pytx Documentation

Release 0.5.2

Mike Goffin

July 29, 2016

Contents

Contents:

CHAPTER 1

Installation

1.1 Recommended Installation

Use pip:

```
$ pip install pytx
```

You might also want to consider using a virtualenv.

1.2 Dependencies

The pytx library is developed on Python 2.7 and tested against Python 2.7 and 3.x. Besides the Python Standard Library, pytx relies on the following Python libraries which can be installed with `pip` or by manually downloading the package from PyPI:

- requests - Python HTTP for Humans.
- python-dateutil - Extensions to the standard Python datetime module.

CHAPTER 2

Quickstart

ThreatExchange requires an access token in each request. An access token is in the form of:

<app-id>|<app-secret>

The app-id is public knowledge but your app-secret is sensitive. These values are provided to you once you've obtained access to ThreatExchange.

pytx will try to find an access token to use or an access token can be passed to pytx.access_token.access_token(). pytx needs an access token before it will function and can properly make requests properly. Here are some examples of how to provide your access token:

```
from pytx.access_token import access_token

# Use environment variables to build the access token.
# 1. Use the value of the 'TX_ACCESS_TOKEN' environment variable.
# 2. Use the concatenation of the 'TX_APP_ID' and 'TX_APP_SECRET' environment variables.
# There is no need to call access_token() if the environment variables are set.

# 3. Use a .pytx file which contains your app-id and app-secret.
# File should be: 'app-id|app-secret' on one line
# pytx will use either '$PWD/.pytx' or ~/.pytx' if they are found.
# There is no need to call access_token() if the environment variables are set.

# 4. Use the concatenation of the app_id and app_secret parameters
access_token(app_id='<app-id>', app_secret='<app-secret>')

# 5. Use the first line of the file 'token_file'
access_token(token_file='/path/to/token/file')
```

If you need to get the value of the access token pytx is using programmatically, you can do something like the following:

```
from pytx.access_token import get_access_token
print get_access_token()
```

If you would like to log debug information as pytx runs, you can setup the logger by doing the following:

```
from pytx import setup_logger
setup_logger('/path/to/my/log/file.log')
```

Once this is done there is nothing else to do. pytx will automatically log information to that file. If the file cannot be written to expect some issues. If you do not provide an argument to setup_logger, no logging will occur.

If you need to setup a proxy, custom headers, or adjust the verify argument for requests, you can use the connection() function to change them. More info can be found here: http://docs.python-requests.org/en/latest/api/#requests.request

```
from pytx import connection()
connection(headers=<your stuff here>,
           proxies=<your stuff here>,
           verify=<your stuff here>)
```

pytx uses classes as the primary method for developer interaction with the ThreatExchange API. There are several main classes:

- ThreatDescriptor
- ThreatIndicator
- Malware
- MalwareFamily
- ThreatExchangeMember

ThreatExchange allows you to upload new ThreatDescriptors. There are several ways to do so:

```
from pytx import ThreatDescriptor
from pytx.vocabulary import PrivacyType as pt

td = ThreatDescriptor()
td.indicator = "foo"
td.privacy_type = pt.VISIBLE
td.save()
```

```
from pytx import ThreatDescriptor
from pytx.vocabulary import PrivacyType as pt
from pytx.vocabulary import ThreatDescriptor as td

result = ThreatDescriptor.new(params={td.INDICATOR: 'foo',
                                      td.PRIVACY_TYPE: pt.VISIBLE
                                     })
```

```
from pytx import ThreatDescriptor
from pytx.vocabulary import PrivacyType as pt
from pytx.vocabulary import ThreatDescriptor as td

result = ThreatDescriptor.send(params={td.INDICATOR: 'foo',
                                       td.PRIVACY_TYPE: pt.VISIBLE
                                      },
                               type_='POST'
                              )
```

To query for objects in ThreatExchange, you can leverage any of the classes like so:

```
from pytx import ThreatDescriptor
from pytx.vocabulary import ThreatDescriptor as td
from pytx.vocabulary import Types as t

results = ThreatDescriptor.objects(text='www.facebook.com')
for result in results:
    print result.get(td.THREAT_TYPES)

# type is type_ because type is a reserved word.
results = ThreatDescriptor.objects(type_=t.IP_ADDRESS,
                                   text='127.0.0.1')
for result in results:
    print result.get(td.INDICATOR)
```

When you query for objects you get a small summary which does not contain all of the available fields. If you want to get all of the data about a specific object, you can request it in one of two ways:

```
from pytx import ThreatDescriptor

results = ThreatDescriptor.objects(text='www.facebook.com')
for result in results:
    # Make another API call to get all of the fields
    result.details()
    print result.to_dict()
```

```
from pytx import ThreatDescriptor

# Provide the list of fields in the .objects() call to save API calls.
results = ThreatDescriptor.objects(text='www.facebook.com',
                                   fields=ThreatDescriptor._fields
                                  )
for result in results:
    print result.to_dict()
```

When you query for objects, pytx will be default provide you with a generator which returns instantiated objects to you. You can change the behavior in a few ways:

```
from pytx import ThreatDescriptor
from pytx.vocabulary import ThreatDescriptor as ti

# Return the full response instead of a generator.
# Takes precedence over dict_generator.
results = ThreatDescriptor.objects(text='www.facebook.com',
                                   full_response=True)

# Return a dictionary instead of an instantiated object.
results = ThreatDescriptor.objects(text='www.facebook.com',
                                   dict_generator=True)
```

This gives some flexibility to developers as to how they interact with the response.

All of the above class methods come with a 'retries' argument which takes an integer. This tells pytx that if you receive a 500 or a 503 from ThreatExchange, try again up until the number of retries has been reached or until you get a 200 (whichever comes first)..

Behind-the-scenes all of the above examples use the pytx Broker to actually make the requests. If you would prefer to use the Broker directly instead of leveraging the classes you can do so:

```
from pytx.request import Broker
from pytx.vocabulary import ThreatExchange as te

b = Broker()
url = te.URL + te.THREAT_DESCRIPTORS
params = {te.TEXT: "www.facebook.com"}
response = b.get(url, params)
```

The Broker will also allow you to POST and DELETE if you need to.

You can also make Batch requests to the graph via pytx. Batch requests allow you to submit multiple Graph requests in a single POST request. Here's an example:

```
import json
from pytx import ThreatDescriptor, Batch
from pytx.errors import pytxFetchError
```

```
a = ThreatDescriptor.objects(text='foo',
                             request_dict=True)
b = {'type': 'GET',
     'url': '{result=foo:$.data.0.id}'}
try:
    result = Batch.submit(foo=a,
                          bar=b)
    print json.dumps(result, indent=4, sort_keys=True)
except pytxFetchError, e:
    print e.message
```

There are several things to notice in this example. First, the call to find all ThreatDescriptor objects with a text of "foo" has an argument of "request_dict=True". By setting that to True, you are telling the objects call that you'd like the dictionary generated instead of it actually submitting the request to the Graph.

