

CTI Operational Procedures with Jupyter Notebooks and MISP

CTIS - October 19/20 - 2022

cudeso.be
We Secure You

<https://www.cudeso.be>
koen.vanimpe@cudeso.be

Koen Van Impe

- **Freelancer**
 - Incident response, threat intelligence, security monitoring
 - **Open source contributions**
 - MISP modules, taxonomies, automation and integration with DFIR tools, ...
 - “*MISP tip-of-the-week*”
 - **BelgoMISP**
 - Belgian MISP User Group
 - **OSINT threat feed**
 - botvrij.eu
- ✉ koen.vanimpe@cudeso.be

🌐 <https://www.cudeso.be>

🌐 <https://www.vanimpe.eu>

⌚ <https://github.com/cudeso>

🐦 @cudeso

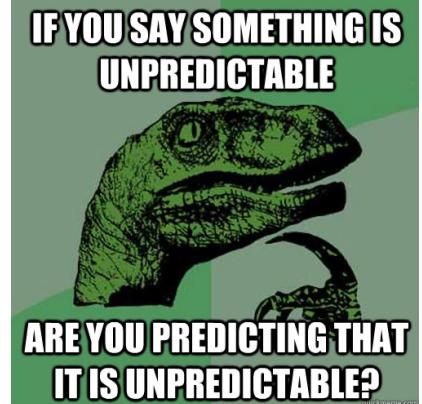
CTI operational procedures

Operational procedures?

Playbooks

Standard
Operating
Procedures
(SOP)

Workflows



Consistent
approach

Recipe for an
investigation

Repeatable

Predictable

Completeness
checks

Documented
actions

Leads up to
automation

Typical formats of operating procedures, workflows or playbooks

- **Markdown**, stored in a wiki / GitLab / GitHub
- **Security Playbooks**: CACAO JSON, shared via MISP objects
 - Collaborative Automated Course of Action Operations (CACAO) security playbooks
- And others ...

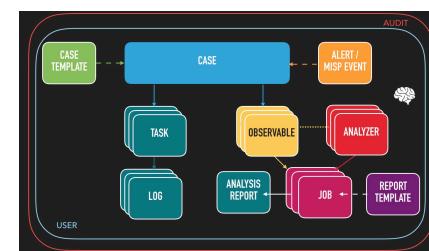


The screenshot shows a GitHub repository page for 'MicrosoftDocs / security'. The repository is public and has 108 issues and 23 pull requests. The 'Code' tab is selected, and the URL is [security / compass / incident-response-playbook-app-consent.md](#). The file content is a CACAO JSON object:

```
security-playbook
```

The security-playbook object provides meta-information and allows managing, storing, and sharing cybersecurity playbooks and orchestration workflows.

i security-playbook is a MISP object available in JSON format at [this location](#). The JSON format can be freely reused in your application or automatically enabled in [MISP](#).



Some basic examples of CTI operating procedures

Consumer

Observed IP address

1. Query matches in the cache of OSINT feeds
2. Query for matches in internal MISP
3. Document title, date and context (campaign, actor, sector)
4. Document advised action (PAP / CoA)
5. Start enrichment
6. Query ASN, rDNS and pDNS
7. Document ASN, rDNS and pDNS matches
8. Discover and document related IPs and domains

Producer

Encode object in TI platform

1. Check completeness of necessary attributes
2. Define MISP event where to add object
3. Check attributes for false positives (warninglist)
4. Check similarities for existing objects
5. Create object
6. Add the attributes
7. Add expected follow-up actions
8. Add relationships with other objects
9. Document object ID, event ID and outcome of creation

Improve this workflow



How to deal with these CTI operating procedures?

Uniform format

Shareable

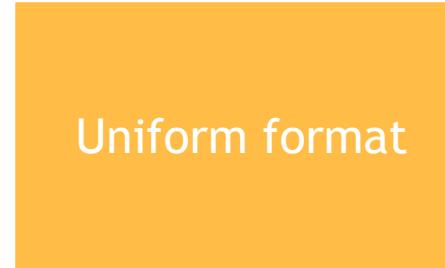
Easy to use

Version tracking

Not just a list of
queries to execute,
must include
documentation

How to deal with these CTI operating procedures?

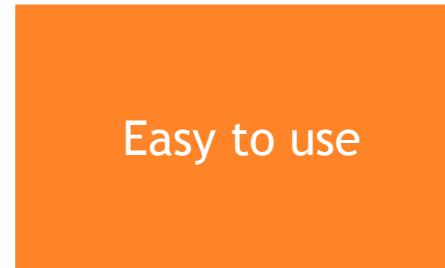
8:48 AM · Jun 10, 2022 · TweetDeck



Uniform format



Shareable



Easy to use



Version tracking

Not just a list of queries to execute, must include documentation

Jupyter notebooks in PyMISP

PyMISP

Python library to access MISP

Automation

Adding and editing data

Integration

...

Jupyter notebooks in PyMISP



Python library to access MISP

Automation

Adding and editing data

Integration

...

A screenshot of a GitHub repository page for "MISP / PyMISP". The repository is public and has 58 issues and 3 pull requests. The "Code" tab is selected. A dropdown menu shows "main". The repository path is "PyMISP / docs / tutorial /". The repository contains several files: "old", "FullOverview.ipynb", "Search-FullOverview.ipynb", and "install_notebook.sh".

MISP / PyMISP Public

Code Issues 58 Pull requests 3 Discussion

main PyMISP / docs / tutorial /

Rafiot chg: Update tutorial for custom objects

..

old

FullOverview.ipynb

Search-FullOverview.ipynb

install_notebook.sh

What are Jupyter notebooks?



Interactive environment	Consumers	Kernel	Distributed
<ul style="list-style-type: none">• Write and execute computer code• Observe the results• Document the code• Text elements• Images	<ul style="list-style-type: none">• Human readable• Executable by machines	<ul style="list-style-type: none">• Computational engine• Executes the machine code	<ul style="list-style-type: none">• Code and documentation are stored in the “execution environment”• But documentation can be edited from anywhere• Web browser

What are Jupyter notebooks?

