MISP playbooks

2023 FIRST Automation SIG

cudeso.be
We Secure You
https://www.cudeso.be
koen.vanimpe@cudeso.be

TLP:white/clear

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

Operational Procedures

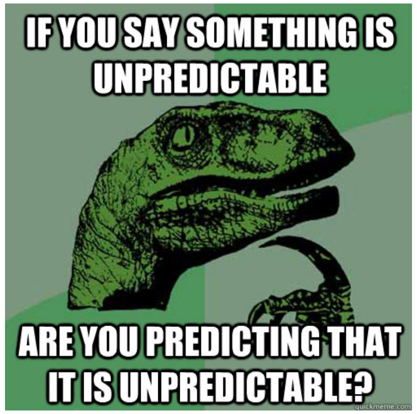
Operational procedures

Operational procedures?

Playbooks

Standard Operating Procedures (SOP)

Workflows


IF YOU SAY SOMETHING IS UNPREDICTABLE
ARE YOU PREDICTING THAT IT IS UNPREDICTABLE?

Consistent approach

Recipe for an investigation

Repeatable

Predictable

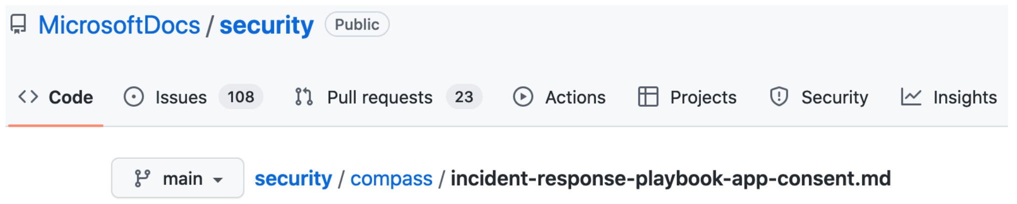
Completeness checks

Documented actions

Leads up to automation

Formats of operational procedures, workflows or playbooks

- **Markdown**
  - Stored in a wiki, GitLab, GitHub


MicrosoftDocs / security Public
Code
Issues 108
Pull requests 23
Actions
Projects
Security
Insights
main
security / compass / incident-response-playbook-app-consent.md

- **JSON**
  - Collaborative Automated Course of Action Operations (CACAO) **Security Playbooks**. Exists also as a MISP object.
    - Workflow for security orchestration
  - TheHive case templates

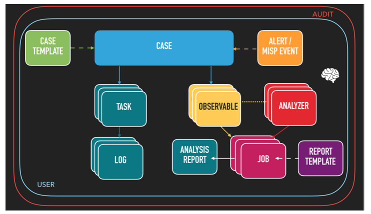
OASIS OPEN

CACAO Security Playbooks Version 1.1

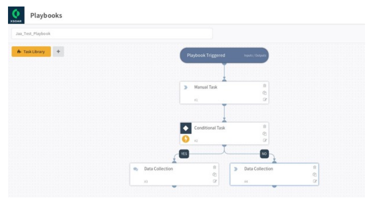
security-playbook

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

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.

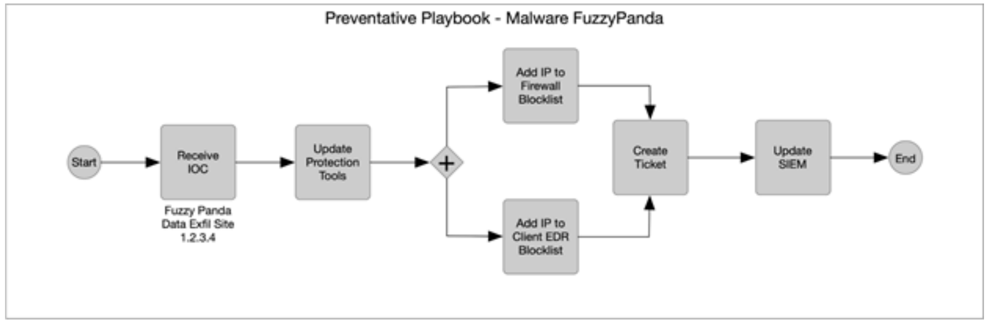


- **COPS**
  - Collaborative Open Playbook Standard
  - Schema based on YAML

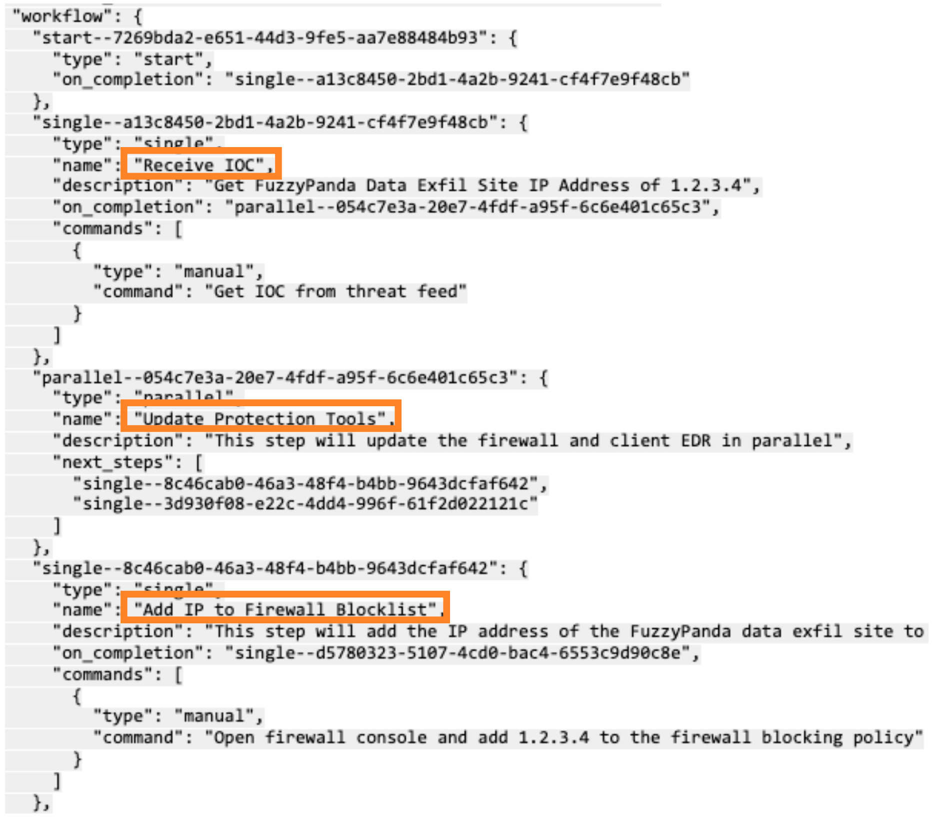


Documentation, execution and report

- **Markdown, JSON, COPS, …**
  - Excellent solutions, primarily focused on security monitoring, orchestration and response



Preventative Playbook - Malware FuzzyPanda
Start
Receive IOC
Fuzzy Panda Data Exfil Site 1.2.3.4
Update Protection Tools
Add IP to Firewall Blocklist
Add IP to Client EDR Blocklist
Create Ticket
Update SIEM
End

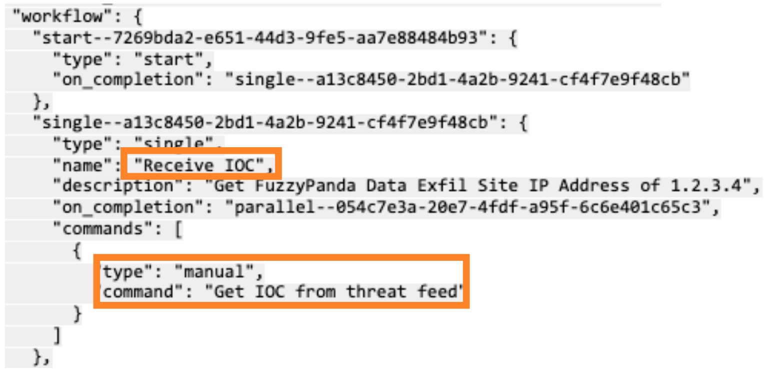


```
"workflow": {
  "start--7269bda2-e651-44d3-9fe5-aa7e88484b93": {
    "type": "start",
    "on_completion": "single--a13c8450-2bd1-4a2b-9241-cf4f7e9f48cb"
  },
  "single--a13c8450-2bd1-4a2b-9241-cf4f7e9f48cb": {
    "type": "single",
    "name": "Receive IOC",
    "description": "Get FuzzyPanda Data Exfil Site IP Address of 1.2.3.4",
    "on_completion": "parallel--054c7e3a-20e7-4fdf-a95f-6c6e401c65c3",
    "commands": [
      {
        "type": "manual",
        "command": "Get IOC from threat feed"
      }
    ]
  },
  "parallel--054c7e3a-20e7-4fdf-a95f-6c6e401c65c3": {
    "type": "parallel",
    "name": "Update Protection Tools",
    "description": "This step will update the firewall and client EDR in parallel",
    "next_steps": [
      "single--8c46cab0-46a3-48f4-b4bb-9643dcfaf642",
      "single--3d930f08-e22c-4dd4-996f-61f2d022121c"
    ]
  },
  "single--8c46cab0-46a3-48f4-b4bb-9643dcfaf642": {
    "type": "single",
    "name": "Add IP to Firewall Blocklist",
    "description": "This step will add the IP address of the FuzzyPanda data exfil site to
    "on_completion": "single--d5780323-5107-4cd0-bac4-6553c9d90c8e",
    "commands": [
      {
        "type": "manual",
        "command": "Open firewall console and add 1.2.3.4 to the firewall blocking policy"
      }
    ]
  },
```

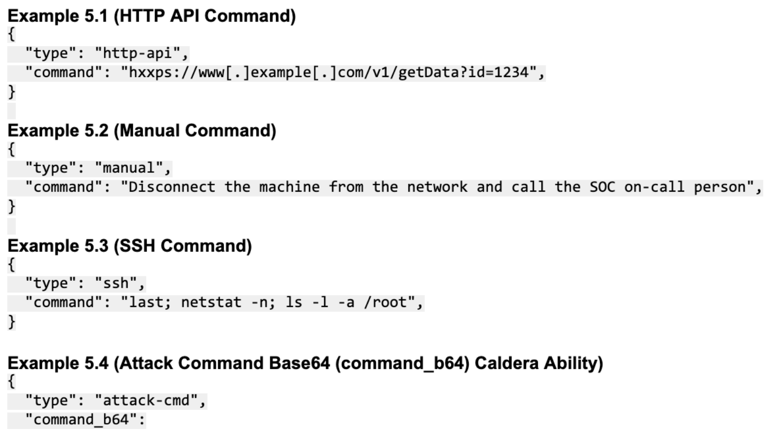
Documentation, execution and report

- **Markdown, JSON, COPS, …**
  - Excellent solutions, primarily focused on security monitoring, orchestration and response

- Notion of "**what**" (and a bit of "how")
  - Commands to execute



```
"workflow": {
  "start--7269bda2-e651-44d3-9fe5-aa7e88484b93": {
    "type": "start",
    "on_completion": "single--a13c8450-2bd1-4a2b-9241-cf4f7e9f48cb"
  },
  "single--a13c8450-2bd1-4a2b-9241-cf4f7e9f48cb": {
    "type": "single",
    "name": "Receive IOC",
    "description": "Get FuzzyPanda Data Exfil Site IP Address of 1.2.3.4",
    "on_completion": "parallel--054c7e3a-20e7-4fdf-a95f-6c6e401c65c3",
    "commands": [
      {
        type": "manual",
        command": "Get IOC from threat feed'
      }
    ]
  },
```



Example 5.1 (HTTP API Command)

```
{
  "type": "http-api",
  "command": "hxxps://www[.]example[.]com/v1/getData?id=1234",
}
```

Example 5.2 (Manual Command)

```
{
  "type": "manual",
  "command": "Disconnect the machine from the network and call the SOC on-call person",
}
```

Example 5.3 (SSH Command)

```
{
  "type": "ssh",
  "command": "last; netstat -n; ls -l -a /root",
}
```

Example 5.4 (Attack Command Base64 (command_b64) Caldera Ability)

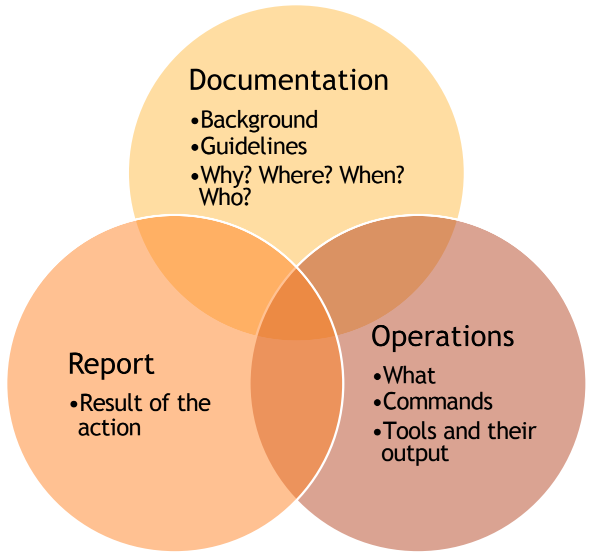
```
{
  "type": "attack-cmd",
  "command_b64":
```

Documentation, execution and report

- **Markdown, JSON, COPS, …**
  - Excellent solutions, primarily focused on security monitoring, orchestration and response

- Notion of "**what**" (and a bit of "how")
  - Commands to execute

- But a disconnect between **documentation** ("why"), the conditions on how to **execute** the action, and where to **report** the result of that action



CTI Operational Procedures

Common use case of CTI operational procedures

Consumer

Observed domain during IR

1. **Query** OSINT feeds and threat events internal MISP
2. **Document** title, date and context (campaign, actor, sector) of events where domain is found
3. **Document** advised follow-up action (PAP / CoA) based on info threat events.
4. **Query** DNS, VirusTotal, URLscan for enrichment
5. **Document** DNS, VirusTotal, URLscan matches
6. Discover and **document** related IPs and domains

Producer

Encode object in MISP

1. Use the object definition to **document** the required attributes for an object
2. **Document** the attributes that you have and search for similarities in existing objects to avoid doubles
3. **Create** the object, add attributes and ensure that attributes have comments and tags for context
4. Add follow-up actions (PAP/CoA) based on **documentation** guidelines
5. Add the relationships with other objects in the threat event
6. Add the object reference and context in a threat **report**

How to use, and re-use, these CTI operational procedures?

