ThreatQ SDK User Guide

Version 1.7.0

Warning and Disclaimer

Last Updated: Friday, February 2, 2018

Contents

Introduction

The purpose of this guide is to provide some basic examples for using the ThreatQ SDK.

Installing Pip

Before you install the SDK, you must first install Pip. If you have not already installed Pip on your system, open a terminal window and run the following command:

```
sudo easy_install pip
```

Upon success, you should see output similar the following:

```
Searching for pip
Best match: pip 8.0.2
Adding pip 8.0.2 to easy-install.pth file
Installing pip script to /usr/local/bin
Installing pip2.7 script to /usr/local/bin
Installing pip2 script to /usr/local/bin

Using /usr/local/lib/python2.7/site-packages
Processing dependencies for pip
Finished processing dependencies for pip
```

Installing the ThreatQ SDK

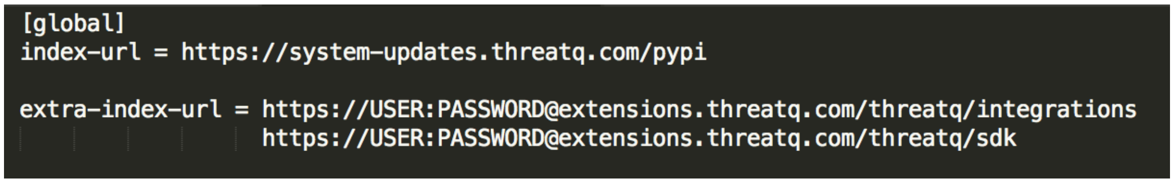
1. Open a terminal window.
2. Create a directory called **~/.pip/** and create a file in that directory called **pip.conf**. Run the following command:

```
mkdir ~/.pip; touch ~/.pip/pip.conf
```

3. Open the **pip.conf** file in a text editor, by running the following command:

```
open -t pip.conf
```

4. In the text editor, enter the following:



```
[global]
index-url = https://system-updates.threatq.com/pypi

extra-index-url = https://USER:PASSWORD@extensions.threatq.com/threatq/integrations
                  https://USER:PASSWORD@extensions.threatq.com/threatq/sdk
```

5. Save and close **pip.conf**.
6. In the terminal window, run the following command:

```
sudo pip install threatqsdk
```

Authentication

Before using the SDK, import the base `Threatq` object. This is required to interact with the ThreatQ API.

```
from threatqsdk import Threatq
```

Next, authenticate to the API, replacing all values with your specific details.

```
tq_host = 'https://localhost:8443'
username = 'threatq@threatq.com'
password = 'threatquotientthreatquotient'
clientid = '< OAUTH TOKEN >'

tq = Threatq(tq_host, {'clientid': clientid, 'auth': {
```

```
                        'email': username, 'password': pass-
word}})
```

Working with Indicators

The following provides several examples of working with indicators.

List All Indicators

To list all the indicators in ThreatQ, you can use the base `tq.get()` method to call API endpoints. This method makes an HTTP GET request, wrapping authentication against the API.

In this example, use the `/api/indicators` endpoint. This will return a `list` of `dict` representations of an indicator. If you print the first element of the `list`, you will see the following data returned by the API.

```
inds = tq.get('/api/indicators')
print inds.get('data')[0]
```

```
{
  "last_detected_at": None,
  "hash": "51d81f46d7a042805c96e512a3e122ba",
  "status_id": 1,
  "created_at": "2016-10-13 14:07:56",
  "type_id": 10,
  "updated_at": "2016-10-13 14:07:56",
  "value": "1.234.62.166",
  "id": 1,
  "class": "network"
}
```

Search for a Specific Indicator

To search for a specific indicator, the base `tq.get()` method accepts a `params` parameter where you can specify an indicator value. This returns the same `dict` representation of an indicator as above.

```
ind = tq.get('/api/indicators', params={'value': '8.8.8.8'})
print ind.get('data')
[
  {
    "last_detected_at": null,
    "hash": "9c709bf480caf30fc107cfbbc107cfbb",
    "status_id": 1,
    "created_at": "2016-10-14 00:02:18",
    "type_id": 10,
    "updated_at": "2016-12-02 09:13:14",
    "value": "8.8.8.8",
    "id": 535253,
    "class": "network"
  }
]
```

