ThreatQuotient

ThreatQuotient for Resilient Functions Guide

Version 1.1.0

Friday, January 31, 2020

ThreatQuotient

11400 Commerce Park Dr., Suite 200

Reston, VA 20191

Support

Email: support@threatq.com

Web: support.threatq.com

Phone: 703.574.9893

Warning and Disclaimer

Last Updated: Friday, January 31, 2020

Contents

Introduction

This package implements the Resilient Functions, allowing it to react to Contextual Actions and Automatic Actions.

Preface

This guide provides the information necessary to implement the ThreatQuotient for Resilient Custom Threat Service. This document is not specifically intended as a site reference guide. It is assumed that the implementation engineer has experience installing and commissioning the ThreatQuotient Apps and integrations covered within the document, as well as the experience necessary to troubleshoot at a basic level.

Audience

This document is intended for use by the following parties:

1. ThreatQ Security/Engineers.
2. ThreatQuotient Professional Services Project Team & Engineers.

Versioning

- Current integration version: `1.1.0`
- Supported on ThreatQ versions `>= 4.24`

Assumptions

The following criteria is assumed to be in place and functional to allow the implementation of the ThreatQuotient for Resilient Custom Threat Service into the managed estate:

- All ThreatQuotient equipment is online and in service.
- All required firewall ports have been opened.

Pre-requisites

Python 3.x

In order to run the Resilient Integrations for ThreatQuotient, you must install a Python 3 environment onto your Resilient (or integration) server.

1. SSH into your Resilient Server or Resilient Integration Server
2. Install Python 3.2+
   - You can follow this guide to install it: https://phoenixnap.com/kb/how-to-install-python-3-centos-7
   - If you prefer not to use SCL, you can follow this guide: https://www.rose-hosting.com/blog/how-to-install-python-3-6-4-on-centos-7/

Installation

Installing the Functions Integration

1. Transfer the Functions for ThreatQ integration (.whl) onto your Resilient Server (or Resilient Integration Server).
2. Change your user to the user that will be running the resilient integrations.

   If you do not have a user created for this, create the user now, or use root (not advised)

3. If not already created, create a `.resilient` folder in your user's home directory:

   ```
   mkdir ~/.resilient
   ```

4. Navigate into that directory:

   ```
   cd ~/.resilient
   ```

5. Create a Python3 virtual environment to house your functions integrations

   You can skip this stepIf you already have a virtual environment setup.

   ```
   python3 -m venv functions-env
   ```

6. Activate the Python3 virtual environment:

```
source functions-env/bin/activate
```

7. Install the .whl file:

Make sure you have the python 3.6 SCL environment activated (if you installed Python using SCL)

```
pip install fn_threatq-<version>-py3-none-
any.whl
```

8. If you have not created a resilient configuration file, do so now

If you run this even though it's already been created, *do not* over-write the old configuration

```
resilient-circuits config -c
```

The default config location is `~/.resilient/app.config`

9. Add the new configuration to the config file by running the following

```
resilient-circuits config -u
```

If you are upgrading from v1.0.x to v1.1.0, see the Updating from v1.0.x to v1.1.0 section.

10. Edit the `app.config` file with your configuration for the integration and Resilient, then save
    - Under the `[resilient]` section, change the following:
        - Comment out `api_key` and `api_key_secret` by prefixing the lines with a `#`.
        - Uncomment out `email` and `password` by removing the `#`.
        - Change the `Org` field to reflect your Resilient Organization name.
        - Uncomment out `cafile` and set the value to `false` if you do not have a certificate file for your Resilient instance. If you do, enter the path here.
    - Under the `[fn_threatq]` section, fill out the following fields:

Field	Description
Host	Your ThreatQ Hostname/IP Address.
Username	Your ThreatQ Username.
Password	Your ThreatQ Password.
CID	Your Client ID (OAuth) found in your ThreatQ profile.
Custom_ Source	Set a custom source name for everything synced to ThreatQ. Default: `Resilient`
Custom_ Attributes	(Optional) A comma-separated, then equals-separated, list of custom Incident fields to map to attributes in ThreatQ (field_name=attribute_name). **Example:** `custom_attributes=internal_severity=Severity, internal_confidence=Confidence`

Field	Description
Custom_ Objects	(Optional) A comma-separated, then equals-separated, list of custom Incident fields to map to obejcts in ThreatQ (field_name=object_name). **Example:** `custom_attributes=internal_severity=Severity, internal_confidence=Confidence` The field name should be the programmatic name associated with the field.

