MISP
INTELLIGENCE SHARING

Automation in MISP

Tutorial and Hands-On

Alexandre Dulaunoy & Sami Mokaddem

MISP Project
https://www.misp-project.org/

Content of the presentation

1. Automation in MISP
2. MISP API / PyMISP
3. PubSub channels (ZeroMQ)
4. MISP Workflows
   - Fundamentals
   - Demo with examples
   - Using the system
   - How it can be extended

Automation in MISP: What is available?

MISP API / PyMISP

- Needs CRON Jobs in place
- Potentially heavy for the server
- Not realtime

PubSub channels

- After the actions happen: No feedback to MISP
- Tougher to put in place & to share
- Full integration amounts to develop a new tool

MISP Workflow

- Realtime and allows feedback
- Easy to use and share but **can be dangerous**

MISP API / PyMISP - Fundamentals

Objective: Get to know how to use the MISP API PyMISP

MISP API / PyMISP - Demo

- Generate an API key
- RestClient overview
- MISP API Overview notebook[1]
- PyMISP Overview notebook[2]

[1] https://github.com/MISP/misp-training/blob/main/a.7-rest-API/Training%20-%20Using%20the%20API%20in%20MISP.ipynb

[2] https://github.com/MISP/PyMISP/blob/main/docs/tutorial/FullOverview.ipynb

PubSub channels (ZeroMQ) - Fundamentals

Objective: Learn how to setup realtime automation using the ZeroMQ channel

ZeroMQ channel - Demo

- What is ZeroMQ?
  - *N-to-N Asynchronous message-processing tasks*
  - *Publisher (MISP) and consumer (scripts)*
- Configuring ZeroMQ in MISP
- Integrating with the ZeroMQ of MISP

MISP Workflows - Fundamentals

Objective: Learn how to use the MISP Worklfow feature

Automation in MISP: What already exists?

MISP API / PyMISP

- Needs CRON Jobs in place
- Potentially heavy for the server
- Not realtime

PubSub channels

- After the actions happen: No feedback to MISP
- Tougher to put in place & to share
- Full integration amounts to develop a new tool

→ No way to **prevent** behavior
→ Difficult to setup **hooks** to execute callbacks

What type of use-cases are we trying to support?

- **Prevent** default MISP behaviors to happen
  - Prevent **publication of events** not passing sanity checks
  - Prevent **querying** thrid-party **services** with sensitive information
  - …
- **Hook** specific actions to run callbacks
  - **Automatically run** enrichment services
  - Modify data on-the-fly: False positives, enable CTI-Pipeline
  - Send notifications in a chat rooms
  - …

Simple automation in MISP made easy

- Why?
  - Everyone loves **simple automation**
  - **Visual** dataflow programming
  - Users want **more control**
- How?
  - **Drag & Drop** editor
  - Prevent actions **before they happen**
  - Flexible **Plug & Play** system
  - **Share** workflows, **debug** and **replay**

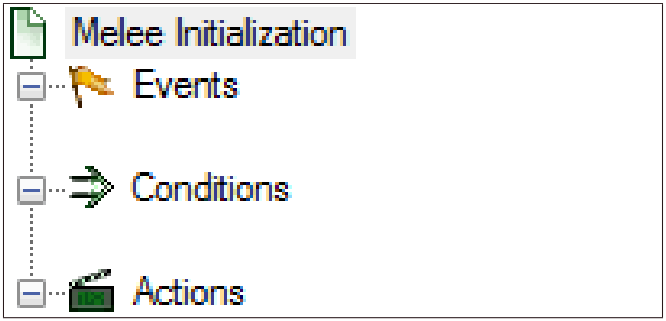
Example of use-cases

- **Notification** on specifc actions
  - New events matching criteria
  - New users
  - Automated alerts for high-priority IOCs
- **Extend** existing MISP behavior
  - Push data to another system
  - Automatic enrichment
  - Sanity check to block publishing / sharing
- **Hook** capabilities
  - Assign tasks and notify incident response team members
  - Run curation pipeline
- ...

Workflow - Fundamentals

Objective: Start with the foundation to understand the basics

How does it work



1. An **event** happens in MISP
2. Check if all **conditions** are satisfied
3. Execute all **actions**
   - May prevent MISP to complete its original event

What kind of events?

Events

- New MISP Event
- Attribute has been saved
- New discussion post
- New user created
- Query against third-party services
- ...

Supported events in MISP are called **Triggers**

A **Trigger** is associated with **1-and-only-1 Workflow**

Triggers currently available

Currently 17 triggers can be hooked. 3 being Blocking.

Triggers

List the available triggers that can be listened to by workflows.
Missing a trigger? Feel free to open a Github issue!
Documentation and concepts

« previous next »

All | attribute | event | object | others | post | user | Blocking | Enabled | Disabled

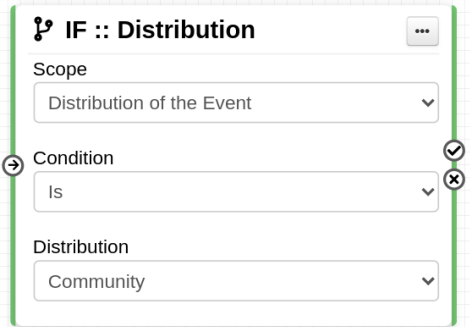
Trigger name	Scope	Trigger overhead	Run counter	Blocking Workflow	MISP Core format	Workflow ID	Last Update	Debug enabled	Enabled	Actions
Attribute After Save	attribute	high	83	×	✓	160	2022-08-03 09:00:41	☐	×	
Enrichment Before Query	others	low	1154	✓	✓	162	2022-10-17 12:35:57	☐	✓	
Event After Save	event	high	49	×	✓	175	2022-10-14 13:32:01	☐	✓	
Event After Save New	event	low	5	×	✓	182	2022-10-17 09:12:14	☑	✓	
Event After Save New From Pull	event	low	6	×	✓	183	2022-10-17 09:01:36	☑	✓	
Event Publish	event	low	126	✓	✓	180	2022-10-13 10:42:53	☑	✓	
Object After Save	object	high	35	×	✓	161	2022-08-05 07:12:52	☐	×	
Post After Save	post	low	36	×	×	176	2022-07-28 13:59:51	☐	×	
User After Save	user	low	0	×	×	181	2022-08-05 07:19:46	☐	×	
User Before Save	user	low	42	✓	×	158	2022-07-28 14:00:32	☐	×	

What kind of conditions?

