PyMISP Documentation

Release master

Raphaël Vinot

Aug 17, 2018

Contents

Contents:

CHAPTER 1

README

CHAPTER 2

PyMISP - Python Library to access MISP

PyMISP is a Python library to access MISP platforms via their REST API.

PyMISP allows you to fetch events, add or update events/attributes, add or update samples or search for attributes.

2.1 Requirements

- requests

2.2 Install from pip

```
pip3 install pymisp
```

2.3 Install the latest version from repo

```
git clone https://github.com/MISP/PyMISP.git && cd PyMISP
git submodule update --init
pip3 install -I .
```

2.4 Samples and how to use PyMISP

Various examples and samples scripts are in the examples/ directory.

In the examples directory, you will need to change the keys.py.sample to enter your MISP url and API key.

```
cd examples
cp keys.py.sample keys.py
vim keys.py
```

The API key of MISP is available in the Automation section of the MISP web interface.

To test if your URL and API keys are correct, you can test with examples/last.py to fetch the last 10 events published.

```
cd examples
python3 last.py -l 10
```

2.5 Debugging

You have two options there:

1. Pass `debug=True` to `PyMISP` and it will enable logging.DEBUG to stderr on the whole module
2. Use the python logging module directly:

```
import logging
logger = logging.getLogger('pymisp')

# Configure it as you whish, for example, enable DEBUG mode:
logger.setLevel(logging.DEBUG)
```

Or if you want to write the debug output to a file instead of stderr:

```
import pymisp
import logging

logger = logging.getLogger('pymisp')
logging.basicConfig(level=logging.DEBUG, filename="debug.log", filemode='w',␣
↪format=pymisp.FORMAT)
```

2.6 Documentation

PyMISP API documentation is available.

Documentation can be generated with epydoc:

```
epydoc --url https://github.com/MISP/PyMISP --graph all --name PyMISP --pdf pymisp -o␣
↪doc
```

2.7 Everything is a Mutable Mapping

. . . or at least everything that can be imported/exported from/to a json blob

`AbstractMISP` is the master class, and inherit `collections.MutableMapping` which means the class can be represented as a python dictionary.

The abstraction assumes every property that should not be seen in the dictionary is prepended with a _, or its name is added to the private list __not_jsonable (accessible through update_not_jsonable and set_not_jsonable.

This master class has helpers that will make it easy to load, and export, to, and from, a json string.

MISPEvent, MISPAttribute, MISPObjectReference, MISPObjectAttribute, and MISPObject are subclasses of AbstractMISP, which mean that they can be handled as python dictionaries.

2.8 MISP Objects

Creating a new MISP object generator should be done using a pre-defined template and inherit AbstractMISPObjectGenerator.

Your new MISPObject generator need to generate attributes, and add them as class properties using add_attribute.

When the object is sent to MISP, all the class properties will be exported to the JSON export.

CHAPTER 3

pymisp

pymisp.**deprecated**(*func*)

This is a decorator which can be used to mark functions as deprecated. It will result in a warning being emitted when the function is used.

3.1 PyMISP

class pymisp.**PyMISP**(*url*, *key*, *ssl=True*, *out_type='json'*, *debug=None*, *proxies=None*, *cert=None*, *asynch=False*)

Python API for MISP

Parameters

- **url** – URL of the MISP instance you want to connect to
- **key** – API key of the user you want to use
- **ssl** – can be True or False (to check ot not the validity of the certificate. Or a CA_BUNDLE in case of self signed certiifcate (the concatenation of all the *.crt of the chain)
- **out_type** – Type of object (json) NOTE: XML output isn't supported anymore, keeping the flag for compatibility reasons.
- **debug** – Write all the debug information to stderr
- **proxies** – Proxy dict as describes here: http://docs.python-requests.org/en/master/user/advanced/#proxies
- **cert** – Client certificate, as described there: http://docs.python-requests.org/en/master/user/advanced/#client-side-certificates
- **asynch** – Use asynchronous processing where possible

add_asn(*event*, *asn*, *category='Network activity'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)

Add network ASN

add_attachment(*event*, *attachment*, *category='Artifacts dropped'*, *to_ids=False*, *comment=None*, *distribution=None*, *proposal=False*, *filename=None*, ***kwargs*)
Add an attachment to the MISP event

Parameters

- **event** – The event to add an attachment to
- **attachment** – Either a file handle or a path to a file - will be uploaded
- **filename** – Explicitly defined attachment filename

add_detection_name(*event*, *name*, *category='Antivirus detection'*, *to_ids=False*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add AV detection name(s)

add_domain(*event*, *domain*, *category='Network activity'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add domain(s)

add_domain_ip(*event*, *domain*, *ip*, *category='Network activity'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add domainlip

add_domains_ips(*event*, *domain_ips*, *category='Network activity'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add multiple domainlip

add_email_attachment(*event*, *email*, *category='Payload delivery'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add an email atachment

add_email_dst(*event*, *email*, *category='Payload delivery'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add a destination email

add_email_header(*event*, *email*, *category='Payload delivery'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add an email header

add_email_src(*event*, *email*, *category='Payload delivery'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add a source email

add_email_subject(*event*, *email*, *category='Payload delivery'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add an email subject