Create a New Indicator

To create an indicator, you must import the `Indicator` and `Source` objects.

```
from threatqsdk import Indicator, Source
```

Next, to create a basic indicator, set the required values:

- value
- type
- status

```
ind = Indicator(tq)
ind.set_value('example.com')
ind.set_type('FQDN')
ind.set_status('Review')
```

Finally, upload the indicator and receive the new indicator ID

```
iid = ind.upload(sources=Source('Test'))
```

Add an Attribute

To add an attribute key/value pair to the indicator you created above:

```
ind.add_attribute('Disposition', 'Safe', sources=Source
('Test'))
```

Update an Indicator's Status

To update an indicator's status, you can utilize the base `tq.put()` method to make an HTTP PUT request, wrapping authentication against the API. To modify an indicator's status, you will need its indicator ID to use the `/api/indicators/INDICATOR_ID` endpoint.

In this example, you will modify the indicator you created above, changing its status from "Review" (set during creation) to "Active." This example will apply as long as `iid` is a valid indicator ID.

```
tq.put('/api/indicators/{}'.format(iid), data={'status': 'Act-
ive'})
```

Get Related/Linked Objects

To retrieve related or linked objects for an indicator, you can use the `get_related_objects()` function. It takes the type of object (`Indicator, Event, Adversary, File, etc.`) as its only argument.

Note: You will need to make sure you have previously imported the object first.

The `get_related_object()` function is also available to the `Event, Adversary, File,` and `Signature` objects.

In this example, we have an `Indicator` object, `ind`, that we want to retrieve all of its related `Indicator` and `Adversary` objects. The result will be a list of `Indicator` and `Adversary` objects respectively.

```
rel_inds = ind.get_related_objects(Indicator)
```

```
rel_advs = ind.get_related_objects(Adversary)
```

Relate Link Objects

To relate or link an indicator with another object, you can use the `relate_object()` function. It takes a separate instance of an object (`Indicator, Event, Adversary, File, etc.`) as its only argument.

The `relate_object()` function is also available to the `Event, Adversary, File`, and `Signature` objects.

In this example, we have two `Indicator` objects, `ind_a` and `ind_b`, that we want to relate or link together.

```
ind_a.relate_object(ind_b)
```

Bulk Uploading Indicators

In most use-cases, you want to upload a large number of indicators at one time. To do this via the SDK, you can use the `BulkIndicator` object and `tq.bulkuploadindicators()` method.

First, import the `BulkIndicator` and `Source` objects:

```
from threatqsdk import BulkIndicator, Source
```

Let's assume that you have a `list` of IOC data you want to parse and upload to ThreatQ. You must translate each into a `BulkIndicator` and then add them to a new list to be uploaded: `bulk_indicators`.

First, create a new `bulk_indicators list`:

```
bulk_indicators = []
```

Next, create a `BulkIndicator` object for each IOC we want to upload. The required values that need to be set are:

- ind_value
- ind_type
- ind_status

```
bi = BulkIndicator(tq)
bi.set_value(ind_value)
bi.set_type(ind_type)
bi.set_status(ind_status)
```

You can also add attributes and relate to other ThreatQ objects:

```
bi.add_attribute('Foo', 'Bar')
bi.relate_indicator('example.com', 'FQDN')
bi.relate_adversary(adversary_id)
bi.relate_event(event_id)
```

You would repeat/iterate the above over each item in your IOC `list` (`for` loop) and append each to `bulk_indicators`:

```
bulk_indicators.append(bi)
```

Lastly, upload the `bulk_indicators` using the `tq.bulkuploadindicators()` method:

```
tq.bulkuploadindicators(bulk_indicators, source=Source('Test')
```

Working with Events

The following provides several examples of working with events.