Conditions

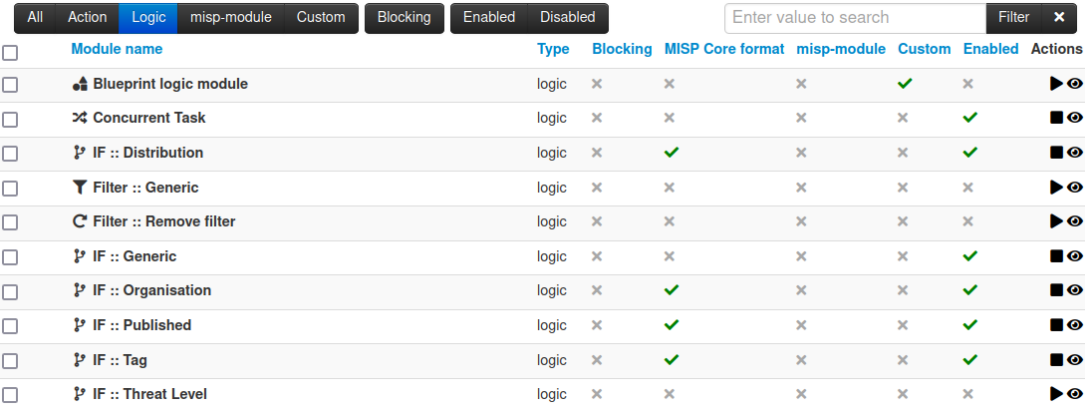
- A MISP Event is tagged with `tlp:red`
- The distribution of an Attribute is a sharing group
- The creator organisation is `circl.lu`
- Or any other **generic** conditions

These are also called **Logic modules**



Workflow - Logic modules

- **logic** modules: Allow to redirect the execution flow.
  - IF conditions
  - Delay execution



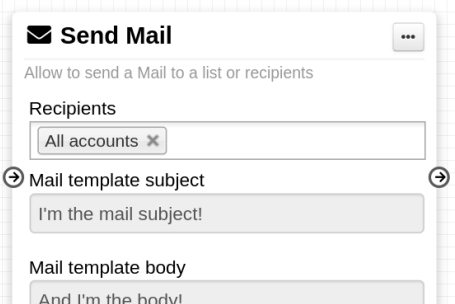
Module name	Type	Blocking	MISP Core format	misp-module	Custom	Enabled	Actions
Blueprint logic module	logic	✗	✗	✗	✓	✗	▶ 👁
Concurrent Task	logic	✗	✗	✗	✗	✓	■ 👁
IF :: Distribution	logic	✗	✓	✗	✗	✓	■ 👁
Filter :: Generic	logic	✗	✗	✗	✗	✗	▶ 👁
Filter :: Remove filter	logic	✗	✗	✗	✗	✗	▶ 👁
IF :: Generic	logic	✗	✗	✗	✗	✓	■ 👁
IF :: Organisation	logic	✗	✓	✗	✗	✓	■ 👁
IF :: Published	logic	✗	✓	✗	✗	✓	■ 👁
IF :: Tag	logic	✗	✓	✗	✗	✓	■ 👁
IF :: Threat Level	logic	✗	✗	✗	✗	✗	▶ 👁

What kind of actions?

Actions

- Send an email notification
- Perform enrichments
- Send a chat message on MS Teams
- Attach a local tag
- ...

These are also called **Action modules**



Workflow - Action modules

- **action** modules: Allow to executes operations
  - Tag operations
  - Send notifications
  - Webhooks & Custom scripts

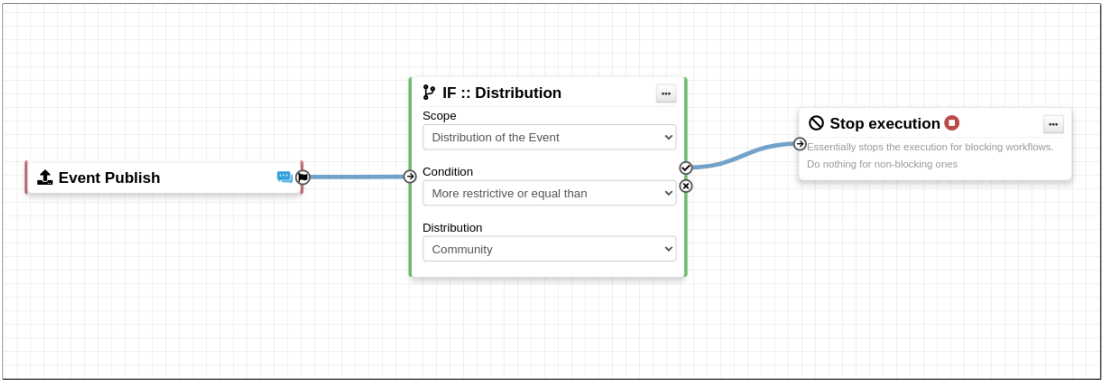
All | Action | Logic | misp-module | Custom | Blocking | Enabled | Disabled

Enter value to search | Filter

Module name	Type	Blocking	MISP Core format	misp-module	Custom	Enabled	Actions
Attach enrichment	action	×	✓	×	×	✓	
Attribute edition operation	action	×	✓	×	×	✓	
Attribute IDS Flag operation	action	×	✓	×	×	✓	
Blueprint action module	action	×	×	×	✓	✓	
Enrich Event	action	×	✓	×	×	✓	
mattermost	action	×	×	✓	×	✓	
MS Teams Webhook	action	×	×	×	×	✓	
Push to ZMQ	action	×	×	×	×	✓	
Send Log Mail	action	×	×	×	×	×	
Send Mail	action	×	×	×	×	✓	
Splunk HEC export	action	×	✓	×	×	×	
	action	✓	×	×	×	✓	

What is a MISP Workflow?

