



# Incident Response Platform Integrations

# QRadar Function V1.0.0

Release Date: July 2018

Resilient Functions simplify development of integrations by wrapping each activity into an individual workflow component. These components can be easily installed, then used and combined in Resilient workflows. The Resilient platform sends data to the function component that performs an activity then returns the results to the workflow. The results can be acted upon by scripts, rules, and workflow decision points to dynamically orchestrate the security incident response activities.

This guide describes the QRadar Function.

Overview

The QRadar integration with the Resilient platform package provides the following:

* Search function to perform a QRadar Ariel query
* Search function to query an item in a QRadar reference set
* Add function to insert a new item in a QRadar reference set
* Delete function to remove an item from a QRadar reference set

With the above functions, this package includes example workflows that demonstrate how to call the functions, rules that start the example workflows, and custom data tables updated by the example workflows.

Installation

Before installing, verify that your environment meets the following prerequisites:

* Resilient platform is version 30 or later.
* You have a Resilient account to use for the integrations. This can be any account that has the permission to view and modify administrator and customization settings, and read and update incidents. You need to know the account username and password.
* You have access to the command line of the Resilient appliance, which hosts the Resilient platform; or to a separate integration server where you will deploy and run the functions code. If using a separate integration server, you must install Python version 2.7.10 or later, or version 3.6 or later, and “pip”. (The Resilient appliance is preconfigured with a suitable version of Python.)
* QRadar version 7.2.8 or later

Install the Python components

The functions package contains Python components that will be called by the Resilient platform to execute the functions during your workflows. These components run in the ‘resilient-circuits’ integration framework.

The package also includes Resilient customizations that will be imported into the platform later.

Complete the following steps to install the Python components:

1. Ensure that the environment is up-to-date, as follows:

sudo pip install --upgrade pip

sudo pip install --upgrade setuptools

sudo pip install --upgrade resilient-circuits

1. Run the following command to install the package:

sudo pip install fn\_qradar\_integration-<*version*>.tar.gz

Configure the Python components

The Resilient Circuits components run as an unprivileged user, typically named `integration`. If you do not already have an `integration` user configured on your appliance, create it now.

Complete the following steps to configure and run the integration:

1. Using sudo, switch to the integration user, as follows:

sudo su - integration

1. Use one of the following commands to create or update the resilient-circuits configuration file. Use –c for new environments or –u for existing environments.

resilient-circuits config -c

or

resilient-circuits config -u

1. Edit the resilient-circuits configuration file, as follows:
   1. In the [resilient] section, ensure that you provide all the information required to connect to the Resilient platform.
   2. In the [fn\_qradar\_integration] section, edit the settings as follows:

host=<*qradar url*>

port=<8089 or the customized port>

username=<*qradar access user*>

qradarpassword=<*qradar access password, key-ring protection recommended*>

verify\_cert=[true|false]

Use ‘false’ for self-signed certificates.

Deploy customizations to the Resilient platform

The package contains four example functions, five example workflows that access the functions, and five rules that trigger the workflows, along with custom fields and a data table for input and output parameters.

1. Use the following command to deploy these customizations to the Resilient platform:

resilient-circuits customize

1. Respond to the prompts to deploy functions, message destinations, workflows and rules.

Run the integration framework

To test the integration package before running it in a production environment, you must run the integration manually with the following command:

resilient-circuits run

The resilient-circuits command starts, loads its components, and continues to run until interrupted. If it stops immediately with an error message, check your configuration values and retry.

Configure Resilient Circuits for restart

For normal operation, Resilient Circuits must run continuously. The recommend way to do this is to configure it to automatically run at startup. On a Red Hat appliance, this is done using a systemd unit file such as the one below. You may need to change the paths to your working directory and app.config.