11. Install the customization features (rules, message destinations, fields, etc.)

```
resilient-circuits customize
```

You will be prompted 2 times to hit yes or no (y/n). Hit **y** to install the customizations.

12. Set a location for the lock file

```
export APP_LOCK_
FILE=~/.resilient/functions.lock
```

This will only persist during your current SSH session.

13. Run resilient-circuits to execute the installed integrations

```
resilient-circuits run
```

14. Test out the integration by creating a test incident in Resilient, or updating a currently existing one.

    Notes

    - You should see the integration reacting to actions. If all is working correctly, you can stop the process.
    - If you run into networking or authentication issues, make sure that your `app.config` file is configured correctly.
    - If you are running into any other issues, please contact support@threatq.com.

Configuring Integration

This section will show you how to configure the ThreatQ integration for Resilient

1. Open your `app.config` file (default location: `~/.resilient/app.config`).
2. Find the `[resilient]` section

   This should be located at the very top.

3. Uncomment out and fill in the following fields, if you have not already:

Field	Description
Host	Your Resilient hostname or IP Address.
Port	The port to communicate with the Resilient API. Default: 443
Email	The email to use to authenticate with Resilient.

Field	Description
Password	The password to use to authenticate with Resilient.
Org	The organization name associated with the authenticated user. This is case-sensitive so make sure this is exactly how it is displayed in the UI.

4. Find and uncomment out the `cafile` field.
5. Enter the pathway for the certificate to use with the Resilient API in the `cafile=`field.

Leave the (`cafile=`) field blank if you are not using a certificate.

6. Find the `[fn_threatq]` section

This section will be located closer to the bottom.

7. Fill out the following fields, if you have not already:

Field	Description
Host	Your ThreatQ Hostname/IP Address.
Username	Your ThreatQ Username.
Password	Your ThreatQ Password.
CID	Your Client ID (OAuth) found in your ThreatQ profile.
Custom_ Source	Set a custom source name for everything synced to ThreatQ. Default: `Resilient`

Field	Description
Custom_ Attributes	(Optional) A comma-separated, then equals-separated, list of custom Incident fields to map to attributes in ThreatQ (field_name=attribute_ name). **Example:**`custom_attributes=internal_sever- ity=Severity, internal_confidence=Confidence`
Custom_ Objects	(Optional) A comma-separated, then equals-separated, list of custom Incident fields to map to obejcts in ThreatQ (field_name=object_name). **Example:**`custom_attributes=internal_sever- ity=Severity, internal_confidence=Confidence` The field name should be the programmatic name associated with the field.

Configuring for Reboot

This section will describe how to configure the integration (resilient-circuits) to auto-run on reboot.

1. Create a file for the new service:

```
sudo vi /etc/systemd/system/resilient_circuits_
functions.service
```

2. Paste the following into the file:

Fill out required fields (between < >).

