

# AN INTRODUCTION TO WORKFLOWS IN MISP

MISP - THREAT SHARING

CIRCL / TEAM MISP PROJECT

MISP PROJECT

<https://www.misp-project.org/>

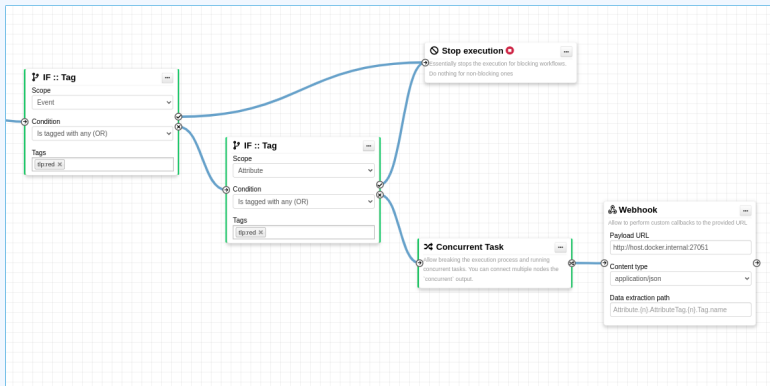
NSPA



**MISP**  
Threat Sharing

# CONTENT OF THE PRESENTATION

- MISP Workflows fundamentals
- Getting started
- Design of the system & how it can be extended





- Initial idea came during GeekWeek7.5<sup>1</sup>
- Needs:
  - ▶ Prevent default MISP behaviors
  - ▶ Hook specific actions to run callbacks
- Use-cases:
  - ▶ Prevent publication of events not meeting some criterias
  - ▶ Prevent querying thrid-party services (e.g. virustotal) with sensitive information
  - ▶ Send notifications in a chat rooms
  - ▶ And much much more..

---

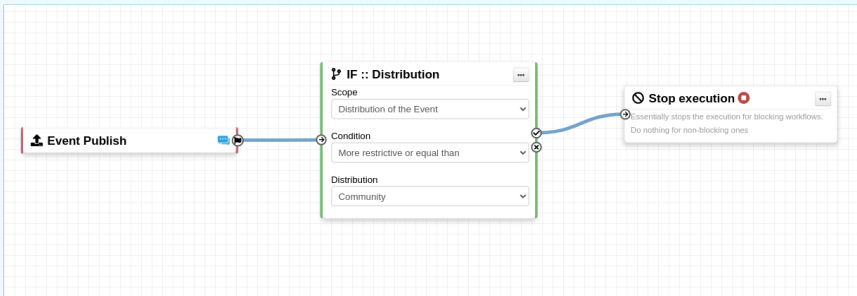
<sup>1</sup>Workshop organized by the Canadian Cyber Center

# **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**



▶ Actions

🔗 Logic

🏠 Blueprints

## 3 classes of 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

## 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**

# TRIGGERS CURRENTLY AVAILABLE

Currently 8 triggers can be hooked. 3 being **blocking**.

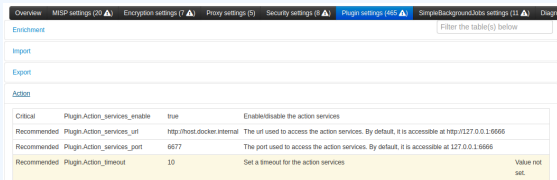
Trigger name	Scope	Trigger overhead	Description	Run counter	Blocking Workflow	MISP Core format	Workflow ID	Last Update	Enabled	Actions
 Attribute After Save	attribute	high 	This trigger is called after an Attribute has been saved in the database	58	✗	✓	160	2022-07-29 06:58:11	✓	   
 Enrichment Before Query	others	low 	This trigger is called just before a query against the enrichment service is done	841	✓	✓	162	2022-07-29 08:32:32	✓	   
 Event After Save	event	medium 	This trigger is called after an Event has been saved in the database	11	✗	✓	175	2022-07-29 08:37:23	✓	   
 Event Publish	event	low 	This trigger is called just before a MISP Event starts the publishing process	1	✓	✓	180	2022-07-29 12:14:10	✓	   
 Object After Save	object	high 	This trigger is called after an Object has been saved in the database	35	✗	✓	161	2022-07-28 13:59:37	✗	   
 Post After Save	post	low 	This trigger is called after a Post has been saved in the database	36	✗	✗	176	2022-07-28 13:59:51	✓	   
 User After Save	user	low 	This trigger is called after a user has been saved in the database	55	✗	✗	159	2022-07-28 14:00:03	✓	   
 User Before Save	user	low 	This trigger is called just before a user is save in the database	42	✓	✗	158	2022-07-28 14:00:32	✓	   