1. The unit file must be named resilient\_circuits.service To create the file, enter the following command:

sudo vi /etc/systemd/system/resilient\_circuits.service

1. Add the following contents to the file and change as necessary:

[Unit]  
Description=Resilient-Circuits Service  
After=resilient.service  
Requires=resilient.service

[Service]  
Type=simple  
User=integration  
WorkingDirectory=/home/integration  
ExecStart=/usr/local/bin/resilient-circuits run  
Restart=always  
TimeoutSec=10  
Environment=APP\_CONFIG\_FILE=/home/integration/.resilient/app.config  
Environment=APP\_LOCK\_FILE=/home/integration/.resilient/resilient\_circuits.lock

[Install]  
WantedBy=multi-user.target

1. Ensure that the service unit file is correctly permissioned, as follows:

sudo chmod 664 /etc/systemd/system/resilient\_circuits.service

1. Use the systemctl command to manually start, stop, restart and return status on the service:

sudo systemctl resilient\_circuits [start|stop|restart|status]

You can view log files for systemd and the resilient-circuits service using the journalctl command, as follows:

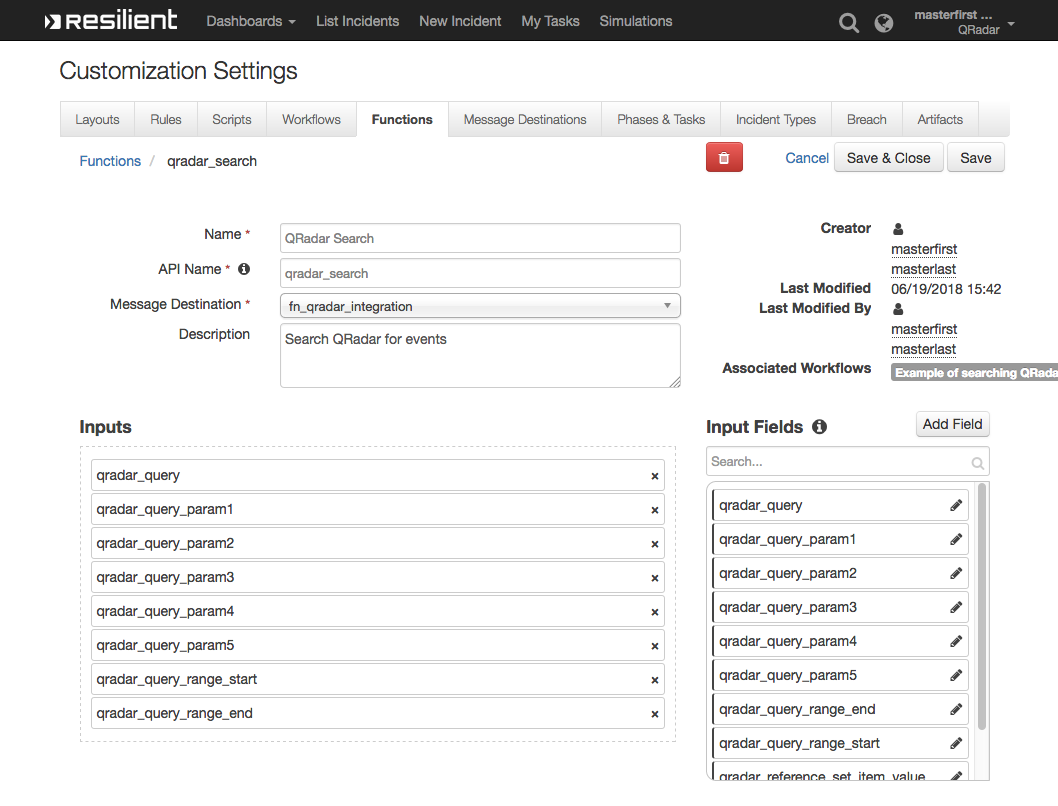
sudo journalctl -u resilient\_circuits --since "2 hours ago"

Function Descriptions

Once the function package deploys, you can verify in the Customization Settings section of the Resilient platform that the following QRadar specific functions, workflows, rules, and message destination are available by clicking their respective tabs. The data table and custom fields need to be added to your custom layout as per your design.