The second thing to notice is that the second request (b) is a manually crafted dictionary. The URL is very cryptic (you can look this up in the Facebook Graph API documentation) but it is saying that for a URL you want the results from the "foo" request and you want the first element's id from the data list.

The submit call for Batch is giving the name "foo" to request "a", and the name "bar" to request "b". The submit call will accept N-number of unnamed arguments and N-number of named arguments. Each one will be considered its own unique request you want to include in the Batch. The only difference between the two is that a named argument will be given a name in the request which can then be used as a reference in other requests in the Batch like the example above.

One thing you might notice is the constant use of vocabulary. pytx comes with a vocabulary which will allow you to write your code using class attributes so if ThreatExchange ever changes a string your code will still function properly.

Error responses can be acquired and leveraged as a dictionary. Here is an example:

```
from pytx import Malware
from pytx.errors import pytxFetchError

m = Malware()
m.id = "19374-19841-4813-408"
response = None
try:
   m.details()
except pytxFetchError, e:
   response = e.message
```

The response variable above will be a dictionary with the following keys:

- code: the TX response code
- fbtrace_id: the TX trace id for the request
- message: the TX server message (what went wrong)
- status_code: the server response status code
- type: the TX error type
- url: the request URL that generated the error

CHAPTER 3

pytx.access_token package

3.1 Module contents

pytx.access_token.**access_token**(*app_id=None*, *app_secret=None*, *token_file=None*)

Use the app_id and app_secret to store the access_token globally for all instantiated objects to leverage.

There are many ways to specify the app_id and app_secret. In order, we will try:

1. Use the value of the 'TX_ACCESS_TOKEN' environment variable.
2. Use the concatenation of the 'TX_APP_ID' and 'TX_APP_SECRET' environment variables.
3. Use the first line of the file '$PWD/.pytx' or ~/.pytx'
4. Use the concatenation of the app_id and app_secret parameters
5. Use the first line of the file 'token_file'

Parameters

- **app_id** (*str*) – The APP-ID to use.
- **app_secret** (*str*) – The APP-SECRET to use.
- **token_file** (*str*) – The full path and filename where to find the access token.

Raises errors.pytxAccessTokenError

pytx.access_token.**get_access_token**()

Returns the existing access token if access_token() has been called. Will attempt to access_token() in the case that there is no access token.

Raises errors.pytxAccessTokenError if there is no access token.

pytx.access_token.**get_app_id**()

Returns the app_id.

CHAPTER 4

pytx.batch package

4.1 Module contents

class pytx.batch.**Batch**(***kwargs*)

Bases: object

Class for making Batch requests to the API.

classmethod get_relative(*url*)

Parse the full URL to get the relative URL.

classmethod prepare_single_request(*request*, *name=None*)

Prepare a single request to be included in batch.

Parameters

- **request** (*dict*) – A dictionary in the format required by Batch.submit().
- **name** (*str*) – A name to give this request.

Returns dict

classmethod submit(**args*, ***kwargs*)

Submit batch request. All non-named args are considered to be dictionaries containing the following:

type: The request type (GET, POST, etc.). url: The full or relative URL for the API call. body: If the type is POST this is the body that will be used.

If you use "method" instead of "type" and/or "relative_urL" instead of "url" (which is accurate to the Graph API) we will use them appropriately.

If you pass a named argument, we will consider the name as the name you wish to include in that specific request. This is useful for referencing a request in another request in the Batch (see FB documentation).

The following named args are considered to be the options below.

Parameters

- **include_headers** (*bool*) – Include headers in response.
- **omit_response** (*bool*) – Omit response on success.
- **retries** (*int*) – Number of retries before stopping.
- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.

Returns dict (using json.loads())

CHAPTER 5

pytx.common package

5.1 Module contents

class pytx.common.**Common**(**kwargs*)
Bases: object

add_connection(*object_id*, *retries=None*, *headers=None*, *proxies=None*, *verify=None*)
Use HTTP POST and add a connection between two objects.

Parameters

- **object_id** (*str*) – The other object-id in the connection.
- **retries** (*int*) – Number of retries to submit before stopping.
- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.

Returns dict (using json.loads())

connections(*cls_or_self*, *id=None*, *connection=None*, *fields=None*, *limit=None*, *full_response=False*, *dict_generator=False*, *request_dict=False*, *retries=None*, *headers=None*, *proxies=None*, *verify=None*, *metadata=False*)
Get object connections. Allows you to limit the fields returned for the objects.

NOTE: This method can be used on both instantiated and uninstantiated classes like so:

foo = ThreatIndicator(id='1234') foo.connections(connection='foo')

foo = ThreatIndicator.connections(id='1234' connection='foo')

Parameters

- **id** (*str*) – The ID of the object to get connections for if the class is not instantiated.
- **fields** (*None, str, list*) – The fields to limit the details to.
- **limit** (*None, int*) – Limit the results.
- **connection** (*None, str*) – The connection to find other related objects with.
- **full_response** (*bool*) – Return the full response instead of the generator. Takes precedence over dict_generator.
- **dict_generator** (*bool*) – Return a dictionary instead of an instantiated object.

- **request_dict** (*bool*) – Return a request dictionary only.
- **retries** (*int*) – Number of retries to fetch a page before stopping.
- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.
- **metadata** (*bool*) – Get extra metadata in the response.

Returns Generator, dict, class, str

delete_connection(*object_id*, *retries=None*, *headers=None*, *proxies=None*, *verify=None*)

Use HTTP DELETE and remove the connection to another object.

Parameters

- **object_id** (*str*) – The other object-id in the connection.
- **retries** (*int*) – Number of retries to submit before stopping.
- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.

Returns dict (using json.loads())

details(*cls_or_self*, *id=None*, *fields=None*, *full_response=False*, *dict_generator=False*, *request_dict=False*, *retries=None*, *headers=None*, *proxies=None*, *verify=None*, *metadata=False*)

Get object details. Allows you to limit the fields returned in the object's details.

NOTE: This method can be used on both instantiated and uninstantiated classes like so:

foo = ThreatIndicator(id='1234') foo.details()

foo = ThreatIndicator.details(id='1234')

BE AWARE: Due to the nature of ThreatExchange allowing you to query for an object by ID but not actually telling you the type of object returned to you, using an ID for an object of a different type (ex: ID for a Malware object using ThreatIndicator class) will result in the wrong object populated with only data that is common between the two objects.

