MultiScanner Documentation

Release 1.0.0

MITRE

Jan 29, 2019

Contents

1 Overview **1**

1.1 Key Capabilities 1

2 Architecture **3**

2.1 High-level Architecture 3

2.2 Complete Workflow 4

2.3 Analysis 5

2.4 Analytics 5

2.5 Reporting 6

3 Use Cases **7**

3.1 Scalable Malware Analysis 7

3.2 Manual Malware Analysis 7

3.3 Analysis-Oriented Malware Repository 7

3.4 Data Enrichment 7

3.5 Data Analytics 8

4 Installation **9**

4.1 System Requirements 9

4.2 Installing Ansible 9

4.3 Installing Analytic Machines 10

4.4 Installing Elasticsearch 10

4.5 Configuration 10

4.6 Standalone Docker Installation 11

5 Using MultiScanner **13**

5.1 Web UI 13

5.2 Python API 20

5.3 RESTful API 20

5.4 Analysis Modules 21

5.5 Analytics 27

6 Custom Development **29**

6.1 Developing an Analysis Module 29

6.2 Developing an Analytic 31

6.3 Writing a Storage Module 31

6.4 Example Module 32

7 Testing **33**

7.1 Front-end Tests with Selenium 33

8 Presentations **35**

CHAPTER 1

Overview

MultiScanner is a distributed file analysis framework that assists the user in evaluating a set of files by automatically running a suite of tools and aggregating the output. Tools can be custom Python scripts, web APIs, software running on another machine, etc. Tools are incorporated by creating modules that run in the MultiScanner framework.

By design, modules can be quickly written and easily incorporated into the framework. While current modules are related to malware analysis, the MultiScanner framework is not limited in scope. For descriptions of current modules, see *Analysis Modules*.

MultiScanner supports a distributed workflow for sample storage, analysis, and report viewing. This functionality includes a web interface, a REST API, a distributed file system (GlusterFS), distributed report storage / searching (Elasticsearch), and distributed task management (Celery / RabbitMQ). See the *Complete Workflow* section for details.

MultiScanner is available as open source in GitHub.

1.1 Key Capabilities

As illustrated in the diagram below, MultiScanner helps the malware analyst, enabling analysis with automated tools and manual tools, providing integration and scaling capabilities, and corrolating analysis results. It allows analysts to associate metadata with samples and also allows integration of data from external sources. MultiScanner is particularly useful because data is linked across tools and samples, allowing pivoting and analytics.

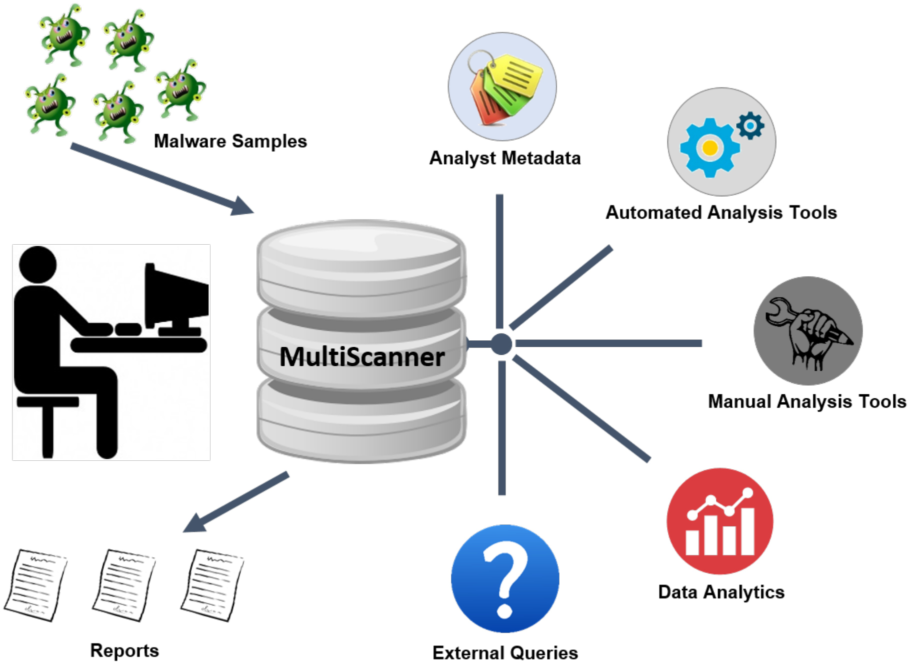


Fig. 1: Key Capabilities

CHAPTER 2

Architecture

2.1 High-level Architecture

There are seven primary components of the MultiScanner architecture, as described below and illustrated in the associated diagram.

Web Frontend

The web application runs on Flask, uses Bootstrap and jQuery, and is served via Apache. It is essentially an aesthetic wrapper around the REST API. All data and services provided are also available by querying the REST API.

REST API

The REST API is also powered by Flask and served via Apache. It has an underlying PostgreSQL database to facilitate task tracking. Additionally, it acts as a gateway to the backend Elasticsearch document store. Searches entered into the web UI will be routed through the REST API and passed to the Elasticsearch cluster. This abstracts the complexity of querying Elasticsearch and gives the user a simple web interface to work with.

Task Queue

We use Celery as our distributed task queue.

Task Tracking

PostgreSQL is our task management database. It is here that we keep track of scan times, samples, and the status of tasks (pending, complete, failed).

Distributed File System

GlusterFS is our distributed file system. Each component that needs access to the raw samples mounts the share via FUSE. We selected GlusterFS because it is more performant in our use case – storing a large number of small samples – than a technology like HDFS would be.

Worker Nodes

The worker nodes are Celery clients running the MultiScanner Python application. Additionally, we implemented some batching within Celery to improve the performance of our worker nodes (which operate better at scale).

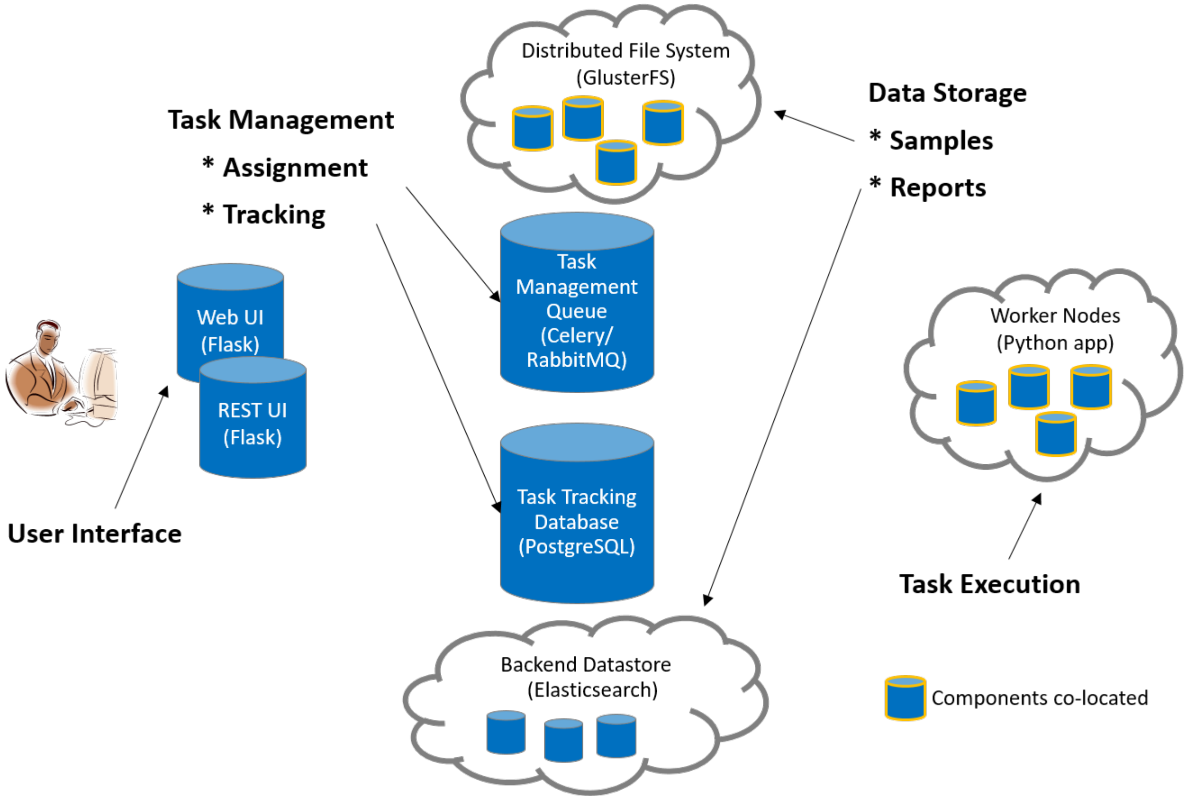


Fig. 1: MultiScanner Architecture

A worker node will wait until there are 100 samples in its queue or 60 seconds have passed (whichever happens first) before kicking off its scan (these values are configurable). All worker nodes have the GlusterFS mounted, which gives access to the samples for scanning. In our setup, we co-locate the worker nodes with the GlusterFS nodes in order to reduce the network load of workers pulling samples from GlusterFS.