- **Open source**
 - Used in data science
 - Other areas, such as troubleshooting, and CTI
- Notebooks are stored in a **JSON** format (**.ipynb**)
 - Ideal for code repositories
- **Engines (kernel)**
 - Python, community provided
 - Ruby, C++
- **JupyterLab**
 - “IDE”



The screenshot shows the Jupyter Notebook interface. At the top, there's a header bar with tabs for 'main' (selected), 'nbformat / nbformat / v4 / nbformat.v4.schema.json'. Below it is a user profile section for 'blink1073' with a green checkmark for auto-formatters and a list of 12 contributors. A stats box shows 471 lines (458 sloc) and 15.7 KB. The main area has a code editor with JSON schema code:

```
1 {
2   "$schema": "http://json-schema.org/draft-04/schema#",
3   "description": "Jupyter Notebook v4.5 JSON schema.",
4   "type": "object",
5   "additionalProperties": false,
6   "required": ["metadata", "nbformat_minor", "nbformat", "cells"],
7   "properties": {
```

Below the schema is a terminal window showing the Lorenz system equations:

$$\begin{aligned}\dot{x} &= \sigma(y - x) \\ \dot{y} &= \rho x - y - xz \\ \dot{z} &= -\beta z + xy\end{aligned}$$

It says "Let's change (σ, β, ρ) with ipwidgeon and examine the trajectories." A code cell [2]:

```
from lorenz import solve_lorenz
interactive(solve_lorenz, sigma=(0.0, 50.0), rho=(0.0, 50.0))
```

The output view shows sliders for sigma (10.00), beta (2.67), and rho (28.00). To the right is a code editor window for 'lorenz.py' containing the Lorenz equations and a plot of the Lorenz attractor.

Jupyter notebooks, PyMISP and CTI operational procedures

 **Koen Van Impe** 

A @MISPPProject tip of the week: Document your #CTI operational procedures with Jupyter notebooks and PyMISP. Use the examples at [github.com/cudeso/misp-ti...](https://github.com/cudeso/misp-ti) and [github.com/MISP/PyMISP/tr...](https://github.com/MISP/PyMISP/tree/main/tutorials) to get started. [github.com/cudeso/misp-ti...](https://github.com/cudeso/misp-ti)

Manually add a MISP event and attribute
 The notebook shows you how to manually add a MISP event and one attribute, as with the use of PyMISP.

```
[1]: import pymisp
misp = pymisp.MISPEvent()
misp.info['name'] = "My first event"
misp.info['info'] = "This is my first event"
misp.info['date'] = "2023-04-01T00:00:00Z"
misp.info['timestamp'] = "2023-04-01T00:00:00Z"
misp.info['published'] = False
misp.info['distribution'] = 1
misp.info['sharing'] = 1
misp.info['status'] = 1
misp.info['locked'] = False
misp.info['comment'] = "This is my first event"
misp.info['raw_info'] = "{'name': 'My first event', 'info': 'This is my first event', 'date': '2023-04-01T00:00:00Z', 'timestamp': '2023-04-01T00:00:00Z', 'published': false, 'distribution': 1, 'sharing': 1, 'status': 1, 'locked': false, 'comment': 'This is my first event'}"
```

The configuration of PyMISP is done via the `misp` variable.

The notebook can be used to learn the basics of the MISP server API, or as a documentation step itself. You can have multiple lines per user, allowing for better readability. It's also possible to have multiple lines per command, which is useful for some commands on the command line.

Instead of what you see here you can modify the configuration directly via variables.

Final output what is in fact desired in the demo path.

```
[1]: !curl -X POST https://127.0.0.1:60434/api/events
```

This function PyMISP will display the "Uncaught HTTP500" warning.

```
[2]: from pymisp import PyMISP
misp = PyMISP('https://127.0.0.1:60434', 'admin', 'admin', 'admin')
misp.add_attribute(misp_event_id=1, type='malware-sample', value='malicious_software', comment='Malicious Software Sample', info='Info about the sample', date='2023-04-01T00:00:00Z', timestamp='2023-04-01T00:00:00Z', distribution=1, sharing=1, status=1, published=False)
```

Uncaught HTTP500: Internal Server Error

```
[3]: !curl -X POST https://127.0.0.1:60434/api/events
```

That's it! We have added our first event and attribute to the MISP instance.

Create event in MISP
 Send the request to the server

```
[4]: result = misp.add_event(event_type="malware-sample", value="malicious_software", comment="Malicious Software Sample", info="Info about the sample", date="2023-04-01T00:00:00Z", timestamp="2023-04-01T00:00:00Z", distribution=1, sharing=1, status=1, published=False)
```

Created event ID with MISP ID: 10001-2177-404d-8d6c-000000000001

Add attributes
 Not yet any attributes

```
[5]: for i in range(1, 10):
    result = misp.add_attribute(event_id=1, type="malware-sample", value="malicious_software", comment="Malicious Software Sample", info="Info about the sample", date="2023-04-01T00:00:00Z", timestamp="2023-04-01T00:00:00Z", distribution=1, sharing=1, status=1, published=False)
```

Value sent to the server and the value, and optionally the `is_uef` flag. It's also a good idea to add `comment` to the attribute.