# **WORKFLOW - GETTING STARTED**

# GETTING STARTED WITH WORKFLOWS (1)

Review MISP settings:

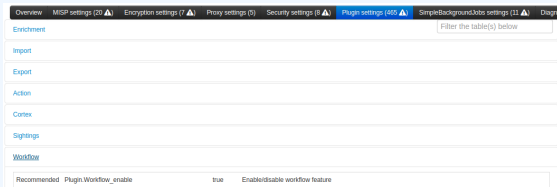
1. Make sure `MISP.background_jobs` is turned on
2. Make sure workers are up-and-running and healthy
3. Turn the setting `Plugin.Workflow_enable` on



The screenshot shows the 'Plugin settings' tab in the MISP configuration interface. The 'Action' section is expanded, displaying a table of settings for action services.

Level	Setting Name	Value	Description	Default
Critical	<code>Plugin.Action_services_enable</code>	true	Enable/disable the action services	
Recommended	<code>Plugin.Action_services_url</code>	<code>http://host.docker.internal</code>	The url used to access the action services. By default, it is accessible at <code>http://127.0.0.1:6666</code>	
Recommended	<code>Plugin.Action_services_port</code>	6677	The port used to access the action services. By default, it is accessible at <code>127.0.0.1:6666</code>	
Recommended	<code>Plugin.Action_timeout</code>	10	Set a timeout for the action services	Value not set

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



The screenshot shows the 'Plugin settings' tab in the MISP configuration interface. The 'Workflow' section is expanded, displaying a table of settings for workflow features.

Level	Setting Name	Value	Description
Recommended	<code>Plugin.Workflow_enable</code>	true	Enable/disable workflow feature