Documentation

(why, where, when, who)

Commands and tools

(what)

Open format

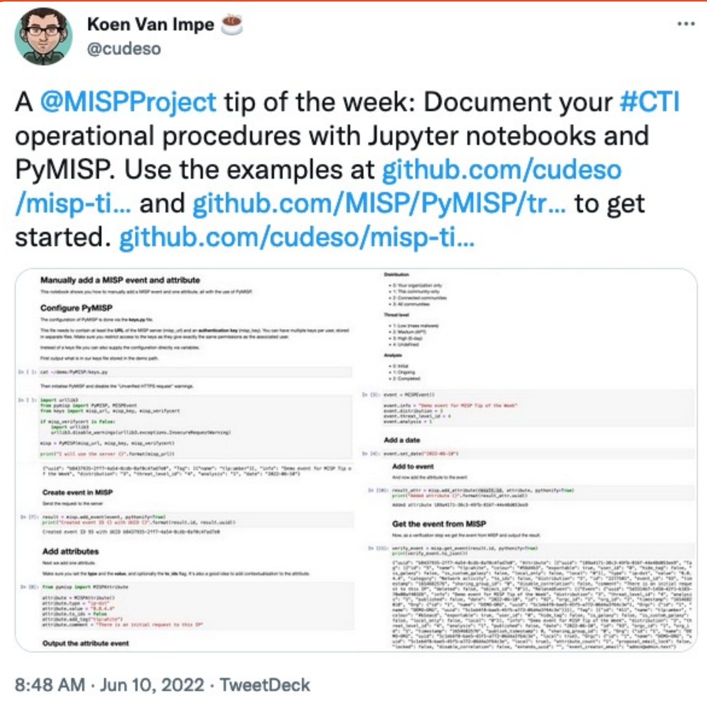
Shareable

Easy to use

Version tracking

Not a 100+ page document that sits in a corner and that no-one reads

How to use, and re-use, these CTI operational procedures?


Koen Van Impe ☕
@cudeso

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

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.

```
In [ ]: 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))
```

```
[illegible]
```

Create event in MISP

Send the request to the server

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

        Created event ID 93 with UUID [illegible]
```

Add attributes

Next we add one attribute.

Make sure you set the **type** and the **value**, and optionally the **to_ids** flag. It's also a good idea to add contextualisation to the attribute.

```
In [8]: from pymisp import MISPAttribute

        attribute = MISPAttribute()
        attribute.type = "ip-dst"
        attribute.value = "8.8.4.4"
        attribute.to_ids = False
        attribute.add_tag("tlp:white")
        attribute.comment = "There is an initial request to this IP"
```

Output the attribute event

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 [3]: event = MISPEvent()

        event.info = "Demo event for MISP Tip of the Week"
        event.distribution = 3
        event.threat_level_id = 4
        event.analysis = 1
```

Add a date

```
In [4]: event.set_date("2022-06-10")
```

Add to event

And now add the attribute to the event

```
In [10]: result_attr = misp.add_attribute(result.id, attribute, pythonify=True)
         print("Added attribute {}".format(result_attr.uuid))

         Added attribute [illegible]
```

Get the event from MISP

Now, as a verification step we get the event from MISP and output the result.

```
In [11]: verify_event = misp.get_event(result.id, pythonify=True)
         print(verify_event.to_json())

         [illegible]
```

8:48 AM · Jun 10, 2022 · TweetDeck

Documentation

(why, where, when, who)

Commands and tools

(what)

Open format

Shareable

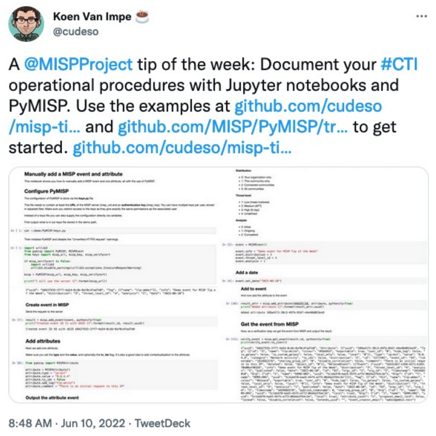
Easy to use

Version tracking

Not a 100+ page document that sits in a corner and that no-one reads

How it started …

$$\begin{tikzcd}
\text{Tweet} \arrow[r] & \text{Proposal} \arrow[r] & \text{Talk} \arrow[r] & \text{Project}
\end{tikzcd}$$

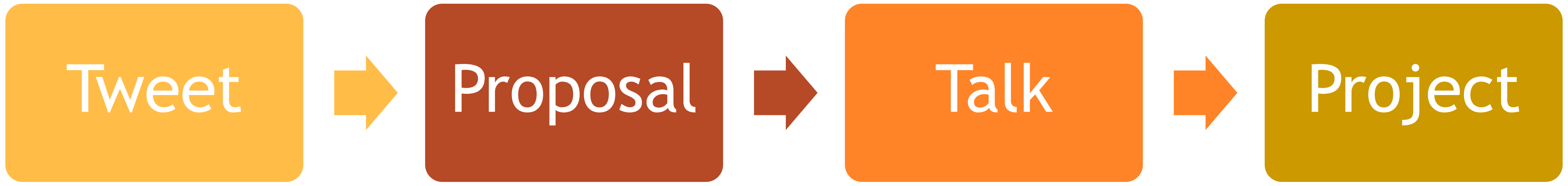


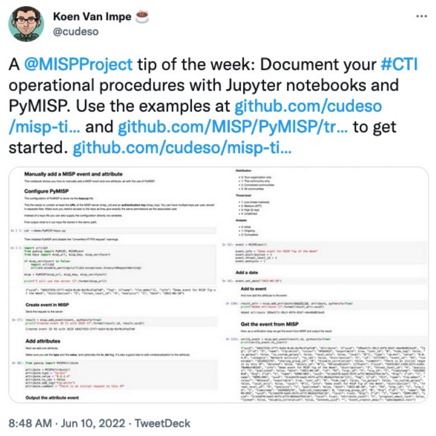
Koen Van Impe ☕
@cudeso

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

8:48 AM · Jun 10, 2022 · TweetDeck

How it started ...

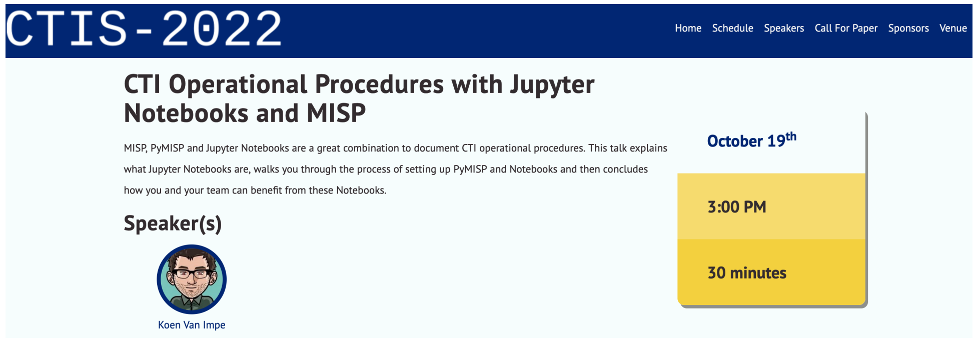




Koen Van Impe ☕
@cudeso

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

8:48 AM · Jun 10, 2022 · TweetDeck



CTI Operational Procedures with Jupyter Notebooks and MISP

MISP, PyMISP and Jupyter Notebooks are a great combination to document CTI operational procedures. This talk explains what Jupyter Notebooks are, walks you through the process of setting up PyMISP and Notebooks and then concludes how you and your team can benefit from these Notebooks.

Speaker(s)

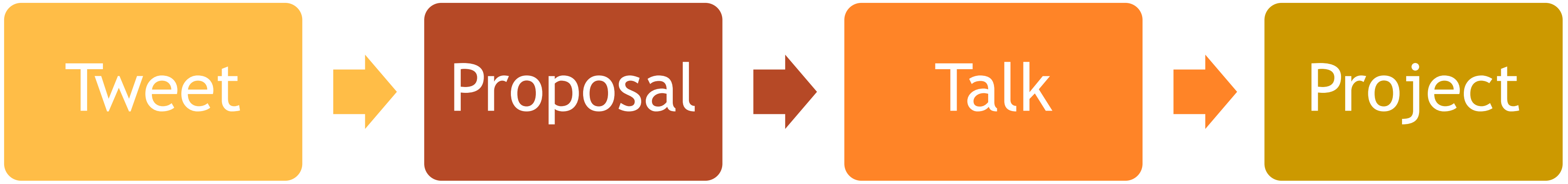
Koen Van Impe

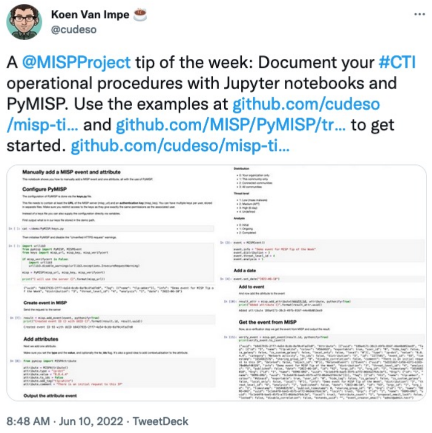
October 19th

3:00 PM

30 minutes

How it started …

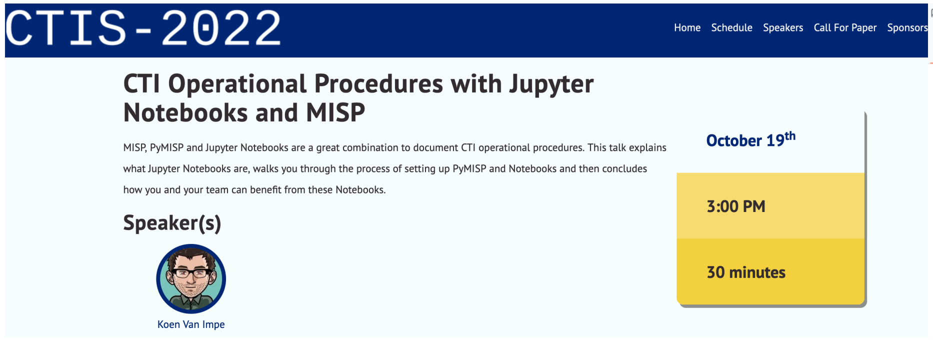




Koen Van Impe ☕
@cudeso

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

8:48 AM · Jun 10, 2022 · TweetDeck



CTIS-2022

Home Schedule Speakers Call For Paper Sponsors

CTI Operational Procedures with Jupyter Notebooks and MISP

MISP, PyMISP and Jupyter Notebooks are a great combination to document CTI operational procedures. This talk explains what Jupyter Notebooks are, walks you through the process of setting up PyMISP and Notebooks and then concludes how you and your team can benefit from these Notebooks.