add_event(*event*)
Add a new event

Parameters **event** – Event as JSON object / string to add

add_feed(*source_format*, *url*, *name*, *input_source*, *provider*, ***kwargs*)
Delete a feed

add_filename(*event*, *filename*, *category='Artifacts dropped'*, *to_ids=False*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add filename(s)

add_hashes(*event*, *category='Artifacts dropped'*, *filename=None*, *md5=None*, *sha1=None*, *sha256=None*, *ssdeep=None*, *comment=None*, *to_ids=True*, *distribution=None*, *proposal=False*, ***kwargs*)
Add hashe(s) to an existing event

add_hostname(*event*, *hostname*, *category='Network activity'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add hostname(s)

add_internal_comment(*event*, *reference*, *category='Internal reference'*, *to_ids=False*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add an internal comment

add_internal_link(*event*, *reference*, *category='Internal reference'*, *to_ids=False*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add an internal link

add_internal_other(*event*, *reference*, *category='Internal reference'*, *to_ids=False*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add an internal reference (type other)

add_internal_text(*event*, *reference*, *category='Internal reference'*, *to_ids=False*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add an internal text

add_ipdst(*event*, *ipdst*, *category='Network activity'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add destination IP(s)

add_ipsrc(*event*, *ipsrc*, *category='Network activity'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add source IP(s)

add_mutex(*event*, *mutex*, *category='Artifacts dropped'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add mutex(es)

add_named_attribute(*event*, *type_value*, *value*, *category=None*, *to_ids=False*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add one or more attributes to an existing event

add_net_other(*event*, *netother*, *category='Network activity'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add a free text entry

add_object(*event_id*, **args*, ***kwargs*)
Add an object :param event_id: Event ID of the event to attach the object to :param template_id: Template ID of the template related to that event (not required) :param misp_object: MISPObject to attach

add_object_reference(*misp_object_reference*)
Add a reference to an object

add_other_comment(*event*, *reference*, *category='Other'*, *to_ids=False*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add other comment

add_other_counter(*event*, *reference*, *category='Other'*, *to_ids=False*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add other counter

add_other_text(*event*, *reference*, *category='Other'*, *to_ids=False*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add other text

add_pattern(*event*, *pattern*, *in_file=True*, *in_memory=False*, *category='Artifacts dropped'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add a pattern(s) in file or in memory

add_pipe(*event*, *named_pipe*, *category='Artifacts dropped'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add pipes(s)

add_regkey(*event*, *regkey*, *rvalue=None*, *category='Artifacts dropped'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add a registry key

add_regkeys(*event*, *regkeys_values*, *category='Artifacts dropped'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add a registry keys

add_snort(*event*, *snort*, *category='Network activity'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add SNORT rule(s)

add_target_email(*event*, *target*, *category='Targeting data'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add an target email

add_target_external(*event*, *target*, *category='Targeting data'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add an target external

add_target_location(*event*, *target*, *category='Targeting data'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add an target location

add_target_machine(*event*, *target*, *category='Targeting data'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add an target machine

add_target_org(*event*, *target*, *category='Targeting data'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add an target organisation

add_target_user(*event*, *target*, *category='Targeting data'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add an target user

add_threat_actor(*event*, *target*, *category='Attribution'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add an threat actor

add_traffic_pattern(*event*, *pattern*, *category='Network activity'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add pattern(s) in traffic

add_url(*event*, *url*, *category='Network activity'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add url(s)

add_useragent(*event*, *useragent*, *category='Network activity'*, *to_ids=True*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add user agent(s)

add_yara(*event*, *yara*, *category='Payload delivery'*, *to_ids=False*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add yara rule(es)

av_detection_link(*event*, *link*, *category='Antivirus detection'*, *to_ids=False*, *comment=None*, *distribution=None*, *proposal=False*, ***kwargs*)
Add AV detection link(s)

cache_all_feeds()
Alias for cache_feeds_all

cache_feed(*feed_id*)
Cache a specific feed

cache_feeds_all()
Cache all the feeds

cache_feeds_freetext()
Cache all the freetext feeds

cache_feeds_misp()
Cache all the MISP feeds

change_analysis_status(*event*, *analysis_status*)
Change the analysis status of an event

change_comment(*attribute_uuid*, *comment*)
Change the comment of attribute

change_sharing_group(*event*, *sharing_group_id*)
Change the sharing group of an event

change_threat_level(*event*, *threat_level_id*)
Change the threat level of an event

change_toids(*attribute_uuid*, *to_ids*)
Change the toids flag

compare_feeds()
Generate the comparison matrix for all the MISP feeds

delete_attribute(*attribute_id*, *hard_delete=False*)
Delete an attribute by ID

delete_event(*event_id*)
Delete an event

Parameters **event_id** – Event id to delete

delete_feed(*feed_id*)
Delete a feed

delete_object(*id*)
Deletes an object

delete_object_reference(*id*)
Deletes a reference to an object

download_all_suricata()
Download all suricata rules events.

download_last(*last*)
Download the last published events.

Parameters **last** – can be defined in days, hours, minutes (for example 5d or 12h or 30m)

download_samples(*sample_hash=None*, *event_id=None*, *all_samples=False*, *unzip=True*)
Download samples, by hash or event ID. If there are multiple samples in one event, use the all_samples switch

download_suricata_rule_event(*event_id*)
Download one suricata rule event.