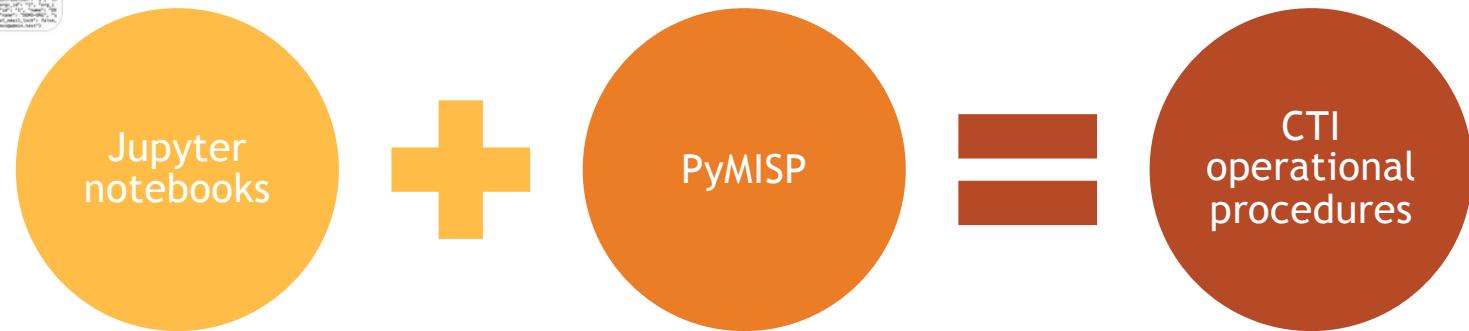
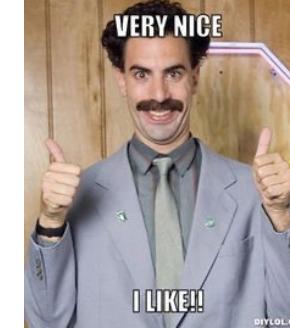
```
[6]: !curl -X POST https://127.0.0.1:60434/api/events
```

Output the attribute event

```
[7]: result = misp.get_event(event_id=1, pyformat=True)
```

```
[8]: print(result)
```

```
[8]: {
  "id": "10001-2177-404d-8d6c-000000000001",
  "info": "{'name': 'My first event', 'info': 'This is my first event', 'date': '2023-04-01T00:00:00Z', 'timestamp': '2023-04-01T00:00:00Z', 'published': false, 'distribution': 1, 'sharing': 1, 'status': 1, 'locked': false, 'comment': 'This is my first event'}",
  "raw_info": "{'name': 'My first event', 'info': 'This is my first event', 'date': '2023-04-01T00:00:00Z', 'timestamp': '2023-04-01T00:00:00Z', 'published': false, 'distribution': 1, 'sharing': 1, 'status': 1, 'locked': false, 'comment': 'This is my first event'}",
  "comment": "This is my first event",
  "date": "2023-04-01T00:00:00Z",
  "timestamp": "2023-04-01T00:00:00Z",
  "distribution": 1,
  "sharing": 1,
  "status": 1,
  "published": false,
  "locked": false,
  "attributes": [
    {
      "id": "10001-2177-404d-8d6c-000000000002",
      "value": "malicious_software",
      "comment": "Malicious Software Sample",
      "info": "Info about the sample",
      "date": "2023-04-01T00:00:00Z",
      "timestamp": "2023-04-01T00:00:00Z",
      "type": "malware-sample",
      "distribution": 1,
      "sharing": 1,
      "status": 1,
      "published": false
    },
    {
      "id": "10001-2177-404d-8d6c-000000000003",
      "value": "malicious_software",
      "comment": "Malicious Software Sample",
      "info": "Info about the sample",
      "date": "2023-04-01T00:00:00Z",
      "timestamp": "2023-04-01T00:00:00Z",
      "type": "malware-sample",
      "distribution": 1,
      "sharing": 1,
      "status": 1,
      "published": false
    },
    {
      "id": "10001-2177-404d-8d6c-000000000004",
      "value": "malicious_software",
      "comment": "Malicious Software Sample",
      "info": "Info about the sample",
      "date": "2023-04-01T00:00:00Z",
      "timestamp": "2023-04-01T00:00:00Z",
      "type": "malware-sample",
      "distribution": 1,
      "sharing": 1,
      "status": 1,
      "published": false
    },
    {
      "id": "10001-2177-404d-8d6c-000000000005",
      "value": "malicious_software",
      "comment": "Malicious Software Sample",
      "info": "Info about the sample",
      "date": "2023-04-01T00:00:00Z",
      "timestamp": "2023-04-01T00:00:00Z",
      "type": "malware-sample",
      "distribution": 1,
      "sharing": 1,
      "status": 1,
      "published": false
    },
    {
      "id": "10001-2177-404d-8d6c-000000000006",
      "value": "malicious_software",
      "comment": "Malicious Software Sample",
      "info": "Info about the sample",
      "date": "2023-04-01T00:00:00Z",
      "timestamp": "2023-04-01T00:00:00Z",
      "type": "malware-sample",
      "distribution": 1,
      "sharing": 1,
      "status": 1,
      "published": false
    },
    {
      "id": "10001-2177-404d-8d6c-000000000007",
      "value": "malicious_software",
      "comment": "Malicious Software Sample",
      "info": "Info about the sample",
      "date": "2023-04-01T00:00:00Z",
      "timestamp": "2023-04-01T00:00:00Z",
      "type": "malware-sample",
      "distribution": 1,
      "sharing": 1,
      "status": 1,
      "published": false
    },
    {
      "id": "10001-2177-404d-8d6c-000000000008",
      "value": "malicious_software",
      "comment": "Malicious Software Sample",
      "info": "Info about the sample",
      "date": "2023-04-01T00:00:00Z",
      "timestamp": "2023-04-01T00:00:00Z",
      "type": "malware-sample",
      "distribution": 1,
      "sharing": 1,
      "status": 1,
      "published": false
    },
    {
      "id": "10001-2177-404d-8d6c-000000000009",
      "value": "malicious_software",
      "comment": "Malicious Software Sample",
      "info": "Info about the sample",
      "date": "2023-04-01T00:00:00Z",
      "timestamp": "2023-04-01T00:00:00Z",
      "type": "malware-sample",
      "distribution": 1,
      "sharing": 1,
      "status": 1,
      "published": false
    },
    {
      "id": "10001-2177-404d-8d6c-000000000010",
      "value": "malicious_software",
      "comment": "Malicious Software Sample",
      "info": "Info about the sample",
      "date": "2023-04-01T00:00:00Z",
      "timestamp": "2023-04-01T00:00:00Z",
      "type": "malware-sample",
      "distribution": 1,
      "sharing": 1,
      "status": 1,
      "published": false
    }
  ]
}
```



