AN INTRODUCTION TO WORKFLOWS IN MISP

MISP - Threat Sharing

CIRCL / Team MISP Project

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

NSPA
CONTENT OF THE PRESENTATION

- MISP Workflows fundamentals
- Getting started
- Design of the system & how it can be extended
Initial idea came during GeekWeek7.5¹

Needs:
- Prevent default MISP behaviors
- Hook specific actions to run callbacks

Use-cases:
- Prevent publication of events not meeting some criterias
- Prevent querying third-party services (e.g. virustotal) with sensitive information
- Send notifications in a chat rooms
- And much much more..

¹Workshop organized by the Canadian Cyber Center
WORKFLOW - FUNDAMENTALS
1. An action happens in MISP
2. If there is an enabled Workflow for that action, run it
3. If all went fine, MISP continue to perform the action
   ▶ The operation can potentially be cancelled by blocking modules
**TERMINOLOGY**

- **workflow**: Sequence of all operations (nodes) to be executed. Basically the whole graph.
- **execution path**: A path composed of nodes
- **trigger**: Starting point of a workflow. Triggers are called when specific actions happen in MISP
  
  - A trigger can only have one workflow and vice-versa
Typical execution process:

1. An action happens in MISP
2. The workflow associated to the trigger is ran
3. Execution result?
   - **success**: Continue the action
   - **failure** | **blocked**: Cancel the action

Example for Event publish:

1. An Event is about to be published
2. MISP executes the workflow listening to the event-publish trigger
   - **success**: Continue the publishing action
   - **failure** | **blocked**: Stop publishing and log the reason
Currently 2 types of workflows:

- **Blocking**: Completion of the action can be prevented
  - If a **blocking module** blocks the action
  - If a **blocking module** raises an exception

- **Non-blocking**: Workflow execution outcome has no impact
  - **Blocking modules** can still stop the execution
Workflows can be triggered by any users

Workflows can be triggered by actions done via the **UI** or **API**

However, the user for which the workflow executes has:
- The site-admin permission
- Is from the MISP.host_org_id

Ensures data is processed regardless of ownership and access: **no ACL**
Classes of Workflow Modules

- **Action**: Allow to executes functions, callbacks or scripts
  - Can stop execution
  - e.g. Webhook, block the execution, perform enrichments, ...

- **Logic**: Allow to redirect the execution flow.
  - IF condition, fork the blocking execution into a non-blocking one, ...

- **Blueprint**: Allow to reuse composition of modules
  - Can save subworkflows and its module’s configuration
Sources of Workflow modules

3 sources of action modules

- **Built-in default modules**
  - Part of the MISP codebase
  - `app/Model/WorkflowModules/action/[@module_name].php`

- **User-defined custom modules**
  - Written in PHP
  - Can extend existing default modules
  - Can use MISP’s built-in functionalities (restsearch, enrichment, push to zmq, ...)
  - Faster and easier to implement new complex behaviors
  - `app/Lib/WorkflowModules/action/[@module_name].php`
3 sources of action modules

- Modules from the **enrichment service**
  - Default and **custom** modules
  - From the **misp-module**
  - Written in Python
  - Can use any python libraries
  - New **misp-module** module type: action

→ Both the PHP and Python systems are **plug-and-play**
Currently 8 triggers can be hooked. 3 being **blocking**.