- Sequence of all nodes to be executed in a specific order



Workflow execution for Event publish

- An Event is about to be published
  - The workflow for the `event-publish` trigger starts
- Conditions are evaluated
  - They might change the path taken during the execution
- Actions are executed
  - **success**: Continue the publishing action

execute_workflow	Finished executing workflow for trigger `event-publish` (180). Outcome: success

  - **failure** | `blocked`: Stop publishing and log the reason

execute_workflow	Execution stopped. Node `stop-execution` (8) from Workflow `Workflow for trigger event-publish` (180) returned the following error: Execution stopped

Blocking and non-blocking

Two types of workflows:

Blocking Workflows

- Can prevent / block the original event to happen
- If a **blocking module** blocks the action

Non blocking Workflows execution doesn't block the request

- Executed in the background **after the request**:
  No way to prevent something that happened in the past

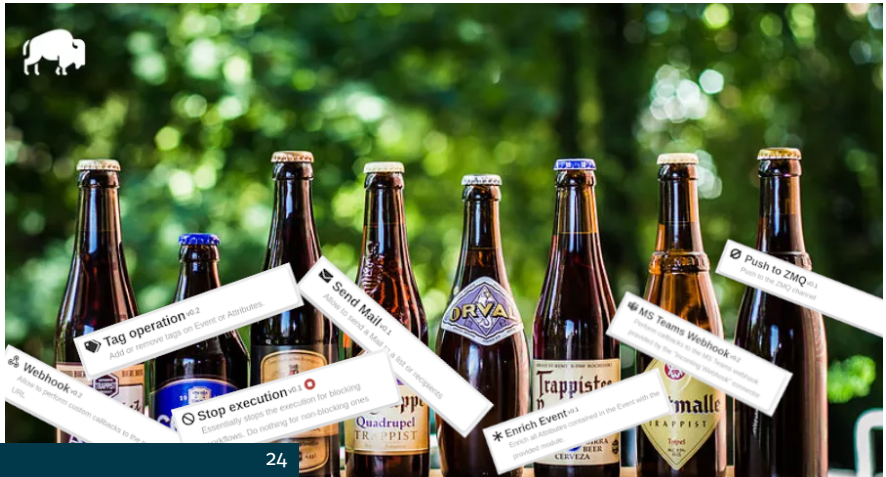
Sources of Workflow modules (0)

Currently 43 built-in modules.

- **Logic** module (11): built-in & **custom**
- **Action** module (32): built-in & **custom**

Sources of Workflow modules (1)

- Built-in **default** modules
  - Part of the MISP codebase
  - Get in touch if you want us to increase the selection (or merge PR!)



Sources of Workflow modules (2)

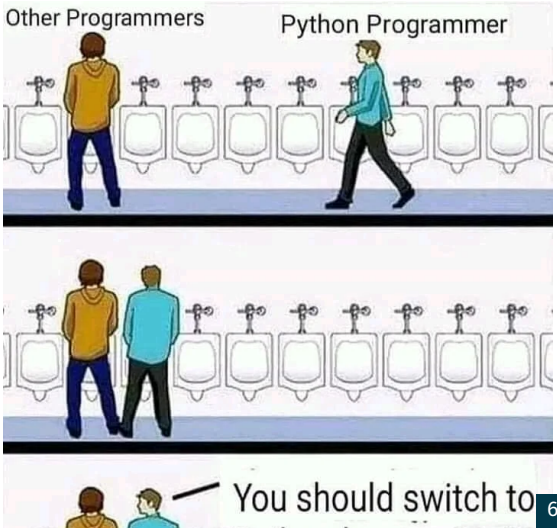
User-defined **custom** modules

- Written in PHP
- Extend existing modules
- MISP code reuse

Sources of Workflow modules (3)

Modules from the misp-module **enrichment service**

- Written in Python
- Can use any python libraries
- Plug & Play



Demo by examples

WF-1. Send an email to **all** when a new event has been pulled

WF-2. Block queries on 3rd party services when **tlp:red** or **PAP:red**

- **tlp:red**: For the eyes and ears of individual recipients only
- **PAP:RED**: Only passive actions that are not detectable from the outside

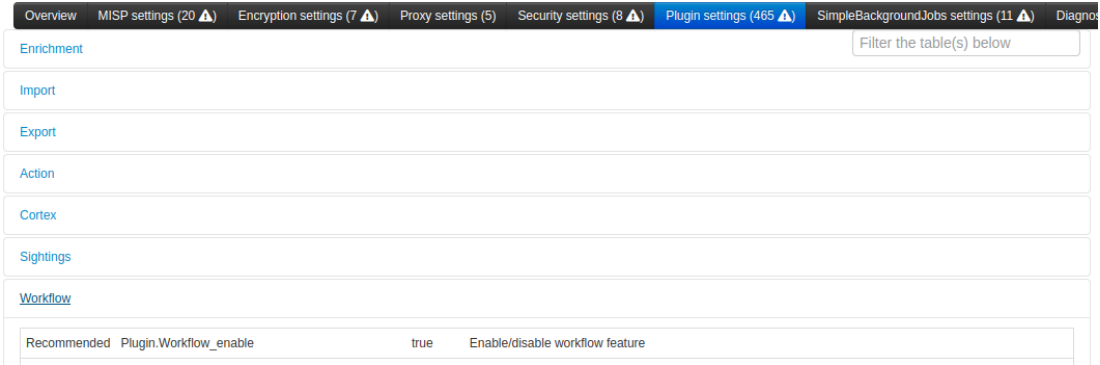
Workflow - Getting started

Objective: How to install & configure workflows

Getting started with workflows (2)