Report Storage

We use Elasticsearch to store the results of our file scans. This is where the true power of this system lies. Elasticsearch allows for performant, full text searching across all our reports and modules. This allows fast access to interesting details from your malware analysis tools, pivoting between samples, and powerful analytics on report output.

2.2 Complete Workflow

Each step of the MultiScanner workflow is described below the diagram.

1. The user submits a sample file through the Web UI (or REST API)
2. The Web UI (or REST API):
   (a) Stores the file in the distributed file system (GlusterFS)
   (b) Places the task on the task queue (Celery)
   (c) Adds an entry to the task management database (PostgreSQL)
3. A worker node:
   (a) Pulls the task from the Celery task queue

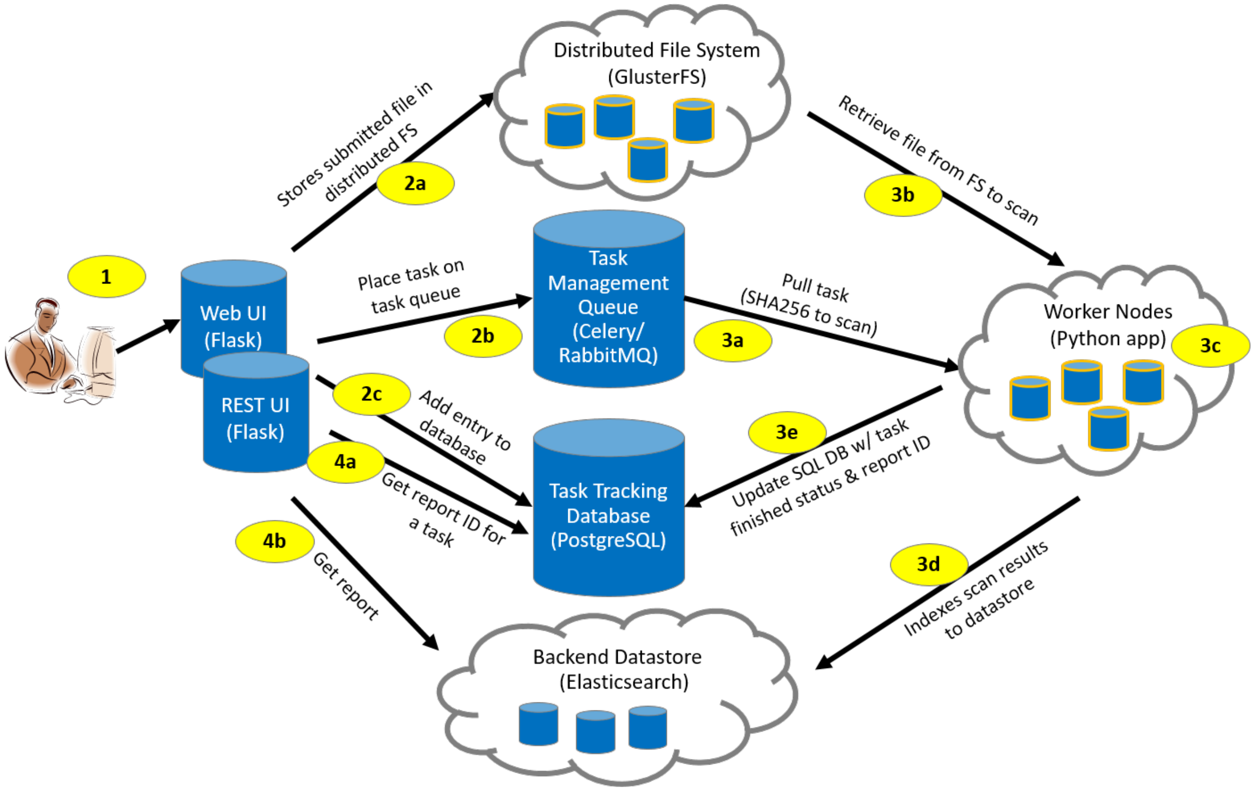

Distributed File System
(GlusterFS)
Stores submitted file in distributed FS
2a
Retrieve file from FS to scan
3b
1
Web UI
(Flask)
Place task on task queue
2b
Task Management Queue (Celery/ RabbitMQ)
Pull task (SHA256 to scan)
3a
Worker Nodes
(Python app)
3c
REST UI
(Flask)
2c
Add entry to database
4a
Get report ID for a task
Task Tracking Database (PostgreSQL)
3e
Update SQL DB w/ task finished status & report ID
4b
Get report
3d
Indexes scan results to datastore
Backend Datastore
(Elasticsearch)

Fig. 2: MultiScanner Workflow

   (b) Retrieves the corresponding sample file from the GlusterFS via its SHA256 value
   (c) Analyzes the file
   (d) Generates a JSON blob and indexes it into Elasticsearch
   (e) Updates the task management database with the task status ("complete")

4. The Web UI (or REST API):
   (a) Gets report ID associated with the Task ID
   (b) Pulls analysis report from the Elasticsearch datastore

2.3 Analysis

Analysis tools are integrated into MultiScanner via modules running in the MultiScanner framework. Tools can be custom built Python scripts, web APIs, or software applications running on different machines. Catagories of existing modules include AV scanning, sandbox detonation, metadata extraction, and signature scanning. Modules can be enabled/disabled via a configuration file. Details are provided in the *Analysis Modules* section.

2.4 Analytics

Enabling analytics and advanced queries is the primary advantage of running several tools against a sample, extracting as much information as possible, and storing the output in a common datastore. For example, the following types of analytics and queries are possible:

- cluster samples
- outlier samples
- samples for deep-dive analysis
- gaps in current toolset
- machine learning analytics on tool outputs

2.5 Reporting

Analysis data captured or generated by MultiScanner is accessible in three ways:

- MultiScanner Web User Interface – Content in the Elasticsearch database is viewable through the Web UI. See *Web UI* section for details.
- MultiScanner Reports – MultiScanner reports reflect the content of the MultiScanner database and are provided in raw JSON and PDF formats. These reports capture all content associated with a sample.
- STIX-based reports *will soon be* available in multiple formats: JSON, PDF, HTML, and text.

CHAPTER 3

Use Cases

MultiScanner is intended to be used by security operations centers, malware analysis centers, and other organizations involved with cyber threat intelligence (CTI) sharing. This section outlines associated use cases.

3.1 Scalable Malware Analysis

Every component of MultiScanner is designed with scaling in mind, enabling analysis of large malware data sets.

Note that scaling required for external analysis tools such as Cuckoo Sandbox is beyond the scope of MultiScanner code, as is auto-scaling (e.g., scaling required to auto-provision virtual machines). New worker nodes must be deployed manually and added to the Ansible playbook for proper configuration (see *Installing Analytic Machines*).

3.2 Manual Malware Analysis

MultiScanner can support manual malware analysis via modules that enable analyst interaction. For example, a module could be developed to allow an analyst to interact with IDA Pro to disassemble and analyze a binary file.

3.3 Analysis-Oriented Malware Repository

MultiScanner enables long term storage of binaries and metadata associated with malware analysis.

3.4 Data Enrichment

Malware analysis results can be enriched in support of CTI sharing objectives. In addition to data derived from analysis of submitted samples, other CTI sources can be integrated with MultiScanner, such as TAXII feeds, commercial CTI providers (FireEye, Proofpoint, CrowdStrike, etc.), and closed-source CTI providers.

3.5 Data Analytics

Data analytics can be performed on malware samples either by interacting with the Elasticsearch datastore or via the Web/REST UI.

CHAPTER 4

Installation

Installation information for the different components of MultiScanner is provided below. To get an idea of how the system works without going through the full process of setting up the distributed architecture, refer to the section on *Standalone Docker Installation*.

The Docker standalone system is less scalable, robust, and feature-rich, but it enables easy stand up the web UI, the REST API, and an Elasticsearch node, allowing users to quickly see how the system works. The standalone container is intended as an introduction to the system and its capabilities, but is not designed for operational use.

4.1 System Requirements

Python 3.6 is recommended. Compatibility with Python 2.7+ and 3.4+ is supported but not thoroughly maintained and tested. Please submit an issue or a pull request fixing any issues found with other versions of Python.

An installer script is included in the project (install.sh), which installs the prerequisites on most systems.

Currently, MultiScanner is deployed with Ansible, and we're working to support distributed architecture deployment via Docker.

4.2 Installing Ansible

The installer script should install the required Python packages for users of RedHat- or Debian-based Linux distributions. Users of other distributions should refer to requirements.txt.

MultiScanner requires a configuration file to run. After cloning the repository, generate the MultiScanner default configuration by running `python multiscanner.py init`. The command can be used to rewrite the configuration file to its default state or, if new modules have been written, to add their configuration details to the configuration file.

4.3 Installing Analytic Machines

Default modules have the option of being run locally or via SSH. The development team runs MultiScanner on a Linux host and hosts the majority of analytical tools on a separate Windows machine. The SSH server used in this environment is freeSSHd.

A network share accessible to both the MultiScanner and the analytic machines is required for the multi-machine setup. Once configured, the network share path must be identified in the configuration file, config.ini (an example can be found here). To do this, set the `copyfilesto` option under `[main]` to be the mount point on the system running MultiScanner. Modules can have a `replacement path` option, which is the network share mount point on the analytic machine.