Parameters

- **id** (*str*) – The ID of the object to get details for if the class is not instantiated.
- **fields** (*None, str, list*) – The fields to limit the details to.
- **full_response** (*bool*) – Return the full response instead of the generator. Takes precedence over dict_generator.
- **dict_generator** (*bool*) – Return a dictionary instead of an instantiated object.
- **retries** (*int*) – Number of retries to fetch a page before stopping.
- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.
- **metadata** (*bool*) – Get extra metadata in the response.

Returns Generator, dict, class

expire(*timestamp*, *retries=None*, *headers=None*, *proxies=None*, *verify=None*)
Expire by setting the 'expired_on' timestamp.

Parameters

- **timestamp** (*str*) – The timestamp to set for an expiration date.
- **retries** (*int*) – Number of retries to submit before stopping.
- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.

Returns dict (using json.loads())

false_positive(*object_id*, *retries=None*, *headers=None*, *proxies=None*, *verify=None*)
Mark an object as a false positive by setting the status to UNKNOWN.

Parameters

- **object_id** (*str*) – The object-id of the object to mark.
- **retries** (*int*) – Number of retries to submit before stopping.
- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.

Returns dict (using json.loads())

get(*attr*)
Wrapper around __getattr__ making it easier to use the vocabulary to get class attributes.

Parameters **attr** (*str*) – The name of the attribute to get.

get_changed()
Generate a dict of all of the changed attributes for this class. Useful for generating parameters to submit for saving.

Returns dict

classmethod new(*params*, *request_dict=False*, *retries=None*, *headers=None*, *proxies=None*, *verify=None*)
Submit params to the graph to add an object. We will submit to the object URL used for creating new objects in the graph. When submitting new objects you must provide privacy type and privacy members if the privacy type is something other than visible.

Parameters

- **params** (*dict*) – The parameters to submit.
- **request_dict** (*bool*) – Return a request dictionary only.
- **retries** (*int*) – Number of retries to submit before stopping.
- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.

Returns dict (using json.loads()), str

classmethod objects(*text=None*, *strict_text=False*, *type_=None*, *threat_type=None*, *sample_type=None*, *fields=None*, *limit=None*, *since=None*, *until=None*, *include_expired=False*, *max_confidence=None*, *min_confidence=None*, *owner=None*, *status=None*, *review_status=None*, *share_level=None*, *sort_order=None*, *__raw__=None*, *full_response=False*, *dict_generator=False*, *request_dict=False*, *retries=None*, *headers=None*, *proxies=None*, *verify=None*)

Get objects from ThreatExchange.

Parameters

- **text** (*str*) – The text used for limiting the search.
- **strict_text** (*bool, str, int*) – Whether we should use strict searching.
- **type** (*str*) – The Indicator type to limit to.
- **threat_type** (*str*) – The Threat type to limit to.
- **sample_type** (*str*) – The Sample type to limit to.
- **fields** (*str, list*) – Select specific fields to pull
- **limit** (*int, str*) – The maximum number of objects to return.
- **since** (*str*) – The timestamp to limit the beginning of the search.
- **until** (*str*) – The timestamp to limit the end of the search.
- **include_expired** (*bool*) – Include expired content in your results.
- **max_confidence** (*int*) – The max confidence level to search for.
- **min_confidence** (*int*) – The min confidence level to search for.
- **owner** (*str*) – The owner to limit to. This can be comma-delimited to include multiple owners.
- **status** (*str*) – The status to limit to.
- **review_status** (*str*) – The review status to limit to.
- **share_level** (*str*) – The share level to limit to.
- **sort_order** (*str*) – The sort order for results.
- **__raw__** (*dict*) – Provide a dictionary to force as GET parameters. Overrides all other arguments.
- **full_response** (*bool*) – Return the full response instead of the generator. Takes precedence over dict_generator.
- **dict_generator** (*bool*) – Return a dictionary instead of an instantiated object.
- **request_dict** (*bool*) – Return a request dictionary only.
- **retries** (*int*) – Number of retries to fetch a page before stopping.
- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.

Returns Generator, dict (using json.loads()), str

populate(*attrs*)

Given a dictionary, populate self with the keys as attributes.

Parameters **attrs** (*dict*) – A dictionary used as attributes and values.

react(*reaction*, *retries=None*, *headers=None*, *proxies=None*, *verify=None*)
React to this object.

Parameters

- **reaction** (*str*) – The reaction to provide.
- **retries** (*int*) – Number of retries to submit before stopping.
- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.

Returns dict (using json.loads())

save(*params=None*, *request_dict=False*, *retries=None*, *headers=None*, *proxies=None*, *verify=None*)
Submit changes to the graph to update an object. We will determine the Details URL and submit there (used for updating an existing object). If no parameters are provided, we will try to use get_changed() which may or may not be accurate (you have been warned!).

Parameters

- **params** (*dict*) – The parameters to submit.
- **request_dict** (*bool*) – Return a request dictionary only.
- **retries** (*int*) – Number of retries to submit before stopping.
- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.

Returns dict (using json.loads()), str

send(*cls_or_self*, *id_=None*, *params=None*, *type_=None*, *request_dict=False*, *retries=None*, *headers=None*, *proxies=None*, *verify=None*)
Send custom params to the object URL. If *id* is provided it will be appended to the URL. If this is an uninstantiated class we will use the object type url (ex: /threat_descriptors/). If this is an instantiated object we will use the details URL. The **type_** should be either GET or POST. We will default to GET if this is an uninstantiated class, and POST if this is an instantiated class.

Parameters

- **id** (*str*) – ID of a graph object.
- **params** (*dict*) – Parameters to submit in the request.
- **type** (*str*) – GET or POST
- **request_dict** (*bool*) – Return a request dictionary only.
- **retries** (*int*) – Number of retries to submit before stopping.
- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.

Returns dict (using json.loads()), str

set(*name*, *value*)
Wrapper around __setattr__ making it easier to use the vocabulary to set class attributes.

Parameters

- **name** (*str*) – The name of the attribute to set.
- **value** (*None, int, str, bool*) – The value to set the attribute to.

to_dict()
Convert this object into a dictionary.

Returns dict

class pytx.common.**class_or_instance_method**(*func*)
Bases: object

Custom decorator. This binds to the class if no instance is available, otherwise it will bind to the instance.

This allows us to use a single method which can take both "self" and "cls" as the first argument.

CHAPTER 6

pytx.connection package

6.1 Module contents

pytx.connection.**connection**(*headers=None*, *proxies=None*, *verify=None*)
Configure headers, proxies, and verify settings for requests. This is a global setting that all requests calls will use unless overridden on a per-call basis.

Parameters

- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.

pytx.connection.**get_headers**()
Returns the existing headers setting.

pytx.connection.**get_proxies**()
Returns the existing proxies setting.

pytx.connection.**get_verify**()
Returns the existing verify setting.