Delete lines 4 and 5 completely (`After=...` and `Required=...`) if you are *not* installing this directly on the Resilient Server.

```
[Unit]
Description=Resilient-Circuits Service for Functions

# Comment out these 2 lines if you are running on an integ-
ration server
After=resilient.service
Requires=resilient.service

[Service]
Type=simple
User=res-integration
WorkingDirectory=/home/<username>
ExecStart=/home/<username>/.resilient/functions-env/bin/re-
silient-circuits run
Restart=always
TimeoutSec=10
Environment=APP_CONFIG_FILE-
E=/home/<username>/.resilient/app.config
Environment=APP_LOCK_
FILE=/home/<username>/.resilient/functions.lock

[Install]
WantedBy=multi-user.target
```

3. Give the service file the correct privileges:

```
sudo chmod 664 /etc/systemd/system/resilient_
circuits_functions.service
```

4. Reload the system daemon and enable the functions service:

```
sudo systemctl daemon-reload

sudo systemctl enable resilient_circuits_
functions.service
```

5. Start the service:

```
sudo systemctl start resilient_circuits_
functions
```

6. You can view the logs for the service by running the following:

```
sudo journalctl -u resilient_circuits_functions
-f
```

This will "follow" the logs in real-time.

Advanced Installation/Usage

See the IBM Integration Server guide if you are attempting an offline installation or updating the configuration file.

https://github.com/ibmresilient/resilient-reference/blob/master/developer_guides/Integration%20Server%20Guide.pdf

Customizing Functionality

By default, the integration includes the following automatic actions:

- Sync Incident
- Sync Indicator
- Sync Task
- Sync Comment

If you do not want those actions to be automated, you can convert them to menu item actions.

Disabling Rules (Actions)

1. Go to **Customization Settings -> Rules**.
2. Find the rule you want to disable.
3. Move the switch to the off position.

Switching to Menu Item Rules

1. Go to `Customization Settings -> Rules`
2. Choose the rule you want to switch to be a menu item

   Make note the name. Usually `ThreatQ:`, followed by the action name - see the Rules section.

3. Delete the rule
4. Create a new rule and then make it a menu item.

5. Enter the name noted in step 2.
6. Set the message destination to `fn_threatq` and then save.

Additionally, two of the actions support some custom fields. If you have converted these automatic actions to menu item actions, see the Rules section to see what fields they support.

- Sync Incident
- Sync Indicator

Rules

Here are the customizations that are installed to the Rules configuration when running the `resilient-circuits customize` command.

Sync Incident

This action will sync an incident with ThreatQ. If the incident is updated, it will update ThreatQ with the new/updated information.

Entry	Details
Rule	Automatic
Type	Incident
Supported Field (If Menu Item)	• Import Artifacts into ThreatQ • Type: Select • Options: [Yes, No] • Import Indicators from ThreatQ • Type: Select • Options: [Yes, No]

Sync Indicator

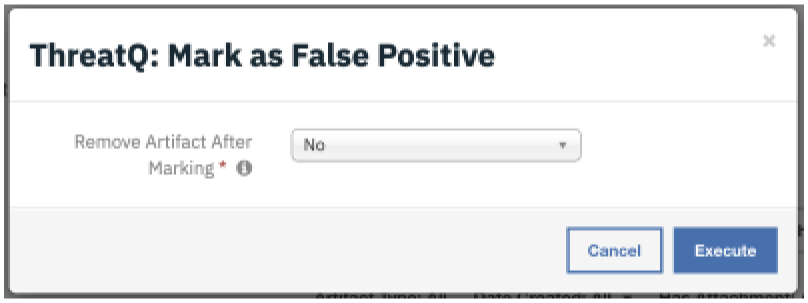
This action will add an artifact to ThreatQ as an indicator.

Entry	Details
Rule	Automatic

Entry	Details
Type	Artifact
Supported Field (If Menu Item)	• Indicator Status • Description: The status for the indicator in ThreatQ • Type: Select • Options: [Active, Review, Indirect, Whitelisted, Expired] • Indicator Confidence • Description: A confidence level for the artifact. This will get set as an attribute within ThreatQ • Type: Select • Options: [Low, Medium, High]

Mark as False Positive

This action will mark the artifact as a false positive within ThreatQ. An attribute will be added to the indicator with the name, **False Positive**, and value, **Yes**.