QRadar Search

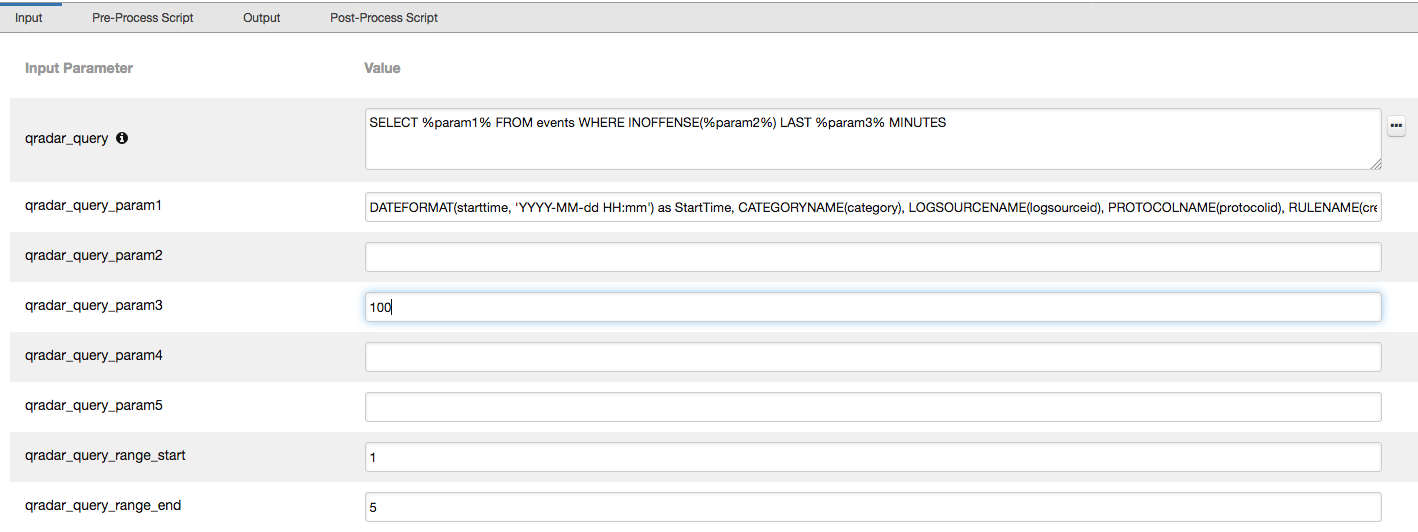
This sample function performs an Ariel query to fetch data from the QRadar server.



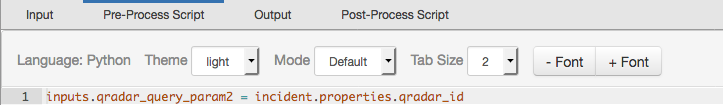
This function takes the following parameters:

* qradar\_query: Query to perform. It contains demo template queries you can select from within the calling workflow. The demo queries contain parameters that are replaced by the qradar\_query\_param[n] below. For example, one template query is: *SELECT %param1% FROM events WHERE INOFFENSE(%param2%) LAST %param3% MINUTES*. Users can set values for qradar\_query\_param1, qradar\_query\_param2, and qradar\_query\_param3 in the workflow.
* qradar\_query\_param[n]: Optional. Parameters used in the query.
* qradar\_query\_range\_start: Optional. An integer specifying QRadar return start range.
* qradar\_query\_range\_end: Optional. An integer specifying QRadar return end range.

