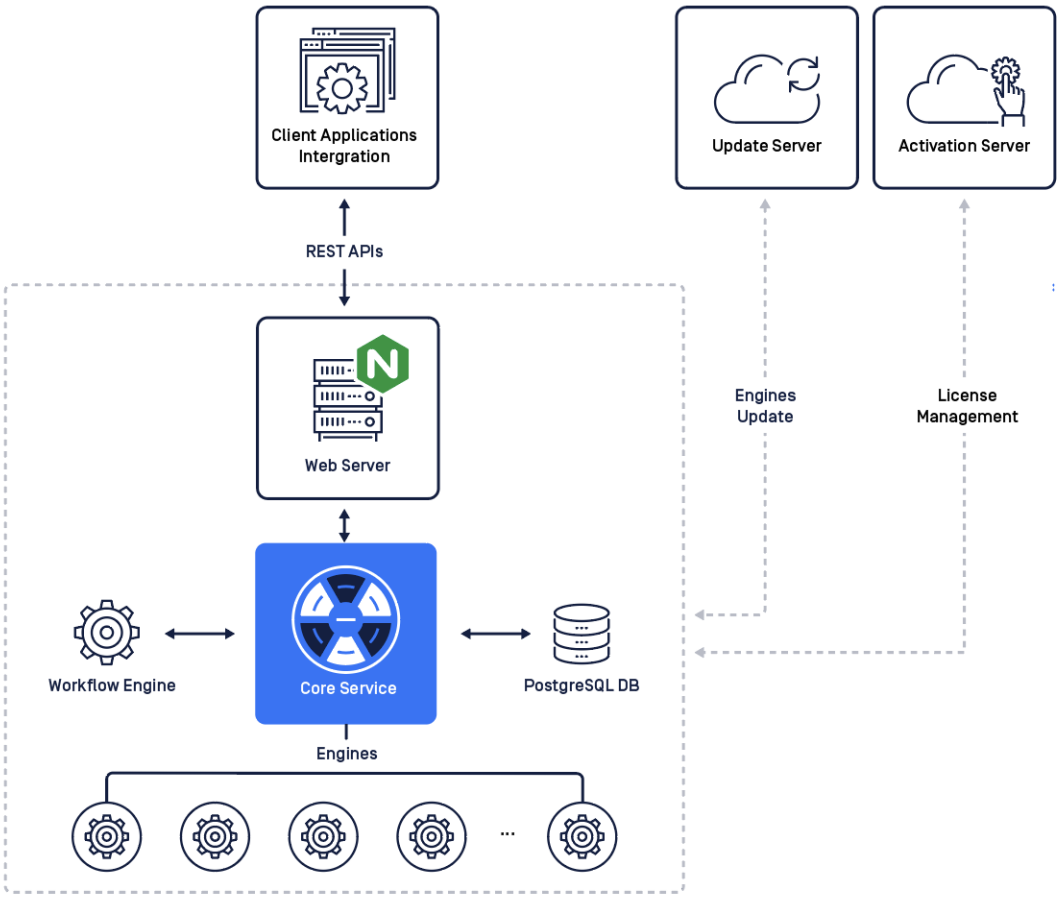

Network Speed (KB/s)
80,000
60,000
40,000
20,000
0
21:04:00
21:04:30
21:05:00
21:05:30
21:06:00
21:06:30
21:07:00
21:07:30
21:08:00
21:08:30
Receive
Send

MetaDefender Core Documentation

The users can consult this web page or, alternatively, they can download the manual in pdf format from the link below:

MetaDefender Core manual (SHA256:).

Recommended System Configuration



MetaDefender Core supports a wide range of deployment options, including air-gapped environments, providing the flexibility to choose the environment that best suits customers' requirements and infrastructure.

- Physical server.
- Virtualization platforms like VMware, Hyper-V and XenServer.
- Cloud Deployment.
- Container Deployment.

> **Note**
> Before installing MetaDefender Core, please refer to the **recommended minimum system configuration** listed below. The requirements are based on a standardized customer deployment and may require more or less resources to achieve the level of performance desired in your environment. Please note that the server specifications are built to allow a high volume daily processing. Please refer to Performance and Load Estimation for more details.

The recommendations below are for MetaDefender Core, API usage only. For any other use cases, please consult the user guide of the licensed products for accurate recommendations.

For certain use cases these might be adjusted and customized on specific needs and SLAs. We highly recommend to engage our ProServ team to assist in fine tuning MetaDefender and get the maximum performance out of your systems.

Microsoft Windows Deployments

[Windows] Supported Operating Systems

Only support 64-bit platforms.

- Windows 10, Windows 11 23H2, 11 IoT (22H2) only for End-User systems (e.g. Kiosk deployments)
  - Based on Microsoft License Terms, you're not allowed to use Windows Desktop as a server, therefore, MetaDefender Core can't be used on these systems for server use cases.
- Windows Server 2016, 2019, 2022

> **Disclaimer**
> End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

> **Plan for legacy OS support**
> OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:
>
> - Ubuntu 20.04 after **April 2025**.
> - Amazon Linux 2 after **June 2025**.
> - Windows 10 after **October 2025**.
>
> If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

[Windows] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 8	8	8 GB	16 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 12	16	16 GB	24 GB + 50 GB + 100 GB
MetaDefender Core 16	16	16 GB	32 GB + 50 GB + 100 GB
MetaDefender Core 20	32	16 GB	40 GB + 50 GB + 100 GB
MetaDefender Core MAX	32	32 GB	120 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core is installed.
- For a better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.
- SBOM processing requires more disk space, please allow at least 15GB free disk for the engine to work properly.

[Windows] Third Party Dependencies

- .NET framework 4.5 or above
- Microsoft Visual C++ Redistributable for Visual Studio 2015-2022 (version 14.38 or higher); x64 version.
- PostgreSQL: Only applicable when using a pre-installed PostgreSQL server running remotely.
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Some engines also have dependencies as described below:

Engine Name	Dependency
Deep CDR	Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 5.8.0 or above)
ESET	MetaDefender Core v4 temporary directory should have more than 200MB free disk space
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
Proactive DLP	• Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 2.0 or above) • .NET 6 dependencies • CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
Scrutiny	Microsoft Visual C++ 2015 - 2019 Redistributable x86 and x64
Systweak	.NET runtime 6.0, 7.0 or 8.0
Bkav Pro	Microsoft Visual C++ 2015 - 2022 Redistributable x64

[Windows] Installation Details

MetaDefender Core on Windows uses "C:\Program Files\OPSWAT" folder for storing resources or the installation directory.

MetaDefender will use its resources folder to store temp files as part of the analysis. **It's recommended to exclude this folder from real-time protection monitoring**. Knowledge base:

- How to set up exclusions on anti-malware software to prevent disruption
- Mitigating scan failure triggered by real-time protection service
- Can local AVs interrupt ongoing scans?

Linux Deployments

[Linux] Supported Operating Systems

Only support 64-bit platforms.

- Red Hat Enterprise Linux 8, 9
- Rocky Linux 9
- Debian 11, 12
- Ubuntu 20.04, 22.04
- Amazon Linux 2 is supported via Amazon Machine Image (AMI)

Plan for legacy OS support

OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:

- Ubuntu 20.04 after **April 2025**.
- Amazon Linux 2 after **June 2025**.
- Windows 10 after **October 2025**.

If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

For Hardended Linux OS

Some AV engines could be failed to initialize and run on MetaDefender Core (e.g. ESET, Kaspersky) due to execution permission required by them on `/var/tmp/ometascan` folder.

Please follow steps at Why have the ESET and Kaspersky scan engines failed to initialize on the hardened Linux OS? for remediation details.

Disclaimer

End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

[Linux] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 5	4	4 GB	10 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 10	8	8 GB	20 GB + 50 GB + 100 GB
MetaDefender Core MAX	16	16 GB	40 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core resides on.
- For better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.

[Linux] Third Party Dependencies

Dependencies list:

- openssl
- grep
- lib32stdc++6 (>= 4.5)
- libc6-i386 (>= 2.10)
- procps
- zlib1g
- libcurl3 (>= 7.19.7)
- libcurl4
- postgresql:
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Not all the above dependencies will need to be installed, it is dependent on different Unix distro and version

Some engines also have dependencies as described below:

Engine name	Dependency
Archive engine	• To extract password protected MS Office files, you need to install .NET 6 dependencies, libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • Rocky Linux 9.x/Red Hat 9.x: libnsl
Deep CDR	• Rocky Linux 9.x: libgomp, ncurses-compat-libs, libnsl • CentOS: libgomp, ncurses-compat-libs • Red Hat 8.x: libgomp, ncurses-compat-libs, libnsl • Red Hat 9.x: libgomp, ncurses-compat-libs, libnsl, compat-openssl11 • Ubuntu/Debian: libgomp1, libncurses5 • libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • .NET 6 dependencies
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
FileType engine	Rocky Linux 9.x/Red Hat 9.x: libnsl
Proactive DLP	libgdiplus, mscorefonts .NET 6 dependencies CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
RocketCyber	libgomp

Note: Some dependencies may need to be installed from external repositories, not from the OS default repositories.

[Linux] Installation Details

MetaDefender Core default installation path is using /var folder for storing resources:

- `/var/lib/ometascan` : installation folder with all its resources (database, DLP processed / CDR sanitized / quarantined file storage, engine package and definition updates).
- `/var/log/ometascan` : application logs.
- `/var/run/ometascan` : runtime folder for Core related processes.
- `/var/lib/ometascan` : all successful deployed engines data files.
- `/usr/lib/ometascan` : all shared libraries required by MetaDefender Core, bundled PostgreSQL server related binaries and libraries.
- `/var/tmp/ometascan/resources` : all temporary processing files for MetaDefender Core to process.

Storage Usage Information

Based on the configuration, MetaDefender Core will need additional disk space to store analysis/result data, quarantined files and temporary files:

	Data Storage
Analysis Reports (Processing History)	Approximately 4.5GB is required for every 1M analysis reports stored. • 100 GB is recommended as a baseline. Database storage requirements are affected by Data Retention settings, engine settings and scanning load.
Quarantined Files	Depends on the customers' dataset and configuration
Sanitized Files	Depends on the customers' dataset and configuration
Temporary processing files	Depends on the customers' dataset and configuration • A 1GB non-archive file requires 1GB free disk space at least. • A 1GB archive file requires (1 * 2)GB free disk space at least. • 50GB is a baseline and should be adjusted based on observation of the solution within the environment, with consideration to peak times and overall file makeup (e.g. complexity, size, archives).
Note	Options are available to control how large and/or complex files are handled to limit the amount of system resources required. See Workflow Configuration Template for more information.

Custom Engines

The recommendations above are specific for MetaDefender pre-packaged bundles.

However for additional Custom Engines, please review the Knowledge base to review additional requirements (if any) for the selected engine.

Browser Requirements for the MetaDefender Core Management Console

One of the following browsers is suggested to view the MetaDefender Core Management Console:

- Microsoft Edge
- Chrome
- Firefox
- Safari

Chrome, Firefox, Safari and Microsoft Edge browsers are tested with the latest available version at the time of release.

Network Bandwidth

In online environment where MetaDefender Core is configured to download engine update packages directly from OPSWAT update server source, then using gigabit network speed (125 MB/sec) is highly recommended to avoid potential network related issue while downloading files, or at least 100 Mbps (12.5 MB/sec) network speed.

Allowlisting URLs

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts':

- https://activation.dl.opswat.com - for license online activation
- https://update.dl.opswat.com - for engines and databases download

Note: *IP address-based allowlisting on your firewall might fail after some time since OPSWAT uses CDN (AWS Cloudfront) to faster delivery updates over the world, and IP address of edge servers might change over time.*

Depending on your location, connections with OPSWAT cloud-based servers be forwarded to specific AWS Cloudfront's edge locations, each is assigned with its own unique IP address ranges. You might also need to allow correct AWS Cloudfront's corresponding IP address ranges by referring to https://ip-ranges.amazonaws.com/ip-ranges.json

Warning

Even the OPSWAT update servers host updates for all of the available engines we support, sometimes custom engines might try to connect to their own cloud for updates, but this can be disabled in firewall and they will be updated just from OPSWAT cloud servers instead.

Differences Between MetaDefender Core v4 and v5

We are very excited to inform you about the release of version 5.0.0 of our MetaDefender Core!

The new version is designed to enhance the product performance, user experience, as well as to provide more flexibility, security, and scalability for our customers.

At the same time, we are also committed to keeping backward compatibility for whom running MetaDefender Core v4, and avoid the need for updating your client integration to move forward with MetaDefender Core v5.

v4 REST API Integration Supported
All public REST API endpoints available on MetaDefender Core v4 will be continously supported on v5.

v4 Product Major Features Supported
All product major features available on MetaDefender Core 4.21.2 (the latest one on v4) will be continously supported on v5.

Seamless Product Upgrade
Upgrading from MetaDefender Core v4 to v5 will be supported and covered with data migration without additional manual steps from users.

Licensing information will be also migrated.

Below are some highlights of the first v5 release, MetaDefender Core 5.0.0:

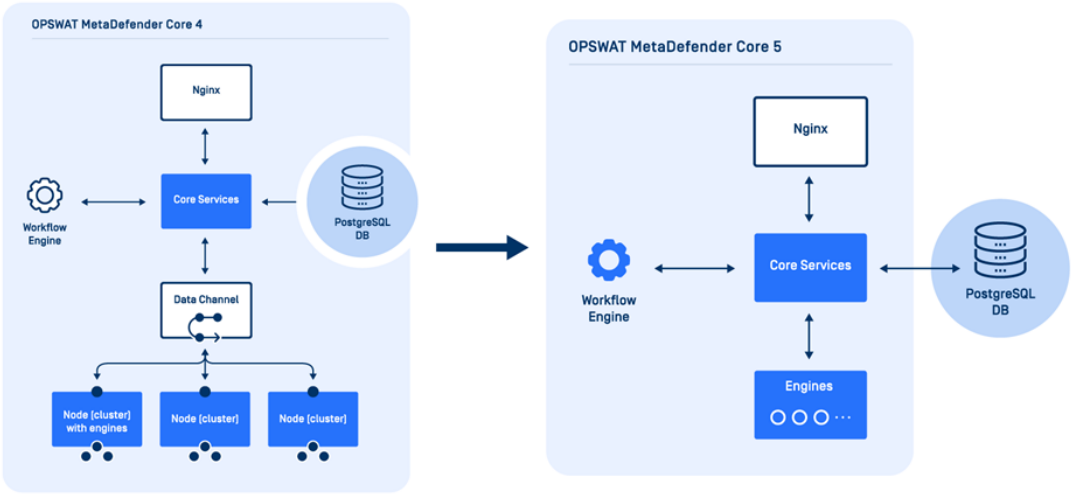
New product architecture

This version introduces a simpler product architecture that speeds up integration and automation, file processing flow, and immensely enhance the product performance and efficiency.

Earlier versions utilized MetaDefender Node concept, designed to support scaling of the scanning infrastructure by distributing scan requests among several different processing Nodes. Product licensing was based on number of Node instances. The benefit of having such a distributed infrastructure is that based on Node loads, Metadefender Core server can always choose the most appropriate Node to assign a new scan task to. However, having nodes in the v4 architecture caused more complexity and possible latency in the processing flow and entailed maintenance cost.

In this release, we retired multi-Node deployment and restructured MetaDefender Core's backend. This alteration significantly improves file processing speed and product efficiency, as well as eliminates extra cost for maintaining node services. Users now can simply install additional MetaDefender Core instances for scalability requirement,

as well as configure their own load balancer to coordinate tasks among multiple MetaDefender Core instances. Accordingly, product licensing is now based on number of the product instances.



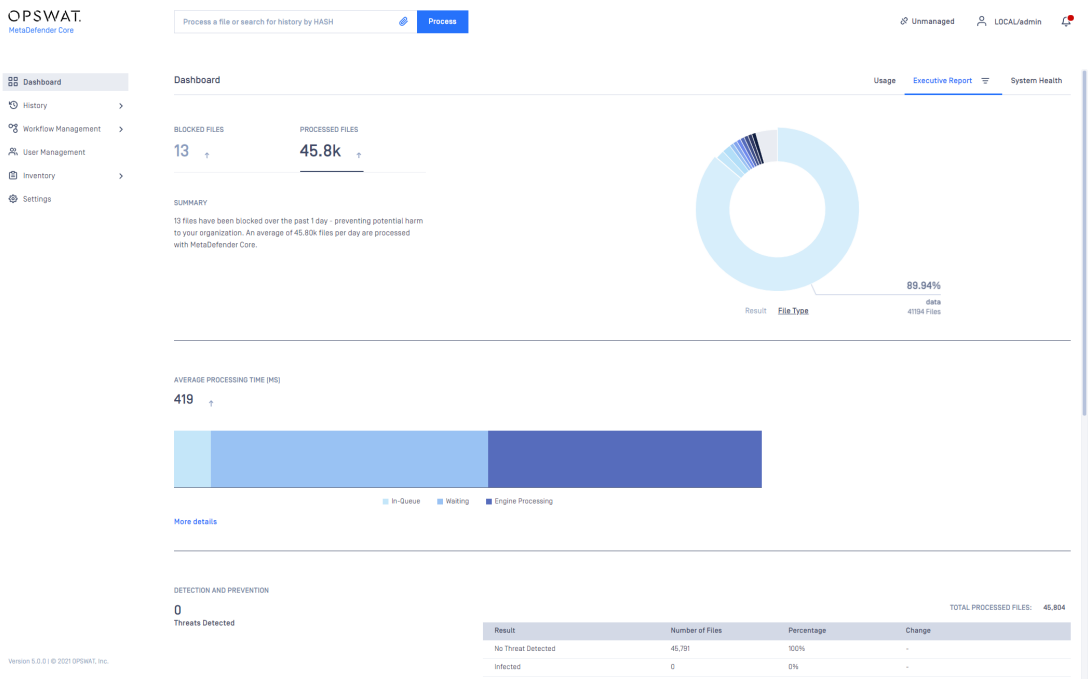
Please be informed that

- All existing administrative APIs related to "Node" on v4 will be kept as it is on v5 to ensure backward compatibility with existing client integrations.
- Applicable settings defined in Windows registry (Windows) or ometascan-node.conf config file (Linux) for Node will be migrated to Core. Details: MetaDefender Configuration

Brand-new product user interface

The second notable update in this version is a brand-new UI for the management console, which was designed with the focus on the quality and thoughtfulness of the user experience. The new administrative console UI provides an advanced set of Core features and observable processing status.

Comprehensive executive dashboard reporting offers dynamic, real-time and actionable threat insights. It also provides visibility of your security posture allowing you to paint a picture of security that aligns with the overall business. The report includes easy-to-read charts, graphs, and statistic explanation that help your security professionals reduce clutter, gather information quicker, spot threat activity early and act swiftly.



We're continuing to improve our product UI to provide users an excellent experience with the application and overall usability. Please expect more powerful UI features coming in next releases.

New compression archive engine for archive sanitization

In previous version, MetaDefender Core's archive engine implemented four functions including decryption, extraction, encryption and compression that with a high volume of files it takes time to process them all. For the purpose of enhancing archive handling performance, this release comes with a separated compression archive engine, which dedicatedly processes archive sanitization (compression and encryption). It is also created to facilitate maintenance and further feature development.

Archive sanitization feature now requires Deep CDR license

As to respect sanitization practice and ensure unknown threats prevention as designed, now starting on version 5, it's mandatory to have Deep CDR licensed for archive sanitization feature enablement on MetaDefender Core workflow configurations.

Archive handing related engines changed

The legacy "Archive" engine on MetaDefender Core v4 will be changed its name to "Archive Extraction" engine, even when you are still on MetaDefender Core v4.

A brand new "compression archive engine" is to dedicatedly handle archive sanitization in v5 (archive compression), that helps relevant archive handling performance improvement, easier for maintainance and further feature development.

Notes when you are still on MetaDefender Core v4 (not upgraded yet)

The legacy "Archive" engine (now name changed to "Archive Extraction" engine) on MetaDefender Core v4.x will still cover both extraction and compression archive tasks as usual.

Even though your MetaDefender Core v4.x might now have a new "Archive Compression" engine auto downloaded, but that engine cannot be deployed (marked as "permanently failed" in relevant engine status check API). This behavior is expected, and not causing any impact to MetaDefender Core processing flow, or its performance though.

- Just in case your client integration has a check for engine status before triggering scan request, then you might want to disable "Archive Compression" engine status on MetaDefender Core v4 using REST API. Detailed instruction: Archive Compression Engine Permanently Failed on MetaDefender Core v4
- When you later upgrade to MetaDefender Core v5 from v4, then you might need to manually re-enable "Archive Compression" engine to make it active and running on MetaDefender Core v5.

Notes for licensing

For offline deployment (MetaDefender Core is not connected to the Internet directly for online update), you will need to re-activate again on OPSWAT portal to retrieve latest engine list in .yml file including "Archive Compression" engine if you are currently licensed with Deep CDR.

Meanwhile, for online deployment, this manual procedure is not required since MetaDefender Core will cover that itself.

Import & export configurations

Not only workflow settings like in previous versions, but also all other general settings (update, security, data retention, server, hashes to skip) and most importantly, user settings, will be able to export and import on MetaDefender Core version 5.

Learn more at Import/Export configuration

Backward compatibility supported

The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

Windows application event logging disabled by default

Applicable only to Windows OS based MetaDefender Core.

Help avoid duplicated logging with separate file option, and improve overall processing performance. Of course, when being still valid to you, you are still supported to enable it back.

CSRF token header renamed

CSRF token originally created for security strengthening and being used by MetaDefender Core management console UI since version 4.21.0 via header `oms_csrf_token`

To get it compliant with some load balancers which do not accept underscore in header name by default, and save you from additional load balancer configuration step, we have now changed the header name to `oms-csrf-token`

This change does not impact to any REST API based client integration.

More comprehensive and accurate error messages on REST APIs

Impacting on following REST API endpoints:

- `PUT /admin/config/session`
- `PUT /admin/config/update`
- `PUT /admin/config/scan`
- `PUT /admin/config/skip`
- `PUT /admin/config/auditlog`
- `PUT /admin/config/webhook`
- `PUT /admin/config/rule`

Upload time and queue time logic updated in JSON scan response

Impacting on following REST API endpoints:

- `GET /file/{data_id}`
- `GET /hash/{md5/sha1/sha256}`
- `GET /archive/{data_id}`
- `POST /file/{async}`

Added new field:

- `receive_data_timestamp`: The timestamp when upload progress started (first byte received) (in milliseconds)
- `upload_time`: Total time elapsed for upload process (in milliseconds)

Modified existing field (new logic):

> **Backward compatibility supported**
> The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

- `upload_timestamp`: The timestamp when upload progress finished (all bytes received) (in milliseconds)
- `queue_time`: Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue time = start_time - upload_ timestamp) (in milliseconds)

Release version 5.0.0 also comes with variety of outstanding features and improvements.

For detailed information about our latest release please view our Release notes

Installation

Starting with MetaDefender Core 4.20.0, the centralized/shared database deployment model is supported (all MetaDefender Core instances share the same database), along with traditional standalone database mode (every MetaDefender Core instance has its own database to manage).

- Standalone mode
- Shared Database Mode

Installation

1. Download the package of your choice from My OPSWAT
2. Install the package on your computer via the Command Line or via the Wizard
3. Open a web browser and point to http://<server name or IP>:<port>
    - The default port is **8008**
    - In case of problem check Inaccessible Management Console page
4. Complete the required steps of the basic configuration wizard
5. You must activate this deployment to use its features

Installation notes

- If the MetaDefender Core package **dependencies are not installed** on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.
- During installation the databases might need to be upgraded. This could take noticeable time depending on database size (eg.: length of scan history).

Standalone mode

Wizard Installation

The Install Wizard is only for the Windows installer (.msi file).

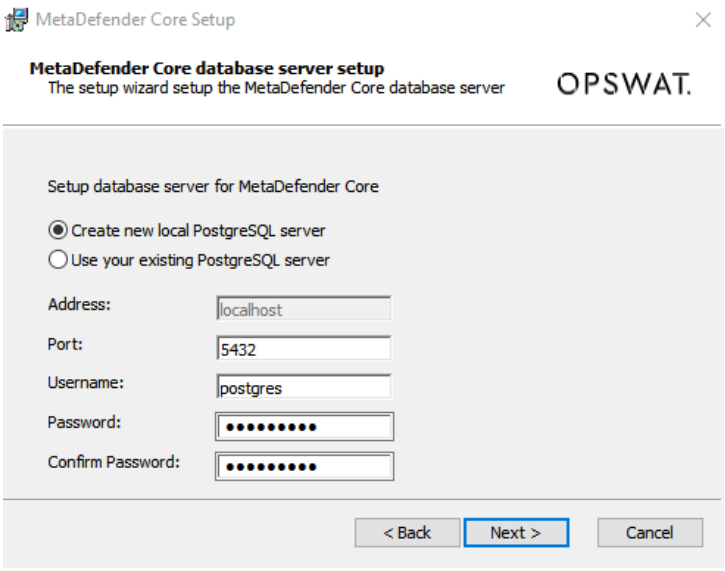
Since MetaDefender Core 4.19.0, we have switched to using PostgreSQL DBMS which needs a dedicated database server. MetaDefender Core supports two modes to setup PostgreSQL server:

Local (Bundle) PostgreSQL server

MetaDefender Core will install a new PostgreSQL server locally in the same box with the product.

> **PostgreSQL user credentials**
> Unicode characters are NOT supported for "Username" and "Password"
>
> "Password" does not accept # (hash) as starting character (e.g. #ABC)

> **The ignition file**
> If the ignition file exist in the designated location, then MetaDefender Core will use the details defined in the ignition file, rather than what is passed via the installation wizard.
>
> More info about the ignition file based installation: Command Line Installation

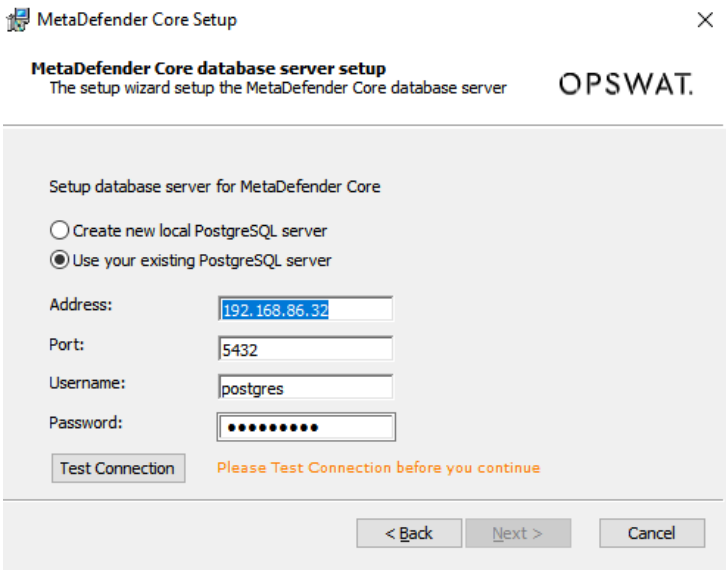


Remote (Existing) PostgreSQL server

MetaDefender Core will connect to remote pre-installed PostgreSQL running remotely.

> **PostgreSQL connection configuration**
> "Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

> **PostgreSQL user privilege requirements**
> MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.
>
> Details: PostgreSQL User Privilege Requirements



Command Line Installation

Ignition File

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Make sure to create the ignition file, if it doesn't exists before you go ahead and install MetaDefender Core via command line.

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

MetaDefender Core supports two modes to setup PostgreSQL server:

1) Local PostgreSQL server

- MetaDefender Core will install a new PostgreSQL server locally, next to the product.
  - A sample iginition file for PostgreSQL server information (***Only non-Unicode characters supported for "user" and "password"__***):

ometascan.conf bash

```
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
```

2) Remote PostgreSQL server

> **Info**
>
> **PostgreSQL version 12 (at least 12.3) or version 14 is required.**
>
> According to PostgreSQL notice, version 12 will stop receiving fixes on November 14, 2024, so we suggest that you use version 14.

MetaDefender Core will connect to a pre-installed PostgreSQL server running remotely and will setup its database.

Make sure to follow steps at Open Connection On PostgreSQL Server first, so that your remote existing PostgreSQL server to accept upcoming connection from MetaDefender Core.

A sample ignition file for PostgreSQL server information:

ometascan.conf bash

```
[dbserver]
type=remote
host=192.168.86.32
port=5432
user=existingserver_admin_user
password=existing_server_pass
```

After installing MetaDefender Core successfully, you may want to remove all credentials info created for [dbserver] section in ignition file for security reason.

> **Warning**
> If the MetaDefender Core package dependencies are not installed on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.

Debian package (.deb)

bash

```
sudo dpkg -i <filename> || sudo apt-get install -f
```

On Red Hat Enterprise 7.x Linux / CentOS package (.rpm)

bash

```
sudo yum install <filename>
```

For systems which enabled GPD check flag:

bash

```
sudo yum install --nogpgcheck <filename>
```

Red Hat Enterprise Linux 8.x package (.rpm)

Install the latest PostgreSQL repository RPM:

> **Note**
> Due to the shorter support cycle on Red Hat Enterprise Linux, all supported versions of PostgreSQL are not available on this platform. The PostgreSQL Yum Repository will integrate with the normal systems and patch management, and provide automatic updates for all supported versions of PostgreSQL throughout the supported lifetime of PostgreSQL. Reference: https://www.postgresql.org/download/linux/redhat/

bash

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

In case installing a remote PostgreSQL on either the same MetaDefender Core server or another server, you need to install the following additional package. PostgreSQL version 12.3 or newer is recommended.

bash

```
sudo dnf install -y postgresql12-contrib
```

Install MetaDefender Core:

bash

```
sudo dnf install -y <filename>
```

Windows package (.msi)

On Windows systems it is possible to install the product by running the corresponding .msi file.

From command line interface it is also possible to install the product by executing

bash

```
msiexec /qn /i <msi file name> <option key>=<option value>
```

where the possible keys and their default values are the following:

Key	Default Value	Description
INSTALLFOLDER	\Program Files\OPSWAT\MetaDefender Core	Customize installation folder for product Example: `INSTALLFOLDER="D:\Products"`
RESTADDRESS	*	REST interface binding IPv4 or IPv6 address ('*' means that service listens on all IPv4 and IPv6 interfaces)
RESTPORT	8008	REST interface binding port
REPORT_ENGINE_ISSUE	true	Enable reporting of engine issue count. (possible values: "true" or "false").

Note

Reporting of engine issue count

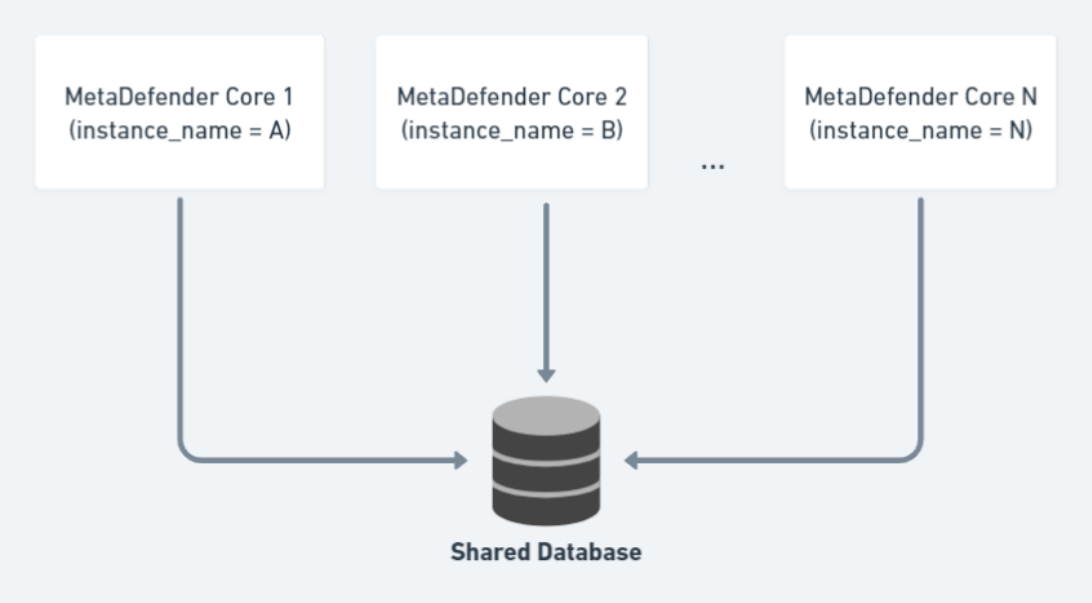
If reporting of engine issue count is enabled, MetaDefender Core server will send only the **number** of initialization errors and **number** of unexpected stops for the specific db/engine version. This information is sent over a HTTPS channel when the product downloads the latest package descriptors. This information is used for early detection of any specific 3rd party engine quality issues.

For details on using msiexec please consult Windows installer documentation.

Shared Database Mode

Starting with MetaDefender 4.20.0, you can use a centralized database deployment model, where all MetaDefender Core instances can share the same external database.

That helps auto-sync all product configurations (**except Single-Sign-On** that requires unique settings for each Core), processing history, live/on-going scan result query, audit, statistics data.



This article demonstrates how to install and setup MetaDefender Core 4.20.0 or newer with shared database mode.

Important notes & current limitations:

Currently not supported

- Does not support Central Management use-case with shared DB mode.
- Does not support to switch from shared database mode back to standalone database mode.
- Does not support to change instance_name of MetaDefender Core in shared database deployment model after installation.
- Does not support upgrading older MetaDefender Core from non-persistent mode to using the shared database mode.
- Does not support upgrading older MetaDefender Core from standalone mode to using the shared database mode.
- Does not support to upgrade multiple MetaDefender Core instances at once in shared database mode
    - Stop all Core instances' scanning service before performing the upgrade.
    - Upgrade each Core server one by one instead.

- For each Core, when the upgrade finished, then you can start using that Core for scanning without waiting for the other Core instances' upgrade to be done.

> **Disclaimer**
> MetaDefender Core does not support mode switching.

Requirements for shared database setup:

- The central remote PostgreSQL server (for shared database) could be utilized with a built-in (native) PostgreSQL database server installed with one of the MetaDefender Core instances in the network. Or using a dedicated existing PostgreSQL database server (see detailed requirements for PostgreSQL version at Recommended System Configuration).
- Before setting up, make sure to follow steps at Open Connection On PostgreSQL Server so that all MetaDefender Core instances could connect to the central remote PostgreSQL server.
- MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core, details: PostgreSQL User Privilege Requirements
- All MetaDefender Core instances must be installed on the same Operating System (Windows / Linux).
- The first MetaDefender Core instance connects to the shared database must be ready first (license activated, users/groups/roles created) before setting up other MetaDefender Core instances to connect and use that shared database.

Engine update:

- If engine updates is configured to be via Folder source, all MetaDefender Core instances must be able to access to the configured update folder, and the option DELETE FILES AFTER IMPORT will be unchecked and prevented for modification.

Authentication:

- Single-Sign-On (SSO) authentication settings (if using) cannot be shared among MetaDefender Core instances because SSO requires an unique login URL for every different Core.
- If an user is locked for login on a certain MetaDefender Core instance(e.g. exceeded the number of failed login times), that user is not prevented for logging in on any other instances.
- If an user is locked for sign-in on a certain MetaDefender Core instance (e.g. exceeded the number of failed login times), if the admin releases lockout on that user on that MetaDefender Core instance, that user is still not able to login on other MetaDefender Core instances.

Sanitized / DLP / Quarantined location changed after MetaDefender Core install:

Windows:

Shared storage path will be stored in 3 strings under key global in registry: quarantinepath, dlppath, sanitizepath as below

(Default)	REG_SZ	(value not set)
dbmode	REG_SZ	4
dlppath	REG_SZ	\\10.40.50.100\test
instance_name	REG_SZ	481rvj
quarantinepath	REG_SZ	\\10.40.50.100\test
restaddress	REG_SZ	*
restport	REG_SZ	8008
sanitizepath	REG_SZ	\\10.40.50.100\test

You can set either 1, 2 or 3 different paths for the 3 settings above (means that quarantined, DLP and sanitized files will be stored in same or different storages corresponding). In case shared storage is not set in installation, values of 3 settings above will be empty.

Restart MetaDefender Core service after changed.

Linux:

Open file `/etc/ometascan/ometascan.conf` and set shared storage path for 3 keys quarantinepath, dlppath, sanitizepath under section [global]

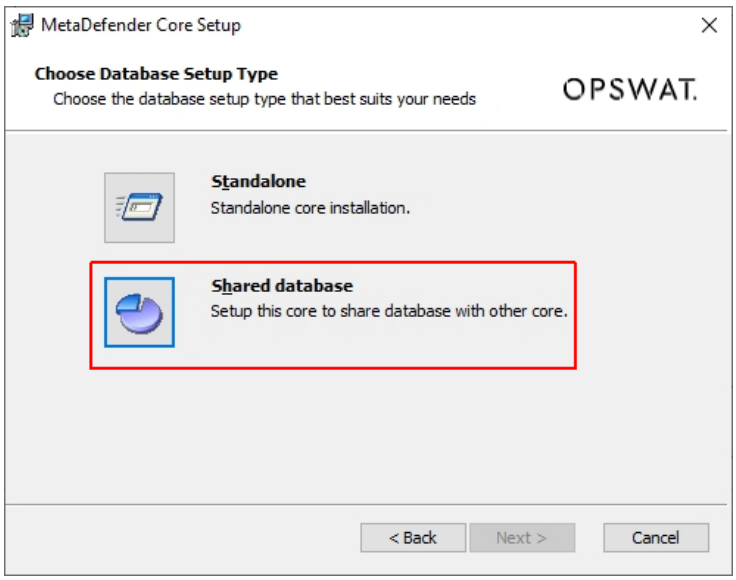
Restart ometascan service after changed.

Wizard Installation (Windows Only)

> **Info**
> Only applicable to MetaDefender Core 4.20.0 or newer using wizard on Windows OS.

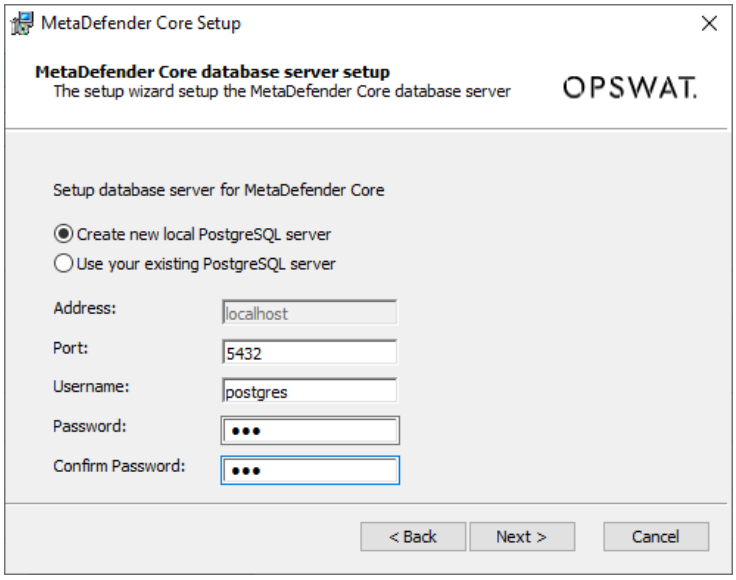
Clean install scenario

1. **Choose Shared database option:**

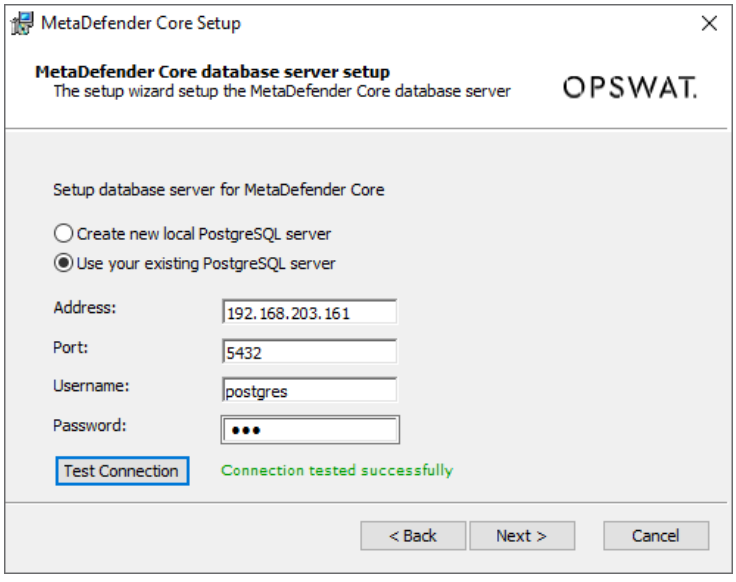


2. **Setup database server for MetaDefender Core:**

You can choose to create a native / built-in PostgreSQL server and setup the shared database on (then other MetaDefender Core instances will want to connect to that PostgreSQL server afterwards)



Or choose to connect to a pre-setup existing PostgreSQL server:



PostgreSQL connection configuration

"Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

PostgreSQL user privilege requirements

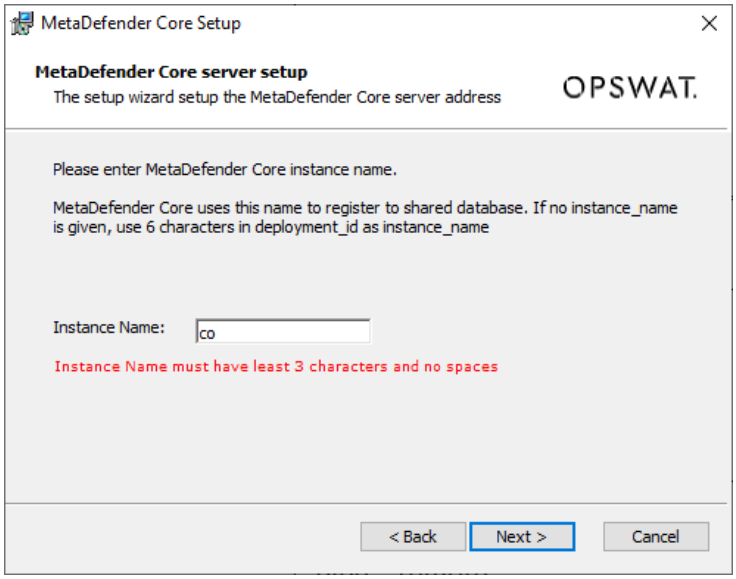
MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.

Details: PostgreSQL User Privilege Requirements

3. **Create an unique instance name for that MetaDefender Core (different from other MetaDefender Core instances):**

Instance name must contain at least 3 characters and no white space.

If you choose to leave it blank, which means it will use the default instance name with 6 characters extracted from that MetaDefender Core deployment ID.

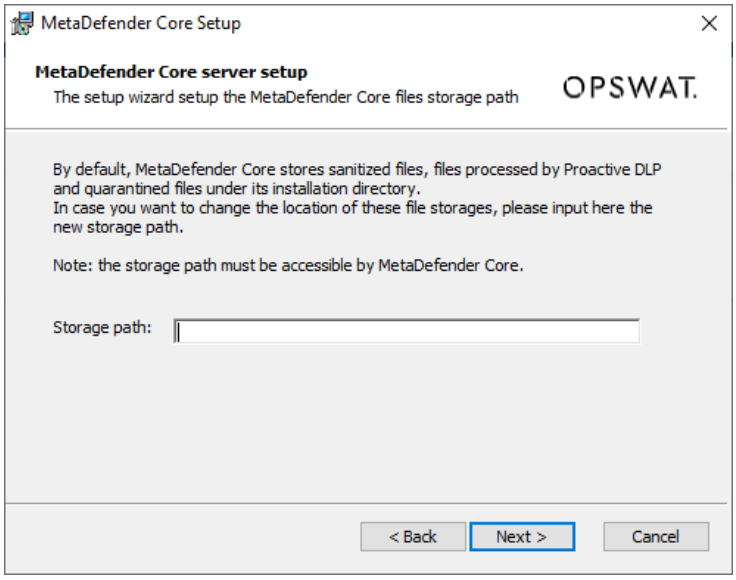


4. **Input the file storage path (sanitized, DLP, quarantined files)**

In the shared database model, all MetaDefender Core instances could use the same folder path to store all their own sanitized, DLP, quarantined files after processed.

The default location is: ***<MetaDefender Core installation folder>\data\ {sanitized|dlp|quarantine}***

If you wish to use that default location, just leave the input box blank and press Next.



Otherwise, you can specify a different location of sanitized/dlp/quarantined files. It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core.

For example:

- C:\new_folder
- \192.168.1.1\new_folder

Command Line Installation

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Clean install scenario

1. Create ignition file if not existed

- Windows: **C:\OPSWAT\ometascan.conf**
- Linux: ***/etc/opswat/ometascan.conf***

An example of ignition file:

ometascan.conf bash

```
eula=true

[global]
dbmode=4
instance_name=MetaDefenderCoreA
storage_path=D:\path\

[user]
name=admin
password=admin
email=admin@local

[dbserver]
type=remote
host=192.168.1.1
port=5432
user=postgres
password=your_own_password
```

Field	Required	Mapping to wizard UI	Possible values
global/dbmode	optional default is 1 (standalone)		1 - Standalone 2 - Central Hub deployment 3 - Non-persistent **4 - Shared**
global/instance_name	optional default is 6 characters of deployment ID		Unicode and number At least 3 characters No whitespace
global/storage_path	optional default is empty		Path to storages (DLP, quarantined, sanitized files) It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core. For example: • C:\folder • \192.168.1.1\folder

2. Run installers using command line to install

Please refer to Command Line Installation for installation command.

Deployment automation support

Note

An Ignition file could be utilized for either / both of following use-cases:

1. Pre-define PostgreSQL server information for MetaDefender Core to connect and setup database: only required on Linux installation, or Windows with command line installation. Learn more details at:
    a. Standalone Mode > Command Line Installation (applicable on version 4.19.0 or newer).
    b. Shared Database Mode > Command Line Installation (only applicable on version 4.20.0 or newer).
2. Automating to accept EULA, create a default local admin user and import MetaDefender Core configurations upon installed: Those steps could be automated using ignition file described in this page.

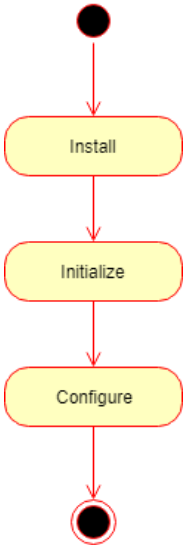
Ignition file location to create:

- **Windows**: `C:\OPSWAT\ometascan.conf`
- **Linux**: `/etc/opswat/ometascan.conf`

The product supports fully automated deployment. It means that it can be installed and configured with no human interaction.

The automated deployment can be split to three steps on a high level:

1. Installation
2. Initialization
3. Configuration



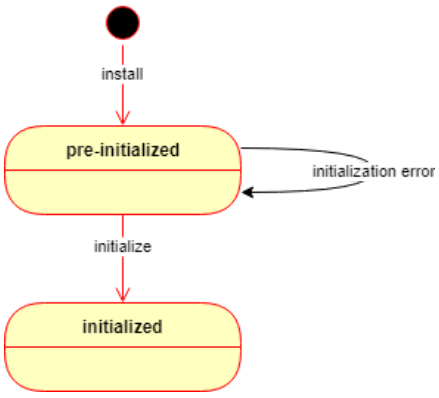
Installation

To automate the installation, install the product from the command line and provide the installation-time options as parameters to the installer. For further details see Command Line Installation

After the installation is complete, the product starts up and waits in a pre-initialized status. The product may be initialized in two ways:

1. Manually using the Wizard Setup or
2. Automatically using an *ignition file* (see below).

> **Warning**
> If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried after fixing the problem (e.g. placing the ignition file in its lookup location) and restarting the OPSWAT MetaDefender Core service.
>
> Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

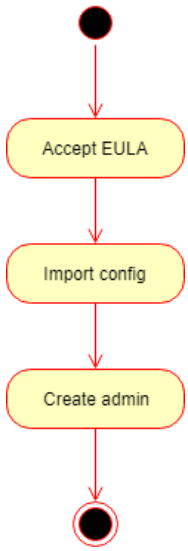


Initialization

Initialization is the process of bringing the product to an operable status.

Basically the initialization consists of the following steps:

1. Accept the End User License Agreement (EULA),
2. Import product configuration and
3. Create the first administrator user account.



Ignition file

The initialization process can be configured in a file called the *ignition file*.

> **Warning**
> In Linux, the ignition file and the folder /etc/opswat should have the permission 755

Ignition file location

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

The ignition file must be in conf format.

ometascan.conf bash

```
eula=true
[user]
name=admin
password=admin
email=admin@local
apikey=44c96cbb4e4e6c0a1d16bcb3c3da24def317
[config]
import=config_package.zip
import_password=123
[security]
session_cookie_samesite=Lax
[internal]
skip_startup_pages=true
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
private_username=my_internal_db_user
private_password=mypassword
[ocm]
ocm_url=https://my.opswat.com
reg_code=XXXXXXXX
```

Ignition file fields

The ignition file accepts the following fields:

Section	Key	Required	Accepted Values	Description
	eula	Optional (mandatory when having [user] section)	true false (default)	Whether to accept the End User License Agreement. This key must be set at the top of ignition file. This key must be set to "true" to accept the EULA. Any other value will cause the initialization to fail. When having [user] section defined, then this key must be set.
[user]		Optional		Initial administrator user account properties. The Administrator role is granted to the account.
	name	Optional	string	User name for the initial administrator user account.
	password	Optional	string	Password for the initial administrator user account. WARNING! Clear text password The password in this configuration file must be stored in its clear-text format and as so it may be visible for unauthorized parties.
	email	Optional	string	E-mail address for the initial administrator user account.
	apikey	Optional	string	API key for the initial administrator user account.
[config]		Optional		
	import	Optional	string	Path to a file in json/zip format that contains a previously exported configuration to be imported. Zip-format file contains a full configuration including user management settings, so it will overwrite [user] section.
	import_password	Optional	string	Password to decrypt .zip configuration package.
[security]		Optional		

Section	Key	Required	Accepted Values	Description
	session_cookie_samesite	Optional	Strict Lax (default) None	Declare if session cookie should be restricted to a first-party or same-site context.
[internal]		Optional		
	skip_startup_pages	Optional	true false (default)	Skip all welcome and other startup pages (e.g. upgrade result) when installing / upgrading MetaDefender, and going straight to the product index default page.
[dbserver]		Optional		
	type	Mandatory when having [dbserver] section	local remote	Where the PostgreSQL database server should physically locate. "remote" means PostgreSQL server and MetaDefender Core are not in the same machine.
	host	Mandatory when having [dbserver] section	string	IP address / domain name of the server where PostgreSQL server locates. "localhost" can be used when applicable.
	port	Mandatory when having [dbserver] section	number	Port of PostgreSQL server is listening for connections from clients (i.e. MetaDefender Core).
	user	Mandatory when having [dbserver] section	string	PostgreSQL server's user. SUPERUSER privilege is required for MetaDefender Core to setup its database and extensions for the first time. **Only non-Unicode characters supported.**

Section	Key	Required	Accepted Values	Description
	password	Mandatory when having [dbserver] section	string	PostgreSQL server's user credentials. **Only non-Unicode characters supported.**
	private_username	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
	private_password	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
[ocm]		Optional		
	ocm_url	Optional	string	URL to My OPSWAT server.
	reg_code	Optional	string	A registration code to enroll to My OPSWAT server.

Detailed initialization process

1) After the product has been started, it looks for the ignition file in the designated location.

Windows: C:\OPSWAT\ometascan.conf ** Linux: /etc/opswat/ometascan.conf

** 2) If an ignition file is found, then

2.1) It gets validated, and if it is valid, then based on the information found in the ignition file:

- The EULA is accepted,
- The configuration is imported,
- The administrator account is created.
- If any of the above steps fails, then the error is logged, and the initialization gets terminated.

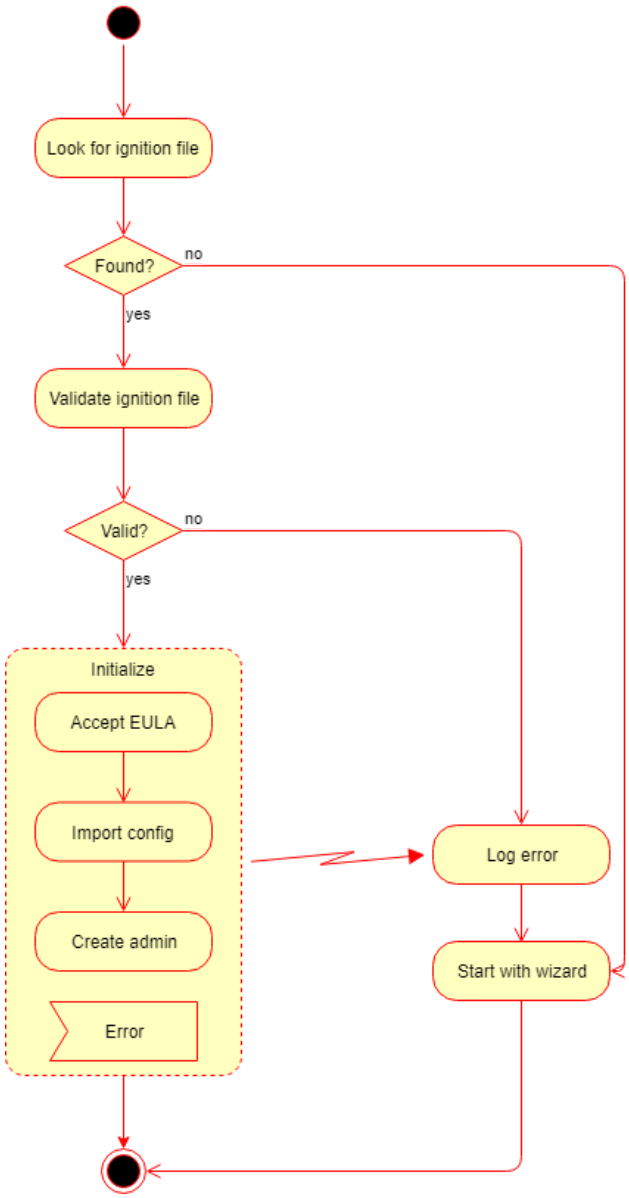
> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

2.2) If it is not valid, then the error is logged, and the initialization gets terminated.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

3) If there is no ignition file, then no initialization is performed.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.



Warning

If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried fixing the problem (e.g. placing the ignition file to its lookup location) and restarting the OPSWAT MetaDefender Core service.

Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

Configuration

After the initialization is complete, the product is ready with the default and the imported configuration.

This configuration can be later changed calling the configuration API functions. For further details about the API see Configuration related APIs API

CIS Level 1 Guidelines

Support Red Hat Enterprise Linux 9
For more details about Center for Internet Security (CIS) please refer to this document

Instruction steps

I. Install OpenSCAP

bash bash

```
yum install openscap-scanner scap-security-guide
```

II. Generate a result file and a HTML report using OpenSCAP scanner tool

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

III. Remediation of CIS Level 1 issues

Generate a remediation script based on the `ssg-rhel9-ds.xml` file:

bash

```
oscap xccdf generate fix --profile
xccdf_org.ssgproject.content_profile_cis_server_l1 --fix-type bash --output
remediations.sh /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

And execute remediation script.

bash

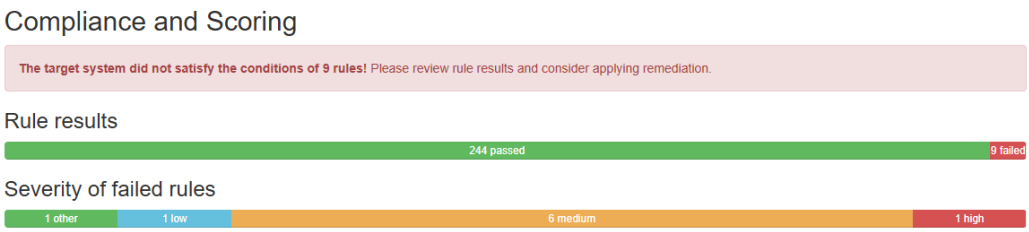
```
./remediations.sh
```

IV. Review the results after remediation

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

Example:



Notes

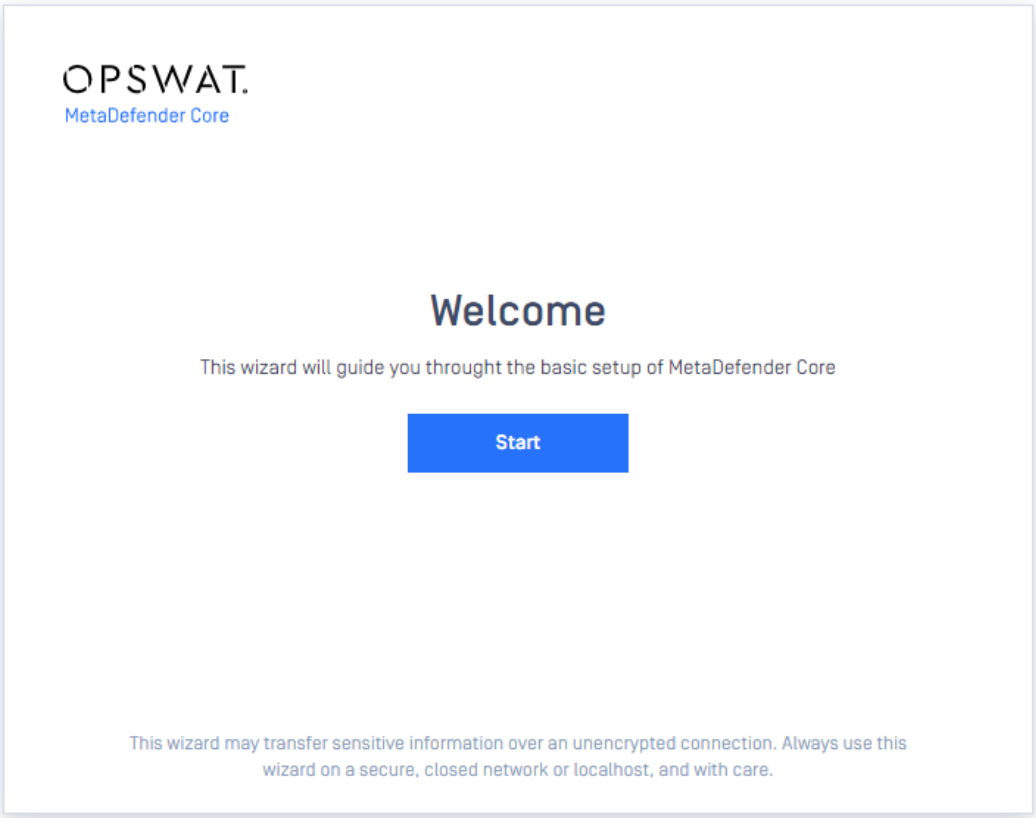
- CIS Level 1 requires `/tmp` folder to be mounted in a separate partition. Please ensure that that new partition have enough disk space for MetaDefender Core to run.
- CIS Level 1 requires that "Ensure No World-Writable Files Exist".
    - For now, when freshly installing MetaDefender Core, all its binary files meet the requirement.
    - When installing/updating engines, some engines might create additional files for its operation, and it might violate this requirement. In this case, you need to again execute the remediation script in the step III.

Wizard Setup

Introduction

Access the MetaDefender Core management console for the first time (i.e. http://localhost:8008), you are to complete a quick walkthrough for wizard setup in order to be able to use the product.

The administrative management console will be available only upon you have successfully finished this wizard stage.

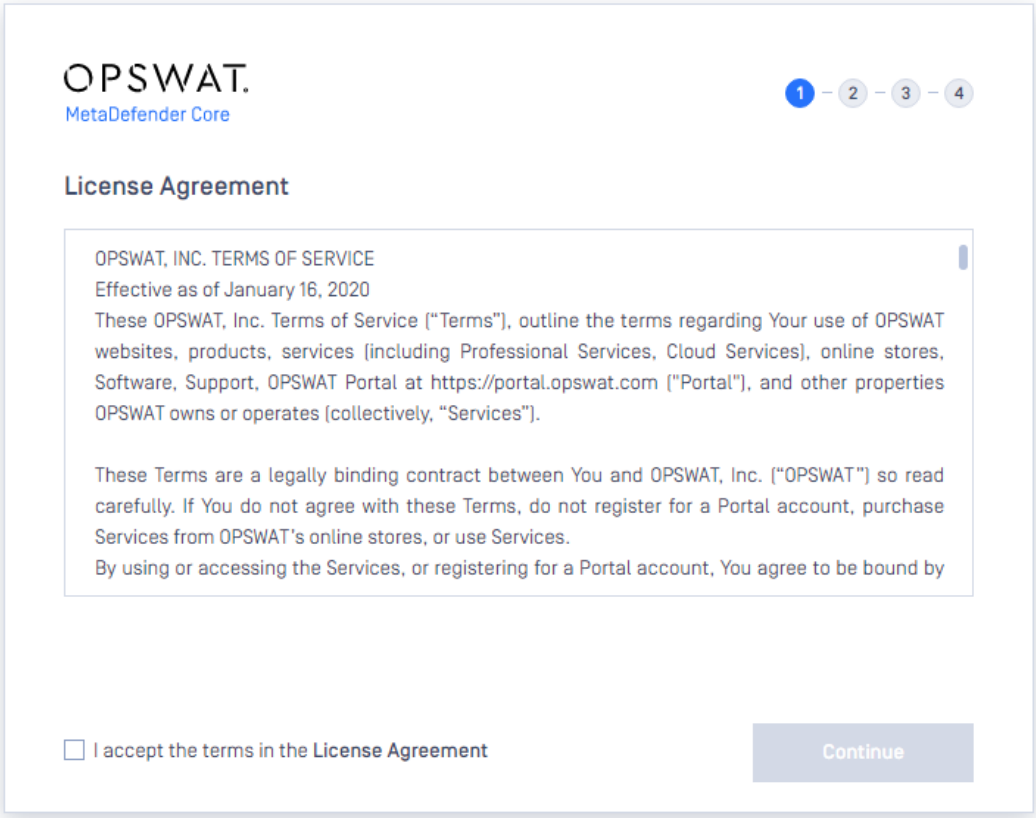


Warning

This wizard may transfer sensitive information over an unencrypted connection. Always use this wizard on a secure, closed network or localhost, and with care.

Basic configuration steps

End-User License Agreement



In the first page you can find the End-User License Agreement. You have to accept the terms before moving on.

Please read through the EULA (https://docs.opswat.com/policies/privacy-policy/portal-privacy-policy) carefully and if you agree with it, check `I accept the terms in the **License Agreement**` and click `Continue` to proceed.

Local Admin User Setup

The next step is to set up a default local administrator account.

This account will be the first one being able to access the administrative management console, and to create accounts for other users. You have to fill out all fields in this page to be able to move forward. When you are done, click NEXT to continue.

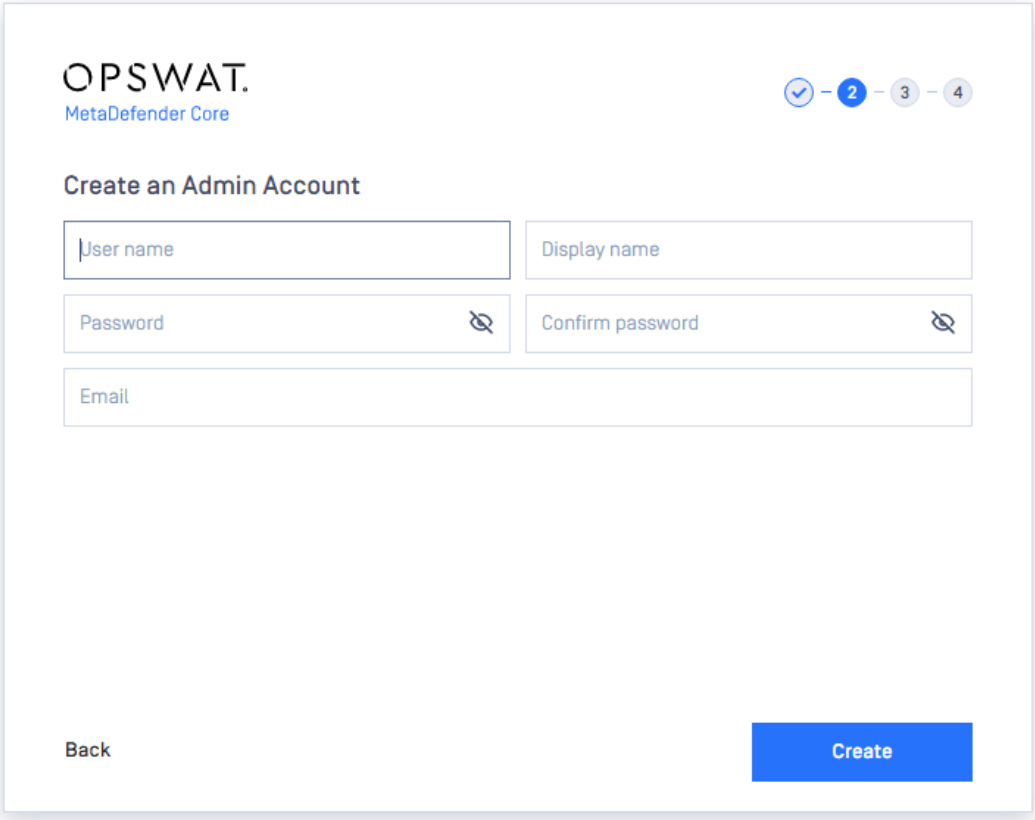
Info - User Directory

The administrator account, that is created via the basic configuration wizard, is always added to the LOCAL user directory as a member.

Learn more about user directory: User directories

The following information is required for the administrator account:

ACCOUNT NAME	The unique name of the account that is used at the time of login and in log messages for accountability.
ACCOUNT DISPLAY NAME	Name of the person bound to this account. This name (appended to the name of the account's user directory) is displayed in the top right corner of the administrative management console.
PASSWORD	Password of the user bound to this account that is used at the time of login. Passwords sent clear-text As long as TLS is not configured for the basic configuration wizard, passwords are sent clear-text over the network and may be disclosed to unauthorized parties. As a mitigation action: 1. Either use the wizard on *localhost* or on a direct network connection, or 2. Enabling HTTPS as soon as possible and change the password immediately if it has already been set.
EMAIL	Email address of the person bound to this account.



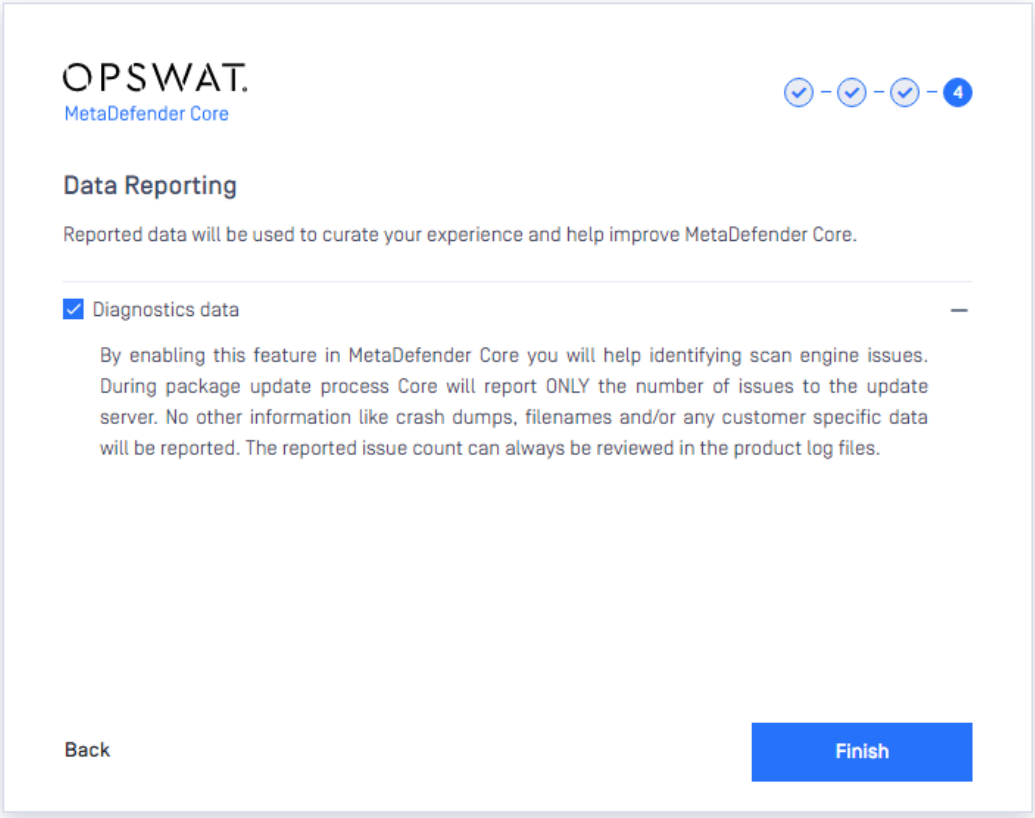
License activation

For license activation details see License Activation

Data Reporting

Enabling this option to allow MetaDefender Core collect its own engine issues and report back to OPSWAT for further analysis.

MetaDefender will not collect any other information.

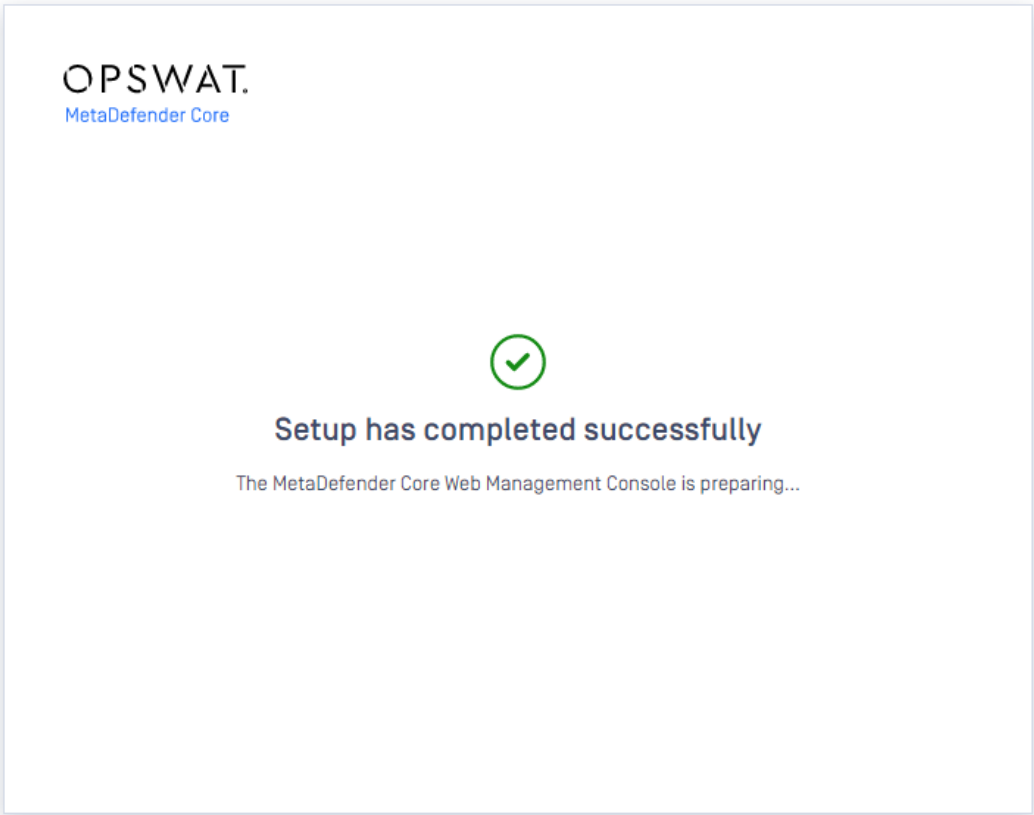


Wizard completion

After you have completed every steps you are ready to finish the wizard and start using the product. Click the `Fishish` button to complete the wizard.

> **Warning**
> The product's service will be restarted and the browser will be redirected to the administrative management console. This could take several seconds.

You can login to the Web Management Console with the administrator user that have just been created in the previous steps.



Setup HTTPS

Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network. Websites, like the Web Management Console, are able to use TLS to secure all communications between their servers and web browsers.

The TLS protocol aims primarily to provide confidentiality (privacy) and data integrity between two communicating computer applications.

Warning

No TLS for the wizard

By default, TLS is not enabled for the basic configuration wizard. As a consequence sessions between the wizard's backend and the browser may be insecure.

Info

Performing the same steps as for the Web Management Console, it is possible to set up TLS for the basic configuration wizard. Remember completing the TLS setup before launching the wizard.

For instructions to set up TLS see Enabling HTTPS

License Activation

##

To activate your installation go to the **Settings > License** menu in the Web Management Console. If you have no valid license, you will only see your installation's **deployment ID**. You will also see a warning in the Web Management Console header.

After successful activation, the product will start downloading the latest available scan engines and malware databases. You can follow the status of the scan engine installation on the **Inventory > Modules** page.

When all the licensed engines are installed, you can start using the MetaDefender Core to process files - File Processing.

Deployment ID

Each MetaDefender Core instance represented by an unique system fingerprint called `Deployment ID`

MetaDefender Core licensing is purely based on deployment ID activated. Therefore, everytime your MetaDefender Core is activated with a different deployment ID, then it will be captured as a different activation record accumulated into your total activation times allowed.

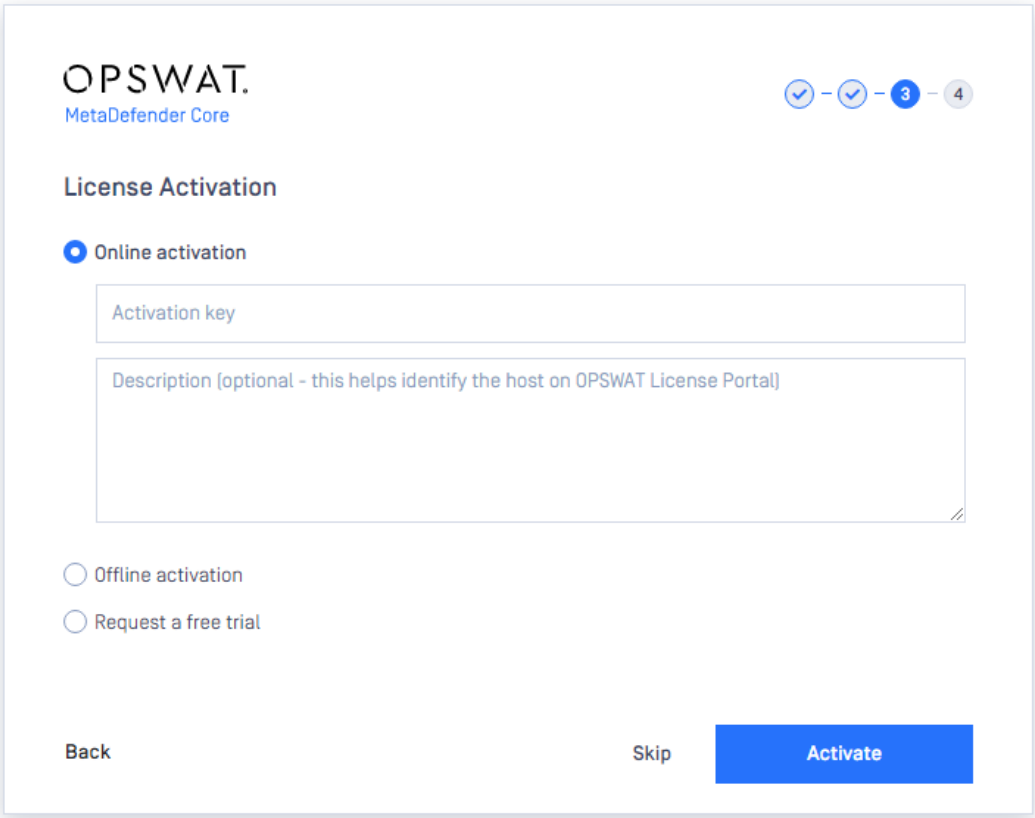
> **Notice**
> Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you alter the system where MetaDefender Core relies on (e.g. network card, OS hostname).
>
> Deployment ID is an UUID version 4 random based string. Learn more at: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you update the system where MetaDefender Core relies on.

Activation Modes

There are three different options for the product activation:

- Online activation: MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.
- Offline activation: Requiring you to upload a license file (.yml). That license file could be retrieved by activating your license via OPSWAT Portal. Follow the displayed instructions for details.
- Request trial key online: For evaluation purpose, you can receive an trial Activation key via email. Follow the displayed instructions for details.



Online activation

MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.

Required to input your Activation License key, and optional for description (but we recommend this for troubleshooting and auditing purpose if needed).

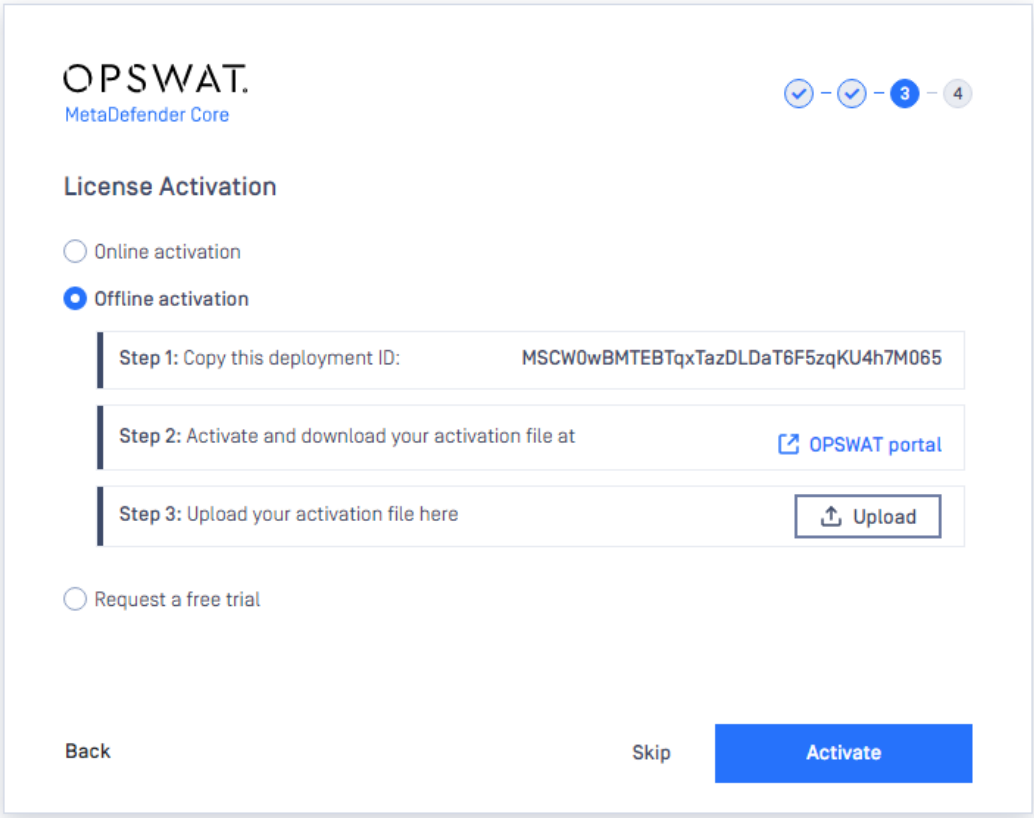
Warning

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts' for license activation:

- https://activation.dl.opswat.com Note: *IP address-based whitelisting on your firewall might fail after some time since OPSWAT uses CDN (Content Delivery Network) to faster delivery updates over the world, and IP address of edge servers might change over time.*

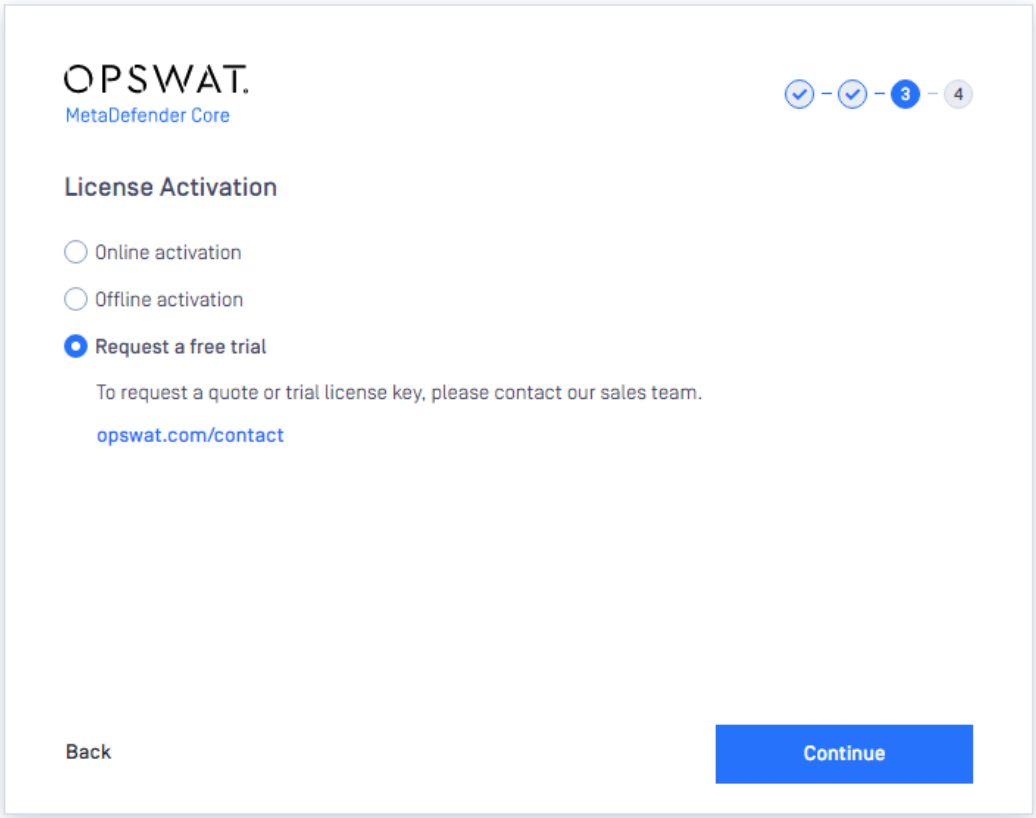
Offline activation

With no internet connection on the server the Metadefender v4 instance may be activated indirectly from a different machine, that has internet connection. The Deployment ID of the Metadefender v4 instance and the the Activation key received at the time of purchasing the product will be required. Follow the steps on the screen to activate the product offline.



Request trial key online

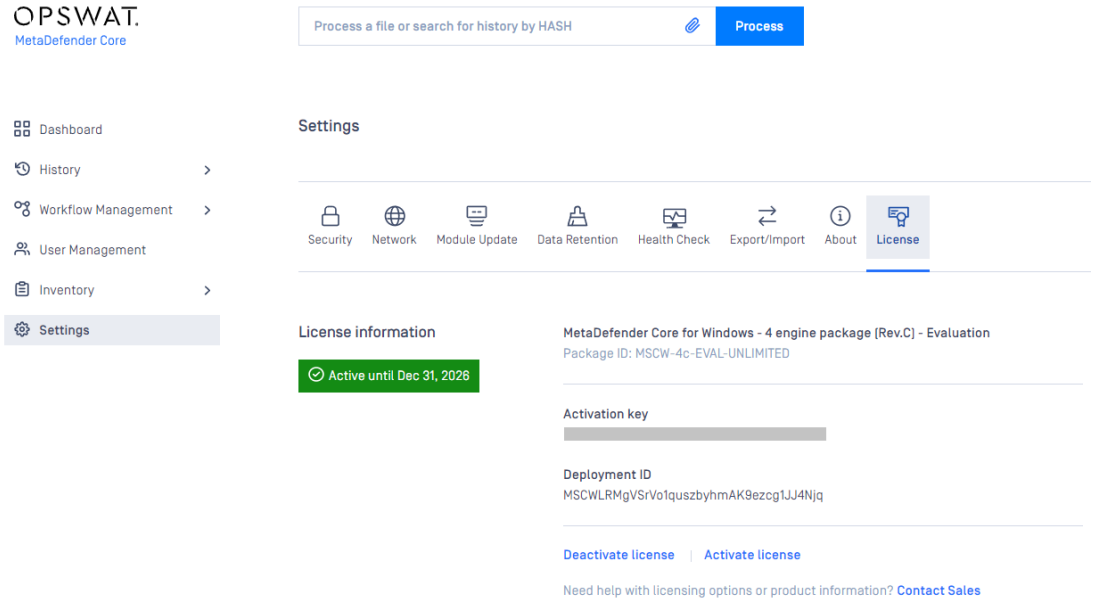
An evaluation license could be acquired by contacting our sales team.



Check Your License Details

For more license details and activating your MetaDefender Core, go to Settings > License menu tab on the Web Management Console:

- Product name: product name as on your order (e.g. MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation)
- Package ID: product identification as on your order (e.g. MSCW-4c-EVAL-UNLIMITED)
- Expiration: last day of license validity
- Activation key: key used for activating product
- Deployment ID: identification of this installation


OPSWAT.
MetaDefender Core
Process a file or search for history by HASH
Process
Dashboard
History
Workflow Management
User Management
Inventory
Settings
Settings
Security
Network
Module Update
Data Retention
Health Check
Export/Import
About
License
License information
Active until Dec 31, 2026
MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation
Package ID: MSCW-4c-EVAL-UNLIMITED
Activation key
Deployment ID
MSCWLRMgVSrVo1quszbyhmAK9ezcg1JJ4Njq
Deactivate license
Activate license
Need help with licensing options or product information? Contact Sales

Scripted license management

Using REST API calls there is an option for scripted activation and deactivation for MetaDefender Core servers with **no Internet connection or if your infrastructure is using automation to create/destroy MetaDefender Core instances**. In this chapter the steps of these two scenarios are described.

Requirements

- an installed MetaDefender Core instance without Internet connection
- another computer that has Internet connection and can run your activation scripts
- a manual or automated way to transfer data between the two computers

Activation steps

1. In order to activate MetaDefender Core, deployment ID and activation key are needed to generate the license file.
    a. Activation key should be purchased from OPSWAT.
    b. MetaDefender Core deployment ID can be queried by using REST API (for details see GET Get Current License Information API page)

> **Info**
> Save this deployment ID in your system, you might need this ID when the instance is unavailable at the time of deactivation.

2. Activate license and get the license file using the following URL, where <quantity> is 1.

activation call bash

