ThreatQuotient

A Securonix Company

ThreatQ SDK User Guide

Version 1.8.11

August 07, 2025

ThreatQuotient
20130 Lakeview Center Plaza Suite 400
Ashburn, VA 20147

ThreatQ Supported

Support
Email: support@threatq.com
Web: support.threatq.com
Phone: 703.574.9893

Contents

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.

Support

This integration is designated as **ThreatQ Supported**.

Support Email: support@threatq.com
Support Web: https://support.threatq.com
Support Phone: 703.574.9893

Integrations/apps/add-ons designated as **ThreatQ Supported** are fully supported by ThreatQuotient's Customer Support team.

ThreatQuotient strives to ensure all ThreatQ Supported integrations will work with the current version of ThreatQuotient software at the time of initial publishing. This applies for both Hosted instance and Non-Hosted instance customers.

> ThreatQuotient does not provide support or maintenance for integrations, apps, or add-ons published by any party other than ThreatQuotient, including third-party developers.

Introduction

The ThreatQ SDK provides methods for accessing the ThreatQ API to extract and customize data. The ThreatQ SDK User Guide provides 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
```

> Pip installation is not required on a Windows OS.

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 on a Mac OS

After you install Pip, as described in Installing Pip, complete the following steps to install 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/pypis
   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
   ```

Installing the ThreatQ SDK on a Windows OS

Complete the following steps to install the ThreatQ SDK.

1. Open a Command Prompt window
2. Navigate to C:\Python27\Scripts:

   ```
   cd c:\Python27\Scripts
   ```

3. Run the following command:

   ```
   pip install -i https://[your yum/repo credentials]:[your yum/repo
   password]@extensions.threatq.com/threatq/sdk threatqsdk
   ```

Authentication

First, import the base `Threatq` object. This will be required to interact with the ThreatQ API.

```
from threatqsdk import Threatq
```

Next, authenticate to the API, replacing all values with your specific details. There are two authentication methods: via username/password and by OAuth Client Credentials.

Username/Password Method

```
tq_host = 'https://localhost:8443'
username = 'threatq@threatq.com'
password = 'threatquotientthreatquotient'
clientid = '< OAuth Client ID >'
tq = Threatq(tq_host, {'clientid': clientid, 'auth': {
                       'email': username, 'password': password}})
```

OAuth Client Credentials Method

> ⚠ You must use the flag **True** in the final line, as shown below. Failing to use this flag will result in the application defaulting to the username/password method.

```
tq_host = '< Your ThreatQ Instance IP >'
clientID = '< OAuth Client ID >'
clientsecretID = '< OAuth Client Secret >'
tq = Threatq(host,(clientID,clientsecretID),True)
```

OAuth Registration Command

You can run a command that allows registering a new private OAuth2 client to be used by custom integrations. The Client ID/Secret generated by the command can be used by custom integrations to interact with the API but can't be used to log into the UI.

1. SSH to your ThreatQ installation.
2. Create a new client id and client secret password using the following command:

   ```
   kubectl exec --namespace threatq --stdin --tty deployment/api-
   schedule-run -- ./artisan threatq:oauth2-client --name="Custom
   Integration"
   ```

   You should see output for the new custom integration user:

   ```
   session_timeout_minutes: 1440
   name: Custom Integration
   type: private
   client_id: ntdjzwe3mduyyjqxyjdiyza5mzyxmtkx
   client_secret:
   YThlOTBlZjM0YTYxNWM1YjVkODdmMTdjNGY5MzZkYTg4M2RmYmRiZGJmNjk1OTRm
   updated_at: 2020-01-14 14:03:27
   created_at: 2020-01-14 14:03:27
   ```

1. SSH to your ThreatQ installation.
2. Navigate to the api directory using the following command:

   ```
   cd /var/www/api
   ```

3. Create a new client id and client secret password using the following command:

   ```
   php artisan threatq:oauth2-client --name="Custom Integration"
   ```

   You should see output for the new custom integration user:

   ```
   session_timeout_minutes: 1440
   name: Custom Integration
   type: private
   client_id: ntdjzwe3mduyyjqxyjdiyza5mzyxmtkx
   client_secret:
   YThlOTBlZjM0YTYxNWM1YjVkODdmMTdjNGY5MzZkYTg4M2RmYmRiZGJmNjk1OTRm
   updated_at: 2020-01-14 14:03:27
   created_at: 2020-01-14 14:03:27
   ```

Flag Options

There are flag options for `type` and `user-groups`.

