Phagescan Documentation

Release 14.1

Narf Industries LLC

Sep 27, 2017

Contents

Phagescan is an open source virus scan aggregator for all you malware analysts and infosec types. It is designed to be an in-house replacement/augment for some cloud based services out there. The current code allows you to:

- Submit a file through the web interface for scanning by several different AV and other analysis engines
- Look at the results of each scan
- Add new engines/analysis tools by writing a thin Python wrapper (tools can run on Linux or Windows VMs)
- Isolate and monitor your "worker" VMs which actually execute the engines and perform scans (Openstack or EC2)

CHAPTER 1

Quick Start

The fastest way to get started is to follow the steps in our Quick Installation Guide.

CHAPTER 2

What do I need?

- Python 2.7
- Django >=1.5, <1.6
- Ubuntu 12.04 recommended for Master
- At least one scanner listed in the engines directory

Please note that some of engine adapters are commercial. This means you'll need to buy your own licenses or find trial versions to test things out. We have extensive experience in setting up custom installs of this project and you can contact us via the contact info on our site.

If you're developing on this project we *highly* recommend Pycharm and Vagrant. Maintaining a development environment and scanners on your box for development is a pain.

CHAPTER 3

Getting Help

- Check out our extensive documentation.
- Look us up on the PhageScan Google Group.

CHAPTER 4

High-level Design

Phagescan is split up into two major components: a scan master and multiple scan workers. The workers receive jobs from the master through rabbitMQ and the awesome Celery framework for RPC. Currently, samples are pushed across the Queue although this can be adapted to different storage models. This means that both the master and worker can be isolated from the Internet. It also means that just the workers can be isolated from the Internet. We're also working on functionality to update the workers online and then move them (one-way) to an Isolated network where they do their scanning.

Scan Master

This is the front end of Phagescan that houses the samples and the web interface and is responsible for the following tasks:

- Sending scan tasks to worker queues
- Storing the results of scan tasks
- Monitoring worker and scanner engine queue health
- Updating workers' virus definitions if Internet connectivity is allowed
- Summarizing scan results

The Scan master consists of a Django application for providing the web interface and a few necessary Celery tasks for keeping everything moving. The master is the only architectural entity that has permanent access to both the samples and the Postgres database. Postgresql is being used as the DB backend and is being accessed through Django's ORM. Postgres is required at this point because we're using the hstore extension to store arbitrary Key/value pairs from the scan results.

Scan Workers

This is the backend of the system and can scale horizontally. Each scan worker is a VM that has one or more scan engines (e.g. ClamAV, PEID) loaded on it. These scan engines can be an external binary or a Python module but

should not require access to the Master's database. In either case, a scan engine needs a Python adapter in the engines/ directory. Adapters in the engines directory are automatically enumerated on the worker and can automatically test which engines are installed. The worker uses this functionality to consume *only* from Queues that it has engines for. Check out the the AbstractEvilnessEngine or AbstractMDEngine classes and other concrete engines in the engine directory if you want to develop your own. Please note that both the master and workers need to have the same engines directory.

Types of Engines

We have two types of "scan engines" from above. They are: evilness engines and metadata engines. Metadata engines are things like PEID or OPAF. The key trait of a metadata engine is that it doesn't make a *judgement* on a sample's goodness.

An evilness engine does make a judgement on a file's good/bad state and needs to provide two pieces of data: a boolean indicating infection and a string indicating the type of infection. The evilness engine can also provide an arbitrary Python dict back to the scan master which contains any other traits that need to be stored in the DB for later retrieval/analysis.