```
curl -X GET https://activation.dl.opswat.com/activation?key=<activation-key>&deployment=<deployment unique-ID>&quantity=<quantity>
```

If the activation is successfully, a license file is downloaded. Save this file.

HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	license file	Activation was successfully.
200 Activation failed	error: '<user conform error message>'	Failed activation
200 Invalid parameter	error: 'Could not activate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not activate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
200 Invalid parameter	error: 'Could not activate your product because the quantity you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key, quantity or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Info
In case of any activation issue, contact OPSWAT support for help

3. Upload license file to MetaDefender Core: The license file should be uploaded to the MetaDefender Core v4 to activate the product. For details see page: POST Upload license key API

Deactivation steps

1. For deactivation, following infomation is required:
    a. Activation key
    b. MetaDefender Core deployment ID: can be queried by using REST API (for details see GET Get Current License Information API page)
2. Deactivate license using the following URL:

deactivation call bash

```
curl -X GET https://activation.dl.opswat.com/deactivation?key=<activation-
key>&deployment=<deployment unique-ID>
```

Response HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	result: ok	Successful deactivation
200 No active license found	error: 'Could not found any active license with the given parameters'	The license has not been activated yet or it has been deactivated already.
200 Invalid parameter	error: 'Could not deactivate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not deactivate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Important notes

Warning

Product activation is tied to several hardware, operating system and software parameters. In case of one or more major hardware or software parameter change the product might turn into deactivated status. Operating system updates, other software updates should not affecting the activation status.

If the product is online activated and has live Internet connection then in case of deployment ID change the product reactivates itself. If the product doesn't have Internet connection then the administrator is responsible to reactivate the product.

Because of the above administrator should consider licensing restrictions of the offline deployments if using any virtualization and/or containerization technologies.

File Processing

MetaDefender Core provides REST API interface for file processing business.

There are several ways to process files with MetaDefender Core:

- Analyze Files via Web Interface
- Process Files via REST API

Product Upgrading

> **Info**
> Simply run installer file, details could be found at Installation

> **Info**
> For command line mode, upgrading (e.g. on **Linux**), prior to running the installer file, please make sure to **create the ignition file** with PostgreSQL server information included. Details:
>
> - Standalone DB mode: Command Line Installation (Windows and Linux)
> - Shared DB mode: Command Line Installation

Recommendation for MetaDefender Core upgrade

Our goal is to help you execute the product upgrade continuously, efficiently, and correctly.

PostgreSQL service upgrade

- **Bundled PostgreSQL Service**: If you are using the bundled PostgreSQL server included in the installer, the PostgreSQL service will be automatically upgraded to the newer version (if available). No additional action is required for this process.
- **Remote PostgreSQL Service**: For remote PostgreSQL servers, customers are responsible for managing and upgrading the service independently. Upgrading MetaDefender Core does not update the remote PostgreSQL service.

Before the upgrade

Prepare a proper disaster recovery plan (DRP) including some common practices:

- Backup, or take a snapshot of your current working MetaDefender Core server.
  - This is to help quickly rollback for any upgrade failure might cause later for any reason.
- Optionally backup your PostgreSQL database, especially when using remote shared database mode.
  - Reference: https://www.postgresql.org/docs/current/backup.html
  - Please do it after stopping all linked MetaDefender Core instances to avoid potential data collision.
- Lastly, if backing up entire server is not feasible, then at least export MetaDefender Core configuration package Import/Export configuration just in case you'll need to start off a new server with current MetaDefender Core configuration to keep the scanning service running without any impact.

On the PostgreSQL server, double check server resource to ensure sufficient RAM memory and disk space to be preserved for the upgrade process.

- 8GB RAM available at least during the upgrade.

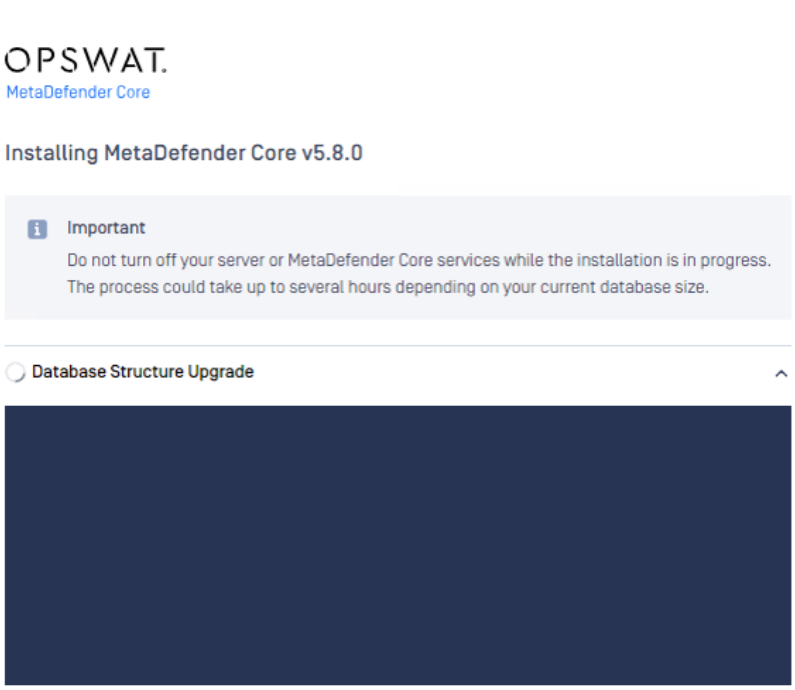
- Disk availability is greater than MetaDefender Core database size.

While doing the upgrade

Applicable for shared database deployment:

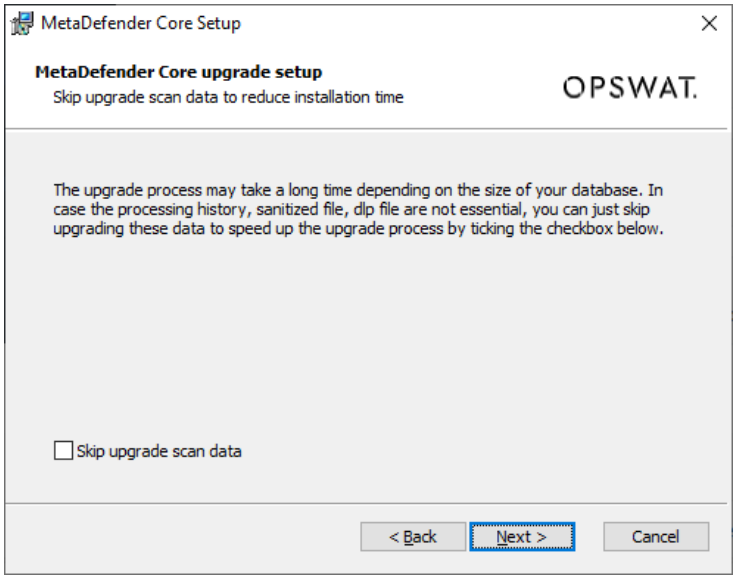
- Stop all MetaDefender Core instances' scanning service before performing the upgrade.
- Upgrade one (1) MetaDefender Core instance at a time.
- Proceed to upgrade MetaDefender Core instance where the bundle local PostgreSQL server is used for shared database setup.
- Otherwise is using remote PostgreSQL then the upgrade order does not matter.

For large size database, mostly for scan history, expecting long time to be spent on the upgrade process.



The product supports an option to skip data migration for scan history to expedite the process drastically and mitigate any failure risk. You can manually backup all data from your PostgreSQL server for offline usage if any.

Check "Skip upgrade scan data" to enable this feature while doing the upgrade.



or add following key in the ignition file:

none

```
[internal]
skip_upgrade_scan_data = 0
```

Troubleshooting

If the upgrade process ever runs into an error:

- Double check your network and services, applicable to remote PostgreSQL server.
- Optionally configure MetaDefender Core to skip migrating scan history.
  - Windows
    - Open Windows registry, and navigate to HKLM\Software\OPSWAT\Metascan
    - Add a new key `internal/skip_upgrade_scan_data` with value = 1
  - Linux
    - Open MetaDefender Core configuration file `/etc/ometascan/ometascan.conf`
    - Add new key into the file
      - [internal]
      - skip_upgrade_scan_data=1
  - Note: This key will be auto removed upon the upgrade finishes.
  - Then restart MetaDefender Core service to retry the upgrade.
    - Might retry this a few times until the upgrade is done.

Just in case the upgrade process still fails, please contact OPSWAT support with the support package - How to Create Support Package? This will help us analyze the root cause and figure out an optimal solution for your system failure.

Using external Load Balancer

Application Load Balancing (Layer 7)

This page provides information about recommended ways to use sessions with a Layer 7 load-balancer.

Most load-balancers have the ability to provide cookies for pinning subsequent traffic from a client to the appropriate server. This method is called sticky session, session persistence or session affinity. Using cookies efficiently requires the client to know when a cookie should be sent, should not be sent or should be deleted.

Sticky session load balancing

Each Metadefender Core instance has its own databases and application sessions that cannot be seen by other instances. Therefore, in order to get the related data/response to our queries we should ask the appropriate Metadefender Core v5 server. On the other hand, to keep the advantages of the used load balancing method, cookies should not be sent if it is not necessary.

Single file scanning

Step	Stage	Task	Cookie usage
1.	Sending file	Initiate processing a file on the client side. Send file through the load-balancer. (See POST File Analysis API)	Cookie should not be sent
2.	Sending file	Save the cookie and the data_id you got from the load-balancer	Save cookie
3.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Send cookie saved in step 2
4.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Send cookie saved in step 2

Batch scanning

Step	Stage	Task	Cookie usage
1.	Open batch	Initiate processing file(s) in batch. Request a batch ID through the load-balancer. (See POST Initiate Batch API)	Cookie should not be sent
2.	Open batch	Save the cookie and the batch_id you got from a Core server through the load-balancer.	Save cookie
3.	Sending files	Send file through the load-balancer. (See POST File Analysis (using batch header) API)	Send cookie saved in step 2
4.	Sending files	Save the data_id you got from the load-balancer.	-
	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Send cookie saved in step 2
5.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
6.	Getting batch status	See GET Batch Status API	Send cookie saved in step 2
7.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Send cookie saved in step 2
8.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API	Send cookie saved in step 2

If it does not matter which upstream server responds, the queries should be sent without cookie.

> **Info**
> It is recommended not to send cookies when it's not necessary to allow load-balancer to use its own method to share the load between Metadefender Core v5 servers.

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration through load-balancer

OPSWAT products that support HTTP load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	4.3.4	-
OPSWAT Client	Windows: 7.6.247.0 Mac: 10.4.243.0	-
MetaDefender Email Security	4.3.0	-
MetaDefender ICAP Server	4.3.0	-
MetaDefender Vault	1.3.0	-
MetaDefender for Secure Storage	2.0.7	-

DNS Load Balancing

Using this method is logically similar to Layer 7 load-balancing.

Briefly how it works

Client uses a domain name to send a query to a server. Client's DNS server has more "A" records for that name with different IPs. When a client resolves the server's domain name DNS server randomly chooses an IP for that name to send back. When a session is used on the application layer, client should know the IP address of the Core server that handles that specific session. In every other case, client should resolve the domain name with DNS query to let requests to be balanced between Metadefender Core servers.

Single file scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Sending file	Initiate processing a file on the client side. (See POST File Analysis API)	Use IP saved in step 2
4.	Sending file	Save the data_id got from the Core server	-
5.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Use IP saved in step 2
6.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Use IP saved in step 2

Batch scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Open batch	Initiate processing file(s) in batch. Request a batch ID. (See POST Initiate Batch API)	Use IP saved in step 2
4.	Sending files	Send file to the specific Core server with the basch ID saved in step 3. (See POST File Analysis (using batch header) API)	Use IP saved in step 2
5.	Sending files	Save the data_id you got from the Core server.	-
6.	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Use IP saved in step 2
7.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
8.	Getting batch status	See GET Batch Status API	Use IP saved in step 2
9.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Use IP saved in step 2
10.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API).	Use IP saved in step 2

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration via DNS load-balancing

OPSWAT products that support DNS load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	does not support yet	-
MetaDefender Client	does not support yet	-
MetaDefender Email Security	does not support yet	-
MetaDefender ICAP Server	does not support yet	-
MetaDefender Vault	does not support yet	-

SSL connection for PostgreSQL communication

> **Note**
> The SSL connection configuration is possible thanks to the support made by PostgreSQL. For reference: https://www.postgresql.org/docs/14/ssl-tcp.html
>
> MetaDefender Core is tested against this mode on particular version 4.21.1

Step 1: Generate SSL certificate for PostgreSQL server

> **Note**
> Following instructions are for self-sign certificate. For production environment, making sure to use the certificate signed by your trusted CA.

1) Generate a private key

bash

```
openssl genrsa -des3 -out server.key 1024
```

Sample output:

bash

```
Generating RSA private key, 1024 bit long modulus ........++++++
...............++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

Type your desired password / pass phrase to encrypt the private key

2) Remove the pass phrase to automatically start up the PostgreSQL server

bash

```
openssl rsa -in server.key -out server.key
```

> **Note**
> On Linux, make sure to set permission on the server.key file

bash

```
chmod og-rwx server.key
```

3) Create a self-signed certificate

bash

```
openssl req -new -key server.key -days 3650 -out server.crt -x509
```

You will be prompted to enter detailed information which is incorporated into your self-signed certificate request.

none

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:CA
Locality Name (eg, city) [Default City]:San Francisco
Organization Name (eg, company) [Default Company Ltd]:OPSWAT
Unit Name (eg, section) []:OPSWAT
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:dummy_email@opswat.com
```

For self-signed certificate, use the server ceriticate as the trusted root certificate:

bash

```
cp server.crt root.crt
```

Copy **server.key**, **server.crt**, **root.crt** to PostgreSQL data folder. For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data**
- Linux: **/var/lib/ometascan/pg_data/**

Step 2: Configure PostgreSQL server for SSL authentication connection

1) Create a custom config file (e.g. ssl.conf) for PostgreSQL.

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\postgres\ssl.conf**
- Linux: **/var/lib/ometascan/postgres/ssl.conf**

2) Content of ssl.conf:

none

```
ssl = on
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
```

3) Modify pg_hba.conf

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data\pg_hba.conf**
- Linux: **/var/lib/ometascan/pg_data/pg_hba.conf**

Add following (modify **::1/128** if needed to match with your environment requirement)

none

```
hostssl all all      ::1/128      scram-sha-256
```

After the modify, the content of pg_hba.conf should be like this (example):

> **Note**
> In a Linux environment, a local connection is required to upgrade from version 5.5.1 or earlier to version 5.6.0 or later.

none

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local         DATABASE  USER  METHOD  [OPTIONS]
# host          DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl     DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostgssenc    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnogssenc  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
local    all all                    scram-sha-256
hostssl all all      ::1/128      scram-sha-256

#More content.......
```

4) Reload the services

- Standalone DB mode: Restart **ometascan** service (MetaDefender Core service)
- Shared DB mode: Restart **ometascan-pg** / **ometascan-postgresql** service (MetaDefender Core PostgreSQL service) and then **ometascan** service (MetaDefender Core service)

Test your SSL database connection:

bash

```
psql.exe "sslmode=require dbname=postgres user=postgres host=localhost port=5432"
Password for user postgres:
psql (12.7)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
Type "help" for help.
```

This message indicate that you are now using SSL connection:

bash

```
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
```

Performance and Load Estimation

> **Disclaimer**
> These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Some factors that affect performance

- MetaDefender Core version
- MetaDefender Core engine package and configuration
  - set of engines (which and how many)
  - product configuration (e.g., thread pool size)
- System environment
  - server profile (CPU, RAM, hard disk)
  - client application location - remote or local
  - system caching and engine level caching
- Dataset
  - encrypted or decrypted
  - file types
    - different file types (e.g., document, image, executable)
    - archive file or compound document format files
  - file size
  - bad or unknown (assume to be clean)
- Performance tool

How test results are calculated

Performance (mainly scanning speed) is measured by throughput rather than unit speed. For example, if it takes 10 seconds to process 1 file, and it also takes 10 seconds to process 10 files, then performance is quantified as 1 second per file, rather than 10 seconds.

- total time / total number of files processed: 10 seconds / 10 files = 1 second / file.

Standalone DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.2.416.0

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

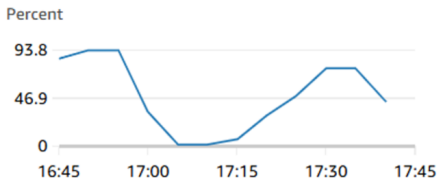
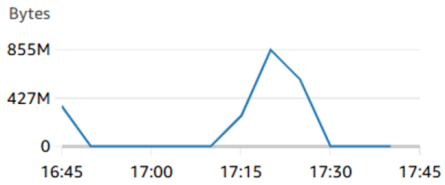
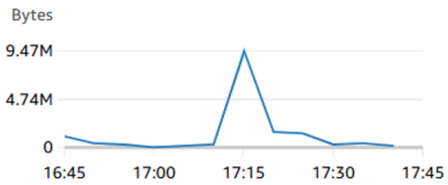
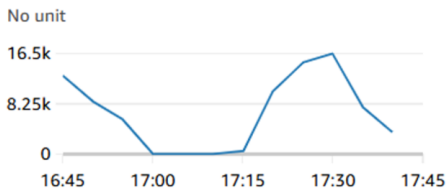
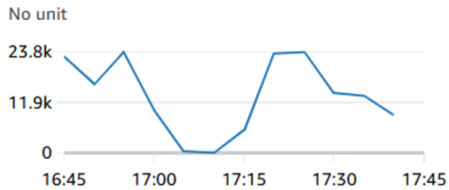
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV)	24.2	47.2 / 80	37.3 / 41.1		315,310	0.0114
Deep CDR	19.9	48.9 / 81.8	32.6 / 37		383,442	0.0094
Proactive DLP	17.1	36.8 / 58.2	30.4 / 33.6		446,228	0.0081
Vulnerability	10.1	59.2 / 85	29.3 / 33.3		755,495	0.0048

MetaDefender Core with common engine packages:

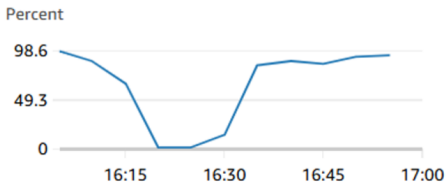
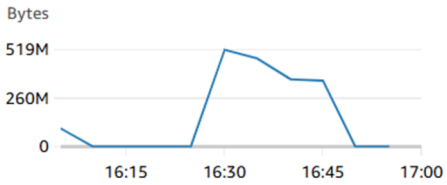
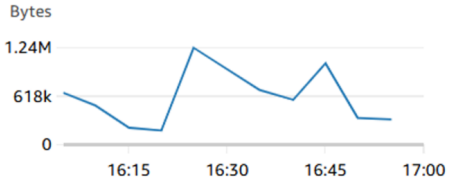
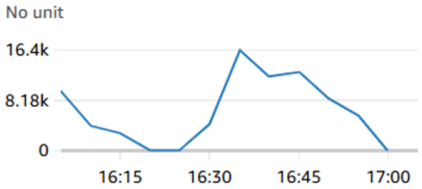
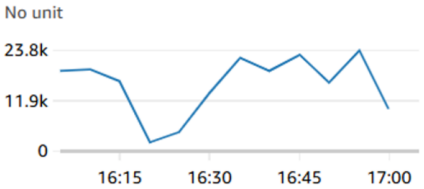
Use case	Scan duration	Avg./Max CPU usage	Avg./Max RAM usage	Avg./Max Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV) + Deep CDR	30.8	68.1 / 88	40.1 / 46.8		247,744	0.0145
Metascan (8 AV) + Deep CDR + Proactive DLP	31.2	74.2 / 98.6	41.5 / 51.1		244,567.3	0.0147
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	31.8	73.5 / 98	40.6 / 51.5		239,953	0.015

System resource utilization charts

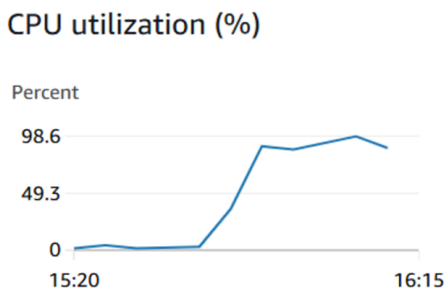
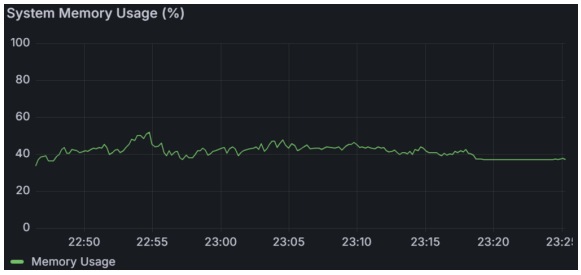
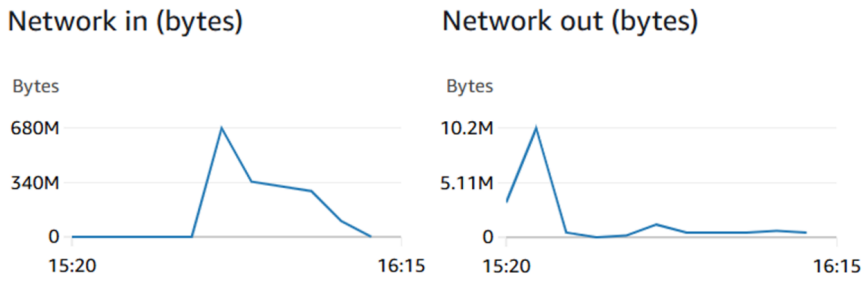
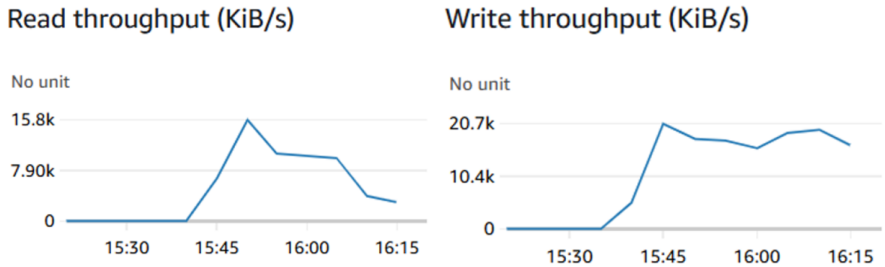
MetaDefender Core with Metascan (8 AV engines):

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

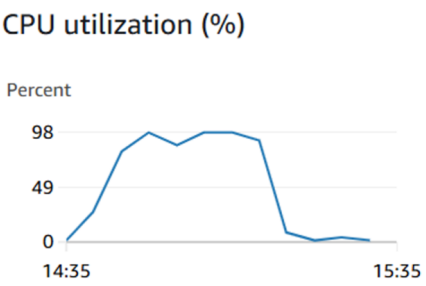
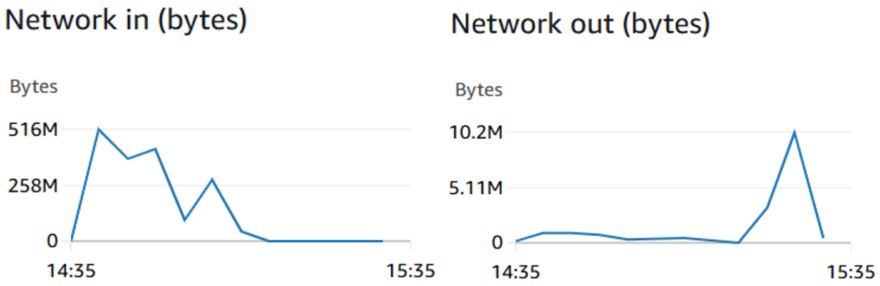
MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

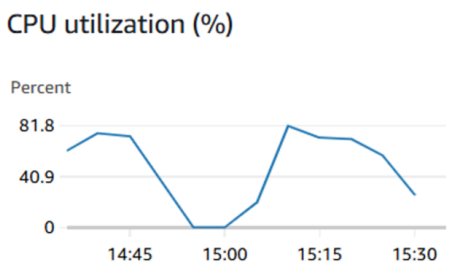
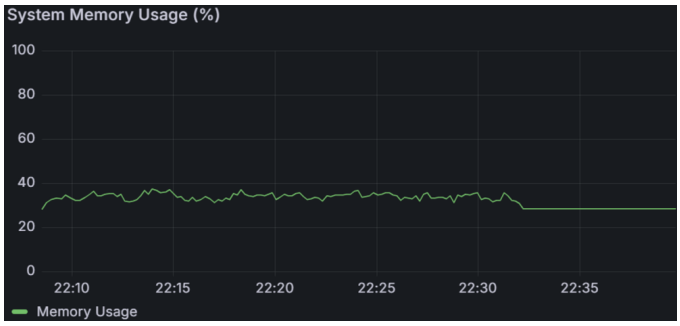
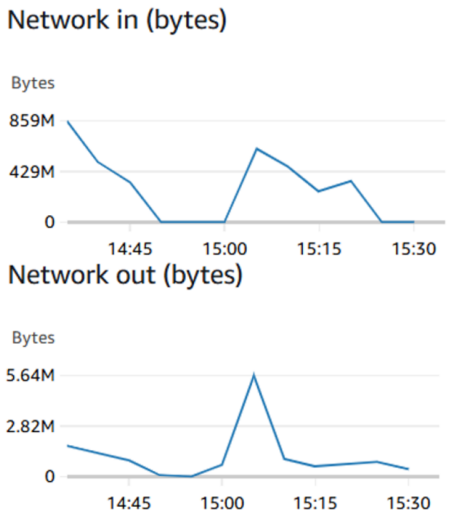
MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

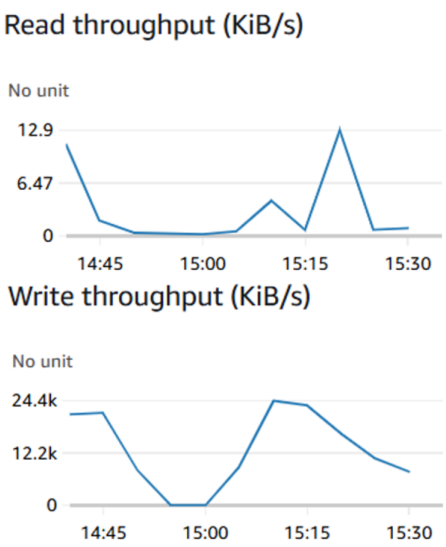
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

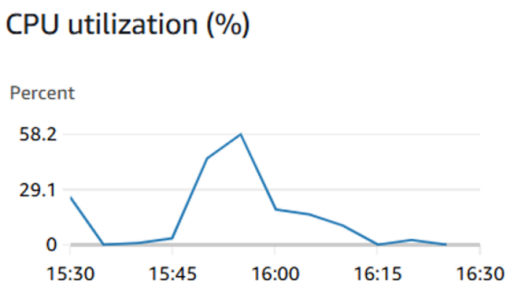
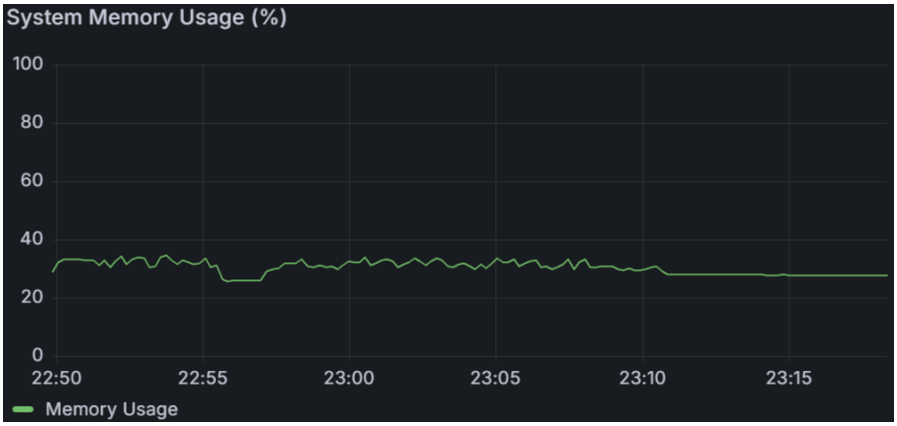
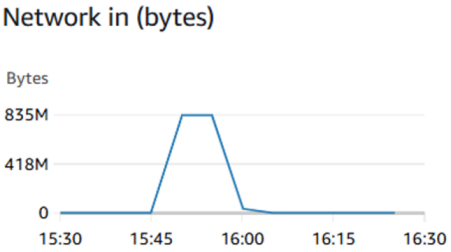
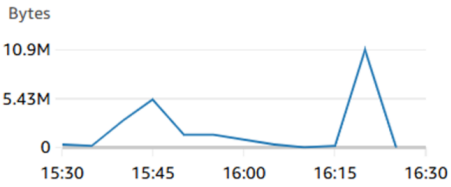
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

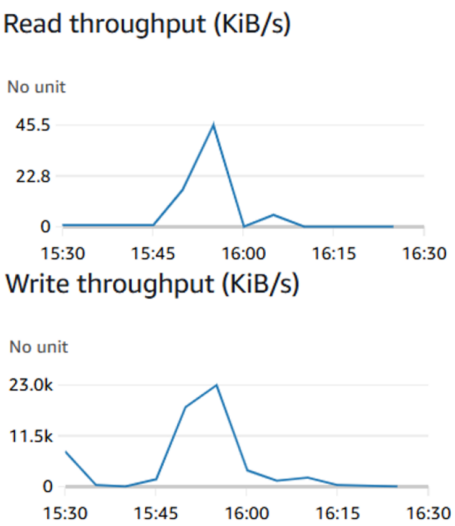
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

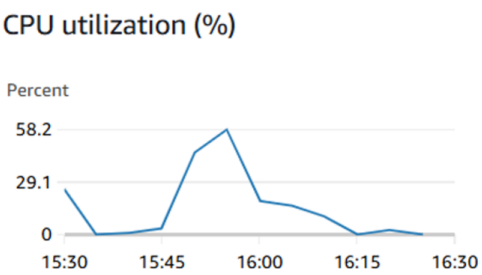
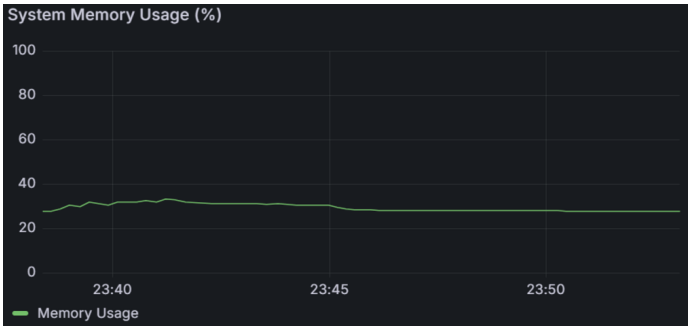
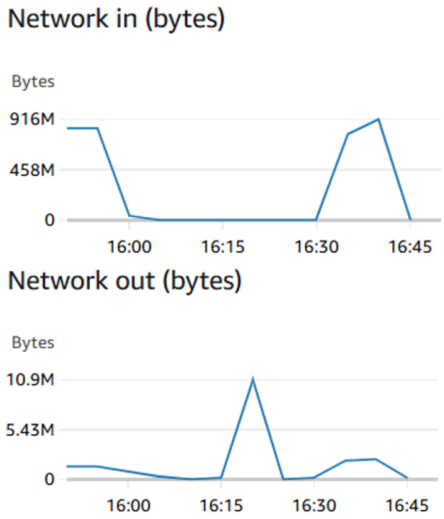
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	 

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability:

	Utilization charts
CPU usage	
Memory usage	
Network usage	

Utilization charts

Disk usage



Standalone DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Ubuntu 22.04	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.57-236

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

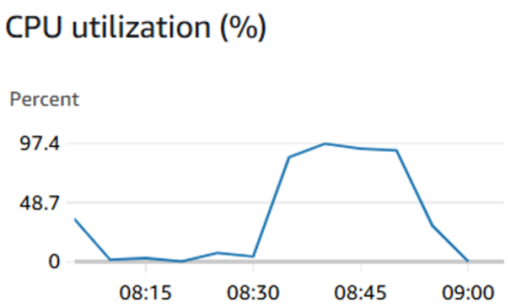
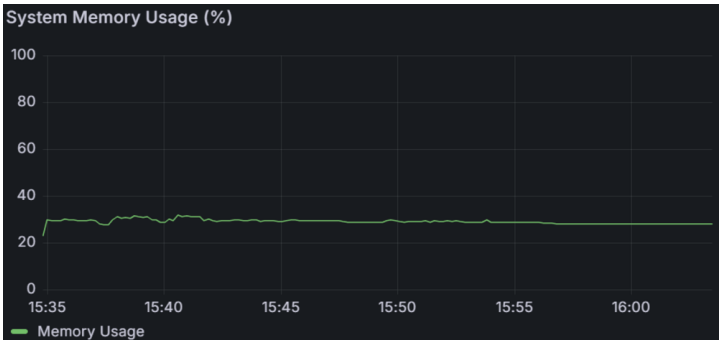
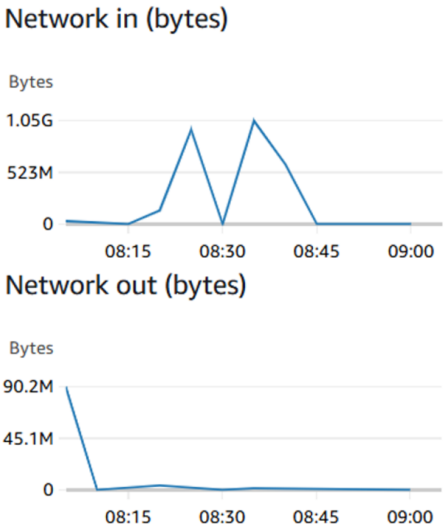
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV)	30.2	69 / 98	29 / 32		252,666	0.0142
Deep CDR	19.4	48 / 62	36 / 49		393,325	0.009
Proactive DLP	16.5	34 / 39	32 / 48		462,455	0.008
Vulnerability	12	50 / 65	46 / 46		635,875	0.006

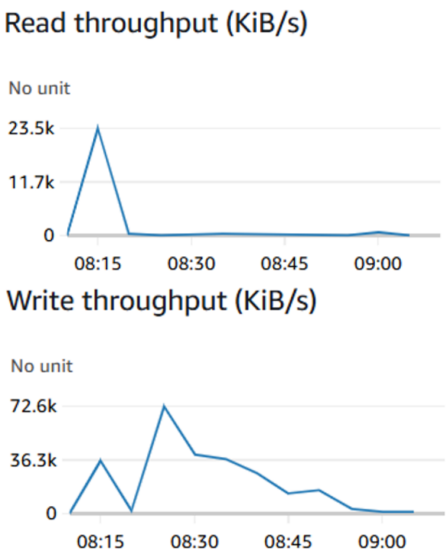
MetaDefender Core with common engine packages:

Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV) + Deep CDR	39.8	78 / 99	62 / 73		191,721	0.0188
Metascan (10 AV) + Deep CDR + Proactive DLP	43.5	61 / 84	47 / 67		175,414	0.0205
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	44.8	62 / 88	69 / 83		170,324	0.0211

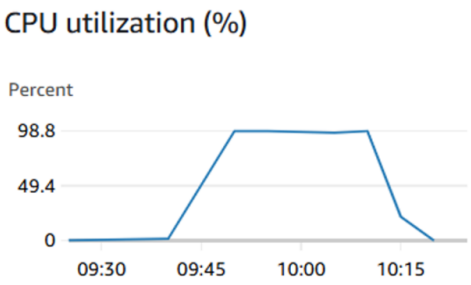
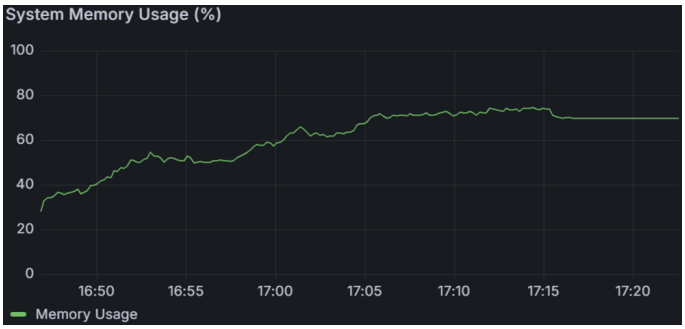
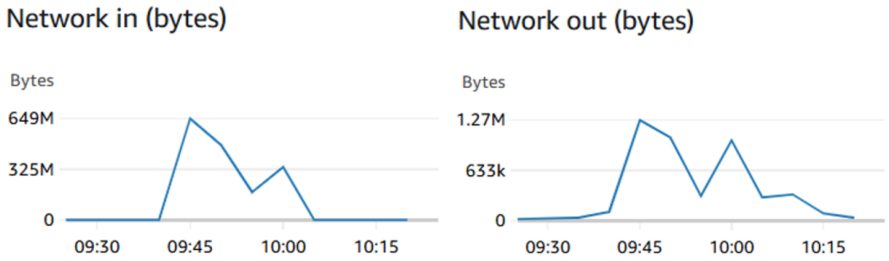
System resource utilization charts

MetaDefender Core with Metascan (10 AV engines) :

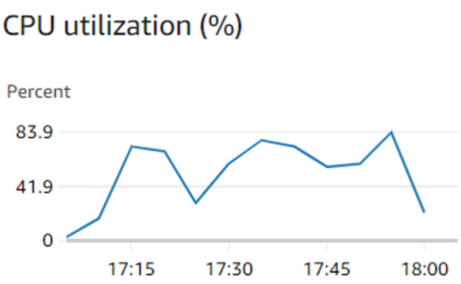
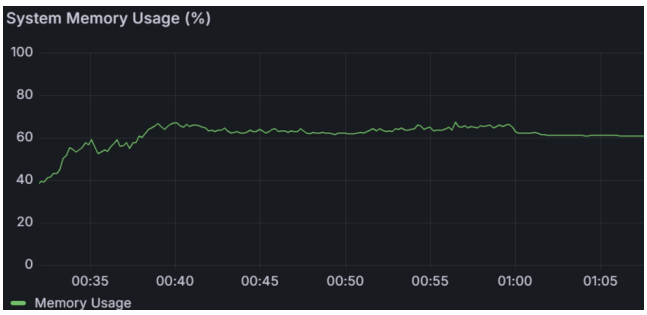
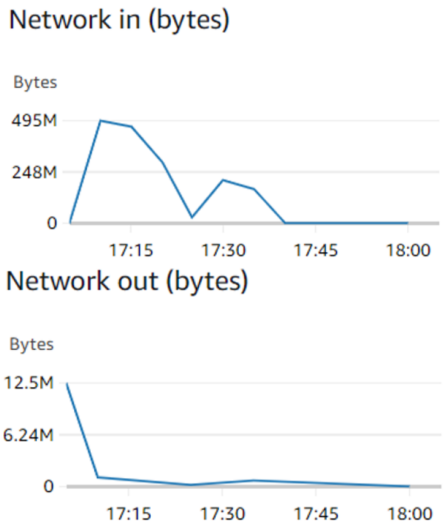
	Utilization charts
CPU usage	
Memory usage	
Network usage	

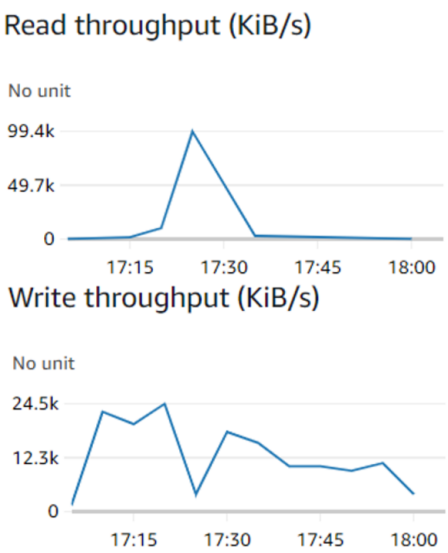
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR :

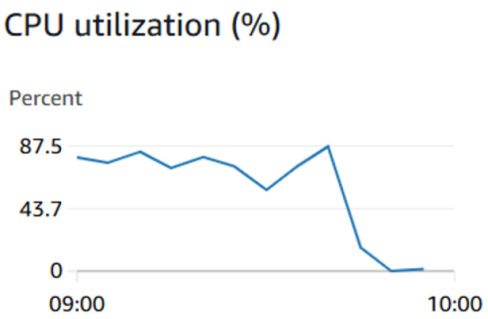
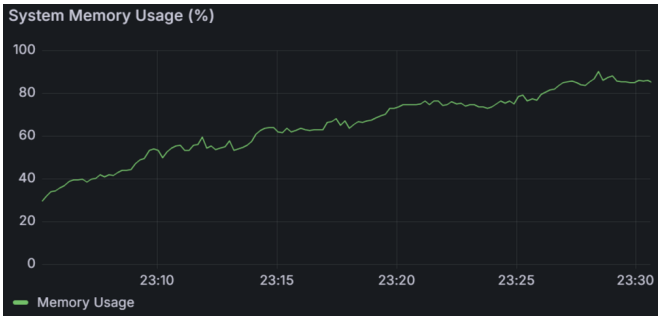
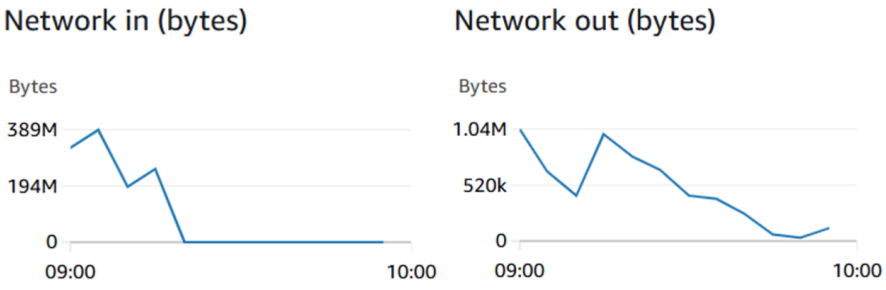
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP :

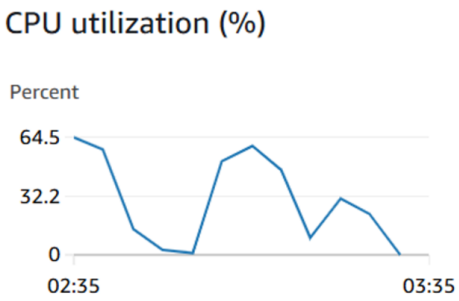
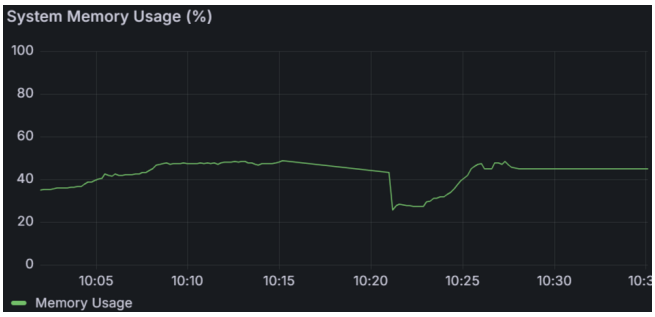
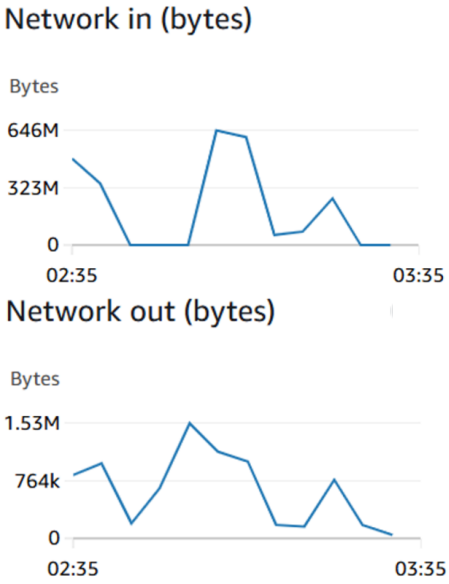
	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP + Vulnerability :

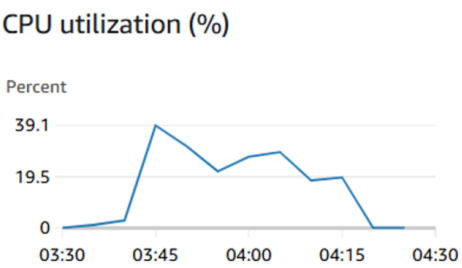
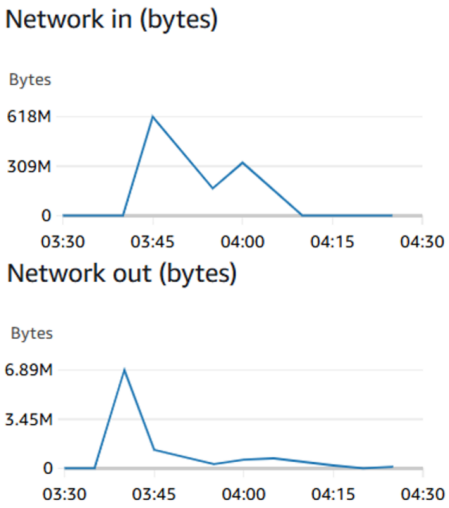
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

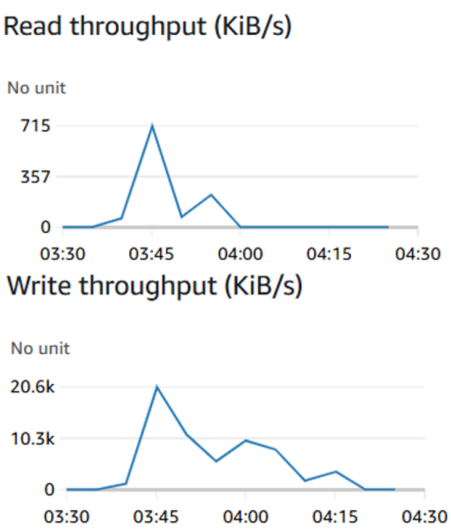
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

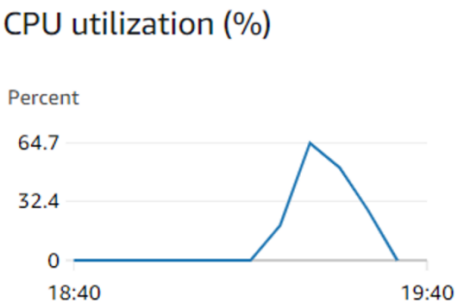
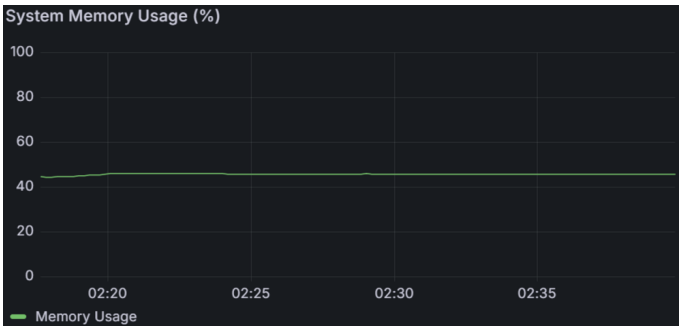
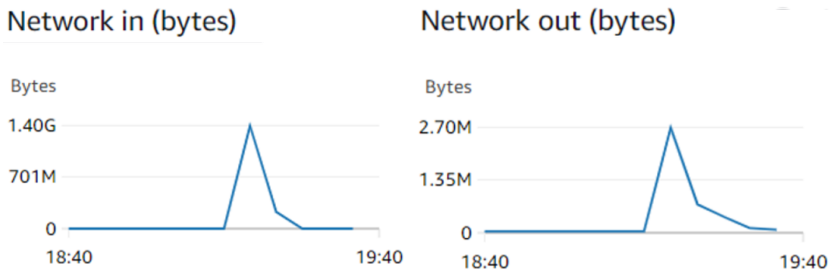
	Utilization charts
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

MetaDefender Core with Vulnerability :

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

Shared DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

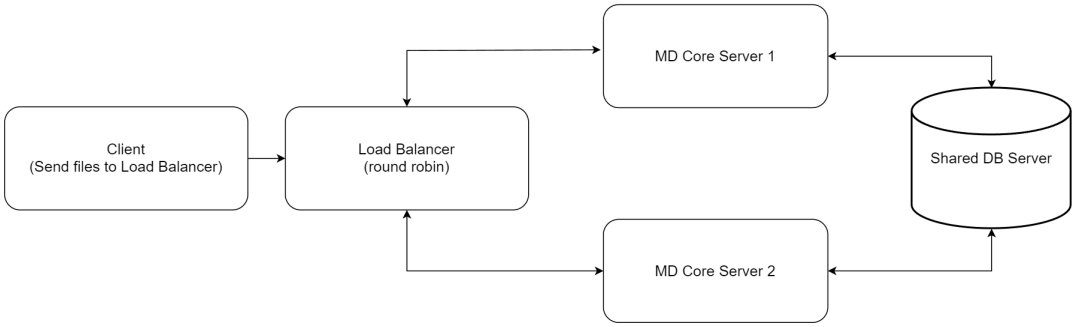
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV)	11.9	37.0	41.9	30.1	30.4	Receive: 6441.1 Send: 769.6	Receive: 6200.2 Send: 693.7	640,821.2	0.0
Deep CDR	10.5	60.1	66.7	30.3	32.2	Receive: 6379.8 Send: 594.8	Receive: 7146.1 Send: 609.6	729,368.5	0.0
Proactive DLP	7.8	38.7	37.5	29.7	31.0	Receive: 8842.6 Send: 791.8	Receive: 8635.6 Send: 729.1	965,455.7	0.0
Vulnerability	4.4	52.2	53.0	26.0	26.9	Receive: 16,215.5 Send: 1006.8	Receive: 15,785.7 Send: 997.2	1714,449.3	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV) + Deep CDR	15.4	72.4	75	34.3	35.4	Receive: 5436.6 Send: 611.3	Receive: 5589.3 Send: 607.2	495,697.4	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP	16.9	77.2	80.2	36.4	37.9	Receive: 5254 Send: 585.4	Receive: 5041.3 Send: 576.1	450,657.6	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	18.0	72.1	80.4	38.6	38.0	Receive: 4504.3 Send: 574.1	Receive: 4649.5 Send: 571.1	423,010	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

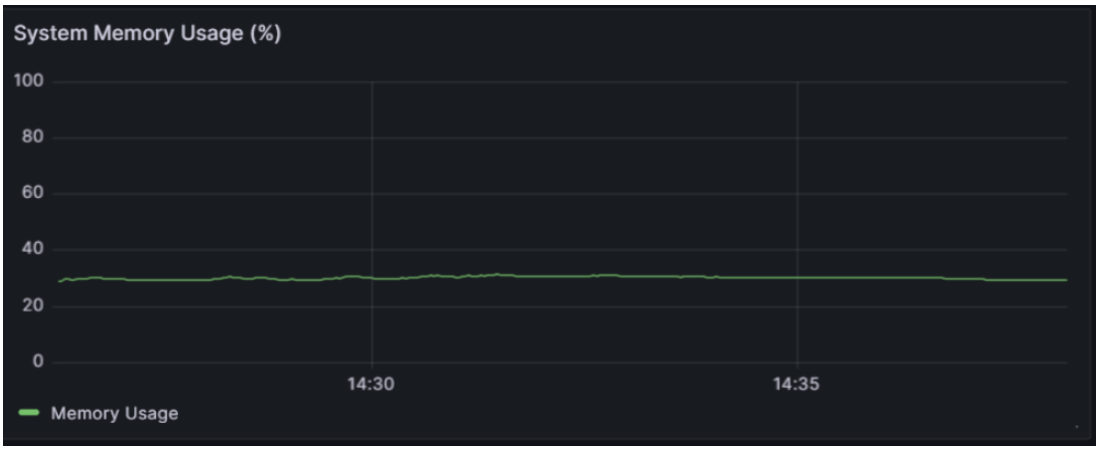
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2

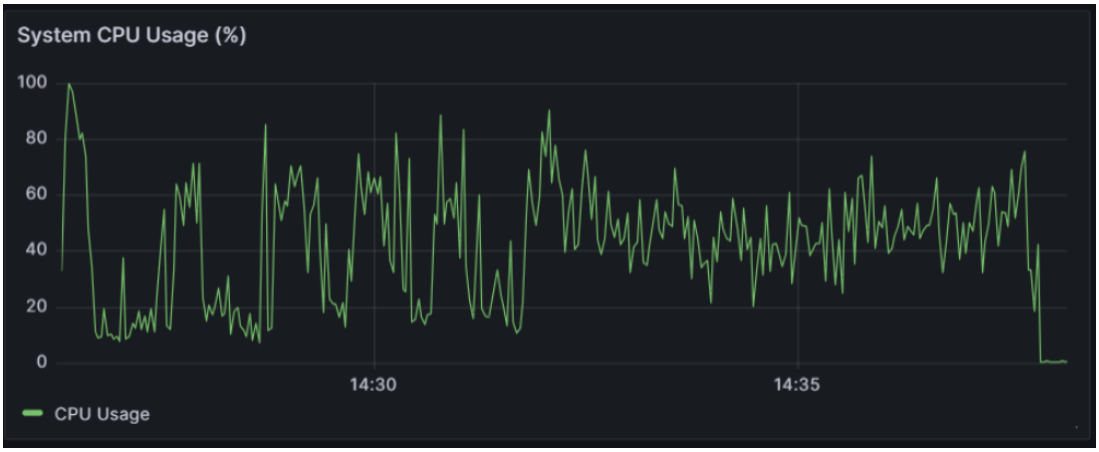


CPU usage

MD Core #1

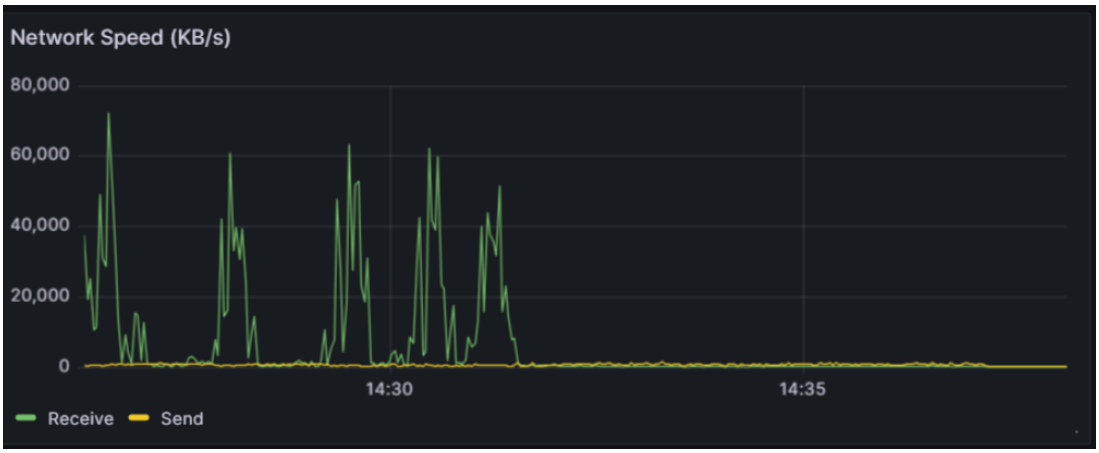


MD Core #2

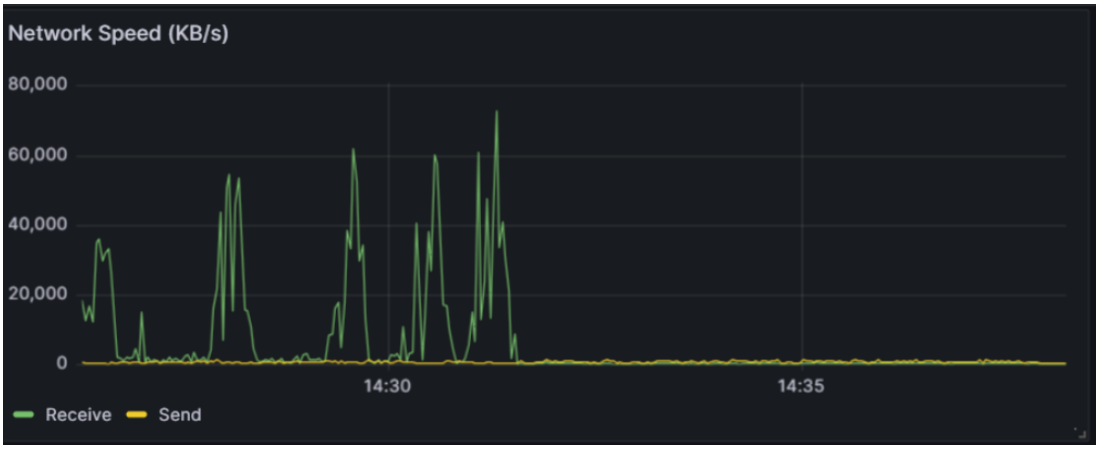


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2

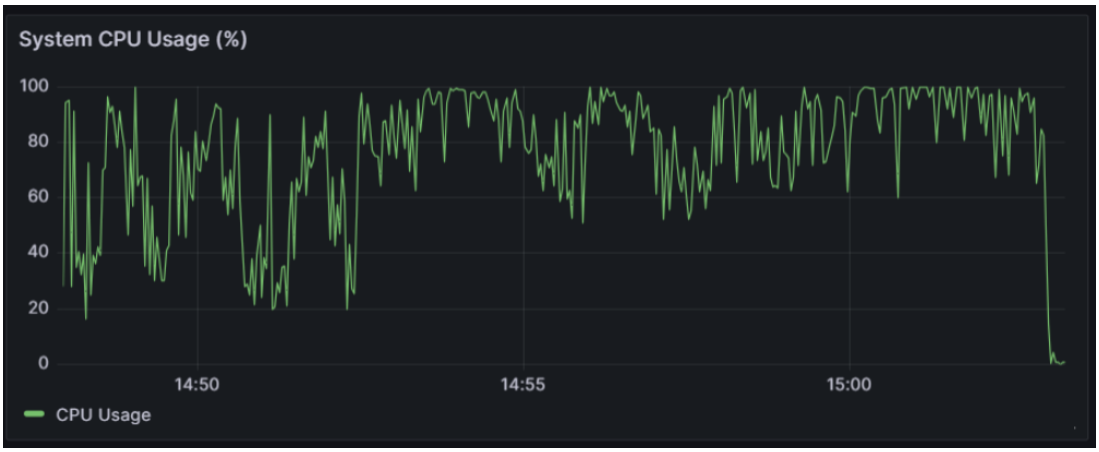


CPU usage

MD Core #1

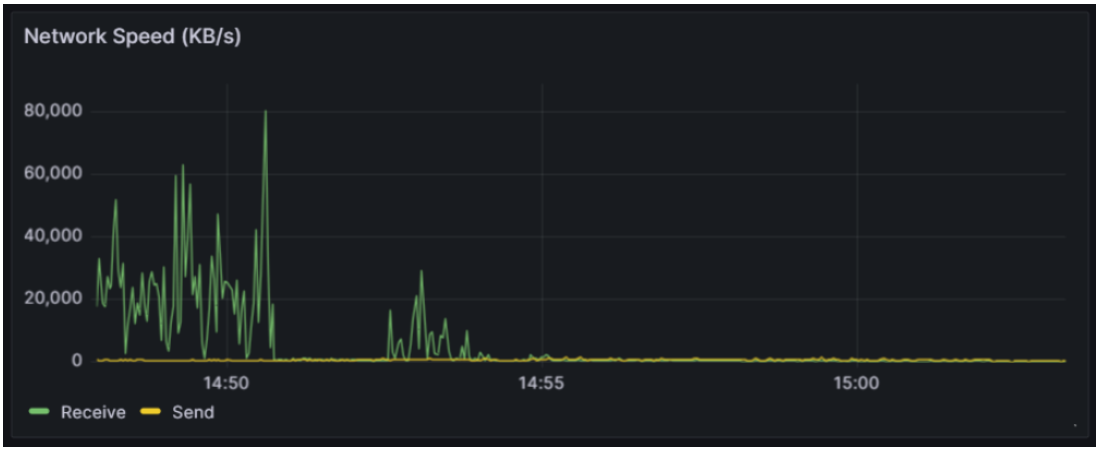


MD Core #2



Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

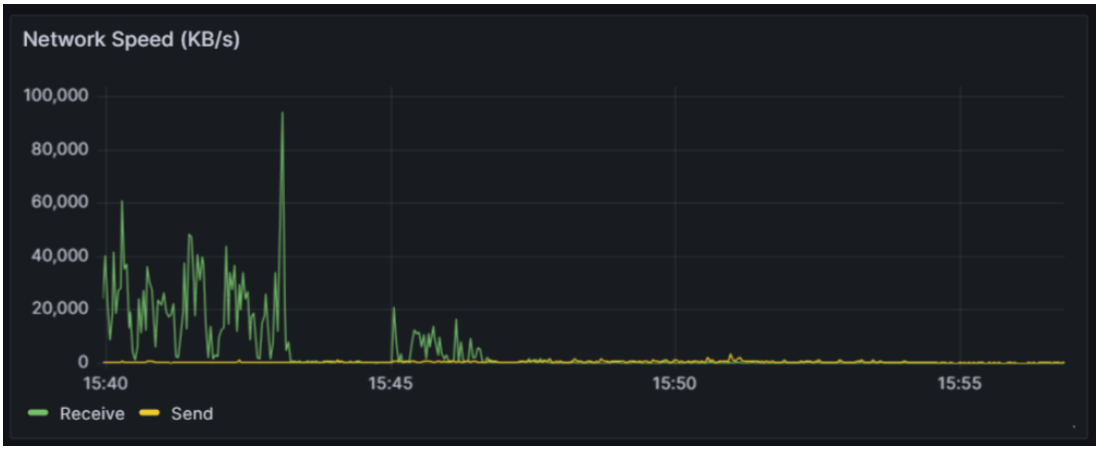


MD Core #2

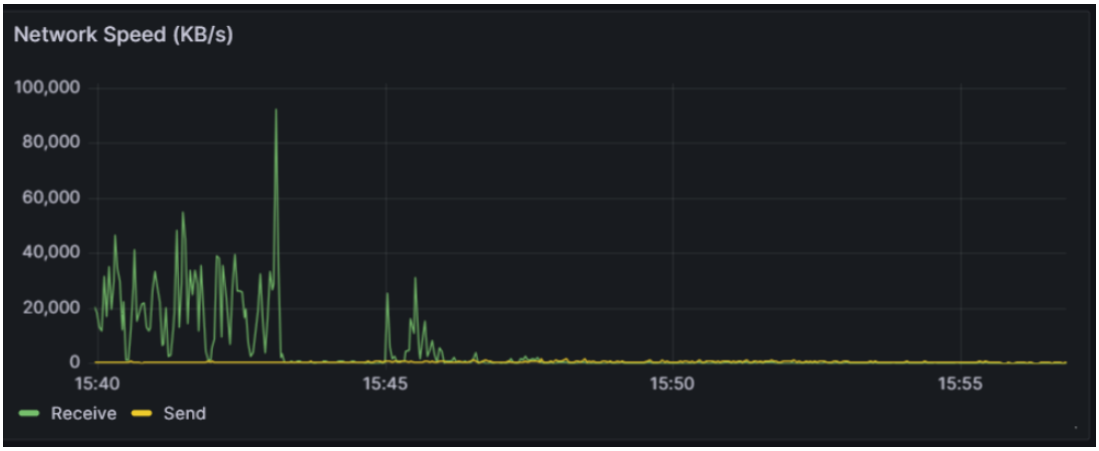


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1

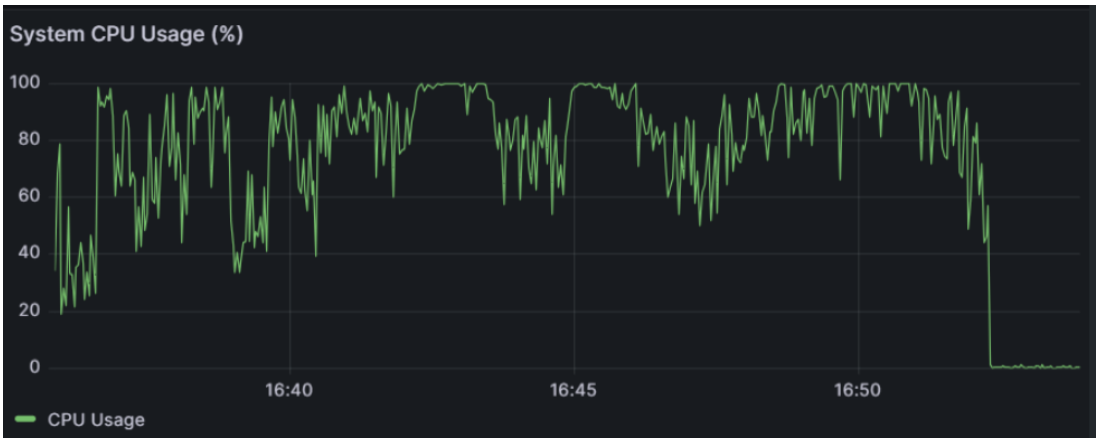


MD Core #2



CPU usage

MD Core #1

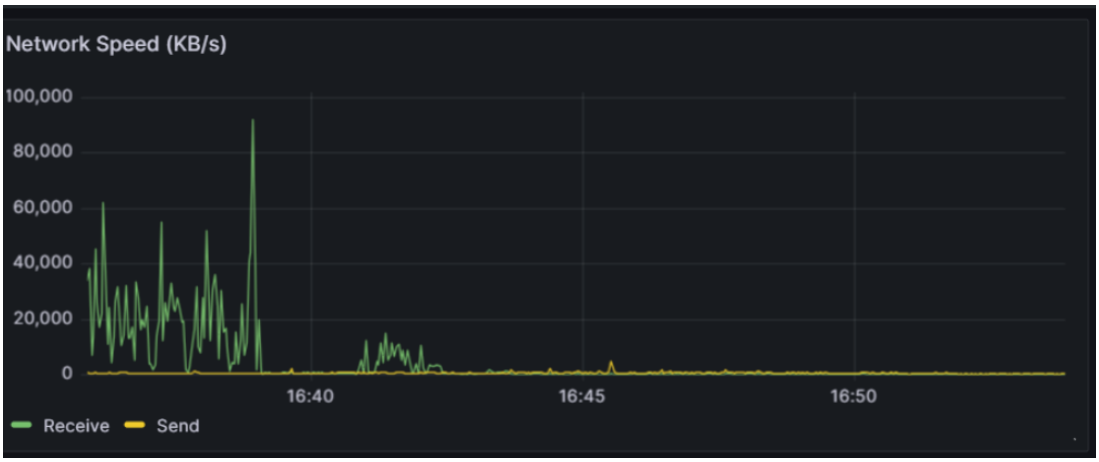


MD Core #2

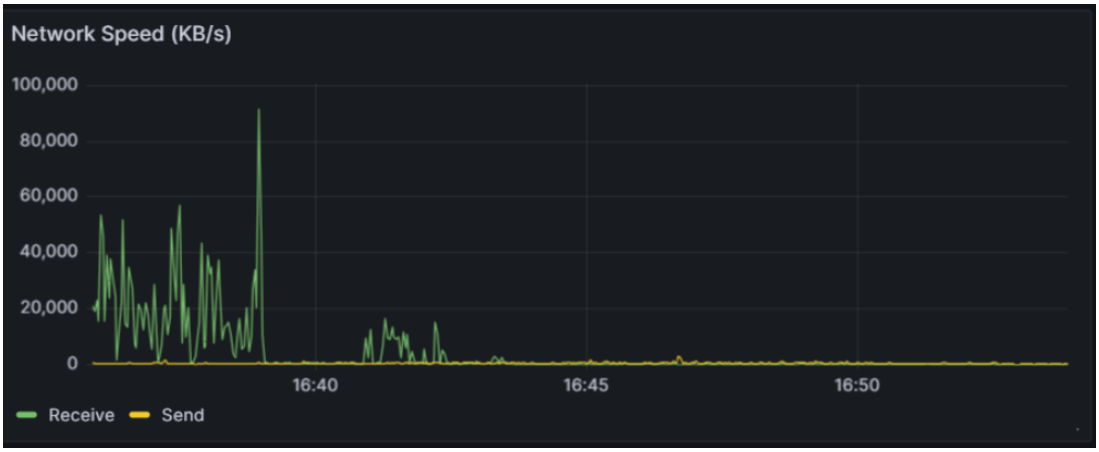


Network usage

MD Core #1



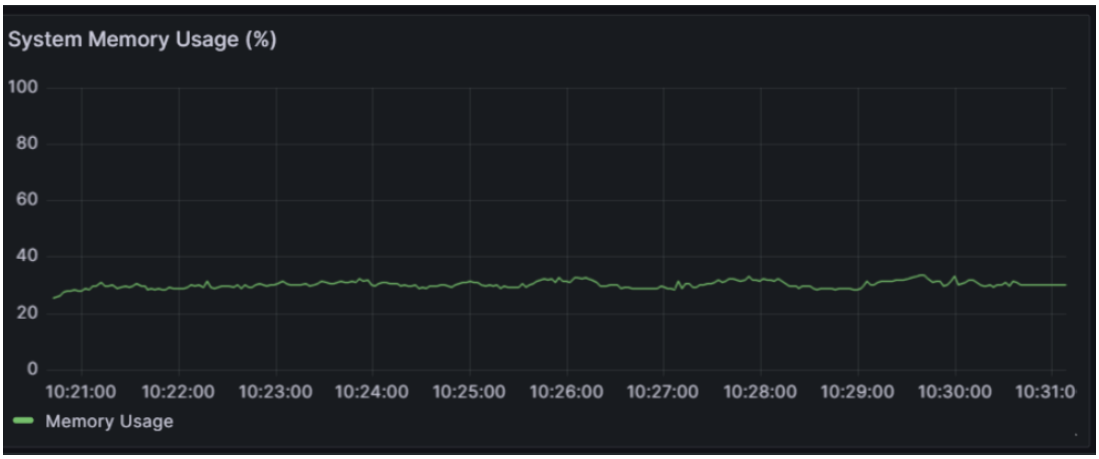
MD Core #2



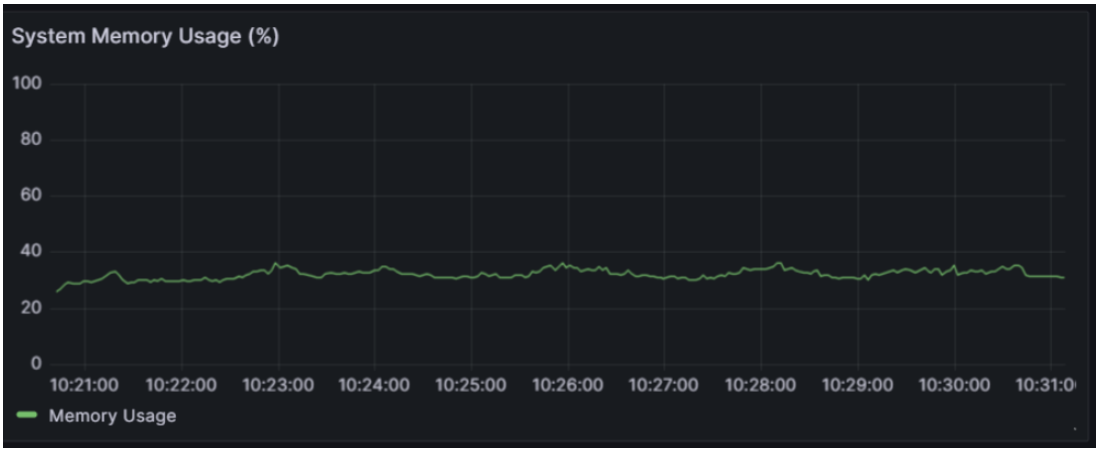
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