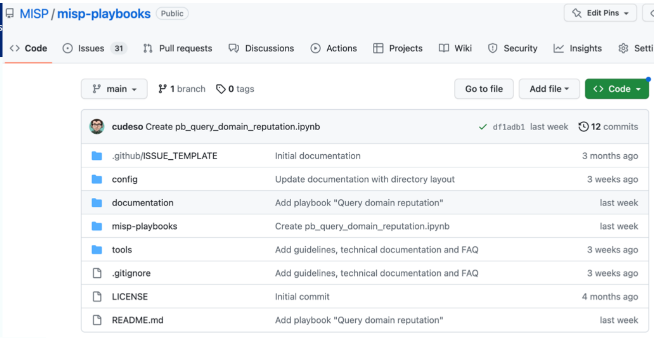
Speaker(s)

Koen Van Impe

October 19th

3:00 PM

30 minutes



MISP / misp-playbooks Public

Code | Issues 31 | Pull requests | Discussions | Actions | Projects | Wiki | Security | Insights | Setti

main · 1 branch · 0 tags · Go to file · Add file · Code

cudeso Create pb_query_domain_reputation.ipynb ✓ df1adb1 last week 12 commits

.github/ISSUE_TEMPLATE	Initial documentation	3 months ago
config	Update documentation with directory layout	3 weeks ago
documentation	Add playbook "Query domain reputation"	last week
misp-playbooks	Create pb_query_domain_reputation.ipynb	last week
tools	Add guidelines, technical documentation and FAQ	3 weeks ago
.gitignore	Add guidelines, technical documentation and FAQ	3 weeks ago
LICENSE	Initial commit	4 months ago
README.md	Add playbook "Query domain reputation"	last week

MISP playbooks

MISP playbooks

Use cases

- For CSIRT, SOC, CTI
- Detect, react and analyse intelligence received by **MISP**

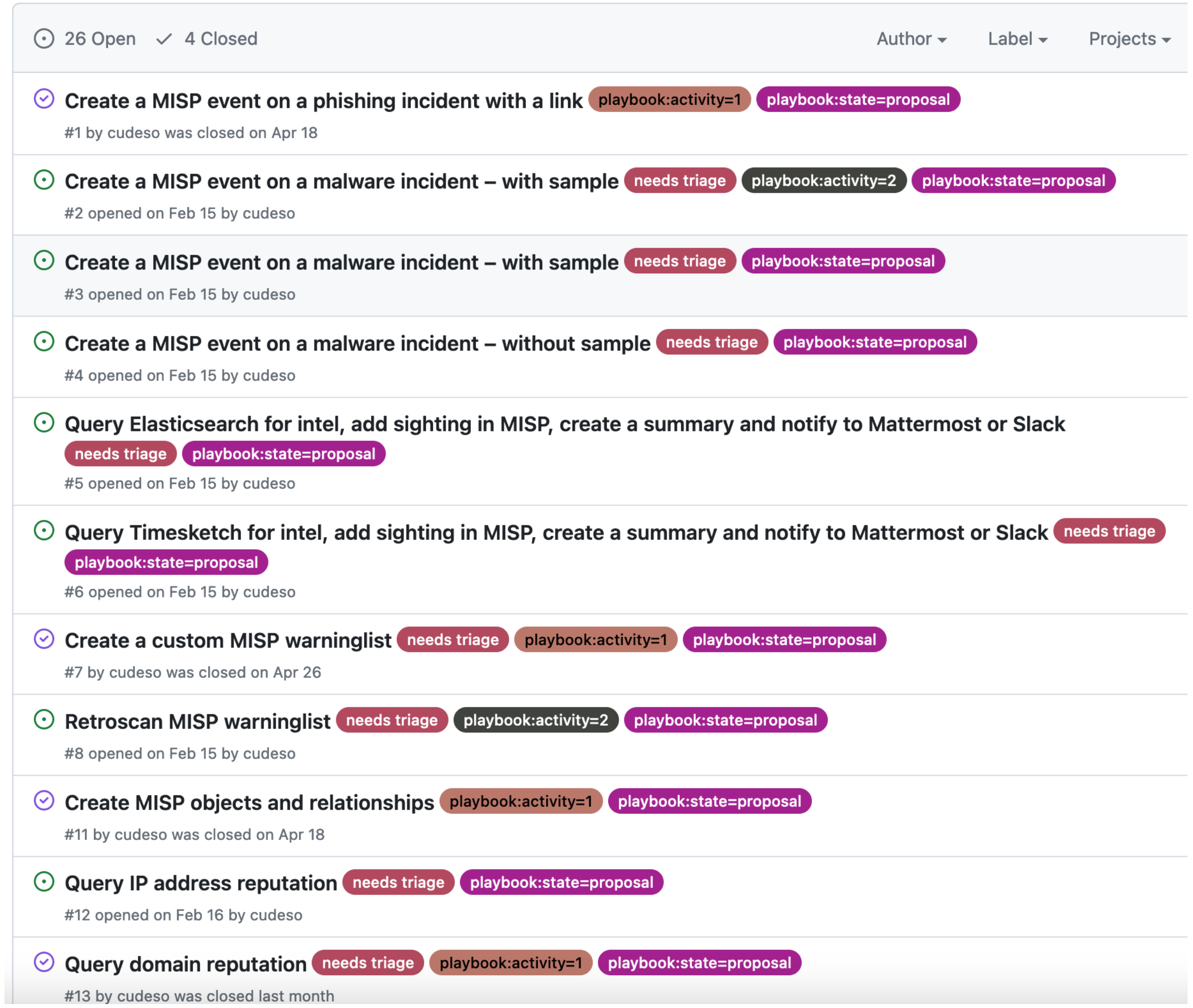
Consist of

- Jupyter notebooks with
  - **Documentation,** describing the “why”
  - Computer **code**, executing the playbook

GitHub

- https://github.com/MISP/misp-playbooks

https://github.com/MISP/misp-playbooks/



26 Open ✓ 4 Closed

Author ▾ Label ▾ Projects ▾

Create a MISP event on a phishing incident with a link playbook:activity=1 playbook:state=proposal
#1 by cudeso was closed on Apr 18

Create a MISP event on a malware incident – with sample needs triage playbook:activity=2 playbook:state=proposal
#2 opened on Feb 15 by cudeso

Create a MISP event on a malware incident – with sample needs triage playbook:state=proposal
#3 opened on Feb 15 by cudeso

Create a MISP event on a malware incident – without sample needs triage playbook:state=proposal
#4 opened on Feb 15 by cudeso

Query Elasticsearch for intel, add sighting in MISP, create a summary and notify to Mattermost or Slack needs triage playbook:state=proposal
#5 opened on Feb 15 by cudeso

Query Timesketch for intel, add sighting in MISP, create a summary and notify to Mattermost or Slack needs triage playbook:state=proposal
#6 opened on Feb 15 by cudeso

Create a custom MISP warninglist needs triage playbook:activity=1 playbook:state=proposal
#7 by cudeso was closed on Apr 26

Retroscan MISP warninglist needs triage playbook:activity=2 playbook:state=proposal
#8 opened on Feb 15 by cudeso

Create MISP objects and relationships playbook:activity=1 playbook:state=proposal
#11 by cudeso was closed on Apr 18

Query IP address reputation needs triage playbook:state=proposal
#12 opened on Feb 16 by cudeso

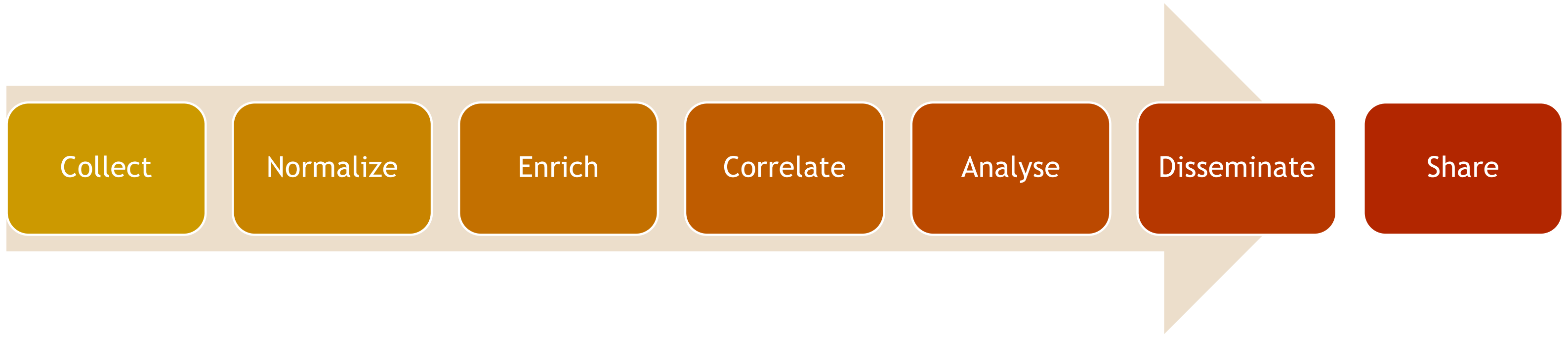
Query domain reputation needs triage playbook:activity=1 playbook:state=proposal
#13 by cudeso was closed last month

Building blocks of MISP playbooks

MISP and PyMISP

Co-financed by the European Union
Connecting Europe Facility

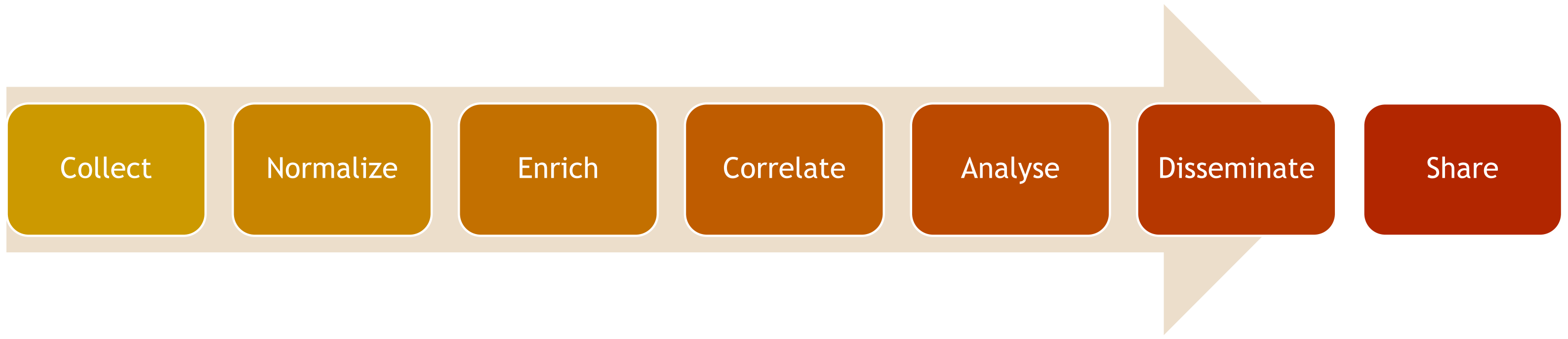
MISP is a **Threat Information Sharing Platform**

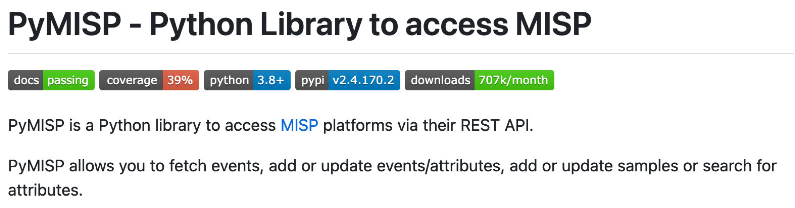


MISP and PyMISP

Co-financed by the European Union
Connecting Europe Facility

MISP is a **Threat Information Sharing Platform**




PyMISP - Python Library to access MISP

docs passing | coverage 39% | python 3.8+ | pypi v2.4.170.2 | downloads 707k/month

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.

- Automation
- Adding and editing data
- Integration
- ...

What are Jupyter notebooks?

Interactive environment

- Write and **execute** computer **code**
  - Observe the results
- **Documentation**
  - Text elements
  - Markdown
  - Images

Consumers

- Machines
  - Execute the code
- Human
  - Results of code execution
  - Documentation

Kernel

- Computational engine
- Executes the machine code

Distributed

- **Code** and **documentation** are stored in the “execution environment”
- But documentation can be edited from anywhere
- Web browser

Different flavours of Jupyter notebooks

Jupyter Notebook

- Single user, classic version

JupterLab

- Single user, new ”slick” look
- *MISP playbooks are tested and developed in JupyterLab, but should work in other flavours of Jupyter as well*

JupyterHub

- Multi user, server version

What are Jupyter notebooks?

- **Open source**
  - Used in data science
  - Other areas, such as … CTI
    - https://infosecjupyterthon.com

- Notebooks are stored in a **JSON** format (.ipynb)
  - Ideal for code repositories

Open format	Shareable
Easy to use	Version tracking

- Engines (kernel)
  - **Python**
  - Ruby, C++

What are Jupyter notebooks?

- **Open source**
  - Used in data science
  - Other areas, such as … CTI
    - https://infosecjupyterthon.com

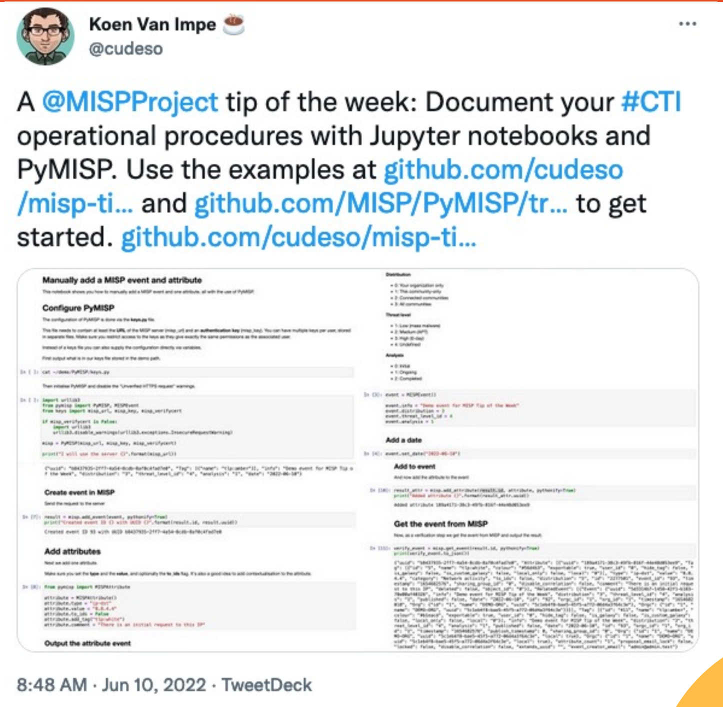
- Notebooks are stored in a **JSON** format (.ipynb)
  - Ideal for code repositories

Open format	Shareable
Easy to use	Version tracking

- Engines (kernel)
  - **Python** python™
  - Ruby, C++

PyMISP

Jupyter notebooks, PyMISP and CTI operational procedures



Koen Van Impe ☕
@cudeso

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

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.

```
In [ ]: 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))

{"uuid": "[illegible]", "Tag": [{"name": "tlp:amber"}], "info": "Demo event for MISP Tip o
f the Week", "distribution": "3", "threat_level_id": "4", "analysis": "1", "date": "2022-06-10"}
```

Create event in MISP

Send the request to the server

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

Created event ID 93 with UUID [illegible]
```

Add attributes

Next we add one attribute.

Make sure you set the **type** and the **value**, and optionally the **to_ids** flag. It's also a good idea to add contextualisation to the attribute.

