HubbleStack Documentation

Release 2.4.1-1

Colton Myers, Christer Edwards

Aug 07, 2018

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 1

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If there are errors installing, it may mean that your setuptools is out of date.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

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
```

(continues on next page)

(continued from previous page)

```
  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, 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:
```

(continues on next page)

(continued from previous page)

```
    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
    - /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
```

(continues on next page)

(continued from previous page)

```
    - \/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`)

Loader and primary interface for nova modules

See README for documentation

Configuration:

- hubblestack:nova:module_dir
- hubblestack:nova:profile_dir
- hubblestack:nova:saltenv
- hubblestack:nova:autoload
- hubblestack:nova:autosync

`hubblestack.extmods.modules.hubble.`**`audit`**(*configs=None*, *tags='*'*, *verbose=None*, *show_success=None*, *show_compliance=None*, *show_profile=None*, *called_from_top=None*, *debug=None*, *labels=None*, ***kwargs*)

Primary entry point for audit calls.

configs List (comma-separated or python list) of yaml configs/directories to search for audit data. Directories are dot-separated, much in the same way as Salt states. For individual config names, leave the .yaml extension off. If a given path resolves to a python file, it will be treated as a single config. Otherwise it will be treated as a directory. All configs found in a recursive search of the specified directories will be included in the audit.

If configs is not provided, this function will call `hubble.top` instead.

tags Glob pattern string for tags to include in the audit. This way you can give a directory, and tell the system to only run the *CIS**-tagged audits, for example.

verbose Whether to show additional information about audits, including description, remediation instructions, etc. The data returned depends on the audit module. Defaults to False. Configurable via *hubblestack:nova:verbose* in minion config/pillar.

show_success Whether to show successful audits in addition to failed audits. Defaults to True. Configurable via *hubblestack:nova:show_success* in minion config/pillar.

show_compliance Whether to show compliance as a percentage (successful checks divided by total checks). Defaults to True. Configurable via *hubblestack:nova:show_compliance* in minion config/pillar.

show_profile DEPRECATED

called_from_top Ignore this argument. It is used for distinguishing between user-calls of this function and calls from hubble.top.

debug Whether to log additional information to help debug nova. Defaults to False. Configurable via *hubblestack:nova:debug* in minion config/pillar.

labels Tests with matching labels are executed. If multiple labels are passed, then tests which have all those labels are executed.

****kwargs** Any parameters & values that are not explicitly defined will be passed directly through to the Nova module(s).

CLI Examples:

```
salt '*' hubble.audit foo
salt '*' hubble.audit foo,bar tags='CIS*'
salt '*' hubble.audit foo,bar.baz verbose=True
```

hubblestack.extmods.modules.hubble.**top**(*topfile='top.nova'*, *verbose=None*, *show_success=None*, *show_compliance=None*, *show_profile=None*, *debug=None*, *labels=None*)

Compile and run all yaml data from the specified nova topfile.

Nova topfiles look very similar to saltstack topfiles, except the top-level key is always `nova`, as nova doesn't have a concept of environments.

```
nova:
  '*':
    - cve_scan
    - cis_gen
  'web*':
    - firewall
    - cis-centos-7-l2-scored
    - cis-centos-7-apache24-l1-scored
  'G@os_family:debian':
    - netstat
    - cis-debian-7-l2-scored: 'CIS*'
    - cis-debian-7-mysql57-l1-scored: 'CIS 2.1.2'
```

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.

Arguments:

topfile The path of the topfile, relative to your hubblestack_nova_profiles directory.

verbose Whether to show additional information about audits, including description, remediation instructions, etc. The data returned depends on the audit module. Defaults to False. Configurable via *hubblestack:nova:verbose* in minion config/pillar.

show_success Whether to show successful audits in addition to failed audits. Defaults to True. Configurable via *hubblestack:nova:show_success* in minion config/pillar.

show_compliance Whether to show compliance as a percentage (successful checks divided by total checks). Defaults to True. Configurable via *hubblestack:nova:show_compliance* in minion config/pillar.

show_profile DEPRECATED

debug Whether to log additional information to help debug nova. Defaults to False. Configurable via *hubblestack:nova:debug* in minion config/pillar.

