ThreatQ SDK User Guide

Version 1.6.7

Warning and Disclaimer

ThreatQuotient, Inc. provides this document “as is”, without representation or warranty of any kind, express or implied, including without limitation any warranty concerning the accuracy, adequacy, or completeness of such information contained herein. ThreatQuotient, Inc. does not assume responsibility for the use or inability to use the software product as a result of providing this information.

Last Updated: Saturday, October 28, 2017

Contents

Introduction

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

Installation

Run the following command to install the SDK.

```
$ pip install threatqsdk-1.6.7-py2-none-any.whl
```

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': 'Active'})
```

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 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

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

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')
```