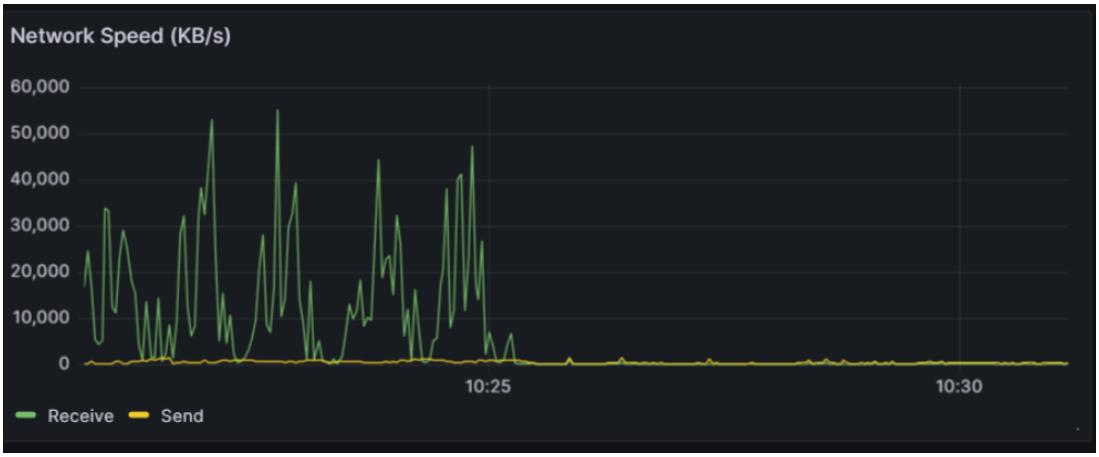


MD Core #2

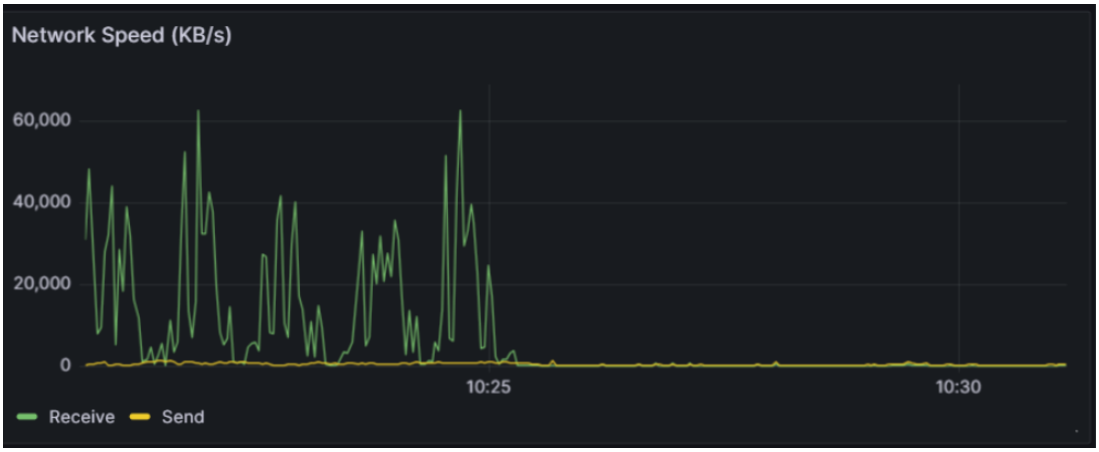


Network usage

MD Core #1



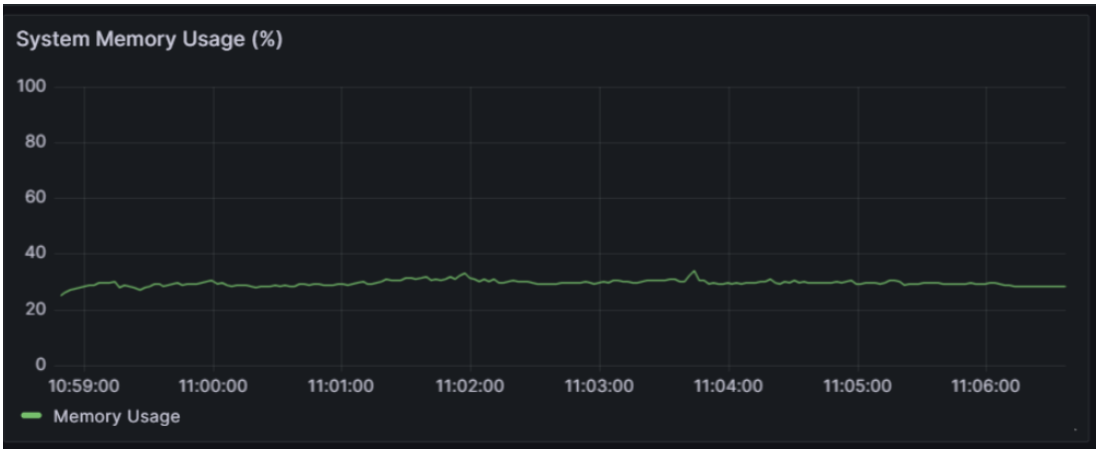
MD Core #2



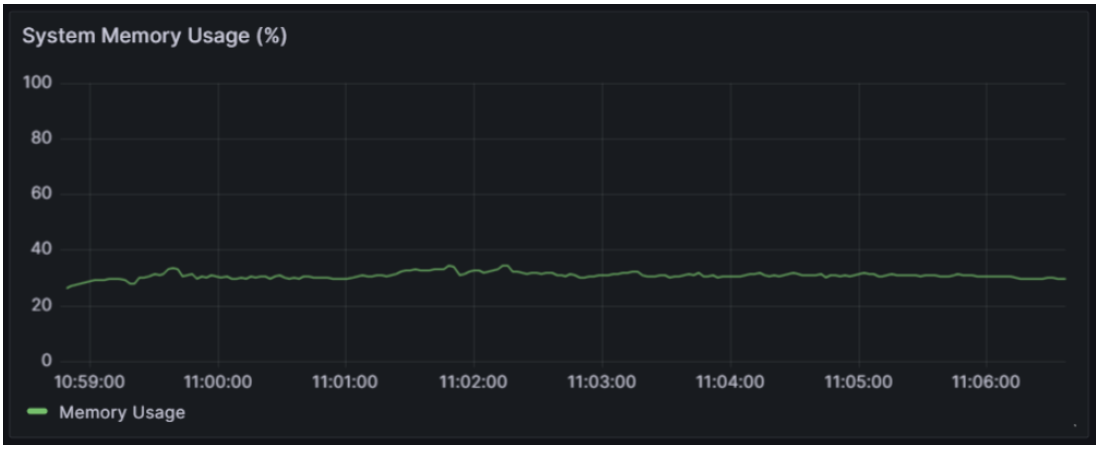
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

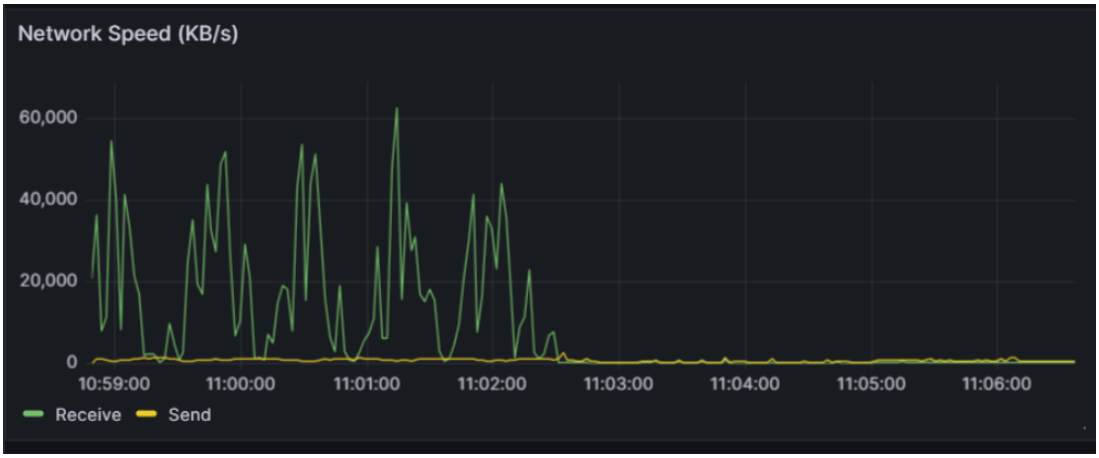


MD Core #2

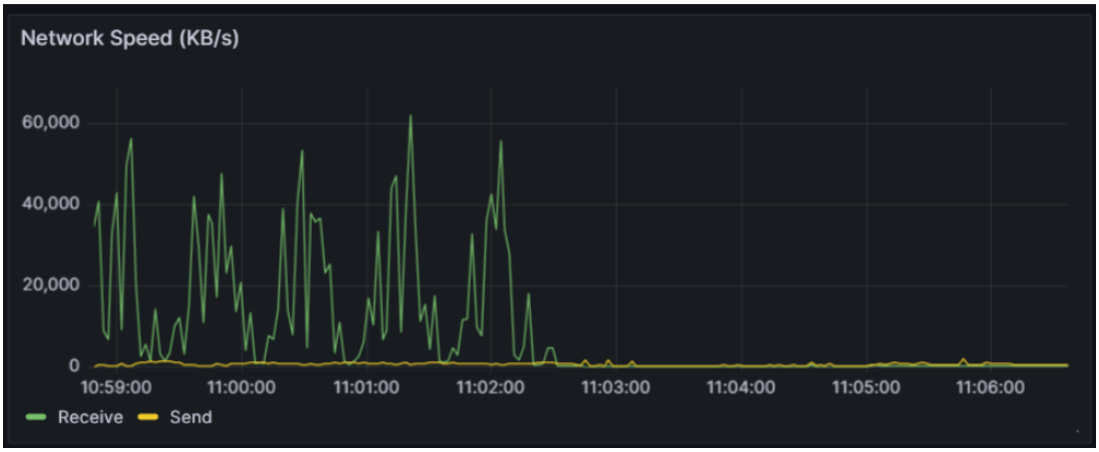


Network usage

MD Core #1



MD Core #2



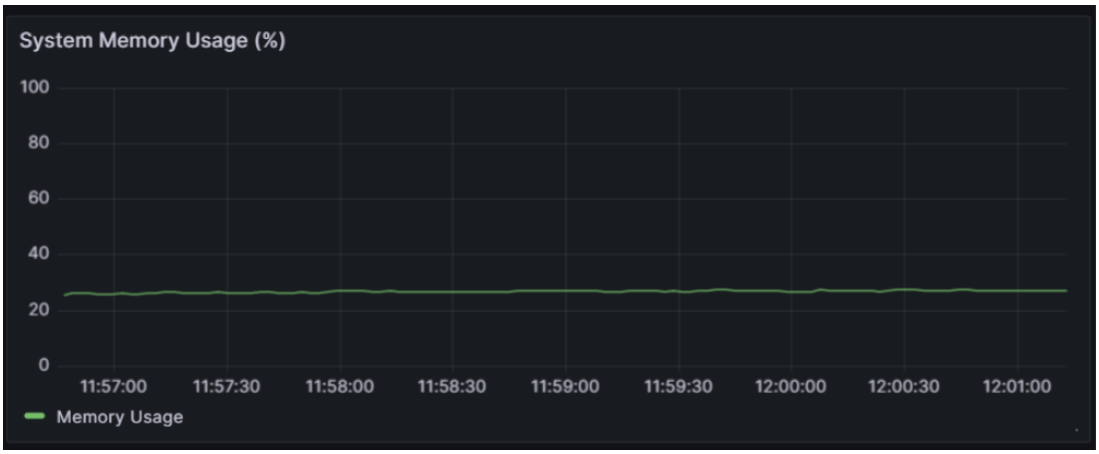
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

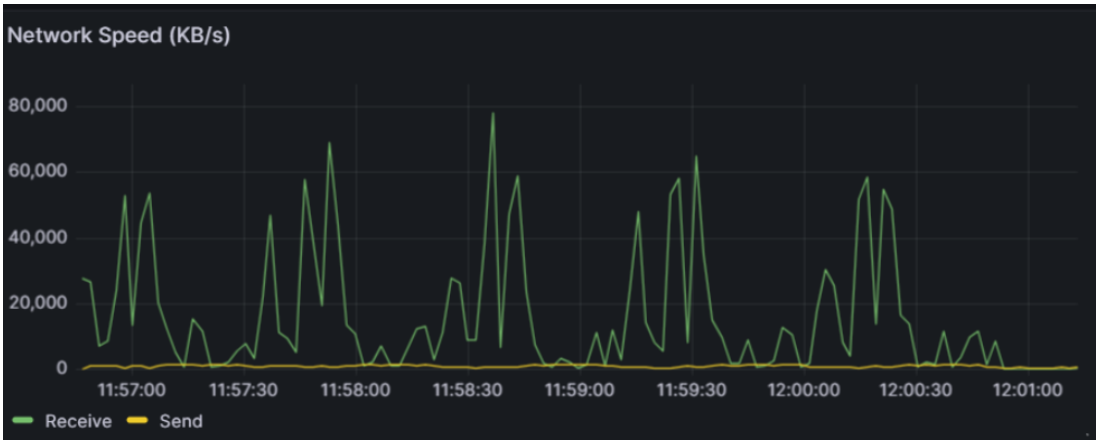


MD Core #2

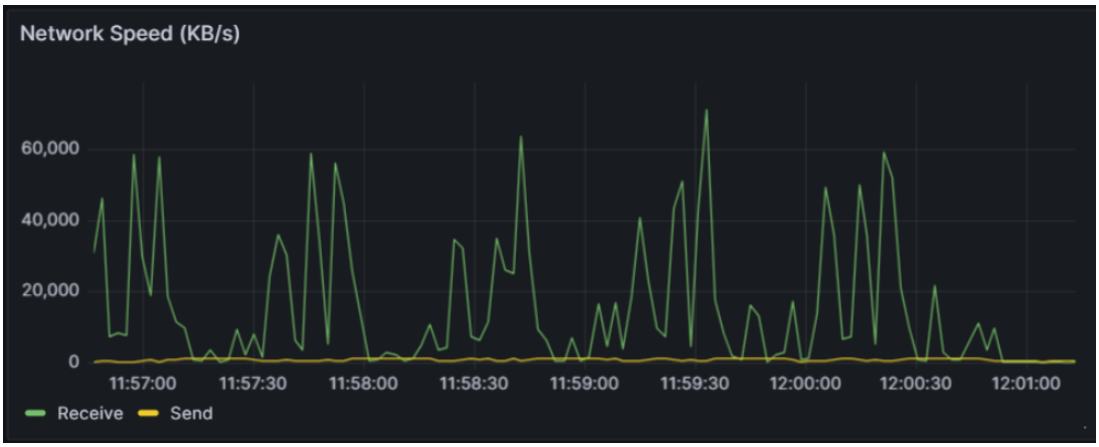


Network usage

MD Core #1



MD Core #2



Shared DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

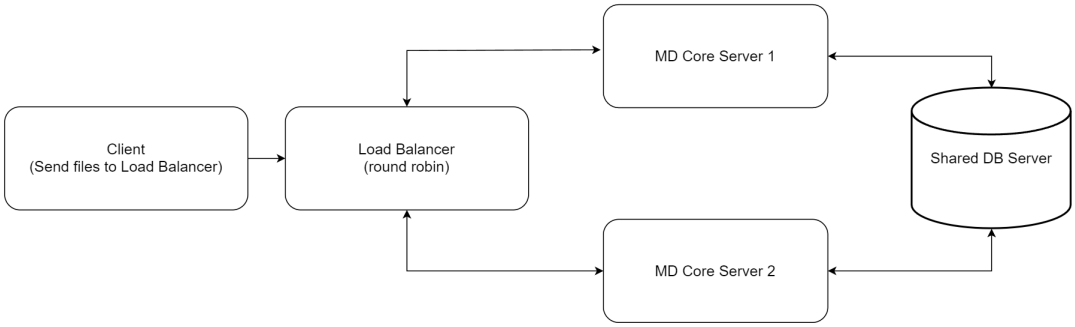
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV)	15.5	63.1	73.3	25.2	25.8	Receive: 4555.9 Send: 694.4	Receive: 4516.2 Send: 692.4	492,620.5	0.0
Deep CDR	9.70	50.7	51.2	31.9	32.5	Receive: 7062.9 Send: 623.2	Receive: 6653 Send: 654.1	786,189.6	0.0
Proactive DLP	7.2	35.2	35.0	27.9	27.5	Receive: 9047.4 Send: 802.4	Receive: 9859.8 Send: 787.5	1,055,110.8	0.0
Vulnerability	5.0	45.7	49.3	24.4	24.5	Receive: 13,585.7 Send: 966.7	Receive: 13,505.8 Send: 987.4	1,519,562.1	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV) + Deep CDR	19.9	79.1	77.6	34.0	34.2	Receive: 3785.9 Send: 573.8	Receive: 3663.1 Send: 577.7	382,716	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP	21.6	80.3	81.5	37.7	44.4	Receive: 3305.6 Send: 543.3	Receive: 3414.3 Send: 556.2	353,694.9	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	22.5	84.2	79.8	37.9	44.7	Receive: 3267.9 Send: 554.9	Receive: 3398.2 Send: 553.8	338,507.6	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

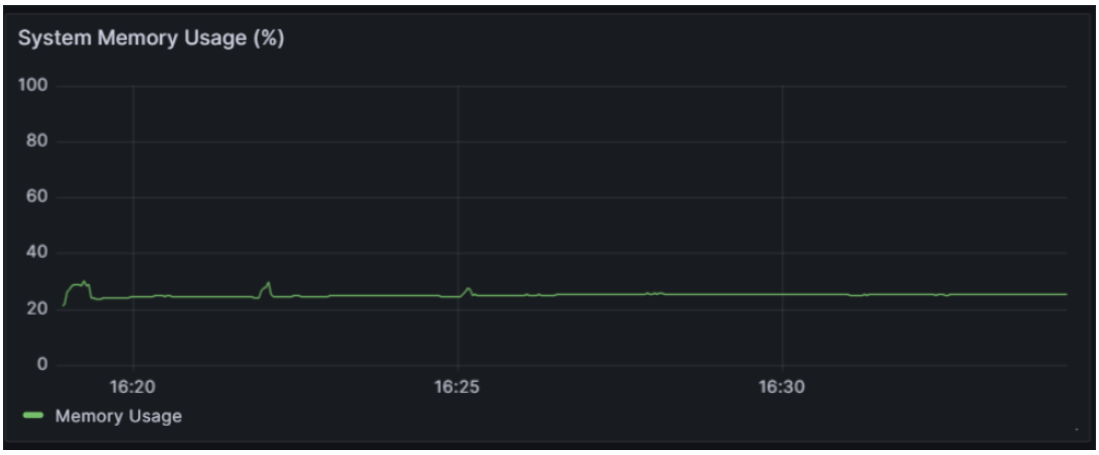
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

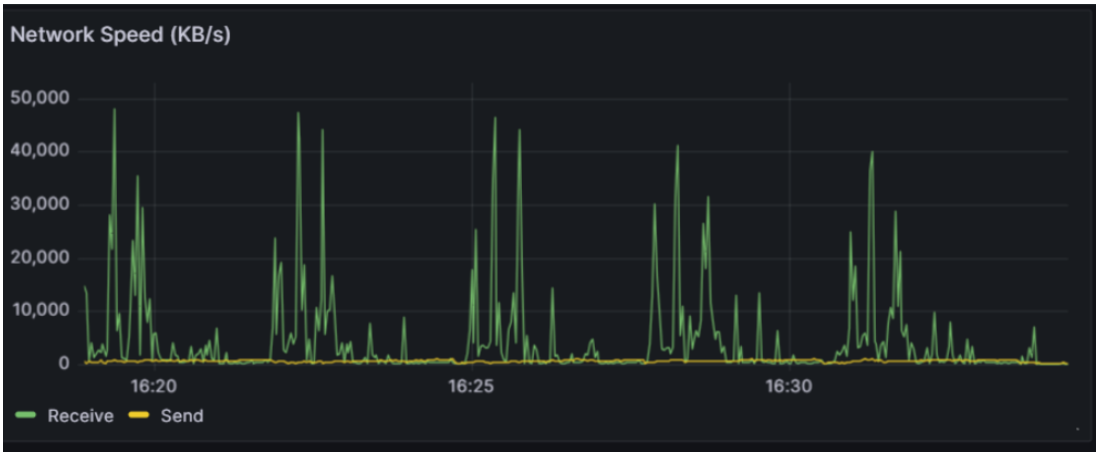


MD Core #2

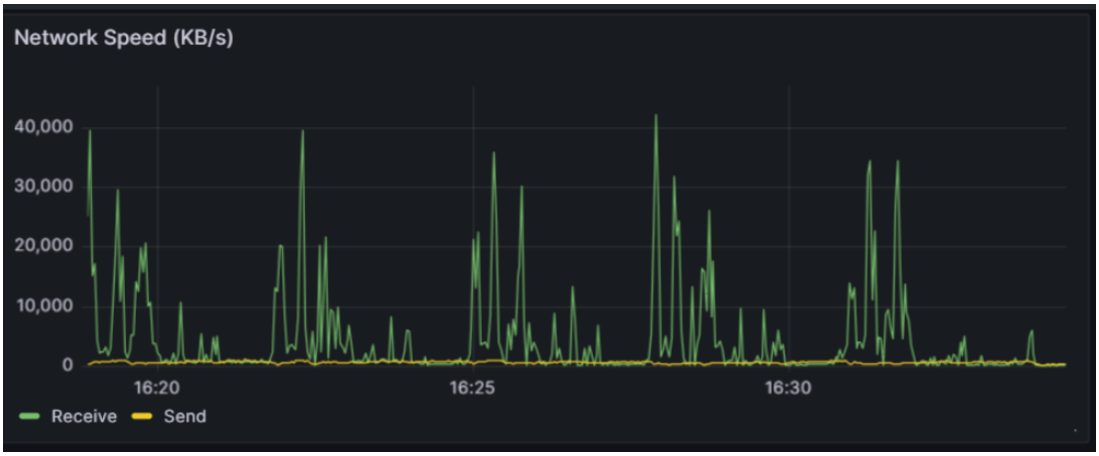


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

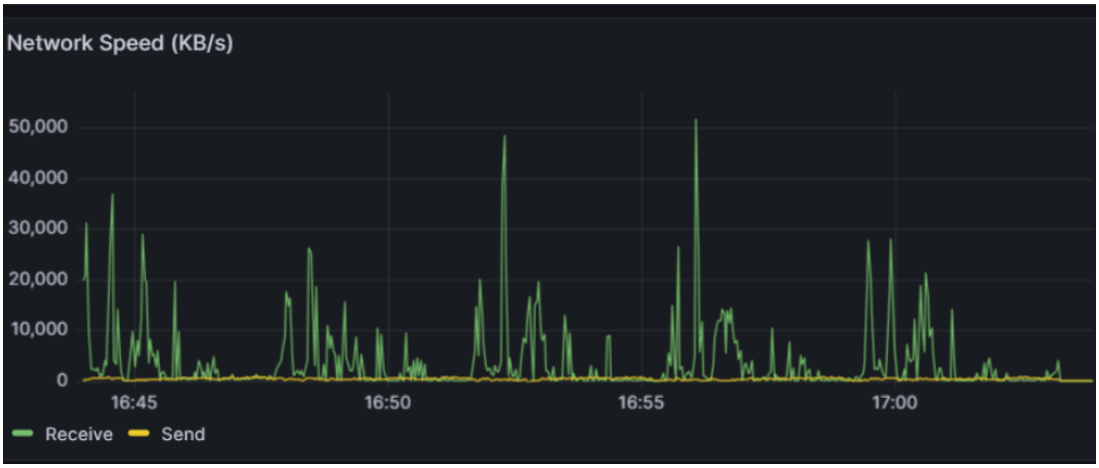


MD Core #2

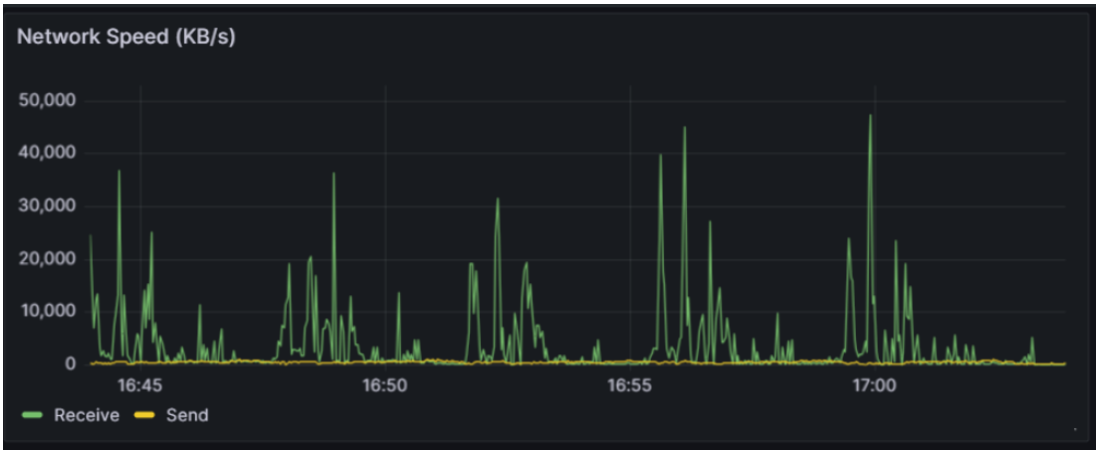


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

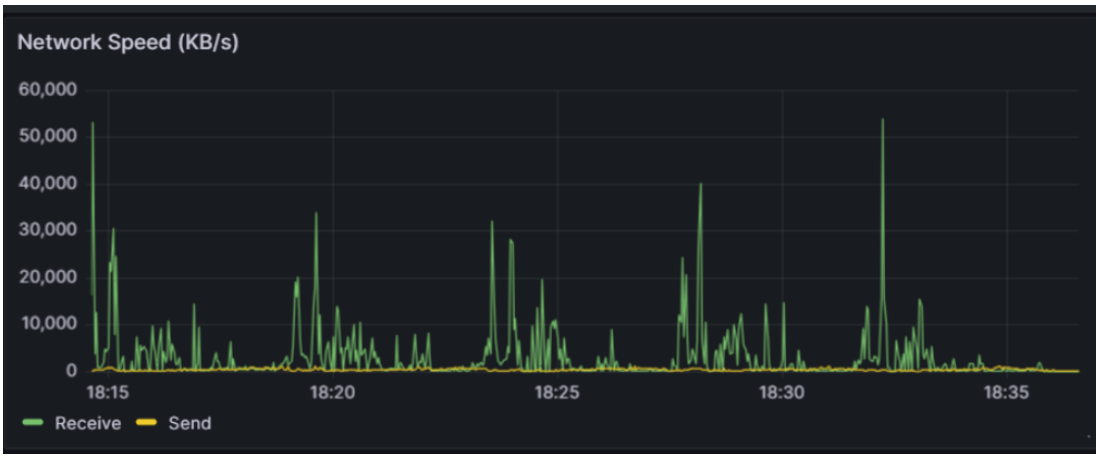


MD Core #2

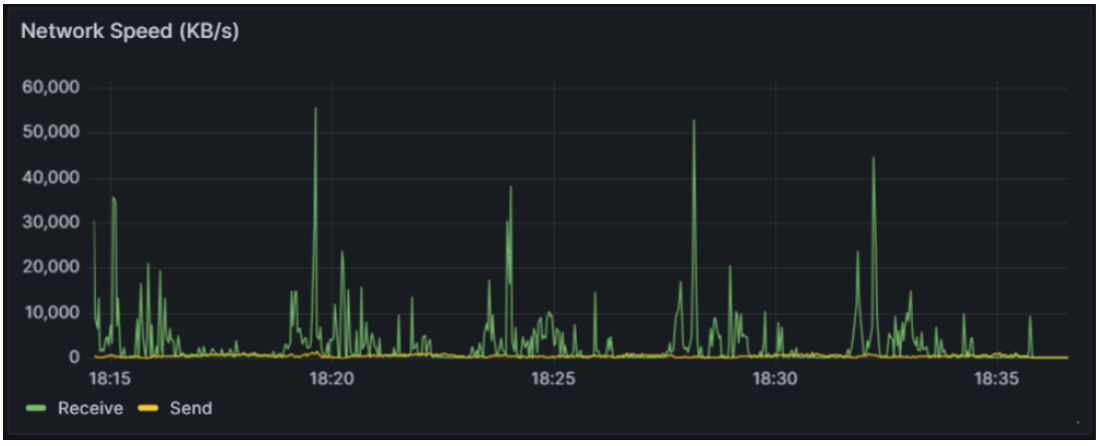


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

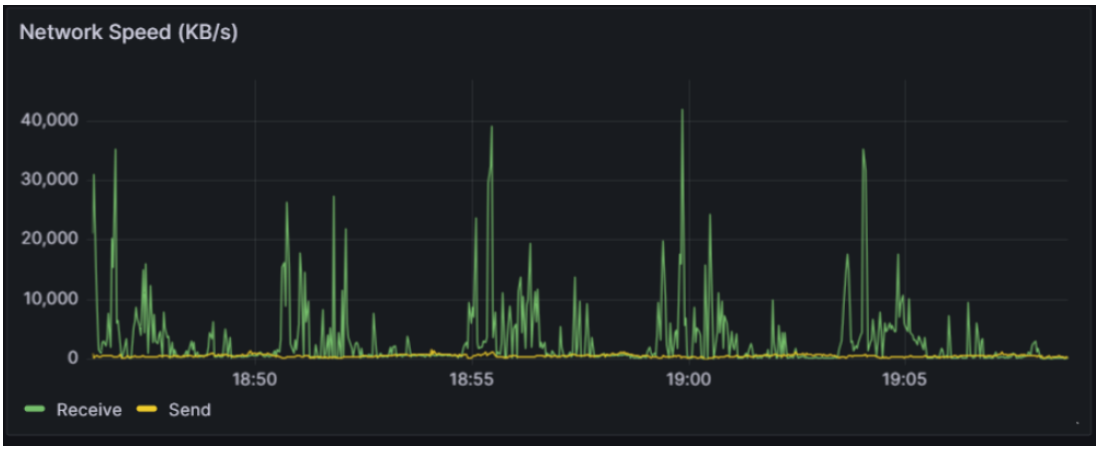


MD Core #2

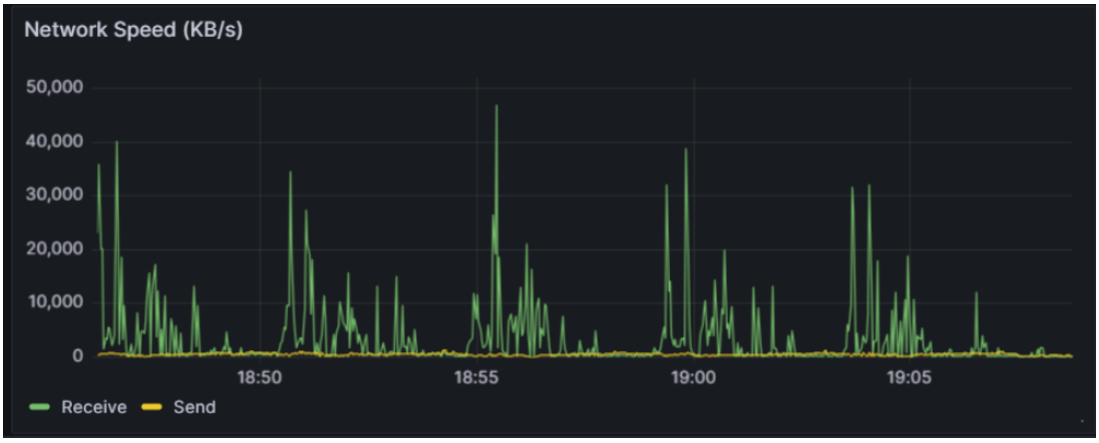


Network usage

MD Core #1



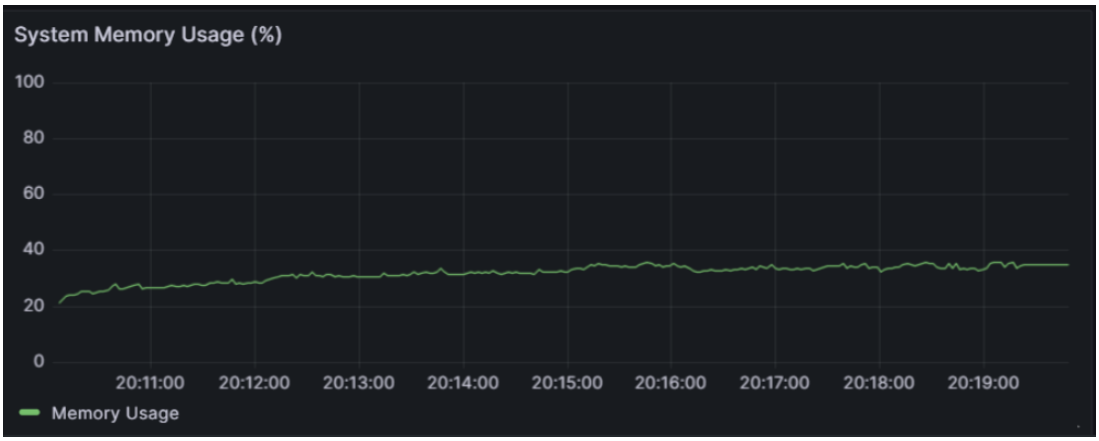
MD Core #2



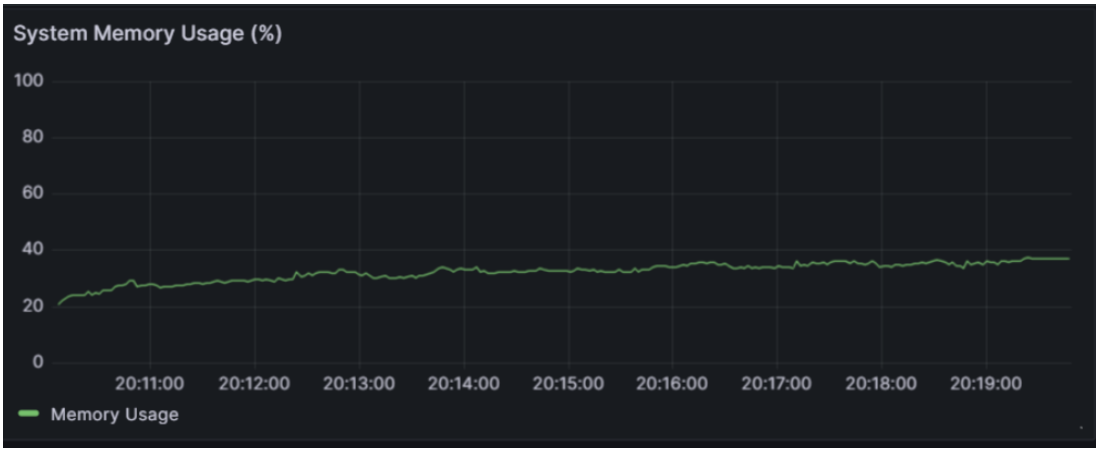
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1

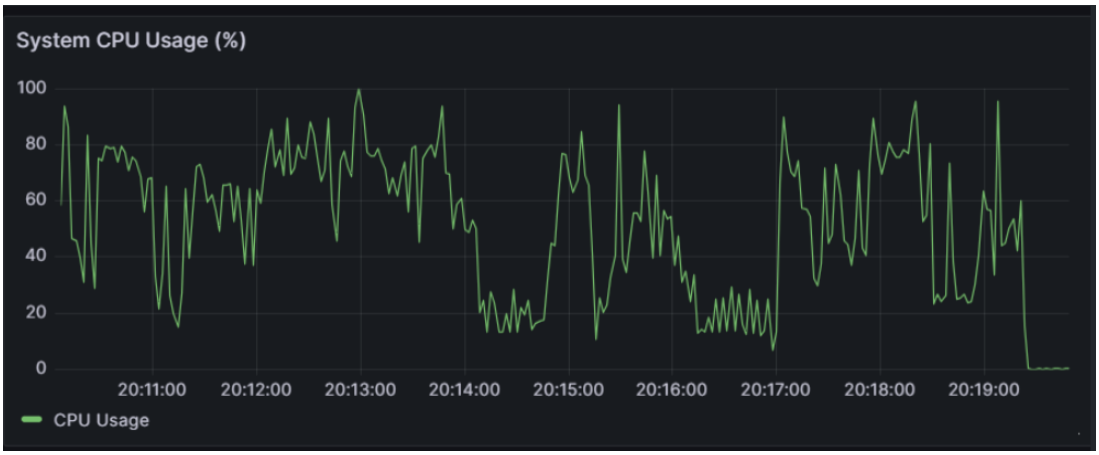


MD Core #2



CPU usage

MD Core #1

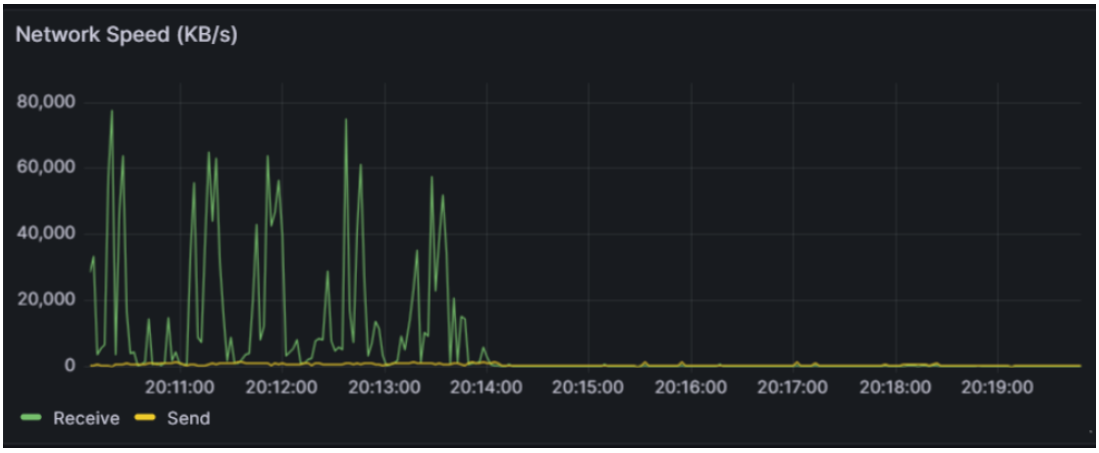


MD Core #2

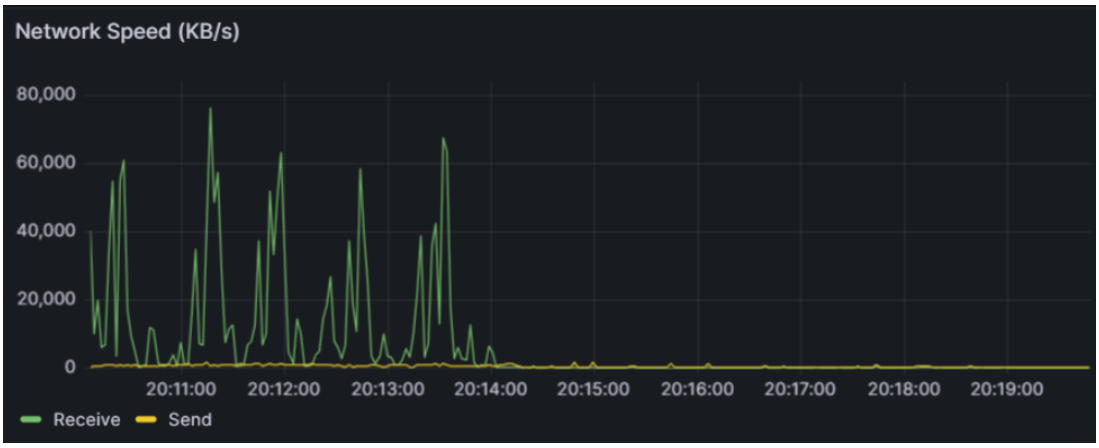


Network usage

MD Core #1



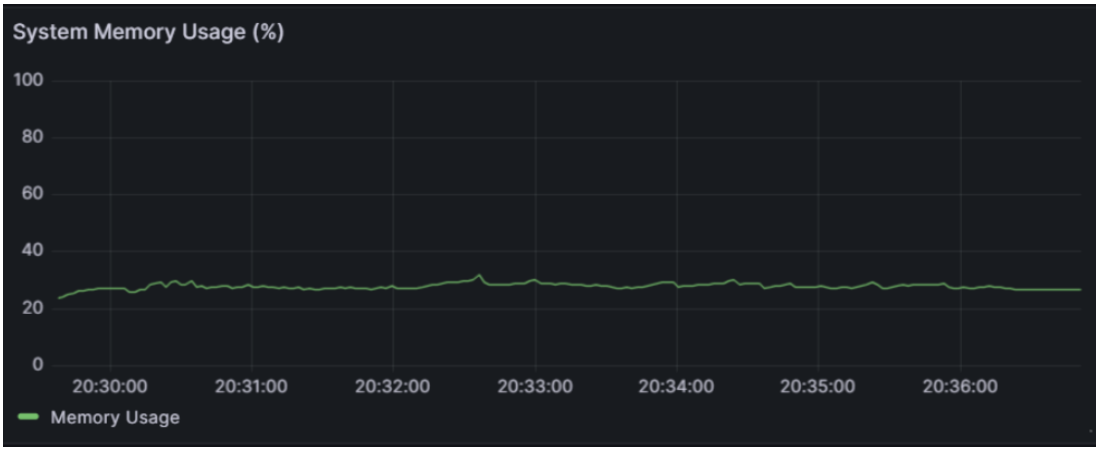
MD Core #2



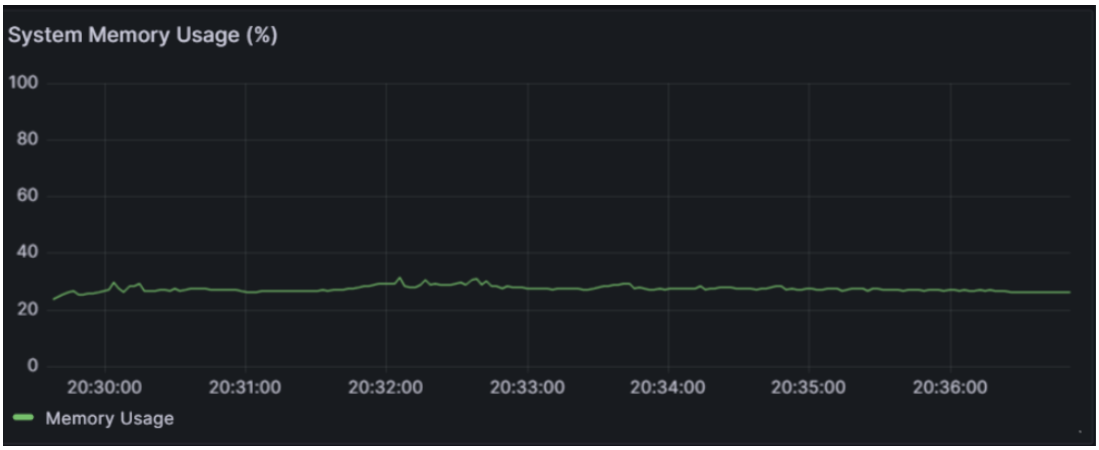
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1

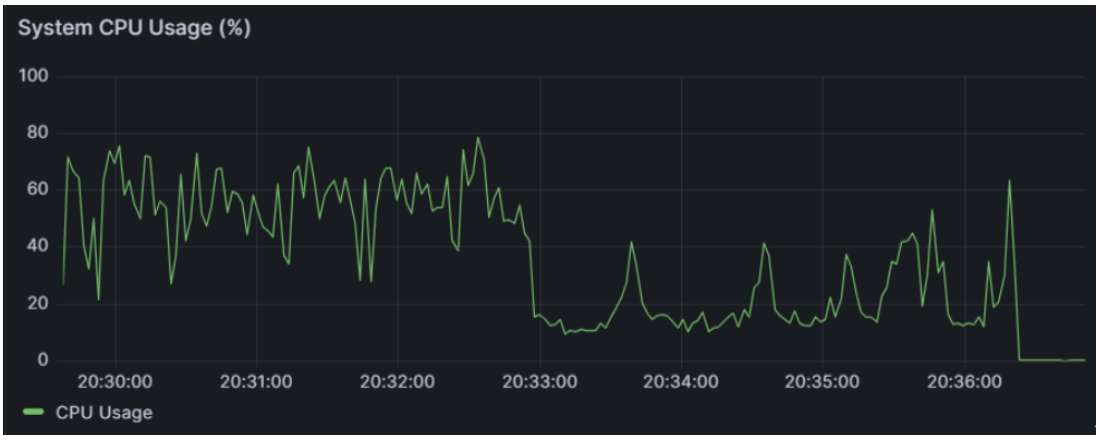


MD Core #2

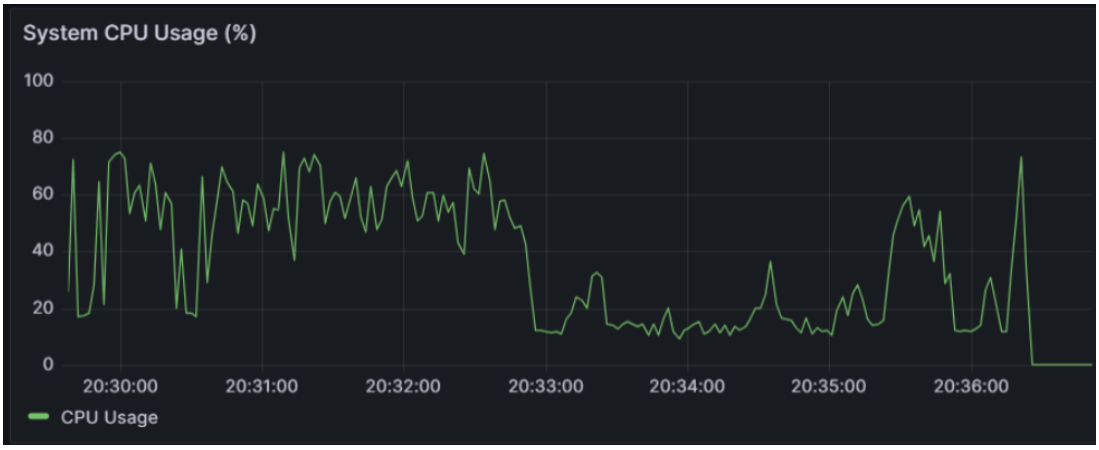


CPU usage

MD Core #1

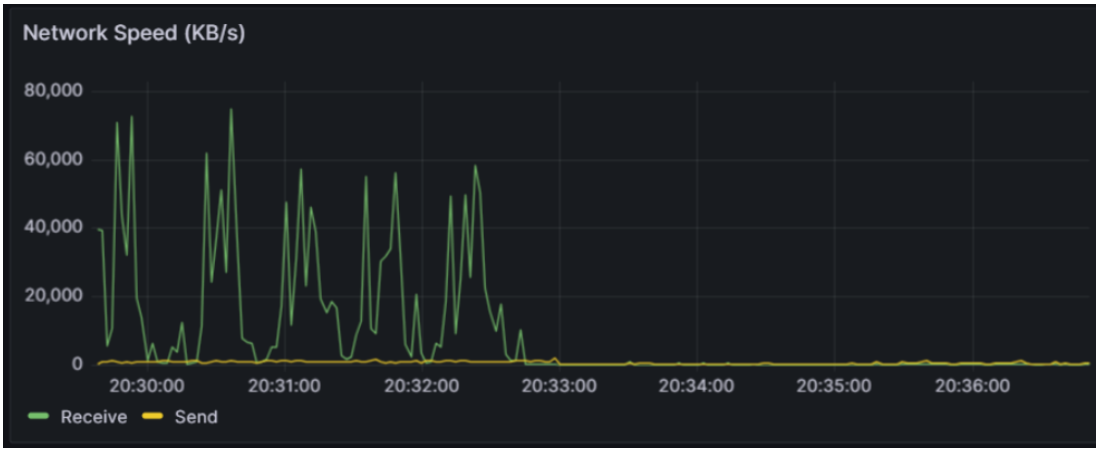


MD Core #2

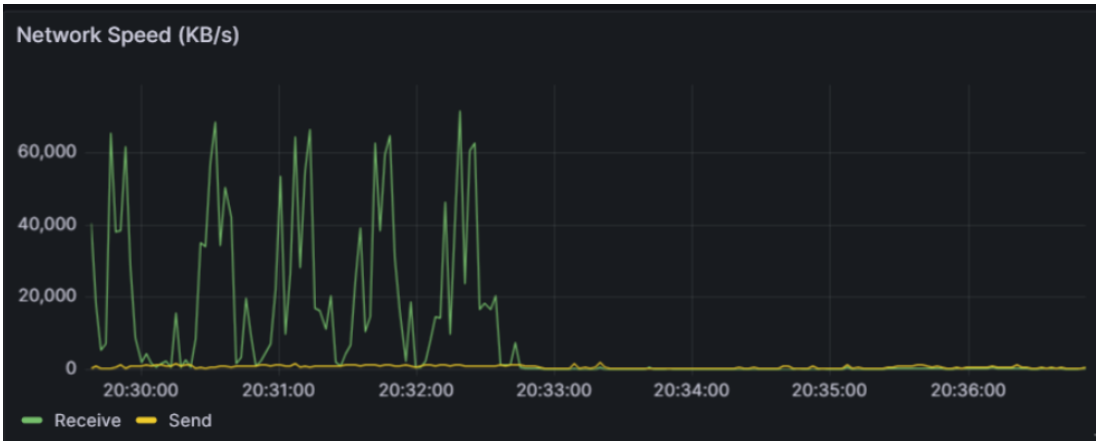


Network usage

MD Core #1



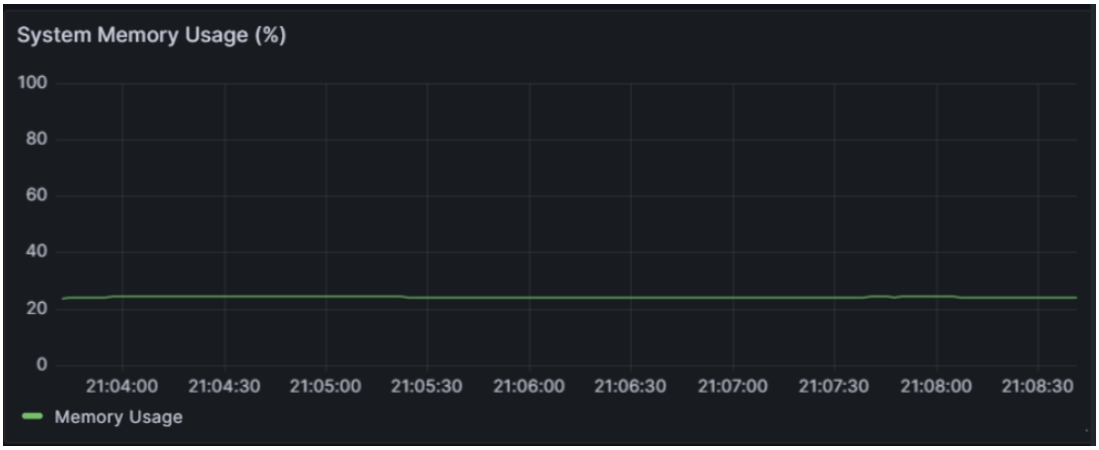
MD Core #2



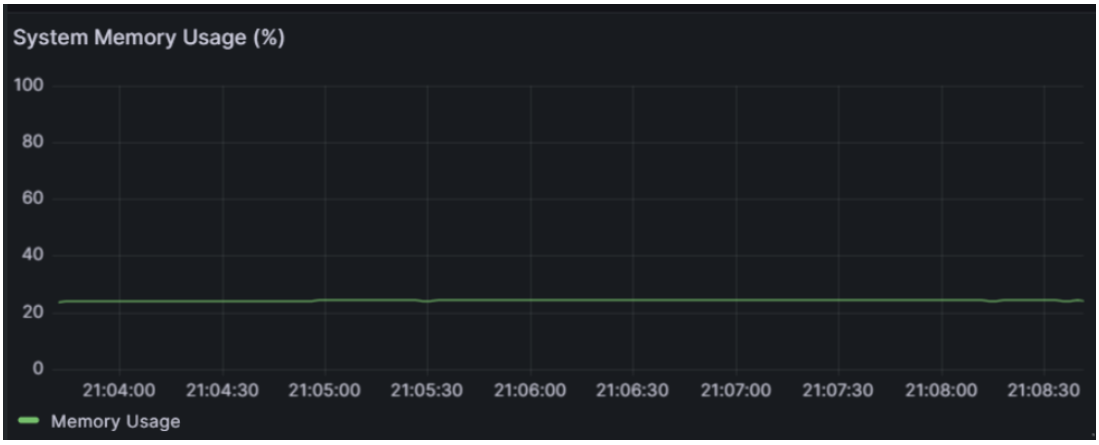
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

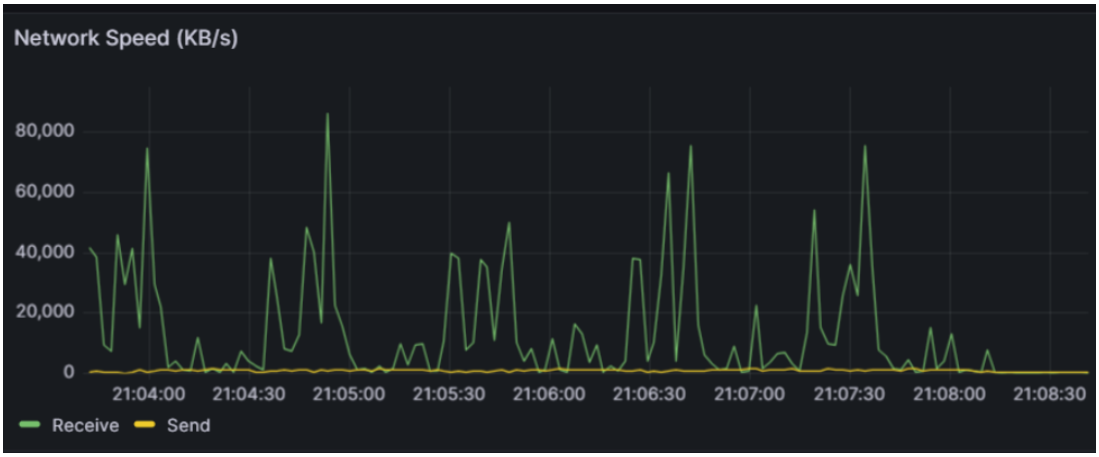


MD Core #2

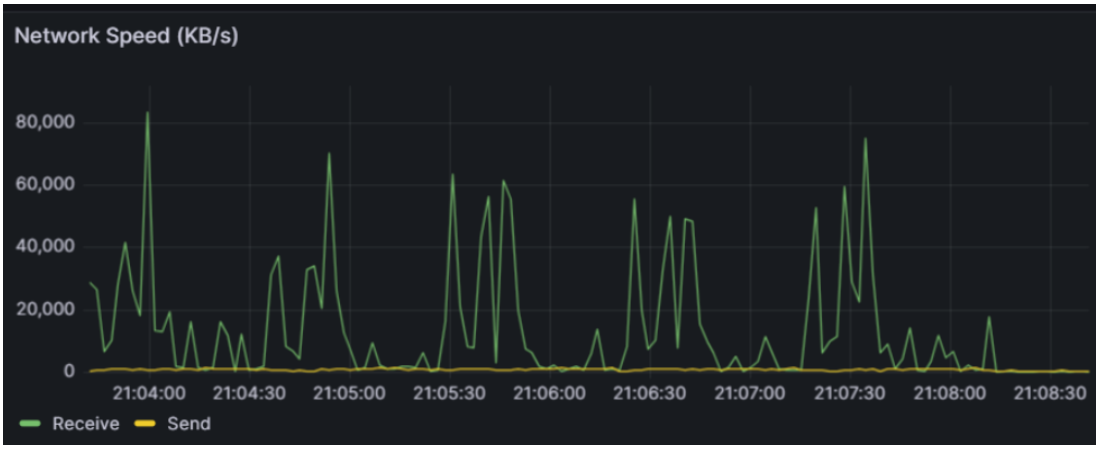


Network usage

MD Core #1



MD Core #2

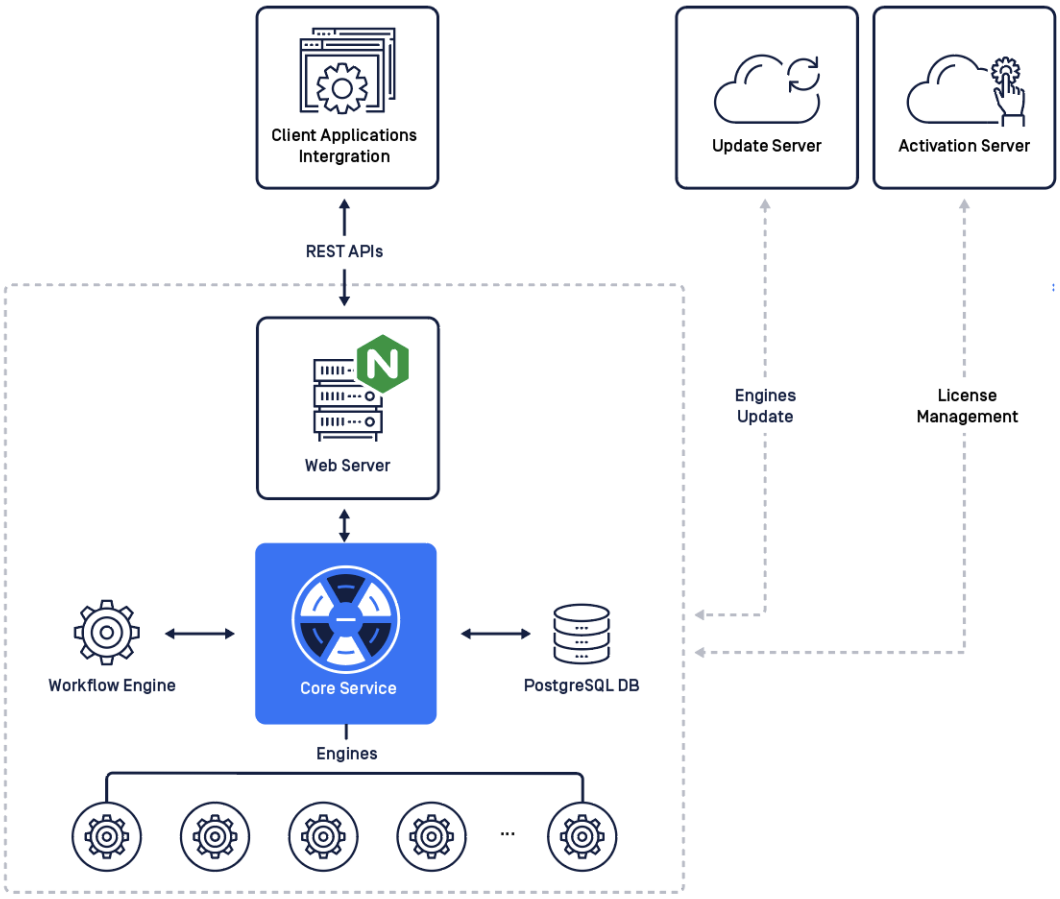

Network Speed (KB/s)
80,000
60,000
40,000
20,000
0
21:04:00
21:04:30
21:05:00
21:05:30
21:06:00
21:06:30
21:07:00
21:07:30
21:08:00
21:08:30
Receive
Send

MetaDefender Core Documentation

The users can consult this web page or, alternatively, they can download the manual in pdf format from the link below:

MetaDefender Core manual (SHA256:).

Recommended System Configuration



MetaDefender Core supports a wide range of deployment options, including air-gapped environments, providing the flexibility to choose the environment that best suits customers' requirements and infrastructure.

- Physical server.
- Virtualization platforms like VMware, Hyper-V and XenServer.
- Cloud Deployment.
- Container Deployment.

> **Note**
> Before installing MetaDefender Core, please refer to the **recommended minimum system configuration** listed below. The requirements are based on a standardized customer deployment and may require more or less resources to achieve the level of performance desired in your environment. Please note that the server specifications are built to allow a high volume daily processing. Please refer to Performance and Load Estimation for more details.

The recommendations below are for MetaDefender Core, API usage only. For any other use cases, please consult the user guide of the licensed products for accurate recommendations.

For certain use cases these might be adjusted and customized on specific needs and SLAs. We highly recommend to engage our ProServ team to assist in fine tuning MetaDefender and get the maximum performance out of your systems.

Microsoft Windows Deployments

[Windows] Supported Operating Systems

Only support 64-bit platforms.

- Windows 10, Windows 11 23H2, 11 IoT (22H2) only for End-User systems (e.g. Kiosk deployments)
  - Based on Microsoft License Terms, you're not allowed to use Windows Desktop as a server, therefore, MetaDefender Core can't be used on these systems for server use cases.
- Windows Server 2016, 2019, 2022

> **Disclaimer**
> End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

> **Plan for legacy OS support**
> OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:
>
> - Ubuntu 20.04 after **April 2025**.
> - Amazon Linux 2 after **June 2025**.
> - Windows 10 after **October 2025**.
>
> If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

[Windows] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 8	8	8 GB	16 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 12	16	16 GB	24 GB + 50 GB + 100 GB
MetaDefender Core 16	16	16 GB	32 GB + 50 GB + 100 GB
MetaDefender Core 20	32	16 GB	40 GB + 50 GB + 100 GB
MetaDefender Core MAX	32	32 GB	120 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core is installed.
- For a better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.
- SBOM processing requires more disk space, please allow at least 15GB free disk for the engine to work properly.

[Windows] Third Party Dependencies

- .NET framework 4.5 or above
- Microsoft Visual C++ Redistributable for Visual Studio 2015-2022 (version 14.38 or higher); x64 version.
- PostgreSQL: Only applicable when using a pre-installed PostgreSQL server running remotely.
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Some engines also have dependencies as described below:

Engine Name	Dependency
Deep CDR	Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 5.8.0 or above)
ESET	MetaDefender Core v4 temporary directory should have more than 200MB free disk space
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
Proactive DLP	• Microsoft Visual C++ 2017 Redistributable Package (Only applicable to engine version 2.0 or above) • .NET 6 dependencies • CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
Scrutiny	Microsoft Visual C++ 2015 - 2019 Redistributable x86 and x64
Systweak	.NET runtime 6.0, 7.0 or 8.0
Bkav Pro	Microsoft Visual C++ 2015 - 2022 Redistributable x64

[Windows] Installation Details

MetaDefender Core on Windows uses "C:\Program Files\OPSWAT" folder for storing resources or the installation directory.

MetaDefender will use its resources folder to store temp files as part of the analysis. **It's recommended to exclude this folder from real-time protection monitoring**. Knowledge base:

- How to set up exclusions on anti-malware software to prevent disruption
- Mitigating scan failure triggered by real-time protection service
- Can local AVs interrupt ongoing scans?

Linux Deployments

[Linux] Supported Operating Systems

Only support 64-bit platforms.

- Red Hat Enterprise Linux 8, 9
- Rocky Linux 9
- Debian 11, 12
- Ubuntu 20.04, 22.04
- Amazon Linux 2 is supported via Amazon Machine Image (AMI)

Plan for legacy OS support

OPSWAT will discontinue support for the following OS in MetaDefender Core and its associated engines:

- Ubuntu 20.04 after **April 2025**.
- Amazon Linux 2 after **June 2025**.
- Windows 10 after **October 2025**.

If you need some more time to transition from legacy OS, refer to this instruction Lock Your Engine and Plan for Migration.

For Hardended Linux OS

Some AV engines could be failed to initialize and run on MetaDefender Core (e.g. ESET, Kaspersky) due to execution permission required by them on `/var/tmp/ometascan` folder.

Please follow steps at Why have the ESET and Kaspersky scan engines failed to initialize on the hardened Linux OS? for remediation details.

Disclaimer

End-customer is responsible of verifying the OS license agreement and choose the right OS based on their planned usage of MetaDefender.

[Linux] Recommended System Configuration

The resources listed below (CPU, RAM, disk space) are the minimum recommended for MetaDefender Core:

Package	CPU cores	Free System RAM	Free Disk Space (See more details in Storage Usage Information)
MetaDefender Core 5	4	4 GB	10 GB (product and engines) + 50GB (temp files during processing) + 100GB (when using local PostgreSQL database)
MetaDefender Core 10	8	8 GB	20 GB + 50 GB + 100 GB
MetaDefender Core MAX	16	16 GB	40 GB + 50 GB + 100 GB

- It is suggested to use SSD for system where MetaDefender Core resides on.
- For better performance with Deep CDR and Proactive DLP technologies (if licensed), consider adding 8GB RAM, 4 CPU cores (at least 12 CPU cores in total) additionally.

[Linux] Third Party Dependencies

Dependencies list:

- openssl
- grep
- lib32stdc++6 (>= 4.5)
- libc6-i386 (>= 2.10)
- procps
- zlib1g
- libcurl3 (>= 7.19.7)
- libcurl4
- postgresql:
    - From 14.5 to 14.15
    - From 15.7
    - From 16.4

Not all the above dependencies will need to be installed, it is dependent on different Unix distro and version

Some engines also have dependencies as described below:

Engine name	Dependency
Archive engine	• To extract password protected MS Office files, you need to install .NET 6 dependencies, libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • Rocky Linux 9.x/Red Hat 9.x: libnsl
Deep CDR	• Rocky Linux 9.x: libgomp, ncurses-compat-libs, libnsl • CentOS: libgomp, ncurses-compat-libs • Red Hat 8.x: libgomp, ncurses-compat-libs, libnsl • Red Hat 9.x: libgomp, ncurses-compat-libs, libnsl, compat-openssl11 • Ubuntu/Debian: libgomp1, libncurses5 • libgdiplus 6.0.5 or above ◦ For Debian 12: use libgdiplus 6.0.4 • .NET 6 dependencies
Adaptive Sandbox (formerly Sandbox)	Embedded and Remote Engine: Dependencies and system Requirements
FileType engine	Rocky Linux 9.x/Red Hat 9.x: libnsl
Proactive DLP	libgdiplus, mscorefonts .NET 6 dependencies CPU must support AVX2 and SSE4.1 instruction set to use OCR feature
RocketCyber	libgomp

Note: Some dependencies may need to be installed from external repositories, not from the OS default repositories.

[Linux] Installation Details

MetaDefender Core default installation path is using /var folder for storing resources:

- `/var/lib/ometascan` : installation folder with all its resources (database, DLP processed / CDR sanitized / quarantined file storage, engine package and definition updates).
- `/var/log/ometascan` : application logs.
- `/var/run/ometascan` : runtime folder for Core related processes.
- `/var/lib/ometascan` : all successful deployed engines data files.
- `/usr/lib/ometascan` : all shared libraries required by MetaDefender Core, bundled PostgreSQL server related binaries and libraries.
- `/var/tmp/ometascan/resources` : all temporary processing files for MetaDefender Core to process.

Storage Usage Information

Based on the configuration, MetaDefender Core will need additional disk space to store analysis/result data, quarantined files and temporary files:

	Data Storage
Analysis Reports (Processing History)	Approximately 4.5GB is required for every 1M analysis reports stored. • 100 GB is recommended as a baseline. Database storage requirements are affected by Data Retention settings, engine settings and scanning load.
Quarantined Files	Depends on the customers' dataset and configuration
Sanitized Files	Depends on the customers' dataset and configuration
Temporary processing files	Depends on the customers' dataset and configuration • A 1GB non-archive file requires 1GB free disk space at least. • A 1GB archive file requires (1 * 2)GB free disk space at least. • 50GB is a baseline and should be adjusted based on observation of the solution within the environment, with consideration to peak times and overall file makeup (e.g. complexity, size, archives).
Note	Options are available to control how large and/or complex files are handled to limit the amount of system resources required. See Workflow Configuration Template for more information.

Custom Engines

The recommendations above are specific for MetaDefender pre-packaged bundles.

However for additional Custom Engines, please review the Knowledge base to review additional requirements (if any) for the selected engine.

Browser Requirements for the MetaDefender Core Management Console

One of the following browsers is suggested to view the MetaDefender Core Management Console:

- Microsoft Edge
- Chrome
- Firefox
- Safari

Chrome, Firefox, Safari and Microsoft Edge browsers are tested with the latest available version at the time of release.

Network Bandwidth

In online environment where MetaDefender Core is configured to download engine update packages directly from OPSWAT update server source, then using gigabit network speed (125 MB/sec) is highly recommended to avoid potential network related issue while downloading files, or at least 100 Mbps (12.5 MB/sec) network speed.

Allowlisting URLs

If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts':

- https://activation.dl.opswat.com - for license online activation
- https://update.dl.opswat.com - for engines and databases download

Note: *IP address-based allowlisting on your firewall might fail after some time since OPSWAT uses CDN (AWS Cloudfront) to faster delivery updates over the world, and IP address of edge servers might change over time.*

Depending on your location, connections with OPSWAT cloud-based servers be forwarded to specific AWS Cloudfront's edge locations, each is assigned with its own unique IP address ranges. You might also need to allow correct AWS Cloudfront's corresponding IP address ranges by referring to https://ip-ranges.amazonaws.com/ip-ranges.json

Warning

Even the OPSWAT update servers host updates for all of the available engines we support, sometimes custom engines might try to connect to their own cloud for updates, but this can be disabled in firewall and they will be updated just from OPSWAT cloud servers instead.

Differences Between MetaDefender Core v4 and v5

We are very excited to inform you about the release of version 5.0.0 of our MetaDefender Core!

The new version is designed to enhance the product performance, user experience, as well as to provide more flexibility, security, and scalability for our customers.

At the same time, we are also committed to keeping backward compatibility for whom running MetaDefender Core v4, and avoid the need for updating your client integration to move forward with MetaDefender Core v5.

v4 REST API Integration Supported
All public REST API endpoints available on MetaDefender Core v4 will be continously supported on v5.

v4 Product Major Features Supported
All product major features available on MetaDefender Core 4.21.2 (the latest one on v4) will be continously supported on v5.

Seamless Product Upgrade
Upgrading from MetaDefender Core v4 to v5 will be supported and covered with data migration without additional manual steps from users.

Licensing information will be also migrated.

Below are some highlights of the first v5 release, MetaDefender Core 5.0.0:

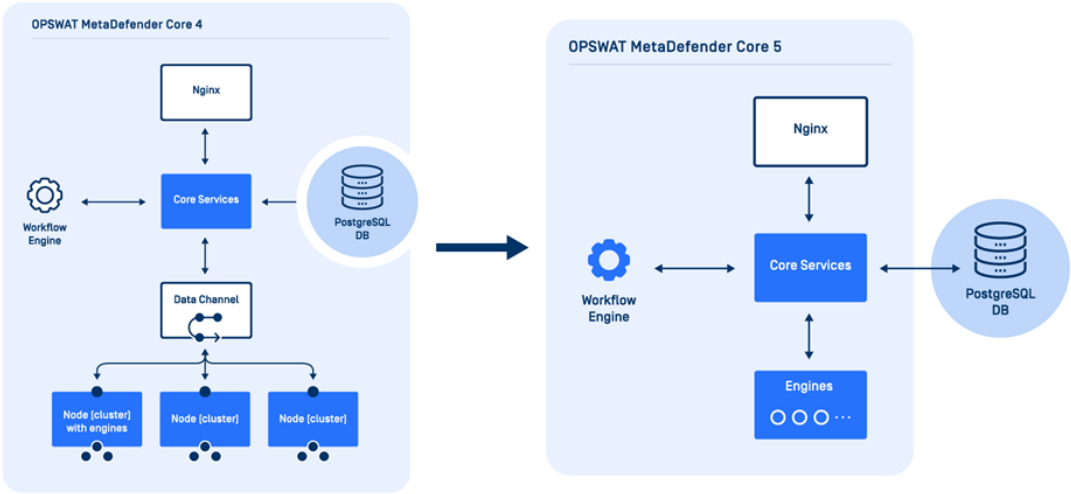
New product architecture

This version introduces a simpler product architecture that speeds up integration and automation, file processing flow, and immensely enhance the product performance and efficiency.

Earlier versions utilized MetaDefender Node concept, designed to support scaling of the scanning infrastructure by distributing scan requests among several different processing Nodes. Product licensing was based on number of Node instances. The benefit of having such a distributed infrastructure is that based on Node loads, Metadefender Core server can always choose the most appropriate Node to assign a new scan task to. However, having nodes in the v4 architecture caused more complexity and possible latency in the processing flow and entailed maintenance cost.

In this release, we retired multi-Node deployment and restructured MetaDefender Core's backend. This alteration significantly improves file processing speed and product efficiency, as well as eliminates extra cost for maintaining node services. Users now can simply install additional MetaDefender Core instances for scalability requirement,

as well as configure their own load balancer to coordinate tasks among multiple MetaDefender Core instances. Accordingly, product licensing is now based on number of the product instances.



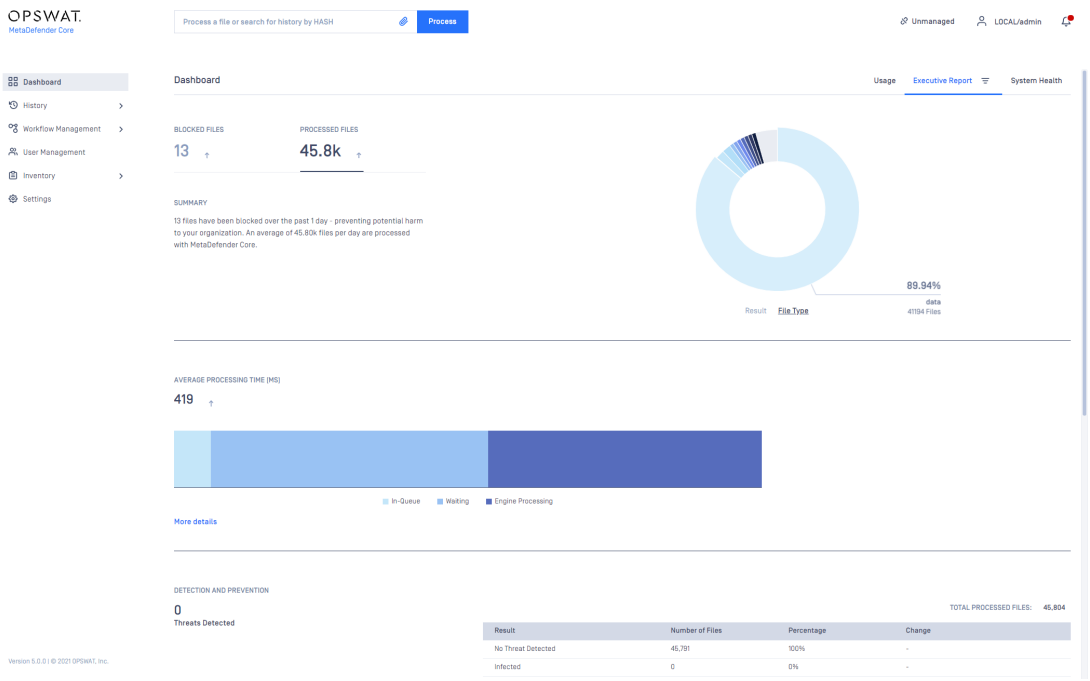
Please be informed that

- All existing administrative APIs related to "Node" on v4 will be kept as it is on v5 to ensure backward compatibility with existing client integrations.
- Applicable settings defined in Windows registry (Windows) or ometascan-node.conf config file (Linux) for Node will be migrated to Core. Details: MetaDefender Configuration

Brand-new product user interface

The second notable update in this version is a brand-new UI for the management console, which was designed with the focus on the quality and thoughtfulness of the user experience. The new administrative console UI provides an advanced set of Core features and observable processing status.

Comprehensive executive dashboard reporting offers dynamic, real-time and actionable threat insights. It also provides visibility of your security posture allowing you to paint a picture of security that aligns with the overall business. The report includes easy-to-read charts, graphs, and statistic explanation that help your security professionals reduce clutter, gather information quicker, spot threat activity early and act swiftly.



We're continuing to improve our product UI to provide users an excellent experience with the application and overall usability. Please expect more powerful UI features coming in next releases.

New compression archive engine for archive sanitization

In previous version, MetaDefender Core's archive engine implemented four functions including decryption, extraction, encryption and compression that with a high volume of files it takes time to process them all. For the purpose of enhancing archive handling performance, this release comes with a separated compression archive engine, which dedicatedly processes archive sanitization (compression and encryption). It is also created to facilitate maintenance and further feature development.

Archive sanitization feature now requires Deep CDR license

As to respect sanitization practice and ensure unknown threats prevention as designed, now starting on version 5, it's mandatory to have Deep CDR licensed for archive sanitization feature enablement on MetaDefender Core workflow configurations.

Archive handing related engines changed

The legacy "Archive" engine on MetaDefender Core v4 will be changed its name to "Archive Extraction" engine, even when you are still on MetaDefender Core v4.

A brand new "compression archive engine" is to dedicatedly handle archive sanitization in v5 (archive compression), that helps relevant archive handling performance improvement, easier for maintainance and further feature development.

Notes when you are still on MetaDefender Core v4 (not upgraded yet)

The legacy "Archive" engine (now name changed to "Archive Extraction" engine) on MetaDefender Core v4.x will still cover both extraction and compression archive tasks as usual.