ThreatQ: Mark as False Positive

Remove Artifact After Marking * : No

Entry	Details
Rule	Menu Item
Type	Artifact

Entry	Details

Fields	• Remove Artifact After Marking
	  ◦ Description: Setting this to 'Yes' will remove the artifact from the Artifact list in Resilient after marking
	  ◦ Type: Select
	  ◦ Options: [Yes, No]

Mark as a True Positive

This action will mark the artifact as a true positive within ThreatQ. An attribute will be added to the indicator with the name, **True Positive**, and value, **Yes**.

Entry	Details
Rule	Menu Item
Type	Artifact

Find Related Indicators

This action will look for any related indicators to the indicator within ThreatQ. Any related indicators will be added to Resilient

Entry	Details
Rule	Menu Item
Type	Artifact

Sync Comment

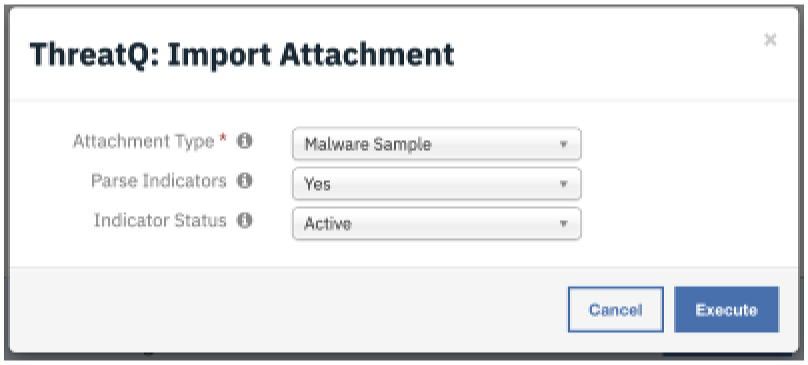
This action will add a note/comment to the associated ThreatQ Incident *or* Task. Comments in ThreatQ do not support markup, so the comment will not maintain the formatting from Resilient..

Entry	Details
Rule	Automatic
Type	Note

- **Rule**: Automatic
- **Type**: Note

Import Attachment

This action will import an attachment into ThreatQ. It gives you the ability to choose what type of attachment it is, as well as the ability to choose whether or not to parse the attachment for indicators.



ThreatQ: Import Attachment

Attachment Type *	Malware Sample
Parse Indicators	Yes
Indicator Status	Active

Entry	Details
Rule	Menu Item
Type	Attachment

Entry	Details

Fields	• Attachment Type
	  ◦ Description: The type of attachment that the attachment will be imported as into ThreatQ
	  ◦ Type: Select
	  ◦ Options: [Malware Sample, Spearphish, PDF, Intel-ligence, Malware Analysis Report, Generic Text]
	• Parse Indicators
	  ◦ Description: Whether or not you want to parse indic-ators out of the attachment
	  ◦ Type: Select
	  ◦ Options: [Yes, No]
	• Indicator Status
	  ◦ Description: The status for any parsed indicators (if enabled)
	  ◦ Type: Select
	  ◦ Options: [Active, Review, Indirect, Whitelisted, Expired]

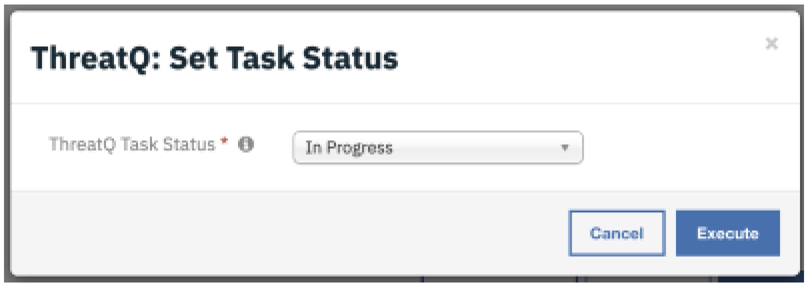
Sync Task

This action will sync a task to ThreatQ. Any updates made to the task will be updated within ThreatQ.

Entry	Details
Rule	Automatic
Type	Task

Set Task Status

This actions allows you to set the task's status within ThreatQ. By default, the integration syncs tasks with the status, "To Do". If you want to mark the task as a different status, you can use this action.

