HubbleStack Documentation

Release 4.0.0-1

Colton Myers, Christer Edwards

Sep 06, 2022

CONTENTS

HubbleStack (Hubble for short) is a modular, open-source, security & compliance auditing framework which is built in python, using SaltStack as a library. Hubble provides on-demand profile-based auditing, real-time security event notifications, alerting and reporting. It also reports the security information to Splunk, Logstash, or other endpoints. HubbleStack is a free and open source project made possible by Adobe.

CHAPTER

ONE

TABLE OF CONTENTS:

1.1 Getting Started with HubbleStack

1.1.1 Installation

Installation Using Released Packages (Recommended)

Various pre-built packages targeting several popular operating systems can be found under Releases.

Alternative Installations and Packaging

Building Hubble packages through Dockerfile

Dockerfile aims to build the Hubble v2 packages easier. Dockerfiles for the distribution you want to build can be found at the path `/pkg`. For example, dockerfile for centos6 distribution is at the path `/pkg/centos6/`

To build an image:

```
docker build -t <image_name>
```

To run the container (which will output the package file in your current directory):

```
docker run -it --rm -v `pwd`:/data <image_name>
```

Installing using setup.py

```
sudo yum install git python-setuptools -y
git clone https://github.com/hubblestack/hubble
cd hubble
sudo python setup.py install
```

If there are errors installing, it may mean that your setuptools is out of date. Try this:

```
easy_install pip
pip install -U setuptools
```

`setup.py` installs a hubble "binary" into `/usr/bin/`.

A config template has been placed in `/etc/hubble/hubble`. Modify it to your specifications and needs. You can do `hubble -h` to see the available runtime options.

The first two commands you should run to make sure things are set up correctly are `hubble --version` and `hubble test.ping`.

1.1.2 Basic Usage

Hubble runs as a standalone agent on each server you wish to monitor. To get started, install Hubble using one of the above installation options. Once Hubble is installed, check that everything is working correctly:

1. Run `hubble test.ping`. This should return true.
2. Run `hubble hubble.audit`. You should see the results of the default audit profiles run against the box

Quickstart via Docker container

Get up and running with any supported distribution by installing net-tools in a running docker container. `docker run -it {distro:tag} sh` the desired agent, then use the appropriate package manager to install net-tools:

To run centos:7 container:

```
docker run -it centos:7 sh
```

To install net-tools:

```
yum install net-tools
```

Follow instructions above in *Installation Using Released Packages (Recommended)*.

1.2 Auditing (Nova)

Hubble supports success/fail auditing via a number of included modules. The codename for the audit piece of hubble is "Nova".

1.2.1 Module Documentation

Nova (hubble.py)

1.2.2 Usage

There are two primary entry points for the Nova module:

`hubble.audit`

> audits the agent using the YAML profile(s) you provide as comma-separated arguments.
>
> `hubble.audit` takes a number of optional arguments. The first is a comma-separated list of paths. These paths can be files or directories within the `hubblestack_nova_profiles` directory, with the `.yaml` suffix removed. For information on the other arguments, please see *Nova (hubble.py)*.
>
> If `hubble.audit` is run without targeting any audit configs or directories, it will instead run `hubble.top` with no arguments.
>
> `hubble.audit` will return a list of audits which were successful, and a list of audits which failed.

`hubble.top`

audits the agent using the `top.nova` configuration. By default, the `top.nova` should be located in the fileserver at `salt://hubblestack_nova_profiles/top.nova`, but a different path can be defined.

Here are some example calls for `hubble.audit`:

```
# Run the cve scanner and the CIS profile:
hubble hubble.audit cve.scan-v2,cis.centos-7-level-1-scored-v1
# Run hubble.top with the default topfile (top.nova)
hubble hubble.top
# Run all yaml configs and tags under salt://hubblestack_nova_profiles/foo/ and salt://
↪hubblestack_nova_profiles/bar, but only run audits with tags starting with "CIS"
hubble hubble.audit foo,bar tags='CIS*'
```

1.2.3 Configuration

For Nova module, configurations can be done via Nova topfiles. Nova topfiles look very similar to saltstack topfiles, except the top-level key is always nova, as nova doesn't have environments.