4.4 Installing Elasticsearch

Starting with Elasticsearch 2.x, field names can no longer contain '.' (dot) characters. Thus, the MultiScanner elasticsearch_storage module adds a pipeline called "dedot" with a processor to replace dots in field names with underscores.

Add the following to the elasticsearch.yml configuration file for the dedot processor to work:

```
script.painless.regex.enabled: true
```

To use the Multiscanner web UI, also add the following:

```
http.cors.enabled: true
http.cors.allow-origin: "<yourOrigin>"
```

4.5 Configuration

MultiScanner and its modules are configured within the configuration file, config.ini. An example can be found here.

The following parameters configure MultiScanner itself, and go in the `[main]` section of the config file.

Parameter	Description
copyfilesto	This is where the script will copy each file that is to be scanned. This can be removed or set to False to disable this feature.
group-types	This is the type of analytics to group into sections for the report. This can be removed or set to False to disable this feature.
storage-config	Path to the storage config file.
api-config	Path to the API config file.
web-config	Path to the Web UI config file.

Modules are intended to be quickly written and incorporated into the framework. Note that:

- A finished module must be placed in the modules folder before it can be used.
- The configuration file does not need to be manually updated.
- Modules are configured within the same configuration file, config.ini.

See *Analysis Modules* for information about all current modules and their configuration parameters.

4.6 Standalone Docker Installation

To introduce new users to the power of the MultiScanner framework, web UI, and REST API, we have built a standalone docker application that is simple to run in new environments. Simply clone the top level directory and run:

```
$ docker-compose up
```

This will build the three necessary containers (one for the web application, one for the REST API, and one for the Elasticsearch backend).