Even though your MetaDefender Core v4.x might now have a new "Archive Compression" engine auto downloaded, but that engine cannot be deployed (marked as "permanently failed" in relevant engine status check API). This behavior is expected, and not causing any impact to MetaDefender Core processing flow, or its performance though.

- Just in case your client integration has a check for engine status before triggering scan request, then you might want to disable "Archive Compression" engine status on MetaDefender Core v4 using REST API. Detailed instruction: Archive Compression Engine Permanently Failed on MetaDefender Core v4
- When you later upgrade to MetaDefender Core v5 from v4, then you might need to manually re-enable "Archive Compression" engine to make it active and running on MetaDefender Core v5.

Notes for licensing

For offline deployment (MetaDefender Core is not connected to the Internet directly for online update), you will need to re-activate again on OPSWAT portal to retrieve latest engine list in .yml file including "Archive Compression" engine if you are currently licensed with Deep CDR.

Meanwhile, for online deployment, this manual procedure is not required since MetaDefender Core will cover that itself.

Import & export configurations

Not only workflow settings like in previous versions, but also all other general settings (update, security, data retention, server, hashes to skip) and most importantly, user settings, will be able to export and import on MetaDefender Core version 5.

Learn more at Import/Export configuration

Backward compatibility supported

The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

Windows application event logging disabled by default

Applicable only to Windows OS based MetaDefender Core.

Help avoid duplicated logging with separate file option, and improve overall processing performance. Of course, when being still valid to you, you are still supported to enable it back.

CSRF token header renamed

CSRF token originally created for security strengthening and being used by MetaDefender Core management console UI since version 4.21.0 via header `oms_csrf_token`

To get it compliant with some load balancers which do not accept underscore in header name by default, and save you from additional load balancer configuration step, we have now changed the header name to `oms-csrf-token`

This change does not impact to any REST API based client integration.

More comprehensive and accurate error messages on REST APIs

Impacting on following REST API endpoints:

- `PUT /admin/config/session`
- `PUT /admin/config/update`
- `PUT /admin/config/scan`
- `PUT /admin/config/skip`
- `PUT /admin/config/auditlog`
- `PUT /admin/config/webhook`
- `PUT /admin/config/rule`

Upload time and queue time logic updated in JSON scan response

Impacting on following REST API endpoints:

- `GET /file/{data_id}`
- `GET /hash/{md5/sha1/sha256}`
- `GET /archive/{data_id}`
- `POST /file/{async}`

Added new field:

- `receive_data_timestamp`: The timestamp when upload progress started (first byte received) (in milliseconds)
- `upload_time`: Total time elapsed for upload process (in milliseconds)

Modified existing field (new logic):

> **Backward compatibility supported**
> The legacy feature for import & export using /admin/import API endpoint will be still supported, and still expecting to have JSON file exported for workflow related settings only.

- `upload_timestamp`: The timestamp when upload progress finished (all bytes received) (in milliseconds)
- `queue_time`: Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue time = start_time - upload_ timestamp) (in milliseconds)

Release version 5.0.0 also comes with variety of outstanding features and improvements.

For detailed information about our latest release please view our Release notes

Installation

Starting with MetaDefender Core 4.20.0, the centralized/shared database deployment model is supported (all MetaDefender Core instances share the same database), along with traditional standalone database mode (every MetaDefender Core instance has its own database to manage).

- Standalone mode
- Shared Database Mode

Installation

1. Download the package of your choice from My OPSWAT
2. Install the package on your computer via the Command Line or via the Wizard
3. Open a web browser and point to http://<server name or IP>:<port>
    - The default port is **8008**
    - In case of problem check Inaccessible Management Console page
4. Complete the required steps of the basic configuration wizard
5. You must activate this deployment to use its features

Installation notes

- If the MetaDefender Core package **dependencies are not installed** on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.
- During installation the databases might need to be upgraded. This could take noticeable time depending on database size (eg.: length of scan history).

Standalone mode

Wizard Installation

The Install Wizard is only for the Windows installer (.msi file).

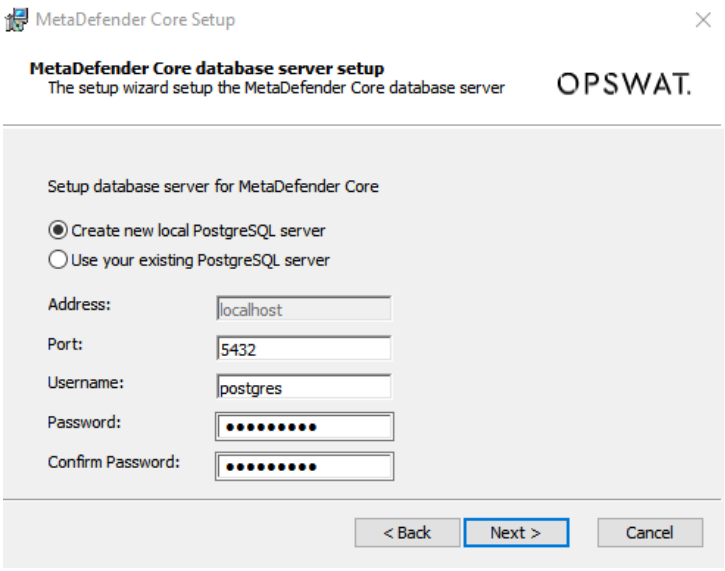
Since MetaDefender Core 4.19.0, we have switched to using PostgreSQL DBMS which needs a dedicated database server. MetaDefender Core supports two modes to setup PostgreSQL server:

Local (Bundle) PostgreSQL server

MetaDefender Core will install a new PostgreSQL server locally in the same box with the product.

> **PostgreSQL user credentials**
> Unicode characters are NOT supported for "Username" and "Password"
>
> "Password" does not accept # (hash) as starting character (e.g. #ABC)

> **The ignition file**
> If the ignition file exist in the designated location, then MetaDefender Core will use the details defined in the ignition file, rather than what is passed via the installation wizard.
>
> More info about the ignition file based installation: Command Line Installation

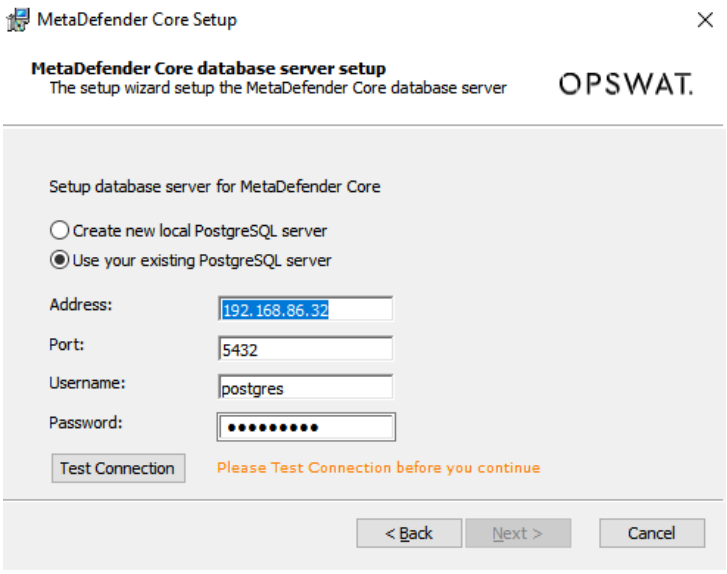


Remote (Existing) PostgreSQL server

MetaDefender Core will connect to remote pre-installed PostgreSQL running remotely.

> **PostgreSQL connection configuration**
> "Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

> **PostgreSQL user privilege requirements**
> MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.
>
> Details: PostgreSQL User Privilege Requirements



Command Line Installation

Ignition File

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Make sure to create the ignition file, if it doesn't exists before you go ahead and install MetaDefender Core via command line.

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

MetaDefender Core supports two modes to setup PostgreSQL server:

1) Local PostgreSQL server

- MetaDefender Core will install a new PostgreSQL server locally, next to the product.
  - A sample iginition file for PostgreSQL server information (***Only non-Unicode characters supported for "user" and "password"__***):

ometascan.conf bash

```
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
```

2) Remote PostgreSQL server

> **Info**
>
> **PostgreSQL version 12 (at least 12.3) or version 14 is required.**
>
> According to PostgreSQL notice, version 12 will stop receiving fixes on November 14, 2024, so we suggest that you use version 14.

MetaDefender Core will connect to a pre-installed PostgreSQL server running remotely and will setup its database.

Make sure to follow steps at Open Connection On PostgreSQL Server first, so that your remote existing PostgreSQL server to accept upcoming connection from MetaDefender Core.

A sample ignition file for PostgreSQL server information:

ometascan.conf bash

```
[dbserver]
type=remote
host=192.168.86.32
port=5432
user=existingserver_admin_user
password=existing_server_pass
```

After installing MetaDefender Core successfully, you may want to remove all credentials info created for [dbserver] section in ignition file for security reason.

> **Warning**
> If the MetaDefender Core package dependencies are not installed on your system you may need to have a working Internet connection or you may have to provide the Installation media during the installation. Consult your Operating System documentation on how to use Installation media as a package repository.

Debian package (.deb)

bash

```
sudo dpkg -i <filename> || sudo apt-get install -f
```

On Red Hat Enterprise 7.x Linux / CentOS package (.rpm)

bash

```
sudo yum install <filename>
```

For systems which enabled GPD check flag:

bash

```
sudo yum install --nogpgcheck <filename>
```

Red Hat Enterprise Linux 8.x package (.rpm)

Install the latest PostgreSQL repository RPM:

> **Note**
> Due to the shorter support cycle on Red Hat Enterprise Linux, all supported versions of PostgreSQL are not available on this platform. The PostgreSQL Yum Repository will integrate with the normal systems and patch management, and provide automatic updates for all supported versions of PostgreSQL throughout the supported lifetime of PostgreSQL. Reference: https://www.postgresql.org/download/linux/redhat/

bash

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

In case installing a remote PostgreSQL on either the same MetaDefender Core server or another server, you need to install the following additional package. PostgreSQL version 12.3 or newer is recommended.

bash

```
sudo dnf install -y postgresql12-contrib
```

Install MetaDefender Core:

bash

```
sudo dnf install -y <filename>
```

Windows package (.msi)

On Windows systems it is possible to install the product by running the corresponding .msi file.

From command line interface it is also possible to install the product by executing

bash

```
msiexec /qn /i <msi file name> <option key>=<option value>
```

where the possible keys and their default values are the following:

Key	Default Value	Description
INSTALLFOLDER	\Program Files\OPSWAT\MetaDefender Core	Customize installation folder for product Example: `INSTALLFOLDER="D:\Products"`
RESTADDRESS	*	REST interface binding IPv4 or IPv6 address ('*' means that service listens on all IPv4 and IPv6 interfaces)
RESTPORT	8008	REST interface binding port
REPORT_ENGINE_ISSUE	true	Enable reporting of engine issue count. (possible values: "true" or "false").

Note

Reporting of engine issue count

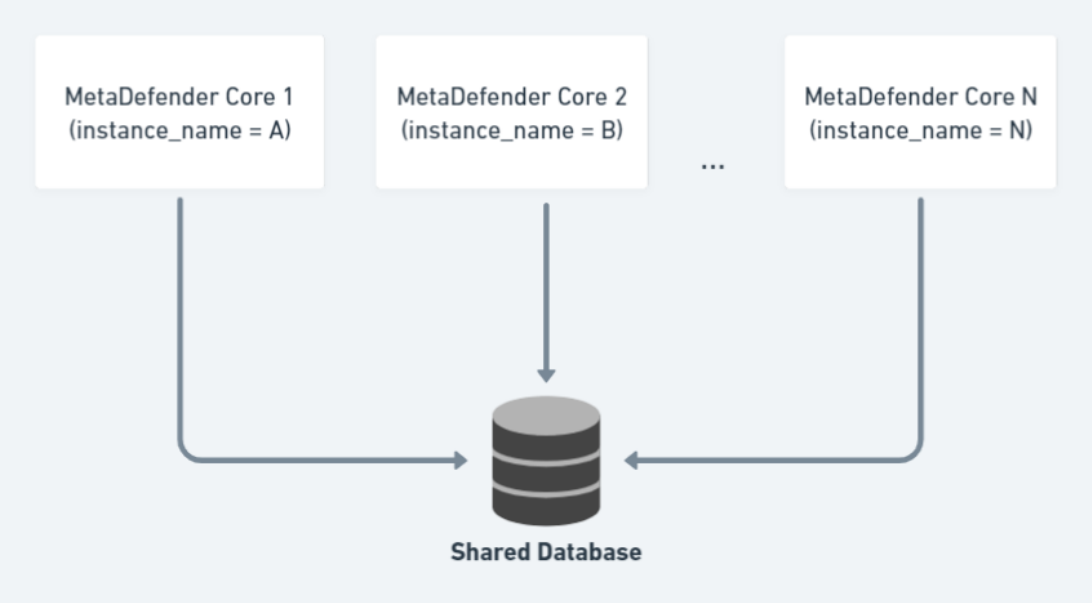
If reporting of engine issue count is enabled, MetaDefender Core server will send only the **number** of initialization errors and **number** of unexpected stops for the specific db/engine version. This information is sent over a HTTPS channel when the product downloads the latest package descriptors. This information is used for early detection of any specific 3rd party engine quality issues.

For details on using msiexec please consult Windows installer documentation.

Shared Database Mode

Starting with MetaDefender 4.20.0, you can use a centralized database deployment model, where all MetaDefender Core instances can share the same external database.

That helps auto-sync all product configurations (**except Single-Sign-On** that requires unique settings for each Core), processing history, live/on-going scan result query, audit, statistics data.



This article demonstrates how to install and setup MetaDefender Core 4.20.0 or newer with shared database mode.

Important notes & current limitations:

Currently not supported

- Does not support Central Management use-case with shared DB mode.
- Does not support to switch from shared database mode back to standalone database mode.
- Does not support to change instance_name of MetaDefender Core in shared database deployment model after installation.
- Does not support upgrading older MetaDefender Core from non-persistent mode to using the shared database mode.
- Does not support upgrading older MetaDefender Core from standalone mode to using the shared database mode.
- Does not support to upgrade multiple MetaDefender Core instances at once in shared database mode
    - Stop all Core instances' scanning service before performing the upgrade.
    - Upgrade each Core server one by one instead.

- For each Core, when the upgrade finished, then you can start using that Core for scanning without waiting for the other Core instances' upgrade to be done.

> **Disclaimer**
> MetaDefender Core does not support mode switching.

Requirements for shared database setup:

- The central remote PostgreSQL server (for shared database) could be utilized with a built-in (native) PostgreSQL database server installed with one of the MetaDefender Core instances in the network. Or using a dedicated existing PostgreSQL database server (see detailed requirements for PostgreSQL version at Recommended System Configuration).
- Before setting up, make sure to follow steps at Open Connection On PostgreSQL Server so that all MetaDefender Core instances could connect to the central remote PostgreSQL server.
- MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core, details: PostgreSQL User Privilege Requirements
- All MetaDefender Core instances must be installed on the same Operating System (Windows / Linux).
- The first MetaDefender Core instance connects to the shared database must be ready first (license activated, users/groups/roles created) before setting up other MetaDefender Core instances to connect and use that shared database.

Engine update:

- If engine updates is configured to be via Folder source, all MetaDefender Core instances must be able to access to the configured update folder, and the option DELETE FILES AFTER IMPORT will be unchecked and prevented for modification.

Authentication:

- Single-Sign-On (SSO) authentication settings (if using) cannot be shared among MetaDefender Core instances because SSO requires an unique login URL for every different Core.
- If an user is locked for login on a certain MetaDefender Core instance(e.g. exceeded the number of failed login times), that user is not prevented for logging in on any other instances.
- If an user is locked for sign-in on a certain MetaDefender Core instance (e.g. exceeded the number of failed login times), if the admin releases lockout on that user on that MetaDefender Core instance, that user is still not able to login on other MetaDefender Core instances.

Sanitized / DLP / Quarantined location changed after MetaDefender Core install:

Windows:

Shared storage path will be stored in 3 strings under key global in registry: quarantinepath, dlppath, sanitizepath as below

(Default)	REG_SZ	(value not set)
dbmode	REG_SZ	4
dlppath	REG_SZ	\\10.40.50.100\test
instance_name	REG_SZ	481rvj
quarantinepath	REG_SZ	\\10.40.50.100\test
restaddress	REG_SZ	*
restport	REG_SZ	8008
sanitizepath	REG_SZ	\\10.40.50.100\test

You can set either 1, 2 or 3 different paths for the 3 settings above (means that quarantined, DLP and sanitized files will be stored in same or different storages corresponding). In case shared storage is not set in installation, values of 3 settings above will be empty.

Restart MetaDefender Core service after changed.

Linux:

Open file `/etc/ometascan/ometascan.conf` and set shared storage path for 3 keys quarantinepath, dlppath, sanitizepath under section [global]

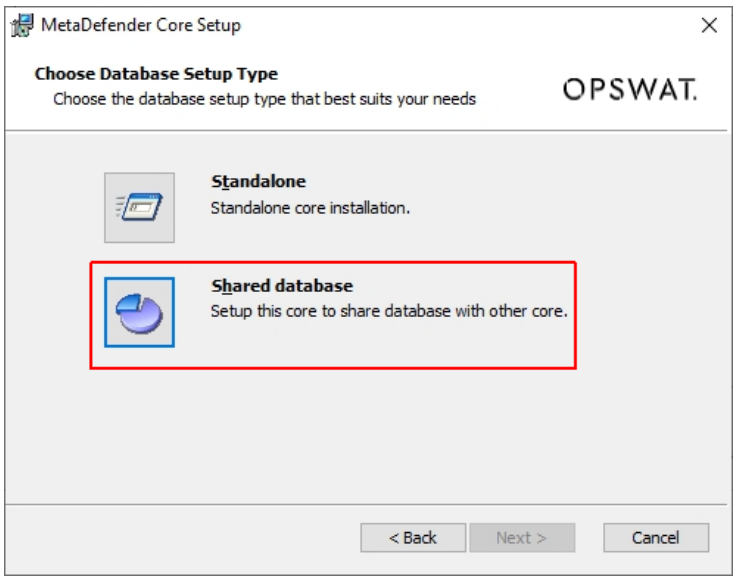
Restart ometascan service after changed.

Wizard Installation (Windows Only)

> **Info**
> Only applicable to MetaDefender Core 4.20.0 or newer using wizard on Windows OS.

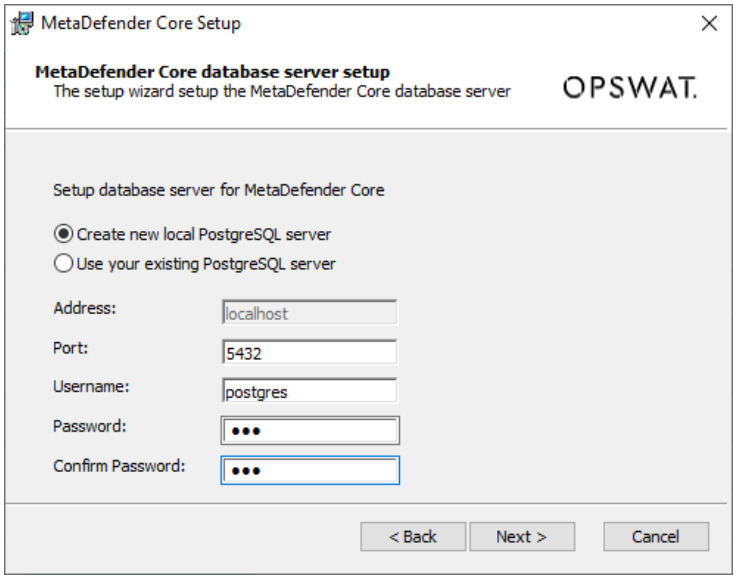
Clean install scenario

1. **Choose Shared database option:**

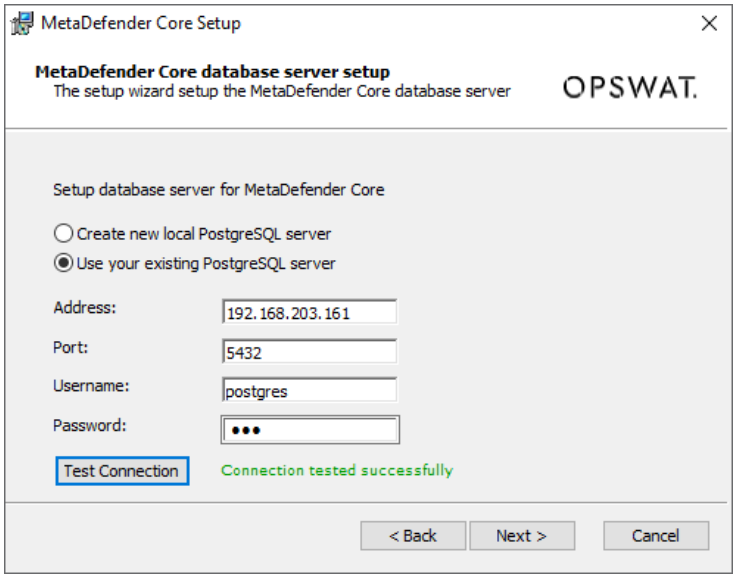


2. **Setup database server for MetaDefender Core:**

You can choose to create a native / built-in PostgreSQL server and setup the shared database on (then other MetaDefender Core instances will want to connect to that PostgreSQL server afterwards)



Or choose to connect to a pre-setup existing PostgreSQL server:



PostgreSQL connection configuration

"Test Connection" button is required, to make sure the PostgreSQL is connected and authenticated successfully. Make sure to check and perform steps at Open Connection On PostgreSQL Server if you are unable to test the connection with remote PostgreSQL server.

PostgreSQL user privilege requirements

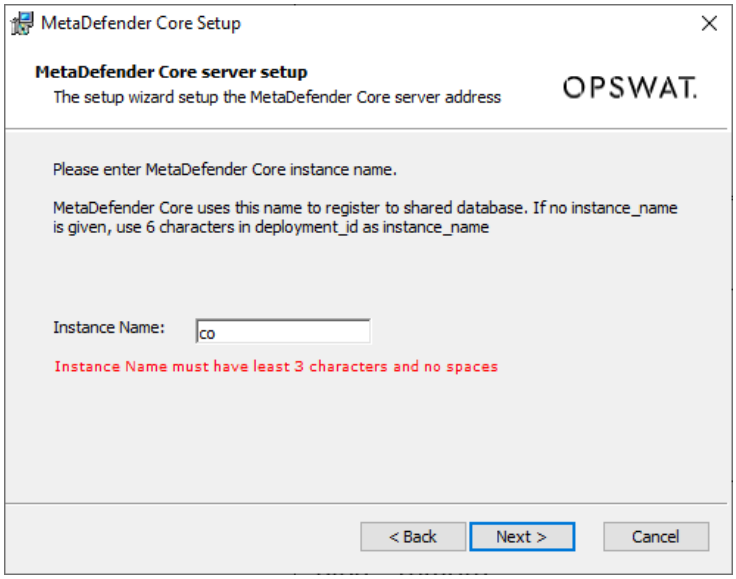
MetaDefender Core will need to create and use dedicated PostgreSQL users for database setup, upgrade and its own operations. Those users requires certain privileges configured on PostgreSQL server, otherwise expecting failures to occur while installing and operating MetaDefender Core.

Details: PostgreSQL User Privilege Requirements

3. **Create an unique instance name for that MetaDefender Core (different from other MetaDefender Core instances):**

Instance name must contain at least 3 characters and no white space.

If you choose to leave it blank, which means it will use the default instance name with 6 characters extracted from that MetaDefender Core deployment ID.

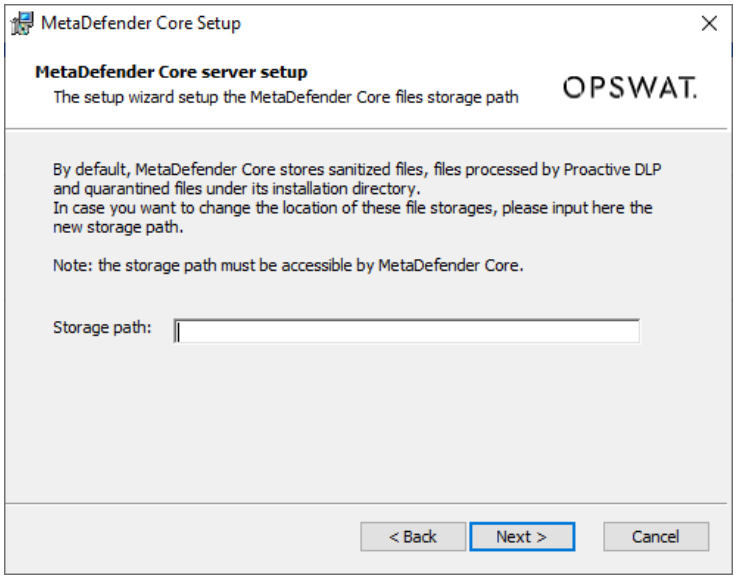


4. **Input the file storage path (sanitized, DLP, quarantined files)**

In the shared database model, all MetaDefender Core instances could use the same folder path to store all their own sanitized, DLP, quarantined files after processed.

The default location is: ***<MetaDefender Core installation folder>\data\ {sanitized|dlp|quarantine}***

If you wish to use that default location, just leave the input box blank and press Next.



Otherwise, you can specify a different location of sanitized/dlp/quarantined files. It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core.

For example:

- C:\new_folder
- \192.168.1.1\new_folder

Command Line Installation

> **Note**
> For command line installation (on both Linux and Windows), it is **mandatory** to use MetaDefender Core ignition file to add PostgreSQL server information **BEFORE** installing MetaDefender Core 4.19.0 or newer.

Clean install scenario

1. Create ignition file if not existed

- Windows: **C:\OPSWAT\ometascan.conf**
- Linux: ***/etc/opswat/ometascan.conf***

An example of ignition file:

ometascan.conf bash

```
eula=true

[global]
dbmode=4
instance_name=MetaDefenderCoreA
storage_path=D:\path\

[user]
name=admin
password=admin
email=admin@local

[dbserver]
type=remote
host=192.168.1.1
port=5432
user=postgres
password=your_own_password
```

Field	Required	Mapping to wizard UI	Possible values
global/dbmode	optional default is 1 (standalone)		1 - Standalone 2 - Central Hub deployment 3 - Non-persistent **4 - Shared**
global/instance_name	optional default is 6 characters of deployment ID		Unicode and number At least 3 characters No whitespace
global/storage_path	optional default is empty		Path to storages (DLP, quarantined, sanitized files) It could be a local path on the same machine, or a shared folder located in a remote connected machine. Make sure that folder is open access (read and write) to MetaDefender Core. For example: • C:\folder • \192.168.1.1\folder

2. Run installers using command line to install

Please refer to Command Line Installation for installation command.

Deployment automation support

Note

An Ignition file could be utilized for either / both of following use-cases:

1. Pre-define PostgreSQL server information for MetaDefender Core to connect and setup database: only required on Linux installation, or Windows with command line installation. Learn more details at:
    a. Standalone Mode > Command Line Installation (applicable on version 4.19.0 or newer).
    b. Shared Database Mode > Command Line Installation (only applicable on version 4.20.0 or newer).
2. Automating to accept EULA, create a default local admin user and import MetaDefender Core configurations upon installed: Those steps could be automated using ignition file described in this page.

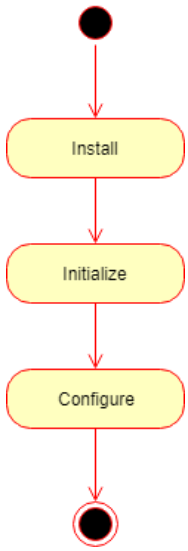
Ignition file location to create:

- **Windows**: `C:\OPSWAT\ometascan.conf`
- **Linux**: `/etc/opswat/ometascan.conf`

The product supports fully automated deployment. It means that it can be installed and configured with no human interaction.

The automated deployment can be split to three steps on a high level:

1. Installation
2. Initialization
3. Configuration



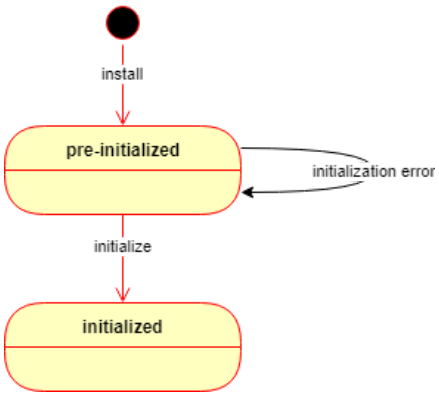
Installation

To automate the installation, install the product from the command line and provide the installation-time options as parameters to the installer. For further details see Command Line Installation

After the installation is complete, the product starts up and waits in a pre-initialized status. The product may be initialized in two ways:

1. Manually using the Wizard Setup or
2. Automatically using an *ignition file* (see below).

> **Warning**
> If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried after fixing the problem (e.g. placing the ignition file in its lookup location) and restarting the OPSWAT MetaDefender Core service.
>
> Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

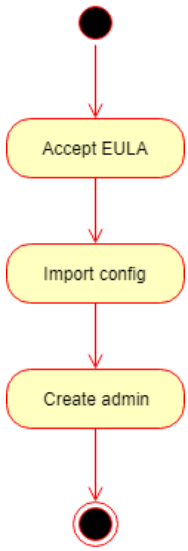


Initialization

Initialization is the process of bringing the product to an operable status.

Basically the initialization consists of the following steps:

1. Accept the End User License Agreement (EULA),
2. Import product configuration and
3. Create the first administrator user account.



Ignition file

The initialization process can be configured in a file called the *ignition file*.

> **Warning**
> In Linux, the ignition file and the folder /etc/opswat should have the permission 755

Ignition file location

- **Windows:** `C:\OPSWAT\ometascan.conf`
- **Linux:** `/etc/opswat/ometascan.conf`

The ignition file must be in conf format.

ometascan.conf bash

```
eula=true
[user]
name=admin
password=admin
email=admin@local
apikey=44c96cbb4e4e6c0a1d16bcb3c3da24def317
[config]
import=config_package.zip
import_password=123
[security]
session_cookie_samesite=Lax
[internal]
skip_startup_pages=true
[dbserver]
type=local
host=localhost
port=5432
user=postgres
password=non_Unicode_password
private_username=my_internal_db_user
private_password=mypassword
[ocm]
ocm_url=https://my.opswat.com
reg_code=XXXXXXXX
```

Ignition file fields

The ignition file accepts the following fields:

Section	Key	Required	Accepted Values	Description
	eula	Optional (mandatory when having [user] section)	true false (default)	Whether to accept the End User License Agreement. This key must be set at the top of ignition file. This key must be set to "true" to accept the EULA. Any other value will cause the initialization to fail. When having [user] section defined, then this key must be set.
[user]		Optional		Initial administrator user account properties. The Administrator role is granted to the account.
	name	Optional	string	User name for the initial administrator user account.
	password	Optional	string	Password for the initial administrator user account. WARNING! Clear text password The password in this configuration file must be stored in its clear-text format and as so it may be visible for unauthorized parties.
	email	Optional	string	E-mail address for the initial administrator user account.
	apikey	Optional	string	API key for the initial administrator user account.
[config]		Optional		
	import	Optional	string	Path to a file in json/zip format that contains a previously exported configuration to be imported. Zip-format file contains a full configuration including user management settings, so it will overwrite [user] section.
	import_password	Optional	string	Password to decrypt .zip configuration package.
[security]		Optional		

Section	Key	Required	Accepted Values	Description
	session_cookie_samesite	Optional	Strict Lax (default) None	Declare if session cookie should be restricted to a first-party or same-site context.
[internal]		Optional		
	skip_startup_pages	Optional	true false (default)	Skip all welcome and other startup pages (e.g. upgrade result) when installing / upgrading MetaDefender, and going straight to the product index default page.
[dbserver]		Optional		
	type	Mandatory when having [dbserver] section	local remote	Where the PostgreSQL database server should physically locate. "remote" means PostgreSQL server and MetaDefender Core are not in the same machine.
	host	Mandatory when having [dbserver] section	string	IP address / domain name of the server where PostgreSQL server locates. "localhost" can be used when applicable.
	port	Mandatory when having [dbserver] section	number	Port of PostgreSQL server is listening for connections from clients (i.e. MetaDefender Core).
	user	Mandatory when having [dbserver] section	string	PostgreSQL server's user. SUPERUSER privilege is required for MetaDefender Core to setup its database and extensions for the first time. **Only non-Unicode characters supported.**

Section	Key	Required	Accepted Values	Description
	password	Mandatory when having [dbserver] section	string	PostgreSQL server's user credentials. **Only non-Unicode characters supported.**
	private_username	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
	private_password	Optional	string	PostgreSQL server's internal user created for MetaDefender Core own operational purpose. If not specified, then MetaDefender Core will auto generate this user. See details at Customize Internal PostgreSQL User
[ocm]		Optional		
	ocm_url	Optional	string	URL to My OPSWAT server.
	reg_code	Optional	string	A registration code to enroll to My OPSWAT server.

Detailed initialization process

1) After the product has been started, it looks for the ignition file in the designated location.

Windows: C:\OPSWAT\ometascan.conf ** Linux: /etc/opswat/ometascan.conf

** 2) If an ignition file is found, then

2.1) It gets validated, and if it is valid, then based on the information found in the ignition file:

- The EULA is accepted,
- The configuration is imported,
- The administrator account is created.
- If any of the above steps fails, then the error is logged, and the initialization gets terminated.

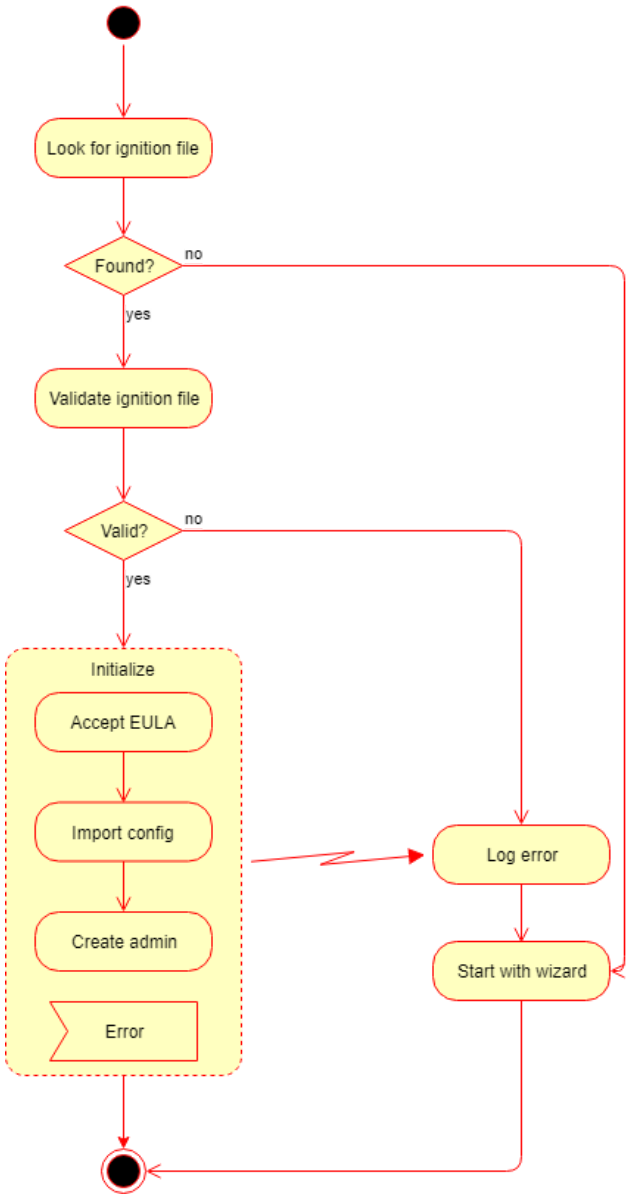
> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

2.2) If it is not valid, then the error is logged, and the initialization gets terminated.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.

3) If there is no ignition file, then no initialization is performed.

> **Info**
> In this case the product starts normally: if for example the basic configuration wizard has not been completed yet, then it must be completed first.



Warning

If the automated initialization fails for some reason (e.g. the ignition file is not in place) then the automated initialization may be retried fixing the problem (e.g. placing the ignition file to its lookup location) and restarting the OPSWAT MetaDefender Core service.

Until the product is in pre-initialized status, it will try the automated initialization every time after a service (re)start.

Configuration

After the initialization is complete, the product is ready with the default and the imported configuration.

This configuration can be later changed calling the configuration API functions. For further details about the API see Configuration related APIs API

CIS Level 1 Guidelines

Support Red Hat Enterprise Linux 9
For more details about Center for Internet Security (CIS) please refer to this document

Instruction steps

I. Install OpenSCAP

bash bash

```
yum install openscap-scanner scap-security-guide
```

II. Generate a result file and a HTML report using OpenSCAP scanner tool

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

III. Remediation of CIS Level 1 issues

Generate a remediation script based on the `ssg-rhel9-ds.xml` file:

bash

```
oscap xccdf generate fix --profile
xccdf_org.ssgproject.content_profile_cis_server_l1 --fix-type bash --output
remediations.sh /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

And execute remediation script.

bash

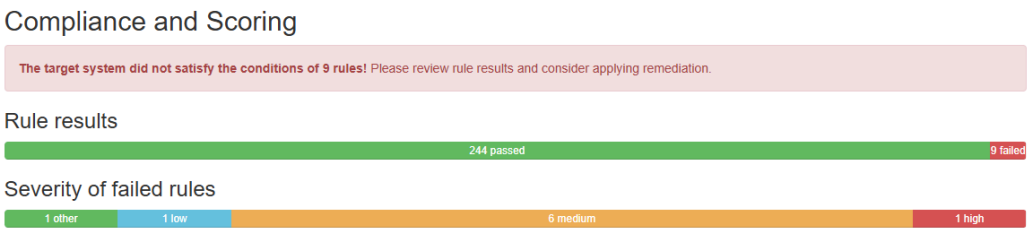
```
./remediations.sh
```

IV. Review the results after remediation

bash

```
oscap xccdf eval --profile xccdf_org.ssgproject.content_profile_cis_server_l1 --
results scan_results.xml --report scan_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

Example:



Notes

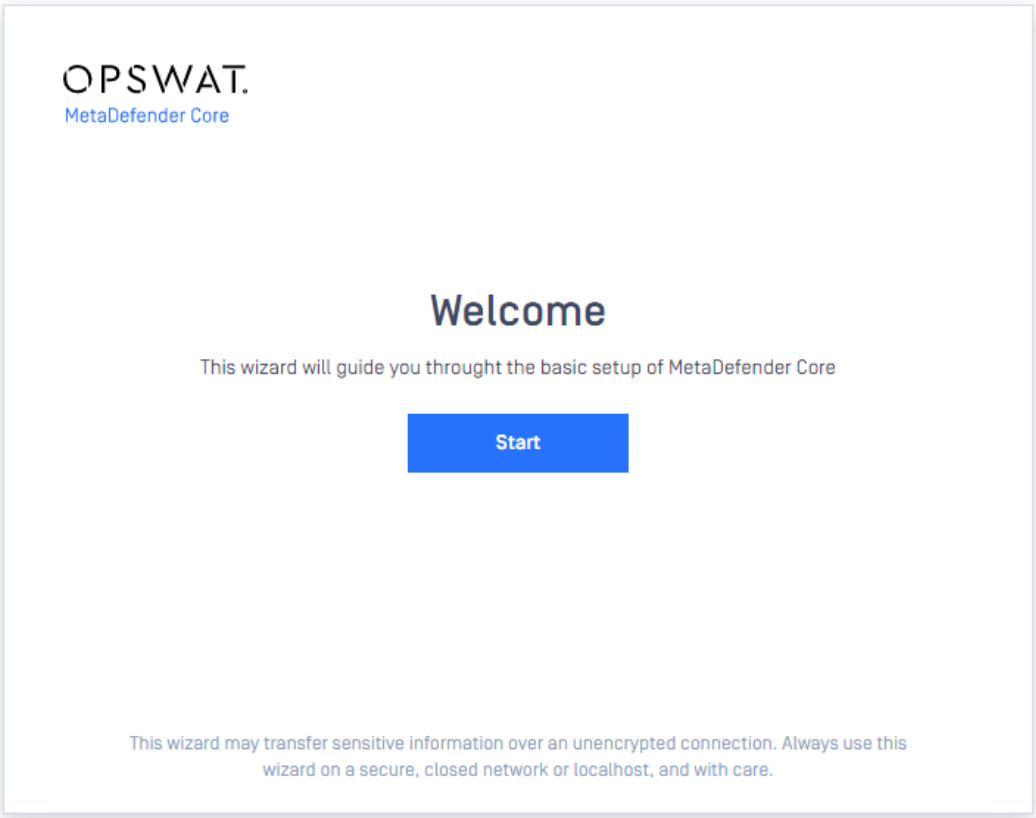
- CIS Level 1 requires `/tmp` folder to be mounted in a separate partition. Please ensure that that new partition have enough disk space for MetaDefender Core to run.
- CIS Level 1 requires that "Ensure No World-Writable Files Exist".
    - For now, when freshly installing MetaDefender Core, all its binary files meet the requirement.
    - When installing/updating engines, some engines might create additional files for its operation, and it might violate this requirement. In this case, you need to again execute the remediation script in the step III.

Wizard Setup

Introduction

Access the MetaDefender Core management console for the first time (i.e. http://localhost:8008), you are to complete a quick walkthrough for wizard setup in order to be able to use the product.

The administrative management console will be available only upon you have successfully finished this wizard stage.

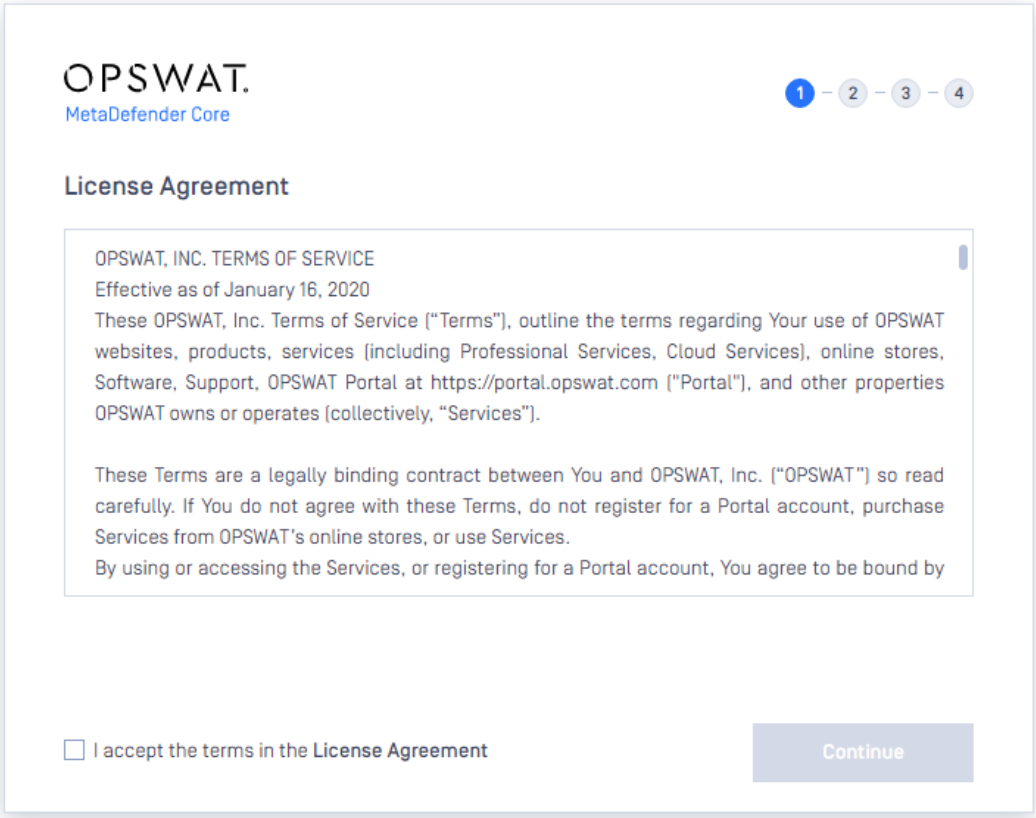


Warning

This wizard may transfer sensitive information over an unencrypted connection. Always use this wizard on a secure, closed network or localhost, and with care.

Basic configuration steps

End-User License Agreement



In the first page you can find the End-User License Agreement. You have to accept the terms before moving on.

Please read through the EULA (https://docs.opswat.com/policies/privacy-policy/portal-privacy-policy) carefully and if you agree with it, check `I accept the terms in the **License Agreement**` and click `Continue` to proceed.

Local Admin User Setup

The next step is to set up a default local administrator account.

This account will be the first one being able to access the administrative management console, and to create accounts for other users. You have to fill out all fields in this page to be able to move forward. When you are done, click NEXT to continue.

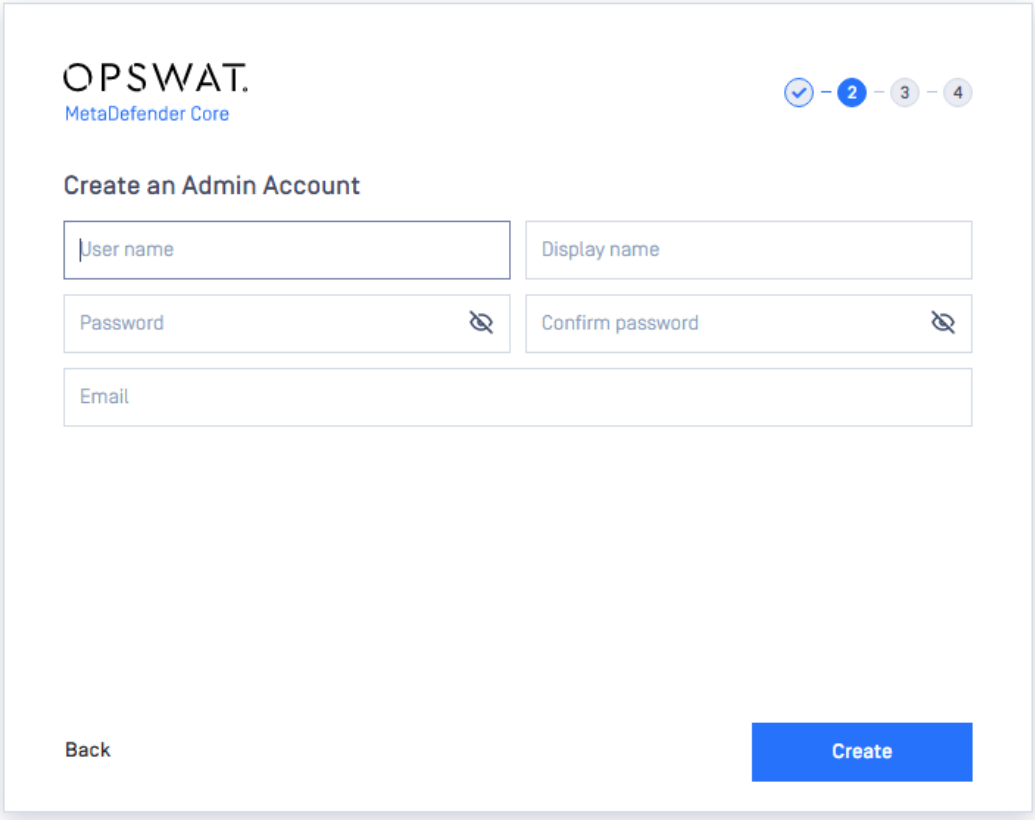
Info - User Directory

The administrator account, that is created via the basic configuration wizard, is always added to the LOCAL user directory as a member.

Learn more about user directory: User directories

The following information is required for the administrator account:

ACCOUNT NAME	The unique name of the account that is used at the time of login and in log messages for accountability.
ACCOUNT DISPLAY NAME	Name of the person bound to this account. This name (appended to the name of the account's user directory) is displayed in the top right corner of the administrative management console.
PASSWORD	Password of the user bound to this account that is used at the time of login. Passwords sent clear-text As long as TLS is not configured for the basic configuration wizard, passwords are sent clear-text over the network and may be disclosed to unauthorized parties. As a mitigation action: 1. Either use the wizard on *localhost* or on a direct network connection, or 2. Enabling HTTPS as soon as possible and change the password immediately if it has already been set.
EMAIL	Email address of the person bound to this account.



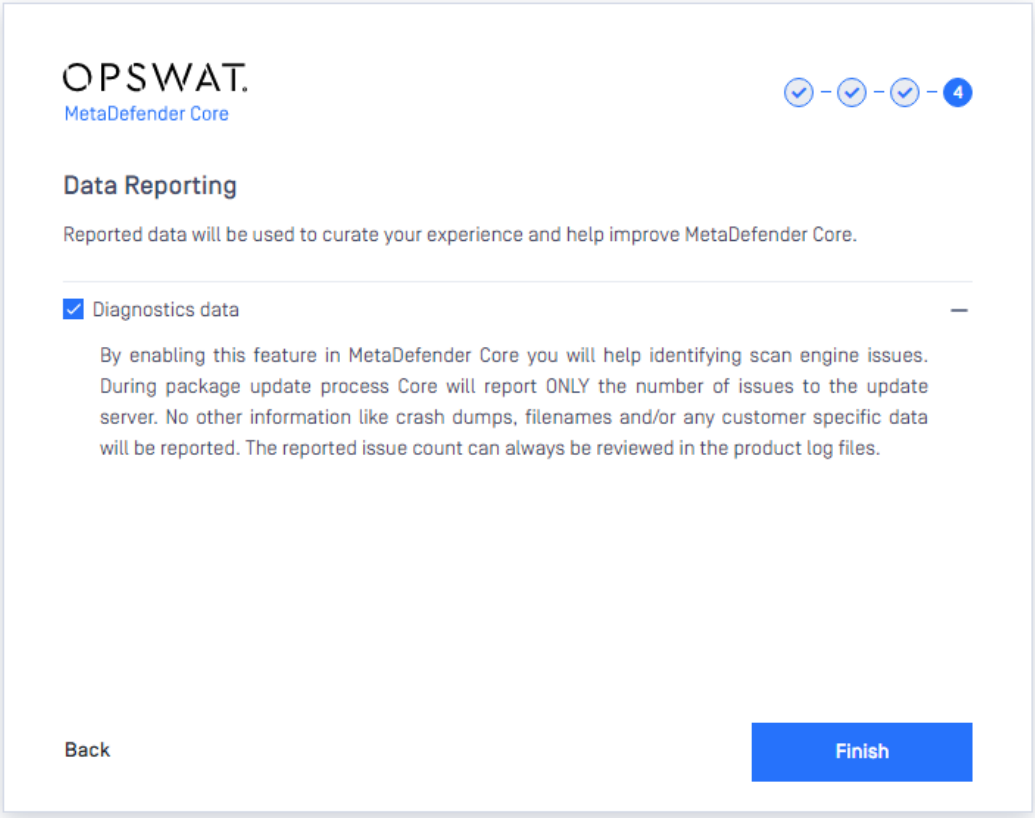
License activation

For license activation details see License Activation

Data Reporting

Enabling this option to allow MetaDefender Core collect its own engine issues and report back to OPSWAT for further analysis.

MetaDefender will not collect any other information.

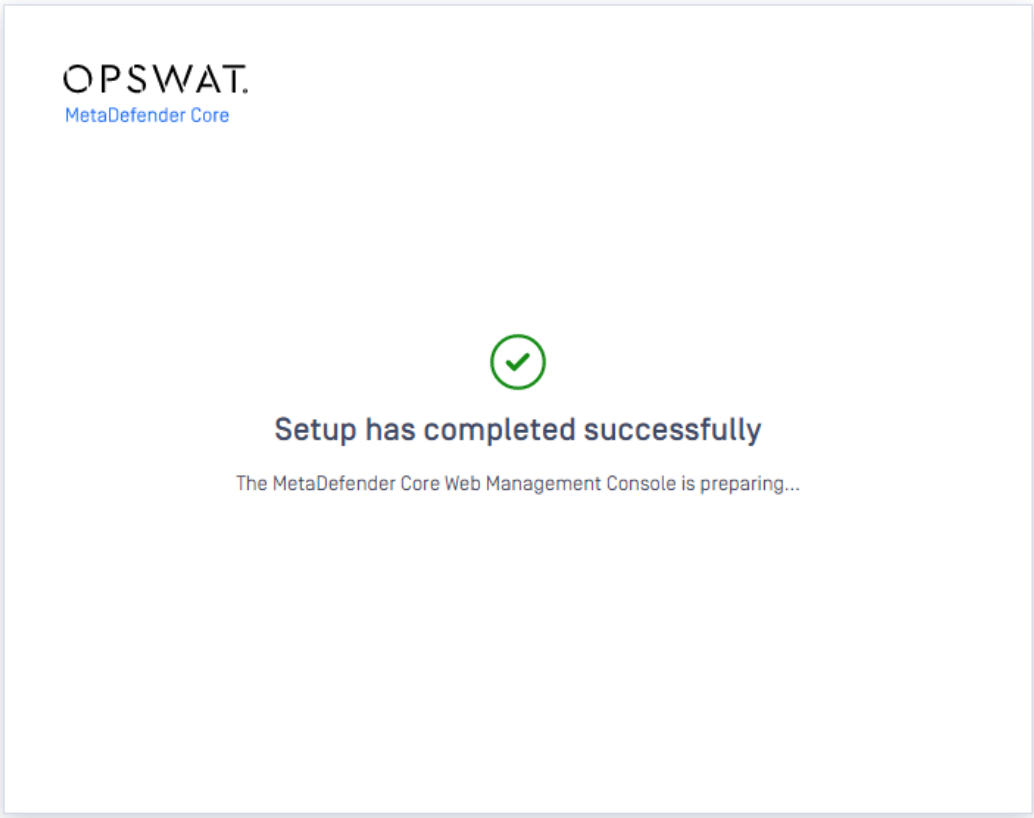


Wizard completion

After you have completed every steps you are ready to finish the wizard and start using the product. Click the `Fishish` button to complete the wizard.

> **Warning**
> The product's service will be restarted and the browser will be redirected to the administrative management console. This could take several seconds.

You can login to the Web Management Console with the administrator user that have just been created in the previous steps.



Setup HTTPS

Transport Layer Security (TLS) is a cryptographic protocol that provides communications security over a computer network. Websites, like the Web Management Console, are able to use TLS to secure all communications between their servers and web browsers.

The TLS protocol aims primarily to provide confidentiality (privacy) and data integrity between two communicating computer applications.

Warning

No TLS for the wizard

By default, TLS is not enabled for the basic configuration wizard. As a consequence sessions between the wizard's backend and the browser may be insecure.

Info

Performing the same steps as for the Web Management Console, it is possible to set up TLS for the basic configuration wizard. Remember completing the TLS setup before launching the wizard.

For instructions to set up TLS see Enabling HTTPS

License Activation

##

To activate your installation go to the **Settings > License** menu in the Web Management Console. If you have no valid license, you will only see your installation's **deployment ID**. You will also see a warning in the Web Management Console header.

After successful activation, the product will start downloading the latest available scan engines and malware databases. You can follow the status of the scan engine installation on the **Inventory > Modules** page.

When all the licensed engines are installed, you can start using the MetaDefender Core to process files - File Processing.

Deployment ID

Each MetaDefender Core instance represented by an unique system fingerprint called `Deployment ID`

MetaDefender Core licensing is purely based on deployment ID activated. Therefore, everytime your MetaDefender Core is activated with a different deployment ID, then it will be captured as a different activation record accumulated into your total activation times allowed.

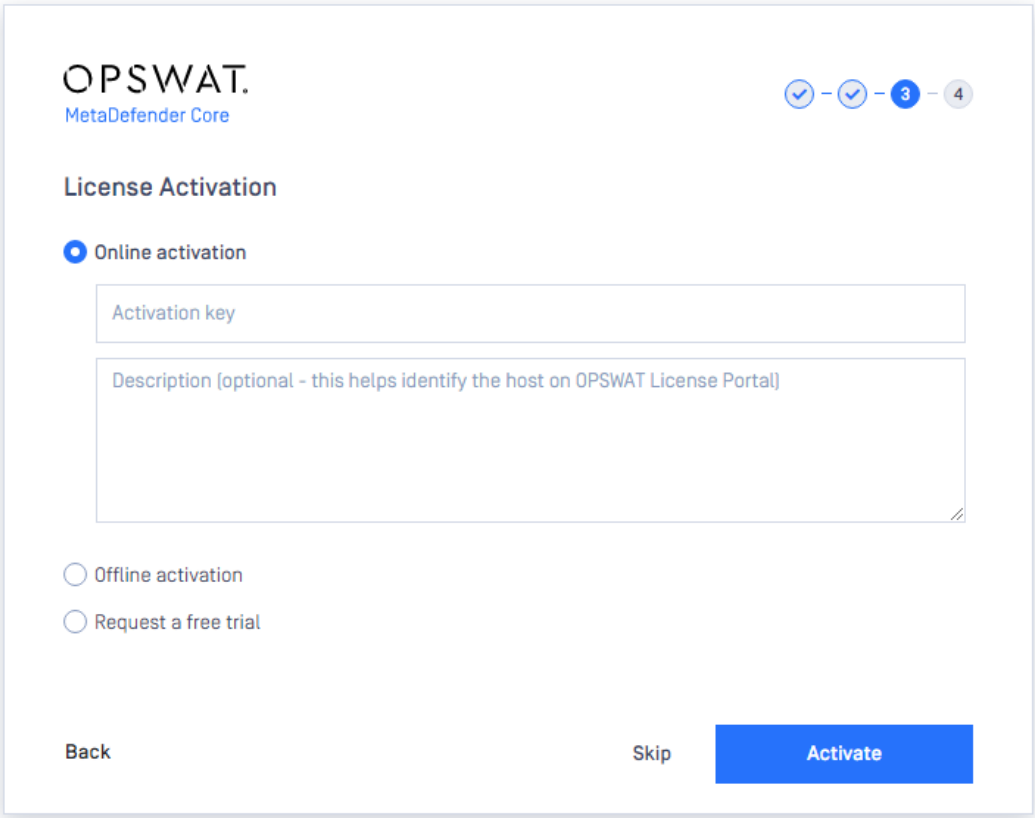
> **Notice**
> Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you alter the system where MetaDefender Core relies on (e.g. network card, OS hostname).
>
> Deployment ID is an UUID version 4 random based string. Learn more at: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

Deployment ID is auto generated by MetaDefender Core product upon installed, and it might be auto changed when you update the system where MetaDefender Core relies on.

Activation Modes

There are three different options for the product activation:

- Online activation: MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.
- Offline activation: Requiring you to upload a license file (.yml). That license file could be retrieved by activating your license via OPSWAT Portal. Follow the displayed instructions for details.
- Request trial key online: For evaluation purpose, you can receive an trial Activation key via email. Follow the displayed instructions for details.



Online activation

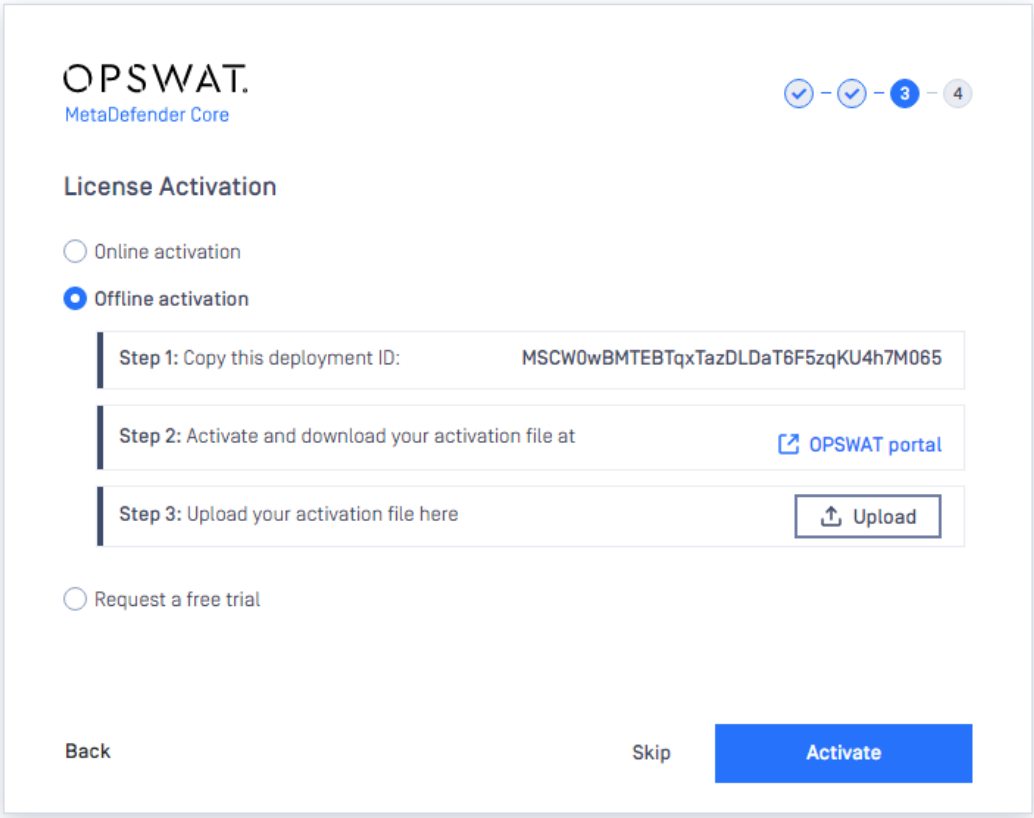
MetaDefender will connect directly to the OPSWAT licensing server online, and acquire its license based on your Activation key and its Deployment ID.

Required to input your Activation License key, and optional for description (but we recommend this for troubleshooting and auditing purpose if needed).

> **Warning**
> If you have installed or if you wish to use the MetaDefender Core in a restricted environment, you will have to allow access to the following hosts' for license activation:
>
> - https://activation.dl.opswat.com Note: *IP address-based whitelisting on your firewall might fail after some time since OPSWAT uses CDN (Content Delivery Network) to faster delivery updates over the world, and IP address of edge servers might change over time.*

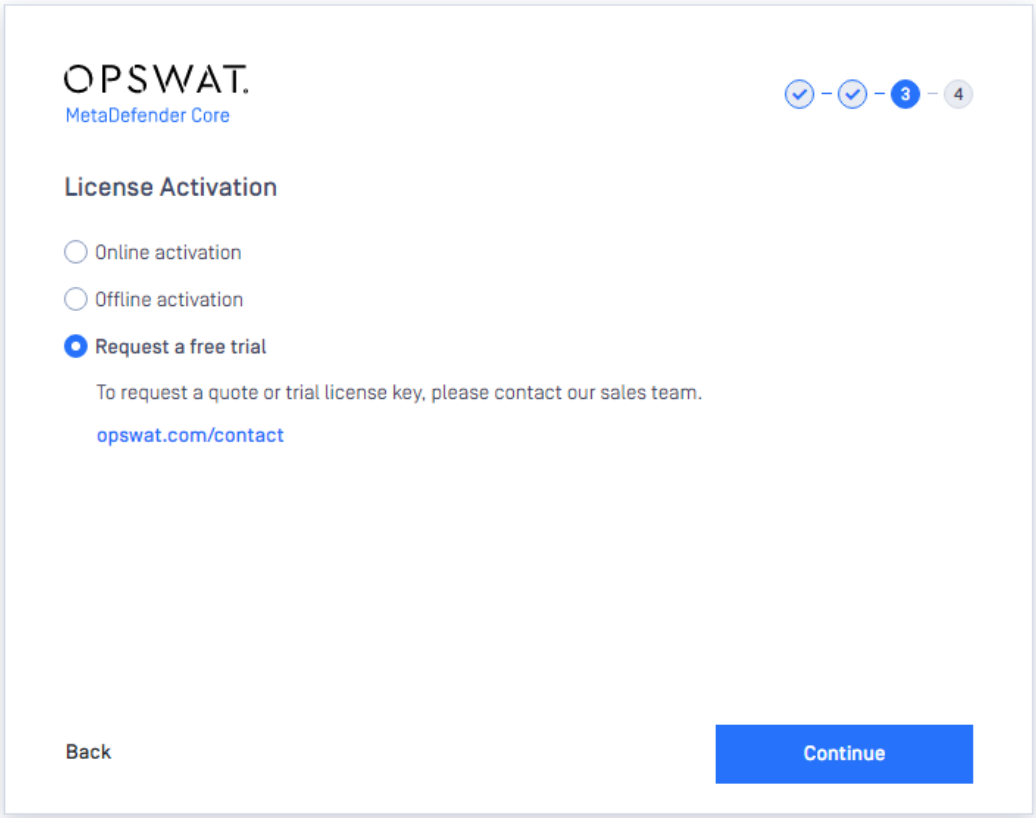
Offline activation

With no internet connection on the server the Metadefender v4 instance may be activated indirectly from a different machine, that has internet connection. The Deployment ID of the Metadefender v4 instance and the the Activation key received at the time of purchasing the product will be required. Follow the steps on the screen to activate the product offline.



Request trial key online

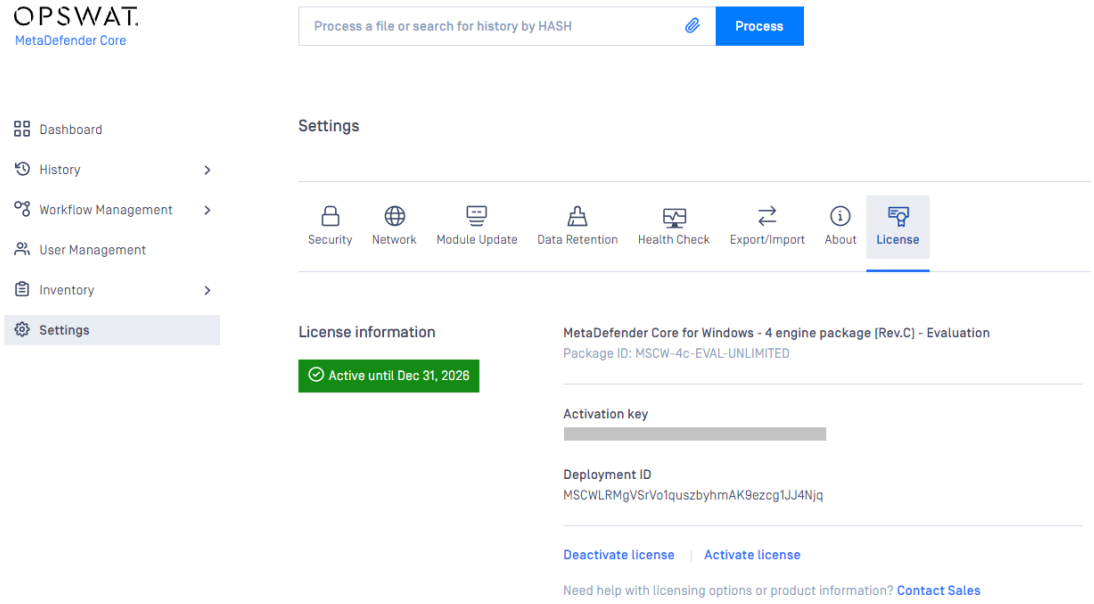
An evaluation license could be acquired by contacting our sales team.



Check Your License Details

For more license details and activating your MetaDefender Core, go to Settings > License menu tab on the Web Management Console:

- Product name: product name as on your order (e.g. MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation)
- Package ID: product identification as on your order (e.g. MSCW-4c-EVAL-UNLIMITED)
- Expiration: last day of license validity
- Activation key: key used for activating product
- Deployment ID: identification of this installation


OPSWAT.
MetaDefender Core
Process a file or search for history by HASH
Process
Dashboard
History
Workflow Management
User Management
Inventory
Settings
Settings
Security
Network
Module Update
Data Retention
Health Check
Export/Import
About
License
License information
Active until Dec 31, 2026
MetaDefender Core for Windows - 4 engine package (Rev.C) - Evaluation
Package ID: MSCW-4c-EVAL-UNLIMITED
Activation key
Deployment ID
MSCWLRMgVSrVo1quszbyhmAK9ezcg1JJ4Njq
Deactivate license
Activate license
Need help with licensing options or product information? Contact Sales

Scripted license management

Using REST API calls there is an option for scripted activation and deactivation for MetaDefender Core servers with **no Internet connection or if your infrastructure is using automation to create/destroy MetaDefender Core instances**. In this chapter the steps of these two scenarios are described.

Requirements

- an installed MetaDefender Core instance without Internet connection
- another computer that has Internet connection and can run your activation scripts
- a manual or automated way to transfer data between the two computers

Activation steps

1. In order to activate MetaDefender Core, deployment ID and activation key are needed to generate the license file.
    a. Activation key should be purchased from OPSWAT.
    b. MetaDefender Core deployment ID can be queried by using REST API (for details see GET Get Current License Information API page)

> **Info**
> Save this deployment ID in your system, you might need this ID when the instance is unavailable at the time of deactivation.

2. Activate license and get the license file using the following URL, where <quantity> is 1.

activation call bash

```
curl -X GET https://activation.dl.opswat.com/activation?key=<activation-key>&deployment=<deployment unique-ID>&quantity=<quantity>
```

If the activation is successfully, a license file is downloaded. Save this file.

HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	license file	Activation was successfully.
200 Activation failed	error: '<user conform error message>'	Failed activation
200 Invalid parameter	error: 'Could not activate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not activate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
200 Invalid parameter	error: 'Could not activate your product because the quantity you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key, quantity or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Info
In case of any activation issue, contact OPSWAT support for help

3. Upload license file to MetaDefender Core: The license file should be uploaded to the MetaDefender Core v4 to activate the product. For details see page: POST Upload license key API

Deactivation steps

1. For deactivation, following infomation is required:
    a. Activation key
    b. MetaDefender Core deployment ID: can be queried by using REST API (for details see GET Get Current License Information API page)
2. Deactivate license using the following URL:

deactivation call bash

```
curl -X GET https://activation.dl.opswat.com/deactivation?key=<activation-
key>&deployment=<deployment unique-ID>
```

Response HTTP status codes can be:

HTTP Response	Body	Comment
200 Ok	result: ok	Successful deactivation
200 No active license found	error: 'Could not found any active license with the given parameters'	The license has not been activated yet or it has been deactivated already.
200 Invalid parameter	error: 'Could not deactivate your product because the Activation Key you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid key format
200 Invalid parameter	error: 'Could not deactivate your product because the Deployment ID you provided is invalid. Check if you typed it correctly or open a support ticket if problem persist.'	Invalid deployment format
400 Bad request		Missing key or deployment
500 Internal server error	error: 'Internal server error (<error ID>). Please contact support'	

Important notes

Warning

Product activation is tied to several hardware, operating system and software parameters. In case of one or more major hardware or software parameter change the product might turn into deactivated status. Operating system updates, other software updates should not affecting the activation status.

If the product is online activated and has live Internet connection then in case of deployment ID change the product reactivates itself. If the product doesn't have Internet connection then the administrator is responsible to reactivate the product.

Because of the above administrator should consider licensing restrictions of the offline deployments if using any virtualization and/or containerization technologies.

File Processing

MetaDefender Core provides REST API interface for file processing business.

There are several ways to process files with MetaDefender Core:

- Analyze Files via Web Interface
- Process Files via REST API

Product Upgrading

> **Info**
> Simply run installer file, details could be found at Installation

> **Info**
> For command line mode, upgrading (e.g. on **Linux**), prior to running the installer file, please make sure to **create the ignition file** with PostgreSQL server information included. Details:
>
> - Standalone DB mode: Command Line Installation (Windows and Linux)
> - Shared DB mode: Command Line Installation

Recommendation for MetaDefender Core upgrade

Our goal is to help you execute the product upgrade continuously, efficiently, and correctly.

PostgreSQL service upgrade

- **Bundled PostgreSQL Service**: If you are using the bundled PostgreSQL server included in the installer, the PostgreSQL service will be automatically upgraded to the newer version (if available). No additional action is required for this process.
- **Remote PostgreSQL Service**: For remote PostgreSQL servers, customers are responsible for managing and upgrading the service independently. Upgrading MetaDefender Core does not update the remote PostgreSQL service.

Before the upgrade

Prepare a proper disaster recovery plan (DRP) including some common practices:

- Backup, or take a snapshot of your current working MetaDefender Core server.
  - This is to help quickly rollback for any upgrade failure might cause later for any reason.
- Optionally backup your PostgreSQL database, especially when using remote shared database mode.
  - Reference: https://www.postgresql.org/docs/current/backup.html
  - Please do it after stopping all linked MetaDefender Core instances to avoid potential data collision.
- Lastly, if backing up entire server is not feasible, then at least export MetaDefender Core configuration package Import/Export configuration just in case you'll need to start off a new server with current MetaDefender Core configuration to keep the scanning service running without any impact.

On the PostgreSQL server, double check server resource to ensure sufficient RAM memory and disk space to be preserved for the upgrade process.

- 8GB RAM available at least during the upgrade.

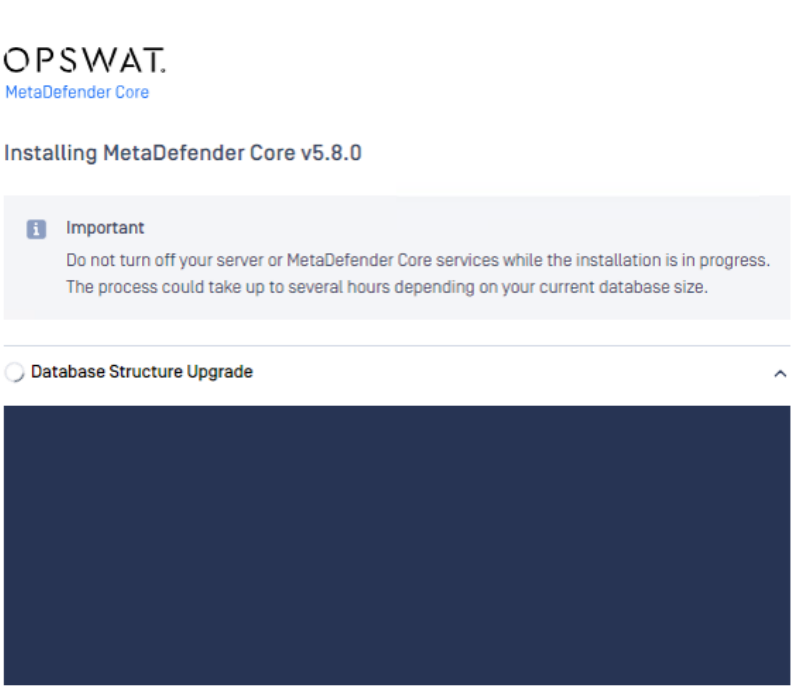
- Disk availability is greater than MetaDefender Core database size.

While doing the upgrade

Applicable for shared database deployment:

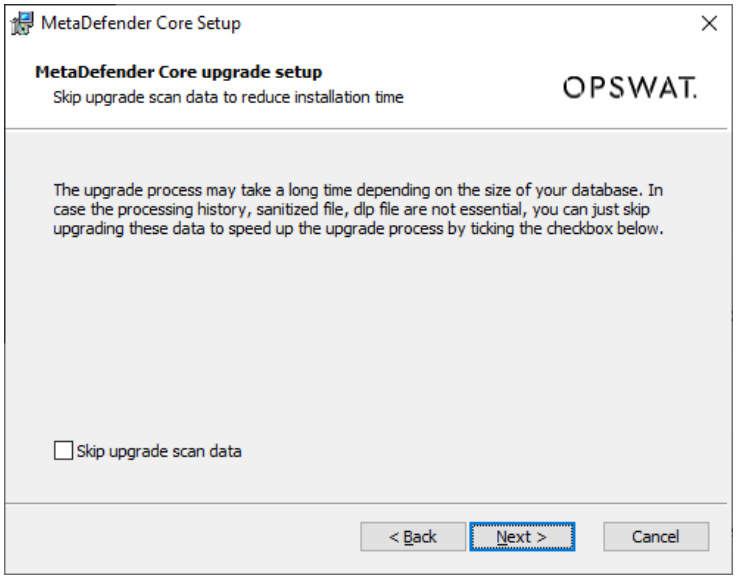
- Stop all MetaDefender Core instances' scanning service before performing the upgrade.
- Upgrade one (1) MetaDefender Core instance at a time.
- Proceed to upgrade MetaDefender Core instance where the bundle local PostgreSQL server is used for shared database setup.
- Otherwise is using remote PostgreSQL then the upgrade order does not matter.

For large size database, mostly for scan history, expecting long time to be spent on the upgrade process.



The product supports an option to skip data migration for scan history to expedite the process drastically and mitigate any failure risk. You can manually backup all data from your PostgreSQL server for offline usage if any.

Check "Skip upgrade scan data" to enable this feature while doing the upgrade.



or add following key in the ignition file:

none

```
[internal]
skip_upgrade_scan_data = 0
```

Troubleshooting

If the upgrade process ever runs into an error:

- Double check your network and services, applicable to remote PostgreSQL server.
- Optionally configure MetaDefender Core to skip migrating scan history.
  - Windows
    - Open Windows registry, and navigate to HKLM\Software\OPSWAT\Metascan
    - Add a new key `internal/skip_upgrade_scan_data` with value = 1
  - Linux
    - Open MetaDefender Core configuration file `/etc/ometascan/ometascan.conf`
    - Add new key into the file
      - [internal]
      - skip_upgrade_scan_data=1
  - Note: This key will be auto removed upon the upgrade finishes.
  - Then restart MetaDefender Core service to retry the upgrade.
    - Might retry this a few times until the upgrade is done.

Just in case the upgrade process still fails, please contact OPSWAT support with the support package - How to Create Support Package? This will help us analyze the root cause and figure out an optimal solution for your system failure.

Using external Load Balancer

Application Load Balancing (Layer 7)

This page provides information about recommended ways to use sessions with a Layer 7 load-balancer.

Most load-balancers have the ability to provide cookies for pinning subsequent traffic from a client to the appropriate server. This method is called sticky session, session persistence or session affinity. Using cookies efficiently requires the client to know when a cookie should be sent, should not be sent or should be deleted.

Sticky session load balancing

Each Metadefender Core instance has its own databases and application sessions that cannot be seen by other instances. Therefore, in order to get the related data/response to our queries we should ask the appropriate Metadefender Core v5 server. On the other hand, to keep the advantages of the used load balancing method, cookies should not be sent if it is not necessary.