- List All Indicators
- Search for a Specific Event
- Create a New Event

List All Events

To list all the events in ThreatQ, you can use the base `tq.get()` method against the `/api/events` endpoint. This will return a `list` of `dist` representations of an event. If you print the first element of the list, you can see the data returned by the API.

```
events = tq.get('/api/events')
print events.get('data')[0]
```

```
{
  "hash": "3ebe478a05e4a7981f94dfcfab31ee14",
  "description": "Desc for Internal Domain Controller Com-
promised",
  "title": "Internal Domain Controller Compromised",
  "created_at": "2016-10-21 11:43:37",
  "type_id": 5,
  "updated_at": "2016-10-21 11:43:37",
  "happened_at": "2016-10-21 11:43:35",
  "id": 2
}
```

Search for a Specific Event

To search for a specific event, pass the `title` to the `params` parameter. This will return the same `dict` representation of an event like above.

```
event = tq.get('/api/events', params={'title': 'Internal
Domain Controller Compromised'})
print event.get('data')
```

```
[
    {
        "hash": "3ebe478a05e4a7981f94dfcfab31ee14",
        "description": "Desc for Internal Domain Controller
Compromised",
        "title": "Internal Domain Controller Compromised",
        "created_at": "2016-10-21 11:43:37",
        "type_id": 5,
        "updated_at": "2016-10-21 11:43:37",
        "happened_at": "2016-10-21 11:43:35",
        "id": 2
    }
]
```

Create a New Event

To create an event, you must import the `Event` and `Source` objects.

```
from threatqsdk import Event, Source
```

Next, to create a basic event, set some required values:

- title
- type
- date

Optionally, you can also set a `description`.

```
event = Event(tq)
event.set_title('OMG MALWARE')
event.set_type('Incident')
event.set_date('2017-01-13 10:59:00')
event.set_desc('Foo')
```

Finally, upload the event and receive the new event ID

```
eid = event.upload(sources='Test')
```

To add an attribute key/value pair to the event we created above:

```
event.add_attribute('Severity', 'High', sources='Test')
```

Working with Adversaries

The following provides several examples of working with adversaries.

- List All Adversaries
- Search for a Specific Adversary
- Create a New Adversary
- Add an Attribute

List All Adversaries

To list all the adversaries in ThreatQ, you can use the base `tq.get` method against the `api/adversaries` endpoint. This will return a `list` of `dict` representations of an adversary. If we print the first element of the `list`, we can see the data returned by the API.

```
adversaries = tq.get('/api/adversaries')
print adversaries.get('data')[0]
```

```
{
    "updated_at": "2017-10-03 14:30:53",
    "touched_at": "2017-10-03 14:31:04",
    "created_at": "2017-10-03 14:30:53",
    "id": 2,
    "name": "Comment Panda"
}
```

Search for a Specific Adversary

To search for a specific adversary, pass the `name` to the `params` parameter. This will return the same `dict` representation of an adversary as above.

```
adversary = tq.get('/api/adversaries' params={'name': 'PLA
Unit 61398'})
print adversary.get('data')
```

```
[
    {
        "updated_at": "2017-10-03 14:30:54",
        "touched_at": "2017-10-03 14:31:04",
        "created_at": "2017-10-03 14:30:54",
        "id": 3,
        "name": "PLA Unit 61398"
    }
]
```

The SDK also has a `search` function for `Adversary` objects. Instead of returning the raw response from the API, the SDK will translate it to an `Adversary` object. Below, perform the same search as above, but instead of a `dict` object, we are now working with an `Adversary` object.

```
from threatqsdk import Adversary
adv = Adversary(tq)
aid = adv.search('PLA Unit 61398')
print aid
```

```
3
```

Create a New Adversary

To create an adversary, you must import the `Adversary` and `Source` objects.

```
from threatqsdk import Adversary, Source
```

Next, to create a basic adversary, set the required name attribute. You can also set a description.

```
adv = Adversary(tq)
adv.name = 'APT 99'
adv.description = 'Malicious attack group'
```

Finally, upload the adversary and receive the new adversary ID

```
aid = adv.upload(sources=Source('Test'))
```

Add an Attribute

To add an attribute key/value pair to the iadversary you created above

```
adv.add_attribute('Vertical', 'Hospitality', sources=Source
('Test'))
```

Working with Files

The following provides several examples of working with files.