Running this command will generate a lot of output and take some time. The system is not ready until you see the following output in your terminal:

```
api_1       |  * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```

Now you can go to the web interface at `http://localhost:8000`.

Note: We are assuming that you are already running latest version of docker and have the latest version of docker-compose installed on your machine. Guides on how to do that are here. and here.

Note: Since this docker container runs two web applications and an ElasticSearch node, there is a fairly high requirement for RAM / computing power. We'd recommend running this on a machine with at least 4GB of RAM.

Warning: THIS CONTAINER IS NOT DESIGNED FOR PRODUCTION USE. This is simply a primer for using MultiScanner's web interface. Users should not run this in production or at scale. The MultiScanner framework is highly scalable and distributed, but that requires a full install. Currently, we support installing the distributed system via ansible. More information about that process can be found in this repo.

Note: This container will only be reachable / functioning on localhost.

Note: Additionally, if you are installing this system behind a proxy, you must edit the docker-compose.yml file in four places. First, uncomment lines 18-20 and lines 35-37. Next, uncomment lines 25-28 and set the correct proxy variables there. Finally, do the same thing in lines 42-45. The docker-compose.yml file has comments to make clear where to make these changes.

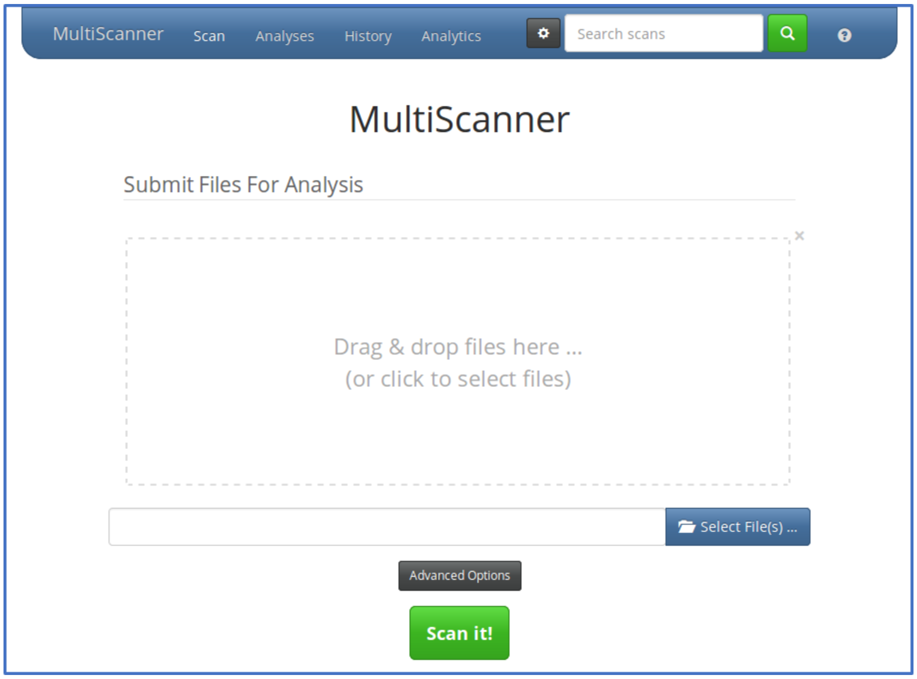
CHAPTER 5

Using MultiScanner

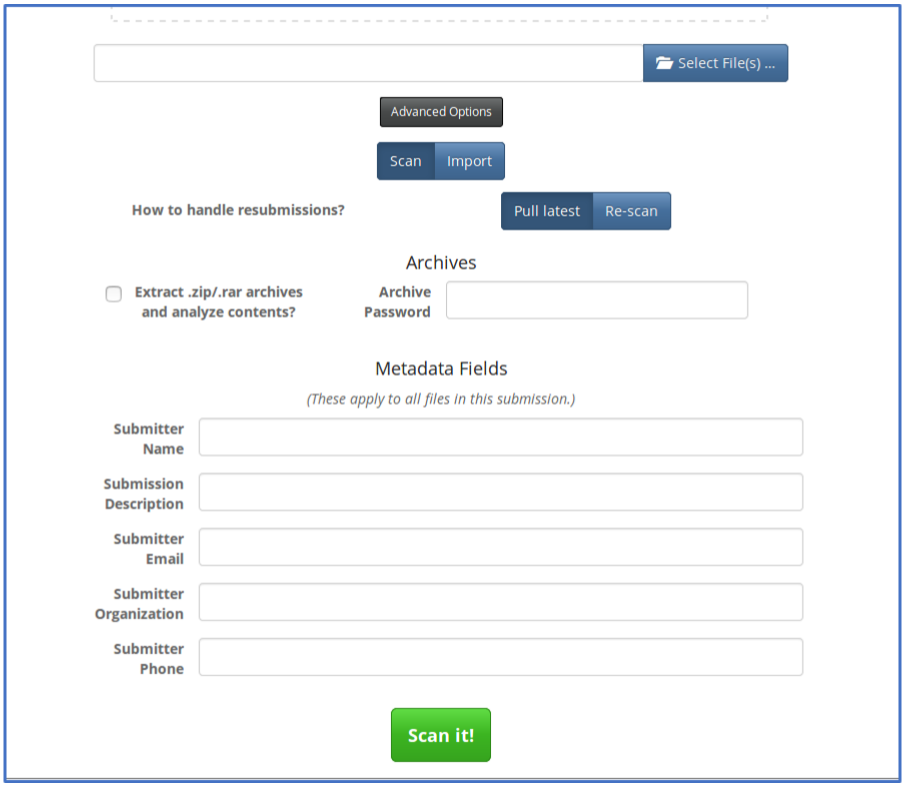
5.1 Web UI

5.1.1 Submitting Files for Analysis

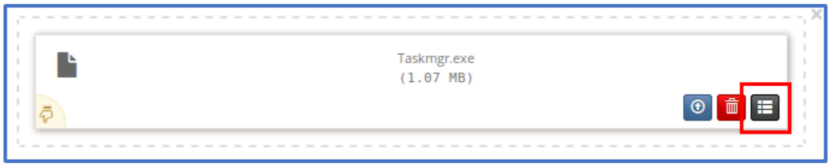
When you visit MultiScanner's web interface in a web browser, you'll be greeted by the file submission page. Drag files onto the large drop area in the middle of the page or click it or the "Select File(s)..." button to select one or more files to be uploaded and analyzed.



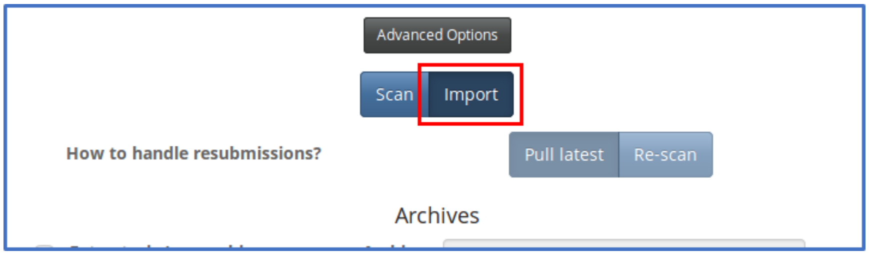
Click on the "Advanced Options" button to change default options and set metadata fields to be added to the scan results.



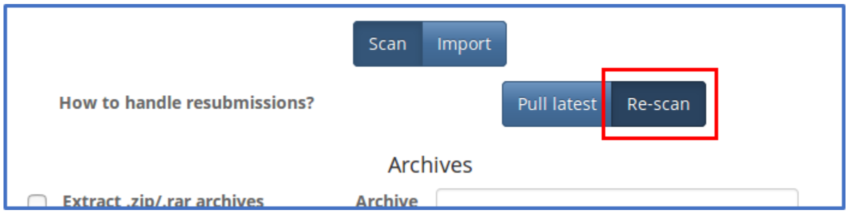
Metadata fields can be added or removed by editing web_config.ini. Metadata field values can be set for individual files by clicking the small black button below and to the right of that filename in the staging area.



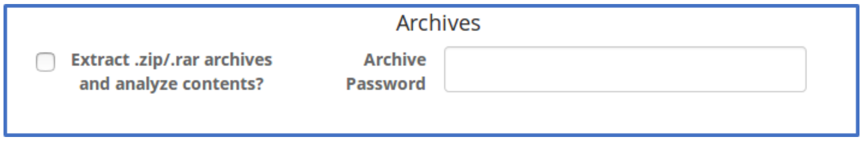
Change from "Scan" to "Import" to import JSON analysis reports into MultiScanner. This is intended only to be used with the JSON reports you can download from a report page in MultiScanner.



By default, if you resubmit a sample that has already been submitted, MultiScanner will pull the latest report of that sample. If you want MultiScanner to re-scan the sample, set that option in Advanced Options.



If you have a directory of samples you wish to scan at once, we recommend zipping them and uploading the archive with the option to extract archives enabled. You can also specify a password, if the archive file is password- protected. Alternatively you can use the REST API for bulk uploads.

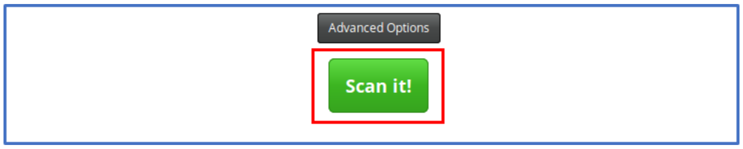


Archives

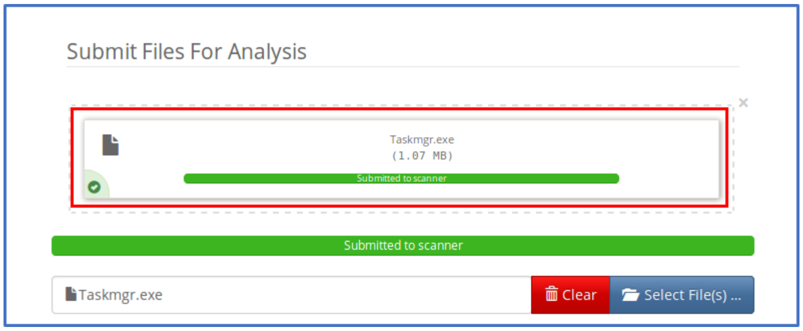
□ Extract .zip/.rar archives and analyze contents?

Archive Password

Click the "Scan it!" button to submit the sample to MultiScanner.



The progress bars that appear in the file staging area do not indicate the progress of the scan; a full bar merely indicates that the file has been uploaded to MultiScanner. Click on the file to go to its report page.



Submit Files For Analysis

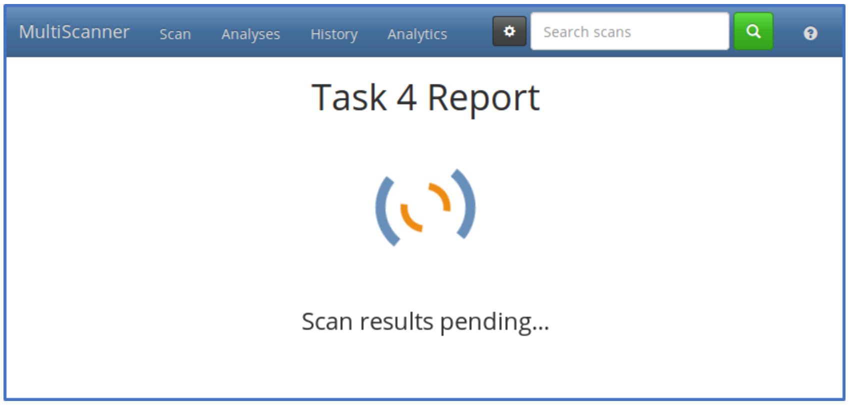
Taskmgr.exe
(1.07 MB)

Submitted to scanner

Submitted to scanner

Taskmgr.exe

If the analysis has not completed yet, you'll see a "Pending" message.