Jupyter notebooks, PyMISP and CTI operational procedures



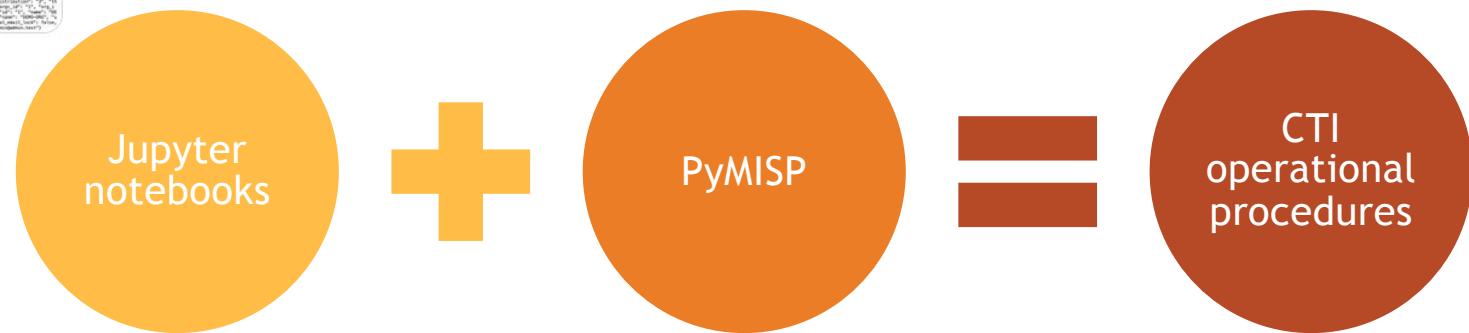
A @MISPProject tip of the week: Document your #CTI operational procedures with Jupyter notebooks and PyMISP. Use the examples at github.com/cedeso/misp-ti... and github.com/MISP/PyMISP/tr... to get started. github.com/cedeso/misp-ti...

8:48 AM · Jun 10, 2022 · TweetDeck

“Remote code execution“ on a MISP server



Run the notebook on a stable, dedicated system (or user environment) with access to MISP



Lesson 1: Run notebooks from dedicated/well-maintained environments

How to get started?

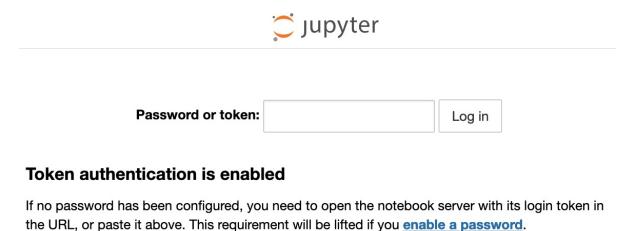


```
mkdir cti-operational-procedure  
cd cti-operational-procedure  
virtualenv venv  
source venv/bin/activate  
pip install notebook  
pip install pymisp  
jupyter notebook --no-browser --ip misp.demo.cudeso.be --port 8888
```

Access the Jupyter notebook server

```
(venv) koenv@misp-demo:~/cti-operational-procedure/notebooks$ jupyter notebook --no-browser --ip misp.demo.cudeso.be --port 8888
[I 14:46:11.877 NotebookApp] Serving notebooks from local directory: /home/koenv/cti-operational-procedure/notebooks
[I 14:46:11.877 NotebookApp] Jupyter Notebook 6.5.1 is running at:
[I 14:46:11.878 NotebookApp] http://misp.demo.cudeso.be:8888/?token=f8840d3b2db[REDACTED]
[I 14:46:11.878 NotebookApp] or http://127.0.0.1:8888/?token=f8840d3b2dbbc2170
[I 14:46:11.878 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```

- When only running on localhost (or firewalled)
 - Login with SSH
 - Setup port forwarding
- Access the URL
 - Token gets set as a cookie in your browser
 - If you forget to include the token you have to enter it in a web form



A note on the Jupyter notebook server

- Notebooks are served from the **directory where the server is executed**
 - Path
 - Environment conditions
 - MISP
 - Custom venv

```
git pull procedure-repo  
git add .  
git commit -m "Update to Procedure 1"  
git push
```

Lesson 2: Pay attention to the location from where you start the notebook server

- The notebook server is meant for **single-user use**
 - Multiple users accessing the interface will clutter the results
 - Need multiple users? Have a look at JupyterHub
 - Multi-user hub that spawns multiple instances of single-user Jupyter notebook server

Notebook interface

The screenshot shows a Jupyter Notebook interface with several annotations:

- Filename:** A red callout points to the title bar which displays "jupyter misp_add_event_and_attribute (autosaved)".
- Block type (code/doc):** A red callout points to the toolbar above the notebook area, specifically the "Markdown" button.
- Add a new block:** A red callout points to the "Cell" menu item in the top navigation bar.
- control execution:** A blue vertical bar with this text is positioned on the left side of the notebook area, indicating the execution context.

The notebook content includes:

manually add a MISP event and attribute

This notebook shows you how to manually add a MISP event and one attribute, all with the use of PyMISP.

Configure PyMISP

The configuration of PyMISP is done via the `keys.py` file.

This file needs to contain at least the **URL** of the MISP server (`misp_url`) and an **authentication key** (`misp_key`). You can have multiple keys per user, stored in separate files. Make sure you restrict access to the keys as they give exactly the same permissions as the associated user.

Instead of a keys file you can also supply the configuration directly via variables.

First output what is in our keys file stored in the demo path.