```
In [8]: from pymisp import MISPAttribute

        attribute = MISPAttribute()
        attribute.type = "ip-dst"
        attribute.value = "8.8.4.4"
        attribute.to_ids = False
        attribute.add_tag("tlp:white")
        attribute.comment = "There is an initial request to this IP"
```

Output the attribute event

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 [3]: event = MISPEvent()

        event.info = "Demo event for MISP Tip of the Week"
        event.distribution = 3
        event.threat_level_id = 4
        event.analysis = 1
```

Add a date

```
In [4]: event.set_date("2022-06-10")
```

Add to event

And now add the attribute to the event

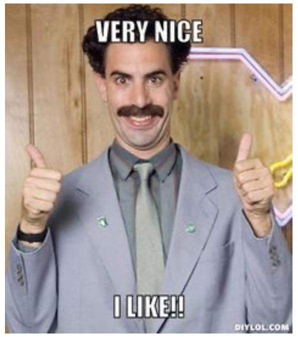
```
In [10]: result_attr = misp.add_attribute(result.id, attribute, pythonify=True)
         print("Added attribute {}".format(result_attr.uuid))

Added attribute [illegible]
```

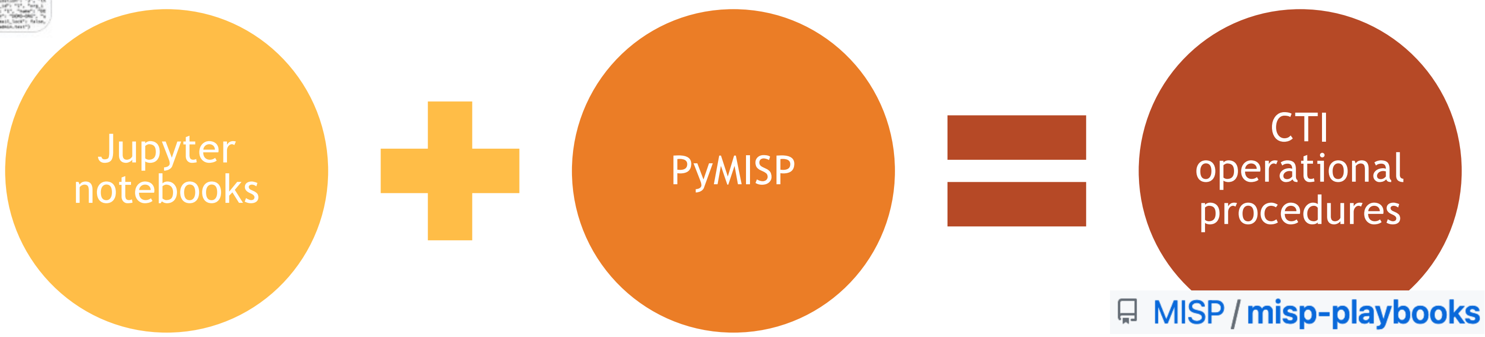
Get the event from MISP

Now, as a verification step we get the event from MISP and output the result.

```
In [11]: verify_event = misp.get_event(result.id, pythonify=True)
         print(verify_event.to_json())

[illegible]
```

8:48 AM · Jun 10, 2022 · TweetDeck

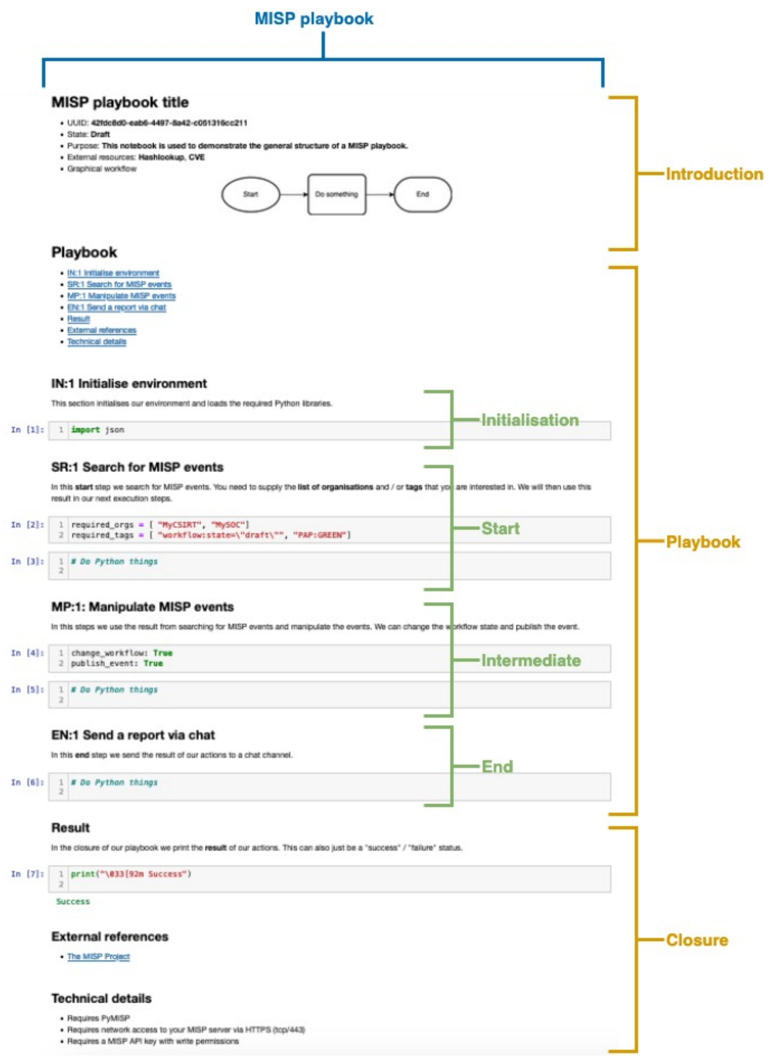




MISP / **misp-playbooks**

Structure of a MISP playbook

- **Introduction**
  - Meta data of playbook
  - Required environment (libraries)
  - Workflow
- **Execution steps**
  - "The playbook"
  - Documentation and code
    - Markdown and Python
- **Closure**
  - Summary of actions
  - Disseminate the results
    - Mattermost and TheHive



MISP playbook title

- UUID: **42fdc8d0-eab6-4497-8a42-c051316cc211**
- State: **Draft**
- Purpose: **This notebook is used to demonstrate the general structure of a MISP playbook.**
- External resources: **Hashlookup, CVE**
- Graphical workflow

Playbook

- IN:1 Initialise environment
- SR:1 Search for MISP events
- MP:1 Manipulate MISP events
- EN:1 Send a report via chat
- Result
- External references
- Technical details

IN:1 Initialise environment

This section initialises our environment and loads the required Python libraries.

```
In [1]: import json
```

SR:1 Search for MISP events

In this **start** step we search for MISP events. You need to supply the **list of organisations** and / or **tags** that you are interested in. We will then use this result in our next execution steps.

```
In [2]: required_orgs = [ "MyCSIRT", "MySOC"]
        required_tags = [ "workflow:state=\"draft\"", "PAP:GREEN"]

In [3]: # Do Python things
```

MP:1: Manipulate MISP events

In this steps we use the result from searching for MISP events and manipulate the events. We can change the workflow state and publish the event.

```
In [4]: change_workflow: True
        publish_event: True

In [5]: # Do Python things
```

EN:1 Send a report via chat

In this **end** step we send the result of our actions to a chat channel.

```
In [6]: # Do Python things
```

Result

In the closure of our playbook we print the **result** of our actions. This can also just be a "success" / "failure" status.

```
In [7]: print("\033[92m Success")

Success
```

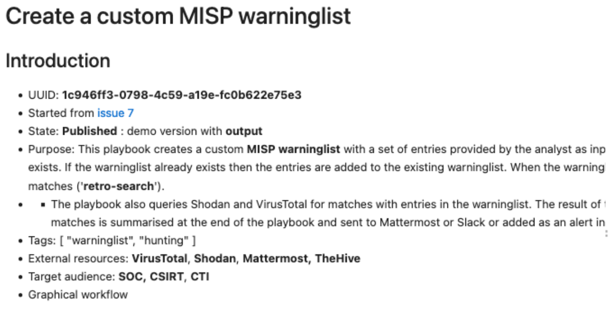
External references

- The MISP Project

Technical details

- Requires PyMISP
- Requires network access to your MISP server via HTTPS (tcp/443)
- Requires a MISP API key with write permissions

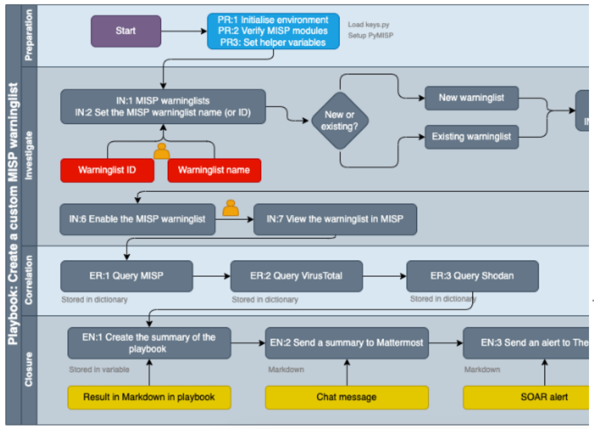
Example of a MISP playbook

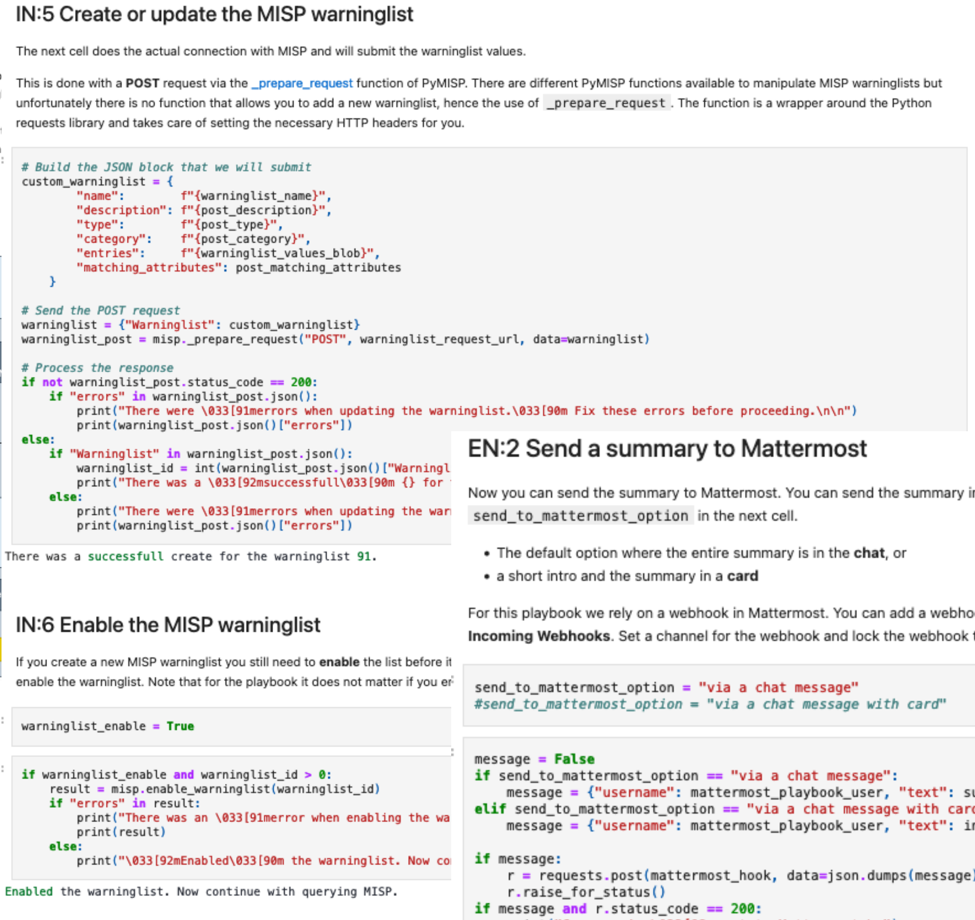


Create a custom MISP warninglist

Introduction

- UUID: **1c946ff3-0798-4c59-a19e-fc0b622e75e3**
- Started from issue 7
- State: **Published** : demo version with **output**
- Purpose: This playbook creates a custom **MISP warninglist** with a set of entries provided by the analyst as inp exists. If the warninglist already exists then the entries are added to the existing warninglist. When the warning matches ('**retro-search**').
-
  - The playbook also queries Shodan and VirusTotal for matches with entries in the warninglist. The result of matches is summarised at the end of the playbook and sent to Mattermost or Slack or added as an alert in
- Tags: ["warninglist", "hunting"]
- External resources: **VirusTotal, Shodan, Mattermost, TheHive**
- Target audience: **SOC, CSIRT, CTI**
- Graphical workflow





IN:5 Create or update the MISP warninglist

The next cell does the actual connection with MISP and will submit the warninglist values.