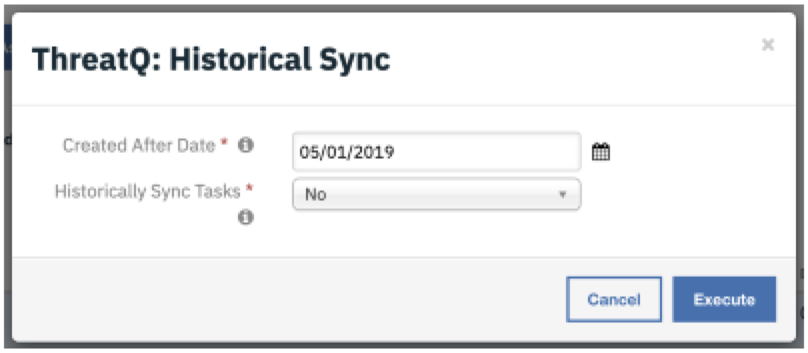


ThreatQ Task Status * In Progress

Entry	Details
Rule	Menu Item
Type	Task
Fields	• Task Status • Description: The status for the task within threatQ • Type: Select • Options: [To Do, In Progress, Review, Done]

Historical Sync

This action allows you to sync incidents historically. You will be able to customize the date to search since, as well as if you want to sync the related tasks or not.



ThreatQ: Historical Sync

Created After Date * 05/01/2019

Historically Sync Tasks * No

Entry	Details
Rule	Menu Item
Type	Incident
Fields	<ul><li>Created After Date<ul><li>Description: The date to go back to sync incidents from</li><li>Type: Date Picker</li></ul></li><li>Historically Sync Tasks<ul><li>Description: Enabling this option will sync related tasks to a historical incident</li><li>Type: Select</li><li>Options: [Yes, No]</li></ul></li></ul>

Functions

Here are the customizations that are installed to the Functions configuration when running "resilient-circuits customize". These functions can be used in Resilient Workflows.

Add Indicator

This function will add an indicator to ThreatQ

Required Parameters:

Parameter	Details
indicator_value	The value for the indicator (artifact.value).
indicator_type	The indicator type as shown in ThreatQ.

If none provided, it will attempted to be parsed.

Optional Parameters:

Parameter	Details
indicator_status	The status of the indicator in ThreatQ.
indicator_confidence	The confidence of the indicator in ThreatQ (added as an attribute).

Find Related Indicators

This function will find all related indicators to the passed indicator and then add them to the Resilient Incident.

Required Parameters:

Parameter	Details
incident_id	The ID for the incident the artifact belongs to (incident.id).
indicator_value	The value for the indicator (artifact.value).

Mark as False Positive

This function will mark an indicator as a false positive in ThreatQ (via an attribute).

Required Parameters:

Parameter	Details
incident_id	The ID for the incident the artifact belongs to (incident.id).
artifact_id	The ID for the artifact (artifact.id.
indicator_value	The value for the indicator (artifact.value).

Optional Parameters:

Parameter	Details
remove_artifact_after_marking	Enabling this will remove the artifact from the Resilient Incident after marking it as a false positive (default: False).

Mark as True Positive

This function will mark an indicator as a true positive in ThreatQ (via an attribute).

Required Parameters:

Parameter	Details
indicator_value	The value for the indicator (artifact.value).

Updating Versions

This section describes any modifications required when upgrading.

Updating from v1.0.0 to v1.0.1

There are a few changes in the v1.0.1 that would call for some manual modifications.

1. In the Resilient UI, remove the deprecated `ThreatQ: Add Indicator` action from the rules and functions (Customization Settings).

   This was replaced with `ThreatQ: Sync Indicator`.

2. In the Resilient UI, remove the deprecated `ThreatQ: Add Comment` action from the rules and functions (Customization Settings).

   This was replaced with `ThreatQ: Sync Comment`.