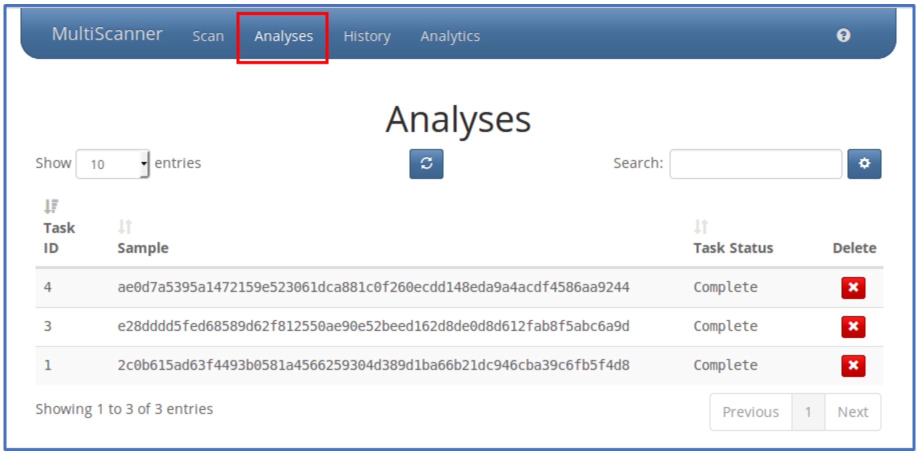


Task 4 Report

Scan results pending...

5.1.2 Viewing Analyses

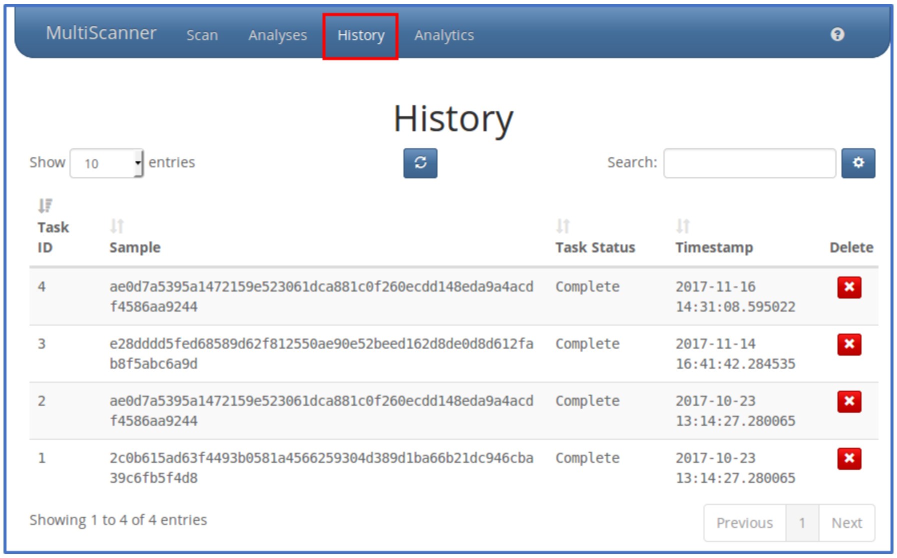
Reports can be listed and searched in two different ways. The Analyses page lists the most recent report per sample.



Task ID	Sample	Task Status	Delete
4	ae0d7a5395a1472159e523061dca881c0f260ecdd148eda9a4acdf4586aa9244	Complete	✖
3	e28dddd5fed68589d62f812550ae90e52beed162d8de0d8d612fab8f5abc6a9d	Complete	✖
1	2c0b615ad63f4493b0581a4566259304d389d1ba66b21dc946cba39c6fb5f4d8	Complete	✖

Showing 1 to 3 of 3 entries

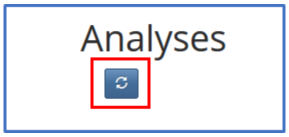
The History page lists every report of each sample. So if a file is scanned multiple times, it will only show up once on the Analyses page, but all of the reports will show up on the History page.



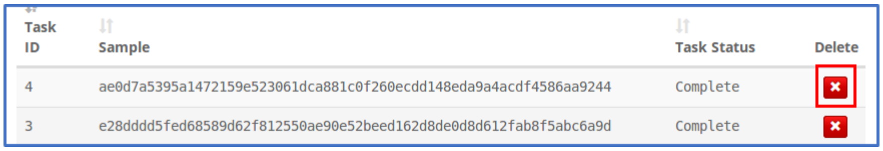
Task ID	Sample	Task Status	Timestamp	Delete
4	ae0d7a5395a1472159e523061dca881c0f260ecdd148eda9a4acdf4586aa9244	Complete	2017-11-16 14:31:08.595022	✖
3	e28dddd5fed68589d62f812550ae90e52beed162d8de0d8d612fab8f5abc6a9d	Complete	2017-11-14 16:41:42.284535	✖
2	ae0d7a5395a1472159e523061dca881c0f260ecdd148eda9a4acdf4586aa9244	Complete	2017-10-23 13:14:27.280065	✖
1	2c0b615ad63f4493b0581a4566259304d389d1ba66b21dc946cba39c6fb5f4d8	Complete	2017-10-23 13:14:27.280065	✖

Showing 1 to 4 of 4 entries

Both pages display the list of reports and allow you to search them. Click the blue button in the middle to refresh the list of reports.



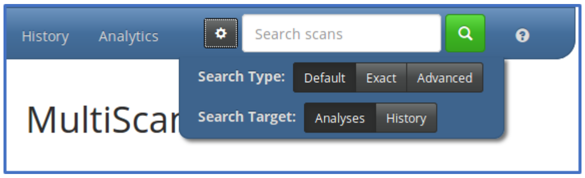
Click on a row in the list to go to that report, and click the red "X" button to delete that report from MultiScanner's Elasticsearch database.



Task ID	Sample	Task Status	Delete
4	ae0d7a5395a1472159e523061dca881c0f260ecdd148eda9a4acdf4586aa9244	Complete	✖
3	e28dddd5fed68589d62f812550ae90e52beed162d8de0d8d612fab8f5abc6a9d	Complete	✖

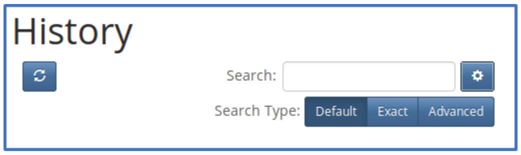
5.1.3 Searching

Reports can be searched from any page, with a few options. You can search Analyses to get the most recent scan per file, or search History to get all scans recorded for each file.



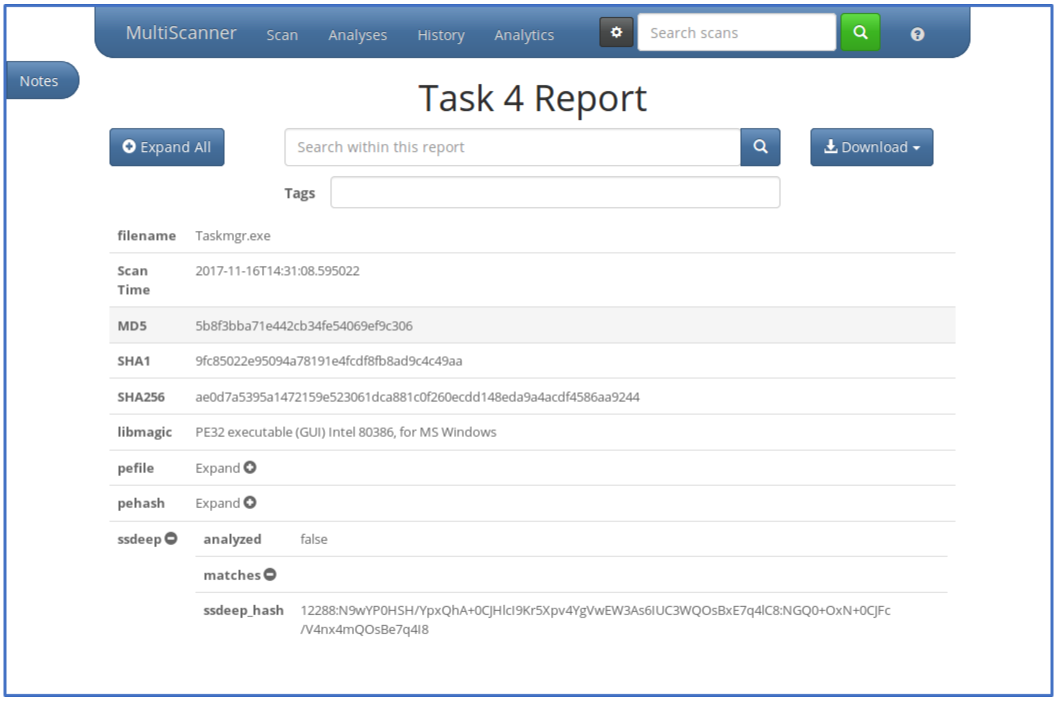
- Use the "Default" search type to have wildcards automatically appended to the beginning and end of your search term.
- Use the "Exact" search type to search automatically append quotes and search for the exact phrase.
- Use the "Advanced" search type to search with the full power of Lucene query string syntax. Nothing will be automatically appended and you will need to escape any reserved characters yourself.

When you click on a search result, the search term will be highlighted on the Report page and the report will be expanded and automatically scrolled to the first match.



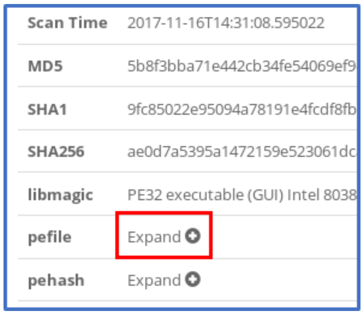
5.1.4 Viewing Reports

Each report page displays the results of a single analysis.



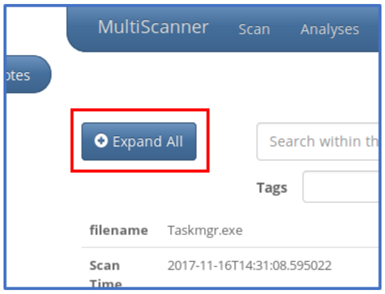
filename	Taskmgr.exe
Scan Time	2017-11-16T14:31:08.595022
MD5	5b8f3bba71e442cb34fe54069ef9c306
SHA1	9fc85022e95094a78191e4fcdf8fb8ad9c4c49aa
SHA256	ae0d7a5395a1472159e523061dca881c0f260ecdd148eda9a4acdf4586aa9244
libmagic	PE32 executable (GUI) Intel 80386, for MS Windows
pefile	Expand ⊕
pehash	Expand ⊕
ssdeep ⊖	analyzed: false matches ⊖ ssdeep_hash: 12288:N9wYP0HSH/YpxQhA+0CJHlcI9Kr5Xpv4YgVwEW3As6IUC3WQOsBxE7q4lC8:NGQ0+OxN+0CJFc/V4nx4mQOsBe7q4l8

Some rows in the report can be expanded or collapsed to reveal more data by clicking on the row header or the "Expand" button. Shift-clicking will also expand or collapse all of it's child rows.



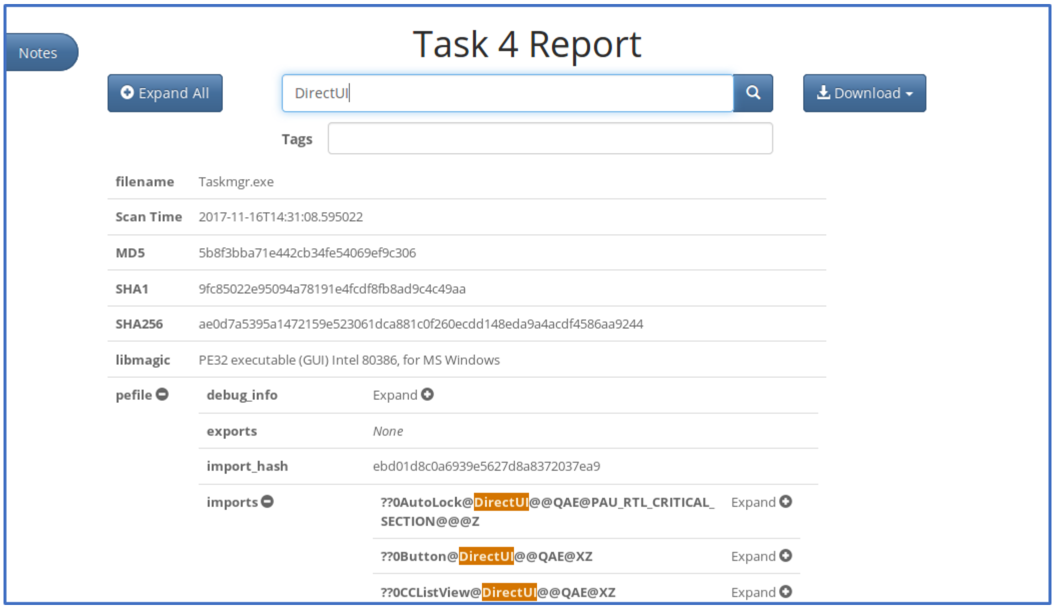
Scan Time	2017-11-16T14:31:08.595022
MD5	5b8f3bba71e442cb34fe54069ef9
SHA1	9fc85022e95094a78191e4fcdf8fb
SHA256	ae0d7a5395a1472159e523061dc
libmagic	PE32 executable (GUI) Intel 8038
pefile	Expand ⊕
pehash	Expand ⊕