This is done with a **POST** request via the _prepare_request function of PyMISP. There are different PyMISP functions available to manipulate MISP warninglists but unfortunately there is no function that allows you to add a new warninglist, hence the use of `_prepare_request`. The function is a wrapper around the Python requests library and takes care of setting the necessary HTTP headers for you.

```
# Build the JSON block that we will submit
custom_warninglist = {
        "name":        f"{warninglist_name}",
        "description": f"{post_description}",
        "type":        f"{post_type}",
        "category":    f"{post_category}",
        "entries":     f"{warninglist_values_blob}",
        "matching_attributes": post_matching_attributes
    }

# Send the POST request
warninglist = {"Warninglist": custom_warninglist}
warninglist_post = misp._prepare_request("POST", warninglist_request_url, data=warninglist)

# Process the response
if not warninglist_post.status_code == 200:
    if "errors" in warninglist_post.json():
        print("There were \033[91merrors when updating the warninglist.\033[90m Fix these errors before proceeding.\n\n")
        print(warninglist_post.json()["errors"])
else:
    if "Warninglist" in warninglist_post.json():
        warninglist_id = int(warninglist_post.json()["Warningl
        print("There was a \033[92msuccessfull\033[90m {} for
    else:
        print("There were \033[91merrors when updating the war
        print(warninglist_post.json()["errors"])
```

```
There was a successfull create for the warninglist 91.
```

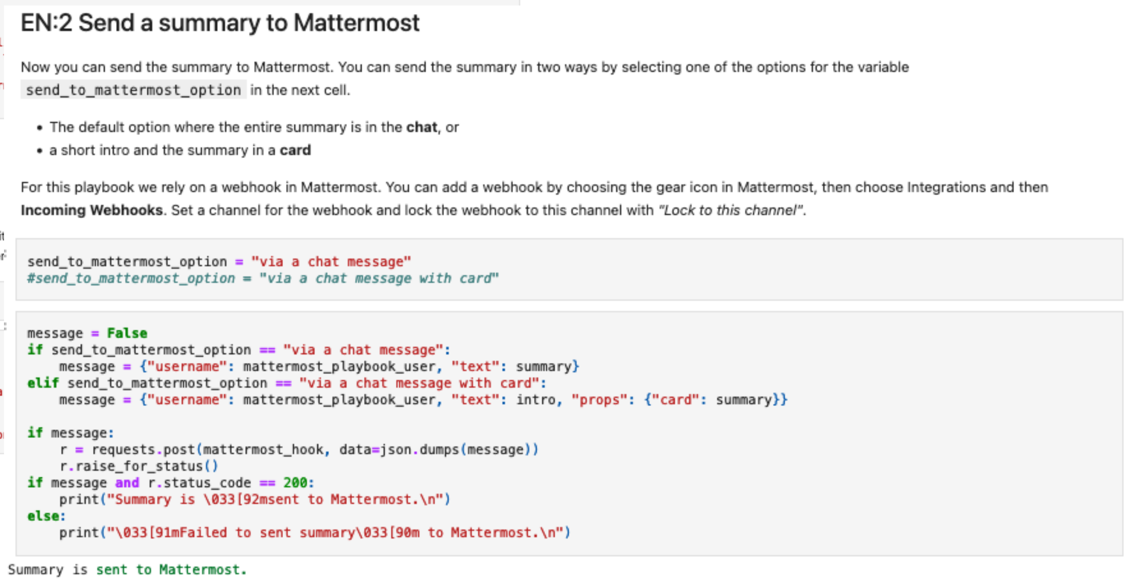
IN:6 Enable the MISP warninglist

If you create a new MISP warninglist you still need to **enable** the list before it enable the warninglist. Note that for the playbook it does not matter if you er

```
warninglist_enable = True
```

```
if warninglist_enable and warninglist_id > 0:
    result = misp.enable_warninglist(warninglist_id)
    if "errors" in result:
        print("There was an \033[91merror when enabling the wa
        print(result)
    else:
        print("\033[92mEnabled\033[90m the warninglist. Now co
```

```
Enabled the warninglist. Now continue with querying MISP.
```



EN:2 Send a summary to Mattermost

Now you can send the summary to Mattermost. You can send the summary in two ways by selecting one of the options for the variable `send_to_mattermost_option` in the next cell.

- The default option where the entire summary is in the **chat**, or
- a short intro and the summary in a **card**

For this playbook we rely on a webhook in Mattermost. You can add a webhook by choosing the gear icon in Mattermost, then choose Integrations and then **Incoming Webhooks**. Set a channel for the webhook and lock the webhook to this channel with *"Lock to this channel"*.

```
send_to_mattermost_option = "via a chat message"
#send_to_mattermost_option = "via a chat message with card"
```

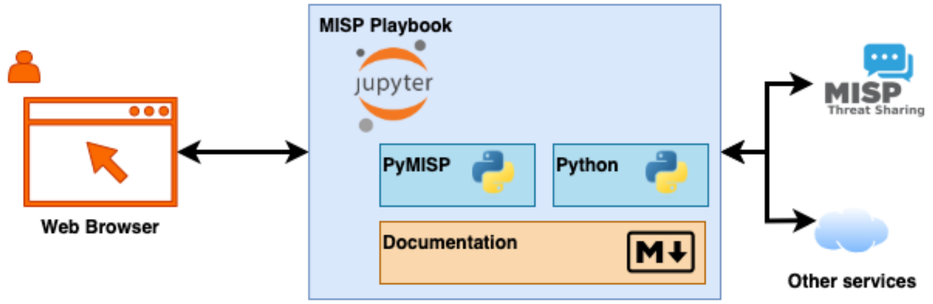
```
message = False
if send_to_mattermost_option == "via a chat message":
    message = {"username": mattermost_playbook_user, "text": summary}
elif send_to_mattermost_option == "via a chat message with card":
    message = {"username": mattermost_playbook_user, "text": intro, "props": {"card": summary}}

if message:
    r = requests.post(mattermost_hook, data=json.dumps(message))
    r.raise_for_status()
if message and r.status_code == 200:
    print("Summary is \033[92msent to Mattermost.\n")
else:
    print("\033[91mFailed to sent summary\033[90m to Mattermost.\n")
```

```
Summary is sent to Mattermost.
```

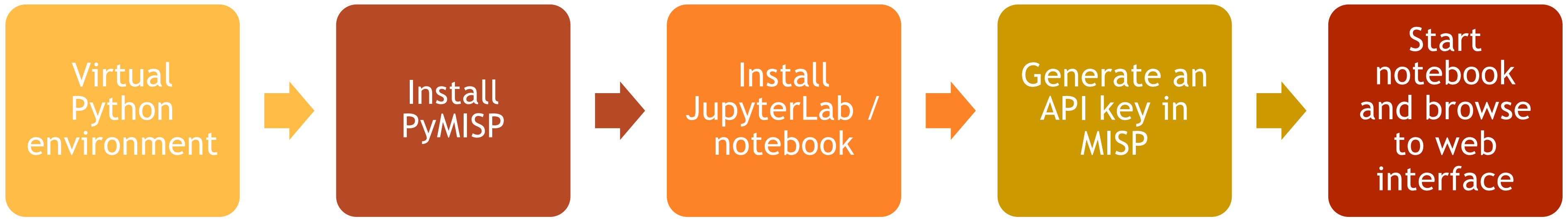
Setting up your environment

What do you need?

- Web **browser**
  - Run and edit the playbooks
  - Any modern browser, no plugins
- **Jupyter notebook** environment
  - Python 3
  - Jupyter Notebooks, JupyterLab, …
  - PyMISP
- A connection to a **MISP** server
  - Accounts at other external services
    - VirusTotal, URLscan.io, …
  - MISP modules (need to be accessible by the notebook)

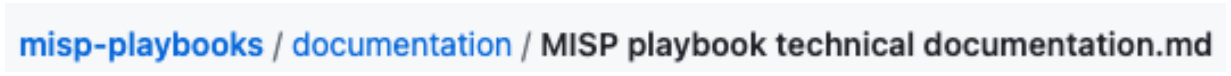


How to get started?


Virtual Python environment
Install PyMISP
Install JupyterLab / notebook
Generate an API key in MISP
Start notebook and browse to web interface

```
virtualenv venv
source venv/bin/activate


pip install pymisp
pip install jupyterlab | notebook


jupyter-lab --no-browser --ip misp.demo.cudeso.be --port 8899
          jupyter notebook --no-browser --ip misp.demo.cudeso.be --port 8899
```

Supporting documentation

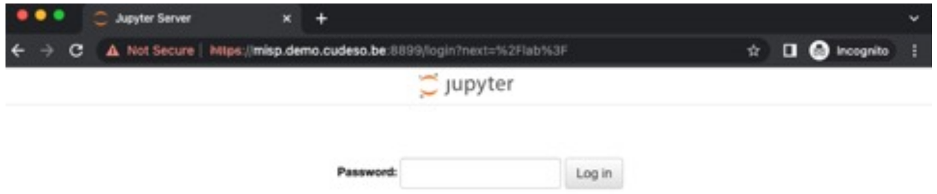

misp-playbooks / documentation / MISP playbook technical documentation.md

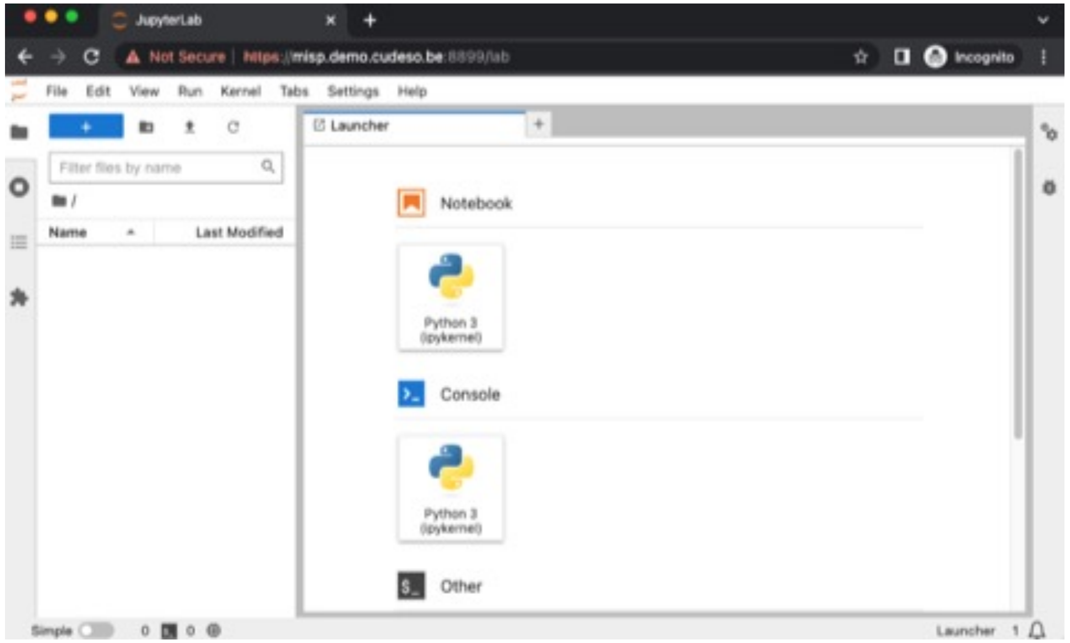
- **Configuration** file for JupyterLab
  - Restrict access to notebooks with a password
  - Set the network port
  - File locations where notebooks are stored

- **Systemd** startup script

- **NGINX** configuration file
  - If you want to put notebooks behind a reverse proxy

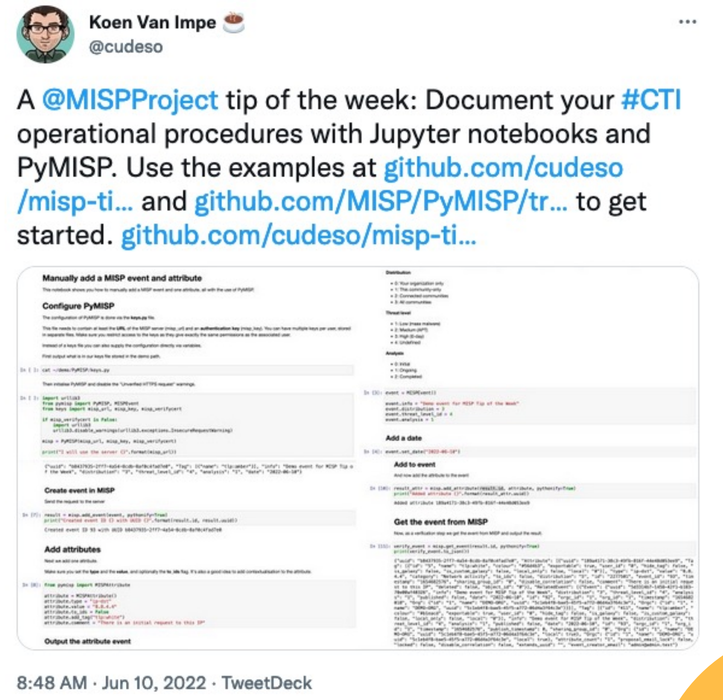
Access the MISP playbooks


Jupyter Server
Not Secure | https://misp.demo.cudeso.be:8899/login?next=%2Flab%3F
Incognito
jupyter
Password:
Log in



Guidelines for playbooks

Jupyter notebooks, PyMISP and CTI operational procedures



Koen Van Impe ☕
@cudeso

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

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.

```
In [ ]: 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))
```

```
{"uuid": "b8437935-2ff7-4a54-8cdb-8af8c4fad7e8", "Tag": [{"name": "tlp:amber"}], "info": "Demo event for MISP Tip o
f the Week", "distribution": "3", "threat_level_id": "4", "analysis": "1", "date": "2022-06-10"}
```

Create event in MISP

Send the request to the server

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

Created event ID 93 with UUID b8437935-2ff7-4a54-8cdb-8af8c4fad7e8
```

Add attributes

Next we add one attribute.

Make sure you set the **type** and the **value**, and optionally the **to_ids** flag. It's also a good idea to add contextualisation to the attribute.