- Upload a New File
- Parse and Import Indicators from a File

Upload a New File

To create a file, you must import the `File` and `Source` objects.

```
from threatqsdk import File, Source
```

Next, to create a basic file, set the required values:

- name
- ftype
- path

Optionally, you can also set a `title`.

```
file = File(tq)
file.name = 'my-intel-report'
file.ftype = 'Intelligence Report'
file.path = '~/report.pdf'
file.title = 'My Threat Report'
```

Finally, upload the file. The SDK will translate the API response and update the `File` object with the new file ID.

Note: This behavior differs from other objects.

```
file.upload(sources=Source('Test'))
print file.fid
```

```
1
```

Parse and Import Indicators from a File

At times, files contain indicator values you may want to parse and add to your Threat Library. The SDK allows for this use case and only requires that a `File` be created and uploaded first before being parsed.

In this example, let's assume that a text file was uploaded and has a file ID of 2. To parse all the indicators, save them as *Active* and with the source *Test Source* The method below uses the default *Generic Text* parser.

```
file = File(tq)
file.fid = 2
file.parse_and_import('Test Source', status='Active')
```

Working with Signatures

The following provides an example of working with signatures.

Create a New Signature

To create a signature, you will first you will need to import the `Signature` and `Source` objects.

```
from threatqsdk import Signature, Source
```

Next, to create a basic signature, set the required values:

- `value`
- `type`
- `status`

```
signature_value = 'alert tcp $HOME_NET 666 -> 1.1.1.1 any
(msg:"MALWARE-BACKDOOR SatansBackdoor.2.0.Beta"; flow:to_cli-
ent,established; content:"Remote|3A| "; depth:11; nocase; con-
tent:"You are connected to me.|0D 0A|Remote|3A| Ready for
commands"; distance:0; nocase; metadata:ruleset community; ref-
erence:url,www.-
megasecurity.org/trojans/s/satanzbackdoor/SBD2.0b.html;
ref-
erence:url,www3.ca.com/securityadvisor/pest/pest.aspx?id=5260;
classtype:trojan-activity; sid:118; rev:12;)'

sig = Signature(tq)
```

```
sig.set_value(signature_value)
sig.set_type('Snort')
sig.set_status('Review')
```

Finally, upload the signature and receive the new signature ID

```
sid = sig.upload(sources=Source('Test'))
```

Advanced Search

To perform an Advanced Search (legacy), you will first you will need to import the AdvancedSearch object.

```
from threatqsdk import AdvancedSearch
```

Next, create a dict of your query parameters. You can copy the request being made by the UI using Developer Tools in your browser. In this example, we are searching for all indicators with the source of Malware Domain List.

```
query_params = {'indicators': [[{'field': 'source',
'operator': 'is', 'value': 'Malware Domain List'}]]}
```

Next, perform the search using the query parameters. The `execute()` function returns an generator of `Indicator` objects. This makes lopping through the results efficient.

```
adv_search = AdvancedSearch(tq, query_params)
search_results = adv_search.execute()
```

You can then iterate over each item in the search results. Each `Indicator` result will have their `value`, `iid`, `typename`, and `statusname` set. Below, we iterate through the results and print each indicator value.

```
for ind in search_results:
   print ind.value

ec2-54-72-9-51.eu-west-1.compute.amazonaws.com
54.72.9.51
ssl-6582datamanager.de
104.31.75.147
60.250.76.52
down.mykings.pw
down.mykings.pw:8888/ups.rar
alegroup.info
```

If instead you want to create a list of the search results, you can do the following:

```
search_results = adv_search.execute()
new_search_results = [r for r in search_results]
```

Working with Operations

The following provides several examples of working with operations.

- Import the Operation Object
- List Enabled Operations and their Actions
- Perform an Operation Action

Import the Operation Object

To interact with Operations, you will first you will need to import the `Operation` object.

```
from threatqsdk import Operation
```

List Enabled Operations and their Actions

To list the enabled Operations, you can use the `list_from_tq()` function. This is a class method and will return a list of `Operation` objects.

```
ops = Operation.list_from_tq(tq)
```

To iterate over each resulting Operation and print their actions, you can run the following:

```
for o in ops:
    print o.name
    for a in o.actions:
        print '{}: {}'.format(a['name'], a['description'])
    print '\n'
```