CHAPTER 7

pytx.errors package

7.1 Module contents

exception pytx.errors.**pytxAccessTokenError**(*message*)
Bases: *pytx.errors.pytxException*

Exception for when we the developer don't set a token before instantiating an object.

exception pytx.errors.**pytxAttributeError**(*message*)
Bases: *pytx.errors.pytxException*

Exception for when we are given a value we are not expecting or is invalid.

exception pytx.errors.**pytxException**(*message*)
Bases: exceptions.Exception

Generic Exception.

exception pytx.errors.**pytxFetchError**(*message*)
Bases: *pytx.errors.pytxException*

Exception for when a GET or POST attempt fails.

exception pytx.errors.**pytxValueError**(*message*)
Bases: *pytx.errors.pytxException*

Exception for when we are given a value we are not expecting or is invalid.

CHAPTER 8

pytx.logger package

8.1 Module contents

pytx.logger.**do_log**()
Should we log?

pytx.logger.**log_message**(*message*)
Logs a message to the logger.

pytx.logger.**setup_logger**(*log_file=None*)
Set a log file to log messages to. Useful for debugging.

Parameters **log_file** (*str*) – A file to log to.

CHAPTER 9

pytx.malware package

9.1 Module contents

class pytx.malware.**Malware**(**kwargs*)

Bases: *pytx.common.Common*

rf

Return the base64-decoded and unzipped sample.

rfh

Return a file handle of the base64-decoded and unzipped sample.

zf

Return the base64-decoded sample in a zip file.

zfh

Return a file handle of the base64-decoded sample in a zip file.

CHAPTER 10

pytx.malware_family package

10.1 Module contents

class pytx.malware_family.**MalwareFamily**(**kwargs*)
Bases: *pytx.common.Common*

CHAPTER 11

pytx.request package

11.1 Module contents

class pytx.request.**Broker**

Bases: object

The Broker handles validation and submission of requests as well as consumption and returning of the result. It is leveraged by the other classes.

Since the Broker takes care of the entire request/response cycle, it can be used on its own to interact with the ThreatExchange API without the need for the other classes if a developer wishes to use it.

classmethod build_get_parameters(*text=None*, *strict_text=None*, *type_=None*, *threat_type=None*, *sample_type=None*, *fields=None*, *limit=None*, *since=None*, *until=None*, *include_expired=None*, *max_confidence=None*, *min_confidence=None*, *owner=None*, *status=None*, *review_status=None*, *share_level=None*, *sort_order=None*)

Validate arguments and convert them into GET parameters.

Parameters

- **text** (*str*) – The text used for limiting the search.
- **strict_text** (*bool, str, int*) – Whether we should use strict searching.
- **type** (*str*) – The Indicator type to limit to.
- **threat_type** (*str*) – The Threat type to limit to.
- **sample_type** (*str*) – The Sample type to limit to.
- **fields** (*str, list*) – Select specific fields to pull
- **limit** (*int, str*) – The maximum number of objects to return.
- **since** (*str*) – The timestamp to limit the beginning of the search.
- **until** (*bool, str, int*) – The timestamp to limit the end of the search.
- **include_expired** – Include expired content in your results.
- **max_confidence** (*int*) – The max confidence level to search for.
- **min_confidence** (*int*) – The min confidence level to search for.
- **owner** (*str*) – The owner to limit to. This can be comma-delimited to include multiple owners.

- **status** (*str*) – The status to limit to.
- **review_status** (*str*) – The review status to limit to.
- **share_level** (*str*) – The share level to limit to.
- **sort_order** (*str*) – The sort order for results.

Returns dict

classmethod build_session(*retries=None*)
Build custom requests session with retry capabilities.

Parameters **retries** (*int*) – Number of retries before stopping.

Returns requests session object

classmethod delete(*url*, *params=None*, *retries=None*, *headers=None*, *proxies=None*, *verify=None*)
Send a DELETE request.

Parameters

- **url** (*str*) – The URL to send the DELETE request to.
- **params** (*dict*) – The DELETE parameters to send in the request.
- **retries** (*int*) – Number of retries before stopping.
- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.

Returns dict (using json.loads())

classmethod get(*url*, *params=None*, *retries=None*, *headers=None*, *proxies=None*, *verify=None*)
Send a GET request.

Parameters

- **url** (*str*) – The URL to send the GET request to.
- **params** (*dict*) – The GET parameters to send in the request.
- **retries** (*int*) – Number of retries before stopping.
- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.

Returns dict (using json.loads())

classmethod get_generator(*klass*, *url*, *to_dict=False*, *params=None*, *retries=None*, *headers=None*, *proxies=None*, *verify=None*)
Generator for managing GET requests. For each GET request it will yield the next object in the results until there are no more objects. If the GET response contains a 'next' value in the 'paging' section, the generator will automatically fetch the next set of results and continue the process until the total limit has been reached or there is no longer a 'next' value.

Parameters

- **klass** (*class*) – The class to use for the generator.
- **url** (*str*) – The URL to send the GET request to.
- **to_dict** (*bool*) – Return a dictionary instead of an instantiated class.

- **params** (*dict*) – The GET parameters to send in the request.
- **retries** (*int*) – Number of retries before stopping.
- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.

Returns Generator

static get_new(*klass*, *attrs*)
Return a new instance of klass.

Parameters

- **klass** – The class to create a new instance of.
- **attrs** (*dict*) – The attributes to set for this new instance.

Returns new instance of klass

static handle_results(*resp*)
Handle the results of a request.

Parameters **resp** (*response object*) – The HTTP response.

Returns dict (using json.loads())

static is_timestamp(*timestamp*)
Verifies the timestamp provided is a valid timestamp.

Valid timestamps are based on PHP's "strtotime" function. As of right now even with python's "dateutil" library there are some strtotime valid strings that do not validate properly. Until such a time as this can become accurate and robust enough to have feature parity with strtotime, this will always return True and leave proper timestamps to the API user.

Parameters **timestamp** (*str*) – Value to verify is a timestamp.

Returns True

classmethod post(*url*, *params=None*, *retries=None*, *headers=None*, *proxies=None*, *verify=None*)
Send a POST request.

Parameters

- **url** (*str*) – The URL to send the POST request to.
- **params** (*dict*) – The POST parameters to send in the request.
- **retries** (*int*) – Number of retries before stopping.
- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.

Returns dict (using json.loads())

classmethod request_dict(*type_*, *url*, *params=None*, *body=None*)
Return a dictionary with the request type, URL, and optionally a body.

Parameters

- **type** (*str*) – The request type.
- **url** (*str*) – The request URL.

- **params** (*dict*) – The parameters to submit.
- **body** (*str*) – The body to submit.

