Deep-dive into PyMISP

MISP - Malware Information Sharing Platform & Threat Sharing

CIRCL
Computer Incident
Response Center
Luxembourg

Team CIRCL

http://www.misp-project.org/
Twitter: *@MISPProject*

Univ. Lorraine
20181124

Context

- MISP is a large project
- Your production environment is even more complex
- 3rd party services are even worse
- Querying MISP via CURL is doable, but get's painful fast
- Talking to MySQL directly can be dangerous
- POST a JSON blob, receive a JSON blob. You can do it manually(-ish)

Big picture

- Core goal: providing stable access to APIs, respect access control
- Simplifying handling & automation of indicators in 3rd party tools
- Hiding complexity of the JSON blobs
- Providing pre-cooked examples for commonly used operations
- Helping integration with existing infrastructure

Common queries: Recent changes on a timeframe

There are 4 main cases here:

- Metadata of the events that have been modified
  - **search_index** ⇒ timestamp (1h, 1d, 7d, ...), returns list of all the modified events
- Full events (metadata + attributes)
  - **search** ⇒ timestamp (1h, 1d, 7d, ...)
- Modified attributes
  - **search** ⇒ controller = attributes and timestamp (1h, 1d, 7d, ...)
- Other use case: get last **published** events by using the last parameter in the **search** method.

Common queries: Search things

There are 3 main cases here:

- Easy, but slow: full text search with **search_all**
- Faster: use the **search** method and search by tag, type, enforce the warning lists, with(-out) attachments, dates interval, ...
- Get malware samples (if available on the instance).

Common queries: create things

There are 3 main cases here:

- Add Event, edit its metadata
- Add attributes or objects to event
- (un)Tag event or attribute (soon object)
- Edit Attributes medatada
- Upload malware sample (and automatically expand it)

Administrative tasks

Assyming you have the right to do it on the instance.

- Managing users
- Managing organisations
- Managing sync servers

Other Capabilities

- Upload/download samples
- **Proposals**: add, edit, accept, discard
- **Sightings**: Get, set, update
- Export **statistics**
- Manage **feeds**
- Get MISP server version, recommended PyMISP version
- And more, look at the api file

MISPEvent - Usecase

```
from pymisp import MISPEvent, EncodeUpdate

# Create a new event with default values
event = MISPEvent()

# Load an existing JSON dump (optional)
event.load_file('Path/to/event.json')
event.info = 'My␣cool␣event'  # Duh.

# Add an attribute of type ip-dst
event.add_attribute('ip-dst', '8.8.8.8')

# Mark an attribute as deleted (From 2.4.60)
event.delete_attribute('<Attribute␣UUID>')

# Dump as json
event_as_jsondump = json.dumps(event, cls=EncodeUpdate)
```

Basics

- Python 3.5+ is recommended
- PyMISP is always inline with current version (pip3 install pymisp)
- Dev version: pip3 install git+https://github.com/MISP/PyMISP.git
- Get your auth key from: `https://misppriv.circl.lu/events/automation`
  - Not available: you don't have "Auth key access" role. Contact your instance admin.
- Source available here: git clone `https://github.com/MISP/PyMISP.git`

Examples

- **PyMISP needs to be installed (duh)**
- Usage:
  - Create examples/keys.py with the following content

```
misp_url = "https://url-to-your-misp"
misp_key = "<API_KEY>"
misp_verifycert = True
```

- Proxy support:

```
proxies = {
        'http': 'http://127.0.0.1:8123',
        'https': 'http://127.0.0.1:8123',
}
PyMISP(misp_url, misp_key, misp_verifycert, proxies=proxies)
```

Examples

- Lots of ideas on how to use the API
- You may also want to look at the tests directory
- All the examples use argparse. Help usage is available: **script.py -h**
    - **add_file_object.py**: Attach a file (PE/ELF/Mach-O) object to an event
    - **upload.py**: Upload a malware sample (use advanced expansion is available on the server)
    - **last.py**: Returns all the most recent events (on a timeframe)
    - **add_named_attribute.py**: Add attribute to an event
    - **sighting.py**: Update sightings on an attribute
    - **stats.py**: Returns the stats of a MISP instance
    - {**add,edit,create**}_**user.py** : Add, Edit, Create a user on MISP

Usage

- Basic example

```
from pymisp import PyMISP
api = PyMISP(url, apikey, verifycert=True, debug=False, proxies=None)
response = api.<function>
if response['error']:
        # <something went wrong>
else:
        # <do something with the output>
```

Concept behind AbstractMISP

- JSON blobs are python dictionaries
- ... Accessing content can be a pain
- **AbstractMISP inherits collections.MutableMapping**, they are all dictionaries!
- ... Has helpers to load, dump, and edit JSON blobs
- **Important**: All the public attributes (not starting with a _) defined in a class are dumped to JSON
- **Tags**: Events and Attributes have tags, soon Objects. Tag handling is defined in this class.
- **edited**: When pushing a full MISPEvent, only the objects without a timestamp, or with a newer timestamp will be updated. This method recursively finds updated events, and removes the timestamp key from the object.

MISPEvent, MISPAttribute, MISPObject, MISPSighting...

- **Pythonic** representation of MISP elements
- **Easy manipulation**
  - Load an existing event
  - Update te metadata, add attributes, objects, tags, mark an attribute as deleted, ...
  - Set relations between objects
  - Load and add attachments or malware samples as pseudo files
- **Dump** to JSON

MISPEvent - Main entrypoints

- load_file(event_path)
- load(json_event)
- add_attribute(type, value, **kwargs)
- add_object(obj=None, **kwargs)
- add_attribute_tag(tag, attribute_identifier)
- get_attribute_tag(attribute_identifier)
- add_tag(tag=None, **kwargs)
- objects[], attributes[], tags[]
- edited, all other paramaters of the MISPEvent element (info, date, ...)
- to_json()

MISPObject - Main entrypoints

- add_attribute(object_relation, **value)
- add_reference(referenced_uuid, relationship_type, comment=None, **kwargs)
- has_attributes_by_relation(list_of_relations)
- get_attributes_by_relation(object_relation)
- attributes[], relations[]
- edited, all other paramaters of the MISPObject element (name, comment, ...)
- to_json()
- Can be validated against their template
- Can have default parameters applied to all attributes (i.e. distribution, category, ...)

MISPAttribute - Main entrypoints

- add_tag(tag=None, **kwargs)
- delete()
- malware_binary (if relevant)
- tags[]
- edited, all other paramaters of the MISPObject element (value, comment, ...)
- to_json()

PyMISP - Tools

- Libraries requiring specfic 3rd party dependencies
- Callable via PyMISP for specific usecases
- Curently implemented:
  - **OpenIOC** to MISP Event
  - MISP to **Neo4J**

PyMISP - Default objects generators

- File - PE/ELF/MachO - Sections
- VirusTotal
- Generic object generator

PyMISP - Logging / Debugging

- debug=True passed to the constructor enable debug to stdout
- Configurable using the standard logging module
- Show everything send to the server and received by the client

```
import pymisp
import logging

logger = logging.getLogger('pymisp')
logger.setLevel(logging.DEBUG)  # enable debug to stdout

logging.basicConfig(level=logging.DEBUG,    # Enable debug to file
                    filename="debug.log",
                    filemode='w',
                    format=pymisp.FORMAT)
```

Q&A

MISP
Threat Sharing

- https://github.com/MISP/PyMISP
- https://github.com/MISP/
- https://pymisp.readthedocs.io/
- We welcome new functionalities and pull requests.