Parameters **event_id** – ID of the event to download (same as get)

edit_feed(*feed_id*, ***kwargs*)
Delete a feed

edit_object(*misp_object*, *object_id=None*)
Edit an existing object

fast_publish(*event_id*, *alert=False*)
Does the same as the publish method, but just try to publish the event even with one single HTTP GET. The default is to not send a mail as it is assumed this method is called on update.

fetch_feed(*feed_id*)
Fetch one single feed

flatten_error_messages(*response*)
Dirty dirty method to normalize the error messages between the API calls. Any response containing the a key 'error' or 'errors' failed at some point, we make one single list out of it.

freetext(*event_id*, *string*, *adhereToWarninglists=False*, *distribution=None*, *returnMetaAttributes=False*)
Pass a text to the freetext importer

get(*eid*)
Get an event by event ID

get_all_attributes_txt(*type_attr*, *tags=False*, *eventId=False*, *allowNonIDS=False*, *date_from=False*, *date_to=False*, *last=False*, *enforceWarninglist=False*, *allowNotPublished=False*)
Get all attributes from a specific type as plain text. Only published and IDS flagged attributes are exported, except if stated otherwise.

get_all_tags(*quiet=False*)
Get all the tags used on the instance

get_api_version()
Returns the current version of PyMISP installed on the system

get_api_version_master()
Get the most recent version of PyMISP from github

get_attachment(*attribute_id*)
Get an attachement (not a malware sample) by attribute ID. Returns the attachment as a bytestream, or a dictionary containing the error message.

Parameters **attribute_id** – Attribute ID to fetched

get_attributes_statistics(*context='type'*, *percentage=None*)
Get attributes statistics from the MISP instance

get_csv(*eventid=None*, *attributes=[]*, *object_attributes=[]*, *misp_types=[]*, *context=False*, *ignore=False*, *last=None*)
Get MISP values in CSV format :param eventid: The event ID to query :param attributes: The column names to export from normal attributes (i.e. uuid, value, type, ...) :param object_attributes: The column names to export from attributes within objects (i.e. uuid, value, type, ...) :param misp_types: MISP types to get (i.e. ip-src, hostname, ...) :param context: Add event level context (event_info,event_member_org,event_source_org,event_distribution,event_threat_level_id,event_analysis,event_date,event_
:param ignore: Returns the attributes even if the event isn't published, or the attribute doesn't have the to_ids flag set

get_event(*event_id*)
Get an event

Parameters **event_id** – Event id to get

get_events_last_modified(*search_from*, *search_to=None*)
Download the last modified events.

Parameters

- **search_from** – Beginning of the interval. Can be either a timestamp, or a date (2000-12-21)
- **search_to** – End of the interval. Can be either a timestamp, or a date (2000-12-21)

get_feed(*feed_id*)
Get the content of a single feed

get_feeds_list()
Get the content of all the feeds

get_index(*filters=None*)
Return the index.

Warning, there's a limit on the number of results

get_live_query_acl()
This should return an empty list, unless the ACL is outdated.

get_object_template_id(*object_uuid*)
Gets the template ID corresponting the UUID passed as parameter

get_object_templates_list()
Returns the list of Object templates available on the MISP instance

get_recommended_api_version()
Returns the recommended API version from the server

get_roles_list()
Get the list of existing roles

get_sharing_groups()
Get the existing sharing groups

get_stix_event(*event_id=None*, *with_attachments=False*, *from_date=False*, *to_date=False*, *tags=False*)
Get an event/events in STIX format

get_tags_list()
Get the list of existing tags

get_tags_statistics(*percentage=None*, *name_sort=None*)
Get tags statistics from the MISP instance

get_version()
Returns the version of the instance.

get_version_master()
Get the most recent version from github

get_yara(*event_id*)
Get the yara rules from an event

new_event(*distribution=None*, *threat_level_id=None*, *analysis=None*, *info=None*, *date=None*, *published=False*, *orgc_id=None*, *org_id=None*, *sharing_group_id=None*)
Create and add a new event

new_tag(*name=None*, *colour='#00ace6'*, *exportable=False*, *hide_tag=False*)
Create a new tag

proposal_accept(*proposal_id*)
Accept a proposal

proposal_add(*event_id*, *attribute*)
Add a proposal

proposal_discard(*proposal_id*)
Discard a proposal

proposal_edit(*attribute_id*, *attribute*)
Edit a proposal

proposal_view(*event_id=None*, *proposal_id=None*)
View a proposal

publish(*event*, *alert=True*)
Publish event (with or without alert email) :param event: pass event or event id (as string or int) to publish :param alert: set to True by default (send alerting email) if False will not send alert :return publish status

pushEventToZMQ(*event_id*)
Force push an event on ZMQ

search(*controller='events'*, *async_callback=None*, ***kwargs*)
Search via the Rest API