Returns dict

static sanitize_bool(*value*)
If value is provided, sanitize it.

'true' will be used if value is in [True, 'true', 'True', 1]. 'false' will be used if value is in [False, 'false', 'False', 0].

If we receive any other value value will be set to None and ignored when building the GET request.

Parameters **value** (*bool, str, int*) – The value to sanitize.

Returns str, None

classmethod validate_get(*limit*, *since*, *until*)
Executes validation for the GET parameters: limit, since, until.

Parameters

- **limit** (*int, str*) – The limit to validate.
- **since** (*str*) – The since timestamp to validate.
- **until** (*str*) – The until timestamp to validate.

static validate_limit(*limit*)
Verifies the limit provided is valid and within the max limit Facebook will allow you to use.

Parameters **limit** (*int, str*) – Value to verify is a valid limit.

Returns `pytxValueError` if invalid.

CHAPTER 12

pytx.threat_descriptor package

12.1 Module contents

class pytx.threat_descriptor.**ThreatDescriptor**(***kwargs*)

Bases: *pytx.common.Common*

CHAPTER 13

pytx.threat_exchange_member package

13.1 Module contents

class pytx.threat_exchange_member.**ThreatExchangeMember**(***kwargs*)

Bases: object

get(*attr*)

Wrapper around __getattr__ making it easier to use the vocabulary to get class attributes.

Parameters **attr** (*str*) – The name of the attribute to get.

classmethod **objects**(*full_response=False*, *dict_generator=False*, *retries=None*, *headers=None*, *proxies=None*, *verify=None*)

Get a list of Threat Exchange Members

Parameters

- **full_response** (*bool*) – Return the full response instead of the generator. Takes precedence over dict_generator.
- **dict_generator** (*bool*) – Return a dictionary instead of an instantiated object.
- **retries** (*int*) – Number of retries to fetch a page before stopping.
- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.

Returns Generator, dict (using json.loads())

to_dict()

Convert this object into a dictionary.

Returns dict

CHAPTER 14

pytx.threat_indicator package

14.1 Module contents

class pytx.threat_indicator.**ThreatIndicator**(***kwargs*)

Bases: *pytx.common.Common*

pytx.threat_privacy_group package

15.1 Module contents

class pytx.threat_privacy_group.**ThreatPrivacyGroup**(***kwargs*)
Bases: *pytx.common.Common*

get_members(*retries=None*, *headers=None*, *proxies=None*, *verify=None*)
Get the members of a Threat Privacy Group

Parameters

- **retries** (*int*) – Number of retries to fetch a page before stopping.
- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.

Returns list

classmethod mine(*role=None*, *full_response=False*, *dict_generator=False*, *retries=None*, *headers=None*, *proxies=None*, *verify=None*)
Find all of the Threat Privacy Groups that I am either the owner or a member.

Parameters

- **role** (*str*) – Whether you are an 'owner' or a 'member'
- **full_response** (*bool*) – Return the full response instead of the generator. Takes precedence over dict_generator.
- **dict_generator** (*bool*) – Return a dictionary instead of an instantiated object.
- **retries** (*int*) – Number of retries to fetch a page before stopping.
- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.

Returns Generator, dict (using json.loads())

set_members(*members=None*, *retries=None*, *headers=None*, *proxies=None*, *verify=None*)
Set the members of a Threat Privacy Group

Parameters

- **members** (*str or list*) – str or list of member IDs to add as members.

- **retries** (*int*) – Number of retries to fetch a page before stopping.
- **headers** (*dict*) – header info for requests.
- **proxies** (*dict*) – proxy info for requests.
- **verify** (*bool, str*) – verify info for requests.

Returns list

CHAPTER 16

pytx.threat_tag package

16.1 Module contents

class pytx.threat_tag.**ThreatTag**(***kwargs*)

Bases: *pytx.common.Common*

CHAPTER 17

pytx.utils package

17.1 Module contents

pytx.utils.**convert_to_header**(*field*)

Converts a ThreatExchange field name to a CSV-writeable string. Also handles nested fields. For example, [TD.OWNER, TE.NAME] becomes "owner_email". :param field: name of ThreatExchange field :type field: list, str :returns: str

pytx.utils.**get_data_field**(*field*, *result*)

Given a field name in ThreatExchange, grabs the resulting field value from the ThreatExchange object. :param field: name of ThreatExchange field :type field: list, str :param result: The resulting object received from ThreatExchange :type result: ThreatExchange object :returns: str, int

pytx.utils.**get_time_params**(*end_date*, *day_counter*, *format_*)

Generates both unix and **format_** specified timestamps for a given 24 hour window ending at day_counter days back from end_date :param end_date: User-specified end_date of data to be gathered :type end_date: str :param day_counter: number of days back to slide the time window :type day_counter: int :param **format_**: The specified datetime format :type **format_**: str :returns: list

CHAPTER 18

pytx.vocabulary package

18.1 Module contents

class pytx.vocabulary.**Attack**

Bases: object

Vocabulary for the Threat Indicator Attack type.

ACCESS_TOKEN_THEFT = 'ACCESS_TOKEN_THEFT'

BOGON = 'BOGON'

BOT = 'BOT'

BRUTE_FORCE = 'BRUTE_FORCE'

CLICKJACKING = 'CLICKJACKING'

COMPROMISED = 'COMPROMISED'

CREEPER = 'CREEPER'

DRUGS = 'DRUGS'

EMAIL_SPAM = 'EMAIL_SPAM'

EXPLICIT_CONTENT = 'EXPLICIT_CONTENT'

EXPLOIT_KIT = 'EXPLOIT_KIT'

FAKE_ACCOUNTS = 'FAKE_ACCOUNT'

FINANCIALS = 'FINANCIAL'

IP_INFRINGEMENT = 'IP_INFRINGEMENT'

MALICIOUS_APP = 'MALICIOUS_APP'

MALICIOUS_NAMESERVER = 'MALICIOUS_NAMESERVER'

MALICIOUS_WEBSERVER = 'MALICIOUS_WEBSERVER'

MALVERTISING = 'MALVERTISING'

MALWARE = 'MALWARE'

PASSIVE_DNS = 'PASSIVE_DNS'

PHISHING = 'PHISHING'

PIRACY = 'PIRACY'

PROXY = 'PROXY'

SCAM = 'SCAM'

SCANNING = 'SCANNING'

SCRAPING = 'SCRAPING'

SELF_XSS = 'SELF_XSS'

SHARE_BAITING = 'SHARE_BAITING'

TARGETED = 'TARGETED'

TERRORISM = 'TERRORISM'

UNKNOWN = 'UNKNOWN'

WEAPONS = 'WEAPONS'

WEB_APP = 'WEB_APP'

class pytx.vocabulary.**Batch**
Bases: object

Vocabulary used for batch operations.