3. Re-run `resilient-circuits customize` to enable the new actions.

Updating from v1.0.x to v1.1.0

There are a few changes in the v1.0.1 that would call for some manual modifications.

Some functions have been removed due to not fully supporting workflows correctly.

1. In your Customization Settings, go to the `Functions` tab and remove the following items:

- ThreatQ: Sync Incident
- ThreatQ: Sync Indicator
- ThreatQ: Import Attachment
- ThreatQ: Sync Task
- ThreatQ: Historical Sync
- ThreatQ: Set Task Status
- ThreatQ: Sync Comment
- ThreatQ: Find Related Indicators
- ThreatQ: Mark as False Positive
- ThreatQ: Mark as True Positive

2. The following items will be re-added back to your Functions configuration when you rerun `resilient-circuits customize`:
   - ThreatQ: Mark as False Positive
   - ThreatQ: Mark as True Positive
   - ThreatQ: Find Related Indicators

   See Updating from v1.0.0 to v1.0.1 if you are currently on v1.0.0.

3. There are 3 new config items you can add to your `app.config` file under the `[fn_threatq]` section to support custom attribute mapping, custom object mapping, and a custom source
   - `custom_attributes=`
   - `custom_objects=`
   - `custom_source=Resilient`
   - See the *Configuration* section on how to properly set these options
4. Re-run `resilient-circuits customize` (if you did not do this in step 2).

Troubleshooting

The following section describes possible troubleshooting steps.

Fixing a Banned IP Address

On the off-chance that the Resilient server blocks an IP address, whether it be to your ThreatQ instance or to your Resilient Integration Server, follow these instructions to unban it..

1. SSH into your Resilient server using the **resadmin** user.
2. Run the following command to confirm there's an IP banned:

```
sudo -u postgres psql -c "select * from
monapp.ipban;" co3
```

3. Remove the banned IP by running command:

```
sudo -u postgres psql -c "delete from
monapp.ipban;" co3
```

4. Restart Resilient service by command (for RHEL systems):

```
sudo systemctl restart resilient
```

Clearing a Resilient Message Destination

Due to the vast breadth of this integration, it supports syncing many actions executed on the Resilient server. If your team creates and updates a large number of tickets on a daily basis, you may experience a "clog" in the actions pipeline. If this is the case, follow these instructions to clear the message destination queue for the ThreatQ integration.

1. SSH into your Resilient Server or Resilient Integration Server.

   Where ever you are running the integration using resilient-circuits

2. Stop/Kill the ThreatQ Integration with Resilient Functions (resilient-circuits):

   ```
   systemctl stop resilient_circuits_functions
   ```

3. Navigate to your `/tmp` directory.
4. Enable your `functions-env` python virtual environment and install `stomp.py:`

   ```
   source ~/.resilient/functions-env/bin/activate

   pip install stomp.py==4.1.18
   ```