CLI Examples:

```
salt '*' hubble.top
salt '*' hubble.top foo/bar/top.nova
salt '*' hubble.top foo/bar.nova verbose=True
```

1.6.2 Nebula (`nebula_osquery.py`)

osquery wrapper for HubbleStack Nebula

Designed to run sets of osquery queries defined in pillar. These sets will have a unique identifier, and be targeted by identifier. Usually, this identifier will be a frequency. ('15 minutes', '1 day', etc). Identifiers are case-insensitive.

You can then use the scheduler of your choice to run sets os queries at whatever frequency you choose.

Sample pillar data:

nebula_osquery:

hour:

- crontab: query: select c.*,t.iso_8601 as _time from crontab as c join time as t;
- query_name: suid_binaries query: select sb.*, t.iso_8601 as _time from suid_bin as sb join time as t;

day:

- query_name: rpm_packages query: select rpm.*, t.iso_8601 from rpm_packages as rpm join time as t;

`hubblestack.extmods.modules.nebula_osquery.`**`queries`**(*query_group*, *query_file=None*, *verbose=False*, *report_version_with_day=True*, *topfile_for_mask=None*, *mask_passwords=False*)

Run the set of queries represented by `query_group` from the configuration in the file query_file

query_group Group of queries to run

query_file salt:// file which will be parsed for osquery queries

verbose Defaults to False. If set to True, more information (such as the query which was run) will be included in the result.

topfile_for_mask This is the location of the top file from which the masking information will be extracted.

mask_passwords Defaults to False. If set to True, passwords mentioned in the return object are masked.

CLI Examples:

```
salt '*' nebula.queries day
salt '*' nebula.queries hour verbose=True
salt '*' nebula.queries hour pillar_key=sec_osqueries
```

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

Watch files and translate the changes into salt events

depends

- pyinotify Python module >= 0.9.5

Caution Using generic mask options like open, access, ignored, and closed_nowrite with reactors can easily cause the reactor to loop on itself. To mitigate this behavior, consider setting the *disable_during_state_run* flag to *True* in the beacon configuration.

`hubblestack.extmods.modules.pulsar.`**`process`**(*configfile='salt://hubblestack_pulsar/hubblestack_pulsar_config.yaml'*, *verbose=False*)

Watch the configured files

Example pillar config

```
beacons:
  pulsar:
    paths:
      - /var/cache/salt/minion/files/base/hubblestack_pulsar/hubblestack_pulsar_
↪config.yaml
    refresh_interval: 300
    verbose: False
```

Example yaml config on fileserver (targeted by pillar)

```
/path/to/file/or/dir:
  mask:
    - open
    - create
    - close_write
  recurse: True
  auto_add: True
  exclude:
    - /path/to/file/or/dir/exclude1
    - /path/to/file/or/dir/exclude2
    - /path/to/file/or/dir/regex[\d]*$:
        regex: True
return:
  splunk:
    batch: True
  slack:
    batch: False  # overrides the global setting
checksum: sha256
stats: True
batch: True
contents_size: 20480
checksum_size: 104857600
```

Note that if *batch: True*, the configured returner must support receiving a list of events, rather than single one-off events.

The mask list can contain the following events (the default mask is create, delete, and modify):

- access - File accessed
- attrib - File metadata changed
- close_nowrite - Unwritable file closed
- close_write - Writable file closed
- create - File created in watched directory
- delete - File deleted from watched directory
- delete_self - Watched file or directory deleted
- modify - File modified
- moved_from - File moved out of watched directory
- moved_to - File moved into watched directory
- move_self - Watched file moved
- open - File opened

The mask can also contain the following options:

- dont_follow - Don't dereference symbolic links
- excl_unlink - Omit events for children after they have been unlinked
- oneshot - Remove watch after one event
- onlydir - Operate only if name is directory

recurse: Recursively watch directories under the named directory

auto_add: Python inotify option, meaning: automatically start watching new directories that are created in a watched directory

watch_new_files: when a new file is created in a watched dir, add a watch on the file (implied by watch_files below)

watch_files: add explicit watches on all files (except excluded) under the named directory

exclude: Exclude directories or files from triggering events in the watched directory. Can use regex if regex is set to True

contents: Retrieve the contents of changed files based on checksums (which must be enabled)

If pillar/grains/minion config key *hubblestack:pulsar:maintenance* is set to True, then changes will be discarded.