BODY = 'body'

INCLUDE_HEADERS = 'include_headers'

METHOD = 'method'

RELATIVE_URL = 'relative_url'

class pytx.vocabulary.**Common**
Bases: object

Vocabulary common to multiple objects.

ADDED_ON = 'added_on'

ID = 'id'

METADATA = 'metadata'

MY_REACTIONS = 'my_reactions'

SHARE_LEVEL = 'share_level'

STATUS = 'status'

VICTIM_COUNT = 'victim_count'

class pytx.vocabulary.**Connection**
Bases: object

Vocabulary specific to searching for, creating, or removing connections between objects.

ADDED_ON = 'added_on'

CRX = 'crx'

DESCRIPTORS = 'descriptors'

DROPPED = 'dropped'

DROPPED_BY = 'dropped_by'

FAMILIES = 'families'

ID = 'id'

MALWARE_ANALYSES = 'malware_analyses'

RELATED = 'related'

STATUS = 'status'

THREAT_INDICATORS = 'threat_indicators'

VARIANTS = 'variants'

VICTIM_COUNT = 'victim_count'

class pytx.vocabulary.**Malware**
Bases: object

Vocabulary specific to searching for, creating, or modifying a Malware object.

ADDED_ON = 'added_on'

CRX = 'crx'

ID = 'id'

IMPHASH = 'imphash'

MD5 = 'md5'

METADATA = 'metadata'

PASSWORD = 'password'

PE_RICH_HEADER = 'pe_rich_header'

PRIVACY_TYPE = 'privacy_type'

SAMPLE = 'sample'

SAMPLE_SIZE = 'sample_size'

SAMPLE_SIZE_COMPRESSED = 'sample_size_compressed'

SAMPLE_TYPE = 'sample_type'

SHA1 = 'sha1'

SHA256 = 'sha256'

SHARE_LEVEL = 'share_level'

SSDEEP = 'ssdeep'

STATUS = 'status'

TAGS = 'tags'

VICTIM_COUNT = 'victim_count'

XPI = 'xpi'

class pytx.vocabulary.**MalwareAnalysisTypes**
Bases: object

Vocabulary specific to Malware Analysis Sample Types

ANDROID_APK = 'ANDROID_APK'

CHROME_EXT = 'CHROME_EXT'

DALVIK_DEX = 'DALVIK_DEX'

ELF_X64 = 'ELF_X64'

ELF_X86 = 'ELF_X86'

FIREFOX_EXT = 'FIREFOX_EXT'

FLASH_DATA = 'FLASH_DATA'

FLASH_VIDEO = 'FLASH_VIDEO'

GENERIC_BINARY = 'GENERIC_BINARY'

GENERIC_IMAGE = 'GENERIC_IMAGE'

GENERIC_TEXT = 'GENERIC_TEXT'

HTML = 'HTML'

IMAGE_BMP = 'IMAGE_BMP'

IMAGE_GIF = 'IMAGE_GIF'

IMAGE_JPEG = 'IMAGE_JPEG'

IMAGE_PNG = 'IMAGE_PNG'

IMAGE_TIFF = 'IMAGE_TIFF'

IOS_APP = 'IOS_APP'

JAR_ARCHIVE = 'JAR_ARCHIVE'

JAVASCRIPT = 'JAVASCRIPT'

MACH_O = 'MACH_O'

OFFICE_DOCX = 'OFFICE_DOCX'

OFFICE_PPTX = 'OFFICE_PPTX'

OFFICE_XLSX = 'OFFICE_XLSX'

PDF_DOCUMENT = 'PDF_DOCUMENT'

PE_X64 = 'PE_X64'

PE_X86 = 'PE_X86'

RAR_ARCHIVE = 'RAR_ARCHIVE'

RTF_FILE = 'RTF_FILE'

UNKNOWN = 'UNKNOWN'

ZIP_ARCHIVE = 'ZIP_ARCHIVE'

class pytx.vocabulary.**MalwareFamilies**

Bases: object

Vocabulary specific to searching for Malware Family objects.

ADDED_ON = 'added_on'

ALIASES = 'aliases'

DESCRIPTION = 'description'

FAMILY_TYPE = 'family_type'

ID = 'id'

MALICIOUS = 'malicious'

NAME = 'name'

PRIVACY_TYPE = 'privacy_type'

SAMPLE_COUNT = 'sample_count'

SHARE_LEVEL = 'share_level'

class pytx.vocabulary.**MalwareFamily**
Bases: object

Vocabulary for the Malware Family Type.

AVSCAN = 'AVSCAN'

AV_SIGNATURE = 'AV_SIGNATURE'

BARF10 = 'BARF10'

FSH_HTML = 'FSH_HTML'

FSH_SSDEEP = 'FSH_SSDEEP'

IMP_HASH = 'IMP_HASH'

JS004 = 'JS004'

JS005 = 'JS005'

MANUAL = 'MANUAL'

PE_CERT_SHA256 = 'PE_CERT_SHA256'

PE_EXPORT = 'PE_EXPORT'

PE_RSRC_SHA256 = 'PE_RSRC_SHA256'

PE_SECTION_SHA256 = 'PE_SECTION_SHA256'

PE_TIMESTAMP = 'PE_TIMESTAMP'

PE_VERSION_VALUE = 'PE_VERSION_VALUE'

RICH_HEADER_HASH = 'RICH_HEADER_HASH'

SSDEEP_HASH = 'SSDEEP_HASH'

UNKNOWN = 'UNKNOWN'

YARA = 'YARA'

class pytx.vocabulary.**Paging**
Bases: object

Vocabulary for the fields available in a GET response specific to paging.

CURSORS = 'cursors'

NEXT = 'next'

PAGING = 'paging'

class pytx.vocabulary.**PagingCursor**
Bases: object

Vocabulary for describing the paging cursor in a GET response.

AFTER = 'after'

BEFORE = 'before'

class pytx.vocabulary.**Precision**
Bases: object

Vocabulary for the Precision Type.

HIGH = 'HIGH'

LOW = 'LOW'

MEDIUM = 'MEDIUM'

UNKNOWN = 'UNKNOWN'

class pytx.vocabulary.**PrivacyType**
Bases: object

Vocabulary for the Threat Indicator Privacy Type.

HAS_PRIVACY_GROUP = 'HAS_PRIVACY_GROUP'

HAS_WHITELIST = 'HAS_WHITELIST'

NONE = 'NONE'

VISIBLE = 'VISIBLE'

class pytx.vocabulary.**Reaction**
Bases: object

Vocabulary for describing a reaction.