```
In [8]: from pymisp import MISPAttribute

        attribute = MISPAttribute()
        attribute.type = "ip-dst"
        attribute.value = "8.8.4.4"
        attribute.to_ids = False
        attribute.add_tag("tlp:white")
        attribute.comment = "There is an initial request to this IP"
```

Output the attribute event

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 [3]: event = MISPEvent()

        event.info = "Demo event for MISP Tip of the Week"
        event.distribution = 3
        event.threat_level_id = 4
        event.analysis = 1
```

Add a date

```
In [4]: event.set_date("2022-06-10")
```

Add to event

And now add the attribute to the event

```
In [10]: result_attr = misp.add_attribute(result.id, attribute, pythonify=True)
         print("Added attribute {}".format(result_attr.uuid))

Added attribute 189a4171-38c3-497b-8167-44e4b0853ee9
```

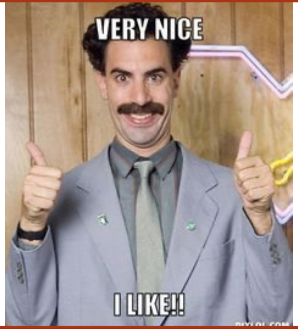
Get the event from MISP

Now, as a verification step we get the event from MISP and output the result.

```
In [11]: verify_event = misp.get_event(result.id, pythonify=True)
         print(verify_event.to_json())

{"uuid": "b8437935-2ff7-4a54-8cdb-8af8c4fad7e8", "Attribute": [{"uuid": "189a4171-38c3-497b-8167-44e4b0853ee9", "Ta
g": [{"id": "5", "name": "tlp:white", "colour": "#ffffff", "exportable": true, "user_id": "0", "hide_tag": false, "
is_galaxy": false, "is_custom_galaxy": false, "local_only": false, "local": "0"}], "type": "ip-dst", "value": "8.8.
4.4", "category": "Network activity", "to_ids": false, "distribution": "5", "id": "2277081", "event_id": "93", "tim
estamp": "1654882570", "sharing_group_id": "0", "disable_correlation": false, "comment": "There is an initial reque
st to this IP", "deleted": false, "object_id": "0"}], [illegible]
```

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

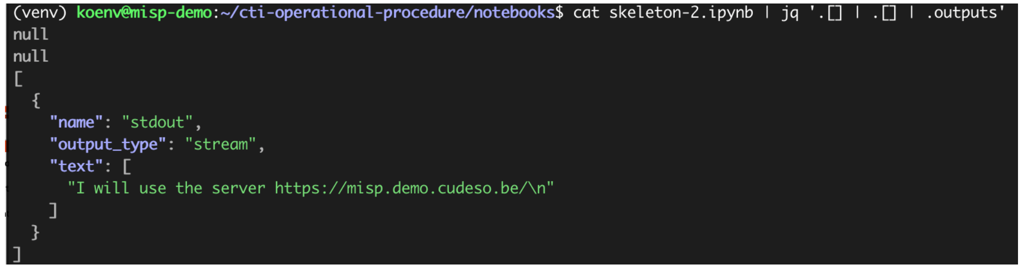
Jupyter notebooks + PyMISP = CTI operational procedures

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

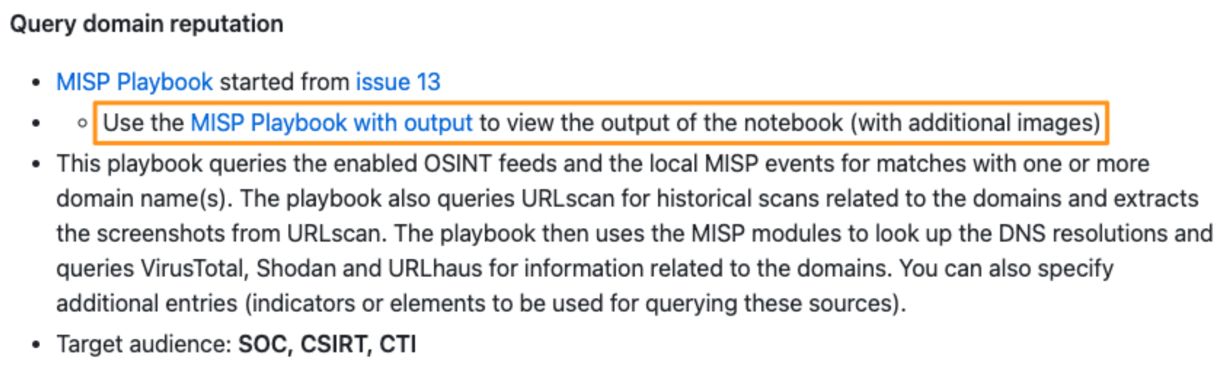


```
(venv) koenv@misp-demo:~/cti-operational-procedure/notebooks$ cat skeleton-2.ipynb | jq '.[] | .[] | .outputs'
null
null
[
  {
    "name": "stdout",
    "output_type": "stream",
    "text": [
      "I will use the server https://misp.demo.cudeso.be/\n"
    ]
  }
]
```

Do you want to share a report (with output) or procedure (clean)?

MISP playbooks exist as

- "Clean" version
- With output, for demonstration



Query domain reputation

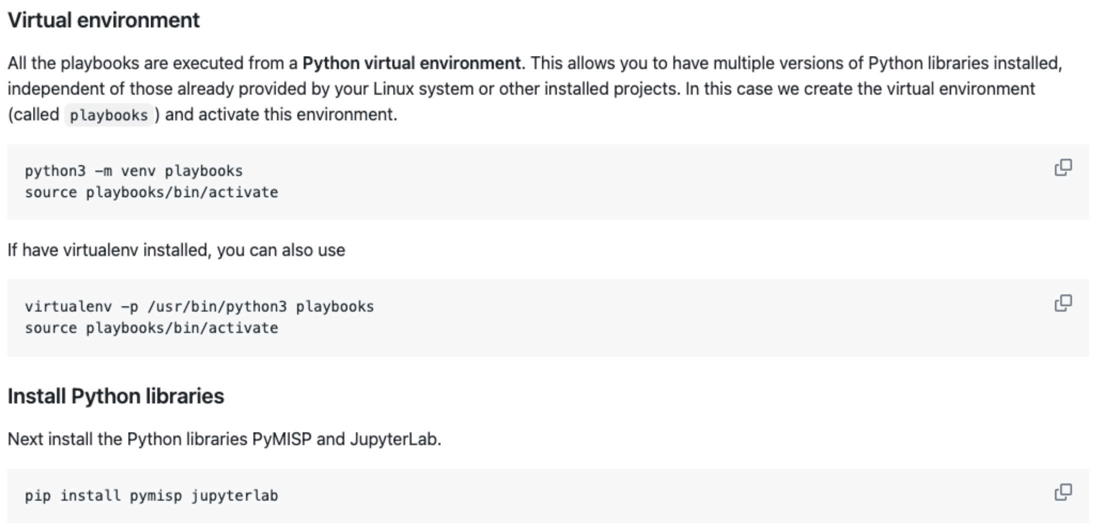
- MISP Playbook started from issue 13
-
  - Use the MISP Playbook with output to view the output of the notebook (with additional images)
- This playbook queries the enabled OSINT feeds and the local MISP events for matches with one or more domain name(s). The playbook also queries URLscan for historical scans related to the domains and extracts the screenshots from URLscan. The playbook then uses the MISP modules to look up the DNS resolutions and queries VirusTotal, Shodan and URLhaus for information related to the domains. You can also specify additional entries (indicators or elements to be used for querying these sources).
- Target audience: **SOC, CSIRT, CTI**

A note on the Jupyter notebook server

- Notebooks are served from the **directory** and **environment** where the server is executed
  - File location (path)
  - System environment conditions
  - (optionally) Access to MISP modules)

You can have multiple virtual environments per server.

- Python virtual environment
  - Install required Python libraries in that environment



Virtual environment

All the playbooks are executed from a **Python virtual environment**. This allows you to have multiple versions of Python libraries installed, independent of those already provided by your Linux system or other installed projects. In this case we create the virtual environment (called `playbooks`) and activate this environment.

```
python3 -m venv playbooks
source playbooks/bin/activate
```

If have virtualenv installed, you can also use

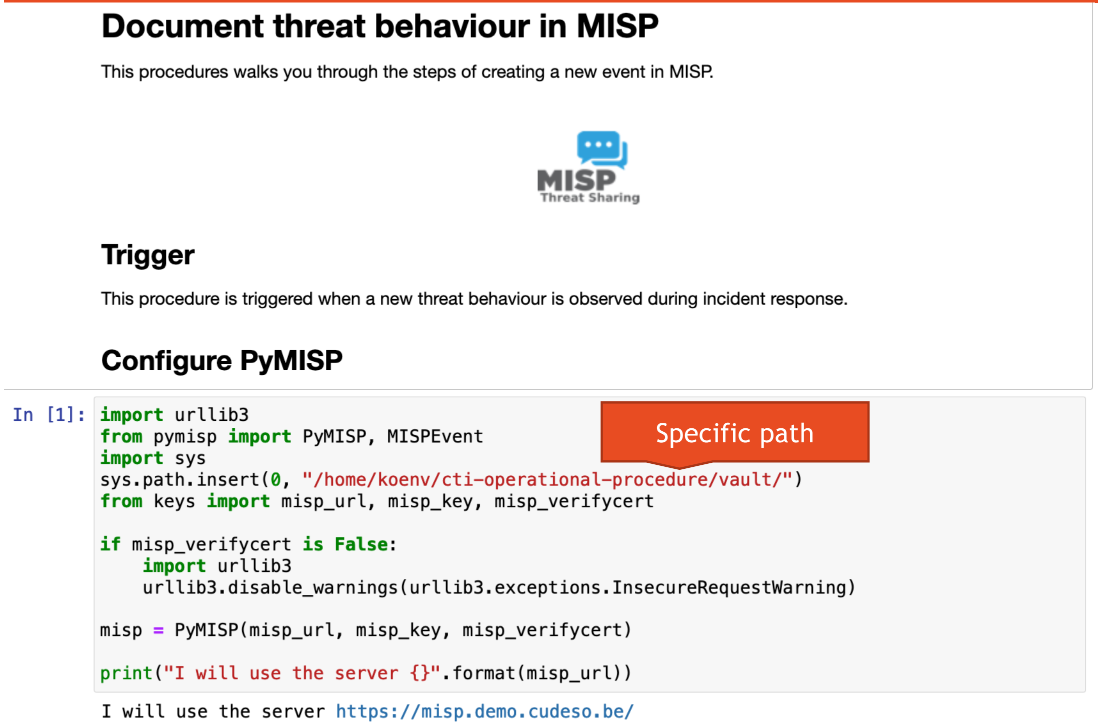
```
virtualenv -p /usr/bin/python3 playbooks
source playbooks/bin/activate
```

Install Python libraries

Next install the Python libraries PyMISP and JupyterLab.

```
pip install pymisp jupyterlab
```

Example: Create a MISP event



Document threat behaviour in MISP

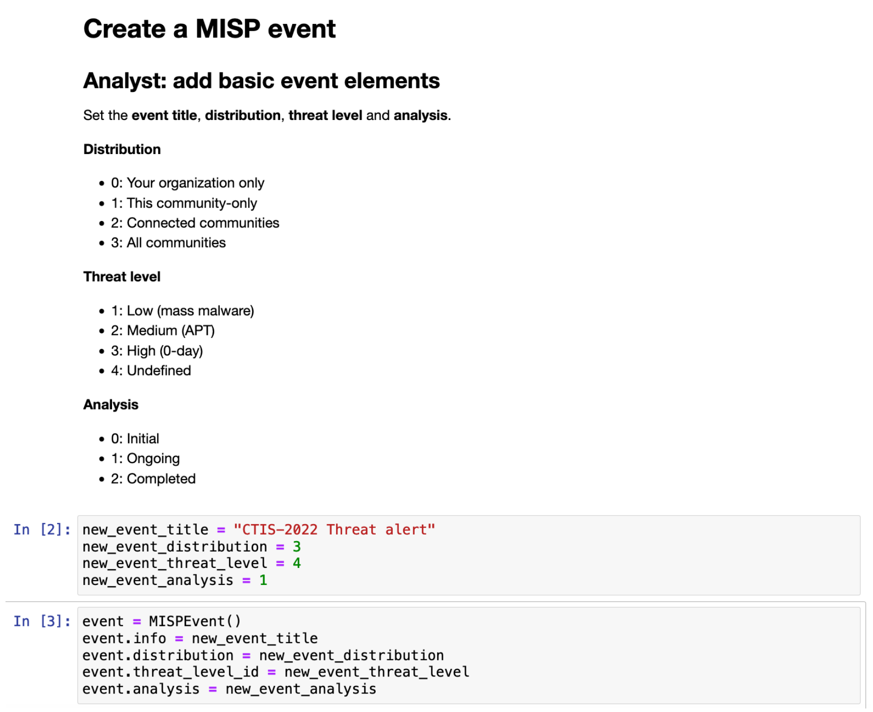
This procedures walks you through the steps of creating a new event in MISP.