Type

The default type is private. The ThreatQ UI uses a FE/UI specific to Client ID to get /request access tokens. Further authentication is required to be sent with the client that is trying to access the API for the command.

The are two options for the type flag:

- `private` - Private Client ID/Secret
- `public` - Client ID without the secret

Flag example:

--type private

User-Groups

All clients, users, and connectors are associated with groups.

There are three options for the user-groups:

- `admin` - can access everything
- `analyst` - can access most threat intel
  Some configuration endpoints are not accessible to analysts.
- `observer` - read-only access

Flag example:

--user_group admin

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, we will use the `/api/indicators` endpoint. This will return a `list` of `dict` representations of an indicator. If we print the first element of the `list`, we can see the 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 Specific Indicator

To search for a specific indicator, the base `tq.get()` method accepts a `params` parameter where we can specify an indicator value. This will return the same `dict` representation of an indicator like 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 will first you will need to 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'))
```

Adding an Attribute

To add an attribute key/value pair to the indicator we 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 we will need its indicator ID and we will use the `/api/indicators/INDICATOR_ID` endpoint.

In this example, we will modify the indicator we created above, changing its status from "Review" (we set this 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'})
```

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 we have a `list` of IOC data we want to parse and upload to ThreatQ. We will need to first translate each into a `BulkIndicator` and then add them to a new `list` to be uploaded: `bulk_indicators`.

First, let's create our new `bulk_indicators` `list`:

```
bulk_indicators = []
```

Next, create a `BulkIndicator` object for each IOC we want to upload. The required values that are needed 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
- 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 `dict` representations of an event. If we print the first element of the `list`, we 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 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
}
```

Search for 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 will first you will need to 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 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 like 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, we 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 will first you will need to 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'))
```

Adding an Attribute

To add an attribute key/value pair to the indicator we 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 will first you will need to 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 is different behavior from other objects**

```
file.upload(sources=Source('Test'))
print file.fid
1
```

Parse and Import the Indicators from a File

Sometimes files contain indicator values we may want to parse and add to our Threat Library. The SDK allows for this use case and only requires that a `File` be created and uploaded first before being able to be parsed.

In this example, let's assume that a text file was uploaded and has a file ID of **2**. We want to parse all the indicators, save them as *Active* and with the source *Test Source*. The below method use 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_client,established; content:"Remote|
3A| "; depth:11; nocase; content:"You are connected to me.|0D 0A|Remote|3A|
Ready for commands"; distance:0; nocase; metadata:ruleset community;
reference:url,www.megasecurity.org/trojans/s/satanzbackdoor/SBD2.0b.html;
reference: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'))
```

Threat Library Search

To perform a Threat Library Search you will first you will need to import the `ThreatLibrary` object.

```
from threatqsdk import ThreatLibrary
search = ThreatLibrary(tq) # tq is an instance of the Threatq object
```

Next, you will need to decide whether you want to construct your own API query, or use a saved search name to execute a search. If you decide the former, you will need to understand the query format. The easiest way to learn about the format is to use the "Network" developer tools in your browser to analyze the searches that you make. Below is an example that will search for `all objects from the source, 'Analyst', and has an attribute named, 'Confidence' with a value of 'High`:

```
query = {'+and': [{'source_name': 'Analyst'}, {'attribute': {'name':
'Confidence', 'value': 'High'}]}
```

If you decide the latter, and want to use a Threat Library Saved Search/Collection, you first need to create the saved search in the Threat Library. In this example, we'll name the saved search/collection `High Confidence Indicators`. Second, you simply need to reference that name within your code, like so:

```
search.get_saved_search('High Confidence Indicators')
```

What the above snippet will do is fetch the saved search/collection, including the search query and other metadata surrounding the saved search/collection such as name, hash, etc.

Next, you will want to execute the actual search in order to get the results. The `execute()` method is a generator, meaning it yields incremental results. By default, the method yields one object at a time. You will need to iterate over the call to get the results, like so:

```
# If you used a custom query
for indicator in search.execute('indicators', custom_query=query,
page_limit=100):
# If you fetched a saved search
for indicator in search.execute('indicators', page_limit=100):
```

The above code tells the SDK to fetch all "indicators" based on the query, and limit the results to 100 items per request. You can change the actual object type by modifying "indicators" string to the name of the object you want.

Here is a full list of parameters that the `execute()` function supports:

- object_type (str): The name of the object you want to fetch
- custom_query (dict): A custom query to use (if you aren't using a saved search) – Optional
- page_limit (int): The number of items to return per-request (default: 1000)
- page_offset (int): The offset to start at (default: 0)
- max_results (int): The maximum amount of results to return (default: None [unlimited])
    - This is only necessary for a use-case where you may only need 'x' amount of results, max.
- yield_batches (bool): Determines whether to yield batches of indicators (by page limit). This forces the method to yield all the results per-request, rather than one at a time (default: False)

- fields (list): List of strings specifying which fields to return from the API (default: All fields)
    - Using this field is optional, but will greatly reduce the memory footprint of your requests by requesting only the information you need.

Below is a full example of using the ThreatLibrary class to execute a saved search, transform the data, and upload it in batches to a given API:

```
from threatqsdk import ThreatLibrary
search = ThreatLibrary(tq) # tq is an instance of the Threatq object
search.get_saved_search('High Confidence Indicators')
uploaded = 0
for data in search.execute('indicators', page_limit=100, yield_batches=True,
fields=['value']:
   to_upload = []
    for indicator in data:
        to_upload.append(convert_to_payload(indicator))
    print('Uploading batch of {} indicators'.format(len(to_upload))
    upload_bundle(to_upload)
```

In addition to executing saved searches, you can also create a named or unnamed saved search. The difference is, a named saved search will be available via the UI. An unnamed saved search will create a "hash" for the saved search, which will allow you to reference it via a URL. Though, it will not create a saved search in the UI.

You can create a saved search executing the `create_search()` method. Here is an example for creating a named saved search that looks for keywords:

```
from threatqsdk import ThreatLibrary
search = ThreatLibrary(tq) # tq is an instance of the Threatq object
search.create_search(
    name='Ransomware Search'
    keywords=[
        'NotPetya',
        'CryptoLocker',
        'WannaCry'
    ]
)
```

You can then execute that newly created search by running the `execute()` function shown in the previous example.

Here is a full list of parameters that the `create_search()` function supports:

- name (str): The name of the saved search – Optional
- keywords (list): List of keywords to search for (OR logic) - Optional
- api_query (dict): If you have a literal saved search API query that you'd built, you can specify that query here, to be saved - Optional

Working with Operations

The following provides several examples of working with operations.

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 Indicators
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()
```

Change Log

- **Version 1.8.11**
  - Removed `refresh_token` usage to prevent `client_credential` auth refresh.
- **Version 1.8.10**
  - The default schema for the ThreatQ host will now default to `https` if not defined.
- **Version 1.8.9**
  - Added support for HTTP connections within a trusted K8s environment.
- **Version 1.8.8**
  - Added a Timeout parameter to the Get function.
- **Version 1.8.7**
  - Fixed an issue where `get_saved_search()` failed to find a data collection if it was not on the first page of the API response.
  - Updated Authentication section of the guide to include authenticating via OAuth Credentials.
- **Version 1.8.6**
  - Fixed an issue that would cause the Threat Library total results to display an incorrect value.
- **Version 1.8.5**
  - Resolved a Python 3/2 issue.
- **Version 1.8.4**
  - Fixed a pagination bug.
- **Version 1.8.3**
  - Added cursormark support.
- **Version 1.8.2**
  - Refactored SDK to allow pylint to give scores greater than 8.
- **Version 1.8.1**
  - Added Monkey Patch to fix requests bug noted at https://github.com/psf/requests/issues/3829.
  - Added optional `sort` field to Threat Library code.
- **Version 1.8.0**
  - Added preliminary Python 3 support.
  - Added dependency fixes.
  - Threat Library Search replaces the Advanced Search.