Review MISP settings:

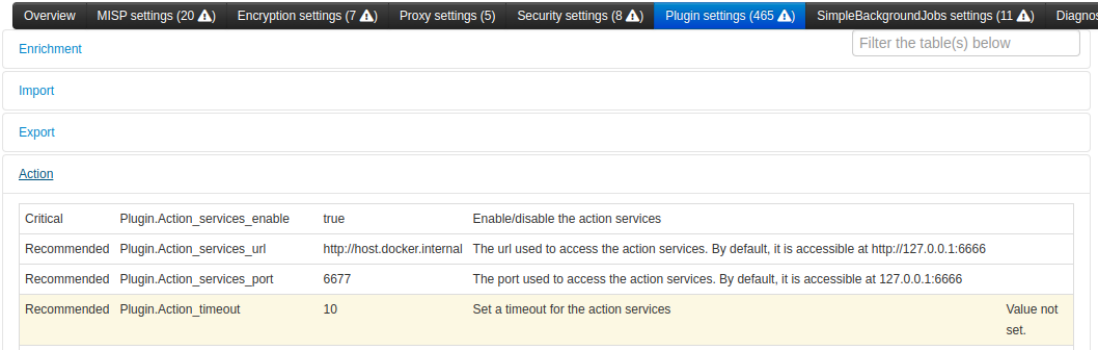
1. Make sure `MISP.background_jobs` is turned on
2. Make sure workers are up-and-running and healthy
3. Turn the setting `Plugin.Workflow_enable` on



Getting started with workflows (3)

Review MISP settings:

4. [optional:misp-module] Turn the setting `Plugin.Action_services_enable` on



Everything is ready?

Let’s see how to build a workflow!

Creating a workflow with the editor

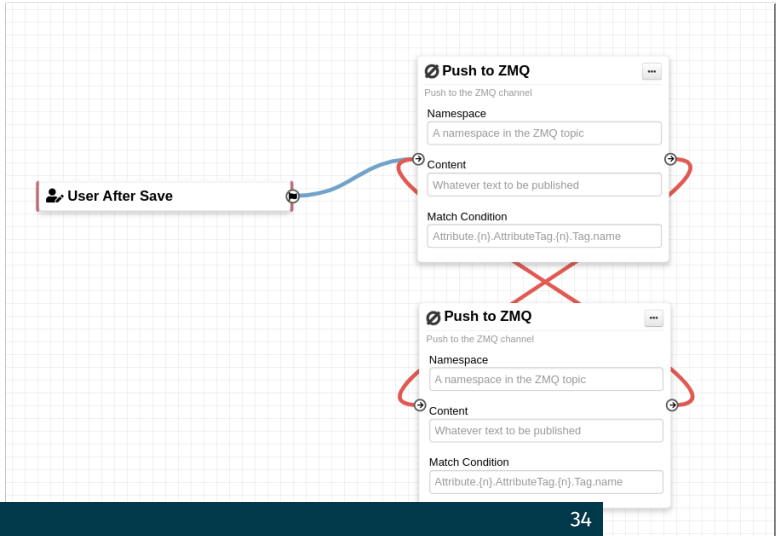
1. Prevent event publication if **tlp:red** tag
2. Send a mail to `admin@admin.test` about potential data leak
3. Otherwise, send a notification on **Mattermost**, **MS Teams**, **Telegram**, ...

Considerations when working with work-flows

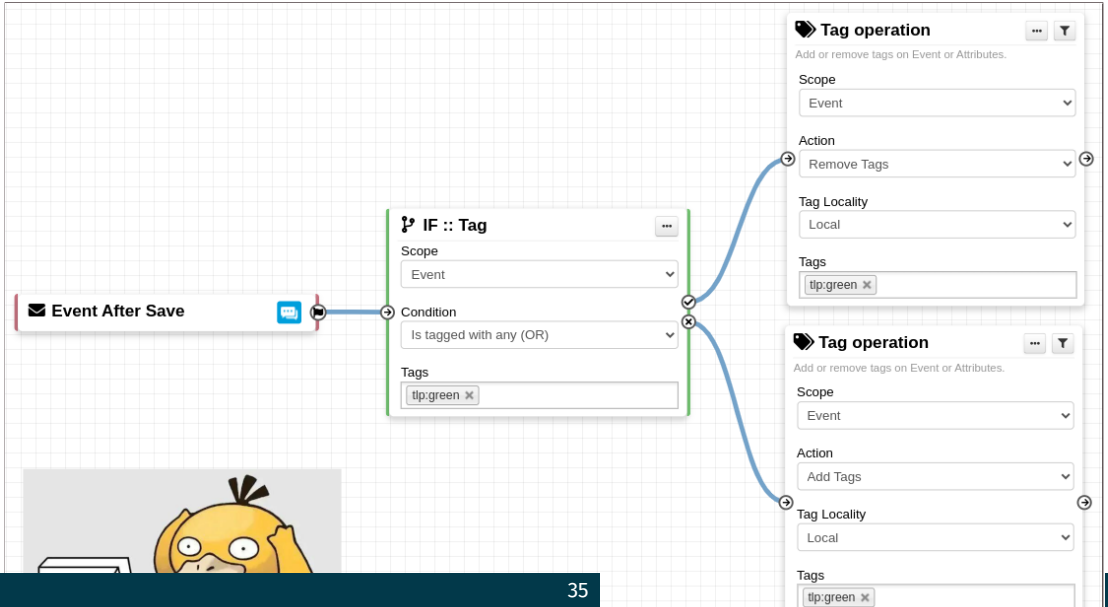
Objective: Overview of some common pitfalls