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

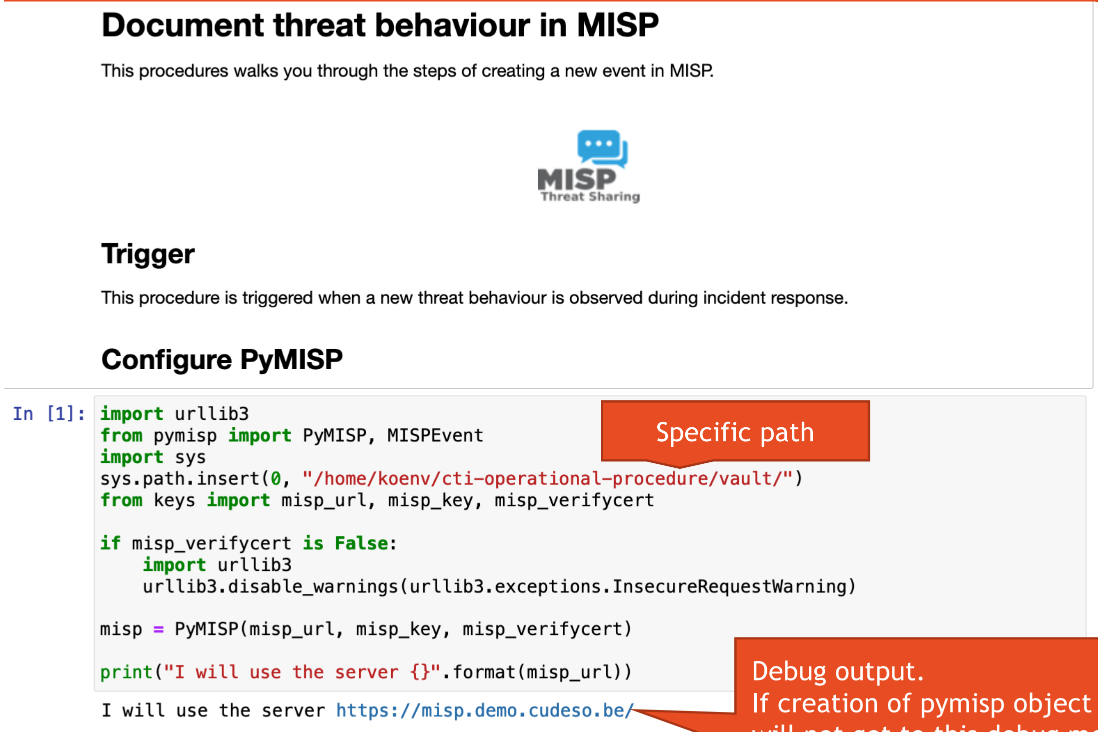
- 0: Initial
- 1: Ongoing
- 2: Completed

```
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
```

Do not store credentials in a notebook!

Example: Create a MISP event



Document threat behaviour in MISP

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

Trigger

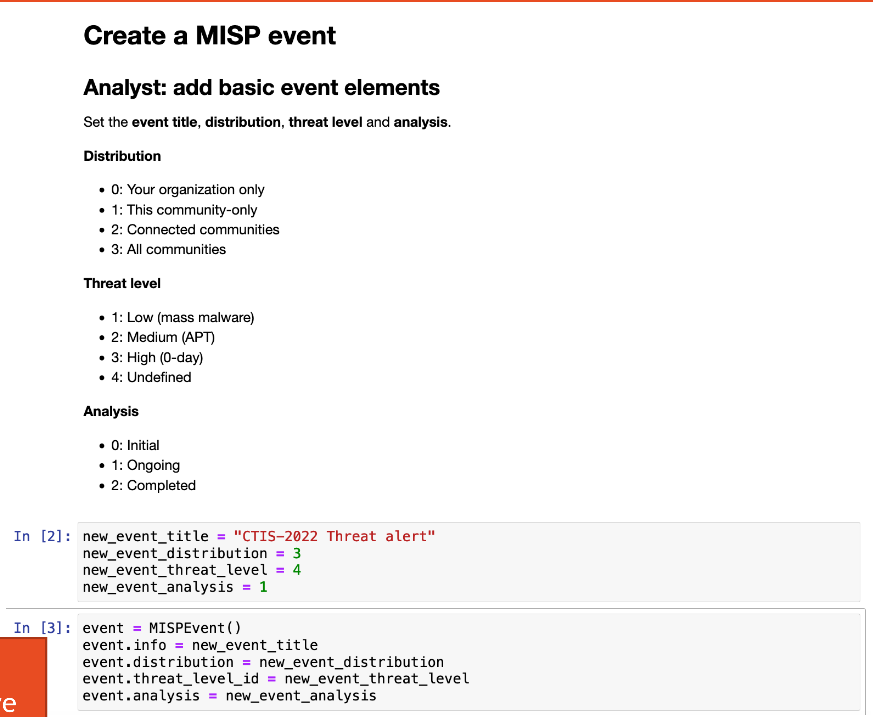
This procedure is triggered when a new threat behaviour is observed during incident response.

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/
```

Specific path

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



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

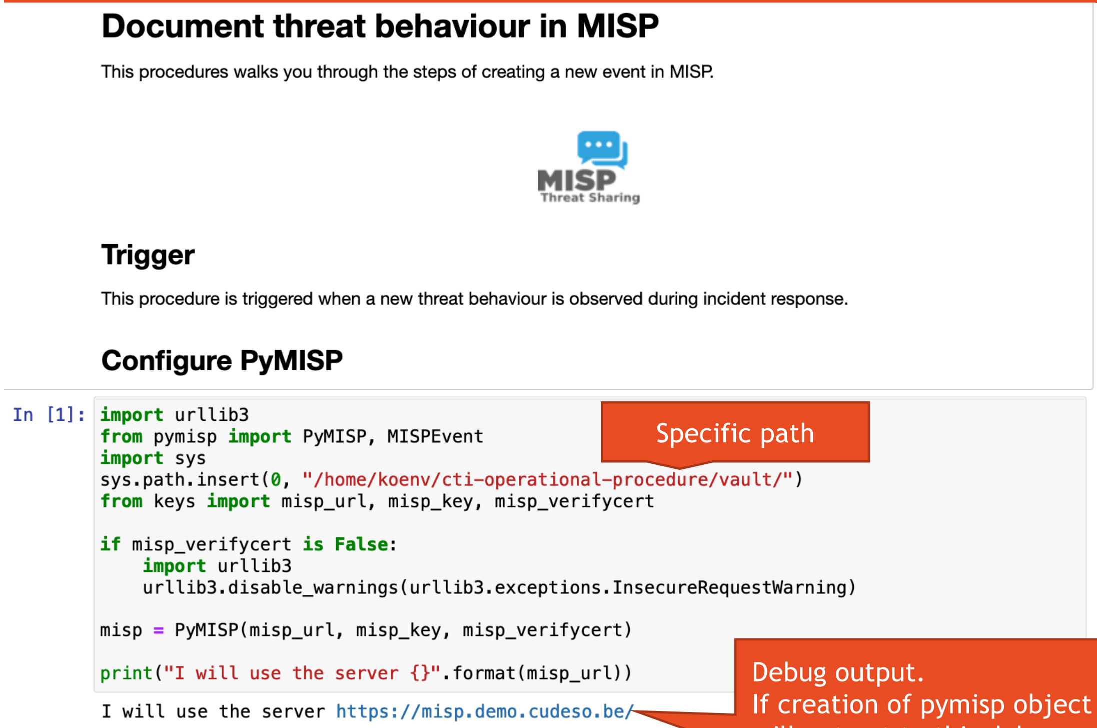
Analysis

- 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
```

Test the connection at the start of the procedure.

Example: Create a MISP event



Document threat behaviour in MISP

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

Trigger

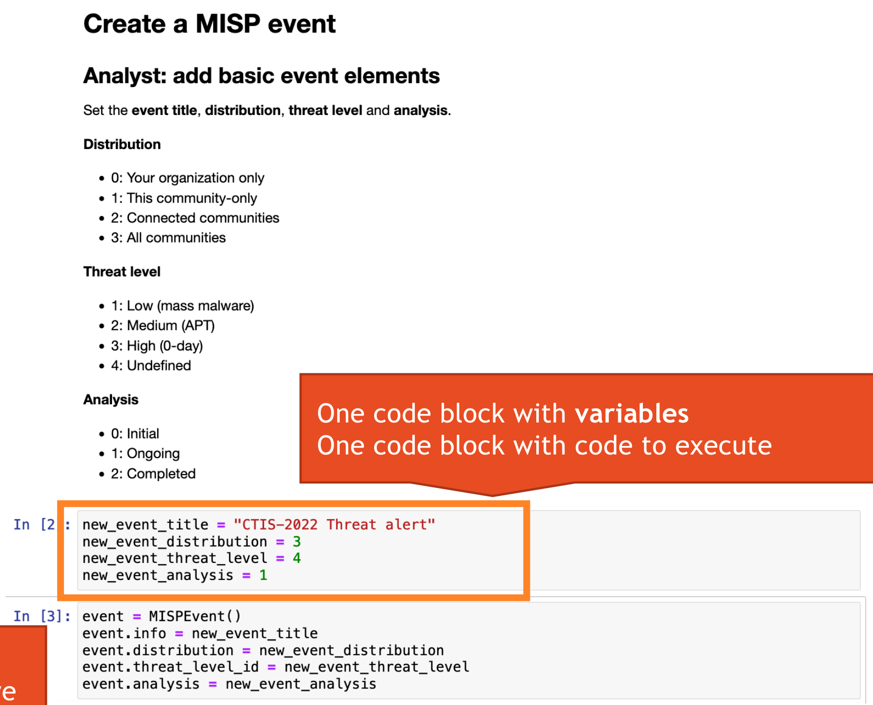
This procedure is triggered when a new threat behaviour is observed during incident response.

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/
```

Specific path

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



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

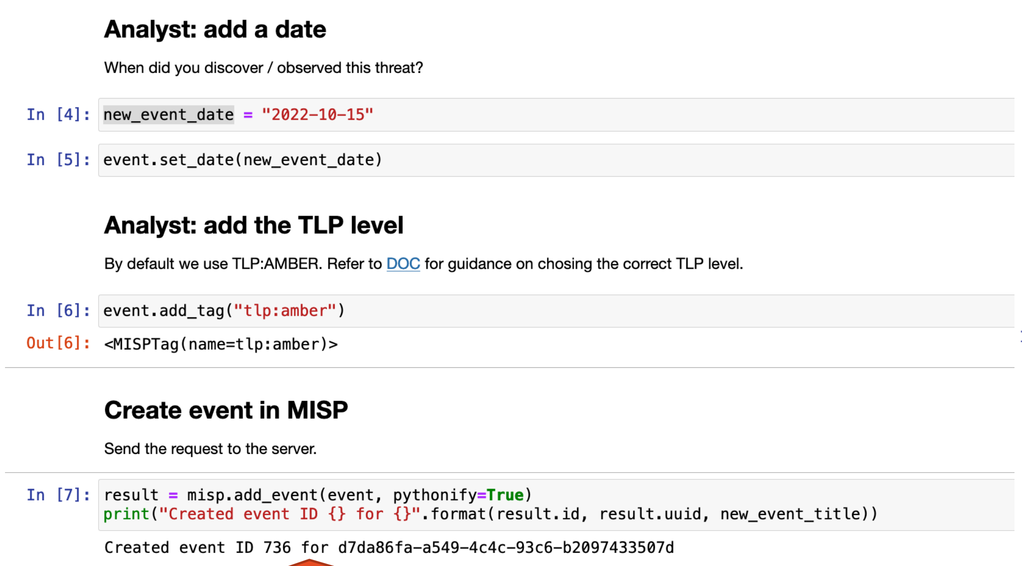
- 0: Initial
- 1: Ongoing
- 2: Completed