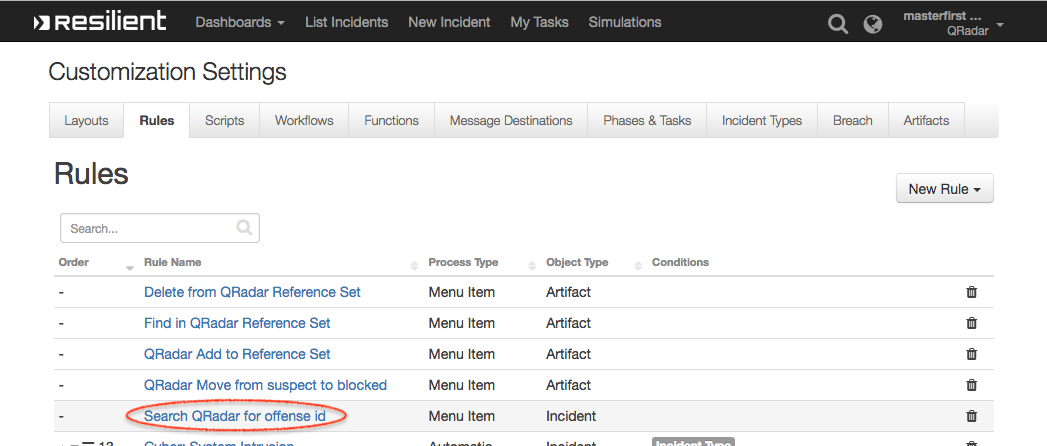
The workflow (object type = Incident), “Example of searching QRadar events using offense id”, sets the function’s input fields and runs the function. The Input tab maps the function’s input fields.



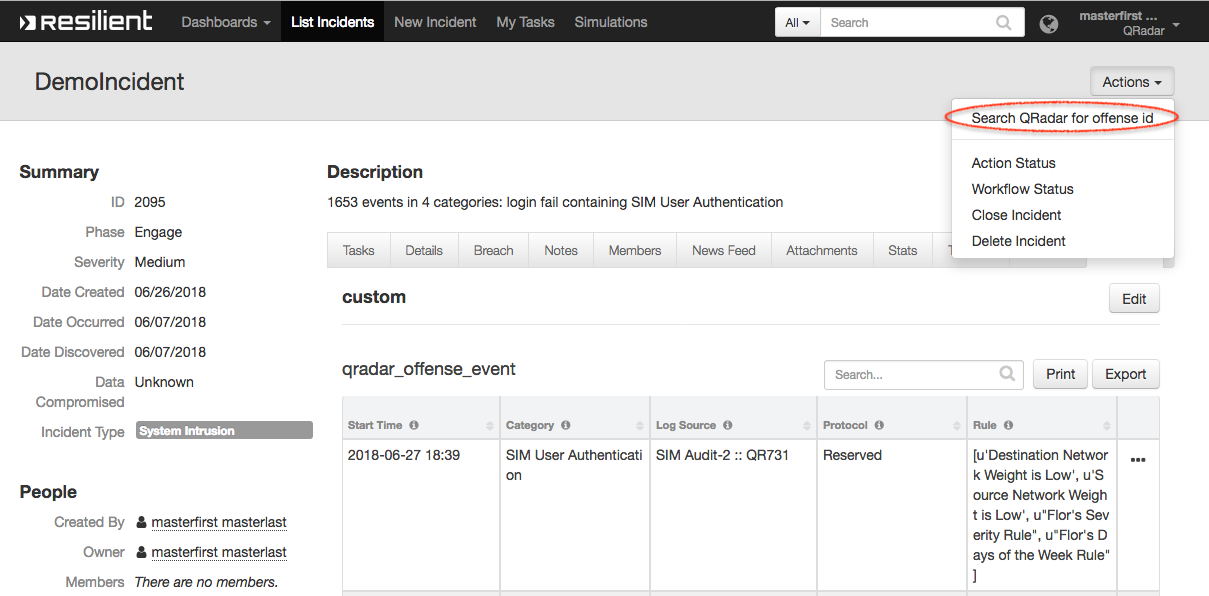
In the Pre-Process Script, the input field, “qradar\_query\_param2” is set to the incident’s custom field, “qradar\_id”.