hubblestack_data/hubblestack_nova_profiles/top.nova:

```
nova:
  '*':
    - cve.scan-v2
    - network.ssh
    - network.smtp
  'web*':
    - cis.centos-7-level-1-scored-v1
    - cis.centos-7-level-2-scored-v1
  'G@os_family:debian':
    - network.ssh
    - cis.debian-7-level-1-scored: 'CIS*'
```

Additionally, all nova topfile matches are compound matches, so you never need to define a match type like you do in saltstack topfiles. Each list item is a string representing the dot-separated location of a yaml file which will be run with `hubble.audit`. You can also specify a tag glob to use as a filter for just that yaml file, using a colon after the yaml file (turning it into a dictionary). See the last two lines in the yaml above for examples.

Examples:

```
hubble hubble.top
hubble hubble.top foo/bar/top.nova
hubble hubble.top foo/bar.nova verbose=True
```

In some cases, your organization may want to skip certain audit checks for certain hosts. This is supported via compensating control configuration.

You can skip a check globally by adding a `control: <reason>` key to the check itself. This key should be added at the same level as description and trigger pieces of a check. In this case, the check will never run, and will output under the Controlled results key.

Nova also supports separate control profiles, for more fine-grained control using topfiles. You can use a separate YAML top-level key called control. Generally, you'll put this top-level key inside of a separate YAML file and only include it in the top-data for the hosts for which it is relevant.

For these separate control configs, the audits will always run, whether they are controlled or not. However, controlled audits which fail will be converted from Failure to Controlled in a post-processing operation.

The control config syntax is as follows:

hubblestack_data/hubblestack_nova_profiles/example_control/example.yaml:

```
control:
  - CIS-2.1.4: This is the reason we control the check
  - some_other_tag:
      reason: This is the reason we control the check
  - a_third_tag_with_no_reason
```

Note that providing a reason for the control is optional. Any of the three formats shown in the yaml list above will work.

Once you have your compensating control config, just target the yaml to the hosts you want to control using your topfile. In this case, all the audits will still run, but if any of the controlled checks fail, they will be removed from Failure and added to Controlled, and will be treated as a Success for the purposes of compliance percentage. To use the above control, you would add the following to your `top.nova` file:

```
nova:
  '*':
    - example_control.example
```

It is quite possible you may want to send the NOVA audit results to a collection system. This can be accomplished by using returners, which are included with Hubble. Currently, there are returners available for **splunk**, **logstash**, **graylog**, and **sqlite**. Hubble supports using multiple returners from the same node so sending data to multiple collectors is possible. Each individual returner has distinct options that need to be set prior to their use. You can find general config options for each returner at the top of each file located here https://github.com/hubblestack/hubble/tree/develop/hubblestack/extmods/returners.

Specify a returner when running Hubble from the command line:

```
hubble hubble.audit --r RETURN
hubble hubble.audit --return RETURN
```

You can also define schedules for returners in the Hubble config file, which is typically located at `/etc/hubble/hubble`.

1.3 Insights and Data Gathering (Nebula)

Hubble can gather incredible amounts of raw data from your hosts for later analysis. The codename for the insights piece of hubble is Nebula. It primarily uses osquery which allows you to query your system as if it were a database.

1.3.1 Module Documentation

Nebula (nebula_osquery.py)