```
In [ ]: cat ~/demo/PyMISP/keys.py
```

Then initialise PyMISP and disable the "Unverified HTTPS request" warnings.

Basic skeleton script

Import
libraries

Create a
PyMISP
object

```
import urllib3
from pymisp import PyMISP, MISPEvent
from keys import misp_url, misp_key, misp_verifycert

if misp_verifycert is False:
    import urllib3
    urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

misp = PyMISP(misp_url, misp_key, misp_verifycert)

print("I will use the server {}".format(misp_url))
```

- Where is the API key? The MISP URL?
 - “keys.py”

Basic skeleton script

Import
libraries

Create a
PyMISP
object

Specific path

```
import urllib3
from pymisp import PyMISP, MISPEvent
import sys
sys.path.insert(0, "/home/koenv/cti-operational-procedure/vault/")
from keys import misp_url, misp_key, misp_verifycert

if misp_verifycert is False:
    import urllib3
    urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

misp = PyMISP(misp_url, misp_key, misp_verifycert)

print("I will use the server {}".format(misp_url))
```

- Where is the API key? The MISP URL?
 - “keys.py”

Lesson 3: Do not store credentials in a notebook!

Example: Create a MISP event

Document threat behaviour in MISP

This procedure walks you through the steps of creating a new event in MISP.



Trigger

This procedure is triggered when a new threat behaviour is observed during incident response.

Configure PyMISP

```
In [1]: import urllib3
from pymisp import PyMISP, MISPEvent
import sys
sys.path.insert(0, "/home/koenv/cti-operational-procedure/vault/")
from keys import misp_url, misp_key, misp_verifycert

if misp_verifycert is False:
    import urllib3
    urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

misp = PyMISP(misp_url, misp_key, misp_verifycert)
print("I will use the server {}".format(misp_url))
```

I will use the server <https://misp.demo.cudeso.be/>

Debug output.
If creation of pymisp object fails we will not get to this debug message

Lesson 4: Test the connection at the start of the procedure.

Example: Create a MISP event

Document threat behaviour in MISP

This procedure walks you through the steps of creating a new event in MISP.



Trigger

This procedure is triggered when a new threat behaviour is observed during incident response.

Configure PyMISP

```
In [1]: import urllib3
from pymisp import PyMISP, MISPEvent
import sys
sys.path.insert(0, "/home/koenv/cti-operational-procedure/vault/")
from keys import misp_url, misp_key, misp_verifycert

if misp_verifycert is False:
    import urllib3
    urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

misp = PyMISP(misp_url, misp_key, misp_verifycert)

print("I will use the server {}".format(misp_url))

I will use the server https://misp.demo.cudeso.be/
```

Create a MISP event

Analyst: add basic event elements

Set the event title, distribution, threat level and analysis.

Distribution

- 0: Your organization only
- 1: This community-only
- 2: Connected communities
- 3: All communities

Threat level

- 1: Low (mass malware)
- 2: Medium (APT)
- 3: High (0-day)
- 4: Undefined

Analysis

- 0: Initial
- 1: Ongoing
- 2: Completed

One code block with variables
One code block with code to execute

```
In [2]: new_event_title = "CTIS-2022 Threat alert"
new_event_distribution = 3
new_event_threat_level = 4
new_event_analysis = 1
```

```
In [3]: event = MISPEvent()
event.info = new_event_title
event.distribution = new_event_distribution
event.threat_level_id = new_event_threat_level
event.analysis = new_event_analysis
```

Lesson 5: Avoid that analysts have to fiddle with ‘raw code.

Example: Create a MISP event

Create a MISP event

Analyst: add basic event elements

Set the event title, distribution, threat level and analysis.

Distribution

- 0: Your organization only
- 1: This community-only
- 2: Connected communities
- 3: All communities

Threat level

- 1: Low (mass malware)
- 2: Medium (APT)
- 3: High (0-day)
- 4: Undefined

Analysis

- 0: Initial
- 1: Ongoing
- 2: Completed

```
In [2]: new_event_title = "CTIS-2022 Threat alert"
new_event_distribution = 3
new_event_threat_level = 4
new_event_analysis = 1
```

```
In [3]: event = MISPEvent()
event.info = new_event_title
event.distribution = new_event_distribution
event.threat_level_id = new_event_threat_level
event.analysis = new_event_analysis
```

Analyst: add a date

When did you discover / observed this threat?

```
In [4]: new_event_date = "2022-10-15"
```

```
In [5]: event.set_date(new_event_date)
```

Analyst: add the TLP level

By default we use TLP:AMBER. Refer to [DOC](#) for guidance on choosing the correct TLP level.

```
In [6]: event.add_tag("tlp:amber")
```

```
Out[6]: <MISPTag(name=tlp:amber)>
```

Create event in MISP

Send the request to the server.

```
In [7]: result = misp.add_event(event, pythonify=True)
print("Created event ID {} for {}".format(result.id, result.uuid, new_event_title))
```

```
Created event ID 736 for d7da86fa-a549-4c4c-93c6-b2097433507d
```

Execution result.

Not just the “Python” success/failure, but the execution result of a step in the procedure.
Can also be used for reporting.

Lesson 6: Print execution results of important steps.

Example: Create a MISP event

Analyst: add a date

When did you discover / observed this threat?

```
In [4]: new_event_date = "2022-10-15"
```

```
In [5]: event.set_date(new_event_date)
```

Analyst: add the TLP level

By default we use TLP:AMBER. Refer to [DOC](#) for guidance on choosing the correct TLP level.

```
In [6]: event.add_tag("tlp:amber")
```

```
Out[6]: <MISPTag(name=tlp:amber)>
```

Create event in MISP

Send the request to the server.

```
In [7]: result = misp.add_event(event, pythonify=True)
print("Created event ID {} for {}".format(result.id, result.uuid, new_event_title))
```

Created event ID 736 for d7da86fa-a549-4c4c-93c6-b2097433507d

Print a summary at the end of execution

Analyst: add attributes

Make sure you set the **type** and the **value**, and the **to_ids** flag. Set **to_ids** to True if the IP address needs to be blocked.

Add contextualisation to the attribute.

By default we use PAP:AMBER for the Permissible Action Protocol and set the expected Courses of Action to Deny.