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}'
```

---

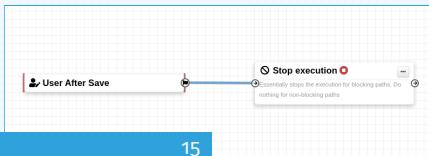


1. Go to the list of modules
  - ▶ Administration > Workflows > List Modules
  - ▶ or /workflows/moduleIndex
2. Make sure **default** modules are loaded
3. [optional:misp-module] Make sure **misp-module** modules are loaded

# 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!

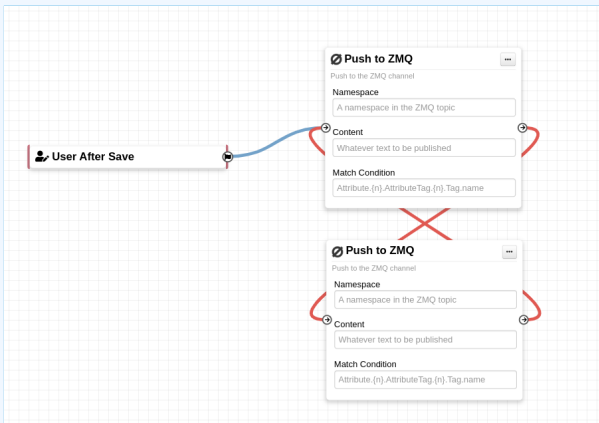
Trigger name	Scope	Trigger overhead	Description	Run counter	Blocking Workflow	MISP Core format	Workflow ID	Last Update	Enabled	Actions
Attribute After Save	attribute	High	This trigger is called after an Attribute has been saved in the database	58	X	✓	160	2022-07-29 06:58:11	✓	⊞ ⊞ ⊞ ⊞
Enrichment Before Query	others	Low	This trigger is called just before a query against the enrichment service is done	841	✓	✓	162	2022-07-29 08:32:32	✓	⊞ ⊞ ⊞ ⊞
Event After Save	event	Medium	This trigger is called after an Event has been saved in the database	11	X	✓	175	2022-07-29 08:37:23	✓	⊞ ⊞ ⊞ ⊞
Event Publish	event	Low	This trigger is called just before a MISP Event starts the publishing process	1	✓	✓	180	2022-07-29 12:14:10	✓	⊞ ⊞ ⊞ ⊞
Object After Save	object	High	This trigger is called after an Object has been saved in the database	35	X	✓	161	2022-07-28 13:59:37	X	▶ ⊞ ⊞ ⊞
Post After Save	post	Low	This trigger is called after a Post has been saved in the database	36	X	X	178	2022-07-28 13:59:51	✓	⊞ ⊞ ⊞ ⊞
User After Save	user	Low	This trigger is called after a user has been saved in the database	55	X	X	158	2022-07-28 14:00:03	✓	⊞ ⊞ ⊞ ⊞
User Before Save	user	Low	This trigger is called just before a user is save in the database	42	✓	X	158	2022-07-28 14:00:32	✓	⊞ ⊞ ⊞ ⊞



# WORKING WITH THE EDITOR

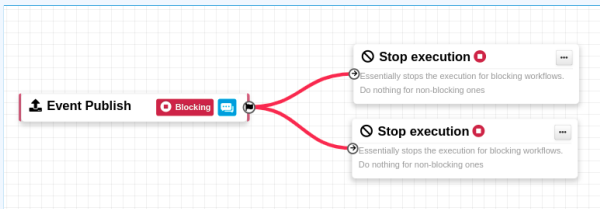
Operations not allowed:

- Execution loop are not authorized
  - ▶ Current caveat: If an action re-run the workflow in any way



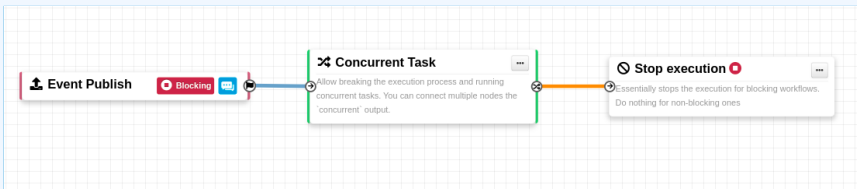
## Operations not allowed:

- 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




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

**Debugging webhook** v1656059209

9ff210dd-ee7e-49c8-a5af-10cd42cdadb6

Default: ✕

Blueprint Content: **1 node**

 1

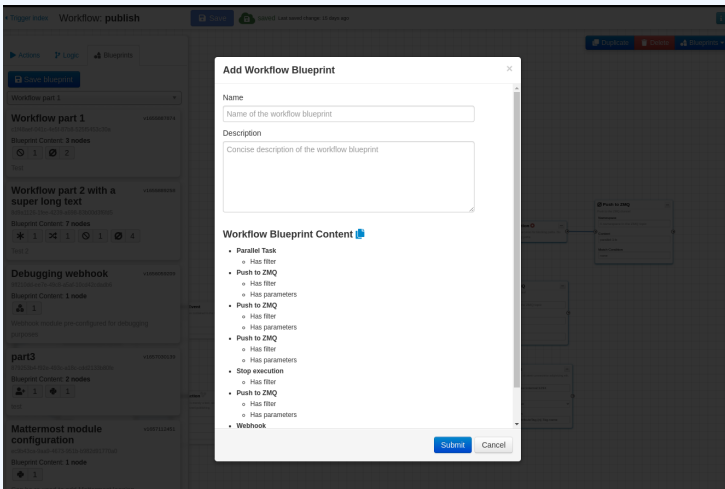
Webhook module pre-configured for debugging purposes

## Blueprints origins:

1. From the "official" `misp-workflow-blueprints` repository
2. Created or imported by users

# WORKFLOW BLUEPRINTS: CREATE

Select one or more modules to be saved as blueprint then click on the save blueprint button



The screenshot shows a workflow editor interface with a modal dialog titled "Add Workflow Blueprint". The dialog has the following fields and content:

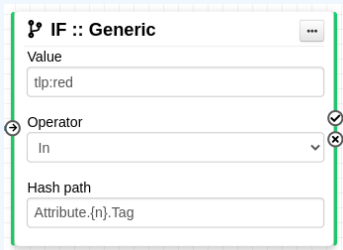
- Name:** A text input field with the placeholder "Name of the workflow blueprint".
- Description:** A text area with the placeholder "Concise description of the workflow blueprint".
- Workflow Blueprint Content:** A list of selected workflow modules with their properties:
  - **Parallel Task**
    - Has filter
  - **Push to ZMQ**
    - Has filter
    - Has parameters
  - **Push to ZMQ**
    - Has filter
    - Has parameters
  - **Push to ZMQ**
    - Has filter
    - Has parameters
  - **Stop execution**
    - Has filter
  - **Push to ZMQ**
    - Has filter
    - Has parameters
  - **Webhook**

The background shows a workflow graph with several nodes, including "Workflow part 1", "Workflow part 2 with a super long text", "Debugging webhook", "part3", and "Mattermost module configuration".

# HASH PATH FILTERING

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