<table>
<thead>
<tr>
<th>Trigger name</th>
<th>Scope</th>
<th>Trigger overhead</th>
<th>Description</th>
<th>Run counter</th>
<th>Blocking Workflow</th>
<th>MISP Core format</th>
<th>Workflow ID</th>
<th>Last Update</th>
<th>Enabled</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute After Save</td>
<td>attribute</td>
<td>high</td>
<td>This trigger is called after an Attribute has been saved in the database</td>
<td>58</td>
<td>x</td>
<td>✓</td>
<td>160</td>
<td>2022-07-29 06:58:11</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Enrichment Before</td>
<td>others</td>
<td>low</td>
<td>Query This trigger is called just before a query against the enrichment service is done</td>
<td>841</td>
<td>✓</td>
<td>✓</td>
<td>162</td>
<td>2022-07-29 08:32:32</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Event After Save</td>
<td>event</td>
<td>medium</td>
<td>This trigger is called after an Event has been saved in the database</td>
<td>11</td>
<td>x</td>
<td>✓</td>
<td>175</td>
<td>2022-07-29 08:37:23</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Event Publish</td>
<td>event</td>
<td>low</td>
<td>This trigger is called just before a MISP Event starts the publishing process</td>
<td>1</td>
<td>✓</td>
<td>✓</td>
<td>180</td>
<td>2022-07-29 12:14:10</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Object After Save</td>
<td>object</td>
<td>high</td>
<td>This trigger is called after an Object has been saved in the database</td>
<td>35</td>
<td>x</td>
<td>✓</td>
<td>161</td>
<td>2022-07-28 13:50:37</td>
<td>xu</td>
<td></td>
</tr>
<tr>
<td>Post After Save</td>
<td>post</td>
<td>low</td>
<td>This trigger is called after a Post has been saved in the database</td>
<td>36</td>
<td>x</td>
<td>x</td>
<td>175</td>
<td>2022-07-28 13:59:51</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>User After Save</td>
<td>user</td>
<td>low</td>
<td>This trigger is called after a user has been saved in the database</td>
<td>55</td>
<td>x</td>
<td>x</td>
<td>150</td>
<td>2022-07-28 14:00:03</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>User Before Save</td>
<td>user</td>
<td>low</td>
<td>This trigger is called just before a user is save in the database</td>
<td>42</td>
<td>✓</td>
<td>x</td>
<td>158</td>
<td>2022-07-28 14:00:32</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
WORKFLOW - GETTING STARTED
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

4. [optional:misp-module] Turn the setting Plugin.Action_services_enable on
If you wish to use action modules from misp-module, make sure to have:

- The latest update of misp-module
  - There should be an `action_mod` module type in `misp-modules/misp_modules/modules`

- Restarted your misp-module application

```
1 # This command should show all 'action' modules
2 $ curl -s http://127.0.0.1:6666/modules | \
3    jq '.[] | select(.meta."module-type"[] | contains("action")) | \
4    {name: .name, version: .meta.version}'
```
1. Go to the list of modules
   ▶ Administration > Workflows > List Modules
   ▶ or/workflows/moduleIndex

2. Make sure default modules are loaded

3. [optional:misp-module] Make sure misp-module modules are loaded
Creating a Workflow with the Editor

1. Go to the list of triggers Administration > Workflows
2. Enable and edit a trigger from the list
3. Drag an action module from the side panel to the canvas
4. From the trigger output, drag an arrow into the action’s input (left side)
5. Execute the action that would run the trigger and observe the effect!

---

#### Table of Triggers

<table>
<thead>
<tr>
<th>Trigger name</th>
<th>Scope</th>
<th>Trigger overhead</th>
<th>Description</th>
<th>Run counter</th>
<th>Blocking Workflow</th>
<th>MSP Core format</th>
<th>Workflow ID</th>
<th>Last Update</th>
<th>Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute After Save</td>
<td>attribute</td>
<td>High</td>
<td>This trigger is called after an Attribute has been saved in the database</td>
<td>58</td>
<td>X</td>
<td>X</td>
<td>160</td>
<td>2022-07-29 05:56:11</td>
<td>True</td>
</tr>
<tr>
<td>Enrichment Before Query</td>
<td>others</td>
<td>Low</td>
<td>This trigger is called just before a query against the enrichment service is done</td>
<td>841</td>
<td>X</td>
<td>X</td>
<td>162</td>
<td>2022-07-29 06:32:32</td>
<td>True</td>
</tr>
<tr>
<td>Event After Save</td>
<td>event</td>
<td>Medium</td>
<td>This trigger is called after an Event has been saved in the database</td>
<td>11</td>
<td>X</td>
<td>X</td>
<td>175</td>
<td>2022-07-29 08:37:23</td>
<td>True</td>
</tr>
<tr>
<td>Event Publish</td>
<td>event</td>
<td>Low</td>
<td>This trigger is called just before a MSP Event starts the publishing process</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>180</td>
<td>2022-07-29 12:14:10</td>
<td>True</td>
</tr>
<tr>
<td>Object After Save</td>
<td>object</td>
<td>High</td>
<td>This trigger is called after an Object has been saved in the database</td>
<td>35</td>
<td>X</td>
<td>X</td>
<td>161</td>
<td>2022-07-28 13:59:37</td>
<td>True</td>
</tr>
<tr>
<td>Post After Save</td>
<td>post</td>
<td>Low</td>
<td>This trigger is called after a Post has been saved in the database</td>
<td>36</td>
<td>X</td>
<td>X</td>
<td>176</td>
<td>2022-07-28 13:59:51</td>
<td>True</td>
</tr>
<tr>
<td>User After Save</td>
<td>user</td>
<td>Low</td>
<td>This trigger is called after a user has been saved in the database</td>
<td>55</td>
<td>X</td>
<td>X</td>
<td>159</td>
<td>2022-07-28 14:00:03</td>
<td>True</td>
</tr>
<tr>
<td>User Before Save</td>
<td>user</td>
<td>Low</td>
<td>This trigger is called just before a user is save in the database</td>
<td>42</td>
<td>X</td>
<td>X</td>
<td>158</td>
<td>2022-07-28 14:00:32</td>
<td>True</td>
</tr>
</tbody>
</table>
Operations not allowed:

- Execution loop are not authorized

  ▶ Current caveat: If an action re-run the workflow in any way
Operations not allowed:

- Multiple connections from the same output
  - Execution order not guaranteed and confusing for users
Operations showing a warning:

- **Blocking** modules after a *concurrent tasks* module
- **Blocking** modules in a *non-blocking* workflow
WORKFLOW BLUEPRINTS

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

Blueprints origins:

1. From the "official" misp-workflow-blueprints repository
2. Created or imported by users
Select one or more modules to be saved as blueprint then click on the save blueprint button.
Hash path filtering

Some modules have the possibility to filter or check conditions using CakePHP’s path expression.

```php
$path_expression = '{n}[name=fred].id';
$users = [
    {'id': 123, 'name': 'fred', 'surname': 'bloggs'},
    {'id': 245, 'name': 'fred', 'surname': 'smith'},
    {'id': 356, 'name': 'joe', 'surname': 'smith'},
];
$ids = Hash::extract($users, $path_expression);
// => $ids will be [123, 245]
```
Some action modules accept **filtering** conditions.

E.g. the *enrich-event* module will only perform the enrichment on Attributes having a `tlp:white` Tag.
All triggers will inject data in a workflow
In some cases, there is no format (e.g. User after-save)
In others, the format is **compliant with the MISP Core format**
In addition to the RFC, the passed data has **additional properties**
  - Attributes are **always encapsulated** in the Event or Object
  - Additional key **_AttributeFlattened**
  - Additional key **_allTags**
  - Additional key **inherited** for Tags
Logic module: Concurrent Task

- Special type of logic module allowing multiple connections
- Allows breaking the execution flow into a concurrent tasks to be executed later on by a background worker
- As a side effect, blocking modules cannot cancel ongoing operations
Workflow execution is logged in the application logs:
▶ /admin/logs/index
Or stored on disk in the following file:
▶ /app/tmp/logs/workflow-execution.log
Use the webhook-listener.py tool
▶ /app/tools/misp-workflows/webhook-listener.py
The Debug Mode 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
LEARNING BY EXAMPLES
1. The Event-Publish trigger uses the MISP core format
2. The IF::Tag module checks if at least one of the Attribute has the tlp:white tag
3. If it does, the Push-to-ZMQ module will be executed
If an event has the tlp:red tag or any of the attribute has it, the publish process will be cancelled.
EXTENDING THE SYSTEM
Creating a new module in PHP

- **app/Lib/WorkflowModules/action/**[module_name]**.php
- Module configuration are defined as public variables
- The `exec` function has to be implemented.
  - If it returns **true**, execution will proceed
  - If it returns **false**
    - And the module is blocking, the execution will stop and the operation will be blocked
Module configuration are defined in the moduleinfo and moduleconfig variables.

The handler function has to be implemented.

Blocking logic is the same as other modules.