Parameters

- **values** – values to search for
- **not_values** – values *not* to search for
- **type_attribute** – Type of attribute
- **category** – Category to search
- **org** – Org reporting the event
- **tags** – Tags to search for
- **not_tags** – Tags *not* to search for
- **date_from** – First date
- **date_to** – Last date
- **last** – Last published events (for example 5d or 12h or 30m)
- **eventid** – Evend ID(s) | str or list
- **withAttachments** – return events with or without the attachments
- **uuid** – search by uuid
- **publish_timestamp** – the publish timestamp
- **timestamp** – the timestamp of the last modification. Can be a list (from->to)
- **enforceWarninglist** – Enforce the warning lists
- **searchall** – full text search on the database
- **metadata** – return only metadata if True
- **published** – return only published events

- **to_ids** – return only the attributes with the to_ids flag set
- **deleted** – also return the deleted attributes
- **event_timestamp** – the timestamp of the last modification of the event (attributes controller only)). Can be a list (from->to)
- **async_callback** – The function to run when results are returned

search_all(*value*)
Search a value in the whole database

search_index(*published=None*, *eventid=None*, *tag=None*, *datefrom=None*, *dateuntil=None*, *eventinfo=None*, *threatlevel=None*, *distribution=None*, *analysis=None*, *attribute=None*, *org=None*, *async_callback=None*, *normalize=False*, *timestamp=None*)
Search only at the index level. Use ! infront of value as NOT, default OR If using async, give a callback that takes 2 args, session and response: basic usage is pymisp.search_index(..., async_callback=lambda ses,resp: print(resp.json()))

Parameters

- **published** – Published (0,1)
- **eventid** – Evend ID(s) | str or list
- **tag** – Tag(s) | str or list
- **datefrom** – First date, in format YYYY-MM-DD
- **dateuntil** – Last date, in format YYYY-MM-DD
- **eventinfo** – Event info(s) to match | str or list
- **threatlevel** – Threat level(s) (1,2,3,4) | str or list
- **distribution** – Distribution level(s) (0,1,2,3) | str or list
- **analysis** – Analysis level(s) (0,1,2) | str or list
- **org** – Organisation(s) | str or list
- **async_callback** – Function to call when the request returns (if running async)
- **normalize** – Normalize output | True or False
- **timestamp** – Interval since last update (in second, or 1d, 1h, ...)

set_sightings(*sightings*)
Push a sighting (python dictionary or MISPSighting) or a list of sightings

sharing_group_org_add(*sharing_group*, *organisation*, *extend=False*)
Add an organisation to a sharing group. :sharing_group: Sharing group's local instance ID, or Sharing group's global UUID :organisation: Organisation's local instance ID, or Organisation's global UUID, or Organisation's name as known to the curent instance :extend: Allow the organisation to extend the group

sharing_group_org_remove(*sharing_group*, *organisation*)
Remove an organisation from a sharing group. :sharing_group: Sharing group's local instance ID, or Sharing group's global UUID :organisation: Organisation's local instance ID, or Organisation's global UUID, or Organisation's name as known to the curent instance

sharing_group_server_add(*sharing_group*, *server*, *all_orgs=False*)
Add a server to a sharing group. :sharing_group: Sharing group's local instance ID, or Sharing group's global UUID :server: Server's local instance ID, or URL of the Server, or Server's name as known to the curent instance :all_orgs: Add all the organisations of the server to the group

sharing_group_server_remove(*sharing_group*, *server*)
Remove a server from a sharing group. :sharing_group: Sharing group's local instance ID, or Sharing group's global UUID :server: Server's local instance ID, or URL of the Server, or Server's name as known to the curent instance

sighting(*value=None*, *uuid=None*, *id=None*, *source=None*, *type=None*, *timestamp=None*, ***kwargs*)
Set a single sighting. :value: Value of the attribute the sighting is related too. Pushing this object

will update the sighting count of each attriutes with thifs value on the instance

Uuid UUID of the attribute to update

Id ID of the attriute to update

Source Source of the sighting

Type Type of the sighting

Timestamp Timestamp associated to the sighting

sighting_list(*element_id*, *scope='attribute'*, *org_id=False*)
Get the list of sighting. :param element_id: could be an event id or attribute id :type element_id: int :param scope: could be attribute or event :return: A json list of sighting corresponding to the search :rtype: list

Example

```
>>> misp.sighting_list(4731) # default search on attribute
[ ... ]
>>> misp.sighting_list(42, event) # return list of sighting for event 42
[ ... ]
>>> misp.sighting_list(element_id=42, org_id=2, scope=event) # return list of␣
↪sighting for event 42 filtered with org id 2
```

sighting_per_id(*attribute_id*)
Add a sighting to an attribute (by attribute ID)

sighting_per_json(*json_file*)
Push a sighting (JSON file)

sighting_per_uuid(*attribute_uuid*)
Add a sighting to an attribute (by attribute UUID)

tag(*uuid*, *tag*)
Tag an event or an attribute

test_connection()
Test the auth key

untag(*uuid*, *tag*)
Untag an event or an attribute

update(*event*)
Update an event by ID

update_attribute(*attribute_id*, *attribute*)
Update an attribute

Parameters

- **attribute_id** – Attribute id/uuid to update
- **attribute** – Attribute as JSON object / string to add

update_event(*event_id*, *event*)
Update an event

Parameters

- **event_id** – Event id to update
- **event** – Event as JSON object / string to add

upload_sample(*filename*, *filepath_or_bytes*, *event_id*, *distribution=None*, *to_ids=True*, *category=None*, *comment=None*, *info=None*, *analysis=None*, *threat_level_id=None*)
Upload a sample

upload_samplelist(*filepaths*, *event_id*, *distribution=None*, *to_ids=True*, *category=None*, *comment=None*, *info=None*, *analysis=None*, *threat_level_id=None*)
Upload a list of samples

view_feed(*feed_ids*)
Alias for get_feed

view_feeds()
Alias for get_feeds_list

3.2 MISPAbstract

class pymisp.**AbstractMISP**(**kwargs*)

edited
Recursively check if an object has been edited and update the flag accordingly to the parent objects