```
In [8]: new_attribute = {
    "type": "ip-dst",
    "value": "8.8.4.4",
    "to_ids": False,
    "tag": ["PAP:AMBER", "course-of-action:active='deny'"],
    "comment": "Initial connectivity check"
}
```

```
In [17]: from pymisp import MISPAtribute

attribute = MISPAtribute()
attribute.type = new_attribute["type"]
attribute.value = new_attribute["value"]
attribute.to_ids = new_attribute["to_ids"]
for t in new_attribute["tag"]:
    attribute.add_tag(t)
attribute.comment = new_attribute["comment"]
```

Add to event

And now add the attribute to the event

```
In [18]: result_attr = misp.add_attribute(result.id, attribute, pythonify=True)
print("Added attribute {}".format(result_attr.uuid))
```

Added attribute 89c03549-0f76-40e5-85db-dd6a904a276a

Summary

```
In [19]: print("The event {} ( {}) was created to deal with the threat.".format(result.info, result.uuid))
print("The defined follow-up actions for {} are {}, in object {}".format(new_attribute["value"], new_attribute["tag"]))
```

The event CTIS-2022 Threat alert (d7da86fa-a549-4c4c-93c6-b2097433507d) was created to deal with the threat.
The defined follow-up actions for 8.8.4.4 are ['PAP:AMBER', 'course-of-action:active="deny"'], in object 89c03549-0f76-40e5-85db-dd6a904a276a

Lesson 7: Conclude procedures with a summary of what was done.

Example: Create a MISP object

Create a MISP file object

Add a MISP file object to a specific threat event. As an analyst set the file attributes and the event ID.

Init MISP

```
In [1]: import urllib3
from pymisp import PyMISP, MISPEvent, MISPOobject
import sys
sys.path.insert(0, "/home/koenv/cti-operational-procedure/vault/")
from keys import *

if misp_verifycert is False:
    import urllib3
    urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

misp = PyMISP(misp_url, misp_key, misp_verifycert)

print("I will use the server {}".format(misp_url))
```

I will use the server <https://misp.demo.cudeso.be/>

Analyst: set file attributes

```
In [2]: new_file = {
    'filename': "MyFileName.docx",
    'size-in-bytes': '2000',
    'md5': '6586f96163923187aaaf4aaa233345162',
}
```

Analyst: set event ID

```
In [3]: event_id = 674
```

Create the object

```
In [4]: file_object = MISPOobject('file')
file_object.add_attribute('filename', new_file.get('filename'))
file_object.add_attribute('md5', new_file.get('md5'))
file_object.add_attribute('size_in_bytes', new_file.get('size-in-bytes'))
res = misp.add_object(event_id, file_object)
print(res)
```

```
{'Object': {'id': '88447', 'name': 'file', 'meta-category': 'file', 'description': 'File object describing a file with meta-information', 'template_uuid': '688c46fb-5edb-40a3-8273-1af7923e2215', 'template_version': '24', 'event_id': '674', 'uuid': 'a4c3d112-52bc-4411-8468-0a924d979996', 'timestamp': '1665862920', 'distribution': '5', 'group_id': '0', 'comment': '', 'deleted': False, 'first_seen': None, 'last_seen': None, 'Attribute': [{"id": "88447", "value": "MyFileName.docx", "type": "string", "category": "file", "comment": ""}, {"id": "88447", "value": "6586f96163923187aaaf4aaa233345162", "type": "string", "category": "md5", "comment": ""}, {"id": "88447", "value": "2000", "type": "string", "category": "size_in_bytes", "comment": ""}]}
```

Example: Lookup MISP feed cache and PDNS results

Procedure to handle IP address

Init MISP

```
In [1]: import urllib3
from pymisp import PyMISP, MISPEvent
import sys
sys.path.insert(0, "/home/koenv/cti-operational-procedure/vault/")
from keys import *

if misp_verifycert is False:
    import urllib3
    urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

misp = PyMISP(misp_url, misp_key, misp_verifycert)

import requests

misp_headers = {
    "Authorization": misp_key,
    "Content-Type": "application/json",
    "Accept": "application/json"
}

import tabulate

print("I will use the server {}".format(misp_url))

I will use the server https://misp.demo.cudeyo.be/
```

Analyst: set IP address

```
In [2]: investigate_ip = "200.159.87.196"
```

Lookup in the cache

Via 'requests'library, not via PyMISP

```
In [3]: misp_cache_url = "{}/feeds/searchCaches/value:{}".format(misp_url, investigate_ip)
cache_results = requests.get(misp_cache_url, headers=misp_headers, verify=misp_verifycert)
direct_urls = []
for el in cache_results.json():
    direct_urls.append(el["Feed"]["direct_urls"])
```

Fetch the cached events

```
In [4]: cache_table = []
for el in direct_urls[0]:
    misp_preview_cache_url = el["url"]
    cached_event = requests.get(misp_preview_cache_url, headers=misp_headers, verify=misp_verifycert)

    cached_event_title = cached_event.json()["Event"]["info"]
    cached_event_data = cached_event.json()["Event"]["date"]
    cache_table.append([cached_event_title, cached_event_data])
```

```
In [5]: print(tabulate.tabulate(cache_table, headers=["MISP Event title", "Date"]))
```

MISP Event title	Date
Scraper: DeftTorero: tactics, techniques and procedures of intrusions revealed	2022-10-14