```
1 $path_expression = '{n}[name=fred].id';  
2 $users = [  
3     {'id': 123, 'name': 'fred', 'surname': 'bloggs'},  
4     {'id': 245, 'name': 'fred', 'surname': 'smith'},  
5     {'id': 356, 'name': 'joe', 'surname': 'smith'},  
6 ];  
7 $ids = Hash::extract($users, $path_expression);  
8 // => $ids will be [123, 245]
```





# MODULE FILTERING

- Some action modules accept **filtering** conditions
- E.g. the `enrich-event` module will only perform the enrichment on Attributes having a `tlp:white` Tag

### Module Filtering ×

Element selector

Value

Operator

▼

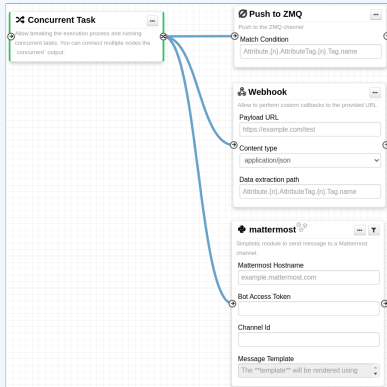
Hash Path



- 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



# DEBUGGING WORKFLOWS: LOG ENTRIES


- 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`

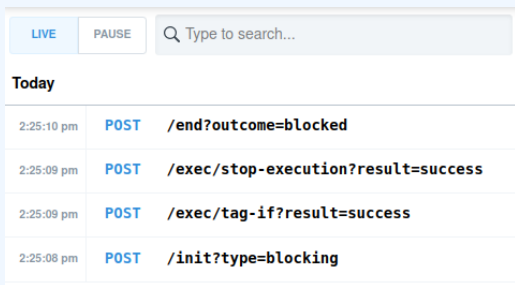
## Logs

« 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

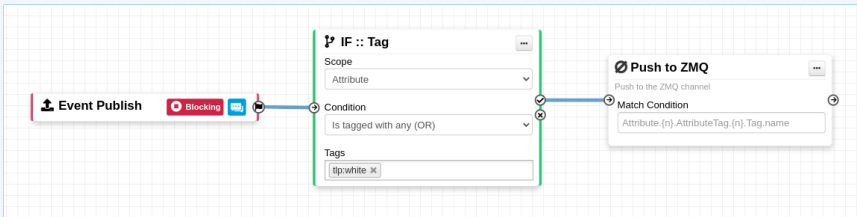
- The  can be turned on for each workflows
- Each nodes will send data to the provided URL
  - ▶ Configure the setting: `Plugin.Workflow_debug_url`
- Result can be visualized in
  - ▶ **offline:** `tools/misp-workflows/webhook-listener.py`
  - ▶ **online:** `requestbin.com` or similar websites



LIVE		PAUSE	🔍 Type to search...
<b>Today</b>			
2:25:10 pm	POST	/end?outcome=blocked	
2:25:09 pm	POST	/exec/stop-execution?result=success	
2:25:09 pm	POST	/exec/tag-if?result=success	
2:25:08 pm	POST	/init?type=blocking	

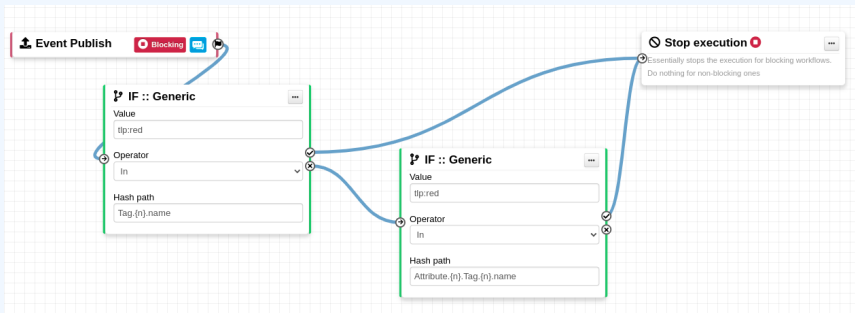
# LEARNING BY EXAMPLES

# WORKFLOW EXAMPLE 1



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

# WORKFLOW EXAMPLE 2



- 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_blueprint_action_module.php > ...
1 <?php
2 include_once APP . 'Model/WorkflowModules/WorkflowBaseModule.php';
3
4 class Module_blueprint_action_module extends WorkflowBaseModule
5 {
6     public $is_blocking = false;
7     public $disabled = true;
8     public $id = 'blueprint-action-module';
9     public $name = 'Blueprint action module';
10    public $description = 'Lorem ipsum dolor, sit amet consectetur adipisicing elit.';
11    public $icon = 'shapes';
12    public $inputs = 1;
13    public $outputs = 1;
14    public $params = [];
15
16    public function exec(array $node, WorkflowRoamingData $roamingData, array &$errors = []): bool
17    {
18        parent::exec($node, $roamingData, $errors);
19        // If $this->is_blocking == true, returning 'false' will stop the execution.
20        $errors[] = __('Execution stopped');
21        return false;
22    }
23 }
```