The example rule, “Search QRadar for offense id”, is a menu item rule for an Incident. The user can select this menu item to initiate the workflow.

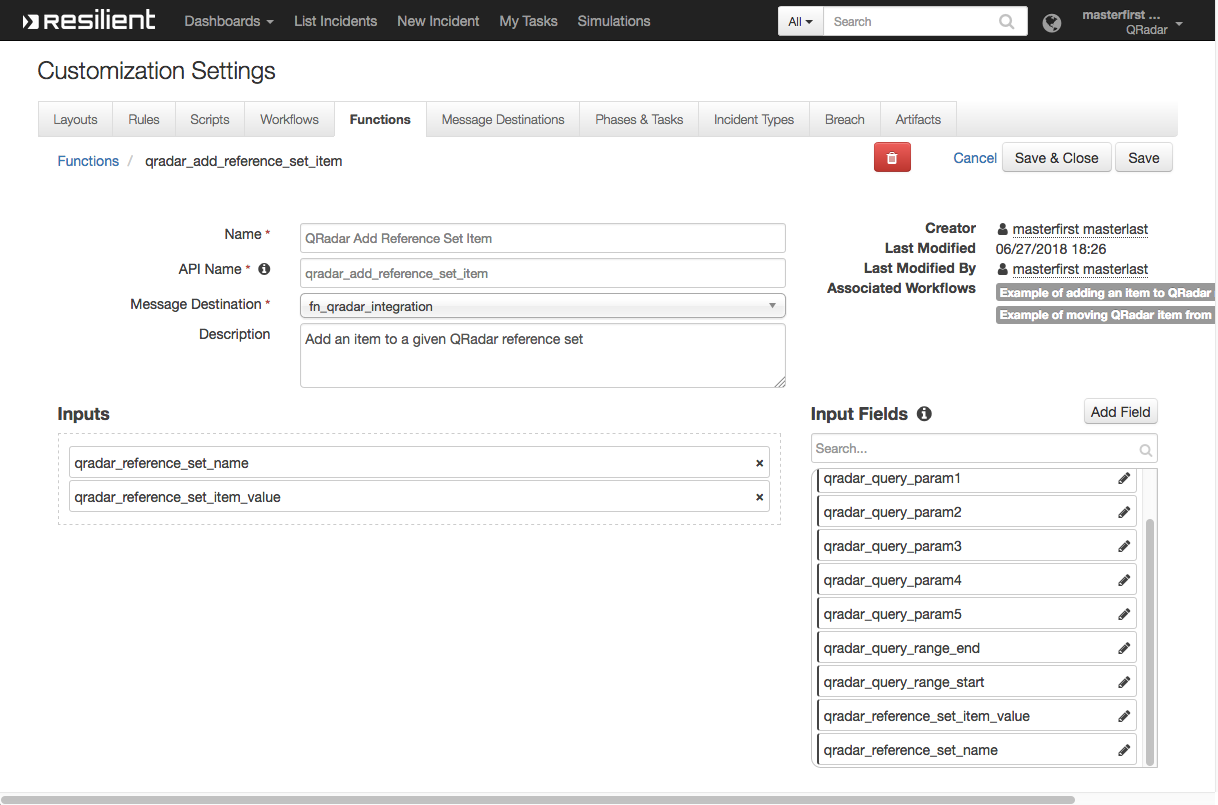


This workflow can work together with the Resilient QRadar App. An incident escalated from a QRadar offense stores the offense ID. This workflow can make use of that offense ID to perform an Ariel query and update a custom data table, which is also included in the package.