Single file scanning

Step	Stage	Task	Cookie usage
1.	Sending file	Initiate processing a file on the client side. Send file through the load-balancer. (See POST File Analysis API)	Cookie should not be sent
2.	Sending file	Save the cookie and the data_id you got from the load-balancer	Save cookie
3.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Send cookie saved in step 2
4.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Send cookie saved in step 2

Batch scanning

Step	Stage	Task	Cookie usage
1.	Open batch	Initiate processing file(s) in batch. Request a batch ID through the load-balancer. (See POST Initiate Batch API)	Cookie should not be sent
2.	Open batch	Save the cookie and the batch_id you got from a Core server through the load-balancer.	Save cookie
3.	Sending files	Send file through the load-balancer. (See POST File Analysis (using batch header) API)	Send cookie saved in step 2
4.	Sending files	Save the data_id you got from the load-balancer.	-
	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Send cookie saved in step 2
5.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
6.	Getting batch status	See GET Batch Status API	Send cookie saved in step 2
7.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Send cookie saved in step 2
8.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API	Send cookie saved in step 2

If it does not matter which upstream server responds, the queries should be sent without cookie.

> **Info**
> It is recommended not to send cookies when it's not necessary to allow load-balancer to use its own method to share the load between Metadefender Core v5 servers.

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration through load-balancer

OPSWAT products that support HTTP load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	4.3.4	-
OPSWAT Client	Windows: 7.6.247.0 Mac: 10.4.243.0	-
MetaDefender Email Security	4.3.0	-
MetaDefender ICAP Server	4.3.0	-
MetaDefender Vault	1.3.0	-
MetaDefender for Secure Storage	2.0.7	-

DNS Load Balancing

Using this method is logically similar to Layer 7 load-balancing.

Briefly how it works

Client uses a domain name to send a query to a server. Client's DNS server has more "A" records for that name with different IPs. When a client resolves the server's domain name DNS server randomly chooses an IP for that name to send back. When a session is used on the application layer, client should know the IP address of the Core server that handles that specific session. In every other case, client should resolve the domain name with DNS query to let requests to be balanced between Metadefender Core servers.

Single file scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Sending file	Initiate processing a file on the client side. (See POST File Analysis API)	Use IP saved in step 2
4.	Sending file	Save the data_id got from the Core server	-
5.	Getting result	Request result related to data_id saved in step 2. (See GET File Analysis API)	Use IP saved in step 2
6.	Getting result	If processing is in progress (See GET File Analysis API), wait a little while and repeat step 3.	Use IP saved in step 2

Batch scanning

Step	Stage	Task	Addressing
1.	Choose a Core server by using DNS load balancing	Resolve the Core servers' common domain name.	Use domain name
2.	Choose a Core server by using DNS load balancing	Save the IP gotten from the DNS server.	Save the IP
3.	Open batch	Initiate processing file(s) in batch. Request a batch ID. (See POST Initiate Batch API)	Use IP saved in step 2
4.	Sending files	Send file to the specific Core server with the basch ID saved in step 3. (See POST File Analysis (using batch header) API)	Use IP saved in step 2
5.	Sending files	Save the data_id you got from the Core server.	-
6.	Sending files	Status/result of scanning of sent files can be queried: (See GET File Analysis API)	Use IP saved in step 2
7.	Sending files	Repeat step 3-4. with files wanted to be in the same batch.	-
8.	Getting batch status	See GET Batch Status API	Use IP saved in step 2
9.	Close batch	Tell the server that no more files will be sent to this batch. (See POST Close Batch API). (This will only be successful if all the files sent to the batch have been processed already. Repeat this step until batch is closed.)	Use IP saved in step 2
10.	Getting results	Request results related to batch ID saved in step 2. (See GET Download Batch Signed Result API).	Use IP saved in step 2

Limitations, additional notes

Using load-balancing between Metadefender Core servers does not support:

- Global scan history
- Core server administration via DNS load-balancing

OPSWAT products that support DNS load balanced MetaDefender Cores

Product name	Minimum version	Further information
MetaDefender Kiosk	does not support yet	-
MetaDefender Client	does not support yet	-
MetaDefender Email Security	does not support yet	-
MetaDefender ICAP Server	does not support yet	-
MetaDefender Vault	does not support yet	-

SSL connection for PostgreSQL communication

> **Note**
> The SSL connection configuration is possible thanks to the support made by PostgreSQL. For reference: https://www.postgresql.org/docs/14/ssl-tcp.html
>
> MetaDefender Core is tested against this mode on particular version 4.21.1

Step 1: Generate SSL certificate for PostgreSQL server

> **Note**
> Following instructions are for self-sign certificate. For production environment, making sure to use the certificate signed by your trusted CA.

1) Generate a private key

bash

```
openssl genrsa -des3 -out server.key 1024
```

Sample output:

bash

```
Generating RSA private key, 1024 bit long modulus ........++++++
..............++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:
Verifying - Enter pass phrase for server.key:
```

Type your desired password / pass phrase to encrypt the private key

2) Remove the pass phrase to automatically start up the PostgreSQL server

bash

```
openssl rsa -in server.key -out server.key
```

> **Note**
> On Linux, make sure to set permission on the server.key file

bash

```
chmod og-rwx server.key
```

3) Create a self-signed certificate

bash

```
openssl req -new -key server.key -days 3650 -out server.crt -x509
```

You will be prompted to enter detailed information which is incorporated into your self-signed certificate request.

none

```
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:CA
Locality Name (eg, city) [Default City]:San Francisco
Organization Name (eg, company) [Default Company Ltd]:OPSWAT
Unit Name (eg, section) []:OPSWAT
Common Name (eg, your name or your server's hostname) []:localhost
Email Address []:dummy_email@opswat.com
```

For self-signed certificate, use the server ceriticate as the trusted root certificate:

bash

```
cp server.crt root.crt
```

Copy **server.key**, **server.crt**, **root.crt** to PostgreSQL data folder. For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data**
- Linux: **/var/lib/ometascan/pg_data/**

Step 2: Configure PostgreSQL server for SSL authentication connection

1) Create a custom config file (e.g. ssl.conf) for PostgreSQL.

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\postgres\ssl.conf**
- Linux: **/var/lib/ometascan/postgres/ssl.conf**

2) Content of ssl.conf:

none

```
ssl = on
ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL'
ssl_prefer_server_ciphers = on
ssl_cert_file = 'server.crt'
ssl_key_file = 'server.key'
ssl_ca_file = 'root.crt'
```

3) Modify pg_hba.conf

For bundled local MetaDefender Core's PostgreSQL:

- Windows: **<Installation folder>\data\pg_data\pg_hba.conf**
- Linux: **/var/lib/ometascan/pg_data/pg_hba.conf**

Add following (modify **::1/128** if needed to match with your environment requirement)

none

```
hostssl all all      ::1/128      scram-sha-256
```

After the modify, the content of pg_hba.conf should be like this (example):

> **Note**
> In a Linux environment, a local connection is required to upgrade from version 5.5.1 or earlier to version 5.6.0 or later.

none

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# Refer to the "Client Authentication" section in the PostgreSQL
# documentation for a complete description of this file.  A short
# synopsis follows.
#
# This file controls: which hosts are allowed to connect, how clients
# are authenticated, which PostgreSQL user names they can use, which
# databases they can access.  Records take one of these forms:
#
# local         DATABASE  USER  METHOD  [OPTIONS]
# host          DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostssl       DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnossl     DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostgssenc    DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
# hostnogssenc  DATABASE  USER  ADDRESS  METHOD  [OPTIONS]
local    all all                    scram-sha-256
hostssl all all      ::1/128      scram-sha-256

#More content.......
```

4) Reload the services

- Standalone DB mode: Restart **ometascan** service (MetaDefender Core service)
- Shared DB mode: Restart **ometascan-pg** / **ometascan-postgresql** service (MetaDefender Core PostgreSQL service) and then **ometascan** service (MetaDefender Core service)

Test your SSL database connection:

bash

```
psql.exe "sslmode=require dbname=postgres user=postgres host=localhost port=5432"
Password for user postgres:
psql (12.7)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
Type "help" for help.
```

This message indicate that you are now using SSL connection:

bash

```
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
compression: off)
```

Performance and Load Estimation

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Some factors that affect performance

- MetaDefender Core version
- MetaDefender Core engine package and configuration
  - set of engines (which and how many)
  - product configuration (e.g., thread pool size)
- System environment
  - server profile (CPU, RAM, hard disk)
  - client application location - remote or local
  - system caching and engine level caching
- Dataset
  - encrypted or decrypted
  - file types
    - different file types (e.g., document, image, executable)
    - archive file or compound document format files
  - file size
  - bad or unknown (assume to be clean)
- Performance tool

How test results are calculated

Performance (mainly scanning speed) is measured by throughput rather than unit speed. For example, if it takes 10 seconds to process 1 file, and it also takes 10 seconds to process 10 files, then performance is quantified as 1 second per file, rather than 10 seconds.

- total time / total number of files processed: 10 seconds / 10 files = 1 second / file.

Standalone DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.2.416.0

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

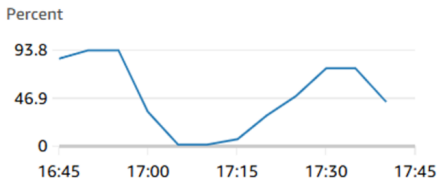
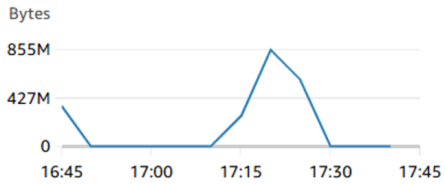
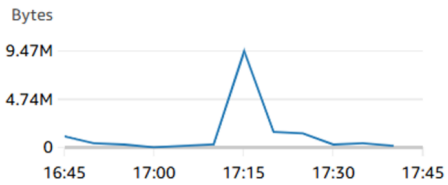
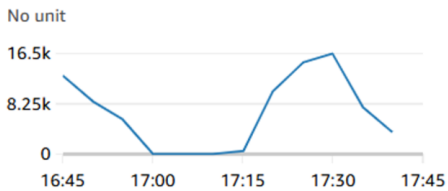
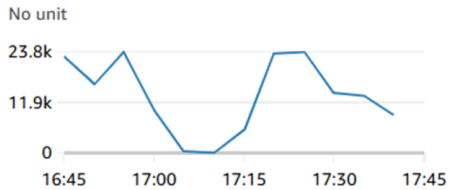
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV)	24.2	47.2 / 80	37.3 / 41.1		315,310	0.0114
Deep CDR	19.9	48.9 / 81.8	32.6 / 37		383,442	0.0094
Proactive DLP	17.1	36.8 / 58.2	30.4 / 33.6		446,228	0.0081
Vulnerability	10.1	59.2 / 85	29.3 / 33.3		755,495	0.0048

MetaDefender Core with common engine packages:

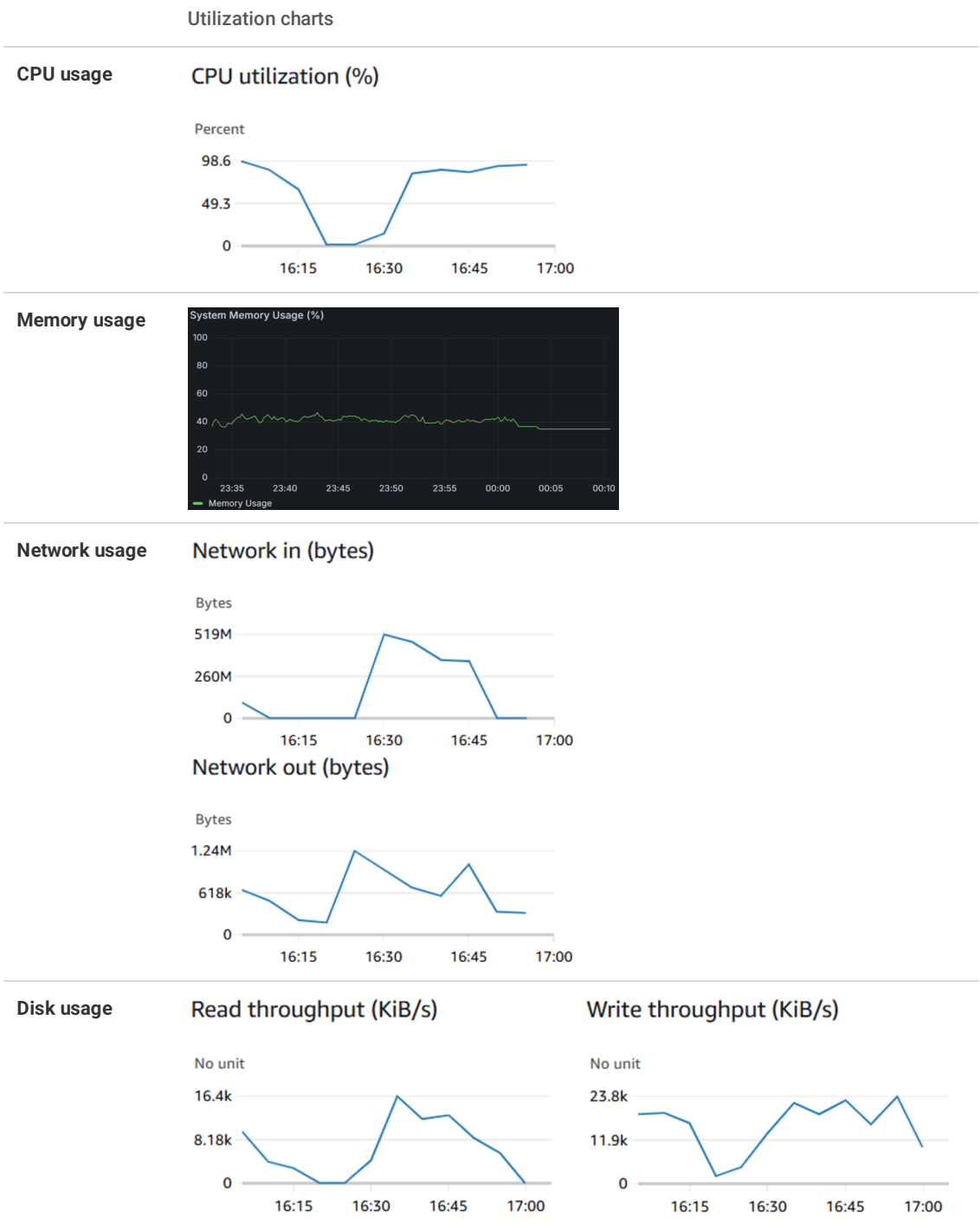
Use case	Scan duration	Avg./Max CPU usage	Avg./Max RAM usage	Avg./Max Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (8 AV) + Deep CDR	30.8	68.1 / 88	40.1 / 46.8		247,744	0.0145
Metascan (8 AV) + Deep CDR + Proactive DLP	31.2	74.2 / 98.6	41.5 / 51.1		244,567.3	0.0147
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	31.8	73.5 / 98	40.6 / 51.5		239,953	0.015

System resource utilization charts

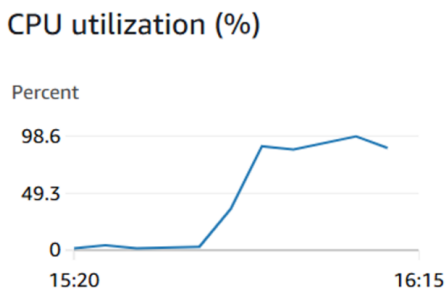
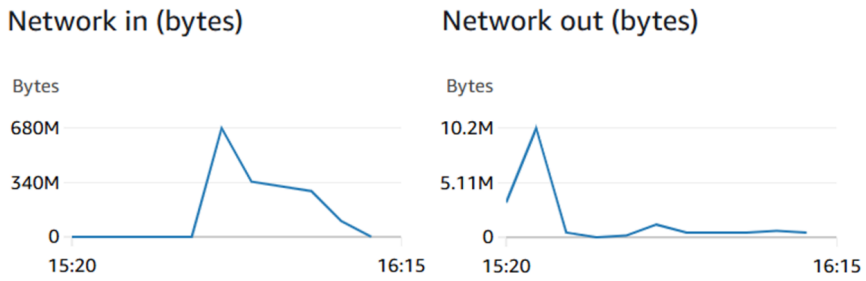
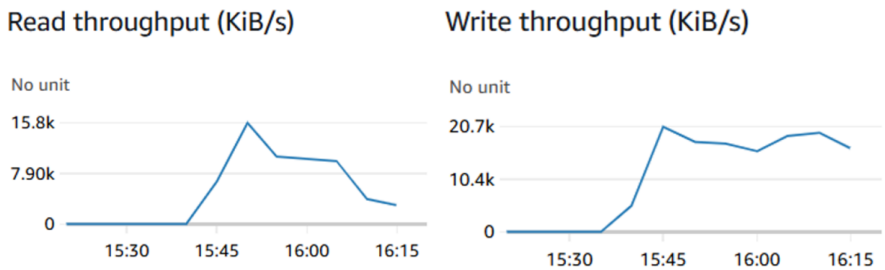
MetaDefender Core with Metascan (8 AV engines):

	Utilization charts
CPU usage	CPU utilization (%) 
Memory usage	
Network usage	Network in (bytes)  Network out (bytes) 
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

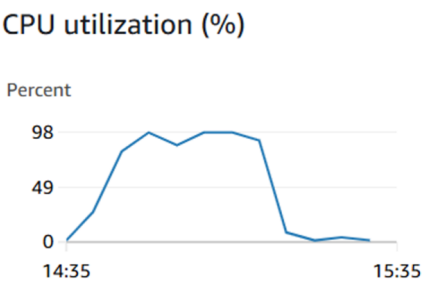
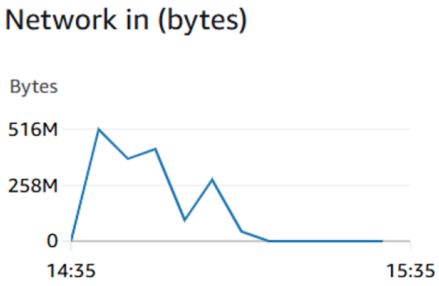
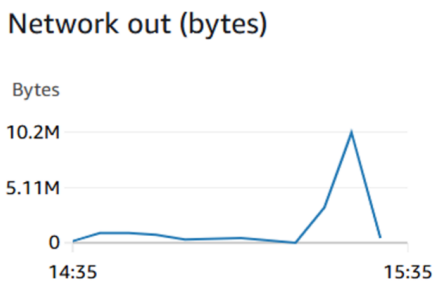
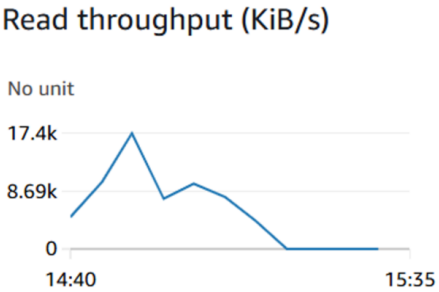
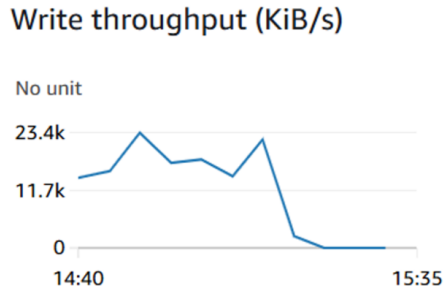
MetaDefender Core with Metascan (8 AV engines) + Deep CDR:



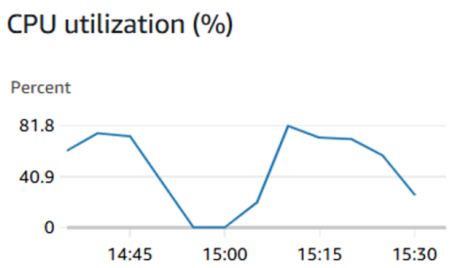
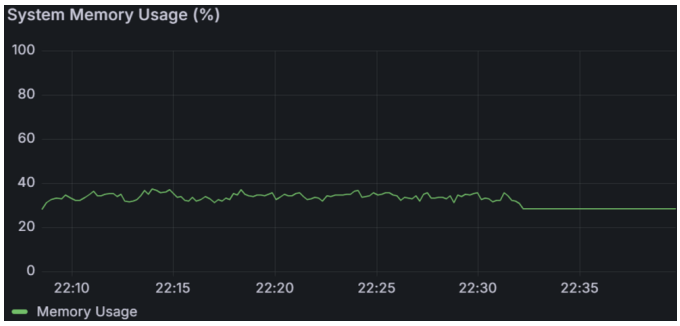
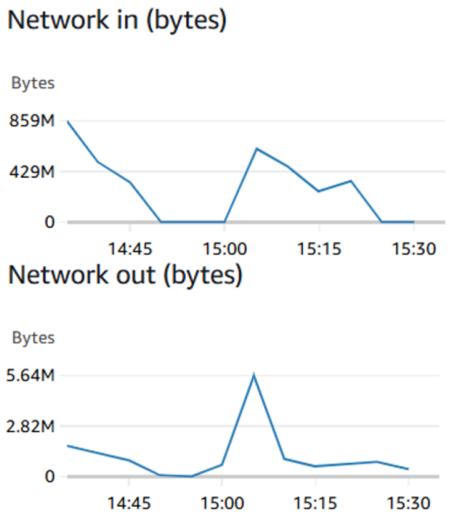
MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

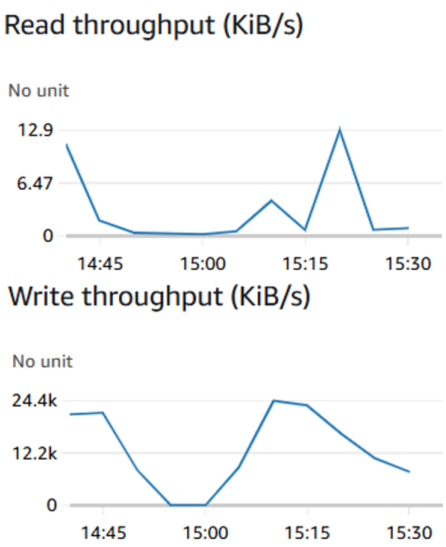
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

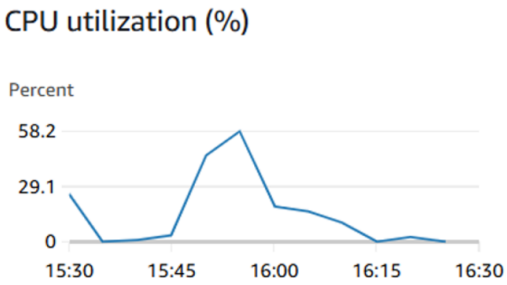
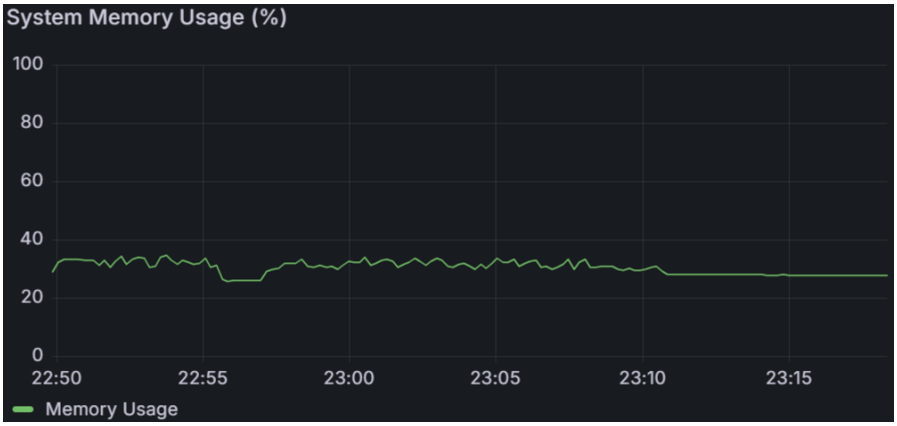
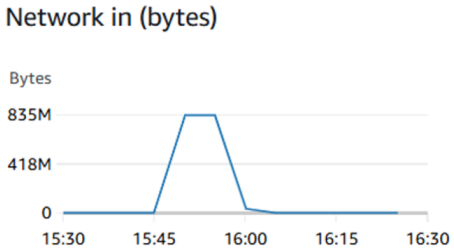
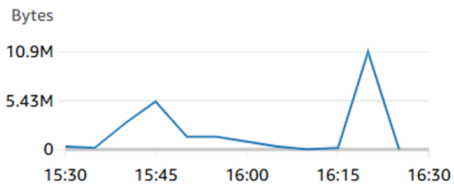
	Utilization charts
CPU usage	
Memory usage	
Network usage	 
Disk usage	 

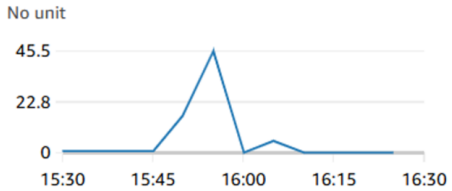
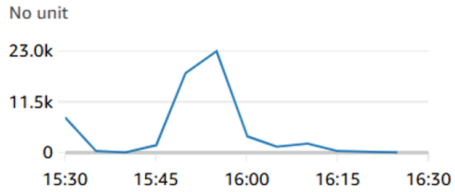
MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

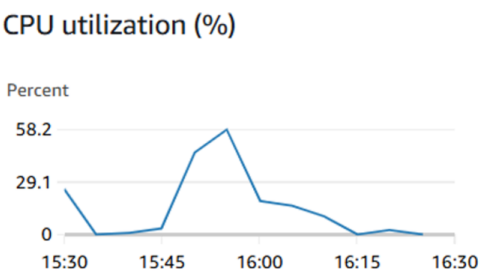
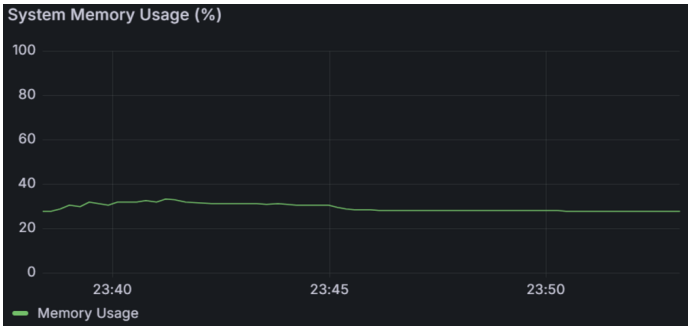
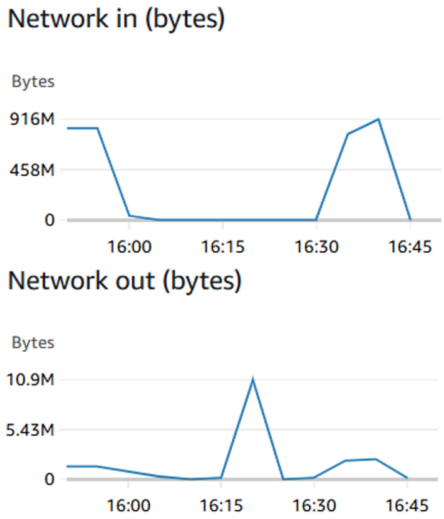
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

	Utilization charts
CPU usage	
Memory usage	
Network usage	 

	Utilization charts
Disk usage	Read throughput (KiB/s)  Write throughput (KiB/s) 

MetaDefender Core with Vulnerability:

	Utilization charts
CPU usage	
Memory usage	
Network usage	

Utilization charts

Disk usage



Standalone DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

MetaDefender Core

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type	Benchmark
Ubuntu 22.04	c5.4xlarge	16	32	Up to 10	SSD	Amazon EC2 c5.4xlarge - Geekbench

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to MetaDefender Core.

python

```
files_to_scan = list of files to scan
scan_futures = [asyncio.create_task(self.scan_file(file_path)) for file_path in
files_to_scan]

async def scan_file(self, file_path):
    api_url = f'{self.core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            response = requests.post(api_url, data=file_content, headers=headers)
            status_code = response.status_code
            if status_code == 200:
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.12.1

- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 7.2.3
  - Proactive DLP: 2.21.1
  - Archive: 7.2.1
  - File type analysis: 7.2.2
  - File-based vulnerability assessment: 4.57-236

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan

- Max file size: 99999999
- Scan timeout: 10 minutes
- Per engine scan timeout: 1 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

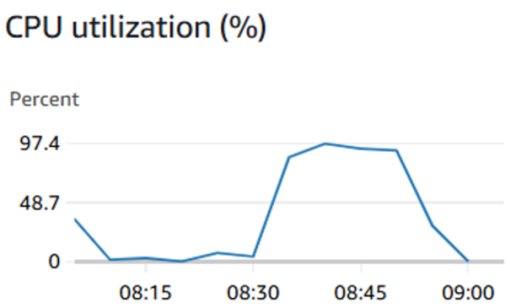
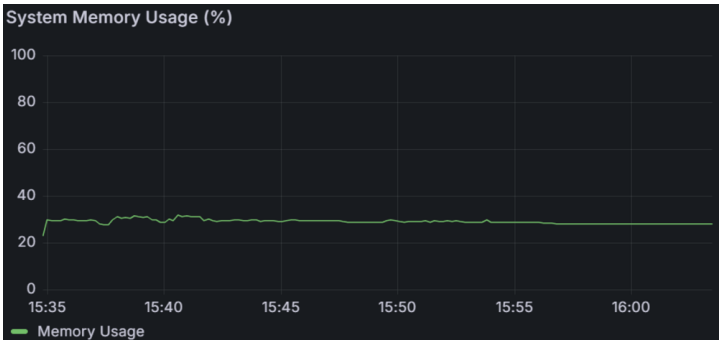
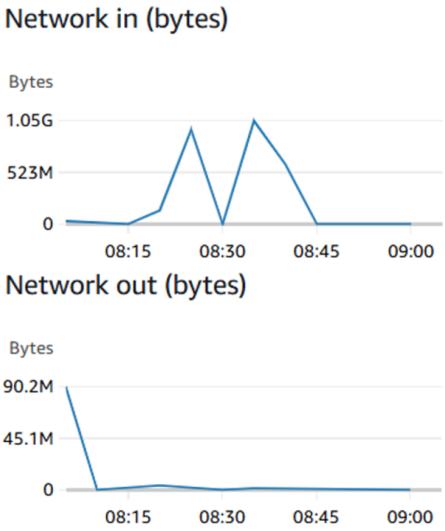
Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV)	30.2	69 / 98	29 / 32		252,666	0.0142
Deep CDR	19.4	48 / 62	36 / 49		393,325	0.009
Proactive DLP	16.5	34 / 39	32 / 48		462,455	0.008
Vulnerability	12	50 / 65	46 / 46		635,875	0.006

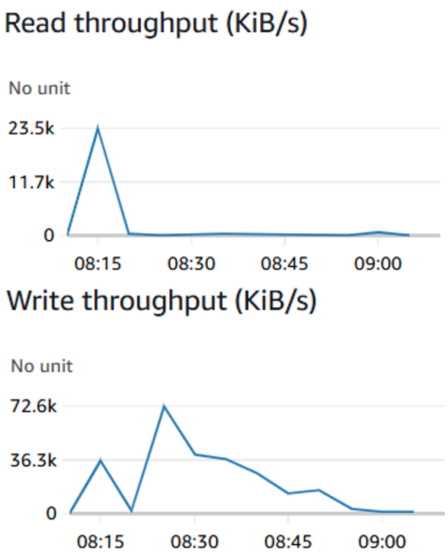
MetaDefender Core with common engine packages:

Use case	Scan duration	Avg./Max CPU usage	Avg./Max Memory usage	Avg. Network speed	Throughput	Avg. processing time
	(minutes)	*(%)*	*(%)*	*(KB/s)*	*(objects/hour)*	*(seconds/object)*
Metascan (10 AV) + Deep CDR	39.8	78 / 99	62 / 73		191,721	0.0188
Metascan (10 AV) + Deep CDR + Proactive DLP	43.5	61 / 84	47 / 67		175,414	0.0205
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	44.8	62 / 88	69 / 83		170,324	0.0211

System resource utilization charts

MetaDefender Core with Metascan (10 AV engines) :

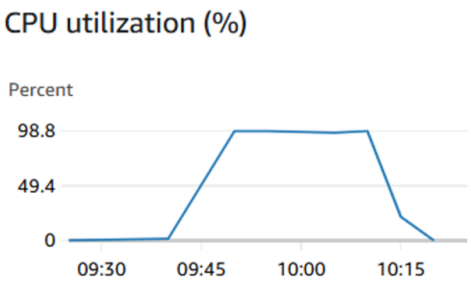
	Utilization charts
CPU usage	
Memory usage	
Network usage	

	Utilization charts
Disk usage	

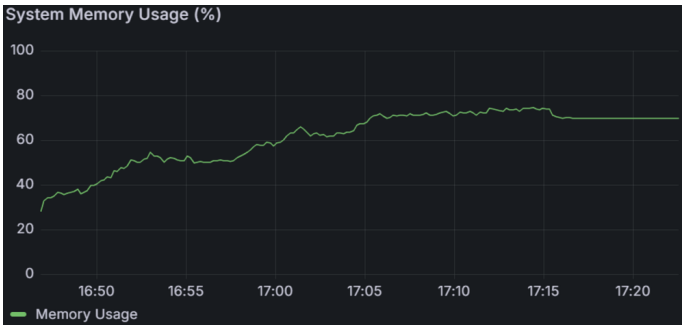
MetaDefender Core with Metascan (10 AV engines) + Deep CDR :

Utilization charts

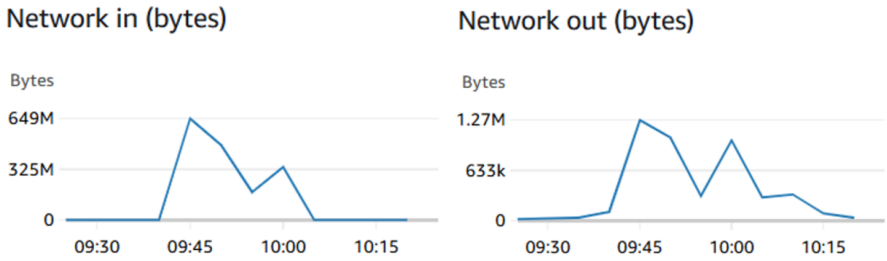
CPU usage



Memory usage



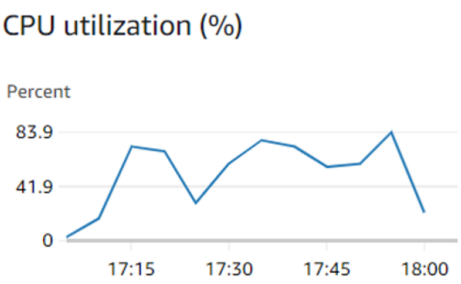
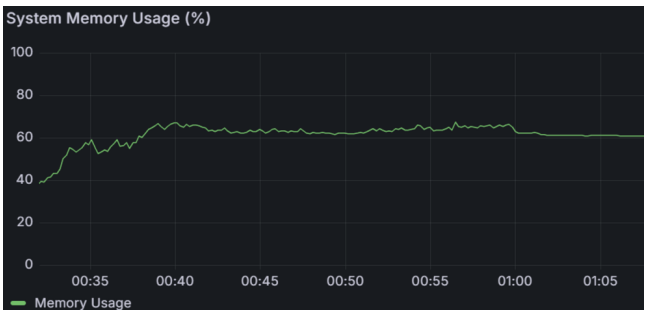
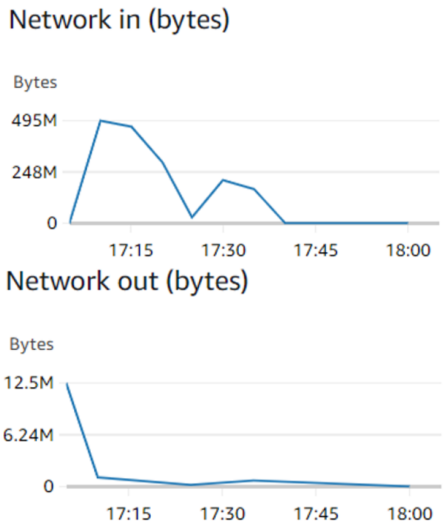
Network usage

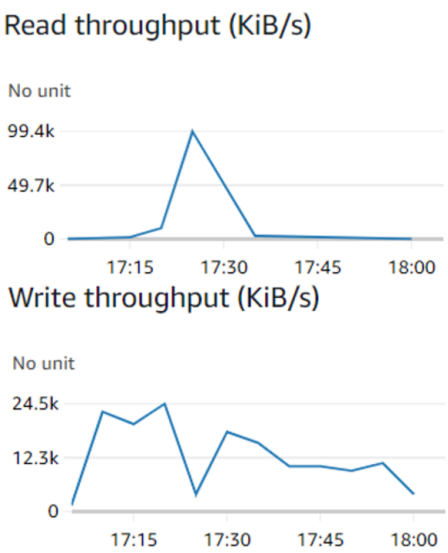


Disk usage

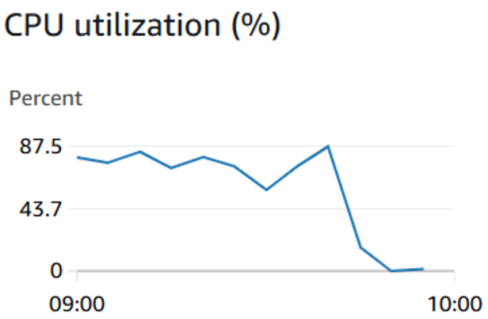
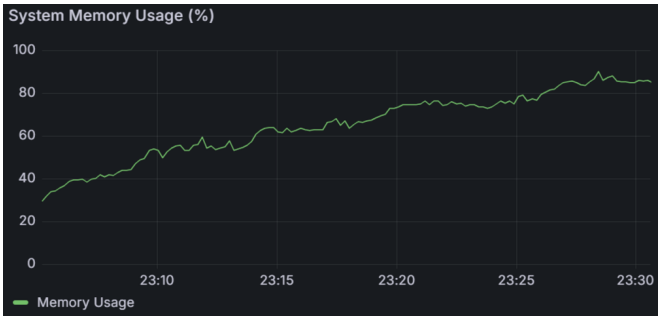
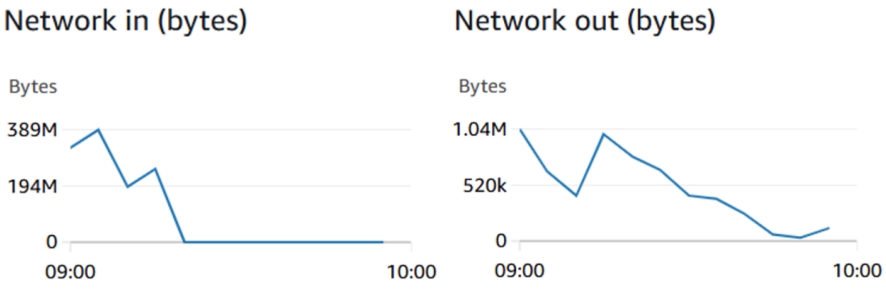


MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP :

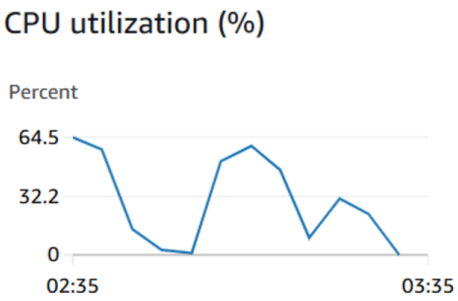
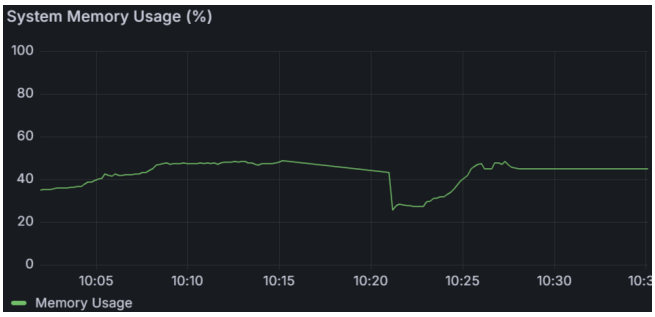
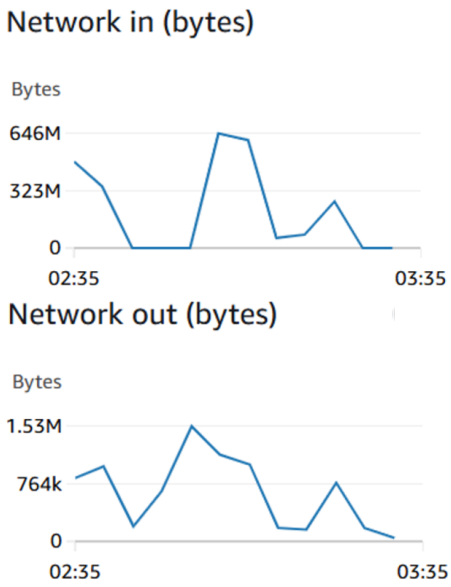
	Utilization charts
CPU usage	
Memory usage	
Network usage	

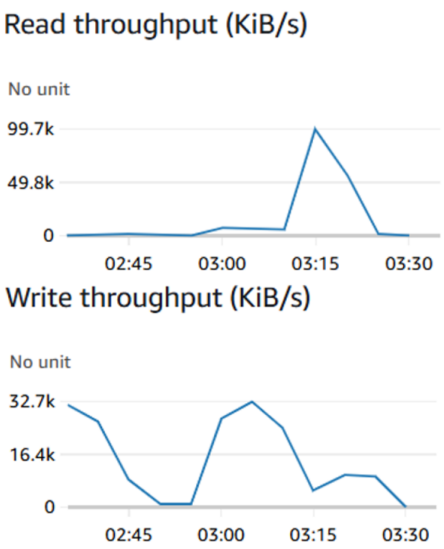
	Utilization charts
Disk usage	

MetaDefender Core with Metascan (10 AV engines) + Deep CDR + Proactive DLP + Vulnerability :

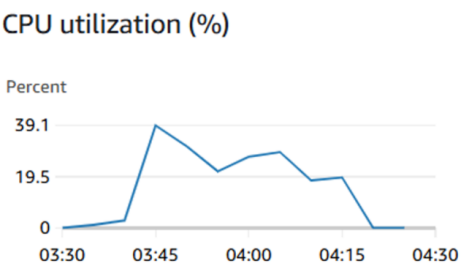
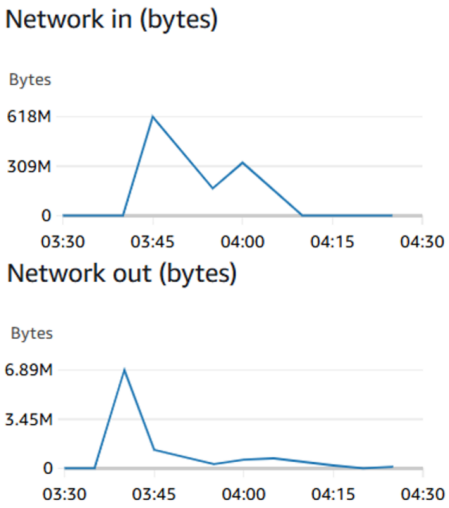
	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

MetaDefender Core with Deep CDR :

	Utilization charts
CPU usage	
Memory usage	
Network usage	

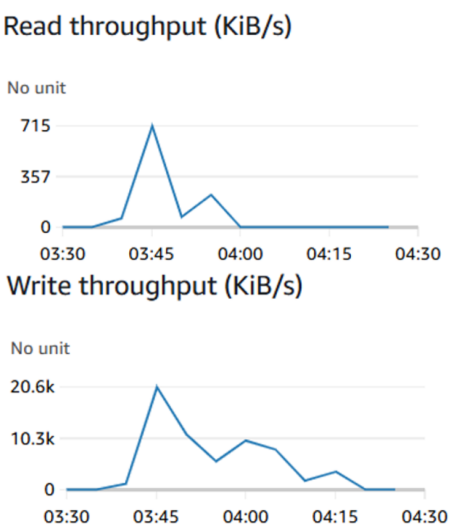
	Utilization charts
Disk usage	

MetaDefender Core with Proactive DLP :

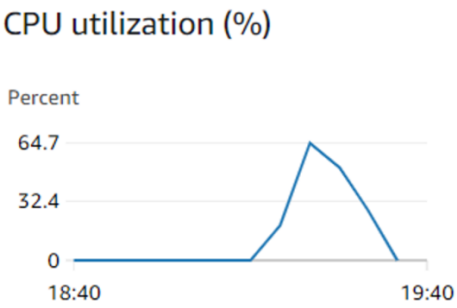
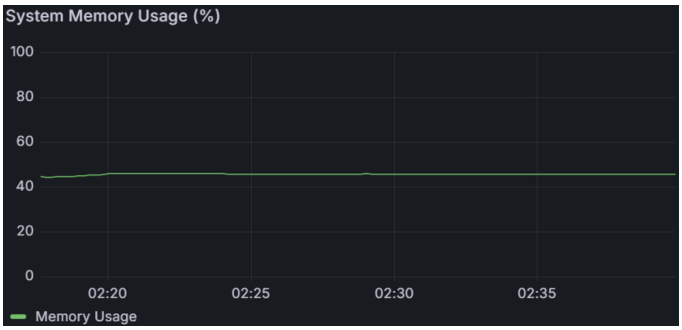
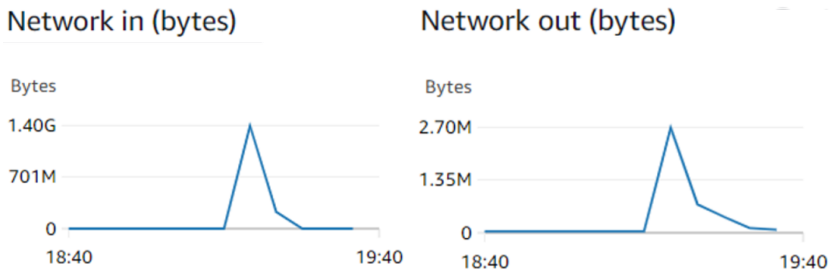
	Utilization charts
CPU usage	
Memory usage	
Network usage	

Utilization charts

Disk usage



MetaDefender Core with Vulnerability :

	Utilization charts
CPU usage	
Memory usage	
Network usage	
Disk usage	

Shared DB - Windows

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

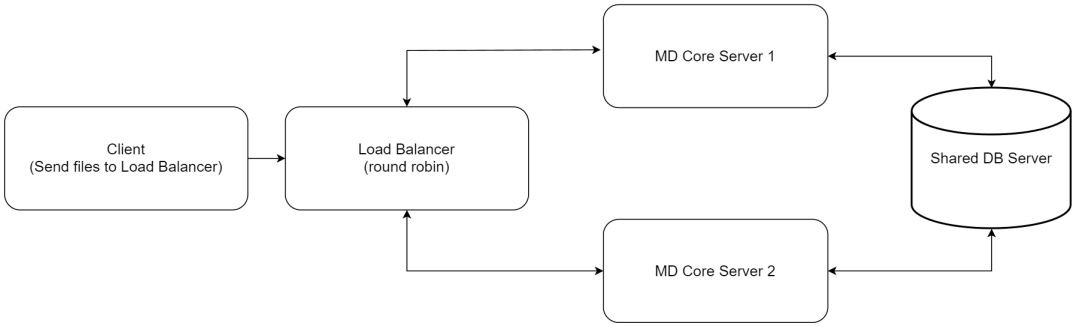
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 8 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, VirIT Explorer
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV)	11.9	37.0	41.9	30.1	30.4	Receive: 6441.1 Send: 769.6	Receive: 6200.2 Send: 693.7	640,821.2	0.0
Deep CDR	10.5	60.1	66.7	30.3	32.2	Receive: 6379.8 Send: 594.8	Receive: 7146.1 Send: 609.6	729,368.5	0.0
Proactive DLP	7.8	38.7	37.5	29.7	31.0	Receive: 8842.6 Send: 791.8	Receive: 8635.6 Send: 729.1	965,455.7	0.0
Vulnerability	4.4	52.2	53.0	26.0	26.9	Receive: 16,215.5 Send: 1006.8	Receive: 15,785.7 Send: 997.2	1714,449.3	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (8 AV) + Deep CDR	15.4	72.4	75	34.3	35.4	Receive: 5436.6 Send: 611.3	Receive: 5589.3 Send: 607.2	495,697.4	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP	16.9	77.2	80.2	36.4	37.9	Receive: 5254 Send: 585.4	Receive: 5041.3 Send: 576.1	450,657.6	0.0
Metascan (8 AV) + Deep CDR + Proactive DLP + Vulnerability	18.0	72.1	80.4	38.6	38.0	Receive: 4504.3 Send: 574.1	Receive: 4649.5 Send: 571.1	423,010	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

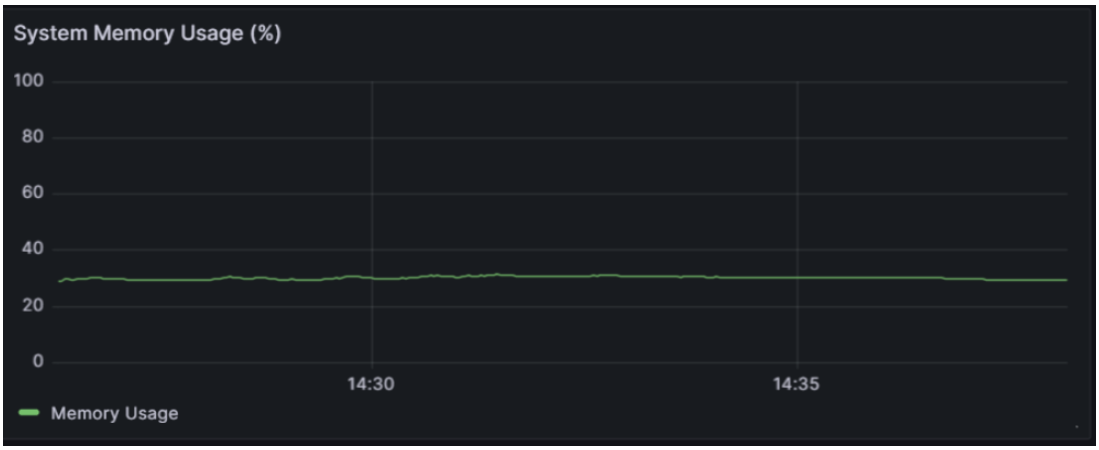
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2

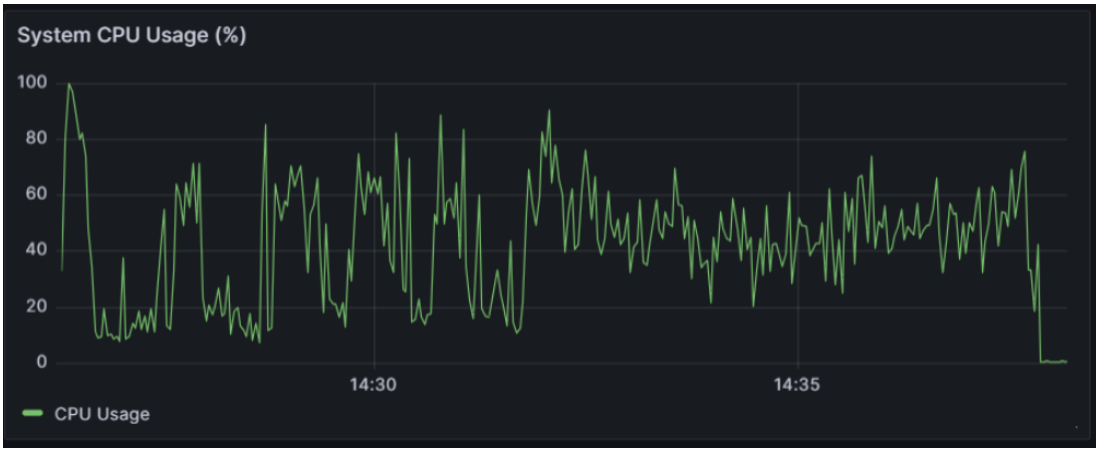


CPU usage

MD Core #1

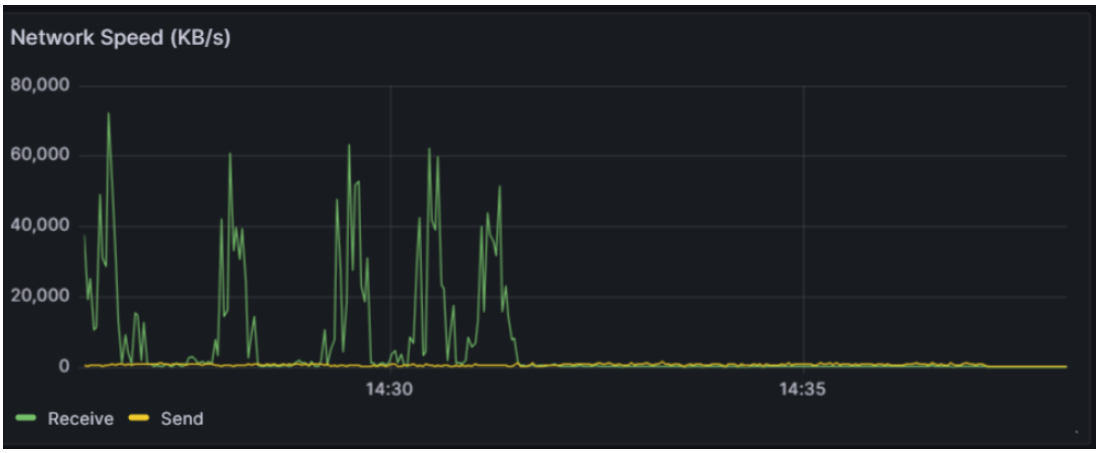


MD Core #2

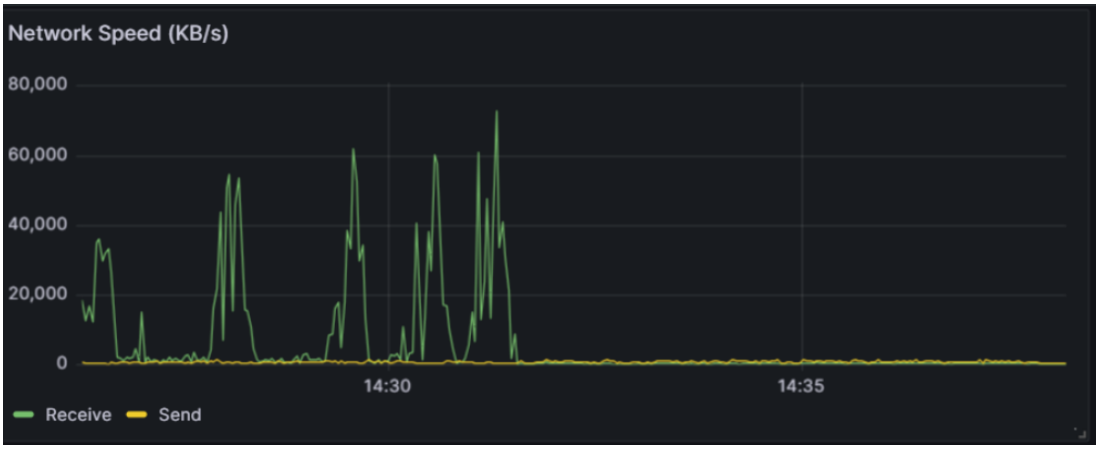


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2

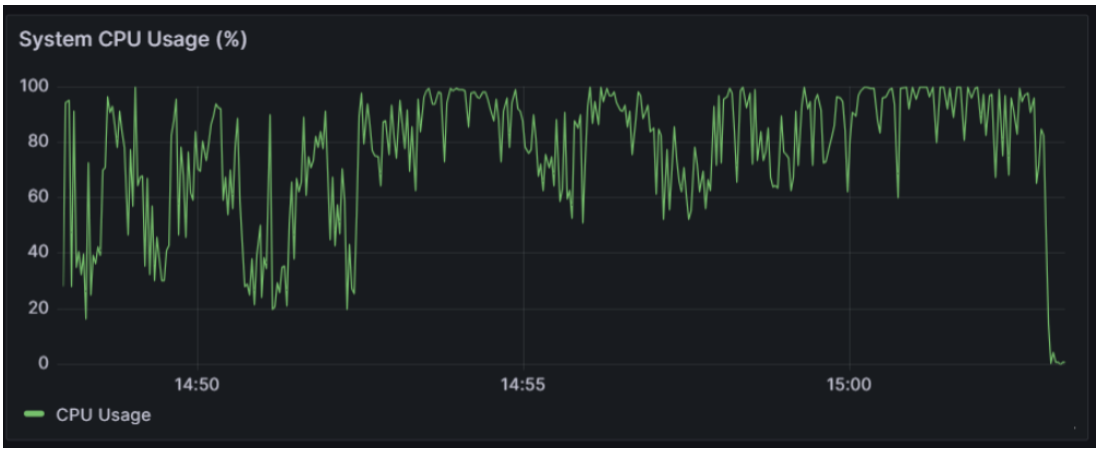


CPU usage

MD Core #1

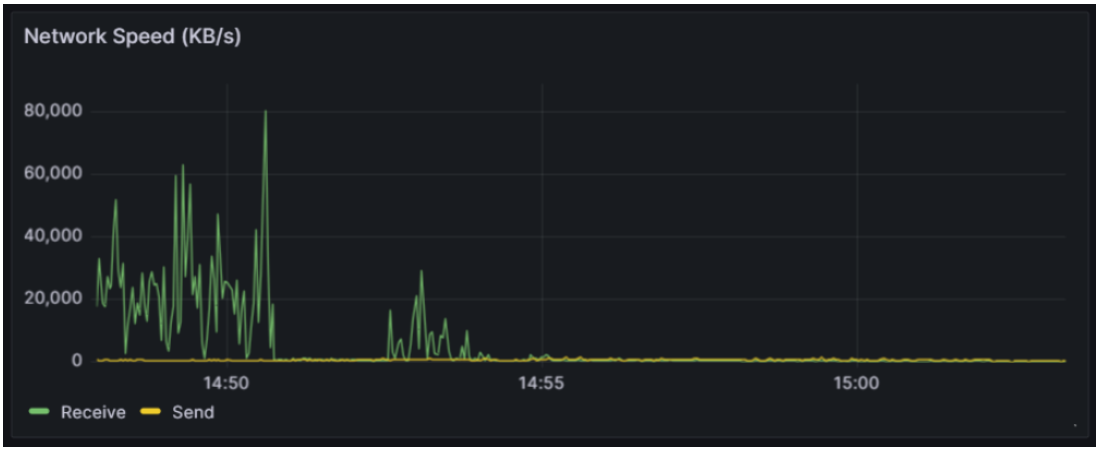


MD Core #2



Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

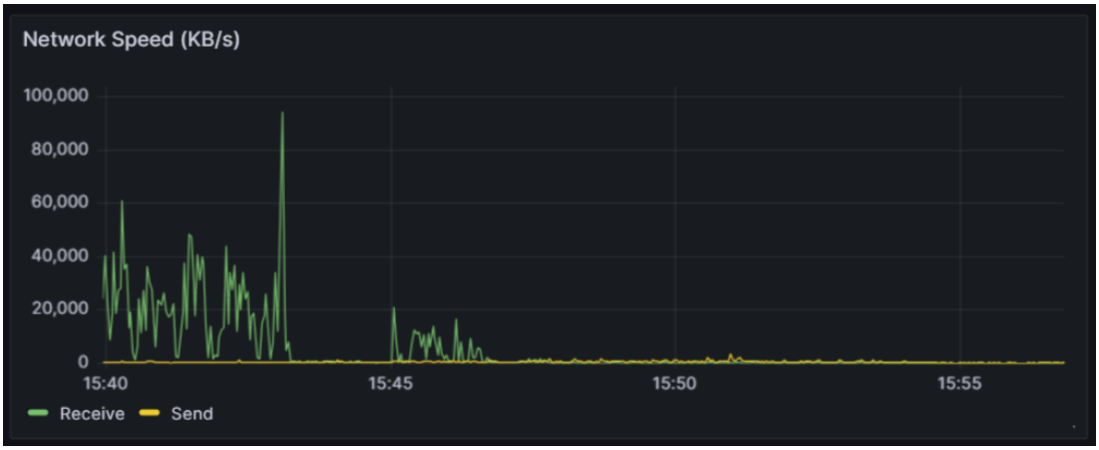


MD Core #2

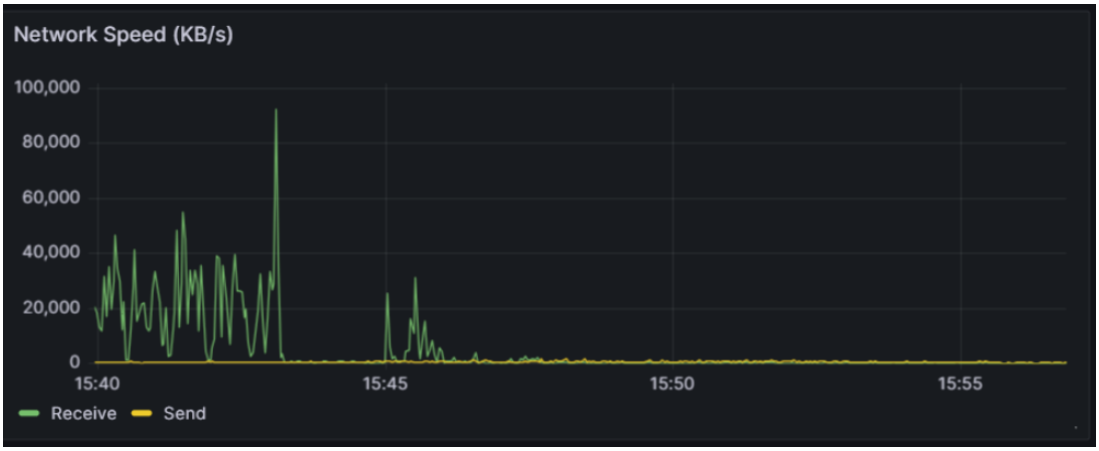


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1

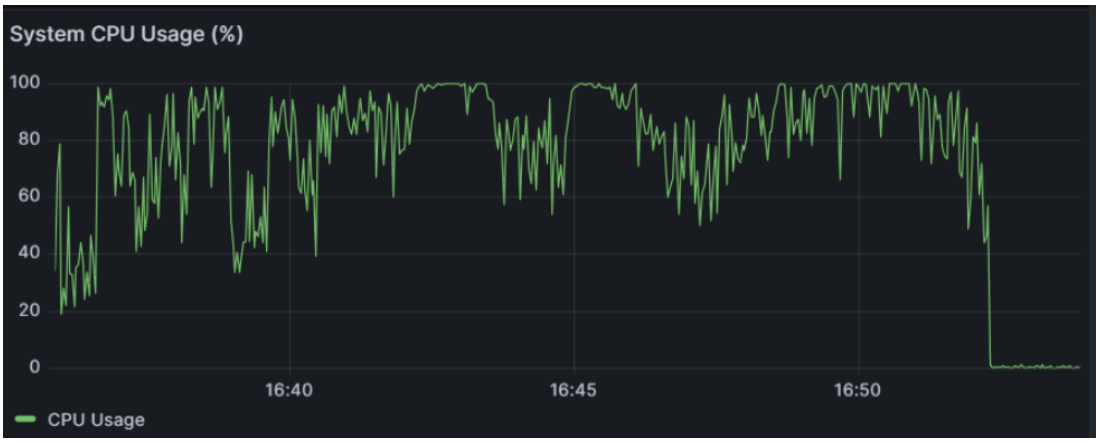


MD Core #2



CPU usage

MD Core #1

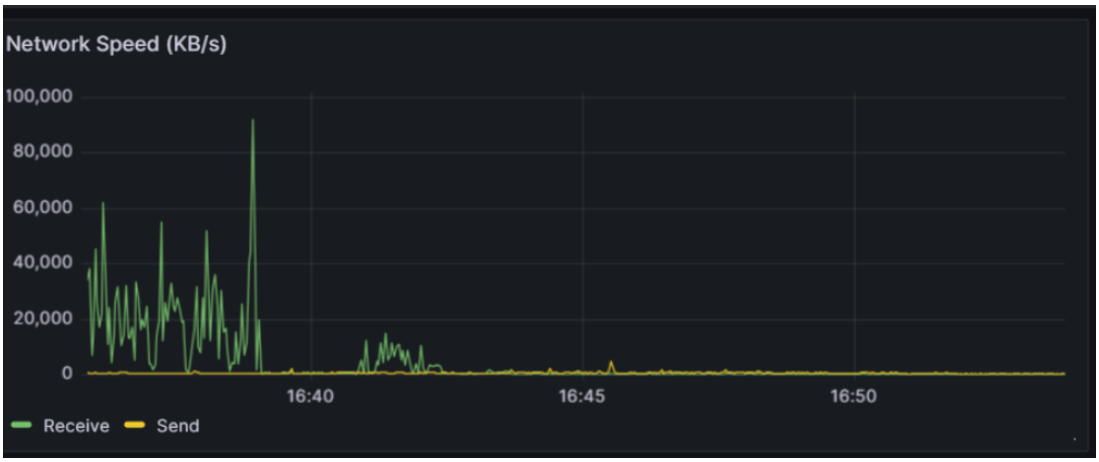


MD Core #2

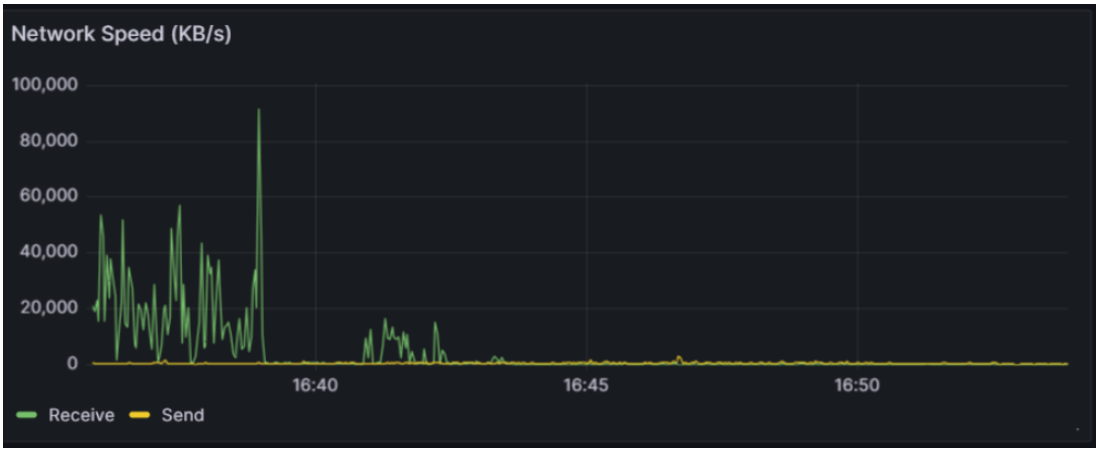


Network usage

MD Core #1



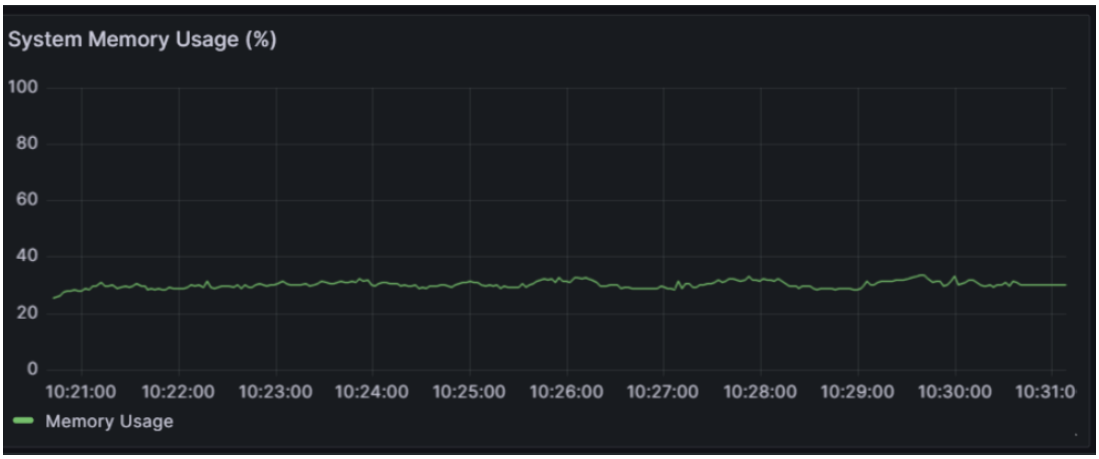
MD Core #2



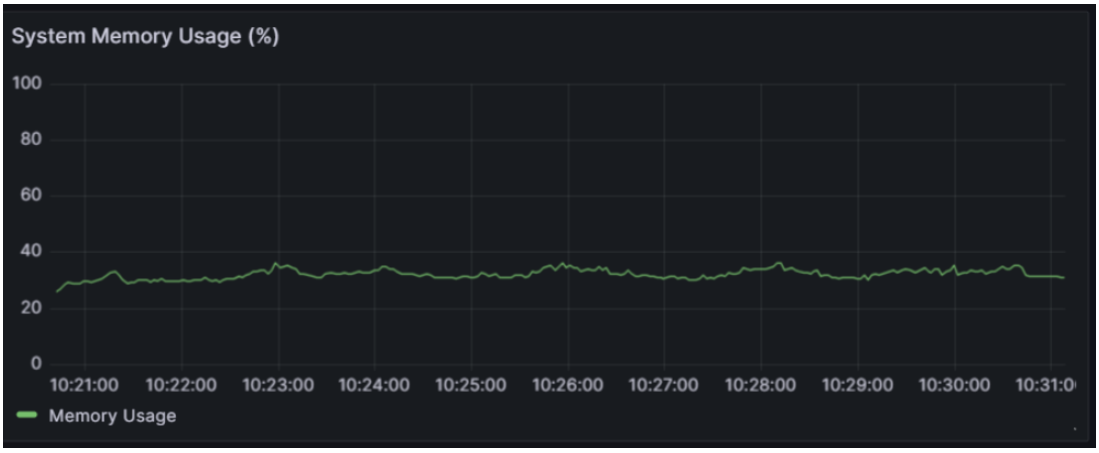
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

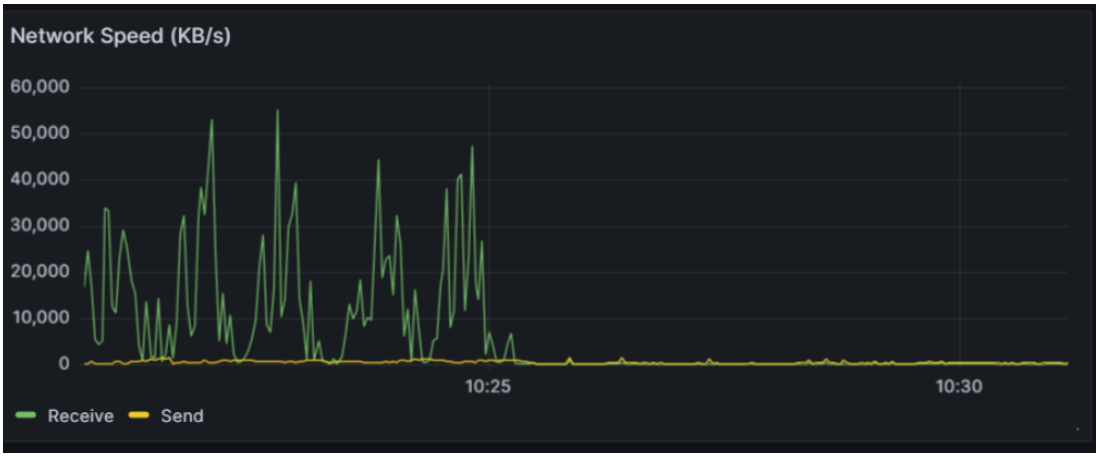


MD Core #2

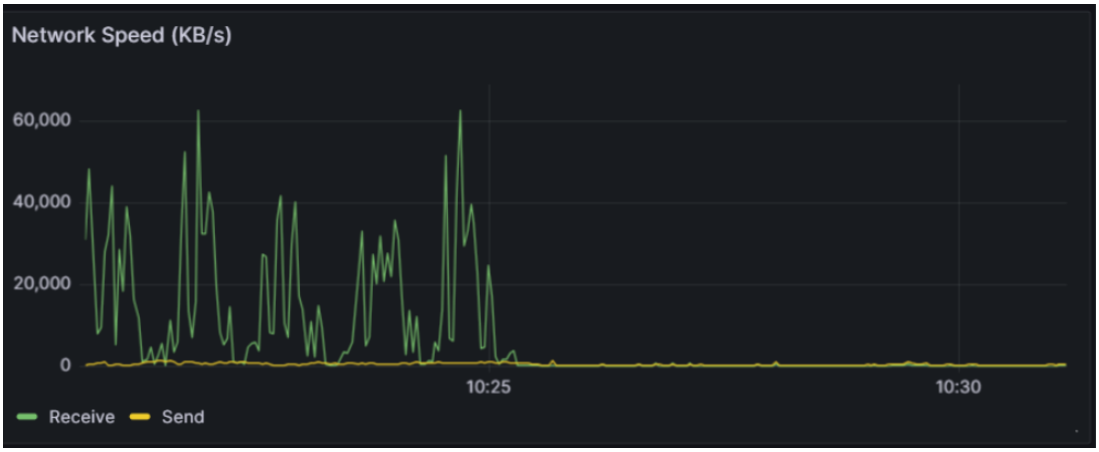


Network usage

MD Core #1



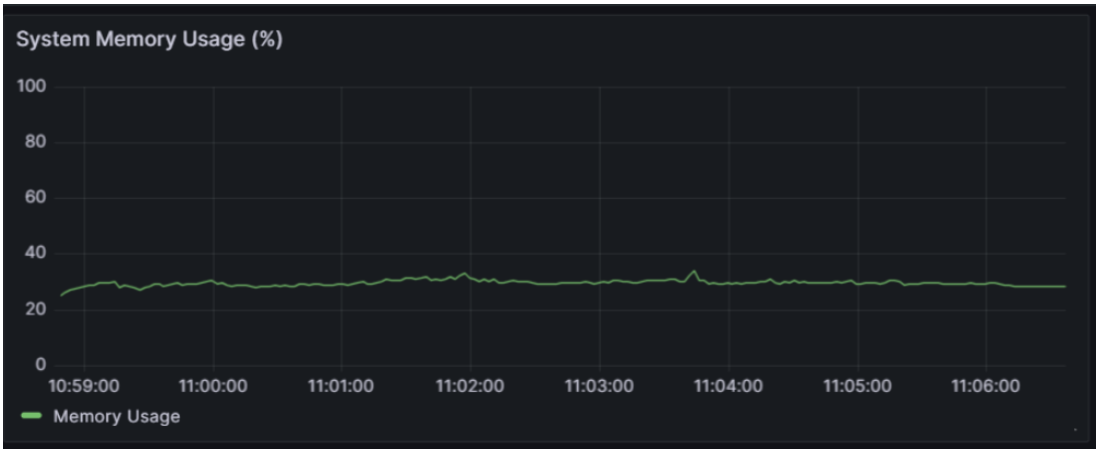
MD Core #2



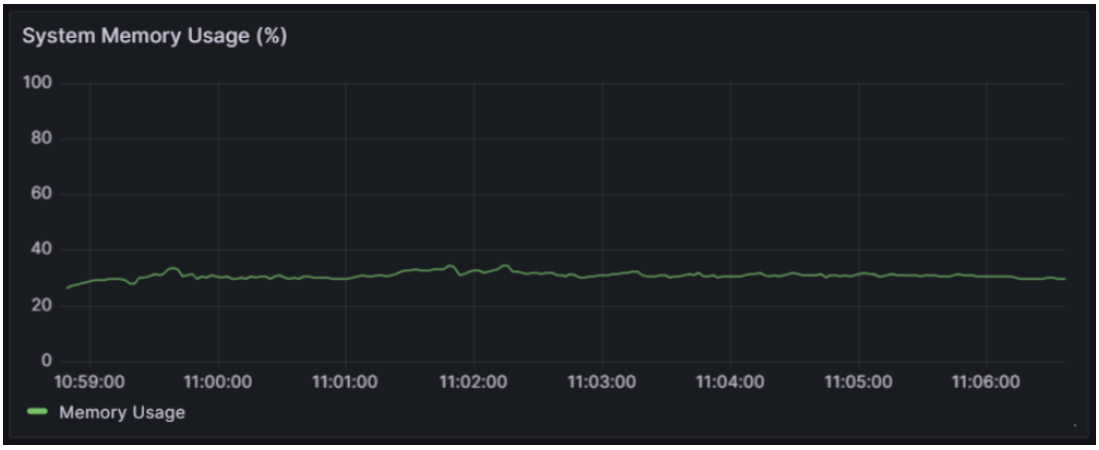
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

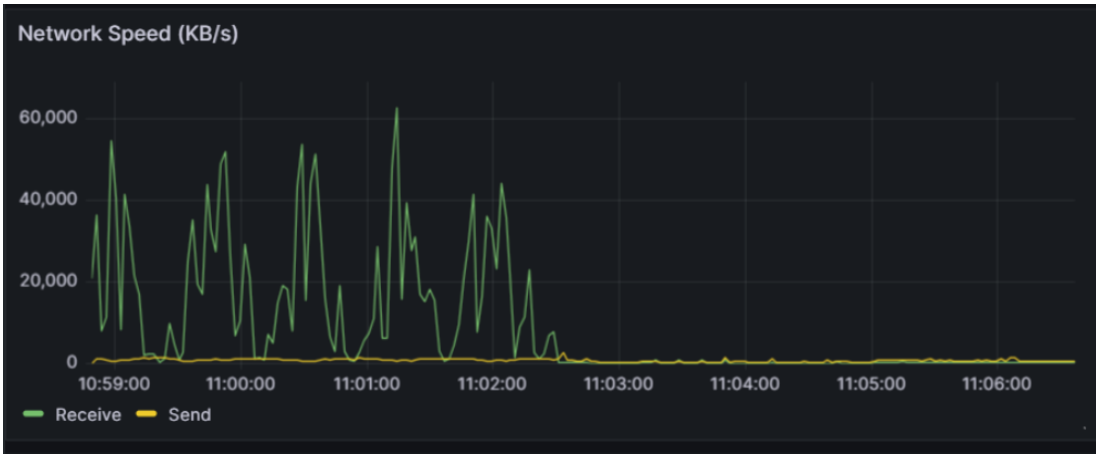


MD Core #2

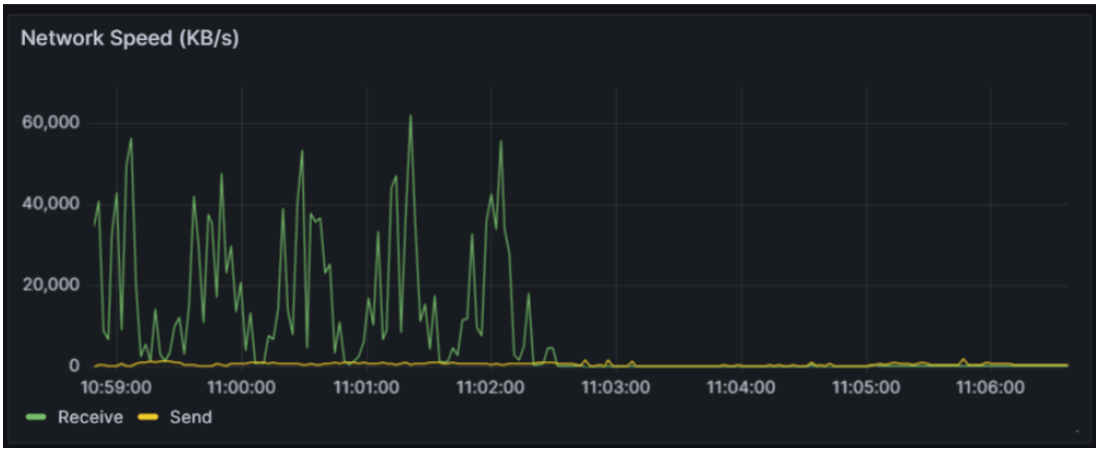


Network usage

MD Core #1



MD Core #2



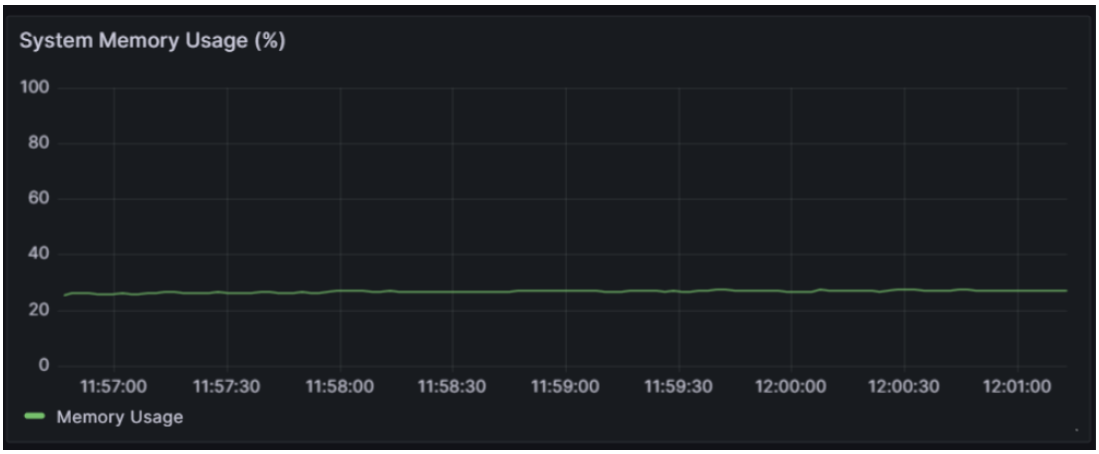
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

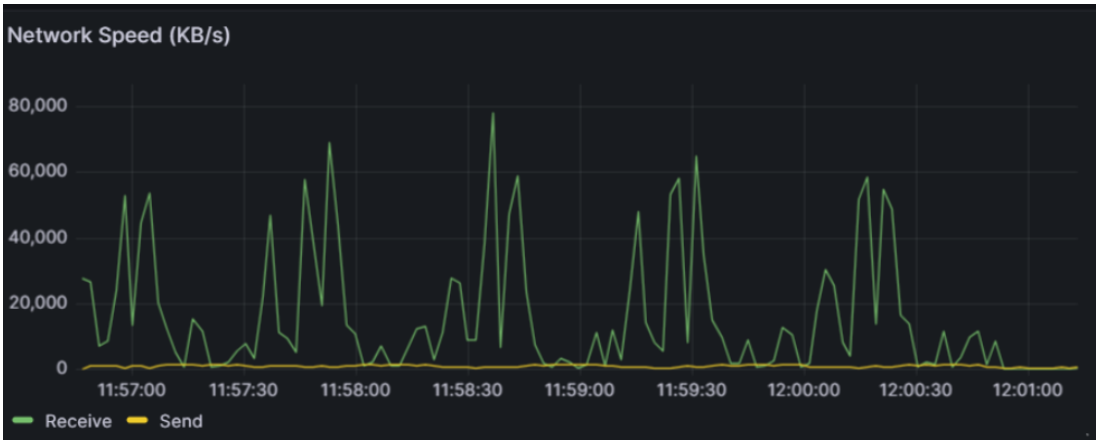


MD Core #2

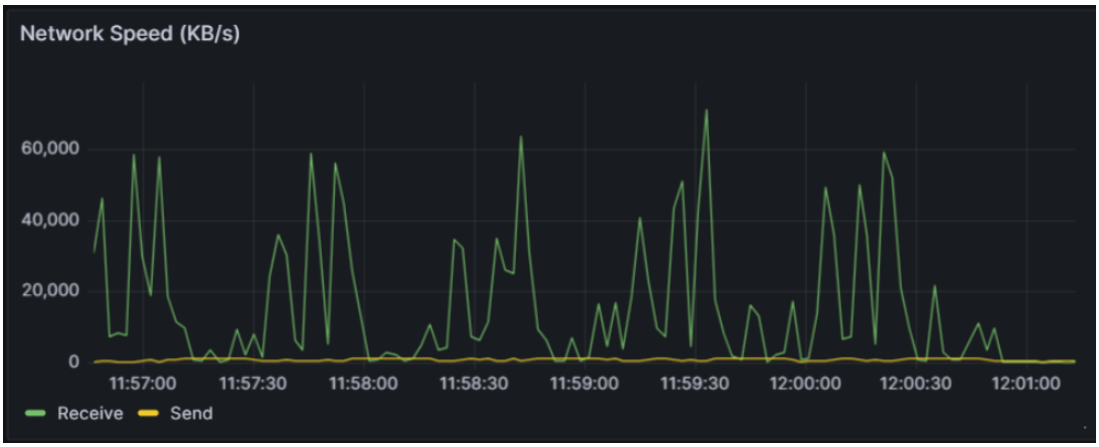


Network usage

MD Core #1



MD Core #2



Shared DB - Linux

Disclaimer

These results should be viewed as guidelines and not performance guarantees, since there are many variables that affect performance (file set, network configurations, hardware characteristics, etc.). If throughput is important to your implementation, OPSWAT recommends site-specific benchmarking before implementing a production solution.