5. Create a python script called `resilient_ack_all_messages.py`.
6. Paste the following code into the python file:

   resilient_ack_all_messages.py

   ```
   """ Subscribe to specific message destinations and ack all messages """

   from __future__ import print_function

   import sys
   import ssl
   import json
   ```

```
import time
import logging
import stomp
import co3
from requests.utils import DEFAULT_CA_BUNDLE_PATH
logging.getLogger("stomp.py").setLevel(logging.ERROR)


class Co3Listener(object):
       """The stomp library will call our listener's on_message
when a message is received."""

       def __init__(self, conn):
               self.conn = conn

       def on_error(self, headers, message):
               """Report an error from the message queues"""
               print('received an error {}'.format(message))

       def on_message(self, headers, message):
               """Handle a message"""
               if message == "SHUTDOWN":
                       print("Shutting down")
                       self.conn.disconnect()
                       sys.exit(0)

       # Convert from a JSON string to a Python dict object.
       json_obj = json.loads(message)
```

```
        # Pull out incident object.
        inc = json_obj['incident']

        print('received action for incident {}:  name={};'.format
(inc['id'], inc['name']))

        # Get the relevant headers.
        reply_to = headers['reply-to']
        correlation_id = headers['correlation-id']
        context = headers['Co3ContextToken']

        # send the reply back to the Resilient server indicating
that everything was OK.
        reply_message = '{"message_type": 0, "message": "Pro-
cessing complete", "complete": true}'

        self.conn.send(reply_to, reply_message, headers={'cor-
relation-id': correlation_id})


def validate_cert(cert, hostname):
        try:
                co3.match_hostname(cert, hostname)
        except Exception as ce:
                return (False, str(ce))

        return (True, "Success")
```

```
def subscribe_to_specific_destination(conn, client, des-
tination):
       print("Subscribing to the destination: ", destination)
       conn.subscribe(id='stomp_listener',
               destination="actions.{0}.{1}".format(client.org_id,
                               destination),
               ack='auto')


def main():
# Parse out the command line options.
######
#
# python2 ack_specific_destinations.py --email user-
@example.com --password Pass4Admin --host app.re-
silientsystems.com --org TestOrg --cafile false --
destination test_queue
#
######
parser = co3.ArgumentParser()
parser.add_argument("--destination",
               help="Message destination to subscribe",
               default="",
               required=True)
opts = parser.parse_args()
```

```
host_port = (opts.host, 65001)

# Setup the STOMP connection.
conn = stomp.Connection(host_and_ports=[host_port], try_
loopback_connect=False)

# Configure a listener.
conn.set_listener('', Co3Listener(conn))

# Give the STOMP library our TLS/SSL configuration.
validator_function = validate_cert
cafile = opts.cafile
if cafile == "false":
       # Explicitly disable TLS certificate validation, if you
need to
       cafile = None
       validator_function = None
       print("TLS without certificate validation: Insecure!
(cafile=false)")
elif cafile is None:
       # Since the REST API (co3 library) uses 'requests', let's
use its default certificate bundle
       # instead of the certificates from ssl.get_default_
verify_paths().cafile
       cafile = DEFAULT_CA_BUNDLE_PATH
       print("TLS validation with default certificate file:
{0}".format(cafile))
else:
       print("TLS validation with certificate file: {0}".format
(cafile))
conn.set_ssl(for_hosts=[host_port],
               ca_certs=cafile,
               ssl_version=ssl.PROTOCOL_TLSv1,
               cert_validator=validator_function)

client = co3.SimpleClient(org_name=opts.org, base_url-
l="https://" + opts.host, verify=cafile or False)
session = client.connect(opts.email, opts.password)

conn.start()

# Actually connect.
conn.connect(login=opts.email, passcode=opts.password)

# Subscribe to the destinations.
subscribe_to_specific_destination(conn, client, opts.des-
tination)

# The listener gets called asynchronously, so we need to
sleep here.
while 1:
       time.sleep(2)



if __name__ == "__main__":
       logging.basicConfig(level=logging.INFO)
```

```
        main()
```

7. Save the file, then open up a screen session:

   ```
   screen -S clear-tq-messages
   ```

8. Run the following command (with substitutions for your email, password, hostname, and organization):

   ```
   python resilient_ack_all_messages.py –email <Resilient Email> --password <Resilient Password> --host <Resilient Host> --org <Organization Name> --cafile false --destination fn_threatq
   ```

9. Replace the "<...>" tags with the correct authentication information for your Resilient instance
10. The script should connect to the ThreatQ message destination, and acknowledge all of the messages.
11. Wait until the script stops processing messages (when there is none left in the queue), before proceeding.
12. Open up the Resilient UI and navigate to the Customization Settings, then the Rules tab.
13. For the sync commands (All rules starting with "ThreatQ: Sync...", apply conditions for the following (at minimum).

    You can skip this step if you are using version 1.1.0+.

    - Created
    - Deleted/Removed

- Description → is changed

> You can add as many conditions as needed.

14. Head back into your SSH session and restart the ThreatQ integration:

```
systemctl start resilient_circuits_functions
```