HELPFUL = 'HELPFUL'

NOT_HELPFUL = 'NOT_HELPFUL'

OUTDATED = 'OUTDATED'

SAW_THIS_TOO = 'SAW_THIS_TOO'

WANT_MORE_INFO = 'WANT_MORE_INFO'

class pytx.vocabulary.**Response**
Bases: object

Vocabulary for describing server responses.

CODE = 'code'

ERROR = 'error'

FBTRACE_ID = 'fbtrace_id'

ID = 'id'

MESSAGE = 'message'

SUCCESS = 'success'

TYPE = 'type'

class pytx.vocabulary.**ReviewStatus**
Bases: object

Vocabulary for the Review Status Type.

PENDING = 'PENDING'

REVIEWED_AUTOMATICALLY = 'REVIEWED_AUTOMATICALLY'

REVIEWED_MANUALLY = 'REVIEWED_MANUALLY'

UNKNOWN = 'UNKNOWN'

UNREVIEWED = 'UNREVIEWED'

class pytx.vocabulary.**Role**
Bases: object

Vocabulary for the Threat Indicator Role type.

BENEFACTOR = 'BENEFACTOR'

C2 = 'C2'

EXPLOIT = 'EXPLOIT'

PHISHING_SITE = 'PHISHING_SITE'

RECON = 'RECON'

TRACKING_PIXEL = 'TRACKING_PIXEL'

UNKNOWN = 'UNKNOWN'

WATERING_HOLE = 'WATERING_HOLE'

class pytx.vocabulary.**Severity**
Bases: object

Vocabulary for the available severity levels for a Threat Indicator. Intentionally out of alphabetical order to reflect order of severity.

APOCALYPSE = 'APOCALYPSE'

INFO = 'INFO'

SEVERE = 'SEVERE'

SUSPICIOUS = 'SUSPICIOUS'

UNKNOWN = 'UNKNOWN'

WARNING = 'WARNING'

class pytx.vocabulary.**ShareLevel**
Bases: object

Vocabulary for the share level of an object. This is based off of TLP.

AMBER = 'AMBER'

GREEN = 'GREEN'

RED = 'RED'

UNKNOWN = 'UNKNOWN'

WHITE = 'WHITE'

class pytx.vocabulary.**SignatureType**
Bases: object

Vocabulary for the Threat Indicator Signature Threat Type.

BRO = 'BRO'

REGEX_URL = 'REGEX_URL'

SNORT = 'SNORT'

SURICATA = 'SURICATA'

UNKNOWN = 'UNKNOWN'

YARA = 'YARA'

class pytx.vocabulary.**Status**
Bases: object

Vocabulary for the status of an object.

MALICIOUS = 'MALICIOUS'

NON_MALICIOUS = 'NON_MALICIOUS'

SUSPICIOUS = 'SUSPICIOUS'

UNKNOWN = 'UNKNOWN'

class pytx.vocabulary.**ThreatDescriptor**
Bases: object

Vocabulary specific to searching for, adding, or modifying a Threat Indicator object.

ADDED_ON = 'added_on'

ATTACK_TYPE = 'attack_type'

CONFIDENCE = 'confidence'

DESCRIPTION = 'description'

EXPIRED_ON = 'expired_on'

ID = 'id'

INDICATOR = 'indicator'

LAST_UPDATED = 'last_updated'

METADATA = 'metadata'

MY_REACTIONS = 'my_reactions'

OWNER = 'owner'

PRECISION = 'precision'

PRIVACY_MEMBERS = 'privacy_members'

PRIVACY_TYPE = 'privacy_type'

RAW_INDICATOR = 'raw_indicator'

REVIEW_STATUS = 'review_status'

SEVERITY = 'severity'

SHARE_LEVEL = 'share_level'

SOURCE_URI = 'source_uri'

STATUS = 'status'

TAGS = 'tags'

THREAT_TYPE = 'threat_type'

TYPE = 'type'

class pytx.vocabulary.**ThreatExchange**

Bases: object

General vocabulary for ThreatExchange.

ACCESS_TOKEN = 'access_token'

ASCENDING = 'ASCENDING'

BATCH = 'batch'

DATA = 'data'

DEC_TOTAL = 1

DEFAULT_LIMIT = 25

DESCENDING = 'DESCENDING'

FIELDS = 'fields'

INCLUDE_EXPIRED = 'include_expired'

INCLUDE_HEADERS = 'include_headers'

LIMIT = 'limit'

MALWARE_ANALYSES = 'malware_analyses/'

MALWARE_FAMILIES = 'malware_families/'

MAX_CONFIDENCE = 'max_confidence'

METADATA = 'metadata'

MIN_CONFIDENCE = 'min_confidence'

MIN_TOTAL = 0

NEXT = 'next'

NO_TOTAL = -1

OMIT_RESPONSE_ON_SUCCESS = 'omit_response_on_success'

OWNER = 'owner'

PAGING = 'paging'

REACTIONS = 'reactions'

RELATED = 'related'

RELATED_ID = 'related_id'

REVIEW_STATUS = 'review_status'

SAMPLE_TYPE = 'sample_type'

SHARE_LEVEL = 'share_level'

SINCE = 'since'

SORT_ORDER = 'sort_order'

STATUS = 'status'

STRICT_TEXT = 'strict_text'

TEXT = 'text'

THREAT_DESCRIPTORS = 'threat_descriptors/'

THREAT_EXCHANGE_MEMBERS = 'threat_exchange_members/'

THREAT_INDICATORS = 'threat_indicators/'

THREAT_PRIVACY_GROUPS = 'threat_privacy_groups/'

THREAT_PRIVACY_GROUPS_MEMBER = 'threat_privacy_groups_member/'

THREAT_PRIVACY_GROUPS_OWNER = 'threat_privacy_groups_owner/'

THREAT_TAGS = 'threat_tags/'

THREAT_TYPE = 'threat_type'

TX_ACCESS_TOKEN = 'TX_ACCESS_TOKEN'

TX_APP_ID = 'TX_APP_ID'

TX_APP_SECRET = 'TX_APP_SECRET'

TYPE = 'type'

UNTIL = 'until'

URL = 'https://graph.facebook.com/'

VERSION = 'v2.6/'

class pytx.vocabulary.**ThreatExchangeMember**

Bases: object

Vocabulary for describing a ThreatExchangeMember.

EMAIL = 'email'

ID = 'id'

NAME = 'name'

class pytx.vocabulary.**ThreatIndicator**

Bases: object

Vocabulary specific to searching for, adding, or modifying a Threat Indicator object.

ID = 'id'

INDICATOR = 'indicator'

METADATA = 'metadata'

TYPE = 'type'

class pytx.vocabulary.**ThreatPrivacyGroup**

Bases: object

Vocabulary for describing a ThreatPrivacyGroup.