Environment

Using AWS environment with the specification below:

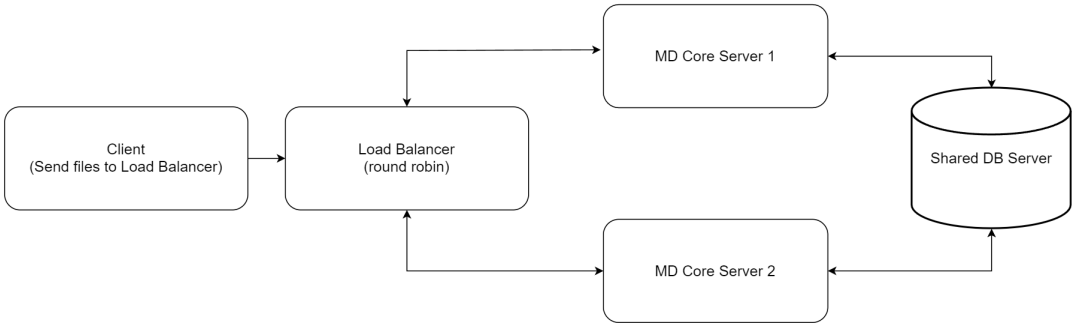
MetaDefender Core

	OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
MetaDefender Core #1	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD
MetaDefender Core #2	Linux CentOS 7	c5.4xlarge	16	32	Up to 10	SSD

Database Server

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
Windows Server 2022	c5.4xlarge	16	32	Up to 10	SSD

Deployment Model



Using a AWS Load Balancer to distribute files sent from the client tool to two (2) different MetaDefender Core servers applying Round Robin algorithm. With this algorithm, each MetaDefender Core server is supposed to receive same number of requests.

Client tool

A simple tool written in Python to collect files in a designated folder and submit requests to Load Balancer mentioned above.

python

```
files_to_scan = list of files to scan
scan_futures =[]
for file_path in files_to_scan:
    scan_futures.append(asyncio.create_task(self.scan_file(file_path,
self.load_balancer_url))

async def scan_file(self, file_path, core_url):
    api_url = f'{core_url}/file'

    with open(file_path, 'rb') as file:
        file_content = file.read()

    headers = {'Content-Type': 'application/octet-stream', 'filename': file_path}

    max_retries = 20  # Maximum number of retry attempts
    retry_delay = 120  # Delay in seconds before retrying

    for attempt in range(max_retries + 1):
        try:
            starttime = time.time()
            response = requests.post(api_url, data=file_content, headers=headers)
            endtime = time.time()
            status_code = response.status_code
            if status_code == 200:
                self.post_wait = self.post_wait + endtime - starttime
                self.num_post_req += 1
                response_json = response.json()
                data_id = response_json.get('data_id')
                return status_code, response_json, data_id

            if status_code == 503:
                print(f"Received status code {status_code}. Retrying in
{retry_delay} seconds...")
                await asyncio.sleep(retry_delay)
            else:
                return None, None, None

        except requests.RequestException as e:
            return None, None, None

    return None, None, None
```

OS	AWS instance type	vCPU	Memory (GB)	Network bandwidth (Gbps)	Disk type
CentOS 7	c5.2xlarge	8	16	Up to 10	SSD

Dataset

Detailed information of dataset below will be used for testing:

File category	File type	Number of files	Total size (MB)	Average file size (MB)
Adobe	PDF	370	385 MB	1.0 MB
Executable	EXE	45	309.5 MB	6.9 MB
	MSI	15	45.75 MB	3.1 MB
Image	BMP	80	515 MB	6.4 MB
	JPG	420	237.5 MB	0.6 MB
	PNG	345	169 MB	0.5 MB
Media	MP3	135	865 MB	6.4 MB
	MP4	50	500 MB	10.0 MB
Office	DOCX	235	190 MB	0.8 MB
	DOC	225	486 MB	2.2 MB
	PPTX	365	860 MB	2.4 MB
	PPT	355	1950 MB	5.5 MB
	XLSX	340	283.5 MB	0.8 MB
	XLS	335	284.5 MB	0.8 MB
Text	CSV	100	236 MB	2.4 MB
	HTML	1075	76 MB	0.1 MB
	TXT	500	210 MB	0.4 MB
Archive	ZIP	Compressed files: 10 Extracted files: 270	Compressed size: 125.5 MB Extracted size: 156.5 MB	Avg compressed size: 12.6 MB Avg extracted size: 0.6 MB
Summary (compressed)		5000	7728.5 MB	1.55 MB average file size
Summary (extracted)		5260	7759.5 MB	1.48 MB average file size

Product Information

Product versions:

- MetaDefender Core version 5.6.0
- Engines:
  - 10 AV engines: Ahnlab, Avira, ClamAV, ESET, Bitdefender, K7, Quick Heal, IKarus, Cyren, Tachyon (nProtect)
  - Deep CDR: 6.6.2
  - Proactive DLP: 2.16.0
  - Archive: 6.3.2
  - File type analysis: 6.3.2
  - File-based vulnerability assessment: 4.2

MetaDefender Core settings

General settings

- Turn off data retention
- Turn off engine update

Archive extraction settings

- Max recursion level: 99999999
- Max number of extracted files: 99999999
- Max total size of extracted files: 99999999
- Timeout: 10 minutes
- Handle archive extraction task as Failed: true
  - Extracted partially: true

Metascan AV settings

- Max file size: 99999999
- Scan timeout: 20 minutes
- Per engine scan timeout: 5 minutes

Performance test results

Summary

MetaDefender Core with single engine (technology):

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV)	15.5	63.1	73.3	25.2	25.8	Receive: 4555.9 Send: 694.4	Receive: 4516.2 Send: 692.4	492,620.5	0.0
Deep CDR	9.70	50.7	51.2	31.9	32.5	Receive: 7062.9 Send: 623.2	Receive: 6653 Send: 654.1	786,189.6	0.0
Proactive DLP	7.2	35.2	35.0	27.9	27.5	Receive: 9047.4 Send: 802.4	Receive: 9859.8 Send: 787.5	1,055,110.8	0.0
Vulnerability	5.0	45.7	49.3	24.4	24.5	Receive: 13,585.7 Send: 966.7	Receive: 13,505.8 Send: 987.4	1,519,562.1	0.0

MetaDefender Core with common engine packages:

Use case	Scan duration (minutes)	Avg. CPU usage - MD Core 1(%)	Avg. CPU usage - MD Core 2(%)	Avg. Memory usage - MD Core 1 (%)	Avg. Memory usage - MD Core 2 (%)	Avg. Network speed - MD Core 1 (KB/s)	Avg. Network speed - MD Core 2 (KB/s)	Throughput (processed objects/hour)	Avg tim (se
Metascan (10 AV) + Deep CDR	19.9	79.1	77.6	34.0	34.2	Receive: 3785.9 Send: 573.8	Receive: 3663.1 Send: 577.7	382,716	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP	21.6	80.3	81.5	37.7	44.4	Receive: 3305.6 Send: 543.3	Receive: 3414.3 Send: 556.2	353,694.9	0.0
Metascan (10 AV) + Deep CDR + Proactive DLP + Vulnerability	22.5	84.2	79.8	37.9	44.7	Receive: 3267.9 Send: 554.9	Receive: 3398.2 Send: 553.8	338,507.6	0.0

Recommendations

Controlling total processing time of each MD Core server:

In this deployment model, we should organize and send files in the way that it best utilizes the load of each MD Core server. It is not a good practice if one Core server is free while the other one is busy. By optimizing the distribution of files, we can ensure that each Core server is utilized efficiently, thereby improving overall system performance. Furthermore, this approach can help prevent bottlenecks and minimize the chances of system overload.

Adding proper number of MD Core servers to the cluster:

Adding more Core servers to this model will increase more load on the shared database. When adding a new MD Core server, users should monitor performance of database server such as memory/CPU consumption, disk usage, network bandwidth, request response time and so on… to see if it still can handle the load. This is important in order to maintain optimal performance and ensure that the database server can continue to efficiently serve the needs of the system.

Optimizing database server for better performance:

Continuing to add more Core servers to this model may result in increased strain on the shared database. As such, it is crucial to ensure that the database is optimized to handle the additional load effectively. Users can consider adjusting default database settings of PostgresSQL to optimize for more data load if needed. Here is where we can adjust PostgresSQL database settings: <PostgreSQL install location\version>\data\postgresql.conf.

Besides that, MD Core also supports a parameter (db_connection) for users to specify max connections that MD Core can handle, take a look at this guideline:

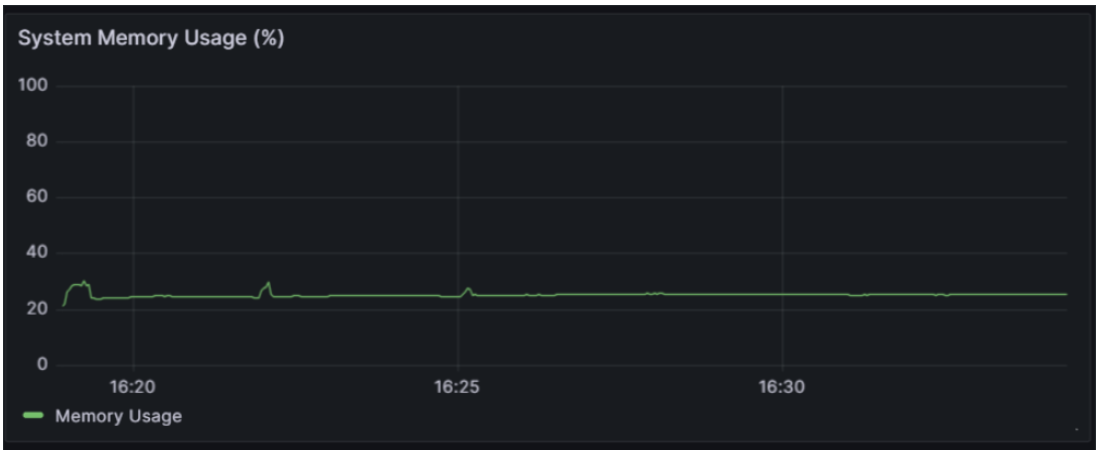
https://docs.opswat.com/mdcore/configuration/metadefender-configuration#internal

System resource utilization charts

MetaDefender Core with Metascan (8 AV engines):

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

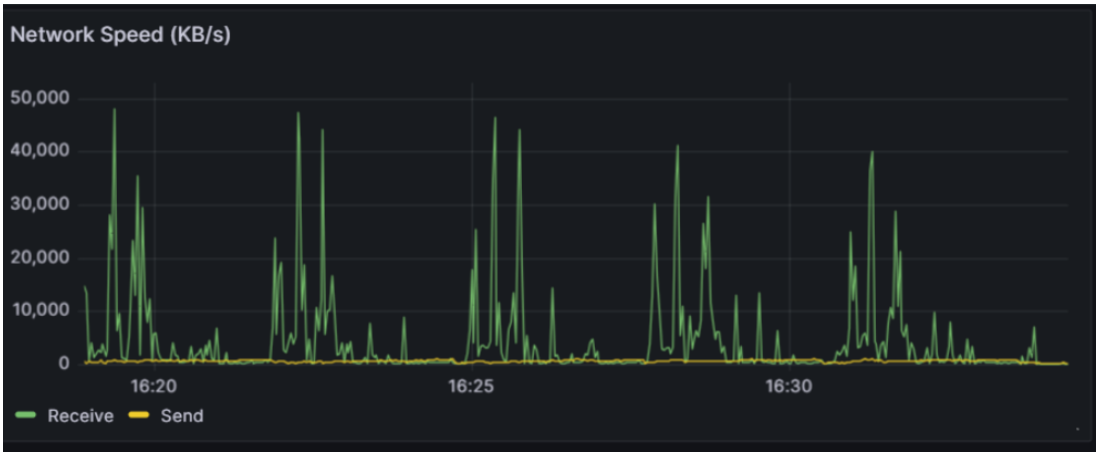


MD Core #2

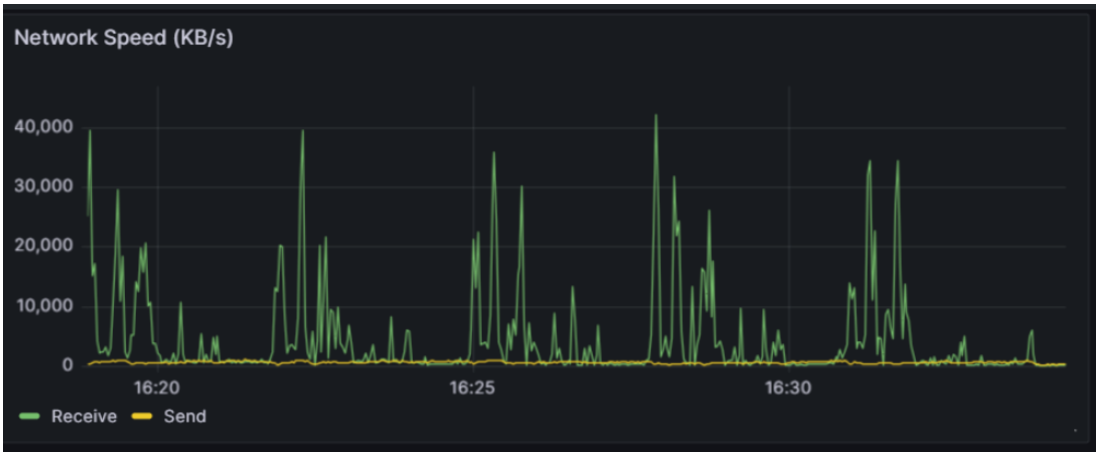


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

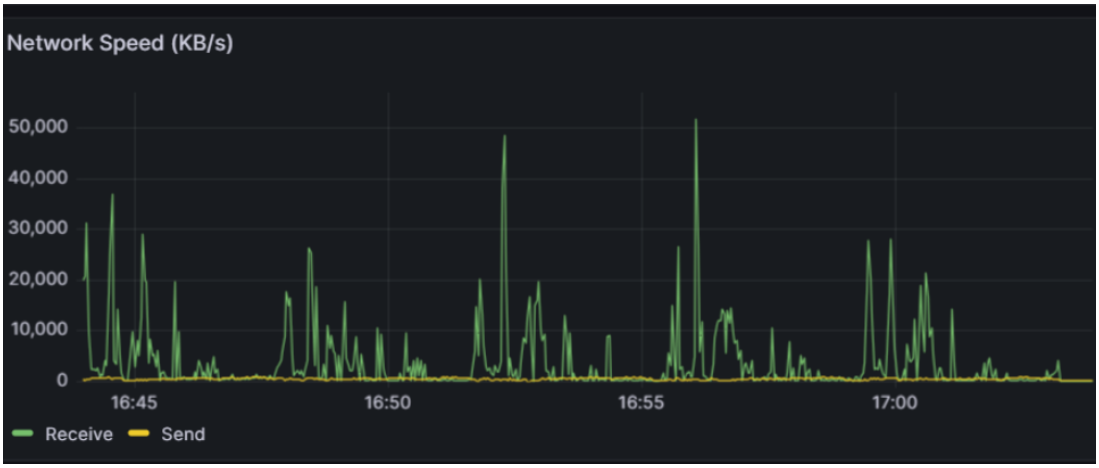


MD Core #2

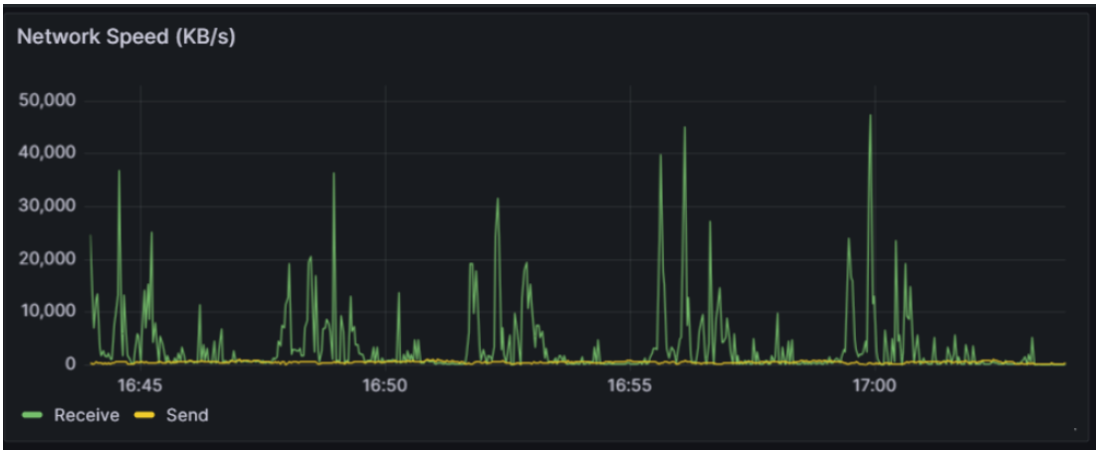


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP:

Memory usage

MD Core #1

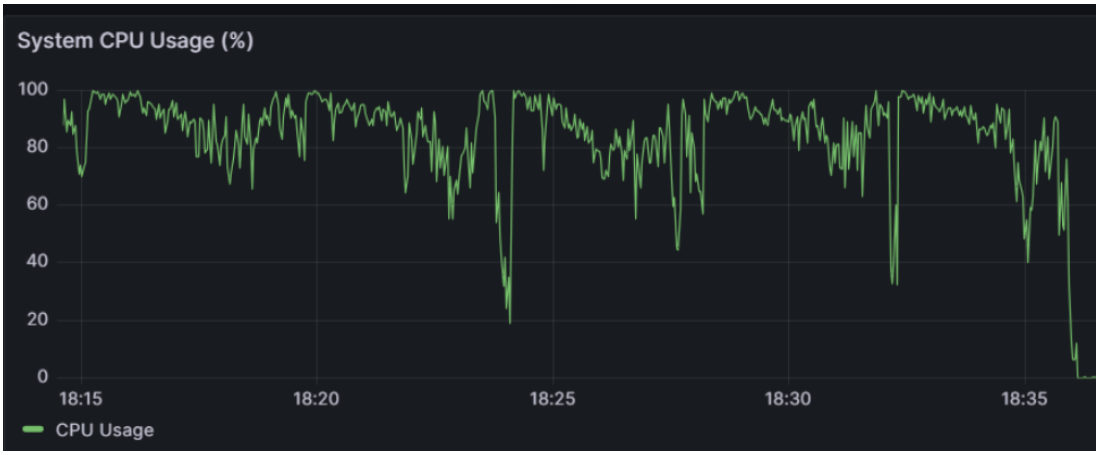


MD Core #2



CPU usage

MD Core #1

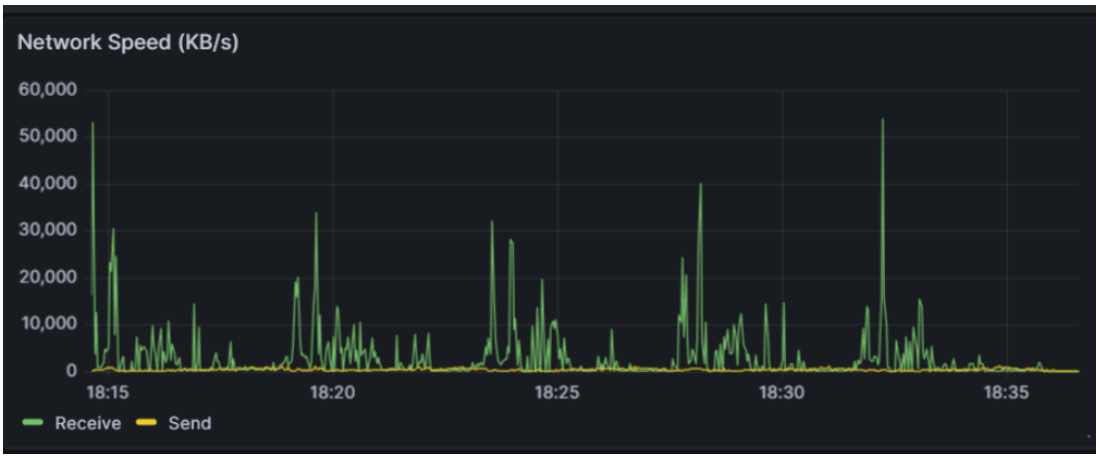


MD Core #2

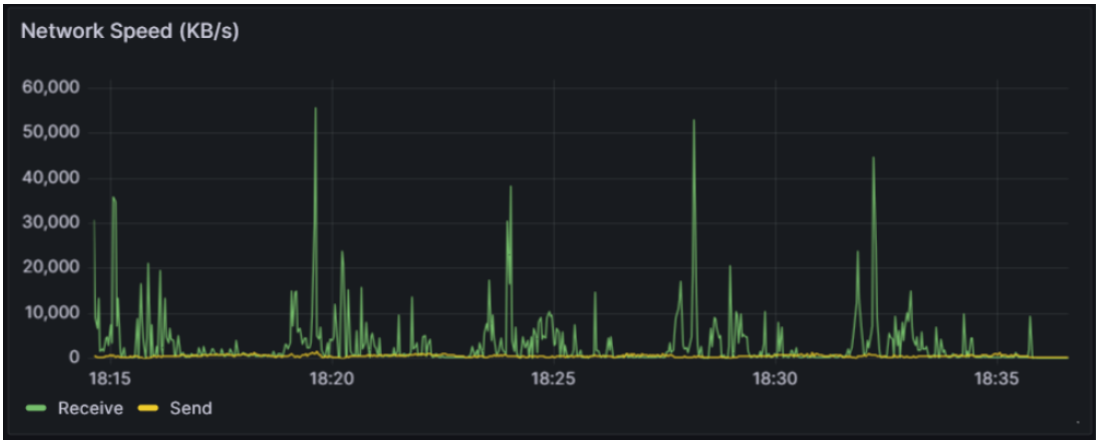


Network usage

MD Core #1



MD Core #2



MetaDefender Core with Metascan (8 AV engines) + Deep CDR + Proactive DLP + Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

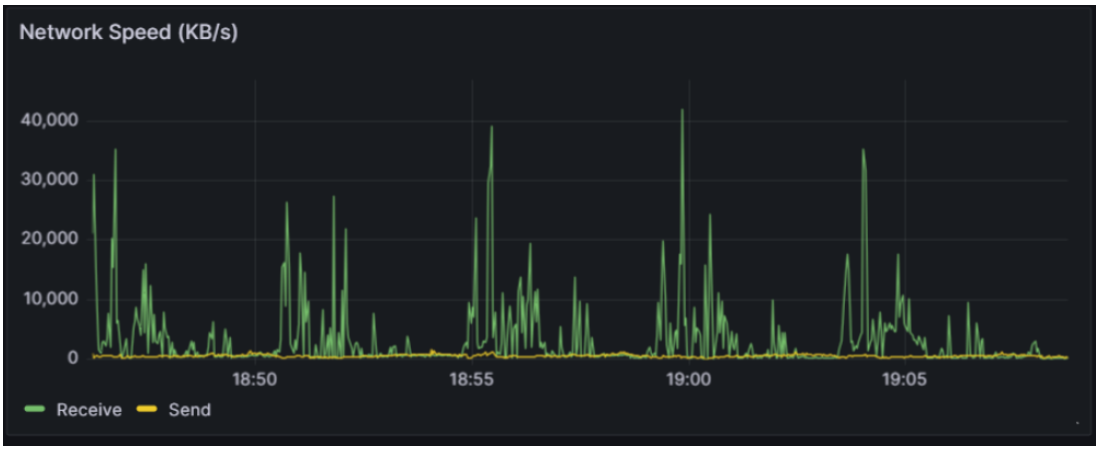


MD Core #2

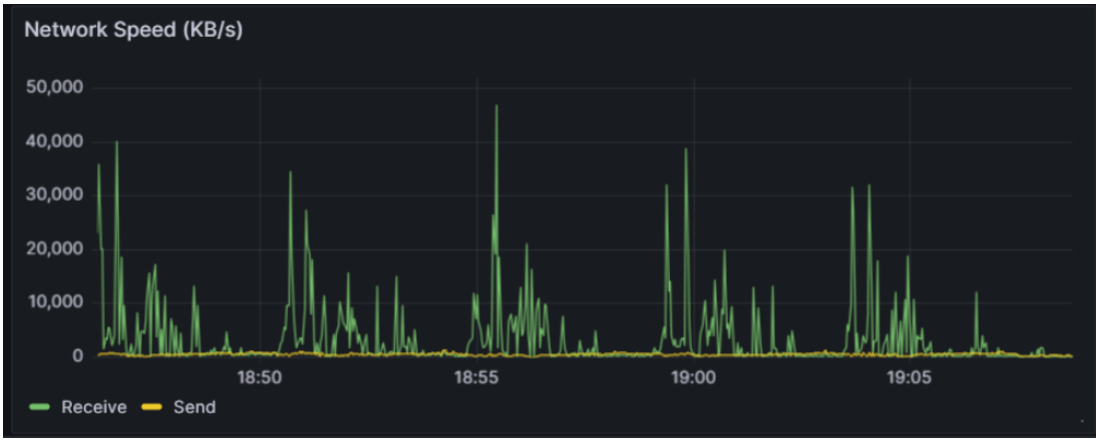


Network usage

MD Core #1



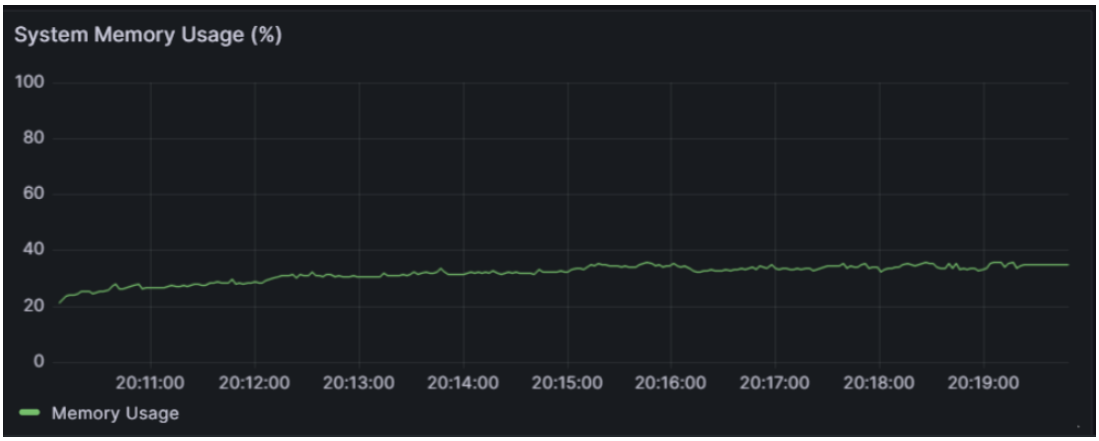
MD Core #2



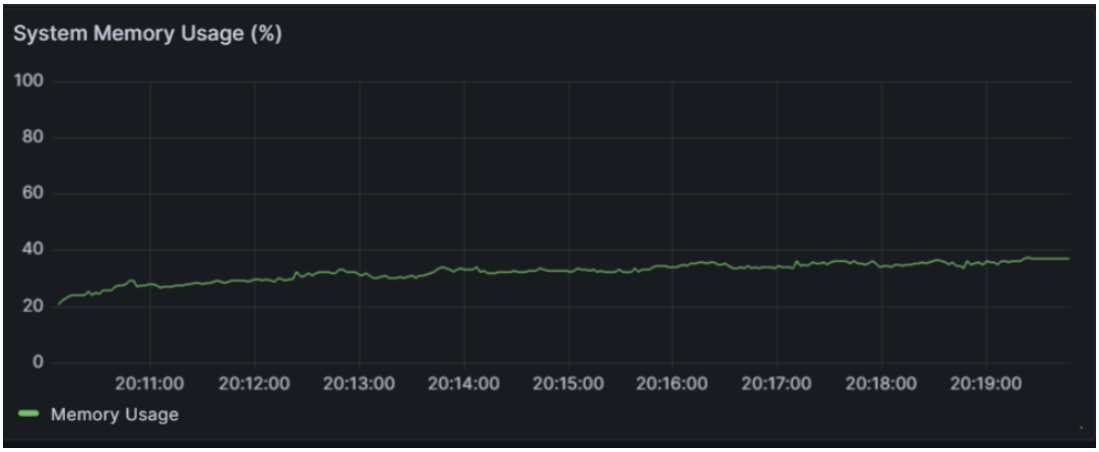
MetaDefender Core with Deep CDR:

Memory usage

MD Core #1

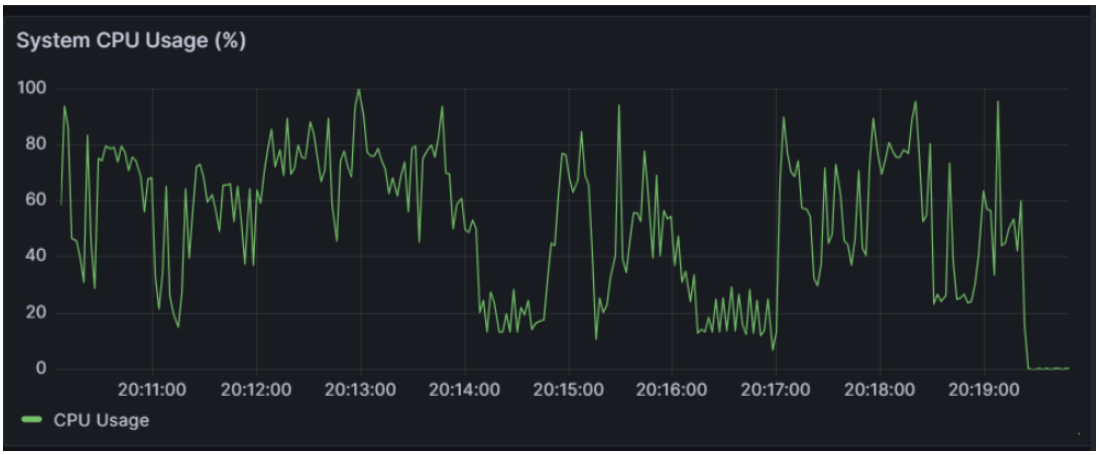


MD Core #2



CPU usage

MD Core #1

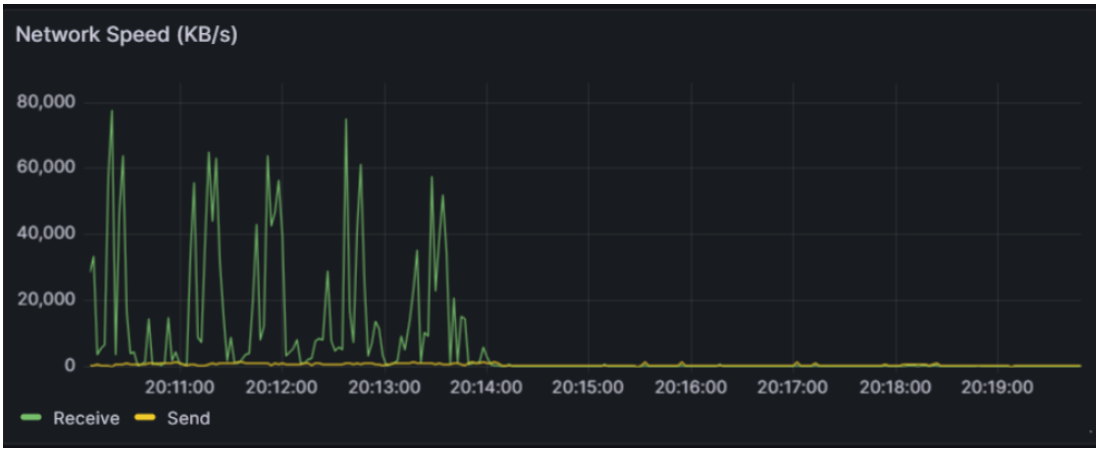


MD Core #2

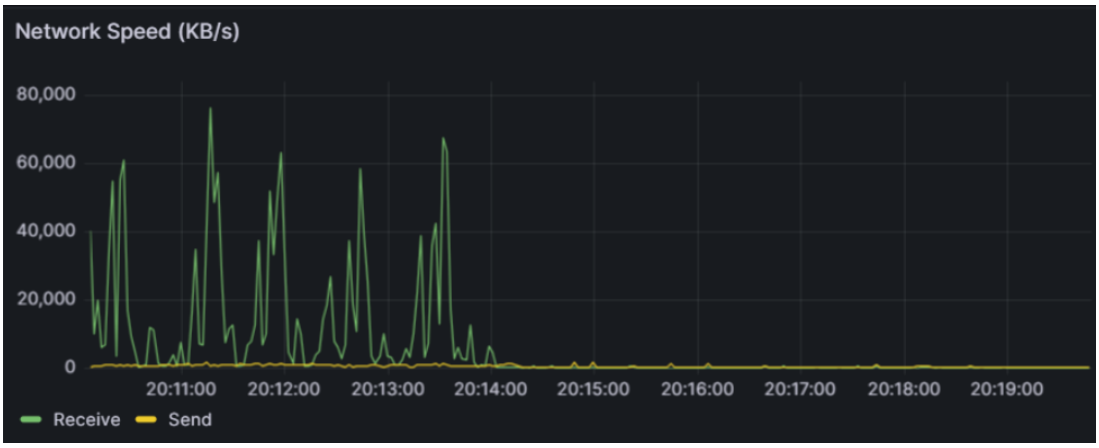


Network usage

MD Core #1



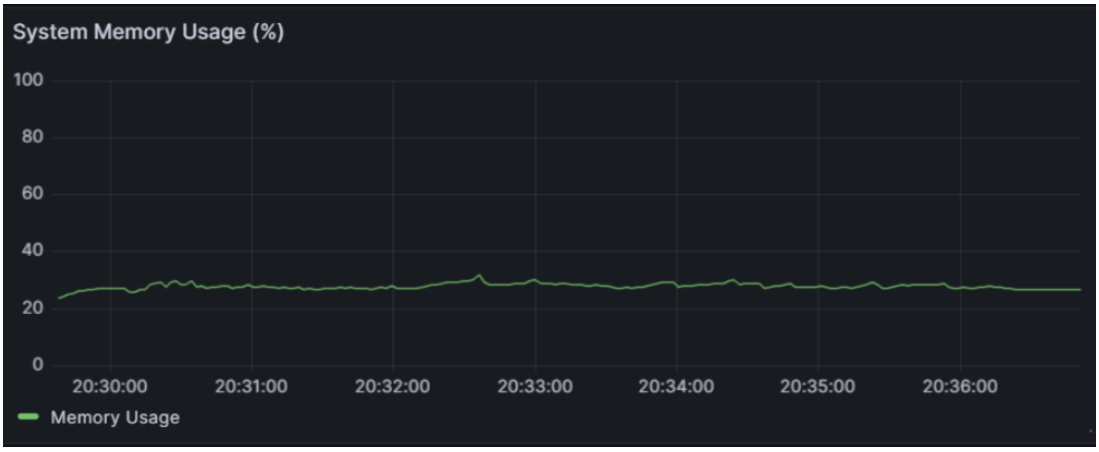
MD Core #2



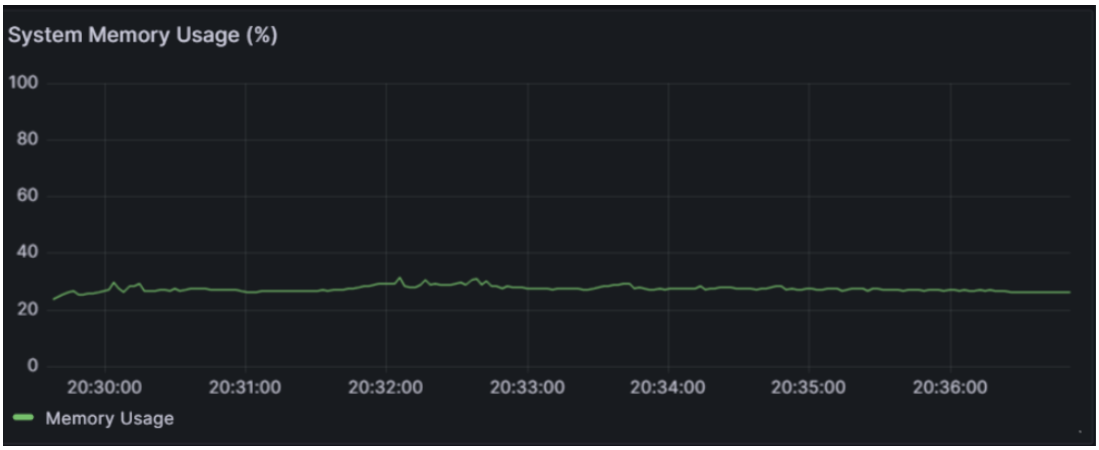
MetaDefender Core with Proactive DLP:

Memory usage

MD Core #1

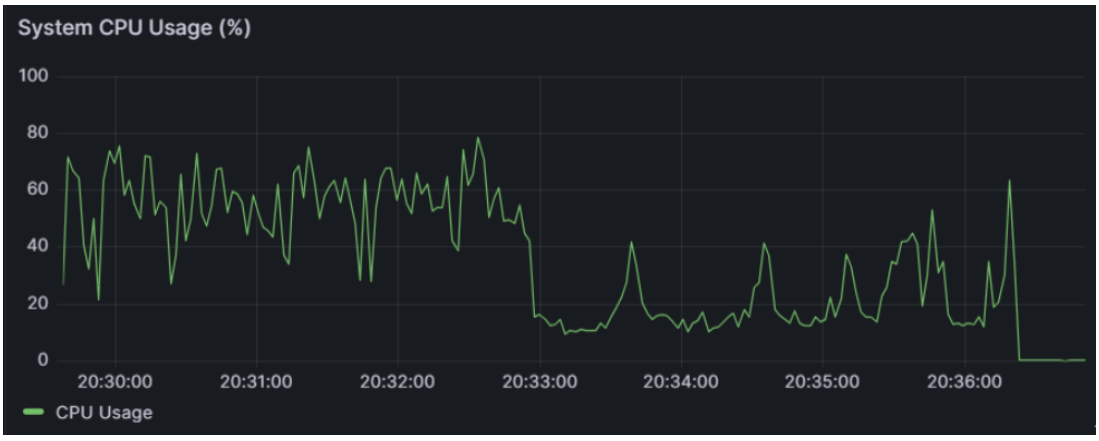


MD Core #2

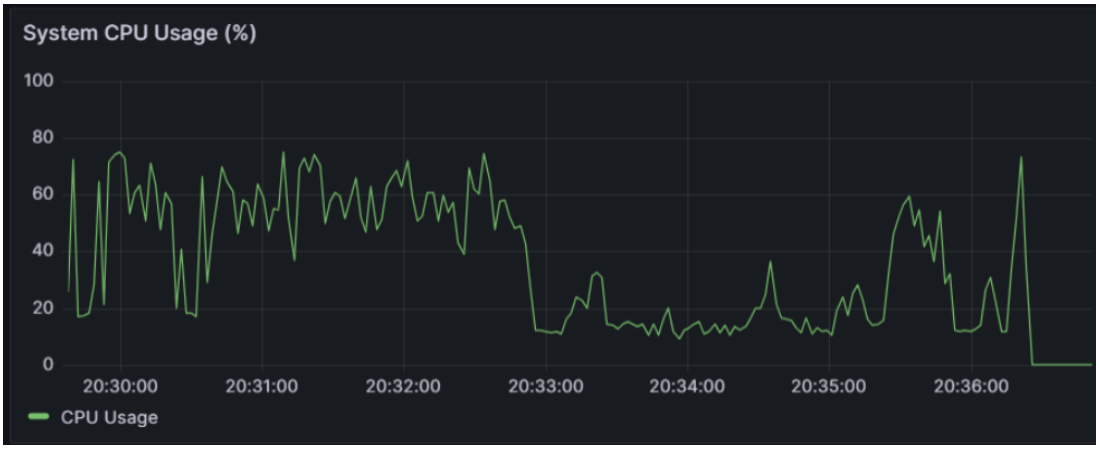


CPU usage

MD Core #1

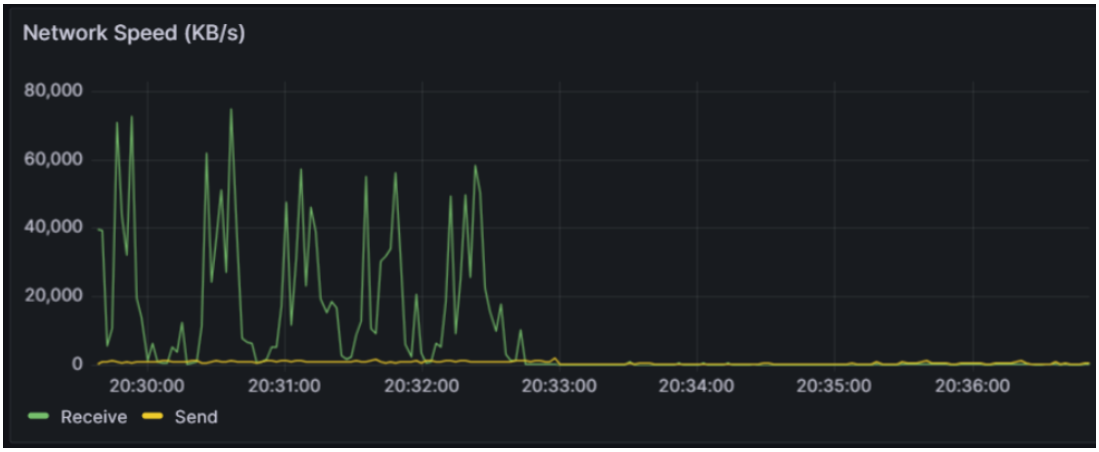


MD Core #2

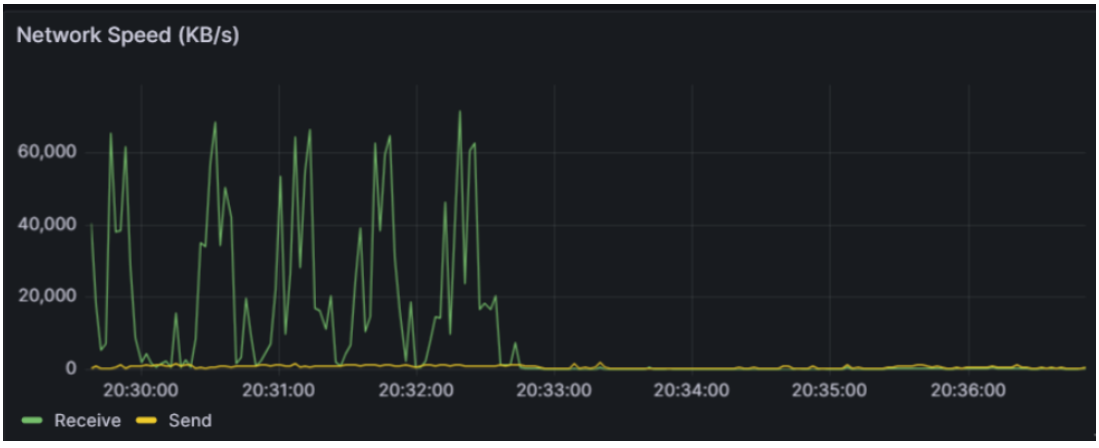


Network usage

MD Core #1



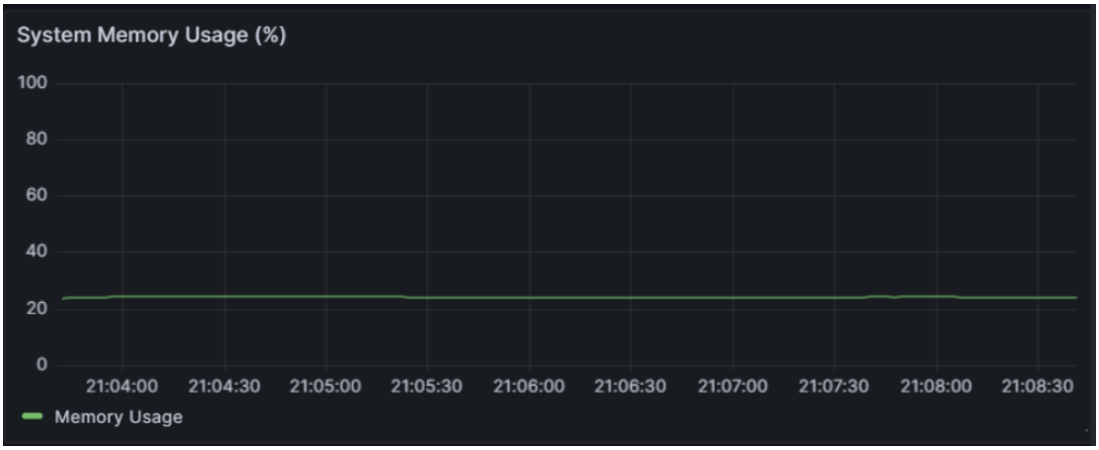
MD Core #2



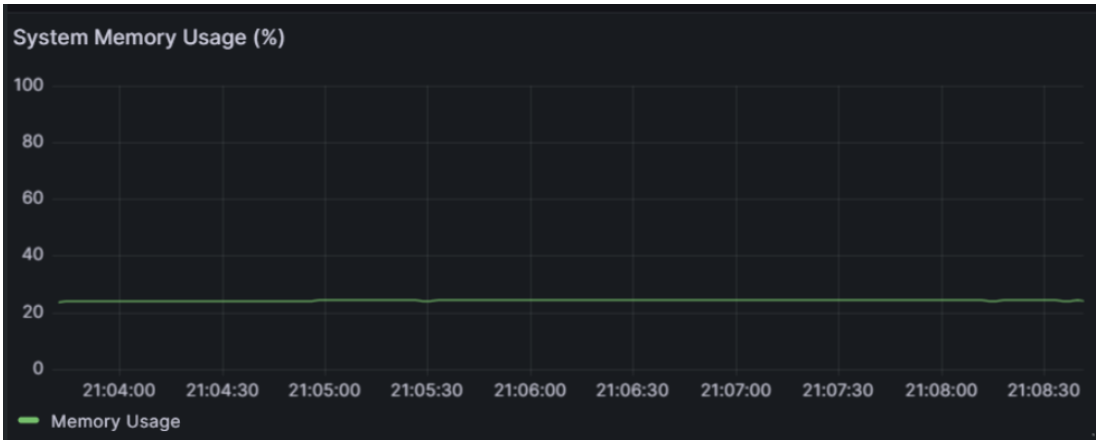
MetaDefender Core with Vulnerability:

Memory usage

MD Core #1



MD Core #2



CPU usage

MD Core #1

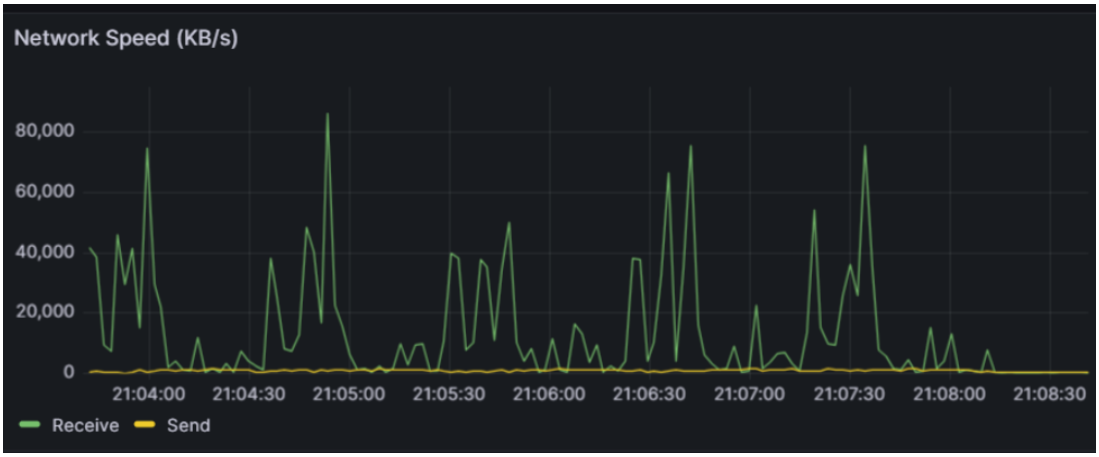


MD Core #2

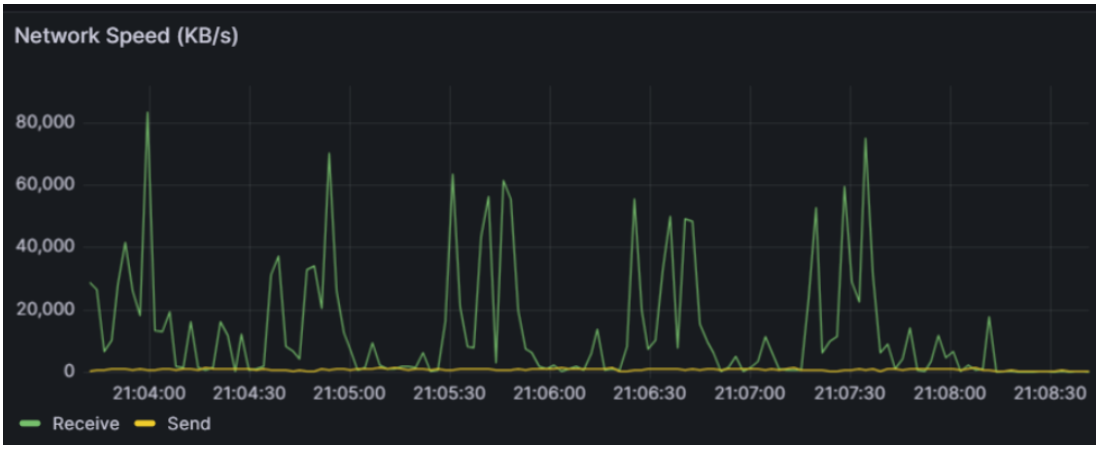


Network usage

MD Core #1



MD Core #2


Network Speed (KB/s)
80,000
60,000
40,000
20,000
0
21:04:00
21:04:30
21:05:00
21:05:30
21:06:00
21:06:30
21:07:00
21:07:30
21:08:00
21:08:30
Receive
Send

MetaDefender Core Documentation

The users can consult this web page or, alternatively, they can download the manual in pdf format from the link below:

MetaDefender Core manual (SHA256:).

API Reference

MetaDefender Core

API Version: v5.13.1

Developer Guide

This is the API documentation for *MetaDefender Core Public API*. If you would like to evaluate or have any questions about this documentation, please contact us via our Contact Us form.

How to Interact with MetaDefender Core using REST

Beginning with MetaDefender Core 4.x, OPSWAT recommends using the JSON-based REST API. The available methods are documented below.

Note: MetaDefender API doesn't support chunk upload, however is recommended to stream the files to MetaDefender Core as part of the upload process.

File Analysis Process

MetaDefender's main functionality is to analyze large volumes with a high throughput. Depending on the configuration and licensed technologies, the analysis times can vary. Below is a brief description of the API integration flow:

1. Upload a file for analysis (POST /file), which returns the data_id: File Analysis).

   Note: The performance depends on:

   - number of engines
   - type of file to be scanned
   - MetaDefender Core's hardware

2. You have 2 ways to retrieve the analysis report:

   - **Polling**: Fetch the result with previously received data_id (GET /file/{data_id} resource) until scan result belonging to data_id doesn't reach the 100 percent progress_percentage: (Fetch analysis result)

   Note: Too many data_id requests can reduce performance. It is enough to just check every few hundred milliseconds.

   - **Callbackurl**: Specify a callbackurl that will be called once the analysis is complete.

3. Retrieve the analysis results anytime after the analysis is completed with hash for files

(md5, sha1, sha256, sha512) by calling Fetch analysis result by hash.

- The hash can be found in the scan results

4. Retrieve processed file (sanitized, redacted, watermarked, etc.) after the analysis is complete.

Note: Based on the configured retention policy, the files might be available for retrieval at a later time.

OPSWAT provides some sample codes on GitHub to make it easier to understand how the MetaDefender REST API works.

CONTACT

NAME: API Support
EMAIL: feedback@opswat.com
URL: https://github.com/OPSWAT/metadefender-core-openapi3
Terms of service: https://onlinehelp.opswat.com/policies/

Security and Authentication

SECURITY SCHEMES

KEY	TYPE	DESCRIPTION
apikey	apiKey	Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

API

1. YARA

YARA engine configuration and source management APIs.

1.1 GET /admin/config/yara/sources

Get YARA sources

Retrieve a list of all sources for the YARA Engine.

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: A list of YARA Engine sources.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
local_sources	array	
source	string	The location (file path) of the files.
state	enum	ALLOWED: disabled, enabled Defining if this source is being used or not
http_sources	array	
source	string	The location (URL) of the package.
state	enum	ALLOWED: disabled, enabled Defining if this source is being used or not

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

1.2 PUT /admin/config/yara/sources

Modify YARA sources

Modify one (or more) of the sources for the YARA Engine.

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
local_sources	array	
source	string	The location (file path) of the files.
state	enum	ALLOWED: disabled, enabled Defining if this source is being used or not
http_sources	array	
	string	

NAME	TYPE	DESCRIPTION
source		The location (URL) of the package.
state	enum	ALLOWED: disabled, enabled Defining if this source is being used or not

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: A list of YARA Engine sources.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
local_sources	array	
source	string	The location (file path) of the files.
state	enum	ALLOWED: disabled, enabled Defining if this source is being used or not
http_sources	array	
source	string	The location (URL) of the package.
state	enum	ALLOWED: disabled, enabled Defining if this source is being used or not

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

1.3 GET /yara/package

YARA Generation Status

Get status of YARA package generation.

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
status	enum	ALLOWED: idle, error, inprogress Current status of generating the package.
start_time	string	Used only when status is inprogress, otherwise its empty
issues	object	
general	array	
severity	string	
message	string	

NAME	TYPE	DESCRIPTION
source	array	
severity	string	
message	string	

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

1.4 POST /yara/generate

Generate YARA package

Generate a new package based on the new added sources.

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
status	enum	ALLOWED: `idle, error, inprogress` Current status of generating the package.
start_time	string	Used only when status is inprogress, otherwise its empty
issues	object	
general	array	
severity	string	
message	string	
source	array	
severity	string	
message	string	

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

2. ADMIN

Admin specific API requests.

2.1 GET /admin/export

Export configuration

Export configuration from file.

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Failed to export configuration

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	

EXAMPLE:

```
{
 "err": "Failed to export configuration"
}
```

2.2 GET /admin/export/v2

Export configuration (new version)

Export configuration from file (new version)

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.
*password	string		Password to encrypt all the settings after exporting from MetaDefender Core. Only applicable when importing ZIP file.
setting	string	{"yara": {"value":false}}	To export advanced settings of YARA engine, set the key "value" in the JSON to true. {"yara":{"value":false}}

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/octet-stream

STATUS CODE - 400: Bad request

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	

EXAMPLE:

```
{
 "err": "Unable to export new configuration"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

2.3 POST /admin/import

Import configuration

Import configuration from file.

REQUEST

REQUEST BODY - application/json

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
result	string	

STATUS CODE - 304: The configuration has not changed

RESPONSE MODEL - application/json

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
```

```
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

2.4 POST /admin/import/v2

Import configuration (new version)

Import configuration from file (new version)

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.
password	string		Password to decrypt all the settings before importing. Only applicable when importing ZIP file.
import_force_update_yara	boolean		To force updating and overriding YARA settings.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		

NAME	TYPE	DESCRIPTION
result	string	

STATUS CODE - 304: The configuration has not changed

RESPONSE MODEL - application/json

STATUS CODE - 400: Bad request

RESPONSE MODEL - application/json

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

2.5 POST /admin/user

Create user

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
api_key	string	Associated apikey with this user
directory_id	integer	To which User Directories belongs to (LOCAL/System/etc.)
display_name	string	Which is the name showed in the Management Console
email	string	User's email address
name	string	User's full name
description	string	User's description, 256 characters maximum
roles	array	
ui_settings	object	
password	string	The user's password

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
api_key	string	Associated apikey with this user
directory_id	integer	To which User Directories belongs to (LOCAL/System/etc.)
display_name	string	Which is the name showed in the Management Console
email	string	User's email address
name	string	User's full name
description	string	User's description, 256 characters maximum

NAME	TYPE	DESCRIPTION
roles	array	
ui_settings	object	

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		

NAME	TYPE	DESCRIPTION
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

2.6 POST /admin/role

Create new role

Add a new user role to the system.

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
name	string	The name identifier of the role
display_name	string	The extended name showed in the Management Console.
rights	object	
agents	array	
cert	array	
configlog	array	
engines	array	
external	array	
license	array	
quarantine	array	
retention	array	
rule	array	
scan	array	
scanlog	array	
skip	array	
update	array	
updatelog	array	
users	array	

NAME	TYPE	DESCRIPTION
workflow	array	
zone	array	

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
name	string	The name identifier of the role
display_name	string	The extended name showed in the Management Console.
rights	object	
agents	array	
cert	array	
configlog	array	
engines	array	
external	array	
license	array	
quarantine	array	
retention	array	
rule	array	
scan	array	
scanlog	array	
skip	array	
update	array	
updatelog	array	
users	array	
workflow	array	
zone	array	

NAME	TYPE	DESCRIPTION
editable*	boolean	If the role can be altered or not.
id*	integer	Internal used identifier

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

2.7 GET /admin/license/backup

Retrieve hot-backup instance

This API to set a MetaDefender Core as a backup instance.

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
is_backup	boolean	The instance is a backup or not.

EXAMPLE:

```
{
 "is_backup": true
}
```

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

2.8 POST /admin/license/backup

Setup hot-backup instance

This API to set a MetaDefender Core as a backup instance.

- Online environment: you need to input activation-key and active-deployment-id headers

- Offline environment
  :
  - Obtain license.yml file that supposed to use to active your MetaDefender Core active / production instance before.
  - Use that yaml file content in body content of the REST API.

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.
active-deployment-id	string		Deployment ID of the main MetaDefender Core which is currently working, and you want to backup it.
activation-key	string		Activation key

RESPONSE

STATUS CODE - 200: Request successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
success	string	Successful message

EXAMPLE:

```
{
 "success": "The backup MetaDefender Core instance is now registered as a backup one."
}
```

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

2.9 DELETE /admin/license/backup

Remove a backup instance

This API to remove a backup instance.

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
success	string	Successful message

EXAMPLE:

```
{
 "success": "This MetaDefender Core instance is no longer registered as a backup one."
}
```

STATUS CODE - 400: Bad Request

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Failed to remove backup license"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

2.10 POST /admin/license/backup/nomination

Nominate a backup instance to a production instance

This API to nominate a backup instance to a production instance.

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
success	string	Successful message

EXAMPLE:

```
{
 "success": "This MetaDefender Core instance is now fully activated and allowed to process
files."
}
```

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		

NAME	TYPE	DESCRIPTION
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

3. ANALYSIS

File analysis APIs

Submit each file to MetaDefender Core individually or group them in batches. Each file submission will return a data_id which will be the unique identifier used to retrieve the analysis results. **Important**: Even though one file is being submitted, if MetaDefender Core is configured to extract the files, all compound file types (archives, Office documents, etc.) will be extracted and each file within will be analyzed as a separate entry.

- This means that if you submit an archive with 100 files in it, MetaDefender Core will process 101 files: original file as it is and each of the 100 child files
  - Note that by opening the files, detection ratio can increase even by 30%.

Note*:* MetaDefender API doesn't support chunk upload. You shouldn't load the file in memory, is recommended to stream the files to MetaDefender Core as part of the upload process.

`3.1 POST /file`

Analyze File (Asynchronous mode)

Scanning a file using a specified workflow. Scan is done asynchronously and each scan request is tracked by data id of which result can be retrieved by API Fetch Scan Result.

Note: Chunked transfer encoding (applying header Transfer-Encoding: Chunked) is **not supported** on /file API.

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.
filename	string		The name of the submitted file
filepath	string		if local file scan is enabled the path to the file (see Security rule configuration)
user_agent	string		user_agent header used to identify (and limit) access to a particular rule. For rule selection, `rule` header should be used. Since MetaDefender Core 5.6.0, when user agent is matched the predefined setting in workflow rule, the product will return X-Core-Id containing identity of MetaDefender Core server as a header of response.
rule	string		Select rule for the analysis, if no header given the default rule will be selected (URL encoded UTF-8 string of rule name)
workflow	string		name of the selected workflow (deprecated, use "rule" header parameter instead)
batch	string		Batch id to scan with, coming from `Initiate Batch` (If it is not given, it will be a single file scan.)
archivepwd	string		password for archive (URL encoded UTF-8 string) Multiple passwords is also supported, format: archivepwdX * X: Could be empty * When having value, X must be a number >= 1 For example: * archivepwd1: "fox" * archivepwd2: "cow" * archivepwd3: "bear"
metadata	json		could be utilized for: Additional parameter for pre-defined post actions and external scanners (as a part of STDIN input). Customized macro variable for watermarking text (Proactive DLP engine feature). Additional context / verbose information for each file submission (appended into JSON response scan result).

NAME	TYPE	EXAMPLE	DESCRIPTION
engines-metadata	json		Since MetaDefender Core 5.0.0, preferred context / verbose information can be sent to the engines. Please see the below pages for the details: * [File Type engine](https://docs.opswat.com/mdcore/utilities-engines/supported-engines-metadata) (supported since Core 5.2.1) * [Archive engine](https://docs.opswat.com/mdcore/utilities-engines/supported-engines-metadata-header) (supported since Core 5.4.1) * [Deep CDR](https://docs.opswat.com/mdcore/deep-cdr/supported-engines-metadata-json) * [Proactive DLP](https://docs.opswat.com/mdcore/proactive-dlp/supported-engines-metadata-json)
callbackurl	uri		Client's URL where MetaDefender Core will notify scan result back to whenever scan is finished (webhooks model). * Format: \<protocol://\>\<ip | domain\>:\<port\>\</path\> * Example: http://10.0.1.100:8081/listenback * Supported protocol: HTTP / HTTPS * Method: POST
sanitizedurl	uri		Client's URL where MetaDefender Core will send the sanitized file as the body of a POST request back to whenever scan is finished (webhooks model). * Format: \<protocol://\>\<ip | domain\>:\<port\>\</path\> * Example: http://10.0.1.100:8081/listensanitizedfile * Supported protocol: HTTP / HTTPS * Method: POST * Mandatory headers: * `user-agent`: "OPSWAT MetaDefender Core V4" * `dataid`: "<data id of original file submission>" * `sanitization-result`: "Sanitized Successfully" or "Sanitized Partially" or "Sanitized File Removed" * `sanitized-file-hash`: "<SHA256 hash value of sanitized file>" * `Content-Disposition`: attachment; filename="<name of sanitized file>" * `Content-Length`: <actual size of sanitized file>

NAME	TYPE	EXAMPLE	DESCRIPTION
downloadfrom	uri		Format: \<protocol://\>\<ip | domain\>:\<port\>\</path\> Since MetaDefender Core 4.19.1, the product started supporting users to process file by just specifying a direct download link. When this header is specified, users must not have body content in HTTP(S) request, otherwise expecting to hit 400 HTTP(S) error. Check response sample of HTTP 400 for details. Specify download link where MetaDefender Core could download the entire payload for processing. * Supported protocol: HTTP / HTTPS * Support both * Individual file scan (both webhook & non-webhook) * File processing in batch via link * Simply use `downloadfrom` header for each file scan request in a batch, no further requirement on batch request itself to utilize scan via link feature. * `Note`: Cancelling a batch will result in auto cancelling all registered individual scan requests which are still in downloading status * Configurable setting under workflow rule (under "SCAN" tab): * Download timeout * Max file size to download * Support to return status of download progress back to client in HTTP(S) response * Pre-check supported to refuse downloading if the file size does not meet configured conditions, and/or MetaDefender Core's temp folder does not have sufficient disk space to save the downloaded file Limitation: * Redirect link is not supported * Retry to download is not supported
global-timeout	integer		This custom global timeout (in seconds) will override the global timeout predefined in corresponding workflow rule.

RESPONSE

STATUS CODE - 200: Successful file submission

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
data_id*	string	Unique submission identifier. Use this value to reference the submission.

EXAMPLE:

```
{
 "data_id": "61dffeaa728844adbf49eb090e4ece0e"
}
```

STATUS CODE - 400: Callbackurl and/or apikey is invalid.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
ONE:OF	object	
OPTION:1	object	
err	string	Both body data and download link were given.
OPTION:2	object	
err	string	Callback url is invalid.
OPTION:3	object	
err	string	File upload rejected due to insufficient disk space

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 411: Content-Length header is missing from the request.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	

EXAMPLE:

```
{
 "err": "Missing Content-Length header."
}
```

STATUS CODE - 422: Body input is empty.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	

EXAMPLE:

```
{
 "err": "File is empty."
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

STATUS CODE - 503: Server is too busy, scan queue is full. Try again later.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	

EXAMPLE:

```
{
 "err": "Server is too busy. Try again later."
}
```

3.2 POST /file/sync

Analyze File (Synchronous mode)

Note: Only available starting with MetaDefender Core 4.21.0

Synchronous (sync) scan mode does not support:

- Non-persistent scan mode (dbmode = 3 described in MetaDefender Core Configuration)
- Individual scan with sync mode linked to a batch closed with a callbackurl header (webhook)

Scanning a file using a specified workflow.

Scan is done **synchronously (blocking mode)**, the connection of file submission request must be held by client side until MetaDefender Core returns processing result of individual scan when finished (response code = 200), or return data ID associated to that scan request if timed out (response code = 408).

Note: In case of receiving data ID, expecting the client side to perform query polling for scan result / progress then, see more at Fetch Analysis Result

The timeout value could be configurable, see more at Synchronous scan connection timeout - Modification

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.
filename	string		Specify name of file captured and displayed on corresponding MetaDefender Core scan result
filepath	string		if local file scan is enabled the path to the file (see Security rule configuration)

NAME	TYPE	EXAMPLE	DESCRIPTION
user_agent	string		user_agent header used to identify (and limit) access to a particular rule. For rule selection, `rule` header should be used. Since MetaDefender Core 5.6.0, when user agent is matched the predefined setting in workflow rule, the product will return X-Core-Id containing identity of MetaDefender Core server as a header of response.
rule	string		Select rule for the analysis, if no header given the default rule will be selected (URL encoded UTF-8 string of rule name)
workflow	string		name of the selected workflow (deprecated, use "rule" header parameter instead)
batch	string		Batch id to scan with, coming from `Initiate Batch` (If it is not given, it will be a single file scan.)
archivepwd	string		password for archive (URL encoded UTF-8 string) Multiple passwords is also supported, format: archivepwd<X> * X: Could be empty * When having value, X must be a number >= 1 For example: * archivepwd1: "fox" * archivepwd2: "cow" * archivepwd3: "bear"
metadata	json		could be utilized for: Additional parameter for pre-defined post actions and external scanners (as a part of STDIN input). Customized macro variable for watermarking text (Proactive DLP engine feature). Additional context / verbose information for each file submission (appended into JSON response scan result).
engines-metadata	json		Since MetaDefender Core 5.0.0, preferred context / verbose information can be sent to the engines. Please see the below pages for the details: * [File Type engine](https://docs.opswat.com/mdcore/utilities-engines/supported-engines-metadata) (supported since Core 5.2.1) * [Archive engine](https://docs.opswat.com/mdcore/utilities-engines/supported-engines-metadata-header) (supported since Core 5.4.1) * [Deep CDR](https://docs.opswat.com/mdcore/deep-cdr/supported-engines-metadata-json) * [Proactive DLP](https://docs.opswat.com/mdcore/proactive-dlp/supported-engines-metadata-json)

NAME	TYPE	EXAMPLE	DESCRIPTION
downloadfrom	uri		Format: \<protocol://\>\<ip | domain\>:\<port\>\</path\> When this header is specified, users must not have body content in HTTP(S) request, otherwise expecting to hit 400 HTTP(S) error. Check response sample of HTTP 400 for details. Specify download link where MetaDefender Core could download the entire payload for processing. * Supported protocol: HTTP / HTTPS * Support both individual file scan (both webhook & non-webhook) and file processing in batch via link * Configurable setting under workflow rule (under "SCAN" tab): * Download timeout * Max file size to download * Support to return status of download progress back to client in HTTP(S) response * Pre-check supported to refuse downloading if the file size does not meet configured conditions, and/or MetaDefender Core's temp folder does not have sufficient disk space to save the downloaded file Limitation: * Redirect link is not supported * Retry to download is not supported

RESPONSE

STATUS CODE - 200: The scan is completed before the connection timeout occurs

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
data_id	string	data identifier of the requested file
dlp_info	object	
certainty	enum	**ALLOWED:** Very Low, Low, Medium, High, Very High Describes how certain the hit is, possible values: * `Very Low` * `Low` * `Medium` * `High` * `Very High`
errors	object	
filename	string	Output processed file name (pre-configured on engine settings under Core's worflow rule)
hits	object	

NAME	TYPE	DESCRIPTION
ccn	object	
display_name	string	Credit Card Number, Social Security Number, or in case of RegEx, the name of the rule that has been given by the user
hits	array	
after	string	The context after the matched data.
before	string	The context before the matched data.
certainty	enum	**ALLOWED:** `Very Low, Low, Medium, High, Very High` The text version of "certainty_score", possible values: * `Very Low` * `Low` * `Medium` * `High` * `Very High`
certainty_score	integer	Is defined by the relevance of the given hit in its context. It is calculated based on multiple factors such as the number of digits, possible values: [0-100]
hit	string	The matched data.
location	string	The location of the hit that is found in a file.
severity	enum	**ALLOWED:** `0, 1` (NOTE: this field is deprecated): can be 0 (detected) or 1 (suspicious).
tryRedact	boolean	If file was redacted or not.
metadata_removal	object	
result	enum	**ALLOWED:** `removed, not removed, failed to remove` Result of the metadata removal process, possible values: * `removed` * `not removed` * `failed to remove`
redact	object	
result	enum	**ALLOWED:** `redacted, not redacted, failed to redact` Result of the redaction process, possible values: * `redacted` * `not redacted` * `failed to redact`
severity	enum	**ALLOWED:** `0, 1` (NOTE: this field is deprecated): represents the severity of the data loss, possible values: * `0` - Certainly is data loss * `1` - Might be data loss
verdict	enum	**ALLOWED:** `0, 1, 2, 3, 4` The overall result for the scanned file. Possible values: * `0` - Clean * `1` - Found matched data * `2` - Suspicious * `3` - Failed * `4` - Not scanned
watermark	object	

NAME	TYPE	DESCRIPTION
result	enum	**ALLOWED:** added, not added, failed to add Result of the watermarking process, possible values: * `added` * `not added` * `failed to add`
download_info	object	
error_detail	string	Revealed detailed reason why the download failed.
progress	number	Only applicable when "status" is `Downloading`, indicates download finished percentage, in a range of [1, 99]. * Once hitting 100, the status will be changed to `Download Success`. * or other problematic status (`Download Cancelled`, `Download Failed`) if the download stopped unexpectedly.

status	string	Indicates download status, which could be either
		- `Downloading`
		- Check `progress` key value for actual download percentage
		``json
		"download_info": {
		"progress": 7,
		"status": "Downloading",
		"url": "http://192.168.200.97:8080/5gb.zip"
		}
		``
		- `Download Success`
		``json
		"download_info": {
		"status": "Download Success",
		"url": "https://secure.eicar.org/eicar.com"
		}
		``
		- `Download Failed`
		- Check `error_detail` key value for an error explanation
		``json
		"download_info": {
		"error_detail": "Connection error",
		"status": "Download Failed",
		"url": "http://192.168.200.97:8080/2gb.zip"
		}
		``
		- `Download Timeout`
		- Expecting to occur when the download progress takes longer than what time window allowed in MetaDefender Core's pre-configured setting under workflow rule (under "SCAN" tab)
		``json
		"download_info": {
		"status": "Download Timeout",
		"url": "http://192.168.200.97:8080/2gb.zip"
		}
		``
		- `Download Cancelled`
		- Expecting to occur when user explicitly cancelled that file scan request, or batch request that the scan belongs to
		``json
		"download_info": {
		"status": "Download Cancelled",
		"url": "http://192.168.200.97:8080/5gb.zip"
		}
		``

NAME	TYPE	DESCRIPTION
url	string	Original download link which was specified in HTTP(S) request's `downloadfrom` header
extraction_info	object	
decrypted_status	enum	**ALLOWED:** `Success, Failed` Indicate that decryption phase is successful or not.
err_category	string	Error category
err_code	integer	Error code
err_details	string	Error message
is_encrypted_file	boolean	Indicate if file is password-protected or not.
file_info	object	
display_name	string	The filename reported via `filename` header.
file_size	integer	Total file size in bytes.
file_type	string	The filetype using mimetype.
file_type_description	string	The filetype in human readable format.
md5	string	File's MD5 hash.
sha1	string	File's SHA1 hash.
sha256	string	File's SHA256 Hash.
sha512	string	File's SHA512 Hash.
signer_infos	array	
digest_algorithm	string	Digest algorithm.
digest_encryption_algorithm	string	Encryption algorithm.
issuer	string	Entity that develops and registers the certificate.
serial_number	string	Serial number of the certificate.
vendor_name	string	Entity that is issued a certificate and utilize it for creating a digital signature.
version	string	Version of X.509 that is used in the certificate. This version field is zero-based. * 0: v1 * 1: v2 * 2: v3
type_category	array	
receive_data_timestamp	string	The timestamp when upload progress started (first byte received) (in milliseconds)
upload_time	integer	Total time elapsed for upload process (in milliseconds).
upload_timestamp	string	The timestamp when upload progress finished (all bytes received) (in milliseconds)
filetype_info	object	
file_info*	object	

NAME	TYPE	DESCRIPTION
description*	string	File type description
encrypted*	boolean	File is password-protected or not
extensions*	string	File type extension
groupID*	string	File type category
groupIDs*	array	
type*	string	MIME type
typeID*	string	File type ID
type_ids*	array	
file_info_details	object	
file_metadata	object	
result_template_hash	string	SHA256 Hash of user-interface template. For web console only.
opswatfilescan_info	object	
process_info	object	
blocked_reason	string	Provides the reason why the file is blocked (if so).
blocked_reasons	array	
file_type_skipped_scan	boolean	Indicates if the input file's detected type was configured to skip scanning.
hash_time	integer	Total time elapsed for computing hashes (in milliseconds).
outdated_data	array	
processing_time	integer	Total time elapsed during processing file (in milliseconds).
processing_time_details	object	
av_scan_time	integer	AV engines' processing time.
cdr_time	integer	Deep CDR engine's sanitization time.
dlp_time	integer	Proactive DLP engine's processing time.
extraction_time	integer	Archive extraction engine's processing time.
filetype_time	integer	FileType engine's processing time.
opswatfilescan_time	integer	OPSWAT Filescan engine's processing time.
others_time	integer	Total time elapsed for following processing tasks in the product (in milliseconds): * Decryption time (if receiving an encrypted file) * External scanner (if configured) * Post action (if configured) * Other internal processing time among components in the product
parse_dgsg_time	integer	Digital signature analyzing time.
vul_time	integer	Vulnerability engine's lookup time.
yara_time	integer	YARA engine's processing time.
profile	string	The used rule name.