```
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
```

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

Avoid that analysts have to fiddle with 'raw' code.

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 chosing the correct TLP level.

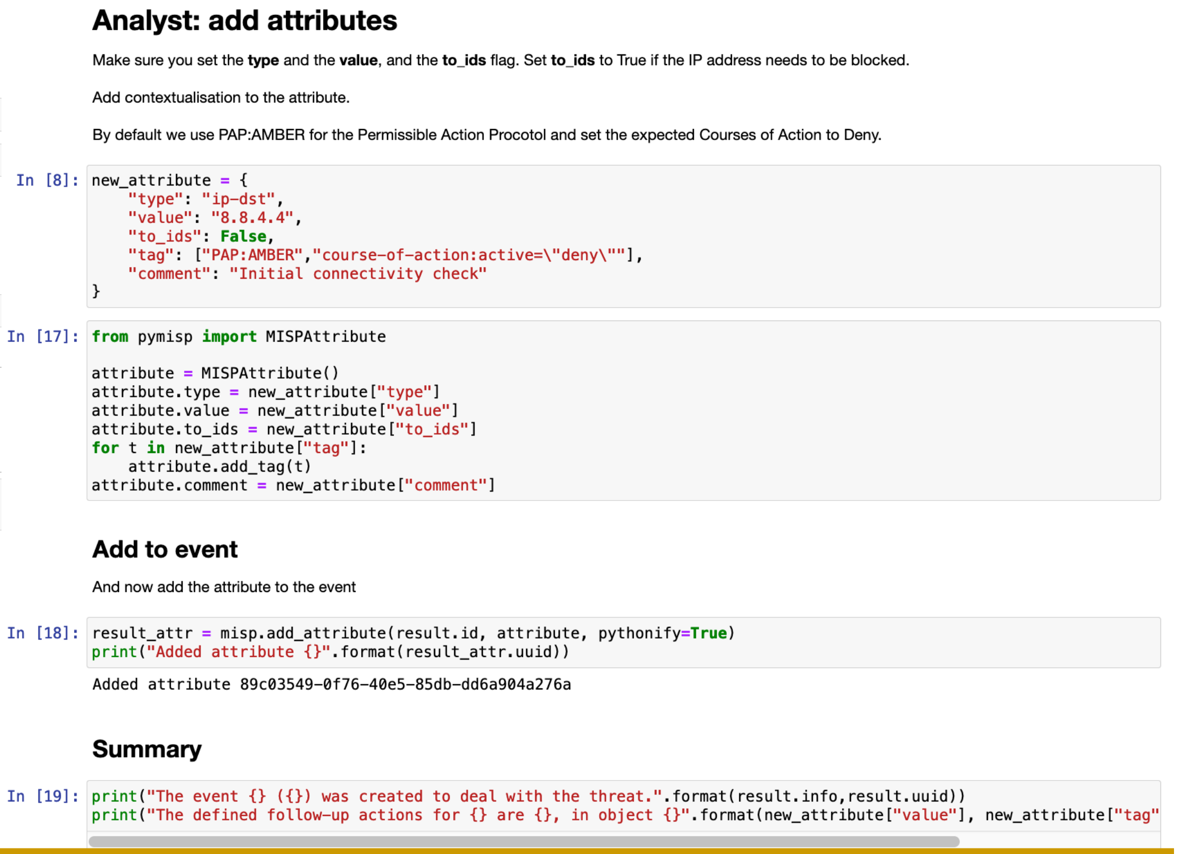
```
In [6]: event.add_tag("tlp:amber")
Out[6]: <MISPTag(name=tlp:amber)>
```

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.



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 Procotol 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 MISPAttribute

         attribute = MISPAttribute()
         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

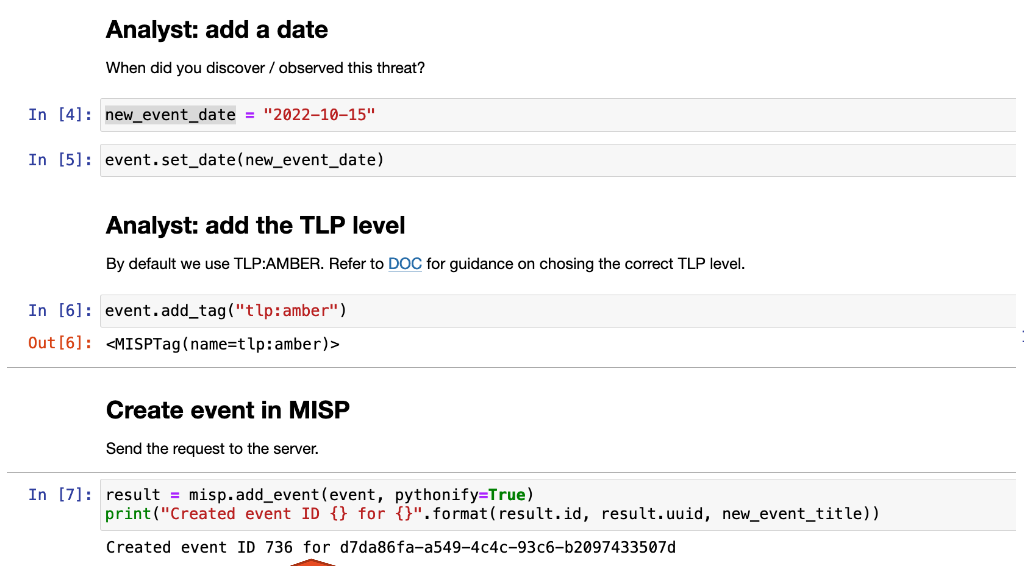
```
In [18]: result_attr = misp.add_attribute(result.id, attribute, pythonify=True)
         print("Added attribute {}".format(result_attr.uuid))

         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"
```

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 chosing the correct TLP level.

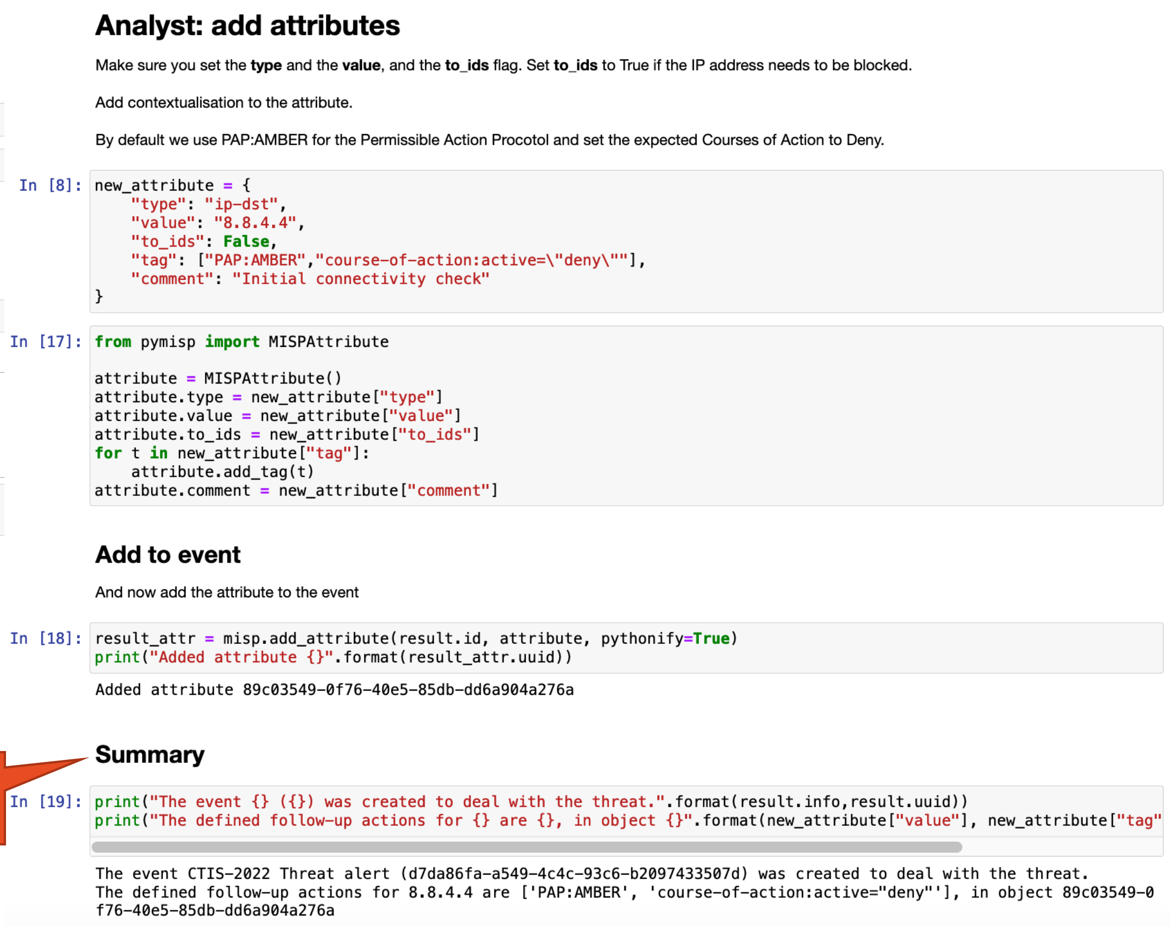
```
In [6]: event.add_tag("tlp:amber")
Out[6]: <MISPTag(name=tlp:amber)>
```

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.



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 Procotol 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 MISPAttribute

         attribute = MISPAttribute()
         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 89c03549-0
         f76-40e5-85db-dd6a904a276a
```

Print a summary at the end of execution

Conclude with a summary of what was done.

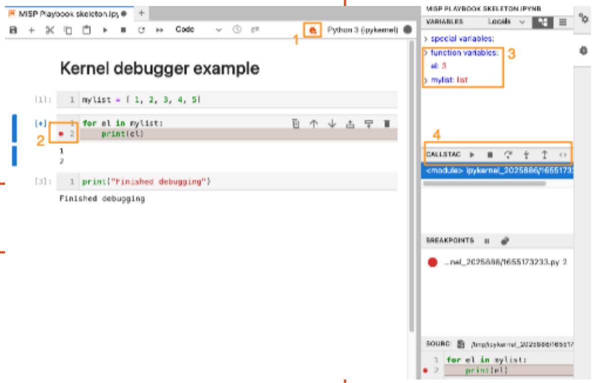
Code maintenance

Modularise the computer code

- Code re-use
- Load additional modules with the code
  - Less transportable

You don’t control the execution sequence

- Balance between error / state checking and ”heavy” code
- Print results of intermediate steps
  - JupyterLab has a built-in debugger



Kernel debugger example

```
[1]:  mylist = [ 1, 2, 3, 4, 5]

[*]:  for el in mylist:
          print(el)
1
2

[3]:  print("Finished debugging")
Finished debugging
```

```
<module> ipykernel_2025886/1655173
...nel_2025886/1655173233.py 2
/tmp/ipykernel_2025886/16551
for el in mylist:
    print(el)
```

Start from a “skeleton” playbook with basic building blocks

- Introduction, setting up the connection to MISP
- Event interaction
- Sending a summary to Mattermost or TheHive

Roadmap

Roadmap is in the list of GitHub issues

List of upcoming playbooks

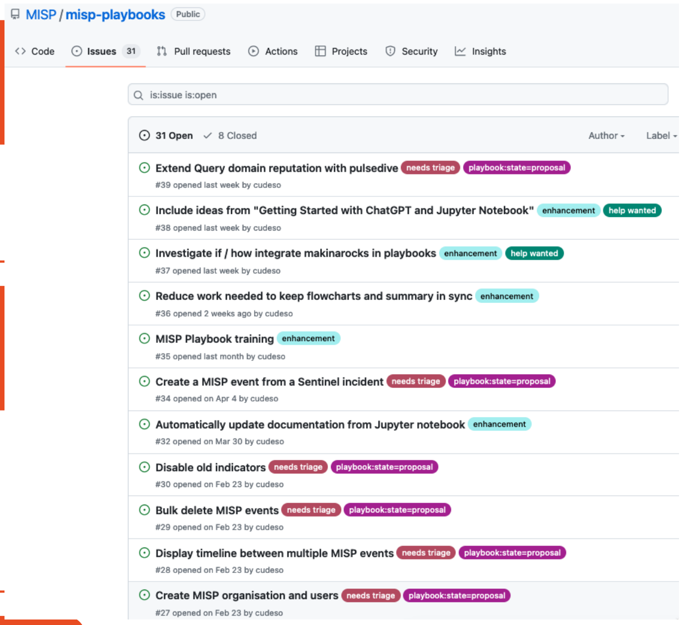
- Details in issue tracker

Desire a new playbook?

- Open an issue
- Or propose a draft

Integrate with MISP workflows

Convert to/from CACAO playbooks



Extend Query domain reputation with pulsedive needs triage playbook:state=proposal
#39 opened last week by cudeso

Include ideas from "Getting Started with ChatGPT and Jupyter Notebook" enhancement help wanted
#38 opened last week by cudeso

Investigate if / how integrate makinarocks in playbooks enhancement help wanted
#37 opened last week by cudeso

Reduce work needed to keep flowcharts and summary in sync enhancement
#36 opened 2 weeks ago by cudeso

MISP Playbook training enhancement
#35 opened last month by cudeso

Create a MISP event from a Sentinel incident needs triage playbook:state=proposal
#34 opened on Apr 4 by cudeso

Automatically update documentation from Jupyter notebook enhancement
#32 opened on Mar 30 by cudeso

Disable old indicators needs triage playbook:state=proposal
#30 opened on Feb 23 by cudeso

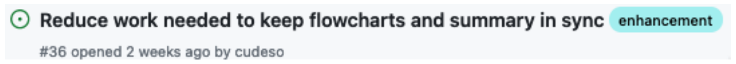
Bulk delete MISP events needs triage playbook:state=proposal
#29 opened on Feb 23 by cudeso

Display timeline between multiple MISP events needs triage playbook:state=proposal
#28 opened on Feb 23 by cudeso

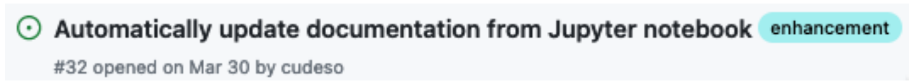
Create MISP organisation and users needs triage playbook:state=proposal
#27 opened on Feb 23 by cudeso

What works. What doesn't?

- **Graphical** workflow
  - Now in Drawio
  - Move to DOT for easier maintenance?
  - Update summary of playbook from DOT?
    - MakinaRocks?


Reduce work needed to keep flowcharts and summary in sync enhancement
#36 opened 2 weeks ago by cudeso

- **Summary** of the playbooks
  - Now manual
  - Extract info from introduction cell


Automatically update documentation from Jupyter notebook enhancement
#32 opened on Mar 30 by cudeso

Demo?

Questions?