Working with the editor - Operations not allowed

Execution loop are not authorized

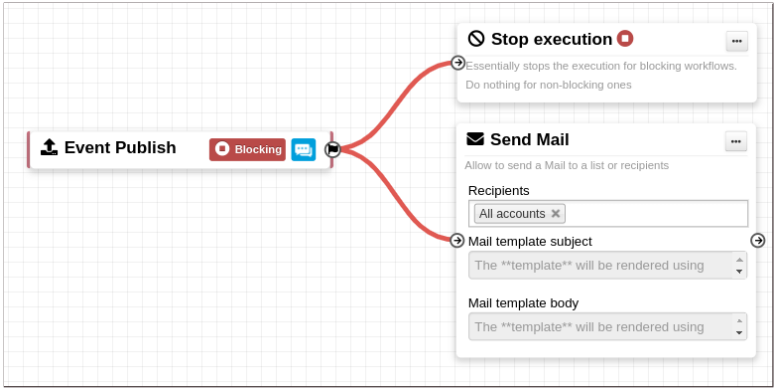


Recursive workflows



Working with the editor - Operations not allowed

Multiple connections from the same output



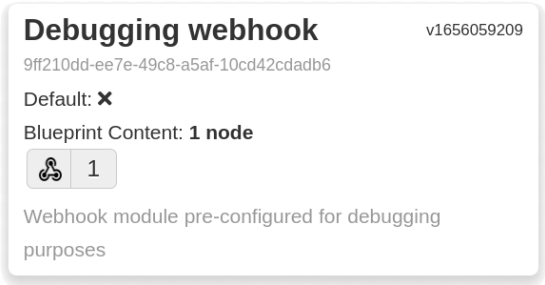
- Execution order not guaranted
- Confusing for users

Advanced usage

Objective: Overview of Blueprints, Data format and Filtering

Workflow blueprints

1. Blueprints allow to **re-use parts** of a workflow in another one
2. Blueprints can be saved, exported and **shared**



Blueprints sources:

1. Created or imported by users
2. From the `MISP/misp-workflow-blueprints` repository[3]

[3] https://github.com/MISP/misp-workflow-blueprints

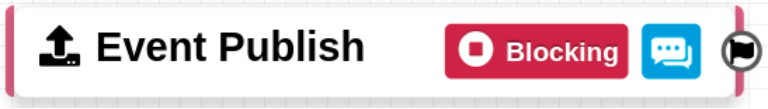
Workflow blueprints

Currently, 13 blueprints available in the official repo:

- Attach the `tlp:clear` tag on elements having the `tlp:white` tag
- Block actions if any attributes have the `PAP:RED` or `tlp:red` tag
- Disable `to_ids` flag for existing hash in *hashlookup*
- Set tag based on *BGP Ranking* maliciousness level
- ...

Built something cool? **Share you blueprints!**

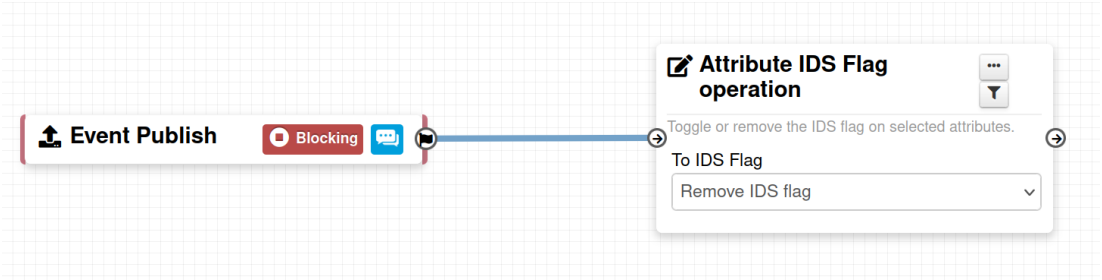
Data format in Workflows



- In most cases, the format is the **MISP Core format**
  - Attributes are **always encapsulated** in the Event or Object
  - ⚠ Event data might be partial
- And has **additional properties**
  - Additional key **`_AttributeFlattened`**
  - Additional key **`_allTags`**
  - Additional key **`inherited`** for Tags

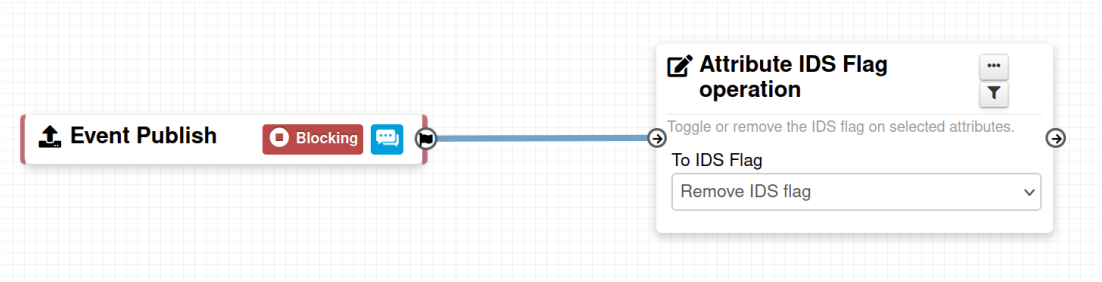
Fitlering data on which to apply a module

What happens when an Event is about to be published?