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

jsonable()
This method is used by the JSON encoder

properties
All the class public properties that will be dumped in the dictionary, and the JSON export. Note: all the properties starting with a _ (private), or listed in __not_jsonable will be skipped.

set_not_jsonable(**args*)
Set __not_jsonable to a new list

to_dict()
Dump the lass to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

to_json()
Dump recursively any class of type MISPAbstract to a json string

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

3.3 MISPEncode

class pymisp.**MISPEncode**(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

default(*obj*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

3.4 MISPEvent

class pymisp.**MISPEvent**(*describe_types=None*, *strict_validation=False*, ***kwargs*)

add_attribute(*type*, *value*, ***kwargs*)

Add an attribute. type and value are required but you can pass all other parameters supported by MISPAttribute

add_attribute_tag(*tag*, *attribute_identifier*)

Add a tag to an existing attribute, raise an Exception if the attribute doesn't exists. :tag: Tag name as a string, MISPTag instance, or dictionary :attribute_identifier: can be an ID, UUID, or the value.

add_object(*obj=None*, ***kwargs*)

Add an object to the Event, either by passing a MISPObject, or a dictionary

add_proposal(*shadow_attribute=None*, ***kwargs*)

Alias for add_shadow_attribute

add_shadow_attribute(*shadow_attribute=None*, ***kwargs*)

Add a tag to the attribute (by name or a MISPTag object)

clear() → None. Remove all items from D.

delete_attribute(*attribute_id*)

Delete an attribute, you can search by ID or UUID

edited

Recursively check if an object has been edited and update the flag accordingly to the parent objects

from_dict(***kwargs*)

Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

get_attribute_tag(*attribute_identifier*)
Return the tags associated to an attribute or an object attribute. :attribute_identifier: can be an ID, UUID, or the value.

get_object_by_id(*object_id*)
Get an object by ID (the ID is the one set by the server when creating the new object)

get_object_by_uuid(*object_uuid*)
Get an object by UUID (UUID is set by the server when creating the new object)

items() → a set-like object providing a view on D's items

jsonable()
This method is used by the JSON encoder

keys() → a set-like object providing a view on D's keys

load(*json_event*)
Load a JSON dump from a pseudo file or a JSON string

load_file(*event_path*)
Load a JSON dump from a file on the disk

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

properties
All the class public properties that will be dumped in the dictionary, and the JSON export. Note: all the properties starting with a _ (private), or listed in __not_jsonable will be skipped.

publish()
Mark the attribute as published

set_date(*date*, *ignore_invalid=False*)
Set a date for the event (string, datetime, or date object)

set_not_jsonable(**args*)
Set __not_jsonable to a new list

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict()
Dump the lass to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

to_json()
Dump recursively any class of type MISPAbstract to a json string

unpublish()
Mark the attribute as un-published (set publish flag to false)

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

values() → an object providing a view on D's values

3.5 MISPAttribute

class pymisp.**MISPAttribute**(*describe_types=None*, *strict=False*)

add_proposal(*shadow_attribute=None*, ***kwargs*)
Alias for add_shadow_attribute

add_shadow_attribute(*shadow_attribute=None*, ***kwargs*)
Add a tag to the attribute (by name or a MISPTag object)

clear() → None. Remove all items from D.

delete()
Mark the attribute as deleted (soft delete)

edited
Recursively check if an object has been edited and update the flag accordingly to the parent objects

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

jsonable()
This method is used by the JSON encoder

keys() → a set-like object providing a view on D's keys

known_types
Returns a list of all the known MISP attributes types

malware_binary
Returns a BytesIO of the malware (if the attribute has one, obvs).

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

properties
All the class public properties that will be dumped in the dictionary, and the JSON export. Note: all the properties starting with a _ (private), or listed in __not_jsonable will be skipped.

set_not_jsonable(**args*)
Set __not_jsonable to a new list

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict()
Dump the lass to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

to_json()
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

values() → an object providing a view on D's values

3.6 MISPObject

class pymisp.**MISPObject**(*name*, *strict=False*, *standalone=False*, *default_attributes_parameters={}*, ***kwargs*)

add_attribute(*object_relation*, ***value*)
Add an attribute. object_relation is required and the value key is a dictionary with all the keys supported by MISPAttribute

add_reference(*referenced_uuid*, *relationship_type*, *comment=None*, ***kwargs*)
Add a link (uuid) to an other object

clear() → None. Remove all items from D.

edited
Recursively check if an object has been edited and update the flag accordingly to the parent objects

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

get_attributes_by_relation(*object_relation*)
Returns the list of attributes with the given object relation in the object

has_attributes_by_relation(*list_of_relations*)
True if all the relations in the list are defined in the object

items() → a set-like object providing a view on D's items

jsonable()
This method is used by the JSON encoder

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

properties
All the class public properties that will be dumped in the dictionary, and the JSON export. Note: all the properties starting with a _ (private), or listed in __not_jsonable will be skipped.