- `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

# CREATING A NEW MODULE IN PYTHON

```
home > sami > git > misp-modules > misp_modules > modules > action_mod > testaction.py > ...
1 | import json-
2 |
3 |
4 | misperrors = {'error': 'Error'}
5 |
6 | # config fields that your code expects from the site admin
7 | moduleconfig = {
8 |     'foo': {
9 |         'type': 'string',
10 |         'description': 'blablaba',
11 |         'value': 'xyz'
12 |     },
13 |     'bar': {
14 |         'type': 'string',
15 |         'value': 'neh'
16 |     }
17 | };
18 |
19 | # blocking modules break the execution of the chain of actions (such as publishing)
20 | blocking = False
21 |
22 | # returns either "boolean" or "data"
23 | # Boolean is used to simply signal that the execution has finished.
24 | # For blocking modules the actual boolean value determines whether we break execution
25 | returns = 'boolean'
26 |
27 | moduleinfo = {'version': '0.1', 'author': 'Andras Iklody',
28 |              'description': 'This module is merely a test, always returning true. Triggers on event publishing.',
29 |              'module-type': ['action']}
30 |
31 |
32 | def handler(q=False):
33 |     if q is False:
34 |         return False
35 |     result = json.loads(q) # nope
36 |     output = result # Insert your magic here!
37 |     r = {"data": output}
38 |     return r
39 |
40 |
41 | def introspection():-
```

- 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