Fitlering data on which to apply a module

What happens when an Event is about to be published?



All Attributes get their `to_ids` turned off.

How could we force that action, only on Attribute of type `comment`?

→ Hash path filtering!

Hash path filtering (1)

Filtering and checking conditions using hash path expression.

```
{
    "person": { "name": "John" }
}
Hash Path 1 = 'person.name';
Result will be 'John'

[
    {'id': 123, 'name': 'fred', 'surname': 'bloggs'},
    {'id': 245, 'name': 'fred', 'surname': 'smith'},
    {'id': 356, 'name': 'joe', 'surname': 'smith'},
]
Hash Path 2 = '{n}.name';
Result will be ['fred', 'fred', 'joe']
```

Hash path filtering can be used to **filter** data **on the node** it is passed to or on the **execution path**.

Hash path filtering - Example

```
{
    "Event": {
        "uuid": ...
        "timestamp": ...
        "distribution": 1,
        "published": false,
        "Attribute": [
            {
                "type": "ip-src",
                "value": "8.8.8.8", ...
            },
            ...
        ],
        ...
    }
}
```

1. Access Event distribution
   - `Event.distribution`

Hash path filtering - Exercise (1)

```
{
    "Event": {
        "uuid": ...
        "distribution": 1,
        "published": false,
        "Attribute": [
            {
                "type": "ip-src",
                "value": "8.8.8.8", ...
            },
            ...
        ],
        ...
    }
}
```

2. Access Event published state

Hash path filtering - Exercise (1)

```
{
    "Event": {
        "uuid": ...
        "distribution": 1,
        "published": false,
        "Attribute": [
            {
                "type": "ip-src",
                "value": "8.8.8.8", ...
            },
            ...
        ],
        ...
    }
}
```

2. Access Event published state
   - `Event.published`

Hash path filtering - Exercise (2)

```
{
    "Event": {
        "uuid": ...
        "Attribute": [
            {
                "type": "ip-src",
                "value": "8.8.8.8", ...
            },
            {
                "type": "domain",
                "value": "misp-project.org", ...
            }
        ],
        ...
    }
}
```

3. Access all Attribute types
   - Hint: Use **{n}** to loop

Hash path filtering - Exercise (2)

```
{
    "Event": {
        "uuid": ...
        "Attribute": [
            {
                "type": "ip-src",
                "value": "8.8.8.8", ...
            },
            {
                "type": "domain",
                "value": "misp-project.org", ...
            }
        ],
        ...
    }
}
```

3. Access all Attribute types
   - Hint: Use **{n}** to loop
   - `Event.Attribute.{n}.type`

Hash path filtering - Exercise (3)

```
{
    "Event": {
        "Attribute": [
            {
                "value": "8.8.8.8",
                "Tag": [
                    { "name": "PAP:AMBER", ... },
                    { "name": "tlp:amber", ... },
                ], ...
            },
        ],
        ...
    }
}
```

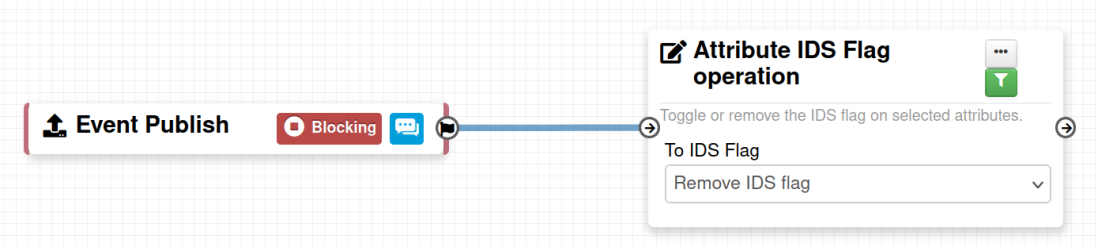
3. Access all Tag names attached to Attributes

Hash path filtering - Exercise (3)

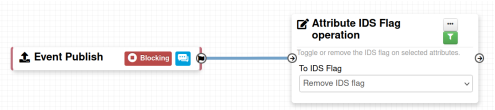
```
{
    "Event": {
        "Attribute": [
            {
                "value": "8.8.8.8",
                "Tag": [
                    { "name": "PAP:AMBER", ... },
                    { "name": "tlp:amber", ... },
                ], ...
            },
        ],
        ...
    }
}
```