1.3.2 Usage

Nebula queries are formatted into query groups which allow you to schedule sets of queries to run at different cadences. The names of these groups are arbitrary, but the queries provided in hubblestack_data are grouped by timing:

```
fifteen_min:
  running_procs:
    query: SELECT t.unix_time AS query_time, p.name AS process, p.pid AS process_id, p.
↪pgroup AS process_group, p.cmdline, p.cwd, p.on_disk, p.resident_size AS mem_used, p.
↪user_time, p.system_time, (SELECT strftime('%s','now')-ut.total_seconds+p.start_time␣
↪FROM uptime AS ut) AS process_start_time, p.parent, pp.name AS parent_name, g.
↪groupname AS 'group', g.gid AS group_id, u.username AS user, u.uid AS user_id, eu.
↪username AS effective_username, eg.groupname AS effective_groupname, p.path, h.md5 AS␣
↪md5, h.sha1 AS sha1, h.sha256 AS sha256, '__JSONIFY__'||(SELECT json_group_array(json_
↪object('fd',pof.fd, 'path',pof.path)) FROM process_open_files AS pof WHERE pof.pid=p.
↪pid GROUP BY pof.pid) AS open_files, '__JSONIFY__'||(SELECT json_group_array(json_
↪object('variable_name',pe.key, 'value',pe.value)) FROM process_envs AS pe WHERE pe.
↪pid=p.pid GROUP BY pe.pid) AS environment FROM processes AS p LEFT JOIN processes AS␣
↪pp ON p.parent=pp.pid LEFT JOIN users AS u ON p.uid=u.uid LEFT JOIN users AS eu ON p.
↪euid=eu.uid LEFT JOIN groups AS g ON p.gid=g.gid LEFT JOIN groups AS eg ON p.gid=eg.
↪gid LEFT JOIN hash AS h ON p.path=h.path LEFT JOIN time AS t WHERE p.parent IS NOT 2␣
↪AND (process NOTNULL OR p.parent NOTNULL);
  established_outbound:
    query: SELECT t.unix_time AS query_time, CASE pos.family WHEN 2 THEN 'ipv4' WHEN 10␣
↪THEN 'ipv6' ELSE pos.family END AS family, h.md5 AS md5, h.sha1 AS sha1, h.sha256 AS␣
↪sha256, h.directory AS directory, ltrim(pos.local_address, ':f') AS src_connection_ip,␣
↪pos.local_port AS src_connection_port, pos.remote_port AS dest_connection_port,␣
↪ltrim(pos.remote_address, ':f') AS dest_connection_ip, p.name AS name, p.pid AS pid, p.
↪parent AS parent_pid, pp.name AS parent_process, p.path AS file_path, f.size AS file_
↪size, p.cmdline AS cmdline, u.uid AS uid, u.username AS username, CASE pos.protocol␣
↪WHEN 6 THEN 'tcp' WHEN 17 THEN 'udp' ELSE pos.protocol END AS transport FROM process_
↪open_sockets AS pos JOIN processes AS p ON p.pid=pos.pid LEFT JOIN processes AS pp ON␣
↪p.parent=pp.pid LEFT JOIN users AS u ON p.uid=u.uid LEFT JOIN time AS t LEFT JOIN hash␣
↪AS h ON h.path=p.path LEFT JOIN file AS f ON f.path=p.path WHERE NOT pos.remote_
↪address='' AND NOT pos.remote_address='::' AND NOT pos.remote_address='0.0.0.0' AND␣
↪NOT pos.remote_address='127.0.0.1' AND (pos.local_port,pos.protocol) NOT IN (SELECT lp.
↪port, lp.protocol FROM listening_ports AS lp);
hour:
  crontab:
    query: SELECT t.unix_time AS query_time, c.event, c.minute, c.hour, c.day_of_month,␣
↪c.month, c.day_of_week, c.command, c.path AS cron_file FROM crontab AS c JOIN time AS␣
↪t;
  login_history:
    query: SELECT t.unix_time AS query_time, l.username AS user, l.tty, l.pid, l.type AS␣
↪utmp_type, CASE l.type WHEN 1 THEN 'RUN_LVL' WHEN 2 THEN 'BOOT_TIME' WHEN 3 THEN 'NEW_
↪TIME' WHEN 4 THEN 'OLD_TIME' WHEN 5 THEN 'INIT_PROCESS' WHEN 6 THEN 'LOGIN_PROCESS'␣
↪WHEN 7 THEN 'USER_PROCESS' WHEN 8 THEN 'DEAD_PROCESS' ELSE l.type END AS utmp_type_
↪name, l.host AS src, l.time FROM last AS l LEFT JOIN time AS t WHERE l.time > strftime(
↪'%s','now') - 3600;
  docker_running_containers:
    query: SELECT t.unix_time AS query_time, dc.id AS container_id, dc.name AS container_
↪name, dc.image AS image_name, di.created AS image_created_time, di.size_bytes AS image_
↪size, di.tags AS image_tags, dc.image_id AS image_id, dc.command AS container_command,␣
```

(continues on next page)