NAME	TYPE	DESCRIPTION
progress_percentage	integer	Percentage of processing completed (from 1-100).
queue_time	integer	Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue_time = start_time - upload_timestamp) (in milliseconds).
result	string	The final result of processing the file (Allowed / Blocked / Processing).
user_agent	string	Identifier for the REST Client that calls the API.
username	string	User identifier who submitted scan request earlier.
verdicts	array	
post_processing	object	
actions_failed	string	Empty string if no action failed or list of failed actions, separated by "|".
actions_ran	string	List of successful actions, separated by "|". Empty string if otherwise.
converted_destination	string	Contains the name of the sanitized file.
converted_to	string	Contains target type name of sanitization.
copy_move_destination	string	Contains target type name of sanitization.
sanitization_details	object	
description	string	Action was successful or not.
details	array	
action*	enum	**ALLOWED:** `sanitized, removed` The type of action that was performed
count	integer	The number of objects that were sanitized/removed.
details	object	
action	enum	**ALLOWED:** `sanitized, removed` The type of action that was performed
count	integer	The number of objects that were sanitized/removed.
object_details	array	
object_name	string	The object type that was sanitized/removed.
description	string	Action was successful or not.
file_name	string	If an embedded file was sanitized.
object_details	array	
object_name*	string	The object type that was sanitized/removed.
failure_category	string	Deep CDR errors are classified into different categories. For more details, please find [Troubleshooting sanitization failures](https://docs.opswat.com/mdcore/deep-cdr/troubleshooting-sanitization-failures)
sanitized_file_info	object	
file_size	integer	Size of sanitized file in bytes.
sha256	string	SHA256 hash of sanitized file.

NAME	TYPE	DESCRIPTION
scan_results	object	
data_id	string	Data ID of the requested file
progress_percentage	integer	Track analysis progress until reaches 100.
scan_all_result_a	enum	**ALLOWED:** No Threat Detected, Infected, Suspicious, Failed, Whitelisted, Blacklisted, Exceeded Archive Depth, Not Scanned, Encrypted Archive, Exceeded Archive Size, Exceeded Archive File Number, Password Protected Document, Exceeded Archive Timeout, Mismatch, Potentially Vulnerable File, Cancelled, Sensitive Data Found, Yara Rule Matched, Potentially Unwanted, Unsupported File Type, Extraction Failed, Scan Failed, Suspicious Verdict by Sandbox, Likely Malicious Verdict by Sandbox, Malicious Verdict by Sandbox, Blocked Verdict by Sandbox, Blocked Verdict by Deep CDR, Global Timeout Exceeded, Vulnerable Verdict by SBOM, Non-vulnerable Verdict by SBOM, Blocked Verdict by SBOM, Blocked by Post Action, Known Bad, Known Good, Unknown, Allowed Verdict by COO, Blocked Verdict by COO, Unknown Verdict by COO, In Progress, Skip Processing Fast Symlink The overall scan result as string
scan_all_result_i	enum	**ALLOWED:** 0, 1, 2, 3, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36, 38, 39, 40, 41, 42, 43, 255, 1014 The overall scan result as index in the Processing Results table.
start_time	string	Timestamp when the scanning process starts.
total_avs	integer	Total number of scanning engines used as part of this analysis.
total_time	integer	Total time elapsed during scan (in milliseconds).
scan_details	object	
ClamAV	object	
def_time	string	The database definition time for this engine
eng_id	string	The unique identification string for the engine
location	string	Where this engine is deployed (local/remote).
scan_result_i	integer	Scan result as index in the Processing Results table
scan_time	integer	The time elapsed during scan with this engine (in milliseconds).
threat_found	string	The threat name, IF scan result is Infected or Suspicious. Otherwise empty string or error message from the engine.
wait_time	integer	Time elapsed between sending file to Core and receiving the result from the engine (in milliseconds).
vulnerability_info	object	
result	object	
code	integer	The result code for vulnerability check, 0 means a successful check
hash	string	The file's SHA1 hash value

NAME	TYPE	DESCRIPTION
method	enum	**ALLOWED:** `50700` The method used by OESIS Framework, it should be 50700 every time.
timestamp	string	Timestamp of the request issued
timing	integer	The vulnerability check's duration in milliseconds
detected_product	object	
has_kb	boolean	Indicates whether any KBs or MSBs exist for this hash
has_vulnerability	boolean	Indicates whether any vulnerabilities have been associated with the particular product
is_current	boolean	True if this product's patch level is current, defaults to true
product	object	
id	integer	The OPSWAT product id
name	string	The product name
remediation_link	string	A link where product updates or patches can be obtained
severity	enum	**ALLOWED:** `LOW, MODERATE, IMPORTANT, CRITICAL, NOT_AVAILABLE, UNKNOWN` String description of Severity level: * `LOW` * `MODERATE` * `IMPORTANT` * `CRITICAL` * `NOT_AVAILABLE` * `UNKNOWN`
sig_name	string	Product signature descriptor
signature	integer	OPSWAT signature id
vendor	object	
id	integer	The OPSWAT vendor id
name	string	The vendor name
version	string	The installed product version
version_data	object	
count_behind	integer	The number of patches behind of the installed product
feed_id	integer	The remote feed ID used to determine patch level
version	string	The current version of the product in the remote feed
vulnerabilites	array	
description	string	A text description of the specific vulnerability
details	object	
cpe	string	A CPE product reference
cve	string	A CVE identification string
cvss	object	

NAME	TYPE	DESCRIPTION
access-complexity	string	A CVSS access-complexity descriptor
access-vector	string	A CVSS access-vector descriptor
authentication	string	A CVSS authentication descriptor
availability-impact	string	A CVSS availability impact descriptor
confidentiality-impact	string	A CVSS confidentiality impact descriptor
generated-on-epoch	string	An epoch timestamp indicating CVSS generation time
integrity-impact	string	A CVSS integrity impact descriptor
score	string	A CVSS 10-point severity score
source	string	A CVSS source descriptor
cwe	string	A CWE group identification string
last_modified_epoch	string	An epoch timestamp indicating source last update time
published-epoch	string	An epoch timestamp indicating source publishing time
references	array	
severity	enum	**ALLOWED:** `LOW, MODERATE, IMPORTANT, CRITICAL, NOT_AVAILABLE, UNKNOWN` String description of Severity level: * `LOW` * `MODERATE` * `IMPORTANT` * `CRITICAL` * `NOT_AVAILABLE` * `UNKNOWN`
severity_index	integer	A 5 point scale numerical description of Severity level with 5 being greatest and 0 being unknown
static_id	integer	An OPSWAT identifier for the vulnerability
verdict	integer	The vulnerability check's duration in milliseconds * `0` - No Vulnerability Found * `1` - Vulnerability Found * `3` - Failed * `16` - Processing Timed Out
yara	object	
hits	object	
verdict	enum	**ALLOWED:** `0, 1, 2, 3, 4` The overall result for the analyzed file. Value will be one of the following: | index | status | |---|---| | 0 | Clean | | 1 | Found matched data | | 2 | Suspicious | | 3 | Failed | | 4 | Not scanned |

STATUS CODE - 400: API key is invalid.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
ONE:OF	object	
OPTION:1	object	
err	string	Invalid apikey given
OPTION:2	object	
err	string	Both body data and download link were given.
OPTION:3	object	
err	string	Batch closed during file upload.
OPTION:4	object	
err	string	Can not scan in given batch.
OPTION:5	object	
err	string	Cannot scan batch in another instance.
OPTION:6	object	
err	string	Both body data and local file path were given.
OPTION:7	object	
err	string	File upload rejected due to insufficient disk space
OPTION:8	object	
err	string	Redirect link not supported.
OPTION:9	object	
err	string	Invalid download url.
OPTION:10	object	
err	string	Download url must start with http:// or https://.

STATUS CODE - 403: Callback URL is not supported.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	callbackurl is not supported

STATUS CODE - 408: Timed out response. The scan is still in progress, but the connection is timed out.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		

NAME	TYPE	DESCRIPTION
data_id	string	

EXAMPLE:

```
{
 "data_id": "61dffeaa728844adbf49eb090e4ece0e"
}
```

STATUS CODE - 411: Content-Length header is missing from the request.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	

EXAMPLE:

```
{
 "err": "Missing Content-Length header."
}
```

STATUS CODE - 422: Body input is empty.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	

EXAMPLE:

```
{
 "err": "File is empty."
}
```

STATUS CODE - 500: Internal Server Error.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
ONE:OF	object	
OPTION:1	object	
err	string	License is expired
OPTION:2	object	
err	string	No available rule is present for scanning.
OPTION:3	object	

NAME	TYPE	DESCRIPTION
err	string	Failed to request scan. Try again later.
OPTION:4	object	
err	string	File not found, invalid path or access.
OPTION:5	object	
err	string	Failed to request scan. File size exceeded the maximum size permitted by your configuration.

STATUS CODE - 503: Server is too busy, scan queue is full. Try again later.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	

EXAMPLE:

```
{
 "err": "Server is too busy. Try again later."
}
```

3.3 GET /file/{data_id}

Fetch Analysis Result

Retrieve scan results.

Scan is done asynchronously and each scan request is tracked by a data ID.

Initiating file scans and retrieving the results need to be done using two separate API calls. This request needs to be made multiple times until the scan is complete. Scan completion can be traced using scan_results.progress_percentage value from the response.

Note: The REST API also supports pagination for archive file result. A completed response description with archive detection:

- extracted_files: information about extracted files

- files_extracted_count: the number of extracted files
- files_in_archive: array of files in archive
  - detected_by: number of engines reported threat
  - scanned_with: number of engines used for scanning the file

- first_index: it tells that from which file (index of the file, 0 is the first) the result JSON contains information about extracted files. (default=0)
- page_size: it tells how many files the result JSON contains information about (default=50). So by default, the result JSON contains information about the first 50 extracted files.
- worst_data_id: data id of the file that has the worst result in the archive

- scan_results
  - last_file_scanned (stored only in memory, not in database): If available, the name of the most recent processed file

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*data_id	string		Unique submission identifier. Use this value to reference the submission.

QUERY PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
first	integer		The first item order in the list child files of archive file
size	integer		The number of items to be fetched next, counting from the item order indicated in first header

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

NAME	TYPE	EXAMPLE	DESCRIPTION
user_agent	string		user_agent header used to identify (and limit) access to a particular rule. For rule selection, `rule` header should be used. Since MetaDefender Core 5.6.0, when user agent is matched the predefined setting in workflow rule, the product will return X-Core-Id containing identity of MetaDefender Core server as a header of response.

RESPONSE

STATUS CODE - 200: Entire analysis report generated by MetaDefender Core

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
data_id	string	data identifier of the requested file
dlp_info	object	
certainty	enum	**ALLOWED:** Very Low, Low, Medium, High, Very High Describes how certain the hit is, possible values: * `Very Low` * `Low` * `Medium` * `High` * `Very High`
errors	object	
filename	string	Output processed file name (pre-configured on engine settings under Core's worflow rule)
hits	object	
ccn	object	
display_name	string	Credit Card Number, Social Security Number, or in case of RegEx, the name of the rule that has been given by the user
hits	array	
after	string	The context after the matched data.
before	string	The context before the matched data.
certainty	enum	**ALLOWED:** Very Low, Low, Medium, High, Very High The text version of "certainty_score", possible values: * `Very Low` * `Low` * `Medium` * `High` * `Very High`
certainty_score	integer	Is defined by the relevance of the given hit in its context. It is calculated based on multiple factors such as the number of digits, possible values: [0-100]
hit	string	The matched data.

NAME	TYPE	DESCRIPTION
location	string	The location of the hit that is found in a file.
severity	enum	**ALLOWED:** `0, 1` (NOTE: this field is deprecated): can be 0 (detected) or 1 (suspicious).
tryRedact	boolean	If file was redacted or not.
metadata_removal	object	
result	enum	**ALLOWED:** `removed, not removed, failed to remove` Result of the metadata removal process, possible values: * `removed` * `not removed` * `failed to remove`
redact	object	
result	enum	**ALLOWED:** `redacted, not redacted, failed to redact` Result of the redaction process, possible values: * `redacted` * `not redacted` * `failed to redact`
severity	enum	**ALLOWED:** `0, 1` (NOTE: this field is deprecated): represents the severity of the data loss, possible values: * `0` - Certainly is data loss * `1` - Might be data loss
verdict	enum	**ALLOWED:** `0, 1, 2, 3, 4` The overall result for the scanned file. Possible values: * `0` - Clean * `1` - Found matched data * `2` - Suspicious * `3` - Failed * `4` - Not scanned
watermark	object	
result	enum	**ALLOWED:** `added, not added, failed to add` Result of the watermarking process, possible values: * `added` * `not added` * `failed to add`
download_info	object	
error_detail	string	Revealed detailed reason why the download failed.
progress	number	Only applicable when "status" is `Downloading`, indicates download finished percentage, in a range of [1, 99]. * Once hitting 100, the status will be changed to `Download Success`. * or other problematic status (`Download Cancelled`, `Download Failed`) if the download stopped unexpectedly.

NAME	TYPE	DESCRIPTION

status	string	Indicates download status, which could be either
		- `Downloading`
		- Check `progress` key value for actual download percentage
		``json
		"download_info": {
		"progress": 7,
		"status": "Downloading",
		"url": "http://192.168.200.97:8080/5gb.zip"
		}
		``
		- `Download Success`
		``json
		"download_info": {
		"status": "Download Success",
		"url": "https://secure.eicar.org/eicar.com"
		}
		``
		- `Download Failed`
		- Check `error_detail` key value for an error explanation
		``json
		"download_info": {
		"error_detail": "Connection error",
		"status": "Download Failed",
		"url": "http://192.168.200.97:8080/2gb.zip"
		}
		``
		- `Download Timeout`
		- Expecting to occur when the download progress takes longer than what time window allowed in MetaDefender Core's pre-configured setting under workflow rule (under "SCAN" tab)
		``json
		"download_info": {
		"status": "Download Timeout",
		"url": "http://192.168.200.97:8080/2gb.zip"
		}
		``
		- `Download Cancelled`
		- Expecting to occur when user explicitly cancelled that file scan request, or batch request that the scan belongs to
		``json
		"download_info": {
		"status": "Download Cancelled",
		"url": "http://192.168.200.97:8080/5gb.zip"
		}
		``

url	string	Original download link which was specified in HTTP(S) request's `downloadfrom` header
extraction_info	object	
decrypted_status	enum	**ALLOWED:** `Success, Failed` Indicate that decryption phase is successful or not.
err_category	string	Error category
err_code	integer	Error code
err_details	string	Error message
is_encrypted_file	boolean	Indicate if file is password-protected or not.

NAME	TYPE	DESCRIPTION
file_info	object	
display_name	string	The filename reported via `filename` header.
file_size	integer	Total file size in bytes.
file_type	string	The filetype using mimetype.
file_type_description	string	The filetype in human readable format.
md5	string	File's MD5 hash.
sha1	string	File's SHA1 hash.
sha256	string	File's SHA256 Hash.
sha512	string	File's SHA512 Hash.
signer_infos	array	
digest_algorithm	string	Digest algorithm.
digest_encryption_algorithm	string	Encryption algorithm.
issuer	string	Entity that develops and registers the certificate.
serial_number	string	Serial number of the certificate.
vendor_name	string	Entity that is issued a certificate and utilize it for creating a digital signature.
version	string	Version of X.509 that is used in the certificate. This version field is zero-based. * 0: v1 * 1: v2 * 2: v3
type_category	array	
receive_data_timestamp	string	The timestamp when upload progress started (first byte received) (in milliseconds)
upload_time	integer	Total time elapsed for upload process (in milliseconds).
upload_timestamp	string	The timestamp when upload progress finished (all bytes received) (in milliseconds)
filetype_info	object	
file_info*	object	
description*	string	File type description
encrypted*	boolean	File is password-protected or not
extensions*	string	File type extension
groupID*	string	File type category
groupIDs*	array	
type*	string	MIME type
typeID*	string	File type ID
type_ids*	array	
file_info_details	object	

NAME	TYPE	DESCRIPTION
file_metadata	object	
result_template_hash	string	SHA256 Hash of user-interface template. For web console only.
opswatfilescan_info	object	
process_info	object	
blocked_reason	string	Provides the reason why the file is blocked (if so).
blocked_reasons	array	
file_type_skipped_scan	boolean	Indicates if the input file's detected type was configured to skip scanning.
hash_time	integer	Total time elapsed for computing hashes (in milliseconds).
outdated_data	array	
processing_time	integer	Total time elapsed during processing file (in milliseconds).
processing_time_details	object	
av_scan_time	integer	AV engines' processing time.
cdr_time	integer	Deep CDR engine's sanitization time.
dlp_time	integer	Proactive DLP engine's processing time.
extraction_time	integer	Archive extraction engine's processing time.
filetype_time	integer	FileType engine's processing time.
opswatfilescan_time	integer	OPSWAT Filescan engine's processing time.
others_time	integer	Total time elapsed for following processing tasks in the product (in milliseconds): * Decryption time (if receiving an encrypted file) * External scanner (if configured) * Post action (if configured) * Other internal processing time among components in the product
parse_dgsg_time	integer	Digital signature analyzing time.
vul_time	integer	Vulnerability engine's lookup time.
yara_time	integer	YARA engine's processing time.
profile	string	The used rule name.
progress_percentage	integer	Percentage of processing completed (from 1-100).
queue_time	integer	Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue_time = start_time - upload_timestamp) (in milliseconds).
result	string	The final result of processing the file (Allowed / Blocked / Processing).
user_agent	string	Identifier for the REST Client that calls the API.
username	string	User identifier who submitted scan request earlier.
verdicts	array	
post_processing	object	
actions_failed	string	Empty string if no action failed or list of failed actions, separated by "|".

NAME	TYPE	DESCRIPTION
actions_ran	string	List of successful actions, separated by "|". Empty string if otherwise.
converted_destination	string	Contains the name of the sanitized file.
converted_to	string	Contains target type name of sanitization.
copy_move_destination	string	Contains target type name of sanitization.
sanitization_details	object	
description	string	Action was successful or not.
details	array	
action*	enum	**ALLOWED:** `sanitized, removed` The type of action that was performed
count	integer	The number of objects that were sanitized/removed.
details	object	
action	enum	**ALLOWED:** `sanitized, removed` The type of action that was performed
count	integer	The number of objects that were sanitized/removed.
object_details	array	
object_name	string	The object type that was sanitized/removed.
description	string	Action was successful or not.
file_name	string	If an embedded file was sanitized.
object_details	array	
object_name*	string	The object type that was sanitized/removed.
failure_category	string	Deep CDR errors are classified into different categories. For more details, please find [Troubleshooting sanitization failures](https://docs.opswat.com/mdcore/deep-cdr/troubleshooting-sanitization-failures)
sanitized_file_info	object	
file_size	integer	Size of sanitized file in bytes.
sha256	string	SHA256 hash of sanitized file.
scan_results	object	
data_id	string	Data ID of the requested file
progress_percentage	integer	Track analysis progress until reaches 100.

NAME	TYPE	DESCRIPTION
scan_all_result_a	enum	**ALLOWED:** No Threat Detected, Infected, Suspicious, Failed, Whitelisted, Blacklisted, Exceeded Archive Depth, Not Scanned, Encrypted Archive, Exceeded Archive Size, Exceeded Archive File Number, Password Protected Document, Exceeded Archive Timeout, Mismatch, Potentially Vulnerable File, Cancelled, Sensitive Data Found, Yara Rule Matched, Potentially Unwanted, Unsupported File Type, Extraction Failed, Scan Failed, Suspicious Verdict by Sandbox, Likely Malicious Verdict by Sandbox, Malicious Verdict by Sandbox, Blocked Verdict by Sandbox, Blocked Verdict by Deep CDR, Global Timeout Exceeded, Vulnerable Verdict by SBOM, Non-vulnerable Verdict by SBOM, Blocked Verdict by SBOM, Blocked by Post Action, Known Bad, Known Good, Unknown, Allowed Verdict by COO, Blocked Verdict by COO, Unknown Verdict by COO, In Progress, Skip Processing Fast Symlink The overall scan result as string
scan_all_result_i	enum	**ALLOWED:** 0, 1, 2, 3, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36, 38, 39, 40, 41, 42, 43, 255, 1014 The overall scan result as index in the Processing Results table.
start_time	string	Timestamp when the scanning process starts.
total_avs	integer	Total number of scanning engines used as part of this analysis.
total_time	integer	Total time elapsed during scan (in milliseconds).
scan_details	object	
ClamAV	object	
def_time	string	The database definition time for this engine
eng_id	string	The unique identification string for the engine
location	string	Where this engine is deployed (local/remote).
scan_result_i	integer	Scan result as index in the Processing Results table
scan_time	integer	The time elapsed during scan with this engine (in milliseconds).
threat_found	string	The threat name, IF scan result is Infected or Suspicious. Otherwise empty string or error message from the engine.
wait_time	integer	Time elapsed between sending file to Core and receiving the result from the engine (in milliseconds).
vulnerability_info	object	
result	object	
code	integer	The result code for vulnerability check, 0 means a successful check
hash	string	The file's SHA1 hash value
method	enum	**ALLOWED:** 50700 The method used by OESIS Framework, it should be 50700 every time.
timestamp	string	Timestamp of the request issued
timing	integer	The vulnerability check's duration in milliseconds

NAME	TYPE	DESCRIPTION
detected_product	object	
has_kb	boolean	Indicates whether any KBs or MSBs exist for this hash
has_vulnerability	boolean	Indicates whether any vulnerabilities have been associated with the particular product
is_current	boolean	True if this product's patch level is current, defaults to true
product	object	
id	integer	The OPSWAT product id
name	string	The product name
remediation_link	string	A link where product updates or patches can be obtained
severity	enum	**ALLOWED:** `LOW, MODERATE, IMPORTANT, CRITICAL, NOT_AVAILABLE, UNKNOWN` String description of Severity level: * `LOW` * `MODERATE` * `IMPORTANT` * `CRITICAL` * `NOT_AVAILABLE` * `UNKNOWN`
sig_name	string	Product signature descriptor
signature	integer	OPSWAT signature id
vendor	object	
id	integer	The OPSWAT vendor id
name	string	The vendor name
version	string	The installed product version
version_data	object	
count_behind	integer	The number of patches behind of the installed product
feed_id	integer	The remote feed ID used to determine patch level
version	string	The current version of the product in the remote feed
vulnerabilites	array	
description	string	A text description of the specific vulnerability
details	object	
cpe	string	A CPE product reference
cve	string	A CVE identification string
cvss	object	
access-complexity	string	A CVSS access-complexity descriptor
access-vector	string	A CVSS access-vector descriptor
authentication	string	A CVSS authentication descriptor
availability-impact	string	A CVSS availability impact descriptor

NAME	TYPE	DESCRIPTION
confidentiality-impact	string	A CVSS confidentiality impact descriptor
generated-on-epoch	string	An epoch timestamp indicating CVSS generation time
integrity-impact	string	A CVSS integrity impact descriptor
score	string	A CVSS 10-point severity score
source	string	A CVSS source descriptor
cwe	string	A CWE group identification string
last_modified_epoch	string	An epoch timestamp indicating source last update time
published-epoch	string	An epoch timestamp indicating source publishing time
references	array	
severity	enum	**ALLOWED:** `LOW, MODERATE, IMPORTANT, CRITICAL, NOT_AVAILABLE, UNKNOWN` String description of Severity level: * `LOW` * `MODERATE` * `IMPORTANT` * `CRITICAL` * `NOT_AVAILABLE` * `UNKNOWN`
severity_index	integer	A 5 point scale numerical description of Severity level with 5 being greatest and 0 being unknown
static_id	integer	An OPSWAT identifier for the vulnerability
verdict	integer	The vulnerability check's duration in milliseconds * `0` - No Vulnerability Found * `1` - Vulnerability Found * `3` - Failed * `16` - Processing Timed Out
yara	object	
hits	object	
verdict	enum	**ALLOWED:** `0, 1, 2, 3, 4` The overall result for the analyzed file. Value will be one of the following: | index | status | |---|---| | 0 | Clean | | 1 | Found matched data | | 2 | Suspicious | | 3 | Failed | | 4 | Not scanned |

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		

NAME	TYPE	DESCRIPTION
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

3.4 GET /file/{data_id}/blocked-leaves

Fetch a list of blocked leaf files inside archive

Retrieve a list of the deepest blocked files inside an input original archive file based on the data_id of that archive file. Leaf file is the one that not having any successful extracted file inside. In case there's no blocked file, user will see an empty array. This list only returns 100 items at most.

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*data_id	string	8101abae27be4d63859c55d9e0ed0135	The data_id comes from the result of `Analyze a file`. In case of sanitizing the content of an archive, the data_id of contained file can be found in `Fetch analysis result`.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Returns the list of blocked leaf files.

RESPONSE MODEL - application/octet-stream

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
details	array	
blocked_reasons	array	
engines	array	
engine	string	Engine's name
threat_name	string	Threat's name
data_id	string	Data identifier of this blocked leaf file.
display_name	string	The name of this blocked leaf file.
file_type	string	The filetype using mimetype.
parent_id	string	Data identifer of the original archive file.
path	string	Full path to this blocked leaf file.
process_info	object	
post_processing	object	
actions_failed	string	Empty string if no action failed or list of failed actions, separated by "|".
actions_ran	string	List of successful actions, separated by "|". Empty string if otherwise.
verdicts	array	
limit_reached	boolean	Indicates if there're indeed over 100 blocked leaf files.
total	integer	Total number of files in the above array.

STATUS CODE - 404: Requests resource was not found.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Not found"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

3.5 GET /archive/{data_id}

Fetch analysis reports containing all files in archive

Retrieving Analysis Reports Using Data ID containing all files in archive.

Under extracted_files key, all the info about extracted files will be listed recursively.

Note: Using this method the following fields will not be shown compared to /file request containing extracted files:

- files_extracted_count
- files_in_archive
- first_index
- page_size
- worst_data_id

Also the "outdated_data" field will only be shown in the root archive.

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*data_id	string		Unique submission identifier. Use this value to reference the submission.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Entire analysis report containing all extracted files generated by MetaDefender Core

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
data_id	string	data identifier of the requested file
dlp_info	object	

NAME	TYPE	DESCRIPTION
certainty	enum	**ALLOWED:** `Very Low, Low, Medium, High, Very High` Describes how certain the hit is, possible values: * `Very Low` * `Low` * `Medium` * `High` * `Very High`
errors	object	
filename	string	Output processed file name (pre-configured on engine settings under Core's worflow rule)
hits	object	
ccn	object	
display_name	string	Credit Card Number, Social Security Number, or in case of RegEx, the name of the rule that has been given by the user
hits	array	
after	string	The context after the matched data.
before	string	The context before the matched data.
certainty	enum	**ALLOWED:** `Very Low, Low, Medium, High, Very High` The text version of "certainty_score", possible values: * `Very Low` * `Low` * `Medium` * `High` * `Very High`
certainty_score	integer	Is defined by the relevance of the given hit in its context. It is calculated based on multiple factors such as the number of digits, possible values: [0-100]
hit	string	The matched data.
location	string	The location of the hit that is found in a file.
severity	enum	**ALLOWED:** `0, 1` (NOTE: this field is deprecated): can be 0 (detected) or 1 (suspicious).
tryRedact	boolean	If file was redacted or not.
metadata_removal	object	
result	enum	**ALLOWED:** `removed, not removed, failed to remove` Result of the metadata removal process, possible values: * `removed` * `not removed` * `failed to remove`
redact	object	
result	enum	**ALLOWED:** `redacted, not redacted, failed to redact` Result of the redaction process, possible values: * `redacted` * `not redacted` * `failed to redact`

NAME	TYPE	DESCRIPTION
severity	`enum`	**ALLOWED:** `0, 1` (NOTE: this field is deprecated): represents the severity of the data loss, possible values: * `0` - Certainly is data loss * `1` - Might be data loss
verdict	`enum`	**ALLOWED:** `0, 1, 2, 3, 4` The overall result for the scanned file. Possible values: * `0` - Clean * `1` - Found matched data * `2` - Suspicious * `3` - Failed * `4` - Not scanned
watermark	`object`	
result	`enum`	**ALLOWED:** `added, not added, failed to add` Result of the watermarking process, possible values: * `added` * `not added` * `failed to add`
download_info	`object`	
error_detail	`string`	Revealed detailed reason why the download failed.
progress	`number`	Only applicable when "status" is `Downloading`, indicates download finished percentage, in a range of [1, 99]. * Once hitting 100, the status will be changed to `Download Success`. * or other problematic status (`Download Cancelled`, `Download Failed`) if the download stopped unexpectedly.

NAME	TYPE	DESCRIPTION

status	string	Indicates download status, which could be either
		- `Downloading`
		- Check `progress` key value for actual download percentage
		``json
		"download_info": {
		"progress": 7,
		"status": "Downloading",
		"url": "http://192.168.200.97:8080/5gb.zip"
		}
		``
		- `Download Success`
		``json
		"download_info": {
		"status": "Download Success",
		"url": "https://secure.eicar.org/eicar.com"
		}
		``
		- `Download Failed`
		- Check `error_detail` key value for an error explanation
		``json
		"download_info": {
		"error_detail": "Connection error",
		"status": "Download Failed",
		"url": "http://192.168.200.97:8080/2gb.zip"
		}
		``
		- `Download Timeout`
		- Expecting to occur when the download progress takes longer than what time window allowed in MetaDefender Core's pre-configured setting under workflow rule (under "SCAN" tab)
		``json
		"download_info": {
		"status": "Download Timeout",
		"url": "http://192.168.200.97:8080/2gb.zip"
		}
		``
		- `Download Cancelled`
		- Expecting to occur when user explicitly cancelled that file scan request, or batch request that the scan belongs to
		``json
		"download_info": {
		"status": "Download Cancelled",
		"url": "http://192.168.200.97:8080/5gb.zip"
		}
		``

url	string	Original download link which was specified in HTTP(S) request's `downloadfrom` header
extracted_files	array	
data_id	string	data identifier of the requested file
dlp_info	object	

NAME	TYPE	DESCRIPTION
certainty	enum	**ALLOWED:** `Very Low, Low, Medium, High, Very High` Describes how certain the hit is, possible values: * `Very Low` * `Low` * `Medium` * `High` * `Very High`
errors	object	
filename	string	Output processed file name (pre-configured on engine settings under Core's worflow rule)
hits	object	
ccn	object	
display_name	string	Credit Card Number, Social Security Number, or in case of RegEx, the name of the rule that has been given by the user
hits	array	
after	string	The context after the matched data.
before	string	The context before the matched data.
certainty	enum	**ALLOWED:** `Very Low, Low, Medium, High, Very High` The text version of "certainty_score", possible values: * `Very Low` * `Low` * `Medium` * `High` * `Very High`
certainty_score	integer	Is defined by the relevance of the given hit in its context. It is calculated based on multiple factors such as the number of digits, possible values: [0-100]
hit	string	The matched data.
location	string	The location of the hit that is found in a file.
severity	enum	**ALLOWED:** `0, 1` (NOTE: this field is deprecated): can be 0 (detected) or 1 (suspicious).
tryRedact	boolean	If file was redacted or not.
metadata_removal	object	
result	enum	**ALLOWED:** `removed, not removed, failed to remove` Result of the metadata removal process, possible values: * `removed` * `not removed` * `failed to remove`
redact	object	
result	enum	**ALLOWED:** `redacted, not redacted, failed to redact` Result of the redaction process, possible values: * `redacted` * `not redacted` * `failed to redact`

NAME	TYPE	DESCRIPTION
severity	enum	**ALLOWED:** `0, 1` (NOTE: this field is deprecated): represents the severity of the data loss, possible values: * `0` - Certainly is data loss * `1` - Might be data loss
verdict	enum	**ALLOWED:** `0, 1, 2, 3, 4` The overall result for the scanned file. Possible values: * `0` - Clean * `1` - Found matched data * `2` - Suspicious * `3` - Failed * `4` - Not scanned
watermark	object	
result	enum	**ALLOWED:** `added, not added, failed to add` Result of the watermarking process, possible values: * `added` * `not added` * `failed to add`
extracted_files	array	
data_id	string	data identifier of the requested file
dlp_info	object	
certainty	enum	**ALLOWED:** `Very Low, Low, Medium, High, Very High` Describes how certain the hit is, possible values: * `Very Low` * `Low` * `Medium` * `High` * `Very High`
errors	object	
filename	string	Output processed file name (pre-configured on engine settings under Core's worflow rule)
hits	object	
ccn	object	
display_name	string	Credit Card Number, Social Security Number, or in case of RegEx, the name of the rule that has been given by the user
hits	array	
after	string	The context after the matched data.
before	string	The context before the matched data.
certainty	enum	**ALLOWED:** `Very Low, Low, Medium, High, Very High` The text version of "certainty_score", possible values: * `Very Low` * `Low` * `Medium` * `High` * `Very High`

NAME	TYPE	DESCRIPTION
certainty_score	integer	Is defined by the relevance of the given hit in its context. It is calculated based on multiple factors such as the number of digits, possible values: [0-100]
hit	string	The matched data.
location	string	The location of the hit that is found in a file.
severity	enum	**ALLOWED:** `0, 1` (NOTE: this field is deprecated): can be 0 (detected) or 1 (suspicious).
tryRedact	boolean	If file was redacted or not.
metadata_removal	object	
result	enum	**ALLOWED:** `removed, not removed, failed to remove` Result of the metadata removal process, possible values: * `removed` * `not removed` * `failed to remove`
redact	object	
result	enum	**ALLOWED:** `redacted, not redacted, failed to redact` Result of the redaction process, possible values: * `redacted` * `not redacted` * `failed to redact`
severity	enum	**ALLOWED:** `0, 1` (NOTE: this field is deprecated): represents the severity of the data loss, possible values: * `0` - Certainly is data loss * `1` - Might be data loss
verdict	enum	**ALLOWED:** `0, 1, 2, 3, 4` The overall result for the scanned file. Possible values: * `0` - Clean * `1` - Found matched data * `2` - Suspicious * `3` - Failed * `4` - Not scanned
watermark	object	
result	enum	**ALLOWED:** `added, not added, failed to add` Result of the watermarking process, possible values: * `added` * `not added` * `failed to add`
extraction_info	object	
decrypted_status	enum	**ALLOWED:** `Success, Failed` Indicate that decryption phase is successful or not.
err_category	string	Error category
err_code	integer	Error code
err_details	string	Error message
is_encrypted_file	boolean	Indicate if file is password-protected or not.

NAME	TYPE	DESCRIPTION
file_info	object	
display_name	string	The filename reported via `filename` header.
file_size	integer	Total file size in bytes.
file_type	string	The filetype using mimetype.
file_type_description	string	The filetype in human readable format.
md5	string	File's MD5 hash.
sha1	string	File's SHA1 hash.
sha256	string	File's SHA256 Hash.
sha512	string	File's SHA512 Hash.
signer_infos	array	
digest_algorithm	string	Digest algorithm.
digest_encryption_algorithm	string	Encryption algorithm.
issuer	string	Entity that develops and registers the certificate.
serial_number	string	Serial number of the certificate.
vendor_name	string	Entity that is issued a certificate and utilize it for creating a digital signature.
version	string	Version of X.509 that is used in the certificate. This version field is zero-based. * 0: v1 * 1: v2 * 2: v3
type_category	array	
receive_data_timestamp	string	The timestamp when upload progress started (first byte received) (in milliseconds)
upload_time	integer	Total time elapsed for upload process (in milliseconds).
upload_timestamp	string	The timestamp when upload progress finished (all bytes received) (in milliseconds)
filetype_info	object	
file_info*	object	
description*	string	File type description
encrypted*	boolean	File is password-protected or not
extensions*	string	File type extension
groupID*	string	File type category
groupIDs*	array	
type*	string	MIME type
typeID*	string	File type ID

NAME	TYPE	DESCRIPTION
type_ids*	array	
file_info_details	object	
file_metadata	object	
result_template_hash	string	SHA256 Hash of user-interface template. For web console only.
process_info	object	
blocked_reason	string	Provides the reason why the file is blocked (if so).
file_type_skipped_scan	boolean	Indicates if the input file's detected type was configured to skip scanning.
outdated_data	array	
processing_time	integer	Total time elapsed during processing file (in milliseconds).
processing_time_details	object	
av_scan_time	integer	AV engines' processing time.
cdr_time	integer	Deep CDR engine's sanitization time.
dlp_time	integer	Proactive DLP engine's processing time.
extraction_time	integer	Archive extraction engine's processing time.
filetype_time	integer	FileType engine's processing time.
opswatfilescan_time	integer	OPSWAT Filescan engine's processing time.
others_time	integer	Total time elapsed for following processing tasks in the product (in milliseconds): * File processing's hash calculation time * External scanner (if configured) * Post action (if configured) * Other internal procssing time among components in the product
vul_time	integer	Vulnerability engine's lookup time.
yara_time	integer	YARA engine's processing time.
profile	string	The used rule name.
progress_percentage	integer	Percentage of processing completed (from 1-100).
queue_time	integer	Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue_time = start_time - upload_timestamp) (in milliseconds).
result	string	The final result of processing the file (Allowed / Blocked / Processing).
user_agent	string	Identifier for the REST Client that calls the API.
username	string	User identifier who submitted scan request earlier.
verdicts	array	
post_processing	object	
actions_failed	string	Empty string if no action failed or list of failed actions, separated by "|".
actions_ran	string	List of successful actions, separated by "|". Empty string if otherwise.
converted_destination	string	Contains the name of the sanitized file.

NAME	TYPE	DESCRIPTION
converted_to	string	Contains target type name of sanitization.
copy_move_destination	string	Contains target type name of sanitization.
sanitization_details	object	
description	string	Action was successful or not.
details	array	
action*	enum	**ALLOWED:** `sanitized, removed` The type of action that was performed
count	integer	The number of objects that were sanitized/removed.
details	object	
action	enum	**ALLOWED:** `sanitized, removed` The type of action that was performed
count	integer	The number of objects that were sanitized/removed.
object_details	array	
object_name	string	The object type that was sanitized/removed.
description	string	Action was successful or not.
file_name	string	If an embedded file was sanitized.
object_details	array	
object_name*	string	The object type that was sanitized/removed.
failure_category	string	Deep CDR errors are classified into different categories. For more details, please find [Troubleshooting sanitization failures](https://docs.opswat.com/mdcore/deep-cdr/troubleshooting-sanitization-failures)
sanitized_file_info	object	
file_size	integer	Size of sanitized file in bytes.
sha256	string	SHA256 hash of sanitized file.
scan_results	object	
data_id	string	Data ID of the requested file
progress_percentage	integer	Track analysis progress until reaches 100.

NAME	TYPE	DESCRIPTION
scan_all_result_a	enum	**ALLOWED:** `No Threat Detected, Infected, Suspicious, Failed, Whitelisted, Blacklisted, Exceeded Archive Depth, Not Scanned, Encrypted Archive, Exceeded Archive Size, Exceeded Archive File Number, Password Protected Document, Exceeded Archive Timeout, Mismatch, Potentially Vulnerable File, Cancelled, Sensitive Data Found, Yara Rule Matched, Potentially Unwanted, Unsupported File Type, Extraction Failed, Scan Failed, Suspicious Verdict by Sandbox, Likely Malicious Verdict by Sandbox, Malicious Verdict by Sandbox, Blocked Verdict by Sandbox, Blocked Verdict by Deep CDR, Global Timeout Exceeded, Vulnerable Verdict by SBOM, Non-vulnerable Verdict by SBOM, Blocked Verdict by SBOM, Blocked by Post Action, Known Bad, Known Good, Unknown, Allowed Verdict by COO, Blocked Verdict by COO, Unknown Verdict by COO, In Progress, Skip Processing Fast Symlink` The overall scan result as string
scan_all_result_i	enum	**ALLOWED:** `0, 1, 2, 3, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36, 38, 39, 40, 41, 42, 43, 255, 1014` The overall scan result as index in the Processing Results table.
start_time	string	Timestamp when the scanning process starts.
total_avs	integer	Total number of scanning engines used as part of this analysis.
total_time	integer	Total time elapsed during scan (in milliseconds).
scan_details	object	
ClamAV	object	
def_time	string	The database definition time for this engine
eng_id	string	The unique identification string for the engine
location	string	Where this engine is deployed (local/remote).
scan_result_i	integer	Scan result as index in the Processing Results table
scan_time	integer	The time elapsed during scan with this engine (in milliseconds).
threat_found	string	The threat name, IF scan result is Infected or Suspicious. Otherwise empty string or error message from the engine.
wait_time	integer	Time elapsed between sending file to Core and receiving the result from the engine (in milliseconds).
vulnerability_info	object	
result	object	
code	integer	The result code for vulnerability check, 0 means a successful check
hash	string	The file's SHA1 hash value
method	enum	**ALLOWED:** `50700` The method used by OESIS Framework, it should be 50700 every time.
timestamp	string	Timestamp of the request issued

NAME	TYPE	DESCRIPTION
timing	integer	The vulnerability check's duration in milliseconds
detected_product	object	
has_kb	boolean	Indicates whether any KBs or MSBs exist for this hash
has_vulnerability	boolean	Indicates whether any vulnerabilities have been associated with the particular product
is_current	boolean	True if this product's patch level is current, defaults to true
product	object	
id	integer	The OPSWAT product id
name	string	The product name
remediation_link	string	A link where product updates or patches can be obtained
severity	enum	**ALLOWED:** `LOW, MODERATE, IMPORTANT, CRITICAL, NOT_AVAILABLE, UNKNOWN` String description of Severity level: * `LOW` * `MODERATE` * `IMPORTANT` * `CRITICAL` * `NOT_AVAILABLE` * `UNKNOWN`
sig_name	string	Product signature descriptor
signature	integer	OPSWAT signature id
vendor	object	
id	integer	The OPSWAT vendor id
name	string	The vendor name
version	string	The installed product version
version_data	object	
count_behind	integer	The number of patches behind of the installed product
feed_id	integer	The remote feed ID used to determine patch level
version	string	The current version of the product in the remote feed
vulnerabilites	array	
description	string	A text description of the specific vulnerability
details	object	
cpe	string	A CPE product reference
cve	string	A CVE identification string
cvss	object	
access-complexity	string	A CVSS access-complexity descriptor
access-vector	string	A CVSS access-vector descriptor
authentication	string	A CVSS authentication descriptor

NAME	TYPE	DESCRIPTION
availability-impact	string	A CVSS availability impact descriptor
confidentiality-impact	string	A CVSS confidentiality impact descriptor
generated-on-epoch	string	An epoch timestamp indicating CVSS generation time
integrity-impact	string	A CVSS integrity impact descriptor
score	string	A CVSS 10-point severity score
source	string	A CVSS source descriptor
cwe	string	A CWE group identification string
last_modified_epoch	string	An epoch timestamp indicating source last update time
published-epoch	string	An epoch timestamp indicating source publishing time
references	array	
severity	enum	**ALLOWED:** `LOW, MODERATE, IMPORTANT, CRITICAL, NOT_AVAILABLE, UNKNOWN` String description of Severity level: * `LOW` * `MODERATE` * `IMPORTANT` * `CRITICAL` * `NOT_AVAILABLE` * `UNKNOWN`
severity_index	integer	A 5 point scale numerical description of Severity level with 5 being greatest and 0 being unknown
static_id	integer	An OPSWAT identifier for the vulnerability
verdict	integer	The vulnerability check's duration in milliseconds * `0` - No Vulnerability Found * `1` - Vulnerability Found * `3` - Failed * `16` - Processing Timed Out
yara	object	
hits	object	
verdict	enum	**ALLOWED:** `0, 1, 2, 3, 4` The overall result for the analyzed file. Value will be one of the following: | index | status | |---|---| | 0 | Clean | | 1 | Found matched data | | 2 | Suspicious | | 3 | Failed | | 4 | Not scanned |
file_info	object	
display_name	string	The filename reported via `filename` header.
file_size	integer	Total file size in bytes.

NAME	TYPE	DESCRIPTION
file_type	string	The filetype using mimetype.
file_type_description	string	The filetype in human readable format.
md5	string	File's MD5 hash.
sha1	string	File's SHA1 hash.
sha256	string	File's SHA256 Hash.
sha512	string	File's SHA512 Hash.
signer_infos	array	
digest_algorithm	string	Digest algorithm.
digest_encryption_algorithm	string	Encryption algorithm.
issuer	string	Entity that develops and registers the certificate.
serial_number	string	Serial number of the certificate.
vendor_name	string	Entity that is issued a certificate and utilize it for creating a digital signature.
version	string	Version of X.509 that is used in the certificate. This version field is zero-based. * 0: v1 * 1: v2 * 2: v3
type_category	array	
receive_data_timestamp	string	The timestamp when upload progress started (first byte received) (in milliseconds)
upload_time	integer	Total time elapsed for upload process (in milliseconds).
upload_timestamp	string	The timestamp when upload progress finished (all bytes received) (in milliseconds)
filetype_info	object	
file_info*	object	
description*	string	File type description
encrypted*	boolean	File is password-protected or not
extensions*	string	File type extension
groupID*	string	File type category
groupIDs*	array	
type*	string	MIME type
typeID*	string	File type ID
type_ids*	array	
file_info_details	object	
file_metadata	object	
result_template_hash	string	SHA256 Hash of user-interface template. For web console only.

NAME	TYPE	DESCRIPTION
process_info	object	
blocked_reason	string	Provides the reason why the file is blocked (if so).
file_type_skipped_scan	boolean	Indicates if the input file's detected type was configured to skip scanning.
outdated_data	array	
processing_time	integer	Total time elapsed during processing file (in milliseconds).
processing_time_details	object	
av_scan_time	integer	AV engines' processing time.
cdr_time	integer	Deep CDR engine's sanitization time.
dlp_time	integer	Proactive DLP engine's processing time.
extraction_time	integer	Archive extraction engine's processing time.
filetype_time	integer	FileType engine's processing time.
opswatfilescan_time	integer	OPSWAT Filescan engine's processing time.
others_time	integer	Total time elapsed for following processing tasks in the product (in milliseconds): * File processing's hash calculation time * External scanner (if configured) * Post action (if configured) * Other internal procssing time among components in the product
vul_time	integer	Vulnerability engine's lookup time.
wait_child_files_time	integer	The waiting time for completely processing all nested files of an archive file.
yara_time	integer	YARA engine's processing time.
profile	string	The used rule name.
progress_percentage	integer	Percentage of processing completed (from 1-100).
queue_time	integer	Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue_time = start_time - upload_timestamp) (in milliseconds).
result	string	The final result of processing the file (Allowed / Blocked / Processing).
user_agent	string	Identifier for the REST Client that calls the API.
username	string	User identifier who submitted scan request earlier.
verdicts	array	
post_processing	object	
actions_failed	string	Empty string if no action failed or list of failed actions, separated by "|".
actions_ran	string	List of successful actions, separated by "|". Empty string if otherwise.
converted_destination	string	Contains the name of the sanitized file.
converted_to	string	Contains target type name of sanitization.
copy_move_destination	string	Contains target type name of sanitization.

NAME	TYPE	DESCRIPTION
sanitization_details	object	
description	string	Action was successful or not.
details	array	
action*	enum	**ALLOWED:** sanitized, removed The type of action that was performed
count	integer	The number of objects that were sanitized/removed.
details	object	
action	enum	**ALLOWED:** sanitized, removed The type of action that was performed
count	integer	The number of objects that were sanitized/removed.
object_details	array	
object_name	string	The object type that was sanitized/removed.
description	string	Action was successful or not.
file_name	string	If an embedded file was sanitized.
object_details	array	
object_name*	string	The object type that was sanitized/removed.
failure_category	string	Deep CDR errors are classified into different categories. For more details, please find [Troubleshooting sanitization failures](https://docs.opswat.com/mdcore/deep-cdr/troubleshooting-sanitization-failures)
sanitized_file_info	object	
file_size	integer	Size of sanitized file in bytes.
sha256	string	SHA256 hash of sanitized file.
scan_results	object	
data_id	string	Data ID of the requested file
progress_percentage	integer	Track analysis progress until reaches 100.

NAME	TYPE	DESCRIPTION
scan_all_result_a	enum	**ALLOWED:** `No Threat Detected, Infected, Suspicious, Failed, Whitelisted, Blacklisted, Exceeded Archive Depth, Not Scanned, Encrypted Archive, Exceeded Archive Size, Exceeded Archive File Number, Password Protected Document, Exceeded Archive Timeout, Mismatch, Potentially Vulnerable File, Cancelled, Sensitive Data Found, Yara Rule Matched, Potentially Unwanted, Unsupported File Type, Extraction Failed, Scan Failed, Suspicious Verdict by Sandbox, Likely Malicious Verdict by Sandbox, Malicious Verdict by Sandbox, Blocked Verdict by Sandbox, Blocked Verdict by Deep CDR, Global Timeout Exceeded, Vulnerable Verdict by SBOM, Non-vulnerable Verdict by SBOM, Blocked Verdict by SBOM, Blocked by Post Action, Known Bad, Known Good, Unknown, Allowed Verdict by COO, Blocked Verdict by COO, Unknown Verdict by COO, In Progress, Skip Processing Fast Symlink` The overall scan result as string
scan_all_result_i	enum	**ALLOWED:** `0, 1, 2, 3, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36, 38, 39, 40, 41, 42, 43, 255, 1014` The overall scan result as index in the Processing Results table.
start_time	string	Timestamp when the scanning process starts.
total_avs	integer	Total number of scanning engines used as part of this analysis.
total_time	integer	Total time elapsed during scan (in milliseconds).
scan_details	object	
ClamAV	object	
def_time	string	The database definition time for this engine
eng_id	string	The unique identification string for the engine
location	string	Where this engine is deployed (local/remote).
scan_result_i	integer	Scan result as index in the Processing Results table
scan_time	integer	The time elapsed during scan with this engine (in milliseconds).
threat_found	string	The threat name, IF scan result is Infected or Suspicious. Otherwise empty string or error message from the engine.
wait_time	integer	Time elapsed between sending file to Core and receiving the result from the engine (in milliseconds).
vulnerability_info	object	
result	object	
code	integer	The result code for vulnerability check, 0 means a successful check
hash	string	The file's SHA1 hash value
method	enum	**ALLOWED:** `50700` The method used by OESIS Framework, it should be 50700 every time.
timestamp	string	Timestamp of the request issued

NAME	TYPE	DESCRIPTION
timing	integer	The vulnerability check's duration in milliseconds
detected_product	object	
has_kb	boolean	Indicates whether any KBs or MSBs exist for this hash
has_vulnerability	boolean	Indicates whether any vulnerabilities have been associated with the particular product
is_current	boolean	True if this product's patch level is current, defaults to true
product	object	
id	integer	The OPSWAT product id
name	string	The product name
remediation_link	string	A link where product updates or patches can be obtained
severity	enum	**ALLOWED:** `LOW, MODERATE, IMPORTANT, CRITICAL, NOT_AVAILABLE, UNKNOWN` String description of Severity level: * `LOW` * `MODERATE` * `IMPORTANT` * `CRITICAL` * `NOT_AVAILABLE` * `UNKNOWN`
sig_name	string	Product signature descriptor
signature	integer	OPSWAT signature id
vendor	object	
id	integer	The OPSWAT vendor id
name	string	The vendor name
version	string	The installed product version
version_data	object	
count_behind	integer	The number of patches behind of the installed product
feed_id	integer	The remote feed ID used to determine patch level
version	string	The current version of the product in the remote feed
vulnerabilites	array	
description	string	A text description of the specific vulnerability
details	object	
cpe	string	A CPE product reference
cve	string	A CVE identification string
cvss	object	
access-complexity	string	A CVSS access-complexity descriptor
access-vector	string	A CVSS access-vector descriptor
authentication	string	A CVSS authentication descriptor

NAME	TYPE	DESCRIPTION
availability-impact	string	A CVSS availability impact descriptor
confidentiality-impact	string	A CVSS confidentiality impact descriptor
generated-on-epoch	string	An epoch timestamp indicating CVSS generation time
integrity-impact	string	A CVSS integrity impact descriptor
score	string	A CVSS 10-point severity score
source	string	A CVSS source descriptor
cwe	string	A CWE group identification string
last_modified_epoch	string	An epoch timestamp indicating source last update time
published-epoch	string	An epoch timestamp indicating source publishing time
references	array	
severity	enum	**ALLOWED:** `LOW, MODERATE, IMPORTANT, CRITICAL, NOT_AVAILABLE, UNKNOWN` String description of Severity level: * `LOW` * `MODERATE` * `IMPORTANT` * `CRITICAL` * `NOT_AVAILABLE` * `UNKNOWN`
severity_index	integer	A 5 point scale numerical description of Severity level with 5 being greatest and 0 being unknown
static_id	integer	An OPSWAT identifier for the vulnerability
verdict	integer	The vulnerability check's duration in milliseconds * `0` - No Vulnerability Found * `1` - Vulnerability Found * `3` - Failed * `16` - Processing Timed Out
yara	object	
hits	object	
verdict	enum	**ALLOWED:** `0, 1, 2, 3, 4` The overall result for the analyzed file. Value will be one of the following: | index | status | |---|---| | 0 | Clean | | 1 | Found matched data | | 2 | Suspicious | | 3 | Failed | | 4 | Not scanned |
file_info	object	
display_name	string	The filename reported via `filename` header.
file_size	integer	Total file size in bytes.
file_type	string	The filetype using mimetype.

NAME	TYPE	DESCRIPTION
file_type_description	string	The filetype in human readable format.
md5	string	File's MD5 hash.
sha1	string	File's SHA1 hash.
sha256	string	File's SHA256 Hash.
sha512	string	File's SHA512 Hash.
signer_infos	array	
digest_algorithm	string	Digest algorithm.
digest_encryption_algorithm	string	Encryption algorithm.
issuer	string	Entity that develops and registers the certificate.
serial_number	string	Serial number of the certificate.
vendor_name	string	Entity that is issued a certificate and utilize it for creating a digital signature.
version	string	Version of X.509 that is used in the certificate. This version field is zero-based. * 0: v1 * 1: v2 * 2: v3
type_category	array	
receive_data_timestamp	string	The timestamp when upload progress started (first byte received) (in milliseconds)
upload_time	integer	Total time elapsed for upload process (in milliseconds).
upload_timestamp	string	The timestamp when upload progress finished (all bytes received) (in milliseconds)
filetype_info	object	
file_info*	object	
description*	string	File type description
encrypted*	boolean	File is password-protected or not
extensions*	string	File type extension
groupID*	string	File type category
groupIDs*	array	
type*	string	MIME type
typeID*	string	File type ID
type_ids*	array	
file_info_details	object	
file_metadata	object	
result_template_hash	string	SHA256 Hash of user-interface template. For web console only.
opswatfilescan_info	object	

NAME	TYPE	DESCRIPTION
process_info	object	
blocked_reason	string	Provides the reason why the file is blocked (if so).
file_type_skipped_scan	boolean	Indicates if the input file's detected type was configured to skip scanning.
outdated_data	array	
processing_time	integer	Total time elapsed during processing file (in milliseconds).
processing_time_details	object	
av_scan_time	integer	AV engines' processing time.
cdr_time	integer	Deep CDR engine's sanitization time.
dlp_time	integer	Proactive DLP engine's processing time.
extraction_time	integer	Archive extraction engine's processing time.
filetype_time	integer	FileType engine's processing time.
opswatfilescan_time	integer	OPSWAT Filescan engine's processing time.
others_time	integer	Total time elapsed for following processing tasks in the product (in milliseconds): * File processing's hash calculation time * External scanner (if configured) * Post action (if configured) * Other internal procssing time among components in the product
parse_dgsg_time	integer	Digital signature analyzing time.
vul_time	integer	Vulnerability engine's lookup time.
wait_child_files_time	integer	The waiting time for completely processing all nested files of an archive file.
yara_time	integer	YARA engine's processing time.
profile	string	The used rule name.
progress_percentage	integer	Percentage of processing completed (from 1-100).
queue_time	integer	Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue_time = start_time - upload_timestamp) (in milliseconds).
result	string	The final result of processing the file (Allowed / Blocked / Processing).
user_agent	string	Identifier for the REST Client that calls the API.
username	string	User identifier who submitted scan request earlier.
verdicts	array	
post_processing	object	
actions_failed	string	Empty string if no action failed or list of failed actions, separated by "|".
actions_ran	string	List of successful actions, separated by "|". Empty string if otherwise.
converted_destination	string	Contains the name of the sanitized file.
converted_to	string	Contains target type name of sanitization.

NAME	TYPE	DESCRIPTION
copy_move_destination	string	Contains target type name of sanitization.
sanitization_details	object	
description	string	Action was successful or not.
details	array	
action*	enum	**ALLOWED:** sanitized, removed The type of action that was performed
count	integer	The number of objects that were sanitized/removed.
details	object	
action	enum	**ALLOWED:** sanitized, removed The type of action that was performed
count	integer	The number of objects that were sanitized/removed.
object_details	array	
object_name	string	The object type that was sanitized/removed.
description	string	Action was successful or not.
file_name	string	If an embedded file was sanitized.
object_details	array	
object_name*	string	The object type that was sanitized/removed.
failure_category	string	Deep CDR errors are classified into different categories. For more details, please find [Troubleshooting sanitization failures](https://docs.opswat.com/mdcore/deep-cdr/troubleshooting-sanitization-failures)
sanitized_file_info	object	
file_size	integer	Size of sanitized file in bytes.
sha256	string	SHA256 hash of sanitized file.
scan_results	object	
data_id	string	Data ID of the requested file
progress_percentage	integer	Track analysis progress until reaches 100.

NAME	TYPE	DESCRIPTION
scan_all_result_a	enum	**ALLOWED:** `No Threat Detected, Infected, Suspicious, Failed, Whitelisted, Blacklisted, Exceeded Archive Depth, Not Scanned, Encrypted Archive, Exceeded Archive Size, Exceeded Archive File Number, Password Protected Document, Exceeded Archive Timeout, Mismatch, Potentially Vulnerable File, Cancelled, Sensitive Data Found, Yara Rule Matched, Potentially Unwanted, Unsupported File Type, Extraction Failed, Scan Failed, Suspicious Verdict by Sandbox, Likely Malicious Verdict by Sandbox, Malicious Verdict by Sandbox, Blocked Verdict by Sandbox, Blocked Verdict by Deep CDR, Global Timeout Exceeded, Vulnerable Verdict by SBOM, Non-vulnerable Verdict by SBOM, Blocked Verdict by SBOM, Blocked by Post Action, Known Bad, Known Good, Unknown, Allowed Verdict by COO, Blocked Verdict by COO, Unknown Verdict by COO, In Progress, Skip Processing Fast Symlink` The overall scan result as string
scan_all_result_i	enum	**ALLOWED:** `0, 1, 2, 3, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36, 38, 39, 40, 41, 42, 43, 255, 1014` The overall scan result as index in the Processing Results table.
start_time	string	Timestamp when the scanning process starts.
total_avs	integer	Total number of scanning engines used as part of this analysis.
total_time	integer	Total time elapsed during scan (in milliseconds).
scan_details	object	
ClamAV	object	
def_time	string	The database definition time for this engine
eng_id	string	The unique identification string for the engine
location	string	Where this engine is deployed (local/remote).
scan_result_i	integer	Scan result as index in the Processing Results table
scan_time	integer	The time elapsed during scan with this engine (in milliseconds).
threat_found	string	The threat name, IF scan result is Infected or Suspicious. Otherwise empty string or error message from the engine.
wait_time	integer	Time elapsed between sending file to Core and receiving the result from the engine (in milliseconds).
vulnerability_info	object	
result	object	
code	integer	The result code for vulnerability check, 0 means a successful check
hash	string	The file's SHA1 hash value
method	enum	**ALLOWED:** `50700` The method used by OESIS Framework, it should be 50700 every time.
timestamp	string	Timestamp of the request issued

NAME	TYPE	DESCRIPTION
timing	integer	The vulnerability check's duration in milliseconds
detected_product	object	
has_kb	boolean	Indicates whether any KBs or MSBs exist for this hash
has_vulnerability	boolean	Indicates whether any vulnerabilities have been associated with the particular product
is_current	boolean	True if this product's patch level is current, defaults to true
product	object	
id	integer	The OPSWAT product id
name	string	The product name
remediation_link	string	A link where product updates or patches can be obtained
severity	enum	**ALLOWED:** `LOW, MODERATE, IMPORTANT, CRITICAL, NOT_AVAILABLE, UNKNOWN` String description of Severity level: * `LOW` * `MODERATE` * `IMPORTANT` * `CRITICAL` * `NOT_AVAILABLE` * `UNKNOWN`
sig_name	string	Product signature descriptor
signature	integer	OPSWAT signature id
vendor	object	
id	integer	The OPSWAT vendor id
name	string	The vendor name
version	string	The installed product version
version_data	object	
count_behind	integer	The number of patches behind of the installed product
feed_id	integer	The remote feed ID used to determine patch level
version	string	The current version of the product in the remote feed
vulnerabilites	array	
description	string	A text description of the specific vulnerability
details	object	
cpe	string	A CPE product reference
cve	string	A CVE identification string
cvss	object	
access-complexity	string	A CVSS access-complexity descriptor
access-vector	string	A CVSS access-vector descriptor
authentication	string	A CVSS authentication descriptor

NAME	TYPE	DESCRIPTION
availability-impact	string	A CVSS availability impact descriptor
confidentiality-impact	string	A CVSS confidentiality impact descriptor
generated-on-epoch	string	An epoch timestamp indicating CVSS generation time
integrity-impact	string	A CVSS integrity impact descriptor
score	string	A CVSS 10-point severity score
source	string	A CVSS source descriptor
cwe	string	A CWE group identification string
last_modified_epoch	string	An epoch timestamp indicating source last update time
published-epoch	string	An epoch timestamp indicating source publishing time
references	array	
severity	enum	**ALLOWED:** `LOW, MODERATE, IMPORTANT, CRITICAL, NOT_AVAILABLE, UNKNOWN` String description of Severity level: * `LOW` * `MODERATE` * `IMPORTANT` * `CRITICAL` * `NOT_AVAILABLE` * `UNKNOWN`
severity_index	integer	A 5 point scale numerical description of Severity level with 5 being greatest and 0 being unknown
static_id	integer	An OPSWAT identifier for the vulnerability
verdict	integer	The vulnerability check's duration in milliseconds * `0` - No Vulnerability Found * `1` - Vulnerability Found * `3` - Failed * `16` - Processing Timed Out
yara	object	
hits	object	
verdict	enum	**ALLOWED:** `0, 1, 2, 3, 4` The overall result for the analyzed file. Value will be one of the following: | index | status | |---|---| | 0 | Clean | | 1 | Found matched data | | 2 | Suspicious | | 3 | Failed | | 4 | Not scanned |

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		

NAME	TYPE	DESCRIPTION
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

3.6 GET /hash/{md5|sha1|sha256|sha512}

Fetch Analysis Result By Hash

Retrieve analysis result by hash

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*md5|sha1|sha256|sha512	string		Hash value to search. This can be md5, sha1, sha256, sha512

QUERY PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
first	integer		The first item order in the list child files of archive file
size	integer		The number of items to be fetched next, counting from the item order indicated in first header

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.
rule	string		Select rule for the analysis, if no header given the default rule will be selected (URL encoded UTF-8 string of rule name)

NAME	TYPE	EXAMPLE	DESCRIPTION
selfonly	boolean		Useful to archive hash lookup. Allow specifying to only perform hash lookup against the original archive file self only, and skip searching all child files result within the original archive. Default value is false.
timerange	integer		Scoping down the recent number of hours that hash lookup task should start from till now, instead of searching the entire scan history in MetaDefender Core database. Default value is 0. That means no time scope.
include-inprogress	boolean		False (default): API will return "Not Found" if the verdict is in progress. True: If the queried hash has a completed processing result before, API will return the completed processing result. If this hash doesn't have any completed processing result, API will return this In-progress result.

RESPONSE

STATUS CODE - 200: Get information of file

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
data_id	string	data identifier of the requested file
dlp_info	object	
certainty	enum	**ALLOWED:** Very Low, Low, Medium, High, Very High Describes how certain the hit is, possible values: * `Very Low` * `Low` * `Medium` * `High` * `Very High`
errors	object	
filename	string	Output processed file name (pre-configured on engine settings under Core's worflow rule)
hits	object	
ccn	object	

NAME	TYPE	DESCRIPTION
display_name	string	Credit Card Number, Social Security Number, or in case of RegEx, the name of the rule that has been given by the user
hits	array	
after	string	The context after the matched data.
before	string	The context before the matched data.
certainty	enum	**ALLOWED:** `Very Low, Low, Medium, High, Very High` The text version of "certainty_score", possible values: * `Very Low` * `Low` * `Medium` * `High` * `Very High`
certainty_score	integer	Is defined by the relevance of the given hit in its context. It is calculated based on multiple factors such as the number of digits, possible values: [0-100]
hit	string	The matched data.
location	string	The location of the hit that is found in a file.
severity	enum	**ALLOWED:** `0, 1` (NOTE: this field is deprecated): can be 0 (detected) or 1 (suspicious).
tryRedact	boolean	If file was redacted or not.
metadata_removal	object	
result	enum	**ALLOWED:** `removed, not removed, failed to remove` Result of the metadata removal process, possible values: * `removed` * `not removed` * `failed to remove`
redact	object	
result	enum	**ALLOWED:** `redacted, not redacted, failed to redact` Result of the redaction process, possible values: * `redacted` * `not redacted` * `failed to redact`
severity	enum	**ALLOWED:** `0, 1` (NOTE: this field is deprecated): represents the severity of the data loss, possible values: * `0` - Certainly is data loss * `1` - Might be data loss
verdict	enum	**ALLOWED:** `0, 1, 2, 3, 4` The overall result for the scanned file. Possible values: * `0` - Clean * `1` - Found matched data * `2` - Suspicious * `3` - Failed * `4` - Not scanned
watermark	object	

NAME	TYPE	DESCRIPTION
result	enum	**ALLOWED:** `added, not added, failed to add` Result of the watermarking process, possible values: * `added` * `not added` * `failed to add`
download_info	object	
error_detail	string	Revealed detailed reason why the download failed.
progress	number	Only applicable when "status" is `Downloading`, indicates download finished percentage, in a range of [1, 99]. * Once hitting 100, the status will be changed to `Download Success`. * or other problematic status (`Download Cancelled`, `Download Failed`) if the download stopped unexpectedly.

status	string	Indicates download status, which could be either
		- `Downloading`
		- Check `progress` key value for actual download percentage
		``json
		"download_info": {
		"progress": 7,
		"status": "Downloading",
		"url": "http://192.168.200.97:8080/5gb.zip"
		}
		``
		- `Download Success`
		``json
		"download_info": {
		"status": "Download Success",
		"url": "https://secure.eicar.org/eicar.com"
		}
		``
		- `Download Failed`
		- Check `error_detail` key value for an error explanation
		``json
		"download_info": {
		"error_detail": "Connection error",
		"status": "Download Failed",
		"url": "http://192.168.200.97:8080/2gb.zip"
		}
		``
		- `Download Timeout`
		- Expecting to occur when the download progress takes longer than what time window allowed in MetaDefender Core's pre-configured setting under workflow rule (under "SCAN" tab)
		``json
		"download_info": {
		"status": "Download Timeout",
		"url": "http://192.168.200.97:8080/2gb.zip"
		}
		``
		- `Download Cancelled`
		- Expecting to occur when user explicitly cancelled that file scan request, or batch request that the scan belongs to
		``json
		"download_info": {
		"status": "Download Cancelled",
		"url": "http://192.168.200.97:8080/5gb.zip"
		}
		``

NAME	TYPE	DESCRIPTION
url	string	Original download link which was specified in HTTP(S) request's `downloadfrom` header
extraction_info	object	
decrypted_status	enum	**ALLOWED:** `Success, Failed` Indicate that decryption phase is successful or not.
err_category	string	Error category
err_code	integer	Error code
err_details	string	Error message
is_encrypted_file	boolean	Indicate if file is password-protected or not.
file_info	object	
display_name	string	The filename reported via `filename` header.
file_size	integer	Total file size in bytes.
file_type	string	The filetype using mimetype.
file_type_description	string	The filetype in human readable format.
md5	string	File's MD5 hash.
sha1	string	File's SHA1 hash.
sha256	string	File's SHA256 Hash.
sha512	string	File's SHA512 Hash.
signer_infos	array	
digest_algorithm	string	Digest algorithm.
digest_encryption_algorithm	string	Encryption algorithm.
issuer	string	Entity that develops and registers the certificate.
serial_number	string	Serial number of the certificate.
vendor_name	string	Entity that is issued a certificate and utilize it for creating a digital signature.
version	string	Version of X.509 that is used in the certificate. This version field is zero-based. * 0: v1 * 1: v2 * 2: v3
type_category	array	
receive_data_timestamp	string	The timestamp when upload progress started (first byte received) (in milliseconds)
upload_time	integer	Total time elapsed for upload process (in milliseconds).
upload_timestamp	string	The timestamp when upload progress finished (all bytes received) (in milliseconds)
filetype_info	object	
file_info*	object	

NAME	TYPE	DESCRIPTION
description*	string	File type description
encrypted*	boolean	File is password-protected or not
extensions*	string	File type extension
groupID*	string	File type category
groupIDs*	array	
type*	string	MIME type
typeID*	string	File type ID
type_ids*	array	
file_info_details	object	
file_metadata	object	
result_template_hash	string	SHA256 Hash of user-interface template. For web console only.
opswatfilescan_info	object	
process_info	object	
blocked_reason	string	Provides the reason why the file is blocked (if so).
blocked_reasons	array	
file_type_skipped_scan	boolean	Indicates if the input file's detected type was configured to skip scanning.
hash_time	integer	Total time elapsed for computing hashes (in milliseconds).
outdated_data	array	
processing_time	integer	Total time elapsed during processing file (in milliseconds).
processing_time_details	object	
av_scan_time	integer	AV engines' processing time.
cdr_time	integer	Deep CDR engine's sanitization time.
dlp_time	integer	Proactive DLP engine's processing time.
extraction_time	integer	Archive extraction engine's processing time.
filetype_time	integer	FileType engine's processing time.
opswatfilescan_time	integer	OPSWAT Filescan engine's processing time.
others_time	integer	Total time elapsed for following processing tasks in the product (in milliseconds): * Decryption time (if receiving an encrypted file) * External scanner (if configured) * Post action (if configured) * Other internal processing time among components in the product
parse_dgsg_time	integer	Digital signature analyzing time.
vul_time	integer	Vulnerability engine's lookup time.
yara_time	integer	YARA engine's processing time.
profile	string	The used rule name.

NAME	TYPE	DESCRIPTION
progress_percentage	integer	Percentage of processing completed (from 1-100).
queue_time	integer	Total time elapsed for file processing task was waiting in MetaDefender Core's queue until being picked up (queue_time = start_time - upload_timestamp) (in milliseconds).
result	string	The final result of processing the file (Allowed / Blocked / Processing).
user_agent	string	Identifier for the REST Client that calls the API.
username	string	User identifier who submitted scan request earlier.
verdicts	array	
post_processing	object	
actions_failed	string	Empty string if no action failed or list of failed actions, separated by "|".
actions_ran	string	List of successful actions, separated by "|". Empty string if otherwise.
converted_destination	string	Contains the name of the sanitized file.
converted_to	string	Contains target type name of sanitization.
copy_move_destination	string	Contains target type name of sanitization.
sanitization_details	object	
description	string	Action was successful or not.
details	array	
action*	enum	**ALLOWED:** `sanitized, removed` The type of action that was performed
count	integer	The number of objects that were sanitized/removed.
details	object	
action	enum	**ALLOWED:** `sanitized, removed` The type of action that was performed
count	integer	The number of objects that were sanitized/removed.
object_details	array	
object_name	string	The object type that was sanitized/removed.
description	string	Action was successful or not.
file_name	string	If an embedded file was sanitized.
object_details	array	
object_name*	string	The object type that was sanitized/removed.
failure_category	string	Deep CDR errors are classified into different categories. For more details, please find [Troubleshooting sanitization failures](https://docs.opswat.com/mdcore/deep-cdr/troubleshooting-sanitization-failures)
sanitized_file_info	object	
file_size	integer	Size of sanitized file in bytes.
sha256	string	SHA256 hash of sanitized file.

NAME	TYPE	DESCRIPTION
scan_results	object	
data_id	string	Data ID of the requested file
progress_percentage	integer	Track analysis progress until reaches 100.
scan_all_result_a	enum	**ALLOWED:** No Threat Detected, Infected, Suspicious, Failed, Whitelisted, Blacklisted, Exceeded Archive Depth, Not Scanned, Encrypted Archive, Exceeded Archive Size, Exceeded Archive File Number, Password Protected Document, Exceeded Archive Timeout, Mismatch, Potentially Vulnerable File, Cancelled, Sensitive Data Found, Yara Rule Matched, Potentially Unwanted, Unsupported File Type, Extraction Failed, Scan Failed, Suspicious Verdict by Sandbox, Likely Malicious Verdict by Sandbox, Malicious Verdict by Sandbox, Blocked Verdict by Sandbox, Blocked Verdict by Deep CDR, Global Timeout Exceeded, Vulnerable Verdict by SBOM, Non-vulnerable Verdict by SBOM, Blocked Verdict by SBOM, Blocked by Post Action, Known Bad, Known Good, Unknown, Allowed Verdict by COO, Blocked Verdict by COO, Unknown Verdict by COO, In Progress, Skip Processing Fast Symlink The overall scan result as string
scan_all_result_i	enum	**ALLOWED:** 0, 1, 2, 3, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36, 38, 39, 40, 41, 42, 43, 255, 1014 The overall scan result as index in the Processing Results table.
start_time	string	Timestamp when the scanning process starts.
total_avs	integer	Total number of scanning engines used as part of this analysis.
total_time	integer	Total time elapsed during scan (in milliseconds).
scan_details	object	
ClamAV	object	
def_time	string	The database definition time for this engine
eng_id	string	The unique identification string for the engine
location	string	Where this engine is deployed (local/remote).
scan_result_i	integer	Scan result as index in the Processing Results table
scan_time	integer	The time elapsed during scan with this engine (in milliseconds).
threat_found	string	The threat name, IF scan result is Infected or Suspicious. Otherwise empty string or error message from the engine.
wait_time	integer	Time elapsed between sending file to Core and receiving the result from the engine (in milliseconds).
vulnerability_info	object	
result	object	
code	integer	The result code for vulnerability check, 0 means a successful check
hash	string	The file's SHA1 hash value

NAME	TYPE	DESCRIPTION
method	enum	**ALLOWED:** `50700` The method used by OESIS Framework, it should be 50700 every time.
timestamp	string	Timestamp of the request issued
timing	integer	The vulnerability check's duration in milliseconds
detected_product	object	
has_kb	boolean	Indicates whether any KBs or MSBs exist for this hash
has_vulnerability	boolean	Indicates whether any vulnerabilities have been associated with the particular product
is_current	boolean	True if this product's patch level is current, defaults to true
product	object	
id	integer	The OPSWAT product id
name	string	The product name
remediation_link	string	A link where product updates or patches can be obtained
severity	enum	**ALLOWED:** `LOW, MODERATE, IMPORTANT, CRITICAL, NOT_AVAILABLE, UNKNOWN` String description of Severity level: * `LOW` * `MODERATE` * `IMPORTANT` * `CRITICAL` * `NOT_AVAILABLE` * `UNKNOWN`
sig_name	string	Product signature descriptor
signature	integer	OPSWAT signature id
vendor	object	
id	integer	The OPSWAT vendor id
name	string	The vendor name
version	string	The installed product version
version_data	object	
count_behind	integer	The number of patches behind of the installed product
feed_id	integer	The remote feed ID used to determine patch level
version	string	The current version of the product in the remote feed
vulnerabilites	array	
description	string	A text description of the specific vulnerability
details	object	
cpe	string	A CPE product reference
cve	string	A CVE identification string
cvss	object	

NAME	TYPE	DESCRIPTION
access-complexity	string	A CVSS access-complexity descriptor
access-vector	string	A CVSS access-vector descriptor
authentication	string	A CVSS authentication descriptor
availability-impact	string	A CVSS availability impact descriptor
confidentiality-impact	string	A CVSS confidentiality impact descriptor
generated-on-epoch	string	An epoch timestamp indicating CVSS generation time
integrity-impact	string	A CVSS integrity impact descriptor
score	string	A CVSS 10-point severity score
source	string	A CVSS source descriptor
cwe	string	A CWE group identification string
last_modified_epoch	string	An epoch timestamp indicating source last update time
published-epoch	string	An epoch timestamp indicating source publishing time
references	array	
severity	enum	**ALLOWED:** `LOW, MODERATE, IMPORTANT, CRITICAL, NOT_AVAILABLE, UNKNOWN` String description of Severity level: * `LOW` * `MODERATE` * `IMPORTANT` * `CRITICAL` * `NOT_AVAILABLE` * `UNKNOWN`
severity_index	integer	A 5 point scale numerical description of Severity level with 5 being greatest and 0 being unknown
static_id	integer	An OPSWAT identifier for the vulnerability
verdict	integer	The vulnerability check's duration in milliseconds * `0` - No Vulnerability Found * `1` - Vulnerability Found * `3` - Failed * `16` - Processing Timed Out
yara	object	
hits	object	
verdict	enum	**ALLOWED:** `0, 1, 2, 3, 4` The overall result for the analyzed file. Value will be one of the following: | index | status | |---|---| | 0 | Clean | | 1 | Found matched data | | 2 | Suspicious | | 3 | Failed | | 4 | Not scanned |

STATUS CODE - 404: Invalid hash format

3.7 GET /file/webhook/{data_id}

Query webhook status

Prior to being notified by Core when webhooks mode is set, client can anytime ask Core for file / batch processing webhooks status.