3. Access all Tag names attached to Attributes
   - `Event.Attribute.{n}.Tag.{n}.name`

Fitlering data on which to apply a module



Fitlering data on which to apply a module



Node Filtering

Element selector

Event._AttributeFlattened.{n}

Value

comment

Operator

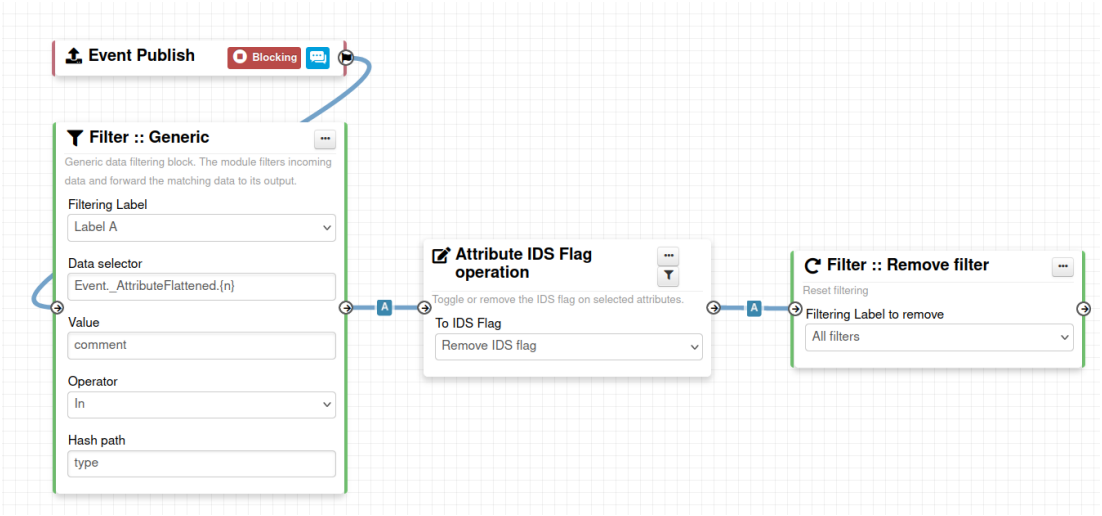
In

Hash Path

type

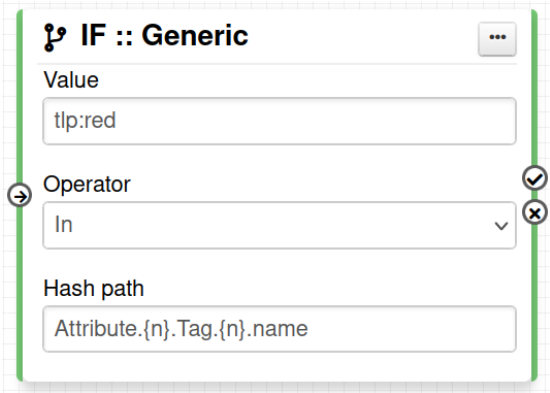
Fitlering data on which to apply on multiple modules

New feature as of **v2.4.171** allows setting filters on a path.

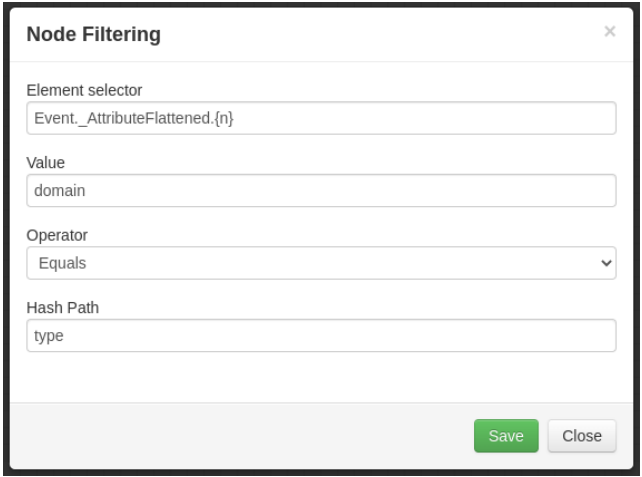


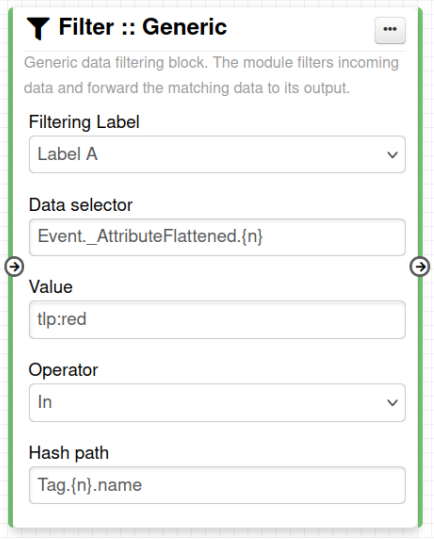
Hash path filtering (2)


```
{
    "Attribute": [
        {
            "type": "domain",
            "value": "cti-summit.org",
            "Tag": [
                {
                    "name": "tlp:red",
                    "colour": "#CC0033"
                }
            ]
        }
    ]
}
```

Hash path filtering (3)





Exercices

Debugging

Debugging Workflows: Log Entries

- Workflow execution is logged in the application logs:
  - `/admin/logs/index`
  - Note: Might be phased out as its too verbose
- Or stored on disk in the following file:
  - `/app/tmp/logs/workflow-execution.log`

Logs

« previous | next »

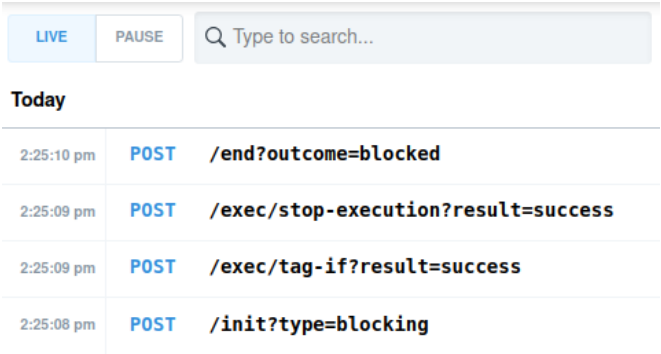
Emails | Authentication issues | MISP Update results | Setting changes | Warnings and errors

Id ↑	Email	Org	Created	Model	Model ID	Action	Title
49146	SYSTEM	SYSTEM	2022-08-01 07:34:40	Workflow	162	execute_workflow	Finished executing workflow for trigger `enrichment-before-query` (162). Outcome: success
49144	SYSTEM	SYSTEM	2022-08-01 07:34:39	Workflow	162	execute_workflow	Started executing workflow for trigger `enrichment-before-query` (162)

Debugging Workflows: Debug mode

- The

can be turned on for each workflows
- Each nodes will send data to the provided URL
  - Configure the setting: `Plugin.Workflow_debug_url`
- Result can be visualized in
  - **offline**: `tools/misp-workflows/webhook-listener.py`
  - **online**: `requestbin.com` or similar websites