`hubblestack.extmods.modules.pulsar.`**`top`**(*topfile='salt://hubblestack_pulsar/top.pulsar'*, *verbose=False*)

Execute pulsar using a top.pulsar file to decide which configs to use for this host.

The topfile should be formatted like this:

```
pulsar:
  '<salt compound match identifying host(s)>':
    - list.of.paths
    - using.dots.as.directory.separators
```

Paths in the topfile should be relative to *salt://hubblestack_pulsar*, and the .yaml should not be included.

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

This will setup your computer to enable auditing for specified folders inputted into a yaml file. It will then scan the ntfs journal for changes to those folders and report when it finds one.

`hubblestack.extmods.modules.win_pulsar.`**`process`**(*configfile='salt://hubblestack_pulsar/hubblestack_pulsar_win_c*
verbose=False)

Watch the configured files

Example yaml config on fileserver (targeted by configfile option)

```
C:\Users: {}
C:\Windows:
  mask:
    - 'File Create'
    - 'File Delete'
    - 'Security Change'
  exclude:
    - C:\Windows\System32\*
C:      emp: {}
```

(continues on next page)

(continued from previous page)

```
return: splunk_pulsar_return
batch: True
```

Note that if 'batch: True', the configured returner must support receiving a list of events, rather than single one-off events

the mask list can contain the following events (the default mask is create, delete, and modify):

1. Basic Info Change A user has either changed file or directory attributes, or one or more time stamps
2. Close The file or directory is closed
3. Compression Change The compression state of the file or directory is changed from or to compressed
4. Data Extend The file or directory is extended (added to)
5. Data Overwrite The data in the file or directory is overwritten
6. Data Truncation The file or directory is truncated
7. **EA Change A user made a change to the extended attributes of a file or directory (These NTFS** file system attributes are not accessible to Windows-based applications)
8. Encryption Change The file or directory is encrypted or decrypted
9. File Create The file or directory is created for the first time
10. File Delete The file or directory is deleted
11. Hard Link Change An NTFS file system hard link is added to or removed from the file or directory
12. **Indexable Change A user changes the FILE_ATTRIBUTE_NOT_CONTENT_INDEXED attribute (changes the file** or directory from one where content can be indexed to one where content cannot be indexed, or vice versa)
13. **Integrity Change A user changed the state of the FILE_ATTRIBUTE_INTEGRITY_STREAM attribute for the give** stream (On the ReFS file system, integrity streams maintain a checksum of all data for that stream, so that the contents of the file can be validated during read or write operations)
14. Named Data Extend The one or more named data streams for a file are extended (added to)
15. Named Data Overwrite The data in one or more named data streams for a file is overwritten
16. Named Data truncation The one or more named data streams for a file is truncated
17. Object ID Change The object identifier of a file or directory is changed
18. **Rename New Name A file or directory is renamed, and the file name in the USN_RECORD_V2 structure is the** new name
19. **Rename Old Name The file or directory is renamed, and the file name in the USN_RECORD_V2 structure is** the previous name
20. **Reparse Point Change The reparse point that is contained in a file or directory is changed, or a reparse** point is added to or deleted from a file or directory
21. Security Change A change is made in the access rights to a file or directory
22. Stream Change A named stream is added to or removed from a file, or a named stream is renamed
23. Transacted Change The given stream is modified through a TxF transaction

exclude: Exclude directories or files from triggering events in the watched directory. **Note that the directory excludes shoud not have a trailing slash**

Returns

hubblestack.extmods.modules.win_pulsar.**top**(*topfile='salt://hubblestack_pulsar/win_top.pulsar'*, *verbose=False*)

Execute pulsar using a top.pulsar file to decide which configs to use for this host.

The topfile should be formatted like this:

```
pulsar:
  '<salt compound match identifying host(s)>':
    - list.of.paths
    - using.dots.as.directory.separators
```

Paths in the topfile should be relative to *salt://hubblestack_pulsar*, and the .yaml should not be included.

CHAPTER 2

Indices and Tables:

- genindex
- modindex
- search

Python Module Index

Index