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*data_id	string		Either `data_id` or `batch_id`.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Webhooks status is fetched successfully.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
data_id	string	The file submission identifier
request_time	string	A timestamp when the request has been made.

NAME	TYPE	DESCRIPTION
status_code	integer	What was the returned HTTP status code. * `200` - Callback was sent successfully * `403` - ContentAccessDenied. The access to the remote content was denied (similar to HTTP(S) error 401). * `404` - ContentNotFoundError. The remote content was not found at the server (similar to HTTP(S) error 404). * `408` - TimeoutError. The connection to the remote server timed out. * `503` - HostNotFoundError. The remote host name was not found (invalid hostname). * `520` - RemoteHostClosedError. The remote server closed the connection prematurely, before the entire reply was received and processed. * `444` - Other error types. Since MetaDefender Core 4.21.2, warning-level messages will be logged in case of Webhook failures. Check the following for detailed error descriptions. <details> 1 - the remote server refused the connection (the server is not accepting requests). 2 - the remote server closed the connection prematurely, before the entire reply was received and processed. 3 - the remote host name was not found (invalid hostname). 4 - the connection to the remote server timed out. 5 - operation canceled before it was finished. 6 - the SSL/TLS handshake failed and the encrypted channel could not be established. 7 - temporary network failure. 8 - the connection was broken due to disconnection from the network or failure to start the network. 9 - the background request is not currently allowed due to platform policy. 10 - while following redirects, the maximum limit was reached. 11 - while following redirects, a redirect from a encrypted protocol (https) to an unencrypted one (http) is detected. 99 - an unknown network-related error was detected. 101 - the connection to the proxy server was refused (the proxy server is not accepting requests). 102 - the proxy server closed the connection prematurely, before the entire reply was received and processed. 103 - the proxy host name was not found (invalid proxy hostname). 104 - the connection to the proxy timed out or the proxy did not reply in time to the request sent. 105 - the proxy requires authentication but did not accept any offered credentials (if any). 199 - an unknown proxy-related error was detected. 201 - the access to the remote content was denied. 202 - the operation requested on the remote content is not permitted. 203 - the remote content was not found at the server. 204 - the remote server requires authentication to serve the content but the credentials provided were not accepted (if any). 205 - the request needed to be sent again, but this failed for example because the upload data could not be read a second time. 206 - the request could not be completed due to a conflict with the current state of the resource. 207 - the requested resource is no longer available at the server. 299 - an unknown error related to the remote content was detected. 301 - the protocol is unknown. 302 - the requested operation is invalid for this protocol. 399 - a breakdown in protocol was detected (parsing error, invalid or unexpected responses, etc.). 401 - the server encountered an unexpected condition which prevented it from fulfilling the request. 402 - the server does not support the functionality required to fulfill the request. 403 - the server is unable to handle the request at this time. 499 - an unknown error related to the server response was detected. </details>
url	string	What was the called URL (should match the `callbackurl` header).

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 404: Requests resource was not found.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Not found"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
```

```
}
```

3.8 GET /file/rules

Fetching Available Analysis Rules

Retrieve all available rules with their custom configurations. Fetching available processing rules.

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication. Only those rules are returned, that: * Match the apikey's role sent using the apikey header, or * Are not restricted to a specific role.
user_agent	string		The user agent string value sent in the header (specified by the client). Only those rules are returned, that: * Match the client's user agent sent using the user_agent header, or * Are not restricted to a specific user agent. For details see KB article [What are Security Policies and how do I use them?](https://onlinehelp.opswat.com/corev4/What_are_Security_Policies_and_how_do_I_use_them_.html).

RESPONSE

STATUS CODE - 200: Returns the list of available rules.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
ARRAY OF OBJECT WITH BELOW STRUCTURE		
max_file_size	integer	The maximum allowed file size (in bytes) for this rule.
name	string	A unique identifier for identify in the used rule for a scan..

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

3.9 GET /file/converted/{data_id}

Download Sanitized Files

Retrieve sanitized file based on the data_id

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*data_id	string	8101abae27be4d63859c55d9e0ed0135	The data_id comes from the result of `Analyze a file`. In case of sanitizing the content of an archive, the data_id of contained file can be found in `Fetch analysis result`.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Returns the sanitized content.

RESPONSE MODEL - application/octet-stream

STATUS CODE - 404: Requests resource was not found.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Not found"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

3.10 GET /file/download/{data_id}

Download either sanitized files or DLP processed files

Retrieve sanitized file based on the data_id. In case there's no sanitized file, and DLP processed file is available, user will retrieve DLP processed file.

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*data_id	string	8101abae27be4d63859c55d9e0ed0135	The data_id comes from the result of `Analyze a file`. In case of sanitizing the content of an archive, the data_id of contained file can be found in `Fetch analysis result`.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Returns the sanitized or DLP processed content.

RESPONSE MODEL - application/octet-stream

STATUS CODE - 404: File could not be found

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "File could not be found"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
```

```
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

3.11 GET /file/{data_id}/export

Export processing result as a PDF file

Retrieve a PDF processing result file. The content of this PDF file can be changed over MD Core version.

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*data_id	string	8101abae27be4d63859c55d9e0ed0135	The data_id comes from the result of `Analyze a file`. In case of sanitizing the content of an archive, the data_id of contained file can be found in `Fetch analysis result`.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Returns the exported result PDF file

RESPONSE MODEL - application/pdf

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 404: Requests resource was not found.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Not found"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

3.12 GET /quarantine/{sha256}/download

Download Quarantined File

Retrieve a quarantined file based on the sha256 of the original file

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*sha256	string	4c4c10348935210f1b5f1a0b6cbb46b56f941570c5aac8ce934c9b594324268c	The SHA256 of the original file. This information can be retrieved from the result of `Analyze a file`.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Returns the quarantined file.

RESPONSE MODEL - application/octet-stream

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
```

```
}
```

STATUS CODE - 404: Requests resource was not found.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Not found"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

3.13 GET /file/{dataid}/sbom/report

Download SBOM Report

Retrieve an SBOM report in JSON format or a specific JSON format, such as CycloneDX or SPDX.

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*dataid	string	8101abae27be4d63859c55d9e0ed0135	The data_id comes from the result of `Analyze a file`. In case of sanitizing the content of an archive, the data_id of contained file can be found in `Fetch analysis result`.

QUERY PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
type	string		The specific value can be `cyclonedx` or `spdx`, with the default being JSON format as specified by SBOM.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Returns the SBOM report in the specified format.

RESPONSE MODEL - application/json

STATUS CODE - 404: Requests resource was not found.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Not found"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

3.14 POST /file/{data_id}/cancel

Cancel File Analysis

When cancelling a file analysis, the connected analysis (e.g. files in an archive) that are still in progress will be cancelled also.

The cancelled analysis will be automatically closed.

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*data_id	string		Unique submission identifier. Use this value to reference the submission.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Analysis was sucessfully cancelled.

RESPONSE MODEL - application/json

EXAMPLE:

```
{
 "<<data_id>>": "cancelled"
}
```

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 404: Data ID not found (invalid id) or Requests resource was not found

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

4. AUTH

Authentication APIs

User authentication is done via username & password. Additional integrations are available within the product:

- **LDAP** integration
- **Active Directory** integration
- **SAML** integration (starting with v4.18.0)

4.1 POST /login

Login

Initiate a new session. Required for using protected REST APIs.

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
user*	string	Username
password*	string	User's password

EXAMPLE:

```
{
 "user": "admin",
 "password": "admin"
}
```

RESPONSE

STATUS CODE - 200: OK

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
oms-csrf-token*	string	The randomly generated token used to prevent CSRF attacks
session_id*	string	The apikey used to make API calls which requires authentication

STATUS CODE - 403: Invalid credentials

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	<error message> will describe the incident. More details would be logged in MetaDefender Core logs

EXAMPLE:

```
{
 "err": "Failed to login"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

4.2 POST /logout

Logout

Destroy session for not using protected REST APIs.

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: OK

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
response*	string	

STATUS CODE - 400: Bad Request.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err*	string	

STATUS CODE - 403: Invalid user information.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err*	string	

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

4.3 POST /user/changepassword

Change Password

Modify the current password for the user identified by apikey

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
old_password	string	The current password in plain text
new_password	string	The new password in plain text

EXAMPLE:

```
{
 "old_password": "admin",
 "new_password": "123456"
}
```

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
success	boolean	

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

5. BATCH

Group the analysis requests in batches.

5.1 POST /file/batch

Initiate Batch

Create a new batch and retrieve the batch_id

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.
rule	string		Select rule for the analysis, if no header given the default rule will be selected (URL encoded UTF-8 string of rule name)
user_agent	string		user_agent header used to identify (and limit) access to a particular rule. For rule selection, `rule` header should be used. Since MetaDefender Core 5.6.0, when user agent is matched the predefined setting in workflow rule, the product will return X-Core-Id containing identity of MetaDefender Core server as a header of response.
user-data	string		Name of the batch (max 1024 bytes, URL encoded UTF-8 string).

RESPONSE

STATUS CODE - 200: Batch created successfully.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
batch_id*	string	The batch identifier used to submit files in the batch and to close the batch.

EXAMPLE:

```
{
 "batch_id": "74c85f475147439bac4d33b181853923"
}
```

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

5.2 POST /file/batch/{batchId}/close

Close Batch

The batch will be closed and files can no longer be added to the current batch.

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*batchId	string		The batch identifier used to submit files in the batch and to close the batch.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Batch successfully closed.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
batch_files	object	
batch_count	integer	The total number of files/entries in the batch.
current_finished_files	integer	The number of files/entries that have been completely processed.
files_in_batch	array	
data_id	string	Unique identifer for the file.
detected_by	integer	Total number of engines that detected this file.
display_name	string	The filename reported via `filename` header.
file_size	integer	Total file size in bytes.
file_type	string	The filetype using mimetype.
file_type_description	string	The filetype in human readable format.
process_info	object	
blocked_reason	string	Provides the reason why the file is blocked (if so).
progress_percentage	integer	Percentage of processing completed (from 1-100).

NAME	TYPE	DESCRIPTION
result	string	The final result of processing the file (Allowed / Blocked / Processing).
verdicts	array	
progress_percentage	integer	Track analysis progress until reaches 100.
scan_all_result_a	enum	**ALLOWED:** No Threat Detected, Infected, Suspicious, Failed, Whitelisted, Blacklisted, Exceeded Archive Depth, Not Scanned, Encrypted Archive, Exceeded Archive Size, Exceeded Archive File Number, Password Protected Document, Exceeded Archive Timeout, Mismatch, Potentially Vulnerable File, Cancelled, Sensitive Data Found, Yara Rule Matched, Potentially Unwanted, Unsupported File Type, Extraction Failed, Scan Failed, Suspicious Verdict by Sandbox, Likely Malicious Verdict by Sandbox, Malicious Verdict by Sandbox, Blocked Verdict by Sandbox, Blocked Verdict by Deep CDR, Global Timeout Exceeded, Vulnerable Verdict by SBOM, Non-vulnerable Verdict by SBOM, Blocked Verdict by SBOM, Blocked by Post Action, Known Bad, Known Good, Unknown, Allowed Verdict by COO, Blocked Verdict by COO, Unknown Verdict by COO, In Progress, Skip Processing Fast Symlink The overall scan result as string
scan_all_result_i	enum	**ALLOWED:** 0, 1, 2, 3, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36, 38, 39, 40, 41, 42, 43, 255, 1014 The overall scan result as index in the Processing Results table.
scanned_with	integer	The total number of engines used to analyze this file.
first_index	integer	The starting index in the batch. Used for pagination.
page_size	integer	The number of entries per page.
batch_id	string	The batch unique identifer
is_closed	boolean	The batch status (open/close).
process_info	object	
blocked_reason	string	Provides the reason why the file is blocked (if so).
file_type_skipped_scan	boolean	Indicates if the input file's detected type was configured to skip scanning.
profile	string	The used rule name.
result	string	The final result of processing the file (Allowed / Blocked / Processing).
user_agent	string	Identifier for the REST Client that calls the API.
username	string	User identifier who submitted scan request earlier.
scan_results	object	
batch_id	string	The batch unique identifer

NAME	TYPE	DESCRIPTION
scan_all_result_a	enum	**ALLOWED:** No Threat Detected, Infected, Suspicious, Failed, Whitelisted, Blacklisted, Exceeded Archive Depth, Not Scanned, Encrypted Archive, Exceeded Archive Size, Exceeded Archive File Number, Password Protected Document, Exceeded Archive Timeout, Mismatch, Potentially Vulnerable File, Cancelled, Sensitive Data Found, Yara Rule Matched, Potentially Unwanted, Unsupported File Type, Extraction Failed, Scan Failed, Suspicious Verdict by Sandbox, Likely Malicious Verdict by Sandbox, Malicious Verdict by Sandbox, Blocked Verdict by Sandbox, Blocked Verdict by Deep CDR, Global Timeout Exceeded, Vulnerable Verdict by SBOM, Non-vulnerable Verdict by SBOM, Blocked Verdict by SBOM, Blocked by Post Action, Known Bad, Known Good, Unknown, Allowed Verdict by COO, Blocked Verdict by COO, Unknown Verdict by COO, In Progress, Skip Processing Fast Symlink The overall scan result as string
scan_all_result_i	enum	**ALLOWED:** 0, 1, 2, 3, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36, 38, 39, 40, 41, 42, 43, 255, 1014 The overall scan result as index in the Processing Results table.
start_time	string	Timestamp when the scanning process starts.
total_avs	integer	Total number of scanning engines used as part of this analysis. Not like files, batch is not processed by engine, so this value is always 0.
total_time	integer	Total time elapsed during scan (in milliseconds).
user_data	string	Metadata submitted at batch creation.

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 404: Requests resource was not found.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Not found"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

5.3 POST /file/batch/{batchId}/close/callback

Close Batch with webhook

The batch will be closed and files can no longer be added to the current batch.

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*batchId	string		The batch identifier used to submit files in the batch and to close the batch.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.
*callbackurl	uri		Client's URL where MetaDefender Core will notify scan result back to whenever scan is finished (webhooks model). * Format: \<protocol://\>\<ip | domain\>:\<port\>\</path\> * Example: http://10.0.1.100:8081/listenback * Supported protocol: HTTP / HTTPS * Method: POST

RESPONSE

STATUS CODE - 200: Batch successfully closed.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
batch_files	object	
batch_count	integer	The total number of files/entries in the batch.
current_finished_files	integer	The number of files/entries that have been completely processed.
files_in_batch	array	
data_id	string	Unique identifer for the file.
detected_by	integer	Total number of engines that detected this file.
display_name	string	The filename reported via `filename` header.
file_size	integer	Total file size in bytes.
file_type	string	The filetype using mimetype.
file_type_description	string	The filetype in human readable format.
process_info	object	
blocked_reason	string	Provides the reason why the file is blocked (if so).
progress_percentage	integer	Percentage of processing completed (from 1-100).
result	string	The final result of processing the file (Allowed / Blocked / Processing).
verdicts	array	
progress_percentage	integer	Track analysis progress until reaches 100.

NAME	TYPE	DESCRIPTION
scan_all_result_a	enum	**ALLOWED:** No Threat Detected, Infected, Suspicious, Failed, Whitelisted, Blacklisted, Exceeded Archive Depth, Not Scanned, Encrypted Archive, Exceeded Archive Size, Exceeded Archive File Number, Password Protected Document, Exceeded Archive Timeout, Mismatch, Potentially Vulnerable File, Cancelled, Sensitive Data Found, Yara Rule Matched, Potentially Unwanted, Unsupported File Type, Extraction Failed, Scan Failed, Suspicious Verdict by Sandbox, Likely Malicious Verdict by Sandbox, Malicious Verdict by Sandbox, Blocked Verdict by Sandbox, Blocked Verdict by Deep CDR, Global Timeout Exceeded, Vulnerable Verdict by SBOM, Non-vulnerable Verdict by SBOM, Blocked Verdict by SBOM, Blocked by Post Action, Known Bad, Known Good, Unknown, Allowed Verdict by COO, Blocked Verdict by COO, Unknown Verdict by COO, In Progress, Skip Processing Fast Symlink The overall scan result as string
scan_all_result_i	enum	**ALLOWED:** 0, 1, 2, 3, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36, 38, 39, 40, 41, 42, 43, 255, 1014 The overall scan result as index in the Processing Results table.
scanned_with	integer	The total number of engines used to analyze this file.
first_index	integer	The starting index in the batch. Used for pagination.
page_size	integer	The number of entries per page.
batch_id	string	The batch unique identifer
is_closed	boolean	The batch status (open/close).
process_info	object	
blocked_reason	string	Provides the reason why the file is blocked (if so).
file_type_skipped_scan	boolean	Indicates if the input file's detected type was configured to skip scanning.
profile	string	The used rule name.
result	string	The final result of processing the file (Allowed / Blocked / Processing).
user_agent	string	Identifier for the REST Client that calls the API.
username	string	User identifier who submitted scan request earlier.
scan_results	object	
batch_id	string	The batch unique identifer

NAME	TYPE	DESCRIPTION
scan_all_result_a	enum	**ALLOWED:** No Threat Detected, Infected, Suspicious, Failed, Whitelisted, Blacklisted, Exceeded Archive Depth, Not Scanned, Encrypted Archive, Exceeded Archive Size, Exceeded Archive File Number, Password Protected Document, Exceeded Archive Timeout, Mismatch, Potentially Vulnerable File, Cancelled, Sensitive Data Found, Yara Rule Matched, Potentially Unwanted, Unsupported File Type, Extraction Failed, Scan Failed, Suspicious Verdict by Sandbox, Likely Malicious Verdict by Sandbox, Malicious Verdict by Sandbox, Blocked Verdict by Sandbox, Blocked Verdict by Deep CDR, Global Timeout Exceeded, Vulnerable Verdict by SBOM, Non-vulnerable Verdict by SBOM, Blocked Verdict by SBOM, Blocked by Post Action, Known Bad, Known Good, Unknown, Allowed Verdict by COO, Blocked Verdict by COO, Unknown Verdict by COO, In Progress, Skip Processing Fast Symlink The overall scan result as string
scan_all_result_i	enum	**ALLOWED:** 0, 1, 2, 3, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36, 38, 39, 40, 41, 42, 43, 255, 1014 The overall scan result as index in the Processing Results table.
start_time	string	Timestamp when the scanning process starts.
total_avs	integer	Total number of scanning engines used as part of this analysis. Not like files, batch is not processed by engine, so this value is always 0.
total_time	integer	Total time elapsed during scan (in milliseconds).
user_data	string	Metadata submitted at batch creation.

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 404: Requests resource was not found.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Not found"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

5.4 GET /file/batch/{batchId}

Status of Batch Analysis

Retrieve status report for the entire batch

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*batchId	string		The batch identifier used to submit files in the batch and to close the batch.

QUERY PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
first	integer		The first item order in the list of files in this batch
size	integer		The number of items to be fetched next, counting from the item order indicated in first header

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Batch progress paginated report (50 entries/page).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
batch_files	object	
batch_count	integer	The total number of files/entries in the batch.
current_finished_files	integer	The number of files/entries that have been completely processed.
files_in_batch	array	
data_id	string	Unique identifer for the file.
detected_by	integer	Total number of engines that detected this file.
display_name	string	The filename reported via `filename` header.
file_size	integer	Total file size in bytes.
file_type	string	The filetype using mimetype.
file_type_description	string	The filetype in human readable format.
process_info	object	
blocked_reason	string	Provides the reason why the file is blocked (if so).
progress_percentage	integer	Percentage of processing completed (from 1-100).
result	string	The final result of processing the file (Allowed / Blocked / Processing).
verdicts	array	
progress_percentage	integer	Track analysis progress until reaches 100.

NAME	TYPE	DESCRIPTION
scan_all_result_a	enum	**ALLOWED:** No Threat Detected, Infected, Suspicious, Failed, Whitelisted, Blacklisted, Exceeded Archive Depth, Not Scanned, Encrypted Archive, Exceeded Archive Size, Exceeded Archive File Number, Password Protected Document, Exceeded Archive Timeout, Mismatch, Potentially Vulnerable File, Cancelled, Sensitive Data Found, Yara Rule Matched, Potentially Unwanted, Unsupported File Type, Extraction Failed, Scan Failed, Suspicious Verdict by Sandbox, Likely Malicious Verdict by Sandbox, Malicious Verdict by Sandbox, Blocked Verdict by Sandbox, Blocked Verdict by Deep CDR, Global Timeout Exceeded, Vulnerable Verdict by SBOM, Non-vulnerable Verdict by SBOM, Blocked Verdict by SBOM, Blocked by Post Action, Known Bad, Known Good, Unknown, Allowed Verdict by COO, Blocked Verdict by COO, Unknown Verdict by COO, In Progress, Skip Processing Fast Symlink The overall scan result as string
scan_all_result_i	enum	**ALLOWED:** 0, 1, 2, 3, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36, 38, 39, 40, 41, 42, 43, 255, 1014 The overall scan result as index in the Processing Results table.
scanned_with	integer	The total number of engines used to analyze this file.
first_index	integer	The starting index in the batch. Used for pagination.
page_size	integer	The number of entries per page.
batch_id	string	The batch unique identifer
is_closed	boolean	The batch status (open/close).
process_info	object	
blocked_reason	string	Provides the reason why the file is blocked (if so).
file_type_skipped_scan	boolean	Indicates if the input file's detected type was configured to skip scanning.
profile	string	The used rule name.
result	string	The final result of processing the file (Allowed / Blocked / Processing).
user_agent	string	Identifier for the REST Client that calls the API.
username	string	User identifier who submitted scan request earlier.
scan_results	object	
batch_id	string	The batch unique identifer

NAME	TYPE	DESCRIPTION
scan_all_result_a	enum	**ALLOWED:** No Threat Detected, Infected, Suspicious, Failed, Whitelisted, Blacklisted, Exceeded Archive Depth, Not Scanned, Encrypted Archive, Exceeded Archive Size, Exceeded Archive File Number, Password Protected Document, Exceeded Archive Timeout, Mismatch, Potentially Vulnerable File, Cancelled, Sensitive Data Found, Yara Rule Matched, Potentially Unwanted, Unsupported File Type, Extraction Failed, Scan Failed, Suspicious Verdict by Sandbox, Likely Malicious Verdict by Sandbox, Malicious Verdict by Sandbox, Blocked Verdict by Sandbox, Blocked Verdict by Deep CDR, Global Timeout Exceeded, Vulnerable Verdict by SBOM, Non-vulnerable Verdict by SBOM, Blocked Verdict by SBOM, Blocked by Post Action, Known Bad, Known Good, Unknown, Allowed Verdict by COO, Blocked Verdict by COO, Unknown Verdict by COO, In Progress, Skip Processing Fast Symlink The overall scan result as string
scan_all_result_i	enum	**ALLOWED:** 0, 1, 2, 3, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36, 38, 39, 40, 41, 42, 43, 255, 1014 The overall scan result as index in the Processing Results table.
start_time	string	Timestamp when the scanning process starts.
total_avs	integer	Total number of scanning engines used as part of this analysis. Not like files, batch is not processed by engine, so this value is always 0.
total_time	integer	Total time elapsed during scan (in milliseconds).
user_data	string	Metadata submitted at batch creation.

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 404: Requests resource was not found.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Not found"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

5.5 GET /file/batch/{batchId}/certificate

Download Signed Batch Result

Download digitally signed status report for the entire batch

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*batchId	string		The batch identifier used to submit files in the batch and to close the batch.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.
metadata	json		In JSON format, this can be used to: Include additional information in the response YML. Currently, one supported field in the metadata is `include_vul_info`, which can be set to `true` or `false` to indicate whether vulnerability processing information should be included.

RESPONSE

STATUS CODE - 200: Signed batch result and certificate are sent back in response body (YAML format).

RESPONSE MODEL - application/x-yaml

EXAMPLE:


```
"--- batch_id: 092876200fb54cfb80b6e3332c410ae9 user_data: the user data from the header from
batch creation cert_sha1_fingerprint: <some cert serial value> batch_files:\n  batch_count: 1\n
files_in_batch:\n  - data_id: 9112b225f0634f189a2bb46ec1a7826f\n    display_name:
New%20Text%20Document.txt\n    file_size: 5\n    scan_all_result_i: 0\n    process_info:\n
blocked_reason:\n      result: Allowed\n    md5: 42b130c3ce46e058f30712838cebf420\n    sha1:
ed94baf55ca851055fb76045f6949bca2f865605\n    sha256:
f4191b3ec6ce93aaf712919a38e52815c5da9c91d2b141df920bc8bcb5cbb8e3\n    sha512: \"\"\n
vulnerabilities:\n      - cve: CVE-2021-45463\n        cvss:\n          score: 6.8\n
cvss_3_0:\n          base_score: 7.8\n      - cve: CVE-2018-12713\n        cvss:\n
score: 6.4\n        cvss_3_0:\n          base_score: 9.1\nprocess_info:\n  blocked_reason:\n
file_type_skipped_scan: false\n  profile: File scan\n  result: Allowed\n  user_agent:
webscan\nscan_results:\n  scan_all_result_a: No Threat Detected\n  scan_all_result_i: 0\n
start_time: 2017-05-23T11:22:03.010Z\n  total_avs: 14\n  total_time: 995\n...\n--- signature:
881d22220c4ca0557d7c7d5c5794d53a8a2780997cd65b27b6e7f1c099a15de03dbcb5edbeaea7aafa6099fab37be07
017b39e3e3a7d66c550f44eb59a096c54d5b9555cb28198546fbec57c33b717751d333a09733d95dd876e2798d044c8
caef828f4352b91f9a6d057253bb1a9461e0e0e0bf4313a80895998d645bebc81841ff3499589c80ffc4e8a190d1ec9
b3e4126d86659d303b0e1f22d9289c9c4671d35532b55ad4620e048a78bb405b573897da63efdd5f036692c934a82d9
bdc9b9862e7fea5e8abeeb1444be0689d50373c5c0632484950c0fe0337ed5f91bdf26986f7cff8aa3431bf4bc948fc
127c16ba13ec679fe9f67e7586075c1f467454fa8cf40e9cd501291c95d862eb16f4477c17d1711294f0ff2b3a1140b
d53dbd1fbb0846af6062e9e4e2e1a09af3448503ed11e342164e535fc268bf7d8fbc28ed946cd2bb8ea075f2295d2fa
8392076d41608c3b5decf8fab3a5ec7de190f07583331e0517e5f361735cd59326622dc8b07b10a464028de781a063e
408f918c1d5534329140f4e4dc1a717d808d6784410410b00d36cb9a345f5bbc11fa1c58ee28f8e7b863f3ea2c923ec
5fb2ac29eaa4ddc0d6d9dfd3f16a97f207dc2858410a577c7f4a92ff01bad3229f5fcdb08e21df9869a113272aa9d96
bfdfe8bfb3a50414c174e16a3504e5780c2718779b0757298546f287ef7ea86e67510d48a8 certificate: |\n
-----BEGIN CERTIFICATE-----\n
MIIGJzCCBA+gAwIBAgIBATANBgkqhkiG9w0BAQUFADCBsjELMAkGA1UEBhMCRlIx\n
DzANBgNVBAgMBkFsc2FjZTETMBEGA1UEBwwKU3RyYXNib3VyZzEYMBYGA1UECgwP\n
d3d3LmZyZWVsYW4ub3JnMRAwDgYDVQQLDAdmcmVlbGFuMS0wKwYDVQQDDCRGcmVl\n
bGFuIFNhbXBsZSBDZXJ0aWZpY2F0ZSBBdXRob3JpdHkxIjAgBgkqhkiG9w0BCQEW\n
E2NvbnRhY3RAZnJlZWxhbi5vcmcwHhcNMTIwNDI3MTAzMTE4WhcNMjIwNDI1MTAz\n
MTE4WjB+MQswCQYDVQQGEwJGUjEPMA0GA1UECAwGQWxzYWNlMRgwFgYDVQQKDA93\n
```

```
d3cuZnJlZWxhbi5vcmcxEDAOBgNVBAsMB2ZyZWVsYW4xDjAMBgNVBAMMBWFsaWNl\n
MSIwIAYJKoZIhvcNAQkBFhNjb250YWN0QGZyZWVsYW4ub3JnMIICIjANBgkqhkiG\n
9w0BAQEFAAOCAg8AMIICCgKCAgEA3W29+ID6194bH6ejLrIC4hb2Ugo8v6ZC+Mrc\n
k2dNYMNPjcOKABvxxEtBamnSaeU/IY7FC/giN622LEtV/3oDcrua0+yWuVafyxmZ\n  yTKUb4/GUgafRQPf/
eiX9urWurtIK7XgNGFNUjYPq4dSJQPPhwCHE/LKAykWnZBX\n
RrX0Dq4XyApNku0IpjIjEXH+8ixE12wH8wt7DEvdO7T3N3CfUbaITl1qBX+Nm2Z6\n  q4Ag/
u5rl8NJfXg71ZmXA3XOj7zFvpyapRIZcPmkvZYn7SMCp8dXyXHPdpSiIWL2\n  uB3KiO4JrUYvt2GzLBUThp+lNSZaZ/
Q3yOaAAUkOx+1h08285Pi+P8lO+H2Xic4S\n
vMq1xtLg2bNoPC5KnbRfuFPuUD2/3dSiiragJ6uYDLOyWJDivKGt/72OVTEPAL9o\n
6T2pGZrwbQuiFGrGTMZOvWMSpQtNl+tCCXlT4mWqJDRwuMGrI4DnnGzt3IKqNwS4\n
Qyo9KqjMIPwnXZAmWPm3FOKe4sFwc5fpawKO01JZewDsYTDxVj+cwXwFxbE2yBiF\n  z2FAHwfopwaH35p3C6lkcgP2k/
zgAlnBluzACUI+MKJ/G0gv/uAhj1OHJQ3L6kn1\n  SpvQ41/
ueBjlunExqQSYD7GtZ1Kg8uOcq2r+WISE3Qc9MpQFFkUVllmgWGwYDuN3\n
Zsez95kCAwEAAaN7MHkwCQYDVR0TBAIwADAsBglghkgBhvhCAQ0EHxYdT3BlblNT\n
TCBHZW5lcmF0ZWQgQ2VydGlmaWNhdGUwHQYDVR0OBBYEFFlfyRO6G8y5qEFKikl5\n
ajb2fT7XMB8GA1UdIwQYMBaAFCNsLT0+KV14uGw+quK7Lh5sh/JTMA0GCSqGSIb3\n
DQEBBQUAA4ICAQAT5wJFPqervbja5+90iKxi1d0QVtVGB+z6aoAMuWK+qgi0vgvr\n
mu9ot2lvTSCSnRhjeiP0SIdqFMORmBtOCFk/kYDp9M/91b+vS+S9eAlxrNCB5VOf\n  PqxEPp/wv1rBcE4GBO/
c6HcFon3F+oBYCsUQbZDKSSZxhDm3mj7pb67FNbZbJIzJ\n
70HDsRe2O04oiTx+h6g6pW3cOQMgIAvFgKN5Ex727K4230B0NIdGkzuj4KSML0NM\n
slSAcXZ41OoSKNjy44BVEZv0ZdxTDrRM4EwJtNyggFzmtTuV02nkUj1bYYYC5f0L\n
ADr6s0XMyaNk8twlWYlYDZ5uKDpVRVBfiGcq0uJIzIvemhuTrofh8pBQQNkPRDFT\n  Rq1iTo1Ihhl3/
Fl1kXk1WR3jTjNb4jHX7lIoXwpwp767HAPKGhjQ9cFbnHMEtkro\n
RlJYdtRq5mccDtwT0GFyoJLLBZdHHMHJz0F9H7FNk2tTQQMhK5MVYwg+LIaee586\n
CQVqfbscp7evlgjLW98H+5zylRHAgoH2G79aHljNKMp9BOuq6SnEglEsiWGVtu2l\n  hnx8SB3sVJZHeer8f/
UQQwqbAO+Kdy70NmbSaqaVtp8jOxLiidWkwSyRTsuU6D8i\n
DiH5uEqBXExjrj0FslxcVKdVj5glVcSmkLwZKbEU1OKwleT/iXFhvooWhQ==\n  -----END CERTIFICATE-----\n...
\n"
```


STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
```

```
 "err": "Access denied"
}
```

STATUS CODE - 404: Requests resource was not found.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Not found"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

5.6 POST /file/batch/{batchId}/cancel

Cancel Batch

When cancelling a batch, the connected analysis that are still in progress will be cancelled also.

The cancelled batch will be closed.

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION

NAME	TYPE	EXAMPLE	DESCRIPTION
*batchId	string		The batch identifier used to submit files in the batch and to close the batch.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Batch cancelled.

RESPONSE MODEL - application/json

EXAMPLE:

```
{
 "<<batch_id>>": "cancelled"
}
```

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
```

```
}
```

STATUS CODE - 404: Batch not found (invalid id)

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6. CONFIG

Configure the product through APIs (especially the Settings). Will require admin apikey.

6.1 PUT /admin/config/auditlog

Audit clean up

Setting audit records (update history) clean up time (clean up records older than).

Note:The clean up range is defined in hours.

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
cleanuprange	integer	The number of hours of data retention. Anything older than this number will be deleted.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
cleanuprange	integer	The number of hours of data retention. Anything older than this number will be deleted.

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6.2 PUT /admin/config/quarantine

Quarantine clean up

Setting quarantine clean up time (clean up records older than).

Note:The clean up range is defined in hours.

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
cleanuprange	integer	The number of hours of data retention. Anything older than this number will be deleted.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
cleanuprange	integer	The number of hours of data retention. Anything older than this number will be deleted.

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6.3 PUT /admin/config/sanitize

Sanitized files clean up

Setting sanitized files clean up time (clean up records older than).

Note:The clean up range is defined in minutes.

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
maxage	integer	The number of minutes of data retention. Anything older than this number will be deleted.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
maxage	integer	The number of minutes of data retention. Anything older than this number will be deleted.

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		

NAME	TYPE	DESCRIPTION
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6.4 PUT /admin/config/scanhistory

Processing history clean up

Setting processing history clean up time (clean up records older than).

Note:The clean up range is defined in hours.

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
cleanuprange	integer	The number of hours of data retention. Anything older than this number will be deleted.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		

NAME	TYPE	DESCRIPTION
cleanuprange	integer	The number of hours of data retention. Anything older than this number will be deleted.

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6.5 PUT /admin/config/session

Session settings

Configure settings for session generated upon a successful login See more at Login

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
absoluteSessionTimeout	integer	The interval (in milliseconds) for overall session length timeout (regardless of activity).
allowCrossIpSessions	boolean	Allow requests from the same user to come from different IPs.
allowDuplicateSession	boolean	Allow same user to have multiple active sessions.
sessionTimeout	integer	The interval (in milliseconds) for the user's session timeout, based on last activity. Timer starts after the last activity for the apikey.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
absoluteSessionTimeout	integer	The interval (in milliseconds) for overall session length timeout (regardless of activity).
allowCrossIpSessions	boolean	Allow requests from the same user to come from different IPs.
allowDuplicateSession	boolean	Allow same user to have multiple active sessions.
sessionTimeout	integer	The interval (in milliseconds) for the user's session timeout, based on last activity. Timer starts after the last activity for the apikey.

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6.6 PUT /admin/config/update

Modules Update Source and Frequency

Setting update frequency for all modules and engines.

Note:The clean up range is defined in minutes.

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
autoupdateperiod	integer	The interval (in minutes) for checking for new updates.
deleteafterimport	boolean	If you want to clean the pickup folder after the updates have been applied.
disabledupdate	array	
days	string	A particular day (index of the week) or a defined interval (Fri-Sun -> 5-7)
from	integer	When to start the lockdown, expressed in minutes (8 AM=480)
to	integer	When to end the lockdown, expressed in minutes (4 PM=960)
pickupfolder	string	The folder where MetaDefender will look for the new engine files.
source	enum	**ALLOWED:** `internet, folder, manual` Define where the updates will be loaded from. <p> This can be either: * `internet` -> if selected, will check for new updates every `autoupdateperiod` minutes * `folder` -> make sure that MetaDefender has access/permission to that folder * `manual` -> requires manually uploading the packages in Inventory > Modules > Upload package.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
autoupdateperiod	integer	The interval (in minutes) for checking for new updates.
deleteafterimport	boolean	If you want to clean the pickup folder after the updates have been applied.
disabledupdate	array	
days	string	A particular day (index of the week) or a defined interval (Fri-Sun -> 5-7)
from	integer	When to start the lockdown, expressed in minutes (8 AM=480)
to	integer	When to end the lockdown, expressed in minutes (4 PM=960)
pickupfolder	string	The folder where MetaDefender will look for the new engine files.
source	enum	**ALLOWED:** `internet, folder, manual` Define where the updates will be loaded from. <p> This can be either: * `internet` -> if selected, will check for new updates every `autoupdateperiod` minutes * `folder` -> make sure that MetaDefender has access/permission to that folder * `manual` -> requires manually uploading the packages in Inventory > Modules > Upload package.

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6.7 GET /config/skip/{skip_type}/template

Download CSV template for 'skip by hash' CSV upload.

Download CSV template for 'skip by hash' CSV upload.

Note: Only available since MetaDefender Core 5.12.1.

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*skip_type	string		The type of 'skip by hash' template to download, can be `blocklist`, `allowlist` or `enginelist`.

RESPONSE

STATUS CODE - 200: CSV template of the requested 'skip by hash' type.

RESPONSE MODEL - text/csv

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6.8 POST /admin/config/skip/{skip_type}

Upload csv file of 'skip by hash' list.

Add hashes by uploading csv file of 'skip by hash' list.

Note: Only available since MetaDefender Core 5.12.1. Skip hashes will be overwritten if it

already exists.

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*skip_type	string		The type of 'skip by hash' list to upload, can be `blocklist`, `allowlist` or `enginelist`.

QUERY PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
summary	boolean		The response body includes summaries on the number of hashes added and modified for each type of 'skip by hash'. The default value is `false`.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: A list of all skip/allow/block-listed hashes or the summary of new/modfied hashes per type.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
ONE:OF	object	
OPTION:1	object	
blacklist*	object	
skip*	object	
whitelist*	object	
total_count*	integer	
OPTION:2	object	

NAME	TYPE	DESCRIPTION
number_blocklist_accepted*	integer	Total number of new or modified hashes of blacklist.
number_enginelist_accepted*	integer	Total number of new or modified hashes of skip engine list.
number_allowlist_accepted*	integer	Total number of new or modified hashes of whitelist.

STATUS CODE - 400: Bad Request(e.g. invalid headers, empty body content).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err*	string	

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 422: Unprocessable Content (e.g. invalid content of file).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Invalid CSV format"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6.9 GET /admin/config/skip

Get 'skip by hash' list

Get list of hashes that match the search conditions.

Note: Since MetaDefender Core 5.12.1:

- Support 4 query strings: first, size, search and types.
- If no query strings are defined, all hash data will be returned.
- The hash list responded includes an additional value for the total search counts and a timestamp for each hash indicating when it was added or modified.

REQUEST

QUERY PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
first	integer		The first item order in the list of hashes. Default is `0`.
size	integer		The number of items to be fetched, counting from the item order indicated in `first` parameter. Default is `20`.

NAME	TYPE	EXAMPLE	DESCRIPTION
search	string		Text to search. The can be a part of (MD5, SHA-1, SHA-256, SHA-512) hashes, or a part of the description and must be URL-encoded. Default is empty (any value).
types	string		The types of hashes to search. This option can include a single type, a couple of types, or all three types in the list (blocklist, allowlist, enginelist) separated by `|`. Default value is `blocklist|enginelist|allowlist`.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Search results are a list of skip/allow/block-listed hashes.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
blacklist*	object	
skip*	object	
whitelist*	object	
total_count*	integer	

STATUS CODE - 400: Bad Request(e.g. invalid parameters).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err*	string	

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6.10 PUT /admin/config/skip

Modify 'skip by hash' list

Modify/Add hashes or overwrite all hashes in 'skip by hash' list.

Note: Since MetaDefender Core 5.12.1:

- Support 2 query strings: overwrite and summary.
- The hash list responded includes an additional value for the total hash counts and a timestamp for each hash indicating when it was added or modified.

REQUEST

QUERY PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
overwrite	boolean		Overwrite all hashes in 'skip by hash' list. The default value is `true`.
summary	boolean		The response body includes summaries on the number of hashes added and/or modified for each type of 'skip by hash'. The default value is `false`.

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
blacklist	object	
skip	object	
whitelist	object	

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: A list of all skip/allow/block-listed hashes or the summary of new/modfied hashes per type.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
ONE:OF	object	
OPTION:1	object	
blacklist*	object	
skip*	object	
whitelist*	object	
total_count*	integer	
OPTION:2	object	

NAME	TYPE	DESCRIPTION
number_blocklist_accepted*	integer	Total number of new or modified hashes of blacklist.
number_enginelist_accepted*	integer	Total number of new or modified hashes of skip engine list.
number_allowlist_accepted*	integer	Total number of new or modified hashes of whitelist.

STATUS CODE - 400: Bad Request(e.g. invalid headers, invalid request body).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err*	string	

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6.11 POST /admin/config/skip

Add hashes to 'skip by hash' list.

Add hashes to 'skip by hash' list.

Note: Since MetaDefender Core 5.12.1:

- Support query string: summary.
- Skip hashes will be overwritten if it already exists.
- The hash list responded includes an additional value for the total hash counts and a timestamp for each hash indicating when it was added or modified.

REQUEST

QUERY PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
summary	boolean		The response body includes summaries on the number of hashes added and modified for each type of 'skip by hash'. The default value is `false`.

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
blacklist	object	
skip	object	
whitelist	object	

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: A list of all skip/allow/block-listed hashes or the summary of new/modfied hashes per type.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
ONE:OF	object	
OPTION:1	object	
blacklist*	object	
skip*	object	
whitelist*	object	
total_count*	integer	
OPTION:2	object	
number_blocklist_accepted*	integer	Total number of new or modified hashes of blacklist.
number_enginelist_accepted*	integer	Total number of new or modified hashes of skip engine list.
number_allowlist_accepted*	integer	Total number of new or modified hashes of whitelist.

STATUS CODE - 400: Bad Request(e.g. invalid header, invalid request body).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err*	string	

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
```

```
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 422: Unprocessable Content (e.g. invalid content of file).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Invalid CSV format"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6.12 DELETE /admin/config/skip

Delete hashes from 'skip by hash' list

Delete multiple hashes from "skip by hash" list.

Note: Since MetaDefender Core 5.12.1:

- Support query string: summary.

REQUEST

QUERY PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
summary	boolean		The response body includes summaries on the number of hashes deleted. The default value is `false`.

REQUEST BODY - application/json

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: A list of all skip/allow/block-listed hashes or the summary of deleted hashes per type.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
ONE:OF	object	
OPTION:1	object	
blacklist*	object	
skip*	object	
whitelist*	object	
total_count*	integer	
OPTION:2	object	

NAME	TYPE	DESCRIPTION
number_hashes_removed*	integer	Total number of hashes deleted.

STATUS CODE - 400: Bad Request(e.g. invalid header, invalid request body).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err*	string	

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
```

```
}
```

6.13 GET /admin/config/webhook

Webhook get configuration

Getting settings supported for webhook mode

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
maxretrytime	number	Number of allowed retries sending callback to client when failed. > _**For example**_: maxretrytime= 3 Sending callback failed for some reasons * 1st retry triggered after delayduration (ms) * 2nd retry triggered after delayduration+ delayprogression (ms) * 3rd retry triggered after delayduration+ 2*delayprogression (ms)
delayduration	number	in milliseconds
delayprogression	number	in milliseconds
requestqueue	number	Total requests can be sent concurrently for each worker.
transfertimeout	number	After this transfer timeout (in milliseconds), if MetaDefender Core still doesn't receive webhook server's response, it will try to send webhook again.
workercount	number	Total number of workers for sending the result back to Webhook client. Default is 1. Service restart is required for changes to take effect.

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6.14 PUT /admin/config/webhook

Webhook set configuration

Modifying settings supported for webhook mode

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
maxretrytime	number	Number of allowed retries sending callback to client when failed. > _**For example**_: maxretrytime= 3 Sending callback failed for some reasons * 1st retry triggered after delayduration (ms) * 2nd retry triggered after delayduration+ delayprogression (ms) * 3rd retry triggered after delayduration+ 2*delayprogression (ms)
delayduration	number	in milliseconds
delayprogression	number	in milliseconds
requestqueue	number	Total requests can be sent concurrently for each worker.
transfertimeout	number	After this transfer timeout (in milliseconds), if MetaDefender Core still doesn't receive webhook server's response, it will try to send webhook again.
workercount	number	Total number of workers for sending the result back to Webhook client. Default is 1. Service restart is required for changes to take effect.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
maxretrytime	number	Number of allowed retries sending callback to client when failed. > _**For example**_: maxretrytime= 3 Sending callback failed for some reasons * 1st retry triggered after delayduration (ms) * 2nd retry triggered after delayduration+ delayprogression (ms) * 3rd retry triggered after delayduration+ 2*delayprogression (ms)
delayduration	number	in milliseconds
delayprogression	number	in milliseconds
requestqueue	number	Total requests can be sent concurrently for each worker.

NAME	TYPE	DESCRIPTION
transfertimeout	number	After this transfer timeout (in milliseconds), if MetaDefender Core still doesn't receive webhook server's response, it will try to send webhook again.
workercount	number	Total number of workers for sending the result back to Webhook client. Default is 1. Service restart is required for changes to take effect.

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 404: Requests resource was not found.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Not found"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6.15 GET /admin/config/proxy

Get proxy

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: A list of all proxy.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
enabled	boolean	
feature_proxy	array	
exclusion	string	
id	enum	**ALLOWED:** `callbackurl, downloadfrom, license, mdcloud, updateengine, sanitizedurl, ssologin, yara` Feature ID.
port	string	
server	string	
use_proxy	enum	**ALLOWED:** `0, 1, 2` 0 - this feature does not use proxy. 1 - this feature use master proxy setting. 2 - this feature use a custom proxy.
username	string	
exclusion	string	
port	string	
server	string	
username	string	

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		

NAME	TYPE	DESCRIPTION
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

6.16 PUT /admin/config/proxy

Modify proxy

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
enabled*	boolean	When true, use a proxy server for MetaDefender Core
feature_proxy	array	
exclusion	string	
id	enum	**ALLOWED:** callbackurl, downloadfrom, license, mdcloud, updateengine, sanitizedurl, ssologin, yara Feature ID.
port	string	
server	string	
use_proxy	enum	**ALLOWED:** 0, 1, 2 0 - this feature does not use proxy. 1 - this feature use master proxy setting. 2 - this feature use a custom proxy.
username	string	
exclusion	string	Not use the proxy server for the addresses starting with the following entries
port*	string	Proxy server port
server*	string	Proxy server host address
username	string	Username for proxy authentication
password	string	Password for proxy authentication

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: A list of all proxy.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
enabled	boolean	
feature_proxy	array	
exclusion	string	
id	enum	ALLOWED: callbackurl, downloadfrom, license, mdcloud, updateengine, sanitizedurl, ssologin, yara Feature ID.
port	string	
server	string	
use_proxy	enum	ALLOWED: 0, 1, 2 0 - this feature does not use proxy. 1 - this feature use master proxy setting. 2 - this feature use a custom proxy.
username	string	
exclusion	string	
port	string	
server	string	
username	string	

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
ONE:OF	object	
OPTION:1	object	
err	string	Missing enabled field
OPTION:2	object	
err	string	Error while parsing input json
OPTION:3	object	
err	string	Missing port
OPTION:4	object	
err	string	Missing server address
OPTION:5	object	
err	string	Test proxy connection failed. Error: Proxy requires authentication

6.17 POST /admin/config/proxy/testconnection

Check connection to proxy server

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
enabled*	boolean	This redundant attribute will be removed soon.
exclusion	string	Not use the proxy server for the addresses starting with the following entries
port*	string	Proxy server port
server*	string	Proxy server host address
username	string	Username for proxy authentication
password	string	Password for proxy authentication

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
result	string	

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
ONE:OF	object	
OPTION:1	object	
err	string	Missing port
OPTION:2	object	

NAME	TYPE	DESCRIPTION
err	string	Missing server address
OPTION:3	object	
err	string	Test proxy connection failed. Error: Proxy requires authentication

6.18 GET /admin/config/file/sync

Synchronous scan connection timeout - Retrieval

Retrieving sync scan related connection timeout value (in minutes)

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
timeout*	number	Sync scan related connection timeout value (in minutes).

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
```

```
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Error while requesting configuration.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
ONE:OF	object	
OPTION:1	object	
err	string	Error while requesting configuration
OPTION:2	object	
err	string	Internal server error

6.19 PUT /admin/config/file/sync

Synchronous scan connection timeout - Modification

Modifying sync scan related connection timeout value (in minutes)

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
timeout*	number	Sync scan related connection timeout value (in minutes). The value is limited in range [1, 30] minutes.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
timeout*	number	Sync scan related connection timeout value (in minutes).

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Error while modifying configuration.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		

NAME	TYPE	DESCRIPTION
ONE:OF	object	
OPTION:1	object	
err	string	Error while modifying configuration Setting timeout value out of the limit range [1, 30] minutes, or the request body is not in JSON format.
OPTION:2	object	
err	string	Internal server error

6.20 GET /admin/config/sessioncookie

Get session cookie attributes

Getting session cookie attributes

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
samesite	number	SameSite attribute accepts three values: * `Strict` - cookies will only be sent in a first-party context, not be sent along with requests initiated by third party websites. * `Lax` - cookies are not sent on normal cross-site subrequests, but are sent when a user is navigating to the origin site. * `None` - cookies will be sent in all contexts. Default value: `Lax`

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6.21 PUT /admin/config/sessioncookie

Update session cookie attributes

Modifying session cookie attributes

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
samesite	number	SameSite attribute accepts three values: * `Strict` - cookies will only be sent in a first-party context, not be sent along with requests initiated by third party websites. * `Lax` - cookies are not sent on normal cross-site subrequests, but are sent when a user is navigating to the origin site. * `None` - cookies will be sent in all contexts. Default value: `Lax`

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
samesite	number	SameSite attribute accepts three values: * `Strict` - cookies will only be sent in a first-party context, not be sent along with requests initiated by third party websites. * `Lax` - cookies are not sent on normal cross-site subrequests, but are sent when a user is navigating to the origin site. * `None` - cookies will be sent in all contexts. Default value: `Lax`

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 404: Requests resource was not found.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Not found"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6.22 GET /admin/config/cacheuser

Get user caching setting

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
enabled	boolean	If you want to enable user caching to improve performance, set this key to true.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
enabled	boolean	If you want to enable user caching to improve performance, set this key to true.

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6.23 PUT /admin/config/cacheuser

Enable user caching

If you want to enable user caching to improve performance, set this key to true.

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
enabled	boolean	If you want to enable user caching to improve performance, set this key to true.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
enabled	boolean	If you want to enable user caching to improve performance, set this key to true.

STATUS CODE - 400: Bad request

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	enum	ALLOWED: Missing key enabled, Invalid value: enabled, Invalid number of keys. There should be 1 key in total err: * Missing key enabled * Invalid value: enabled * Invalid number of keys. There should be 1 key in total

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6.24 GET /admin/config/customresponseheader

Get a list of custom response headers

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
include_address	object	
enabled	boolean	Include data from address_str.
address_str	string	IP address.
include_deployment_id	boolean	Include generated deployment ID.
specify_users_agent	object	
enabled	boolean	True: required matched header user_agent to given list in allowed_users_agent to return new header X-Core-Id in response.
allowed_users_agent	array	
authorized_only	boolean	Only return new X-Core-Id header for authorized user.

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6.25 PUT /admin/config/customresponseheader

Update list of custom response headers

Modifying list of custom response headers

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
include_address	object	
enabled	boolean	Include data from address_str.
address_str	string	IP address.
include_deployment_id	boolean	Include generated deployment ID.
specify_users_agent	object	
enabled	boolean	True: required matched header user_agent to given list in allowed_users_agent to return new header X-Core-Id in response.
allowed_users_agent	array	
authorized_only	boolean	Only return new X-Core-Id header for authorized user.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
include_address	object	
enabled	boolean	Include data from address_str.
address_str	string	IP address.
include_deployment_id	boolean	Include generated deployment ID.
specify_users_agent	object	
enabled	boolean	True: required matched header user_agent to given list in allowed_users_agent to return new header X-Core-Id in response.
allowed_users_agent	array	
authorized_only	boolean	Only return new X-Core-Id header for authorized user.

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 404: Requests resource was not found.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Not found"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6.26 GET /admin/config/healthcheck

Get health check

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Health check criterias to determine whether MetaDefender Core server status is healthy or not.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
database	boolean	
enable_feature	boolean	
number_active_av_engines	integer	
engines	array	

NAME	TYPE	DESCRIPTION
id	string	
display_name	string	
license	boolean	
scan_queue	integer	unit is percentage "%". To disable health check on scan queue status, set 0.

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

6.27 POST /admin/config/healthcheck

Modify health check

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
enable_feature	boolean	
scan_queue	integer	

NAME	TYPE	DESCRIPTION
number_active_av_engines	integer	
required_engines	array	

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Health check criterias to determine whether MetaDefender Core server status is healthy or not.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
database	boolean	
enable_feature	boolean	
number_active_av_engines	integer	
engines	array	
id	string	
display_name	string	
license	boolean	
scan_queue	integer	unit is percentage "%". To disable health check on scan queue status, set 0.

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
ONE:OF	object	
OPTION:1	object	
err	string	Failed to set required_engines when license is not valid
OPTION:2	object	
err	string	Failed to set required_engines. Required_engines must be a licensed engine

STATUS CODE - 500: Error while modifying configuration.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error while modifying configuration Setting timeout value out of the limit range [1, 30] minutes, or the request body is not in JSON format.

6.28 GET /admin/config/logger/syslog

Retrieve the syslog configuration details

This API is for retrieving the current details of the syslog configuration.

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
syslog_level	enum	**ALLOWED:** dump, debug, info, warning, error The syslog output level.
sources	array	
syslog	string	The address of the syslog server: `local"|"<protocol>://<address>:<port>`
status	enum	**ALLOWED:** connecting, retrying, connected, ready The status of the connection to the syslog server. connecting - attempting to establish a connection to the syslog server. retrying - connection to the syslog server failed. Retrying to establish the connection.... connected - successfully connected to the syslog server and ready to send log messages. ready - successfully connected to the local syslog server (for requests to add a 'local' syslog on Linux OS).
message	string	The message to describe information or error.

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error while `<action>` configuration for syslog: `<reason>` action - can be `getting`, `setting`, `adding`, `deleting`, `verifying` reason - is the unhandled error reason

EXAMPLE:

```
{
 "err": "Error while <action> configuration for syslog: <reason>"
}
```

6.29 PUT /admin/config/logger/syslog

Add new syslog servers or change syslog level

This API is used to add new syslog servers or to request a change in the syslog level. This will

not impact any existing syslog servers. If a syslog server already exists, it will be ignored. If no syslog_level is specified in the API request header, the existing one will be retained. The default value for syslog_level is info.

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
syslog_level	enum	**ALLOWED:** dump, debug, info, warning, error The syslog output level. The default value is `info`
sources	array	
syslog	string	The address of the syslog server: `local"|"<protocol>://<address>:<port>`

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
syslog_level	enum	**ALLOWED:** dump, debug, info, warning, error The syslog output level.
sources	array	
syslog	string	The address of the syslog server: `local"|"<protocol>://<address>:<port>`
result	enum	**ALLOWED:** success, initializing_failed, existed, failed Result of the request to add syslog servers. success - successfully added the syslog server. initializing_failed - failed to establish a connection to the syslog server and retrying... existed - successfully connected to the syslog server and ready to send log messages. failed - failed to add the syslog server while creating a local syslog on Windows OS or due to other errors (further investigation is needed to determine the cause).
message	string	The message to describe information or error.

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 409: Conflicting request

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Logger configuration setting is processing, please wait for it to finish"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	`string`	Error while `<action>` configuration for syslog: `<reason>` action - can be `getting`, `setting`, `adding`, `deleting`, `verifying` reason - is the unhandled error reason

EXAMPLE:

```
{
 "err": "Error while <action> configuration for syslog: <reason>"
}
```

6.30 POST /admin/config/logger/syslog

Forcefully apply the new syslog server configurations

This API is used to forcefully apply new syslog server configurations, replacing any existing configurations or creating a completely new syslog connection instance. If no syslog_level is specified in the API request header, the current value will be retained. The default value for syslog_level is info.

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
syslog_level	`enum`	**ALLOWED:** `dump, debug, info, warning, error` The syslog output level. The default value is `info`
sources	`array`	
syslog	`string`	The address of the syslog server: `local"|"<protocol>://<address>:<port>`

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	`string`		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
syslog_level	enum	**ALLOWED:** dump, debug, info, warning, error The syslog output level.
sources	array	
syslog	string	The address of the syslog server: `local"|"<protocol>://<address>:<port>`
result	enum	**ALLOWED:** success, initializing_failed, existed, failed Result of the request to add syslog servers. success - successfully added the syslog server. initializing_failed - failed to establish a connection to the syslog server and retrying... existed - successfully connected to the syslog server and ready to send log messages. failed - failed to add the syslog server while creating a local syslog on Windows OS or due to other errors (further investigation is needed to determine the cause).
message	string	The message to describe information or error.

STATUS CODE - 400: Invalid request

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	

EXAMPLE:

```
{
 "err": "Request JSON body of logger configuration API is invalid"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 409: Conflicting request

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Logger configuration setting is processing, please wait for it to finish"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error while `<action>` configuration for syslog: `<reason>` action - can be `getting`, `setting`, `adding`, `deleting`, `verifying` reason - is the unhandled error reason

EXAMPLE:

```
{
 "err": "Error while <action> configuration for syslog: <reason>"
}
```

6.31 DELETE /admin/config/logger/syslog

Remove syslog servers or clear all syslog configurations

This API is used to request the removal of syslog servers or to clear all syslog configurations from the MD Core.

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
ONE:OF	object	
OPTION:1	object	
sources	array	
syslog	string	The address of the syslog server: `local"|"<protocol>://<address>:<port>`
OPTION:2	object	
delete_all	boolean	Set this to true if you wish to completely remove the syslog configuration.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
syslog_level	enum	ALLOWED: dump, debug, info, warning, error The current syslog output level.
sources	array	
syslog	string	The address of the syslog server: `local"|"<protocol>://<address>:<port>`
result	enum	ALLOWED: success, initializing_failed, existed, failed Result of the request to remove syslog servers. success - successfully removed the syslog server. not_existed - the specified address does not exist and cannot be removed failed - indicates that the syslog server could not be removed. This is a rare occurrence, potentially caused by an unhandled error. It should be reported for further investigation.
message	string	The message to describe the error.

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		

NAME	TYPE	DESCRIPTION
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 409: Conflicting request

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Logger configuration setting is processing, please wait for it to finish"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error while `<action>` configuration for syslog: `<reason>` action - can be `getting`, `setting`, `adding`, `deleting`, `verifying` reason - is the unhandled error reason

EXAMPLE:

```
{
 "err": "Error while <action> configuration for syslog: <reason>"
}
```

7. ENGINES

Enable/disable or pin/unpin the engines through API.

7.1 POST /admin/engine/{engineId}/pin

Pin engine to prevent auto-updates

Pin engines to prevent applying automatic updates on them. Manual updates still can be applied.

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*engineId	string		The unique engine identifier

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.
type	enum ALLOWED: engine, database		Pin engine or database to prevent applying automatic updates on it. (If the type is not defined both engine and database will be pinned.)

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
result	enum	ALLOWED: Engine is pinned, Database is pinned, Both engine and database are pinned

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

7.2 POST /admin/engine/{engineId}/unpin

Unpin engine to prevent auto-updates

Unpin engines so automatic updates will be applied on them.

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*engineId	string		The unique engine identifier

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.
type	enum ALLOWED: engine, database		Unpin engine or database to applying automatic updates on it. (If it is not defined both engine and database will be unpinned.)

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
result	enum	ALLOWED: Engine is unpinned, Database is unpinned, Both engine and database are unpinned

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

7.3 POST /admin/engine/{engineId}/enable

Enable engines

Enable to use the selected engine on the nodes.

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*engineId	string		The unique engine identifier

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
result	string	

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

7.4 POST /admin/engine/{engineId}/disable

Disable engines

Disable to use the selected engines on the nodes.

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*engineId	string		The unique engine identifier

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
result	string	

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

7.5 GET /admin/engine/{engineId}/config

Retrieve engine configuration

Retrieve a list of advanced configuration of an engine.

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*engineId	string		The unique engine identifier

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

7.6 PUT /admin/engine/{engineId}/config

Update engine configuration

Modifying advanced configurations of an engine.

REQUEST

PATH PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*engineId	string		The unique engine identifier

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

8. LICENSE

Activate the product or get licensing information. Will require admin apikey.

8.1 POST /admin/license/activation

Activate license online

This API initiates an online activation of the deployment. In case of error, check the application logs for more details.

REQUEST

REQUEST BODY - application/json

NAME	TYPE	DESCRIPTION
activationKey*	string	The license key provided by OPSWAT (either eval or production license).
quantity*	integer	The number of MetaDefender nodes attached to this instance.
comment	string	Metadata field allowing you to provide relevant identification information for this instance.

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
success	boolean	

STATUS CODE - 400: Bad Request (e.g. invalid header, apikey is missing or invalid).

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

8.2 GET /admin/license

Get current license information

Fetch all details about the licensing status of the product.

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Information about the licensed product (product type, number of activations, deploymentId, expiration date and days left)

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
days_left	integer	Number of days left before expiration
deployment	string	Unique identifier which is maps the current deployment to the activation
expiration	string	Expiration date in MM/DD/YYYY format.
licensed_engines	array	
max_node_count	string	Total number of deployed MetaDefender Nodes attached to this MetaDefender Core instance.
online_activated	boolean	Track online/offline activation mode
product_id	string	Official MetaDefender base SKU licensed.
product_name	string	Official MetaDefender base product name licensed.

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

8.3 POST /admin/license

Upload license key file

Uploading a license file to the Metadefender Core.

There are two ways two obtain a license key file:

- via https://portal.opswat.com/activation portal
- via activation server REST API: https://activation.dl.opswat.com/activation?key=%activation_key%&deployment=%deployment_unique_ID%&quantity=%quantity%

Deployment unique ID can be fetched via Get Current License Information API.

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Request processed successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
success	boolean	

STATUS CODE - 400: Invalid license file

RESPONSE MODEL - application/json

EXAMPLE:

```
{
 "err": "Invalid license"
}
```

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

9. MULTIPART

Upload a file for scan by multiple parts.

9.1 POST /file/multipart

Create multipart uploading session

Note: Only available starting with MetaDefender Core 5.11.0. The timeout value could be configurable

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*total-length	integer		Size of the whole submitted file in bytes
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.
filename	string		Specify name of file captured and displayed on corresponding MetaDefender Core scan result
user_agent	string		user_agent header used to identify (and limit) access to a particular rule. For rule selection, `rule` header should be used.
rule	string		Select rule for the analysis, if no header given the default rule will be selected (URL encoded UTF-8 string of rule name)
batch	string		Batch id to scan with, coming from `Initiate Batch` (If it is not given, it will be a single file scan.)

NAME	TYPE	EXAMPLE	DESCRIPTION
metadata	json		could be utilized for: Additional parameter for pre-defined post actions and external scanners (as a part of STDIN input). Customized macro variable for watermarking text (Proactive DLP engine feature). Additional context / verbose information for each file submission (appended into JSON response scan result).
engines-metadata	json		Since MetaDefender Core 5.0.0, preferred context / verbose information can be sent to the engines. Please see the below pages for the details: * [File Type engine](https://docs.opswat.com/mdcore/utilities-engines/supported-engines-metadata) (supported since Core 5.2.1) * [Archive engine](https://docs.opswat.com/mdcore/utilities-engines/supported-engines-metadata-header) (supported since Core 5.4.1) * [Deep CDR](https://docs.opswat.com/mdcore/deep-cdr/supported-engines-metadata-json) * [Proactive DLP](https://docs.opswat.com/mdcore/proactive-dlp/supported-engines-metadata-json)
callbackurl	uri		Client's URL where MetaDefender Core will notify scan result back to whenever scan is finished (webhooks model). * Format: \<protocol://\>\<ip | domain\>:\<port\>\</path\> * Example: http://10.0.1.100:8081/listenback * Supported protocol: HTTP / HTTPS * Method: POST
sanitizedurl	uri		Client's URL where MetaDefender Core will send the sanitized file as the body of a POST request back to whenever scan is finished (webhooks model). * Format: \<protocol://\>\<ip | domain\>:\<port\>\</path\> * Example: http://10.0.1.100:8081/listensanitizedfile * Supported protocol: HTTP / HTTPS * Method: POST * Mandatory headers: * `user-agent`: "OPSWAT MetaDefender Core V4" * `dataid`: "<data id of original file submission>" * `sanitization-result`: "Sanitized Successfully" or "Sanitized Partially" or "Sanitized File Removed" * `sanitized-file-hash`: "<SHA256 hash value of sanitized file>" * `Content-Disposition`: attachment; filename="<name of sanitized file>" * `Content-Length`: <actual size of sanitized file>
global-timeout	integer		This custom global timeout (in seconds) will override the global timeout predefined in corresponding workflow rule.

RESPONSE

STATUS CODE - 200: Successful creation of multipart uploading session

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
data_id*	string	Unique submission identifier. Use this value to reference the upload and submission.

EXAMPLE:

```
{
 "data_id": "62dffeaa728944adbf49eb090e4ece0f"
}
```

STATUS CODE - 400: Bad request.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
ONE:OF	object	
OPTION:1	object	
err	string	Invalid apikey given
OPTION:2	object	
err	string	Batch closed during file upload.
OPTION:3	object	
err	string	Can not scan in given batch.
OPTION:4	object	
err	string	No available multipart session is present for uploading.
OPTION:5	object	
err	string	Invalid engine metadata.
OPTION:6	object	
err	string	Invalid callback URL.
OPTION:7	object	
err	string	Null metadata is not allowed.
OPTION:8	object	
err	string	Invalid metadata format.

STATUS CODE - 403: Unavailable workflow rule.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Request is refused because of unavailable workflow rule.

STATUS CODE - 415: API not supported.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	

EXAMPLE:

```
{
 "err": "API is not supported"
}
```

STATUS CODE - 500: Internal Server Error.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
ONE:OF	object	
OPTION:1	object	
err	string	License is expired
OPTION:2	object	
err	string	No available rule is present for scanning.
OPTION:3	object	
err	string	Failed to request scan. Try again later.
OPTION:4	object	
err	string	Failed to request scan. File size exceeded the maximum size permitted by your configuration.

STATUS CODE - 503: Server is too busy, scan queue is full. Try again later.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	

EXAMPLE:

```
{
 "err": "Server is too busy. Try again later."
}
```

9.2 GET /file/multipart/{data_id}

Fetch uploading status

Fetch status of uploading a file by multiple parts

Note:

- Sticky session must be provided if this API calls to a Load Balancer that is in front of multiple MetaDefender Core instances. The session is responded by Load Balancer from POST /file/multipart API.

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Uploading status fetched successfully

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
data_id	string	Data identifier of the requested file.
status	string	Uploading status, can be one of `completed`, `uploading`, `timed out`, or `canceled`.
total_length	number	The size of whole file in bytes.
ranges_received	string	A list of parts that have been received successfully so far, in format of `<start_1>-<end_1>, <start_2>-<end_2>`.

EXAMPLE:

```
{
```

```
 "data_id": "62dffeaa728944adbf49eb090e4ece0f",
 "status": "uploading",
 "total_length": 10485760,
 "ranges_received": "0-2000000, 4000000-10485759"
}
```


STATUS CODE - 400: Bad request.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Cannot get multipart status from another instance.

STATUS CODE - 404: Not found.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Multipart uploading session not found.

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:


```
{
 "err": "Access denied"
}
```


STATUS CODE - 500: Internal Server Error.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Failed to request scan. Try again later.

9.3 POST /file/multipart/{data_id}

Upload individual part of file

Upload an individual part of file to the created multipart uploading session

Note:

- Invidual parts can be uploaded simultaneously.
- When all parts are fully received, MetaDefender Core will automatically combine them into a single file and start processing it.
- Clients are free to upload broken parts again via this API if the network or service is interrupted.
- Sticky session must be provided if the part is uploaded to a Load Balancer that is in front of multiple MetaDefender Core instances. The session is responded by Load Balancer from POST /file/multipart API.

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*content-range	string		Offset and size of the uploaded part, in format of `offset-size/total_size`, in which * `offset`: the offset of uploaded part in whole file. * `size`: the size of uploaded part in bytes. * `total_size`: the size of whole file in bytes. For example: the submitted file is of 10,485,760 bytes (10MBs) and divided into 3 parts. Parts are uploaded with ranges as following: * Part 01: 0-2000000/10485760 * Part 02: 2000000-2000000/10485760 * Part 03: 4000000-6485760/10485760 _**Note**_: An uploaded part can overlap its next and previous.
*content-length	integer		Size of request body in bytes
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Part uploaded successfuly

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
ranges_received*	string	A list of parts that have been received successfully so far, in format of `<start_1>-<end_1>, <start_2>-<end_2>`.

EXAMPLE:

```
{
 "ranges_received": "0-2000000, 4000000-10485759"
}
```

STATUS CODE - 400: Bad request.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
ONE:OF	object	
OPTION:1	object	
err	string	Invalid apikey given
OPTION:2	object	
err	string	Invalid content type.
OPTION:3	object	
err	string	Cannot upload part from another instance.

STATUS CODE - 404: Not found.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Multipart uploading session not found.

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
```

```
}
```

STATUS CODE - 409: Conflict.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
ONE:OF	object	
OPTION:1	object	
err	string	Multipart uploading session timed out.
OPTION:2	object	
err	string	Multipart uploading session has already finished.

STATUS CODE - 416: Range not satisfiable.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
ONE:OF	object	
OPTION:1	object	
err	string	Invalid content range.
OPTION:2	object	
err	string	The part is duplicated.

STATUS CODE - 422: Cancelled.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Multipart uploading session cancelled.

STATUS CODE - 500: Internal Server Error.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Failed to request scan. Try again later.

9.4 DELETE /file/multipart/{data_id}

Abort multipart uploading

Abort uploading a file by multiple parts

Note:

- Sticky session must be provided if this API calls to a Load Balancer that is in front of multiple MetaDefender Core instances. The session is responded by Load Balancer from POST /file/multipart API.

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: Uploading session aborted successfully

RESPONSE MODEL - application/json

EXAMPLE:

```
{
 "62dffeaa728944adbf49eb090e4ece0f": "cancelled"
}
```

STATUS CODE - 400: Bad request.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
ONE:OF	object	
OPTION:1	object	
err	string	Invalid apikey given
OPTION:2	object	

NAME	TYPE	DESCRIPTION
err	string	Cannot abort from another instance.

STATUS CODE - 404: Not found.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Multipart uploading session not found.

STATUS CODE - 409: Conflict.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
ONE:OF	object	
OPTION:1	object	
err	string	Multipart uploading session timed out.
OPTION:2	object	
err	string	Multipart uploading session has already finished.
OPTION:3	object	
err	string	Multipart uploading session already cancelled.

STATUS CODE - 500: Internal Server Error.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Failed to request scan. Try again later.

10. STATS

Health check and statistics about MetaDefender Core usage.

10.1 GET /version

Get Product Version

Fetch details about the product version.

REQUEST

No request parameters

RESPONSE

STATUS CODE - 200: Provides productId and version information.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
product_id	string	Product identifier.
version	string	The product version currently used.

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

10.2 GET /stat/engines

Engines Status

The response is an array of engines with database information.

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: An array with all the engines and their details.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
ARRAY OF OBJECT WITH BELOW STRUCTURE		
active	boolean	If used by at least one engine
def_time	string	The database definition time for this engine
download_progress	integer	The percentage progress of download
download_time	string	When this engine downloaded from the update server.
eng_id	string	Engine internal ID
eng_name	string	Engine name
eng_type	string	Engine type in human readable form
eng_ver	string	Engine's version (format differs from one engine to another).
engine_type	enum	**ALLOWED:** av, archive, filetype Engine's type: * av * archive * filetype

NAME	TYPE	DESCRIPTION
state	enum	**ALLOWED:** downloading, downloaded, staging, production, removed, temporary failed, permanently failed, content invalid, download failed Status of the engine: * downloading * downloaded * staging * production * removed * temporary failed * permanently failed * content invalid * download failed

10.3 GET /stat/nodes

Core Status

Get a list of connected Core and status.

REQUEST

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.

RESPONSE

STATUS CODE - 200: An array with all the engines and their details.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
ARRAY OF OBJECT WITH BELOW STRUCTURE		
external_nodes_allowed	boolean	Configuration option if additional external nodes could be connected.
max_node_count	integer	Remaining available slots to connect to this instance.
statuses	array	
address	string	The location of the Core. If local, is empty string.
available_mem	integer	The number of available RAM in this system.

NAME	TYPE	DESCRIPTION
cpu_cores	integer	The number of CPU Cores allocated to this Core instance.
engines	array	
active	boolean	If used by at least one engine
db_ver	string	The database version for this engine
def_time	string	The database definition time for this engine
eng_name	string	Engine name
eng_ver	string	Engine's version (format differs from one engine to another).
engine_type	enum	**ALLOWED:** av, archive, filetype Engine's type: * av * archive * filetype
id	string	Engine internal ID
free_disk_space	integer	Reported available disk on Core instance (in bytes).
id	string	Core identfier
issues	array	
description	string	Text detailing the issue.
severity	string	How important is the reported issue.
load	integer	Current CPU utilization on Core instance (in percentage).
os	string	Current used OS
scan_queue	integer	Current load on the Core instance, how many files are in the queue
scan_queue_details	object	
total_scan_queue	integer	Total scan queue
file_slots	integer	The number of slots is taken by files scan from REST API
extracted_file_slots	integer	The number of slots is taken by extracted files
available_slots	integer	The number of slots is available
total_disk_space	integer	The amount of disk space is allocated on Core instance (in Byte).
total_mem	integer	How much memory is allocated on Core instance (in MB).
total_scan_queue	integer	The maximum queue size is allowed.
uptime	integer	How long this Core is in operation (in second).
version	string	Product version

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		

NAME	TYPE	DESCRIPTION
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

10.4 GET /stat/activeperformance

Active performance

Get active performance statistics.

REQUEST

QUERY PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
hours_range	integer		Collect data from last hours

HEADER PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
*apikey	string		Generated `session_id` from [Login](/docs/mdcore/metadefender-core/ref#userlogin) call can be used as an `apikey` for API calls that require authentication.
rule_name	string		The name of workflow rule to filter.

RESPONSE

STATUS CODE - 200: Active performance report on MetaDefender Core

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
active_performance	object	
failed_results	integer	Number of failed processing results.
hash_time	object	
avg	integer	Average hash calculating time.
max	integer	Maximum hash calculating time.
min	integer	Minimum hash calculating time.
processing_time	object	
avg	integer	Average overall processing time.
max	integer	Maximum overall processing time.
min	integer	Minimum overall processing time.
queue_time	object	
avg	integer	Average queue time.
max	integer	Maximum queue time.
min	integer	Minimum queue time.
success_results	integer	Total number of successfully-processed objects (excluding failed or not-scanned results)
total_filesize	integer	Total processed file size in bytes.
total_requests	integer	Total number of file submissions.
wait_time	object	
avg	integer	Average waiting time.
max	integer	Maximum waiting time.
min	integer	Minimum waiting time.
from_time	integer	The data is gathered in this recent milliseconds.

STATUS CODE - 403: Invalid user information or Not Allowed

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 405: The user has no rights for this operation.

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "Access denied"
}
```

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	string	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

10.5 GET /readyz

Get health check status

Fetch current health check status of MetaDefender Core server.

REQUEST

QUERY PARAMETERS

NAME	TYPE	EXAMPLE	DESCRIPTION
verbose	boolean		Show detailed result of health check

RESPONSE

STATUS CODE - 200: Service is currently healthy

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
database	`object`	
status	`boolean`	Connection status
license	`object`	
status	`enum`	**ALLOWED:** `expired, invalid, ok` License status
scan_queue	`object`	
max_number_in_queue	`integer`	Maximum slots in scan queue
number_in_queue	`integer`	Current queue status
threshold	`integer`	User-defined threshold
status	`boolean`	status
number_active_av_engines	`object`	
threshold	`integer`	User-defined threshold
status	`boolean`	Status
engines	`array`	
engine_id	`string`	Engine ID
required	`boolean`	This engine is mandatorily active
status	`boolean`	Status

STATUS CODE - 500: Unexpected event on server

RESPONSE MODEL - application/json

NAME	TYPE	DESCRIPTION
OBJECT WITH BELOW STRUCTURE		
err	`string`	Error reason

EXAMPLE:

```
{
 "err": "<error message>"
}
```

STATUS CODE - 503: Service is currently unhealthy

OPSWAT.