(continued from previous page)

```
↪dc.created AS container_start_time, dc.state AS container_state, dc.status AS status,
↪'__JSONIFY__'||(SELECT json_group_array(json_object('key',dcl.key, 'value',dcl.value))␣
↪FROM docker_container_labels AS dcl WHERE dcl.id=dc.id GROUP BY dcl.id) AS container_
↪labels, '__JSONIFY__'||(SELECT json_group_array(json_object('mount_type',dcm.type,
↪'mount_name',dcm.name, 'mount_host_path',dcm.source,  'mount_container_path',dcm.
↪destination, 'mount_driver',dcm.driver, 'mount_mode',dcm.mode, 'mount_rw',dcm.rw,
↪'mount_progpagation',dcm.propagation)) FROM docker_container_mounts AS dcm WHERE dcm.
↪id=dc.id GROUP BY dcm.id) AS container_mounts, '__JSONIFY__'||(SELECT json_group_
↪array(json_object('port_type',dcport.type, 'port',dcport.port, 'host_ip',dcport.host_
↪ip,  'host_port',dcport.host_port)) FROM docker_container_ports AS dcport WHERE dcport.
↪id=dc.id GROUP BY dcport.id) AS container_ports, '__JSONIFY__'||(SELECT json_group_
↪array(json_object('network_name',dcnet.name, 'network_id',dcnet.network_id, 'endpoint_
↪id',dcnet.endpoint_id, 'gateway',dcnet.gateway, 'container_ip',dcnet.ip_address,
↪'container_ip_prefix_len',dcnet.ip_prefix_len, 'ipv6_gateway',dcnet.ipv6_gateway,
↪'container_ipv6_address',dcnet.ipv6_address, 'container_ipv6_prefix_len',dcnet.ipv6_
↪prefix_len, 'container_mac_address',dcnet.mac_address)) FROM docker_container_networks␣
↪AS dcnet WHERE dcnet.id=dc.id GROUP BY dcnet.id) AS container_networks FROM docker_
↪containers AS dc JOIN docker_images AS di ON di.id=dc.image_id LEFT JOIN time AS t;
day:
  rpm_packages:
    query: SELECT t.unix_time AS query_time, rpm.name, rpm.version, rpm.release, rpm.
↪source AS package_source, rpm.size, rpm.sha1, rpm.arch FROM rpm_packages AS rpm JOIN␣
↪time AS t;
  os_info:
    query: SELECT t.unix_time AS query_time, os.* FROM os_version AS os LEFT JOIN time␣
↪AS t;
  interface_addresses:
    query: SELECT t.unix_time AS query_time, ia.interface, ia.address, id.mac FROM␣
↪interface_addresses AS ia JOIN interface_details AS id ON ia.interface=id.interface␣
↪LEFT JOIN time AS t WHERE NOT ia.interface='lo';
```

Nebula query data is very verbose, and is really meant to be sent to a central aggregation location such as splunk or logstash for further processing. However, if you would like to run the queries manually you can call the *nebula execution module*:

```
hubble nebula.queries day
hubble nebula.queries hour verbose=True
```

1.3.3 topfiles

Nebula supports organizing query groups across files, and combining/targeting them via a `top.nebula` file (similar to topfiles in SaltStack):

```
nebula:
  - '*':
      - hubblestack_nebula_queries
  - 'G@splunk_index:some_team':
      - some_team
```

Each entry under `nebula` is a SaltStack style compound match that describes which hosts should receive the list of queries. All queries are merged, and conflicts go to the last-defined file.

The files referenced are relative to `salt://hubblestack_nebula_v2/` and leave off the `.yaml` extension.

You can also specify an alternate `top.nebula` file.

For more details, see the module documentation: *Nebula (nebula_osquery.py)*

1.4 File Integrity Monitoring/FIM (Linux) (Pulsar)

Pulsar is designed to monitor for file system events, acting as a real-time File Integrity Monitoring (FIM) agent. Pulsar uses python-inotify to watch for these events and report them to your destination of choice.

1.4.1 Module Documentation

Pulsar (Linux) (pulsar.py)

1.4.2 Usage

Once Pulsar is configured there isn't anything you need to do to interact with it. It simply runs quietly in the background and sends you alerts.

Note: Running pulsar outside of hubble's scheduler will never return results. This is because the first time you run pulsar it will set up the watches in inotify, but no events will have been generated. Only subsequent runs under the same process can receive events.