This will print something similar to:

```
passive_total
get_passive_dns: Retrieve Passive DNS associated with Indic-
ators
get_WHOIS: Get WHOIS
enrich: Enrichment
get_samples: Get Malware Samples
query_for_registered_domains: Searches WHOIS data by Email
Address to return all domains registered to that Email Address


vulners
search_CVE: Query CVE against Vulners DB
```

Perform an Operation Action

To perform a specific Operation action, first create a new instance of the Operation object with your intended Operation name (`friendly_name`) as the second argument. In this case, we will be leveraging the **PassiveTotal** Operation.

```
op = Operation(tq, 'passive_total')
```

Next, you can use the `execute()` function. It takes the following arguments:

- `action name`
- `ThreatQ object ID`
- `ThreatQ object type`

In this example, we are running the *Get WHOIS* (`get_WHOIS`) action against and *Indicator* of ID *43*.

```
iid = 43
resp = op.execute('get_WHOIS', iid, 'indicator')
```

The `execute()` function will return the resulting data in JSON format.

```
{
    "indicators": [
        {
            "type": "Email Address",
            "value": "abuse@godaddy.com"
        },
        {
            "type": "FQDN",
            "value": "ns11.domaincontrol.com"
        },
        {
            "type": "FQDN",
            "value": "ns12.domaincontrol.com"
        },
        {
            "type": "FQDN",
            "value": "whois.godaddy.com"
        }
    ],
    "attributes": [
        {
            "name": "Registrant Contact Name",
            "value": "******** ********"
        },
        {
            "name": "Registrar",
            "value": "GoDaddy.com, LLC"
        },
        {
            "name": "Updated At",
            "value": "May 23 2017 11:52:46 AM "
        },
        {
            "name": "Registered Date",
            "value": "May 22 2014 05:11:26 PM "
        },
        {
            "name": "Expires At",
            "value": "May 22 2018 05:11:26 PM "
        }
    ],
    "raw_data": {
        "contactEmail": "abuse@godaddy.com",
        "whoisServer": "whois.godaddy.com",
        "name": "******** ********",
        "billing": [],
        "nameServers": [
            "ns11.domaincontrol.com",
            "ns12.domaincontrol.com"
        ],
        "registered": "2014-05-22T17:11:26.000-0700",
        "lastLoadedAt": "2017-12-19T11:13:18.419-0800",
        "telephone": "N/A",
        "registryUpdatedAt": "2017-05-23T11:52:46.000-0700",
        "admin": [],
        "expiresAt": "2018-05-22T17:11:26.000-0700",
        "tech": [],
        "registrar": "GoDaddy.com, LLC",
        "domain": "aadroid.net",
        "organization": "N/A",
        "zone": [],
        "registrant": {
            "name": "******** ********",
            "email": "abuse@godaddy.com"
        }
    }
}
```

Working with Feeds

The following provides several examples of working with feeds.

- Import the Feed Object
- Retrieve a Specific Feed by Name
- Retrieve a Specific Feed by ID
- Enable and Disable a Feed

Import the Feed Object

To interact with Incoming Feeds, you will first you will need to import the `Feed` object.

```
from threatqsdk import Feed
```

Retrieve a Specific Feed by Name

To retrieve the settings for a specific feed, querying by name, you will first need to create a new instance of a `Feed` object and use the `by_name()` function, which takes the feed name as its only parameter.

```
f = Feed(tq)
f.by_name('Bambenek Consulting - Murofet Master')
```

This will fill the properties of the Feed object you created (in the example above, `f`). Feed objects have the following properties:

- category
- gate_oauth2_client_id
- name
- connector_definition_id
- updated_at
- is_active
- created_at
- namespace
- last_import_at
- last_import_count
- frequency
- tlp_id
- indicator_status_id
- category_id
- id
- custom_fields

Retrieve a Specific Feed by ID

Similar to above, if you want to query by feed ID instead of name, you will first need to create a new instance of a Feed object and use the `by_id()` function, which takes the feed ID as its only parameter.

```
f = Feed(tq)
f.by_id(1)
```

Enable and Disable a Feed

To enable a feed, you can use the enable() function.

```
f.enable()
```

Likewise, to disable a feed, you can use the disable() function.

```
f.disable()
```