Deep-div

Deep-dive into PyMISP

DEEP-DIVE INTO PYMISP
MISP - THREAT SHARING

HTTP://WWW.MISP-PROJECT.ORG TWITTER: @MISPPROJECT



DEEP-DIVE INTO PYMISP

MISP - THREAT SHARING

CIRCL / TEAM MISP PROJECT

HTTP://WWW.MISP-PROJECT.ORG/

TWITTER: @MISPPROJECT

MISP PROJECT



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)

Deep-dive into PyMISP

└─Context

XT

MISP is a large project.

Your production environment is even more complex
 3rd party services are even worse

- 3rd party services are even worse
 Ouerving MISP via CURL is doable, but get's pain
- Querying MISP via CURL is doable, but get's painfu
 Talking to MySOL directly can be dangerous
- POST a JSON blob, receive a JSON blo 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

Deep-dive into PyMISP

∟Big picture

TURE

Core goal: providing stable access to APIs, respect access control

 Simplifying handling & automation of indicators in 3rd p tools

Hiding complexity of the JSON blobs
 Providing pro-cooked examples for a

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** \Rightarrow 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.

Deep-dive into PyMISP

2024-04-15

-Common queries: Recent changes on a timeframe MON QUERIES: RECENT CHANGES ON A TIMEFRAME

There are 4 main cases here:

• Metadata of the events that have been modified.

 search_index -> timestamp (th, 1d, 7d, ...), returns list of a the modified events
 Full events (metadata + attributes)

➤ search ⇒ timestamp (th, td, 7d, ...)

▶ search ⇒ controller = attributes and timestamp (sh, sd, 7d, ...
 ■ Other use case: get last published events by using the last

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).

Deep-dive into PyMISP

∟_Co

-Common queries: Search things

COMMON QUERIES: SEARCH THINGS

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)

Deep-dive into PyMISP

-Common queries: create things

Add Event, edit its metadata

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

Deep-dive into PyMISP

-Administrative tasks

MINISTRATIVE TASKS

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

Deep-dive into PyMISP

-Other Capabilities

- Upload/download samples
 Proposals: add, edit, accept, discard
- m 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)
```

Deep-dive into PyMISP

-MISPEvent - Usecase

SPEVENT - USECASE

from pyminp import MISPEvent, EncodeUpdate

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

Load on existing | SON dump (option event.load_file('Path/to/event.json' event.info = 'My cool event' # Duh.

Add on ottribute of type ip-dat event.add_attribute('ip-dat', '8.8.8.8') # Mork on attribute as deleted (from 2.4.60) event dalate attribute ('Attribute Willor')

Dump on joon
event_as_joondump = joon.dumps(event, cls=EncodeUp

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

Deep-dive into PyMISP

—Basics

cs

Python 3.5+ is recommended
PyMISP is always inline with current version (pip3 install

pymisp)

B Dev version: pip3 install

git+https://github.com/MISP/PyMISP.g Get your auth key from:

Not available: you don't have "Auth key access" role. Co your instance admin.

 Source available here: git clone

https://github.com/MISP/PyMIS

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)
```

Deep-dive into PyMISP

2024-04-15

-Examples

m PyMISP needs to be installed (duh) # Usage:

■ Proxy support:

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

Deep-dive into PyMISP

-Examples

- You may also want to look at the tests directory
- ► add_file_object.py: Attach a file (PE/EI
- upload.py: Upload a malware sample (use adv

- ► {add,edit,create}_user.py : Add, Edit, Create a user on MISI

USAGE

■ Basic example

Deep-dive into PyMISP

- Usage

2

2024-04-15

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.

Deep-dive into PyMISP

2024-04-15 DE

-Concept behind AbstractMISP

NCEPT BEHIND ABSTRACTMISP

- JSON blobs are python dictionaries
- Accessing content can be a pain
 AbstractMISP inherits collections.MutableMapping. they are
- Has helpers to load, dump, and edit JSON blobs
 Has helpers to load, dump, and edit JSON blobs
- Important: All the public attributes (not starting with a _) defined in a class are dumned to ISON
 - defined in a class are dumped to JSON
 - Tags: Events and Attributes have tags, soon Objects. Tag handling is defined in this class.
 - maintaining's definited in this class; a 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, an

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

Deep-dive into PyMISP

-MISPEvent, MISPAttribute, MISPObject, MISPSighting...

NT, MISPATTRIBUTE, MISPOBJECT, HTING...

- m Pythonic representation of MISP elements
 m Easy manipulation
- Load an existing event
 Update te metadata, add attributes, objects, tags, mari
- attribute as deleted, ...

 Set relations between objects

 Load and add attributents or makeum complex or occur
- 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()

Deep-dive into PyMISP

2024-04-15 G

-MISPEvent - Main entrypoints

ISPEVENT - MAIN ENTRYPOINTS

- load_file(event_path) ■ load(json_event)
- dd_attribute(type, value, **kwa
- add_attribute(type, value, -- kwa
- add_object(obj=None, **kwargs)
- set attribute tag(ag, attribute identific
- 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, ...)

Deep-dive into PyMISP

-MISPObject - Main entrypoints

- add attribute(object relation, **value)

- m edited, all other paramaters of the MISPObject element
- m to ison()
- m Can be validated against their template
- Can have default parameters applied to all attributes (i.e.

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 = 100ES

➤ OpenIOC to MISP Event

► MISP to Neo4J

Libraries requiring specfic 3rd party dependencies
 Callable via PyMISP for specific usecases
 Ourently implemented:

PyMISP - Default objects generators

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

Deep-dive into PyMISP

PyMISP - Default objects generators

SP - 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

Deep-dive into PyMISP

2024-04-

└─PyMISP - Logging / Debugging

IISP - LOGGING / DEBUGGING

debug=True passed to the constructor enable debug to

Configurable using the standard logging module
 Show everything send to the server and received by

import pynisp import logging

logger = logging .getlogger('pymisp')
logger.settava((logging DEBUG) # enable debug to zndout
logging.basicConfig((level-logging.DEBUG, # Enable debug to
linearmer*debug log*

p.FORMAT)

Q&A



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

Deep-dive into PyMISP

-Q&A