```
In [6]: import pypdns
x = pypdns.PyPDNS(basic_auth=(pdns_user,pdns_password))
pdns_results = x.query(investigate_ip)
pdns_results_table = []
if len(pdns_results) > 0:
    for el in pdns_results:
        pdns_results_table.append([el["rdata"], el["rrname"], el["rrtype"]])

print(tabulate.tabulate(pdns_results_table, headers=["Data", "Name", "Type"]))
```

Data	Name	Type
200.159.87.196	c.ns.abys.com.br	A
200.159.87.196	a.ns.oscarcalcados.com.br	A

Example: Building integrations with a 3rd party

Task: integrate a new data provider in MISP

Learn 3rd party API functionality

A notebook to document your progress

Implement code snippets from provider

Link together and convert to MISP API queries

Code is stored together with findings

Demonstrate a PoC to colleagues

Show alternatives without multiple versions of a PoC

Your API usage complements documentation 3rd party

Some examples in ‘MISP tip of the week’

The screenshot shows a GitHub repository page for 'cudeyo / misp-tip-of-the-week'. The repository is public and has 15 issues and 1 pull request. The 'Code' tab is selected. The main branch is 'main'. The repository contains several files:

- cudeyo Add notebook examples for CTIS
- ..
- MISP Sharing Groups noaudio.mov Tip 20220520
- MISP-background-jobs-tips.md Tip 20220722
- misp-private-taxonomy_machinetag.json Tip 20220923
- misp_add_correlation_exclusion.ipynb Jupyter notebook for MISP correlation exclusions
- misp_add_event_and_attribute.ipynb Tip 20220610
- misp_create_object.ipynb Add notebook examples for CTIS
- misp_modules.ipynb Create misp_modules.ipynb
- misp_procedure_ip.ipynb Add notebook examples for CTIS
- skeleton-procedure.ipynb Add notebook examples for CTIS

Resource monitoring

The Jupyter notebook executes code as a Python process

- Open a second console and monitor load for Python processes

Large output can delay the response

- Whenever possible, limit the output

Lesson 8: Monitor system resources when executing code.

Multiple versions of a code cell

The screenshot shows a Jupyter Notebook interface with two code cells and a 'raw' block.

Code Cell 1: A code cell containing Python code. It imports `urllib3`, `PyMISP`, and `MISPEvent` from `pymisp`. It checks if `misp_verifycert` is `False` and disables urllib3's warning for insecure requests. It creates a `PyMISP` object and prints a message indicating the server being used.

```
In [1]: import urllib3
from pymisp import PyMISP, MISPEvent
from keys import misp_url, misp_key, misp_verifycert

if misp_verifycert is False:
    import urllib3
    urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

misp = PyMISP(misp_url, misp_key, misp_verifycert)

print("I will use the server {}".format(misp_url))
```

Code Cell 2: A code cell containing Python code. It imports `urllib3`, `PyMISP`, `MISPEvent`, `sys`, and `keys`. It inserts a path into `sys.path` and imports `misp_url`, `misp_key`, and `misp_verifycert` from `keys`.

```
In [2]: import urllib3
from pymisp import PyMISP, MISPEvent
import sys
sys.path.insert(0, "/home/koenv/cti-operational-procedure/vault/")
from keys import misp_url, misp_key, misp_verifycert
```

Raw NBConvert Block: A 'raw' block containing the same Python code as Code Cell 1, intended for operational procedures with Jupyter notebooks and PyMISP.

```
import urllib3
from pymisp import PyMISP, MISPEvent
from keys import misp_url, misp_key, misp_verifycert

if misp_verifycert is False:
    import urllib3
    urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

misp = PyMISP(misp_url, misp_key, misp_verifycert)

print("I will use the server {}".format(misp_url))
```

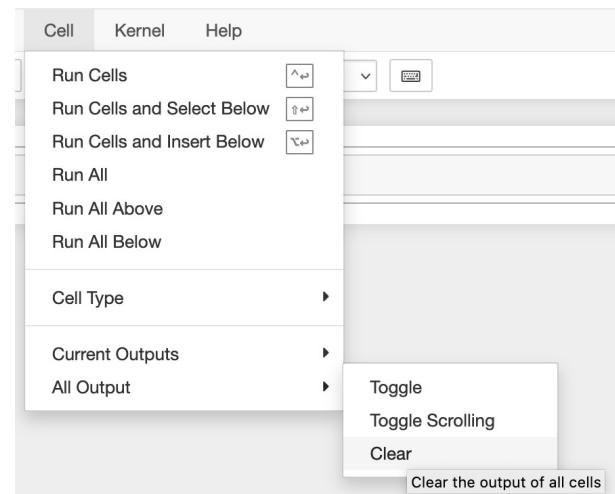
Lesson 9: Disable a code block with ESC R to test different versions of your code.

Output of code execution is stored in the notebook

Desired if you create a report

Not-desired if you create the base procedure

Not-desired if you share the script