DESCRIPTION = 'description'

ID = 'id'

MEMBERS = 'members'

MEMBERS_CAN_SEE = 'members_can_see'

MEMBERS_CAN_USE = 'members_can_use'

NAME = 'name'

class pytx.vocabulary.**ThreatTag**
Bases: object

Vocabulary specific to searching for, adding, or modifying a Threat Tag object.

ID = 'id'

OBJECTS = 'objects'

TAGGED_OBJECTS = 'tagged_objects'

TEXT = 'text'

class pytx.vocabulary.**ThreatType**
Bases: object

Vocabulary for the available Threat Types for a Threat Indicator.

BAD_ACTOR = 'BAD_ACTOR'

COMMAND_EXEC = 'COMMAND_EXEC'

COMPROMISED_CREDENTIAL = 'COMPROMISED_CREDENTIAL'

HT_VICTIM = 'HT_VICTIM'

MALICIOUS_AD = 'MALICIOUS_AD'

MALICIOUS_API_KEY = 'MALICIOUS_API_KEY'

MALICIOUS_CONTENT = 'MALICIOUS_CONTENT'

MALICIOUS_DOMAIN = 'MALICIOUS_DOMAIN'

MALICIOUS_INJECT = 'MALICIOUS_INJECT'

MALICIOUS_IP = 'MALICIOUS_IP'

MALICIOUS_SSL_CERT = 'MALICIOUS_SSL_CERT'

MALICIOUS_SUBNET = 'MALICIOUS_SUBNET'

MALICIOUS_URL = 'MALICIOUS_URL'

MALICIOUS_URL_CHUNK = 'MALICIOUS_URL_CHUNK'

MALWARE_ARTIFACTS = 'MALWARE_ARTIFACTS'

MALWARE_SAMPLE = 'MALWARE_SAMPLE'

MALWARE_SIGNATURE = 'MALWARE_SIGNATURE'

MALWARE_VICTIM = 'MALWARE_VICTIM'

PROXY_IP = 'PROXY_IP'

SIGNATURE = 'SIGNATURE'

SINKHOLE_EVENT = 'SINKHOLE_EVENT'

SMS_SPAM = 'SMS_SPAM'

UNKNOWN = 'UNKNOWN'

VICTIM_IP_USAGE = 'VICTIM_IP_USAGE'

WEB_REQUEST = 'WEB_REQUEST'

WHITELIST_DOMAIN = 'WHITELIST_DOMAIN'

WHITELIST_IP = 'WHITELIST_IP'

WHITELIST_URL = 'WHITELIST_URL'

class pytx.vocabulary.**Types**
Bases: object

Vocabulary for the Threat Indicator Types.

ADJUST_TOKEN = 'ADJUST_TOKEN'

API_KEY = 'API_KEY'

AS_NUMBER = 'AS_NUMBER'

BANNER = 'BANNER'

CMD_LINE = 'CMD_LINE'

COOKIE_NAME = 'COOKIE_NAME'

CRX = 'CRX'

DEBUG_STRING = 'DEBUG_STRING'

DEST_PORT = 'DEST_PORT'

DIRECTORY_QUERIED = 'DIRECTORY_QUERIED'

DOMAIN = 'DOMAIN'

EMAIL_ADDRESS = 'EMAIL_ADDRESS'

FILE_CREATED = 'FILE_CREATED'

FILE_DELETED = 'FILE_DELETED'

FILE_MOVED = 'FILE_MOVED'

FILE_NAME = 'FILE_NAME'

FILE_OPENED = 'FILE_OPENED'

FILE_READ = 'FILE_READ'

FILE_WRITTEN = 'FILE_WRITTEN'

GET_PARAM = 'GET_PARAM'

HASH_IMPHASH = 'HASH_IMPHASH'

HASH_MD5 = 'HASH_MD5'

HASH_SHA1 = 'HASH_SHA1'

HASH_SHA256 = 'HASH_SHA256'

HASH_SSDEEP = 'HASH_SSDEEP'

HTML_ID = 'HTML_ID'

HTTP_REQUEST = 'HTTP_REQUEST'

IP_ADDRESS = 'IP_ADDRESS'

IP_SUBNET = 'IP_SUBNET'

ISP = 'ISP'

LATITUDE = 'LATITUDE'

LAUNCH_AGENT = 'LAUNCH_AGENT'

LOCATION = 'LOCATION'

LONGITUDE = 'LONGITUDE'

MALWARE_NAME = 'MALWARE_NAME'

MEMORY_ALLOC = 'MEMORY_ALLOC'

MEMORY_PROTECT = 'MEMORY_PROTECT'

MEMORY_WRITTEN = 'MEMORY_WRITTEN'

MUTANT_CREATED = 'MUTANT_CREATED'

MUTEX = 'MUTEX'

NAME_SERVER = 'NAME_SERVER'

OTHER_FILE_OP = 'OTHER_FILE_OP'

PASSWORD = 'PASSWORD'

PASSWORD_SALT = 'PASSWORD_SALT'

PAYLOAD_DATA = 'PAYLOAD_DATA'

PAYLOAD_TYPE = 'PAYLOAD_TYPE'

POST_DATA = 'POST_DATA'

PROTOCOL = 'PROTOCOL'

REFERER = 'REFERER'

REGISTRAR = 'REGISTRAR'

REGISTRY_KEY = 'REGISTRY_KEY'

REG_KEY_CREATED = 'REG_KEY_CREATED'

REG_KEY_DELETED = 'REG_KEY_DELETED'

REG_KEY_ENUMERATED = 'REG_KEY_ENUMERATED'

REG_KEY_MONITORED = 'REG_KEY_MONITORED'

REG_KEY_OPENED = 'REG_KEY_OPENED'

REG_KEY_VALUE_CREATED = 'REG_KEY_VALUE_CREATED'

REG_KEY_VALUE_DELETED = 'REG_KEY_VALUE_DELETED'

REG_KEY_VALUE_MODIFIED = 'REG_KEY_VALUE_MODIFIED'

REG_KEY_VALUE_QUERIED = 'REG_KEY_VALUE_QUERIED'

SIGNATURE = 'SIGNATURE'

SOURCE_PORT = 'SOURCE_PORT'

TELEPHONE = 'TELEPHONE'

URI = 'URI'

USER_AGENT = 'USER_AGENT'

VOLUME_QUERIED = 'VOLUME_QUERIED'

WEBSTORAGE_KEY = 'WEBSTORAGE_KEY'

WEB_PAYLOAD = 'WEB_PAYLOAD'

WHOIS_ADDR1 = 'WHOIS_ADDR1'

WHOIS_ADDR2 = 'WHOIS_ADDR2'

WHOIS_NAME = 'WHOIS_NAME'

XPI = 'XPI'

CHAPTER 19

Indices and tables

- genindex
- modindex
- search

p

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z