set_not_jsonable(**args*)
Set __not_jsonable to a new list

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*strict=False*)
Dump the lass to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

to_json(*strict=False*)
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

values() → an object providing a view on D's values

3.7 MISPObjectAttribute

class pymisp.**MISPObjectAttribute**(*definition*)

add_proposal(*shadow_attribute=None*, ***kwargs*)
Alias for add_shadow_attribute

add_shadow_attribute(*shadow_attribute=None*, ***kwargs*)
Add a tag to the attribute (by name or a MISPTag object)

clear() → None. Remove all items from D.

delete()
Mark the attribute as deleted (soft delete)

edited
Recursively check if an object has been edited and update the flag accordingly to the parent objects

from_dict(*object_relation*, *value*, ***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

jsonable()
This method is used by the JSON encoder

keys() → a set-like object providing a view on D's keys

known_types
Returns a list of all the known MISP attributes types

malware_binary
Returns a BytesIO of the malware (if the attribute has one, obvs).

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

properties
All the class public properties that will be dumped in the dictionary, and the JSON export. Note: all the properties starting with a _ (private), or listed in __not_jsonable will be skipped.

set_not_jsonable(**args*)
Set __not_jsonable to a new list

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict()
Dump the lass to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

to_json()
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

values() → an object providing a view on D's values

3.8 MISPObjectReference

class pymisp.**MISPObjectReference**

clear() → None. Remove all items from D.

edited
Recursively check if an object has been edited and update the flag accordingly to the parent objects

from_dict(*object_uuid*, *referenced_uuid*, *relationship_type*, *comment=None*, ***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

jsonable()
This method is used by the JSON encoder

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

properties
All the class public properties that will be dumped in the dictionary, and the JSON export. Note: all the properties starting with a _ (private), or listed in __not_jsonable will be skipped.

set_not_jsonable(**args*)
Set __not_jsonable to a new list

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict()
Dump the lass to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

to_json()
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

values() → an object providing a view on D's values

3.9 MISPTag

class pymisp.**MISPTag**

clear() → None. Remove all items from D.

edited
Recursively check if an object has been edited and update the flag accordingly to the parent objects

from_dict(*name*, ***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

jsonable()
This method is used by the JSON encoder

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

properties
All the class public properties that will be dumped in the dictionary, and the JSON export. Note: all the properties starting with a _ (private), or listed in __not_jsonable will be skipped.

set_not_jsonable(**args*)
Set __not_jsonable to a new list

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict()
Dump the lass to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

to_json()
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

values() → an object providing a view on D's values

3.10 MISPUser

class pymisp.**MISPUser**

clear() → None. Remove all items from D.

edited
Recursively check if an object has been edited and update the flag accordingly to the parent objects

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

jsonable()
This method is used by the JSON encoder

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

properties
All the class public properties that will be dumped in the dictionary, and the JSON export. Note: all the properties starting with a _ (private), or listed in __not_jsonable will be skipped.

set_not_jsonable(**args*)
Set __not_jsonable to a new list

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict()
Dump the lass to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

to_json()
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

values() → an object providing a view on D's values

3.11 MISPOrganisation

class pymisp.**MISPOrganisation**

clear() → None. Remove all items from D.

edited
Recursively check if an object has been edited and update the flag accordingly to the parent objects

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

items() → a set-like object providing a view on D's items

jsonable()
This method is used by the JSON encoder

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

properties
All the class public properties that will be dumped in the dictionary, and the JSON export. Note: all the properties starting with a _ (private), or listed in __not_jsonable will be skipped.

set_not_jsonable(**args*)
Set __not_jsonable to a new list

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict()
Dump the lass to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

to_json()
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

values() → an object providing a view on D's values

CHAPTER 4

pymisp - Tools

4.1 File Object

class pymisp.tools.**FileObject**(*filepath=None*, *pseudofile=None*, *filename=None*, *standalone=True*, ***kwargs*)

add_attribute(*object_relation*, ***value*)
Add an attribute. object_relation is required and the value key is a dictionary with all the keys supported by MISPAttribute

add_reference(*referenced_uuid*, *relationship_type*, *comment=None*, ***kwargs*)
Add a link (uuid) to an other object

clear() → None. Remove all items from D.

edited
Recursively check if an object has been edited and update the flag accordingly to the parent objects

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

generate_attributes()
Contains the logic where all the values of the object are gathered

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

get_attributes_by_relation(*object_relation*)
Returns the list of attributes with the given object relation in the object

has_attributes_by_relation(*list_of_relations*)
True if all the relations in the list are defined in the object

items() → a set-like object providing a view on D's items

jsonable()
This method is used by the JSON encoder

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

properties
All the class public properties that will be dumped in the dictionary, and the JSON export. Note: all the properties starting with a _ (private), or listed in __not_jsonable will be skipped.