Debugging modules: Stateless execution

- Test custom modules with custom input

Stateless module execution

Module parameters

Payload URL

https://localhost:8443

Content type

application/json

Data extraction path

Attribute.{n}.AttributeTag.{n}.Tag.name

Input data

Convert input data into MISP core format

Module Input Data

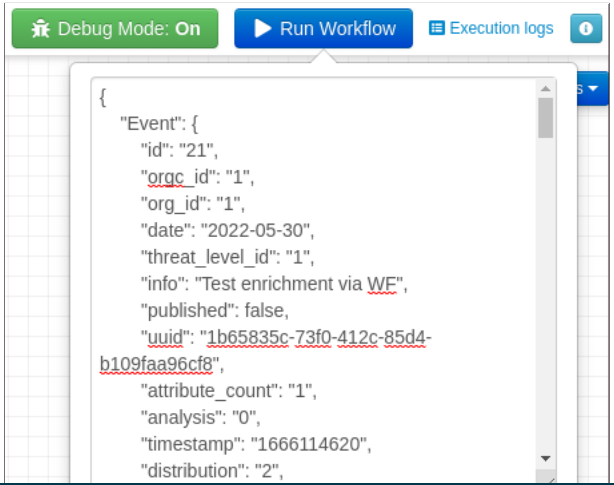
```
{
    "foo": "bar"
}
```

Execute module

Execution result: 200 [56 ms]

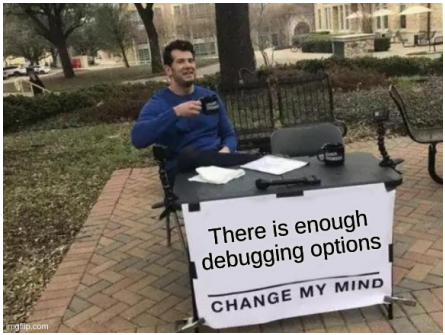
Debugging modules: Re-running workflows

- Try workflows with custom input
- Re-run workflows to ease debugging



Debugging options

- Workflow **execution and outcome**
- Module **execution and outcome**
- **Live** workflow debugging with module inspection
- **Re-running/testing** workflows with custom data
- **Stateless** module execution



Extending the system

Creating a new module in PHP

- **app/Lib/**`WorkflowModules/action/[module_name].php`
- Designed to be easilty extended
  - Helper functions
  - Module configuration as variables
  - Implement runtime logic
- Main benefits
  - Fast
  - Re-use existing functionalities
  - No need for misp-modules

Creating a new module in PHP

app > Lib > WorkflowModules > action > Module_blueprint_action_module.php > ...

```php
<?php
include_once APP . 'Model/WorkflowModules/WorkflowBaseModule.php';

class Module_blueprint_action_module extends WorkflowBaseModule
{
    public $is_blocking = false;
    public $disabled = true;
    public $id = 'blueprint-action-module';
    public $name = 'Blueprint action module';
    public $description = 'Lorem ipsum dolor, sit amet consectetur adipisicing elit.';
    public $icon = 'shapes';
    public $inputs = 1;
    public $outputs = 1;
    public $params = [];

    public function exec(array $node, WorkflowRoamingData $roamingData, array &$errors = [])
    : bool
    {
        parent::exec($node, $roamingData, $errors);
        // If $this->is_blocking == true, returning `false` will stop the execution.
        $errors[] = __('Execution stopped');
        return false;
```

Creating a new module in Python

- Similar to how other `misp-modules` are implemented
  - Helper functions
  - Module configuration as variables
  - Implement runtime logic
- Main benefits
  - Easier than PHP
  - Lots of libraries for integration

Creating a new module in Python

home > sami > git > misp-modules > misp_modules > modules > action_mod > testaction.py > ...

```
1  import json
4
5  misperrors = {'error': 'Error'}
6
7  # config fields that your code expects from the site admin
8  moduleconfig = {
9      'foo': {
10         'type': 'string',
11         'description': 'blablabla',
12         'value': 'xyz'
13     },
14     'bar': {
15         'type': 'string',
16         'value': 'meh'
17     }
18 };
19
20 # blocking modules break the exection of the chain of actions (such as publishing)
21 blocking = False
22
23 # returns either "boolean" or "data"
24 # Boolean is used to simply signal that the execution has finished.
25 # For blocking modules the actual boolean value determines whether we break execution
26 returns = 'boolean'
27
28 moduleinfo = {'version': '0.1', 'author': 'Andras Iklody',
29                 'description': 'This module is merely a test, always returning true. Triggers on event publishing.',
```

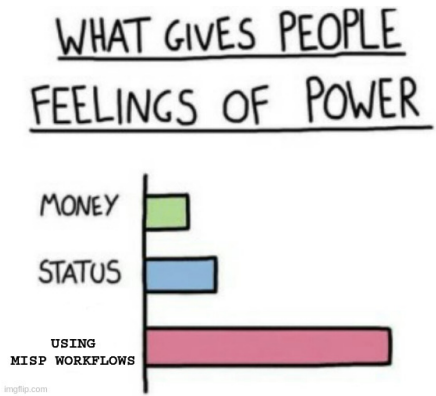
Should I migrate to MISP Workflows

I have automation in place using the API / ZMQ. Should I move to Workflows?

- I (have/am planning to create) a curation pipeline using the API, should I port them to workflows?
  - **No** in general, but WF can be used to start the curation process
- What if I want to **block** some actions
  - Put the blocking logic in the WF, the remaining outside
- Currently, workflows with **lots of node are not encouraged**
- Bottom line is **Keep it simple**

Future works

- More modules
- More modules
- More triggers
- More documentation
- Recursion prevention system

Final words

- Designed to **quickly** and **cheaply** integrate MISP in CTI pipelines
- **Beta** Feature unlikely to change. But still..
- Waiting for feedback!
  - New triggers?
  - New modules?
  - What's acheivable