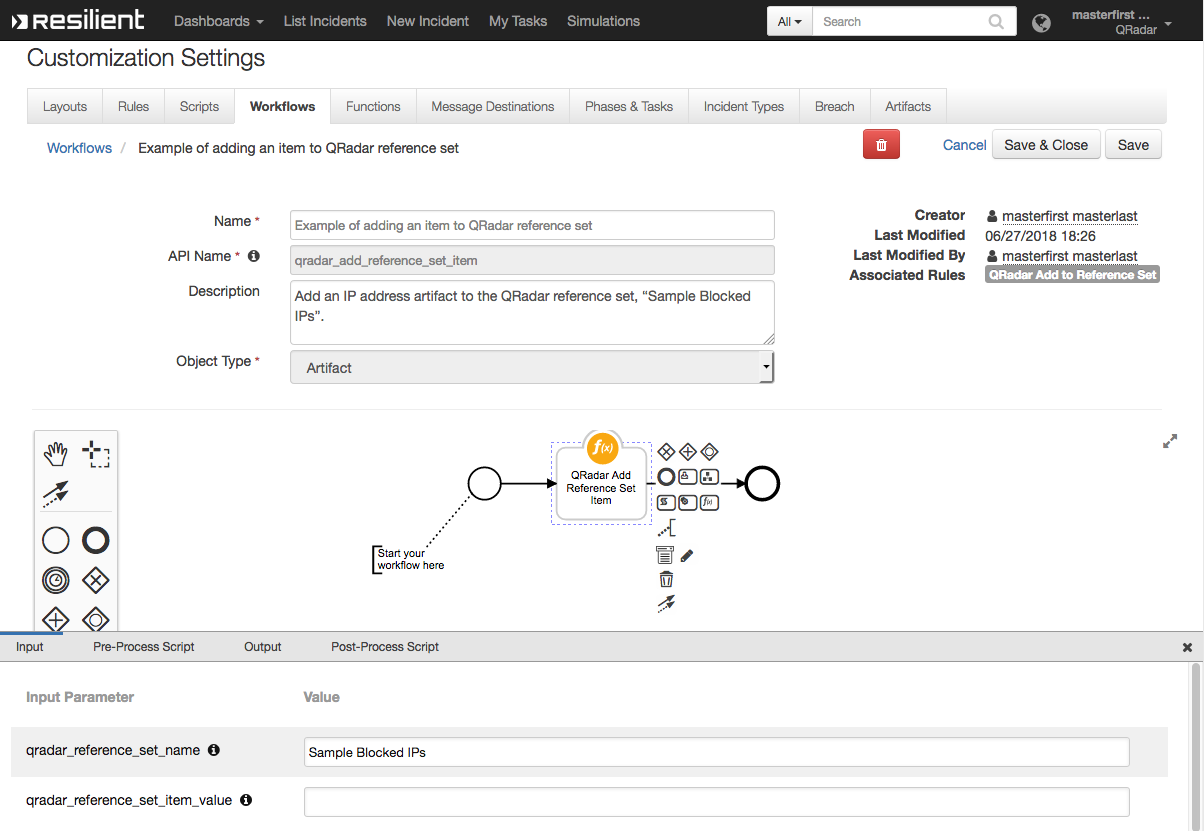


QRadar Add Reference Set Item

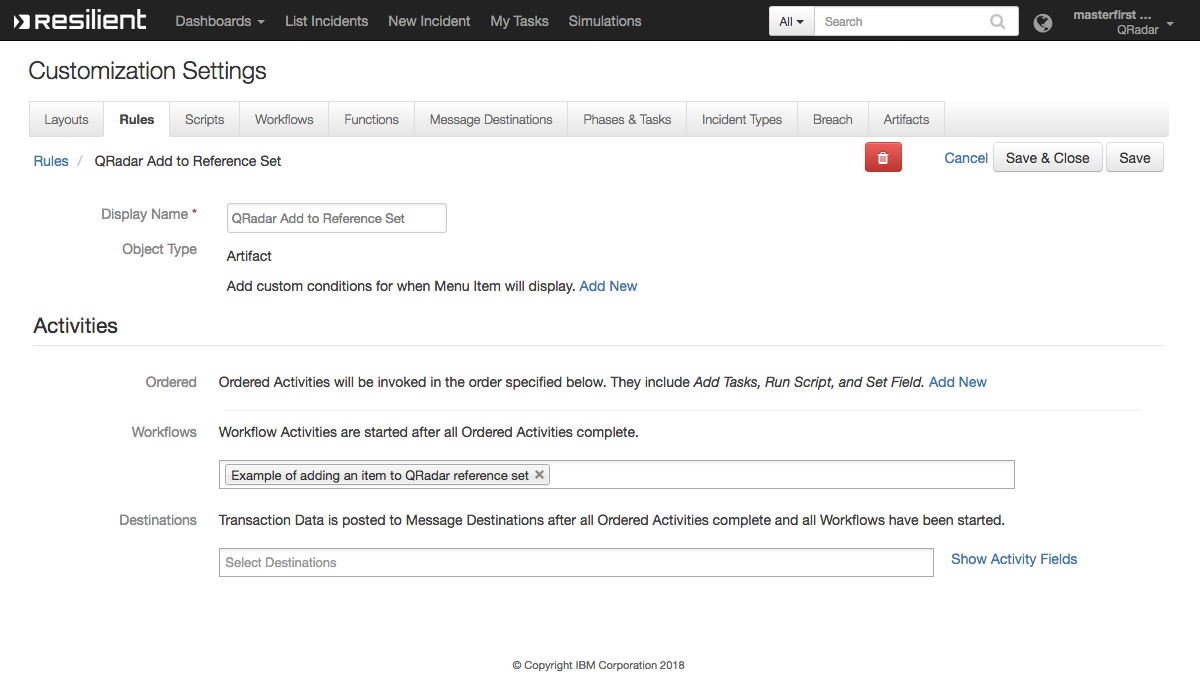
This function adds a new item to an existing QRadar reference set. It uses two input parameters: qradar\_reference\_set\_name is the name of an existing reference set in QRadar, and qradar\_reference\_set\_item\_value is the value to be added. The input is populated by the workflow, “Example of adding an item to QRadar reference set”.