set_not_jsonable(**args*)
Set __not_jsonable to a new list

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*strict=False*)
Dump the lass to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

to_json(*strict=False*)
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

values() → an object providing a view on D's values

4.2 ELF Object

class pymisp.tools.**ELFObject**(*parsed=None*, *filepath=None*, *pseudofile=None*, *standalone=True*, ***kwargs*)

add_attribute(*object_relation*, ***value*)
Add an attribute. object_relation is required and the value key is a dictionary with all the keys supported by MISPAttribute

add_reference(*referenced_uuid*, *relationship_type*, *comment=None*, ***kwargs*)
Add a link (uuid) to an other object

clear() → None. Remove all items from D.

edited
Recursively check if an object has been edited and update the flag accordingly to the parent objects

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

generate_attributes()
Contains the logic where all the values of the object are gathered

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

get_attributes_by_relation(*object_relation*)
Returns the list of attributes with the given object relation in the object

has_attributes_by_relation(*list_of_relations*)
True if all the relations in the list are defined in the object

items() → a set-like object providing a view on D's items

jsonable()
This method is used by the JSON encoder

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

properties
All the class public properties that will be dumped in the dictionary, and the JSON export. Note: all the properties starting with a _ (private), or listed in __not_jsonable will be skipped.

set_not_jsonable(**args*)
Set __not_jsonable to a new list

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*strict=False*)
Dump the lass to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

to_json(*strict=False*)
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

values() → an object providing a view on D's values

class pymisp.tools.**ELFSectionObject**(*section*, *standalone=True*, ***kwargs*)

add_attribute(*object_relation*, ***value*)
Add an attribute. object_relation is required and the value key is a dictionary with all the keys supported by MISPAttribute

add_reference(*referenced_uuid*, *relationship_type*, *comment=None*, ***kwargs*)
Add a link (uuid) to an other object

clear() → None. Remove all items from D.

edited
Recursively check if an object has been edited and update the flag accordingly to the parent objects

from_dict(**kwargs)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

generate_attributes()
Contains the logic where all the values of the object are gathered

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

get_attributes_by_relation(*object_relation*)
Returns the list of attributes with the given object relation in the object

has_attributes_by_relation(*list_of_relations*)
True if all the relations in the list are defined in the object

items() → a set-like object providing a view on D's items

jsonable()
This method is used by the JSON encoder

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

properties
All the class public properties that will be dumped in the dictionary, and the JSON export. Note: all the properties starting with a _ (private), or listed in __not_jsonable will be skipped.

set_not_jsonable(*args)
Set __not_jsonable to a new list

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*strict=False*)
Dump the lass to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

to_json(*strict=False*)
Dump recursively any class of type MISPAbstract to a json string

update([*E*], **F) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(*args)
Add entries to the __not_jsonable list

values() → an object providing a view on D's values

4.3 PE Object

class pymisp.tools.**PEObject**(*parsed=None*, *filepath=None*, *pseudofile=None*, *standalone=True*, ***kwargs*)

add_attribute(*object_relation*, ***value*)
Add an attribute. object_relation is required and the value key is a dictionary with all the keys supported by MISPAttribute

add_reference(*referenced_uuid*, *relationship_type*, *comment=None*, ***kwargs*)
Add a link (uuid) to an other object

clear() → None. Remove all items from D.

edited
Recursively check if an object has been edited and update the flag accordingly to the parent objects

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

generate_attributes()
Contains the logic where all the values of the object are gathered

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

get_attributes_by_relation(*object_relation*)
Returns the list of attributes with the given object relation in the object

has_attributes_by_relation(*list_of_relations*)
True if all the relations in the list are defined in the object

items() → a set-like object providing a view on D's items

jsonable()
This method is used by the JSON encoder

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

properties
All the class public properties that will be dumped in the dictionary, and the JSON export. Note: all the properties starting with a _ (private), or listed in __not_jsonable will be skipped.

set_not_jsonable(**args*)
Set __not_jsonable to a new list

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*strict=False*)
Dump the lass to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

to_json(*strict=False*)
Dump recursively any class of type MISPAbstract to a json string

update(*[E]*, ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

values() → an object providing a view on D's values

class pymisp.tools.**PESectionObject**(*section*, *standalone=True*, ***kwargs*)

add_attribute(*object_relation*, ***value*)
Add an attribute. object_relation is required and the value key is a dictionary with all the keys supported by MISPAttribute

add_reference(*referenced_uuid*, *relationship_type*, *comment=None*, ***kwargs*)
Add a link (uuid) to an other object

clear() → None. Remove all items from D.

edited
Recursively check if an object has been edited and update the flag accordingly to the parent objects

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

generate_attributes()
Contains the logic where all the values of the object are gathered

get(*k[, d]*) → D[k] if k in D, else d. d defaults to None.

get_attributes_by_relation(*object_relation*)
Returns the list of attributes with the given object relation in the object

has_attributes_by_relation(*list_of_relations*)
True if all the relations in the list are defined in the object

items() → a set-like object providing a view on D's items

jsonable()
This method is used by the JSON encoder

keys() → a set-like object providing a view on D's keys

pop(*k[, d]*) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

properties
All the class public properties that will be dumped in the dictionary, and the JSON export. Note: all the properties starting with a _ (private), or listed in __not_jsonable will be skipped.