```
(venv) koen@misp-demo:~/cti-operational-procedure/notebooks$ cat skeleton-2.ipynb | jq '.[] | .[] | .outputs[]'
```

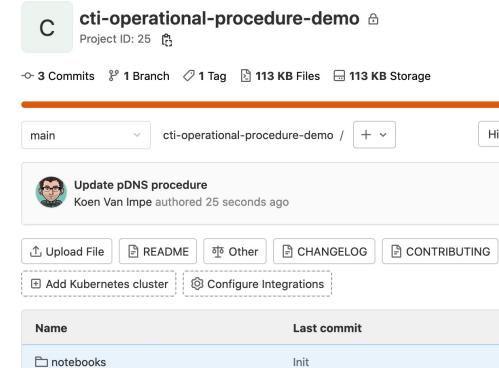
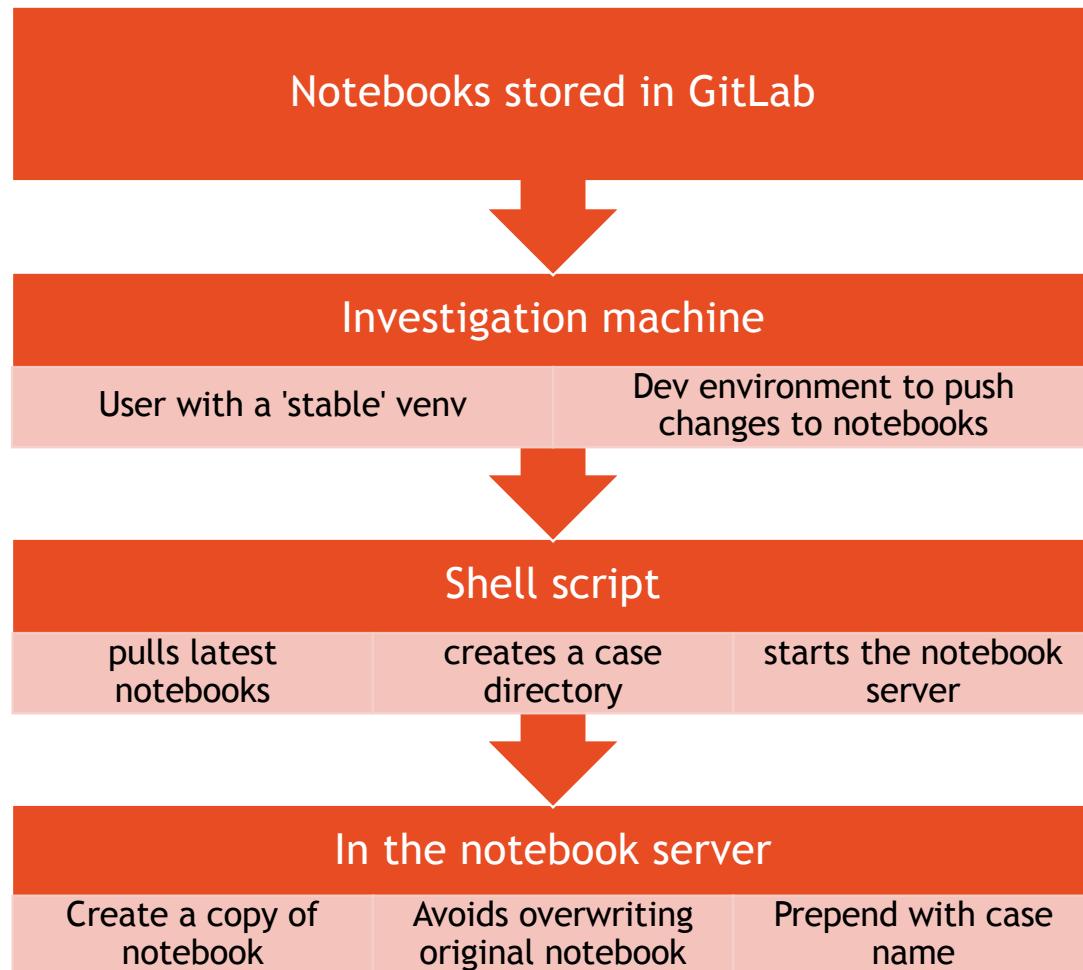
```
null
null
[
  {
    "name": "stdout",
    "output_type": "stream",
    "text": [
      "I will use the server https://misp.demo.cudeso.be/\n"
    ]
  }
]
```

Add a custom shortcut under Help>Edit Keyboard Shortcuts

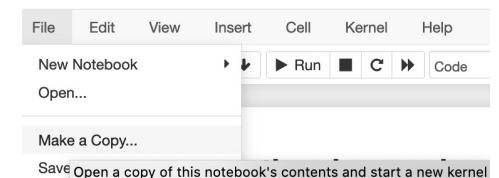
toggle all cells output scrolled	add shortcut
clear all cells output	Shift-o
save notebook	S
find and replace	F
paste dialog	V

Lesson 10: Clean the output of notebooks when sharing them.

Workflow for notebooks



```
git pull  
cp -r notebooks $1  
cd $1  
jupyter notebook --no-browser --ip misp.demo.cudeso.be --port 8888
```



Moving forward

- Still discovering the possibilities of using Jupyter notebooks for operational procedures.
- Something to work on in the future ...

Lesson 10,5: It never stops ...

Moving forward

Shared repository of (PyMISP) notebooks

- Contribute to the existing repository?
- Assist communities that want to start writing their own procedures
- Help organisations who start with CTI

main ▾

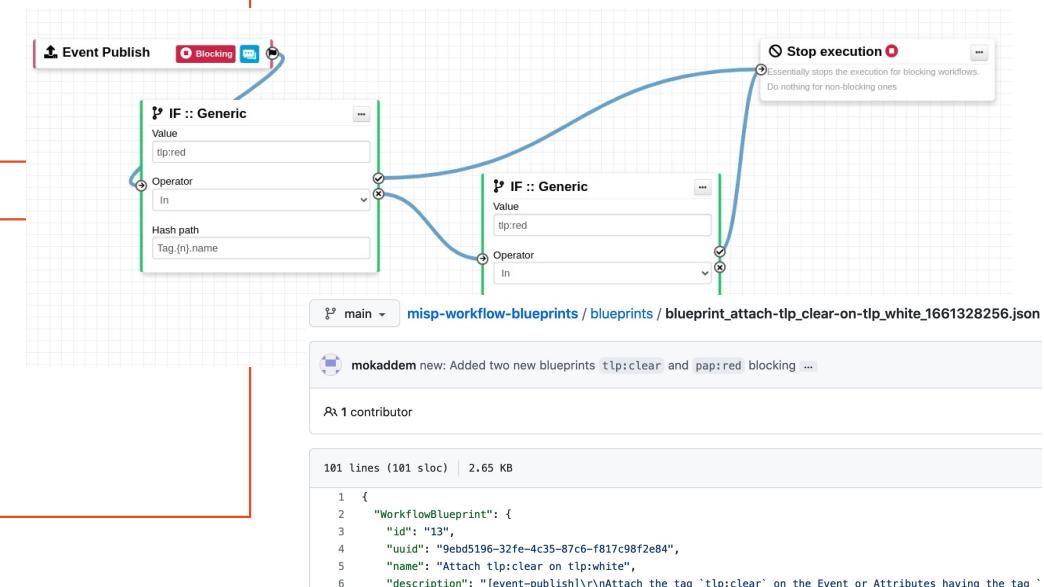
[PyMISP / docs / tutorial /](#)

Develop good practices for documentation

- “Templates” on how to encode (threat) information
- Always add context and completeness
- Include expected actions
- ...

MISP workflows

- Integrate notebooks with the MISP workflows?
 - JSON
- Trigger workflow actions from the notebook?
- Process response from actions in the notebook?
- Document execution path / workflow in a notebook



Questions?