1.4.3 Configuration

The list of files and directories that pulsar watches is defined in salt://hubblestack_pulsar/hubblestack_pulsar_config.yaml:

```
/lib: { recurse: True, auto_add: True }
/bin: { recurse: True, auto_add: True }
/sbin: { recurse: True, auto_add: True }
/boot: { recurse: True, auto_add: True }
/lib64: { recurse: True, auto_add: True }
/usr/lib: { recurse: True, auto_add: True }
/usr/bin: { recurse: True, auto_add: True }
/usr/sbin: { recurse: True, auto_add: True }
/usr/lib64: { recurse: True, auto_add: True }
/usr/libexec: { recurse: True, auto_add: True }
/usr/local/etc: { recurse: True, auto_add: True }
/usr/local/bin: { recurse: True, auto_add: True }
/usr/local/lib: { recurse: True, auto_add: True }
/usr/local/sbin: { recurse: True, auto_add: True }
/usr/local/libexec: { recurse: True, auto_add: True }
/opt/bin: { recurse: True, auto_add: True }
/opt/osquery: { recurse: True, auto_add: True }
/opt/hubble: { recurse: True, auto_add: True }
/etc:
  exclude:
    - /etc/passwd.lock
```

(continues on next page)

(continued from previous page)

```
    - /etc/shadow.lock
    - /etc/gshadow.lock
    - /etc/group.lock
    - /etc/passwd+
    - /etc/passwd-
    - /etc/shadow+
    - /etc/shadow-
    - /etc/group+
    - /etc/group-
    - /etc/gshadow+
    - /etc/gshadow-
    - /etc/cas/timestamp
    - /etc/resolv.conf.tmp
    - /etc/pki/nssdb/key4.db-journal
    - /etc/pki/nssdb/cert9.db-journal
    - /etc/salt/gpgkeys/random_seed
    - /etc/blkid/blkid.tab.old
    - \/etc\/blkid\/blkid\.tab\-\w{6}$:
        regex: True
    - \/etc\/passwd\.\d*$:
        regex: True
    - \/etc\/group\.\d*$:
        regex: True
    - \/etc\/shadow\.\d*$:
        regex: True
    - \/etc\/gshadow\.\d*$:
        regex: True
  recurse: True
  auto_add: True
return: splunk_pulsar_return
checksum: sha256
stats: True
batch: True
```

Note some of the available options: you can recurse through directories, auto_add new files and directories as they are created, or exclude based on glob or regex patterns.

topfiles

Pulsar supports organizing query groups across files, and combining/targeting them via a `top.pulsar` file (similar to topfiles in SaltStack):

```
pulsar:
  '*':
    - hubblestack_pulsar_config
```

Each entry under `pulsar` is a SaltStack style compound match that describes which hosts should receive the list of queries. All queries are merged, and conflicts go to the last-defined file.

The files referenced are relative to `salt://hubblestack_pulsar/` and leave off the `.yaml` extension.

You can also specify an alternate `top.pulsar` file.

For more details, see the module documentation: *Pulsar (Linux) (pulsar.py)*

1.5 File Integrity Monitoring/FIM (Windows) (Pulsar)

Pulsar for Windows is designed to monitor for file system events, acting as a real-time File Integrity Monitoring (FIM) agent. On Windows systems, pulsar uses ntfs journaling watch for these events and report them to your destination of choice.

1.5.1 Module Documentation

Pulsar (Windows) (win_pulsar.py)

1.5.2 Usage

Once Pulsar is configured there isn't anything you need to do to interact with it. It simply runs quietly in the background and sends you alerts.

Note: Running pulsar outside of hubble's scheduler will never return results. This is because the first time you run pulsar it will set up the watches in inotify, but no events will have been generated. Only subsequent runs under the same process can receive events.

1.6 Module Documentation

1.6.1 Nova (`hubble.py`)

1.6.2 Nebula (`nebula_osquery.py`)

1.6.3 Pulsar (Linux) (`pulsar.py`)

1.6.4 Pulsar (Windows) (`win_pulsar.py`)

CHAPTER

TWO

INDICES AND TABLES:

- genindex
- modindex
- search