set_not_jsonable(**args*)
Set __not_jsonable to a new list

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*strict=False*)
Dump the lass to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

to_json(*strict=False*)
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

values() → an object providing a view on D's values

4.4 Mach-O Object

class pymisp.tools.**MachOObject**(*parsed=None*, *filepath=None*, *pseudofile=None*, *standalone=True*, ***kwargs*)

add_attribute(*object_relation*, ***value*)
Add an attribute. object_relation is required and the value key is a dictionary with all the keys supported by MISPAttribute

add_reference(*referenced_uuid*, *relationship_type*, *comment=None*, ***kwargs*)
Add a link (uuid) to an other object

clear() → None. Remove all items from D.

edited
Recursively check if an object has been edited and update the flag accordingly to the parent objects

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

generate_attributes()
Contains the logic where all the values of the object are gathered

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

get_attributes_by_relation(*object_relation*)
Returns the list of attributes with the given object relation in the object

has_attributes_by_relation(*list_of_relations*)
True if all the relations in the list are defined in the object

items() → a set-like object providing a view on D's items

jsonable()
This method is used by the JSON encoder

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

properties
All the class public properties that will be dumped in the dictionary, and the JSON export. Note: all the properties starting with a _ (private), or listed in __not_jsonable will be skipped.

set_not_jsonable(**args*)
Set __not_jsonable to a new list

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*strict=False*)
Dump the lass to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

to_json(*strict=False*)
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

values() → an object providing a view on D's values

class pymisp.tools.**MachOSectionObject**(*section*, *standalone=True*, ***kwargs*)

add_attribute(*object_relation*, ***value*)
Add an attribute. object_relation is required and the value key is a dictionary with all the keys supported by MISPAttribute

add_reference(*referenced_uuid*, *relationship_type*, *comment=None*, ***kwargs*)
Add a link (uuid) to an other object

clear() → None. Remove all items from D.

edited
Recursively check if an object has been edited and update the flag accordingly to the parent objects

from_dict(***kwargs*)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

generate_attributes()
Contains the logic where all the values of the object are gathered

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

get_attributes_by_relation(*object_relation*)
Returns the list of attributes with the given object relation in the object

has_attributes_by_relation(*list_of_relations*)
True if all the relations in the list are defined in the object

items() → a set-like object providing a view on D's items

jsonable()
This method is used by the JSON encoder

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

properties
All the class public properties that will be dumped in the dictionary, and the JSON export. Note: all the properties starting with a _ (private), or listed in __not_jsonable will be skipped.

set_not_jsonable(**args*)
Set __not_jsonable to a new list

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*strict=False*)
Dump the lass to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

to_json(*strict=False*)
Dump recursively any class of type MISPAbstract to a json string

update([*E*], ***F*) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(**args*)
Add entries to the __not_jsonable list

values() → an object providing a view on D's values

4.5 VT Report Object

class pymisp.tools.**VTReportObject**(*apikey*, *indicator*, *vt_proxies=None*, *standalone=True*, ***kwargs*)

VirusTotal Report

Apikey VirusTotal API key (private works, but only public features are supported right now)

Indicator IOC to search VirusTotal for

add_attribute(*object_relation*, ***value*)
Add an attribute. object_relation is required and the value key is a dictionary with all the keys supported by MISPAttribute

add_reference(*referenced_uuid*, *relationship_type*, *comment=None*, ***kwargs*)
Add a link (uuid) to an other object

clear() → None. Remove all items from D.

edited
Recursively check if an object has been edited and update the flag accordingly to the parent objects

from_dict(**kwargs)
Loading all the parameters as class properties, if they aren't *None*. This method aims to be called when all the properties requiring a special treatment are processed. Note: This method is used when you initialize an object with existing data so by default, the class is flaged as not edited.

from_json(*json_string*)
Load a JSON string

generate_attributes()
Parse the VirusTotal report for relevant attributes

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

get_attributes_by_relation(*object_relation*)
Returns the list of attributes with the given object relation in the object

has_attributes_by_relation(*list_of_relations*)
True if all the relations in the list are defined in the object

items() → a set-like object providing a view on D's items

jsonable()
This method is used by the JSON encoder

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair
as a 2-tuple; but raise KeyError if D is empty.

properties
All the class public properties that will be dumped in the dictionary, and the JSON export. Note: all the properties starting with a _ (private), or listed in __not_jsonable will be skipped.

set_not_jsonable(*args)
Set __not_jsonable to a new list

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict(*strict=False*)
Dump the lass to a dictionary. This method automatically removes the timestamp recursively in every object that has been edited is order to let MISP update the event accordingly.

to_json(*strict=False*)
Dump recursively any class of type MISPAbstract to a json string

update([*E*], **F) → None. Update D from mapping/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_not_jsonable(*args)
Add entries to the __not_jsonable list

values() → an object providing a view on D's values

4.6 STIX

pymisp.tools.stix.**load_stix**(*stix*, *distribution=3*, *threat_level_id=2*, *analysis=0*)
Returns a MISPEvent object from a STIX package

pymisp.tools.stix.**make_stix_package**(*misp_event*, *to_json=False*, *to_xml=False*)
Returns a STIXPackage from a MISPEvent.

Optionally can return the package in json or xml.

4.7 OpenIOC

tools.**load_openioc**()

tools.**load_openioc_file**()

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

A

C

D

E

F

G

H

I

J

K

L

M

N

P

S

T

U

V