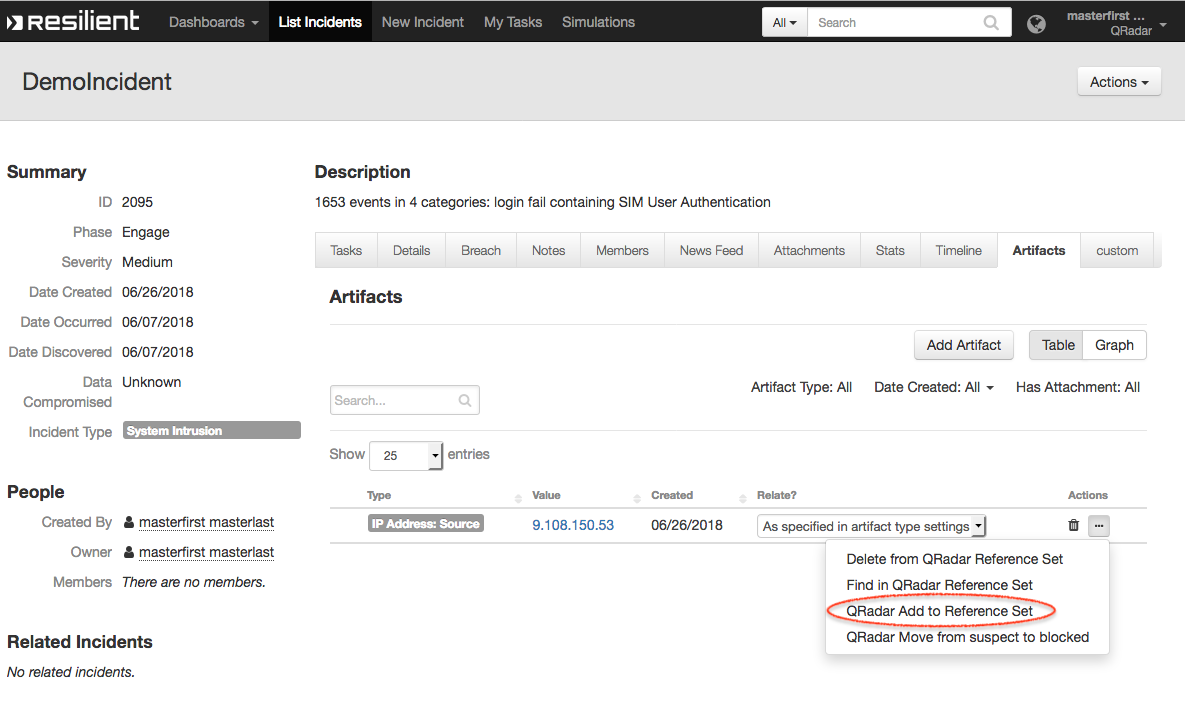
The workflow, “Example of adding an item to QRadar reference set”, sets the function’s input fields: “qradar\_reference\_set\_name” is mapped to “Sample Blocked IPs”, and qradar\_reference\_set\_item\_value is mapped to the artifact value, and then runs the function. The workflow is initiated by the rule, “QRadar Add to Reference Set”.