The "Expand All" button will expand all rows at once. If they are all expanded, this will turn into a "Collapse All" button that will collapse them all again.



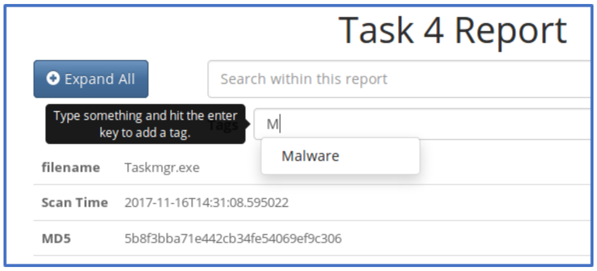
filename	Taskmgr.exe
Scan Time	2017-11-16T14:31:08.595022

As reports can contain a great deal of content, you can search the report to find the exact data you are looking for with the search field located under the report title. The search term, if found, will be highlighted, the matching fields will be expanded, and the page automatically scrolled to the first match.



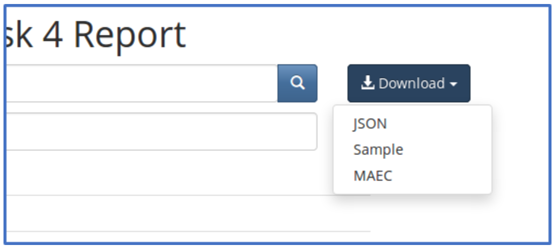
filename	Taskmgr.exe	
Scan Time	2017-11-16T14:31:08.595022	
MD5	5b8f3bba71e442cb34fe54069ef9c306	
SHA1	9fc85022e95094a78191e4fcdf8fb8ad9c4c49aa	
SHA256	ae0d7a5395a1472159e523061dca881c0f260ecdd148eda9a4acdf4586aa9244	
libmagic	PE32 executable (GUI) Intel 80386, for MS Windows	
pefile ⊖	debug_info	Expand ⊕
	exports	*None*
	import_hash	ebd01d8c0a6939e5627d8a8372037ea9
	imports ⊖	??0AutoLock@DirectUI@@QAE@PAU_RTL_CRITICAL_SECTION@@@Z Expand ⊕
		??0Button@DirectUI@@QAE@XZ Expand ⊕
		??0CCListView@DirectUI@@QAE@XZ Expand ⊕

Reports can be tagged by entering text in the Tags input box and hitting the enter key. As you type, a dropdown will appear with suggestions from the tags already in the system. It will pull the list of tags from existing reports, but a pre-populated list of tags can also be provided in web_config.ini when the web interface is set up.

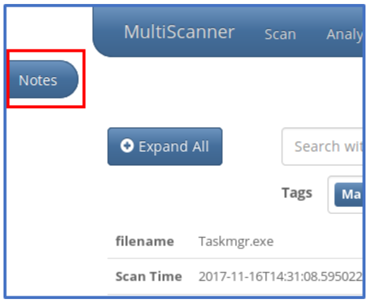

Task 4 Report
Expand All
Search within this report
Type something and hit the enter key to add a tag.
Tags
M
Malware

filename Taskmgr.exe
Scan Time 2017-11-16T14:31:08.595022
MD5 5b8f3bba71e442cb34fe54069ef9c306

You can download the report in a number of different formats using the Download button on the right side. You can download a JSON-formatted version of the report containing all the same data shown on the page. You can also download a MAEC-formatted version of the reports from Cuckoo Sandbox. Finally, you can also download the original sample file as a password-protected ZIP file. The password will be "infected".

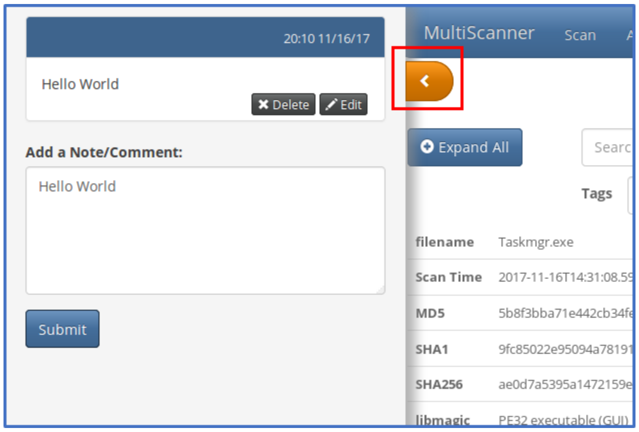


Click on "Notes" to open a sidebar where analysts may enter notes or comments.



filename Taskmgr.exe
Scan Time 2017-11-16T14:31:08.595022

These notes and comments can be edited and deleted. Click the "<" button to collapse this sidebar.



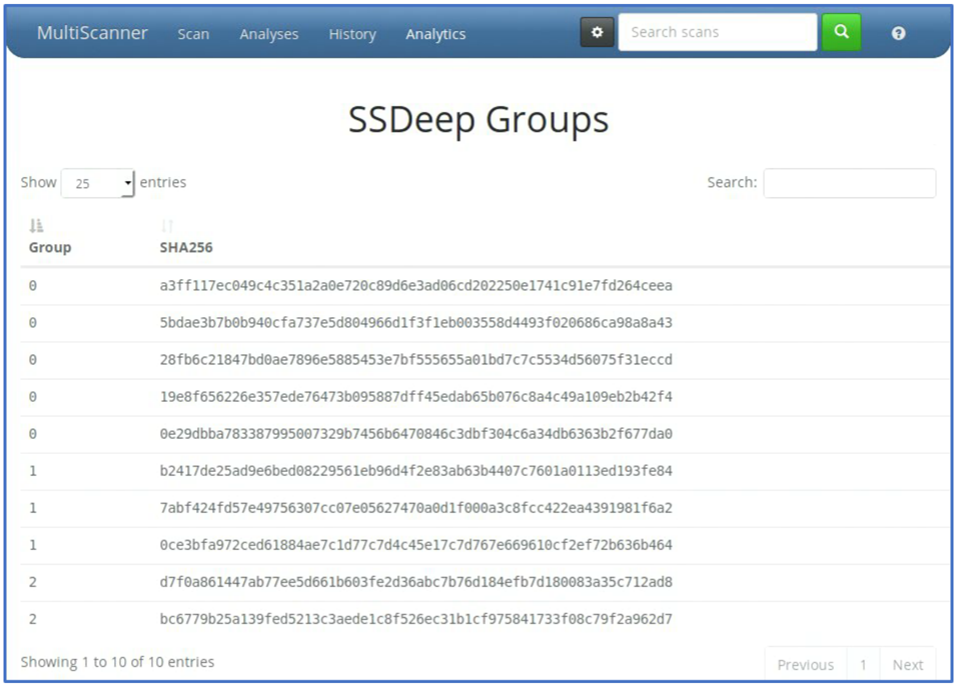
20:10 11/16/17
Hello World

Add a Note/Comment:
Hello World

filename Taskmgr.exe
Scan Time 2017-11-16T14:31:08.59
MD5 5b8f3bba71e442cb34fe
SHA1 9fc85022e95094a78191
SHA256 ae0d7a5395a1472159e
libmagic PE32 executable (GUI)

5.1.5 Viewing Analytics

The Analytics page displays various pieces of advanced analysis. For now, this is limited to ssdeep comparisons.



SSDeep Groups

Group	SHA256
0	a3ff117ec049c4c351a2a0e720c89d6e3ad06cd202250e1741c91e7fd264ceea
0	5bdae3b7b0b940cfa737e5d804966d1f3f1eb003558d4493f020686ca98a8a43
0	28fb6c21847bd0ae7896e5885453e7bf555655a01bd7c7c5534d56075f31eccd
0	19e8f656226e357ede76473b095887dff45edab65b076c8a4c49a109eb2b42f4
0	0e29dbba783387995007329b7456b6470846c3dbf304c6a34db6363b2f677da0
1	b2417de25ad9e6bed08229561eb96d4f2e83ab63b4407c7601a0113ed193fe84
1	7abf424fd57e49756307cc07e05627470a0d1f000a3c8fcc422ea4391981f6a2
1	0ce3bfa972ced61884ae7c1d77c7d4c45e17c7d767e669610cf2ef72b636b464
2	d7f0a861447ab77ee5d661b603fe2d36abc7b76d184efb7d180083a35c712ad8
2	bc6779b25a139fed5213c3aede1c8f526ec31b1cf975841733f08c79f2a962d7

Showing 1 to 10 of 10 entries

The table lists samples, with those that have very similar ssdeep hashes grouped together. Other analytics will be added in the future. For more information, see the analytics section.

5.2 Python API

MultiScanner can be incorporated as a module in another project. Below is a simple example of how to import MultiScanner into a Python script.