The example rule, “QRadar Add to Reference Set”, is a menu item rule for an artifact.



The user can select this action in the menu to initiate the workflow.



QRadar Delete Reference Set Item

This function deletes an item from an existing QRadar reference set. It has two input fields: “qradar\_reference\_set\_name” and “qradar\_reference\_set\_item\_value”. The function is called by the workflow, “Example of deleting QRadar reference set item”.

The workflow, “Example of deleting QRadar reference set item”, sets the function’s input fields: “qradar\_reference\_set\_name” is mapped to “Sample Suspect IPs”, and “qradar\_reference\_set\_item\_value” is mapped to the artifact value, and then runs the function. The workflow is initiated by the rule, “Delete from QRadar Reference Set”.

The rule, “Delete from QRadar Reference Set” is a menu item rule for artifacts. The user can select this menu item to initiate the workflow.

QRadar Find Reference Set Item

This function looks for an item in an existing QRadar reference set. It has two input fields: “qradar\_reference\_set\_name” and “qradar\_reference\_set\_item\_value”. The function is called by the workflow, “Example of finding an item from a QRadar reference set”.

The workflow, “Example of finding an item from a QRadar reference set”, sets the function’s input fields: “qradar\_reference\_set\_name” is mapped to “Sample Blocked IPs”, and “qradar\_reference\_set\_item\_value” is mapped to the artifact value. After running the function, the workflow adds a note to the corresponding incident.

The rule, “Find in QRadar Reference Set” is a menu item rule for artifacts. The user can select this action from the menu to initiate the workflow.

Troubleshooting

There are several ways to verify the successful operation of a function.

* Resilient Action Status

When viewing an incident, use the Actions menu to view Action Status. By default, pending and errors are displayed. Modify the filter for actions to also show Complete actions. Clicking on an action displays additional information on the progress made or what error occurred.

* Resilient Scripting Log

A separate log file is available to review scripting errors. This is useful when issues occur in the pre-processing or post-processing scripts. The default location for this log file is: /var/log/resilient-scripting/resilient-scripting.log.

* Resilient Logs

By default, Resilient logs are retained at /usr/share/co3/logs. The client.log may contain additional information regarding the execution of functions.

* Resilient-Circuits

The log is controlled in the .resilient/app.config file under the section [resilient] and the property logdir. The default file name is app.log. Each function will create progress information. Failures will show up as errors and may contain python trace statements.

Support

For additional support, contact [support@resilientsystems.com](mailto:support@resilientsystems.com).

Including relevant information from the log files will help us resolve your issue.