```python
import multiscanner
output = multiscanner.multiscan(file_list)
results = multiscanner.parse_reports(output, python=True)
```

`results` is a dictionary object where each key is a filename of a scanned file.

`multiscanner.config_init(filepath)` will create a default configuration file at the location defined by `filepath`.

5.3 RESTful API

The RESTful API is provided by a Flask app that supports the following operations:

Method	URI	Description
GET	/	Test functionality. Should produce: `{'Message': 'True'}`
GET	/api/v1/files/<sha256>?raw={t|f}	Download sample, defaults to passwd protected zip
GET	/api/v1/modules	Receive list of modules available
GET	/api/v1/tags	Receive list of all tags in use
GET	/api/v1/tasks	Receive list of tasks in MultiScanner
POST	/api/v1/tasks	POST file and receive report id. Sample POST usage: `curl -i -X POST http://localhost:8080/api/v1/tasks -F file=@/bin/ls`
GET	/api/v1/tasks/<task_id>	Receive task in JSON format
DELETE	/api/v1/tasks/<task_id>	Delete task_id
GET	/api/v1/tasks/search/	Receive list of most recent report for matching samples
GET	/api/v1/tasks/search/history	Receive list of most all reports for matching samples
GET	/api/v1/tasks/<task_id>/file?raw={t|f}	Download sample, defaults to passwd protected zip
GET	/api/v1/tasks/<task_id>/maec	Download the Cuckoo MAEC 5.0 report, if it exists
GET	/api/v1/tasks/<task_id>/notes	Receive list of this tasks notes
POST	/api/v1/tasks/<task_id>/notes	Add a note to task
PUT	/api/v1/tasks/<task_id>/notes/<note_id>	Edit a note
DELETE	/api/v1/tasks/<task_id>/notes/<note_id>	Delete a note
GET	/api/v1/tasks/<task_id>/report?d={t|f}	Receive report in JSON, set d=t to download
GET	/api/v1/tasks/<task_id>/pdf	Receive PDF report
POST	/api/v1/tasks/<task_id>/tags	Add tags to task
DELETE	/api/v1/tasks/<task_id>/tags	Remove tags from task
GET	/api/v1/analytics/ssdeep_compare	Run ssdeep.compare analytic
GET	/api/v1/analytics/ssdeep_group	Receive list of sample hashes grouped by ssdeep hash

The API endpoints all have Cross Origin Resource Sharing (CORS) enabled. By default it will allow requests from any port on localhost. Change this setting by modifying the `cors` setting in the `api` section of the api config file.

5.4 Analysis Modules

The analysis modules currently available in MultiScanner are listed by catagory below.

AV Scans	
AVG 2014	Scans sample with AVG 2014 anti-virus
ClamAVScan	Scans sample with ClamAV
McAfeeScan	Scans sample with McAfee AntiVirus Command Line
Microsoft Security Essentials	Scans sample with Microsoft Security Essentials
Metadefender	Interacts with OPSWAT Metadefender Core 4 Version 3.x, polling Metadefender for scan results.
vtsearch	Searches VirusTotal for sample's hash and downloads the report if available
VFind	Runs the CyberSoft VFind anti-malware scanner, part of the VFind Security Toolkit.

Database	
NSRL	Looks up a hash in the National Software Reference Library.

Sandbox Detonation	
Cuckoo Sandbox	Submits a sample to Cuckoo Sandbox cluster for analysis.
FireEye API	Detonates the sample in FireEye AX via FireEye's API.
VxStream	Submits a file to a VxStream Sandbox cluster for analysis.

Machine Learning	
MaliciousMacroBot	Triage office files with MaliciousMacroBot.

Metadata	
entropy	Calculates the Shannon entropy of a file.
ExifToolsScan	Scans sample with Exif tools and returns the results.
fileextensions	Determines possible file extensions for a file.
floss	FireEye Labs Obfuscated String Solver uses static analysis techniques to deobfuscate strings from malware binaries.
impfuzzy	Calculates a fuzzy hash using ssdeep on Windows PE imports.
libmagic	Runs libmagic against the files to identify filetype.
MD5	Generates the MD5 hash of the sample.
officemeta	Extracts metadata from Microsoft Office documents.
pdfinfo	Extracts feature information from PDF files using pdf-parser.
PEFile	Extracts features from EXE files.
pehasher	Computes pehash values using a variety of algorithms: totalhase, anymaster, anymaster_v1_0_1, endgame, crits, and pehashng.
SHA1	Generates the SHA1 hash of the sample.
SHA256	Generates the SHA256 hash of the sample.
ssdeep	Generates context triggered piecewise hashes (CTPH) for files. More information can be found on the ssdeep website.
Tika	Extracts metadata from the sample using Tika.
TrID	Runs TrID against a file.
UAD	Runs the CyberSoft Universal Atomic Disintegrator (UAD) tool, part of the VFind Security Toolkit.

Signatures	
YaraScan	Scans the sample with Yara and returns the results.

5.4.1 Configuration Options

Parameters common to all modules are listed in the next section, followed by notes and module-specific parameters for those that have them.

Common Parameters

The parameters below may be used by all modules.

Parameter	Description
path	Location of the executable.
cmdline	An array of command line options to be passed to the executable.
host	The hostname, port, and username of the machine that will be SSH'd into to run the analytic if the executable is not present on the local machine.
key	The SSH key to be used to SSH into the host.
replacement path	If the main config is set to copy the scanned files this will be what it replaces the path with. It should be where the network share is mounted.
ENABLED	When set to false, the module will not run.

[Cuckoo]

This module submits a file to a Cuckoo Sandbox cluster for analysis.

Parameter	Description
API URL	The URL to the API server.
WEB URL	The URL to the Web server.
timeout	The maximum time a sample will run.
running timeout	An additional timeout, if a task is in the running state this many seconds past `timeout`, the task is considered failed.
delete tasks	When set to True, tasks will be deleted from Cuckoo after detonation. This is to prevent filling up the Cuckoo machine's disk with reports.
maec	When set to True, a MAEC JSON-based report is added to Cuckoo JSON report. **NOTE**: Cuckoo needs MAEC reporting enabled to produce results.

[ExifToolsScan]

This module scans the file with Exif tools and returns the results.

Parameter	Description
remove-entry	A Python list of ExifTool results that should not be included in the report. File system level attributes are not useful and stripped out.

[FireEyeAPI]

This module detonates the sample in FireEye AX via FireEye's API. This "API" version replaces the "FireEye Scan" module.

Parameter	Description
API URL	The URL to the API server.
fireeye images	A Python list of the VMs in fireeye. These are used to generate where to copy the files.
username	Username on the FireEye AX.
password	Password for the FireEye AX.
info level	Options are concise, normal, and extended.
timeout	The maximum time a sample will run.
force	If set to True, will rescan if the sample matches a previous scan.
analysis type	0 = sandbox, 1 = live.
application id	For AX Series appliances (7.7 and higher) and CM Series appliances that manage AX Series appliances (7.7 and higher), setting the application value to -1 allows the AX Series appliance to choose the application. For other appliances, setting the application value to 0 allows the AX Series appliance to choose the application.

[floss]

This module extracts ASCII, UTF-8, stack and obfuscated strings from executable files. More information about module configuration can be found at the flare-floss documentation.

[impfuzzy]

This module calculates a fuzzy hash using ssdeep where Windows PE imports is the input. This strategy was originally described in a blog post from JPCERT/CC.

[libmagic]

This module runs libmagic against the files.

Parameter	Description
magicfile	The path to the compiled magic file you wish to use. If None it will use the default one.

[Metadefender]

This module runs Metadefender against the files.

Parameter	Description
timeout	The maximum time a sample will run.
running timeout	An additional timeout, if a task is in the running state this many seconds past `timeout`, the task is considered failed.
fetch delay seconds	The number of seconds for the module to wait between submitting all samples and polling for scan results. Increase this value if Metadefender is taking a long time to store the samples.
poll interval	The number of seconds between successive queries to Metadefender for scan results. Default is 5 seconds.
user agent	Metadefender user agent string, refer to your Metadefender server configuration for this value. Default is “user agent”.

[NSRL]

This module looks up hashes in the NSRL database. These two parameters are automatically generated. Users must run nsrl_parse.py tool in the utils/ directory before using this module.

Parameter	Description
hash_list	The path to the NSRL database on the local filesystem, containing the MD5 hash, SHA1 hash, and original file name.
offsets	A file that contains the pointers into hash_list file. This is necessary to speed up searching of the NSRL database file.

[officemeta]

This module extracts metadata from Microsoft Office documents.

Note: This module does not support OOXML documents (e.g., docx, pptx, xlsx).

[pdfinfo]

This module extracts out feature information from PDF files. It uses pdf-parser.

[PEFile]

This module extracts out feature information from EXE files. It uses pefile which is currently not available for python 3.

[ssdeeper]

This module generates context triggered piecewise hashes (CTPH) for the files. More information can be found on the ssdeep website.

[Tika]

This module extracts metadata from the file using Tika. For configuration of the module see the tika-python documentation.

Parameter	Description
remove-entry	A Python list of Tika results that should not be included in the report.

[TrID]

This module runs TrID against the files. The definition file should be in the same folder as the executable.

[vtsearch]

This module searches virustotal for the files hash and download the report if available.

Parameter	Description
apikey	Public/private api key. Can optionally make it a list and the requests will be distributed across them. This is useful when two groups with private api keys want to share the load and reports.

[VxStream]

This module submits a file to a VxStream Sandbox cluster for analysis.

Parameter	Description
BASE URL	The base URL of the VxStream server.
API URL	The URL to the API server (include the /api/ in this URL).
API Key	The user's API key to the API server.
API Secret	The user's secret to the API server.
Environment ID	The environment in which to execute the sample (if you have different sandboxes configured).
Verify	Set to false to ignore TLS certificate errors when querying the VxStream server.
timeout	The maximum time a sample will run
running timeout	An additional timeout, if a task is in the running state this many seconds past `timeout`, the task is considered failed.

[YaraScan]

This module scans the files with yara and returns the results. You will need yara-python installed for this module.

Parameter	Description
ruledir	The directory to look for rule files in.
fileextensions	A Python array of all valid rule file extensions. Files not ending in one of these will be ignored.
ignore-tags	A Python array of yara rule tags that will not be included in the report.

5.5 Analytics

Currently, one analytic is available.

5.5.1 ssdeep Comparison

Fuzzy hashing is an effective method to identify similar files based on common byte strings despite changes in the byte order and structure of the files. ssdeep provides a fuzzy hash implementation and provides the capability to compare hashes. Virus Bulletin originally described a method for comparing ssdeep hashes at scale.

Comparing ssdeep hashes at scale is a challenge. Therefore, the ssdeep analytic computes `ssdeep.compare` for all samples where the result is non-zero and provides the capability to return all samples clustered based on the ssdeep hash.

Elasticsearch

When possible, it can be effective to push work to the Elasticsearch cluster which support horizontal scaling. For the ssdeep comparison, Elasticsearch NGram Tokenizers are used to compute 7-grams of the chunk and double-chunk portions of the ssdeep hash, as described here. This prevents the comparison of two ssdeep hashes where the result will be zero.

Python

Because we need to compute `ssdeep.compare`, the ssdeep analytic cannot be done entirely in Elasticsearch. Python is used to query Elasticsearch, compute `ssdeep.compare` on the results, and update the documents in Elasticsearch.

Deployment

celery beat is used to schedule and kick off the ssdeep comparison task nightly at 2am local time, when the system is experiencing less load from users. This ensures that the analytic will be run on all samples without adding an exorbinant load to the system.

CHAPTER 6

Custom Development

6.1 Developing an Analysis Module

Modules are intended to be quickly written and incorporated into the MultiScanner framework. A module must be in the modules folder for it to be used on the next run. The configuration file does not need to be manually updated.

See this *Example Module*.

6.1.1 Mandatory Functions

When writing a new module, two mandatory functions must be defined: check() and scan(). Additional functions can be written as required.

check()

The check() function tests whether or not the scan function should be run.

Inputs: There are two supported argument sets with this function: `check()` and `check(conf=DEFAULTCONF)`. If a module has a global variable DEFAULTCONF, the second argument set is required.

Outputs: The return value of the check() function is a boolean (True or False). A True return value indicated the scan() function should be run; a False return value indicates the module should no longer be run.

scan()

The scan() function performs the analytic and returns the results.

Inputs: There are two supported argument sets with this function: `scan(filelist)` and `scan(filelist, conf=DEFAULTCONF)`. If a module has a global variable DEFAULTCONF, the second argument set is required.

Outputs: There are two return values of the scan() function: Results and Metadata (i.e., `return (Results, Metadata)`).

- **Results** is a list of tuples, the tuple values being the filename and the corresponding scan results (e.g.,``[("file1.exe", "Executable"), ("file2.jpg", "Picture")]``).
- **Metadata** is a dictionary of metadata information from the module. There are two required pieces of metadata `Name` and `Type`. `Name` is the name in the module and will be used in the report. `Type` is what type of module it is (e.g., Antivirus, content detonation). This information is used for a grouping feature in the report generation and provides context to a newly written module. Optionally, metadata information can be disabled and not be included in the report by setting `metadata["Include"] = False`.

6.1.2 Special Globals

There are two global variables that when present, affect the way the module is called.

DEFAULTCONF - This is a dictionary of configuration settings. When set, the settings will be written to the configuration file, making it user editable. The configuration object will be passed to the module's check and scan function and must be an argument in both functions.

REQUIRES - This is a list of analysis results required by a module. For example, `REQUIRES = ['MD5']` will be set to the output from the module MD5.py. An *Example Module* is provided.

6.1.3 Module Interface

The module interface is a class that is put into each module as it is run. This allows for several features to be added for interacting with the framework at runtime. It is injected as *multiscanner* in the global namespace.

Variables

- `write_dir` - This is a directory path that the module can write to. This will be unique for each run.
- `run_count` - This is an integer that increments for each subscan that is called. It is useful for preventing infinite recursion.

Functions

- `apply_async(func, args=(), kwds={}, callback=None)` - This mirrors multiprocessing.Pool.apply_async and returns a multiprocessing.pool.AsyncResult. The pool is shared by all modules.
- `scan_file(file_path, from_filename)` - This will scan a file that was found inside another file. *file_path* should be the extracted file on the filesystem (you can write it in path at *multiscanner.write_dir*). *from_filename* is the file it was extracted from.

6.1.4 Configuration

If a module requires configuration, the DEFAULTCONF global variable must be defined. This variable is passed to both check() and scan(). The configuration will be read from the configuration file if it is present. If the file is not present, it will be written into the configuration file.

If `replacement path` is set in the configuration, the module will receive file names, with the folder path replaced with the variable's value. This is useful for analytics which are run on a remote machine.

By default, ConfigParser reads everything in as a string, before options are passed to the module and `ast.literal_eval()` is run on each option. If a string is not returned when expected, this is why. This does mean that the correct Python type will be returned instead of all strings.

6.2 Developing an Analytic

Enabling analytics and advanced queries is the primary advantage of running several tools against a sample, extracting as much information as possible, and storing the output in a common datastore. For example, the following types of analytics and queries might be of interest:

- cluster samples
- outlier samples
- samples for deep-dive analysis
- gaps in current toolset
- machine learning analytics on tool outputs

Analytic development is currently ad hoc. Until interfaces are created to standardize development, the *Analytics* section might prove useful - it contains development details of the **ssdeep** analytic.

Here's the ssdeep code to use as a reference for how one might implement an analytic.

6.3 Writing a Storage Module

Each storage object is a class which needs to be derived from `storage.Storage`. You can have more than one storage object per python file.

6.3.1 Required components

You will need to override `store(self, results)`. `results` is a python dictionary that is one of two formats. It is either:

```
{
  "Files": {
    "file1": {},
    "file2": {}
  }
  "Metadata": {
    "module1": {},
    "module2": {}
  }
}
```

or

```
{
  "file1": {},
  "file2": {}
}
```

A storage module should support both, even if the metadata is discarded.

6.3.2 Optional components

- You can override `DEFAULTCONF` in your storage module. This is a dictionary of config options which will appear in the storage config file.

- You can override `setup(self)`. This should be anything that can be done once to prepare for mutliple calls to `store`, e.g. opening a network connection or file handle.
- You can override `teardown(self)`. This will be called when no more `store` calls are going to be made.

6.4 Example Module

```python
from __future__ import (division, absolute_import, with_statement,
                        print_function, unicode_literals)

TYPE = "Example"
NAME = "include example"
REQUIRES = ["libmagic", "MD5"]
DEFAULTCONF = {
    'ENABLED': True,
}

def check(conf=DEFAULTCONF):
    # If the config disabled the module don't run
    if not conf['ENABLED']:
        return False
    # If one of the required modules failed, don't run
    if None in REQUIRES:
        return False
    return True


def scan(filelist, conf=DEFAULTCONF):
    # Define our results array
    results = []
    # Pull out the libmagic results and metadata
    libmagicresults, libmagicmeta = REQUIRES[0]

    # Pull out the md5 results and metadata
    md5results, md5meta = REQUIRES[1]
    # Make the md5 results a dictionary
    md5dict = dict(md5results)

    # Run through each value in the libmagic results
    for filename, libmagicresult in libmagicresults:
        if libmagicresult.startswith('PDF document'):
            # If the file's md5 is present we will use that in the results
            if filename in md5dict:
                results.append((filename, md5dict[filename] + " is a pdf"))
            # If we don't know the md5 use the filename instead
            else:
                results.append((filename, "is a pdf"))

    # Create out metadata dictionary
    metadata = {}
    metadata["Name"] = NAME
    metadata["Type"] = TYPE

    # Return our super awesome results
    return (results, metadata)
```

CHAPTER 7

Testing

Running the MultiScanner test suite is fairly straight forward. We use the pytest framework, which you can install by running:

```
$ pip install pytest
```

After that, simply cd into the top level multiscanner directory and run the command:

```
$ pytest
```

This will automatically find all the tests in the tests/ directory and run them. We encourage developers of new modules and users to contribute to our testing suite!

7.1 Front-end Tests with Selenium

Running front-end tests with Selenium requires installation and configuration outside of the Python environment, namely the installation of Firefox and geckodriver.

1. Install Firefox.
2. Download latest geckodriver release from GitHub.
3. Add geckodriver to system path.

Additional information about geckodriver setup can be found here.

If pytest is unable to find Firefox or geckodriver, the front-end tests will be skipped. This is indicated by a 's' in the pytest output.

Tests have been run successfullly with Firefox 58 and geckodriver 0.19.1 on macOS and Ubuntu 14.04, 16.04.

7.1.1 CentOS

The Firefox version available in the base repo is too far out-of-date to be compatible with the tests. Manually update Firefox to the latest version.

1. Remove old version of Firefox:

   ```
   $ yum remove firefox
   ```

2. You may need to install these dependencies for Firefox:

   ```
   $ yum install -y gtk3 glib-devel glib pango pango-devel
   ```

3. Download latest version of Firefox:

   ```
   $ cd /usr/local
   $ curl -L http://ftp.mozilla.org/pub/firefox/releases/58.0/linux-x86_64/en-US/
   ↪firefox-58.0.tar.bz2 | tar -xjf
   ```

4. Add symlink to bin dir:

   ```
   $ ln -s /usr/local/firefox/firefox /usr/bin/firefox
   ```

CHAPTER 8

Presentations

An overview of MultiScanner architecture, use cases, and capabilities (with specific focus on data science applications) was presented at AnacondaCON 2018. The video can be found here.