Splunk Search Actions

This component supports the actions that update incidents with the results of Splunk searches

# Component Overview

The project runs as a standalone Python application. When this application starts, it connects to the Resilient platform and starts listening for actions. A manual or automatic action sends a message to the application. The application will look for a Query definition file with the same name as the action that was triggered. Once the query execution completes, the incident will be updated with the results as specified in the Query definition’s mapping rules.

## Requirements

This component has been testing with Resilient IRP version 26. It should be run with Python2.7.

## Installation

This component requires Resilient’s co3 and resilient-circuits packages. These are available from Resilient’s Github respository.

<https://github.com/Co3Systems/co3-api/tree/master/python>

It also requires Splunk’s developer API which can be installed from PyPi with “pip install splunk-sdk”.

## Basic Configuration

First, edit the ‘app.config’ file and supply values needed to connect to your Resilient server. You will need the hostname, port, user credentials that the application will use to authenticate, and the name of the user’s Organization.

For the user credentials, we recommend creating an “API user account” specifically for integration purposes, distinct from normal users. Note that any edits that this integration makes to incidents in Resilient (adding attachments, etc.) will be attributed to this user.

In the “[splunk]” section, populate the following required values:

|  |  |  |
| --- | --- | --- |
| Key | Description | Sample Value |
| queue | The message destination in Resilient to listen on | splunk |
| host | Hostname or IP for Splunk | https://splunk\_hostname |
| port | Port for Splunk API | 8089 |
| user | Username for Splunk | admin |
| password | Password for Splunk user | MyPassw0rd |
| query\_definitions\_dir | Directory where the query definition files are stored | /home/resilient/queries |
| query\_timeout | How long should a search run before we give up? Set this value to 0 to never abandon the Query. In seconds. | 600 |
| polling\_interval | How often should we check the query status? In seconds. | 5 |

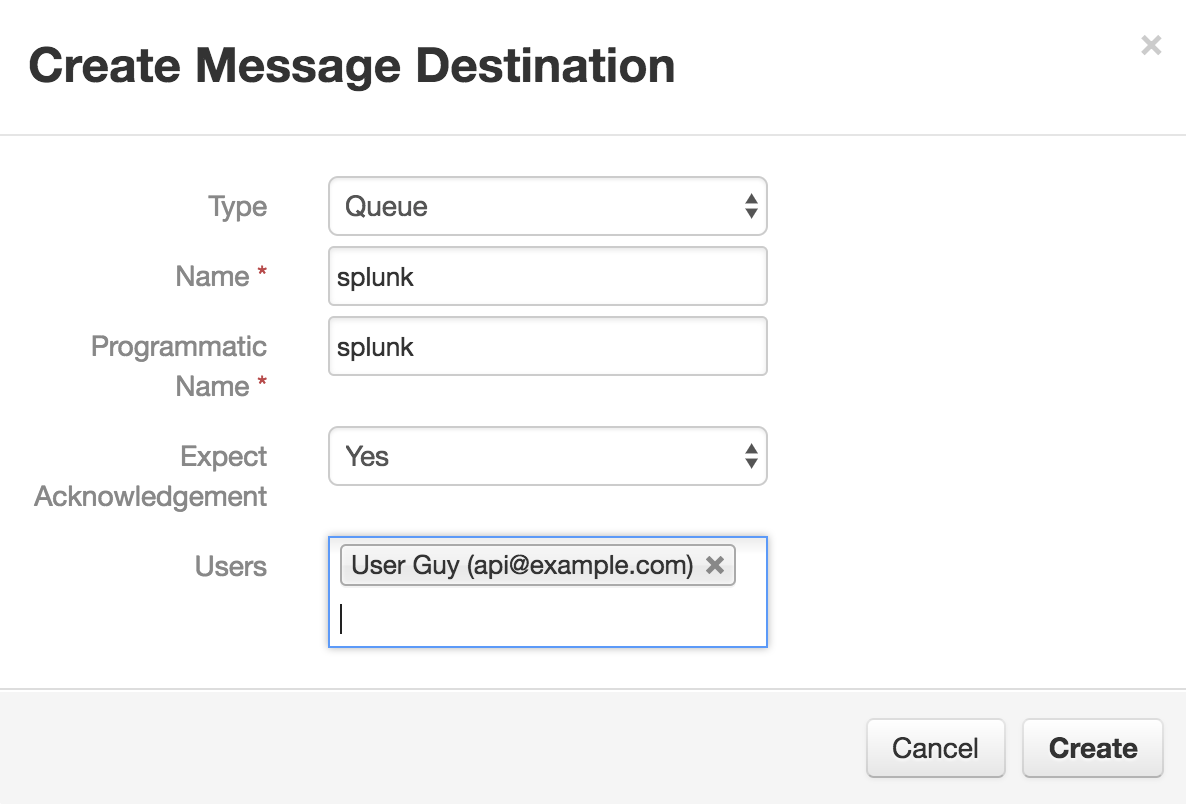
In order to run the application as configured in the example app.config, you must create a message destination, and then any number of query definition files and actions as desired.

### Message Destination

In Resilient, open Administrator Settings 🡪 Actions 🡪 Message Destinations.

Add a message destination, type “Queue”, with programmatic name “splunk”.

Add your API user account to the Users list for this message destination.

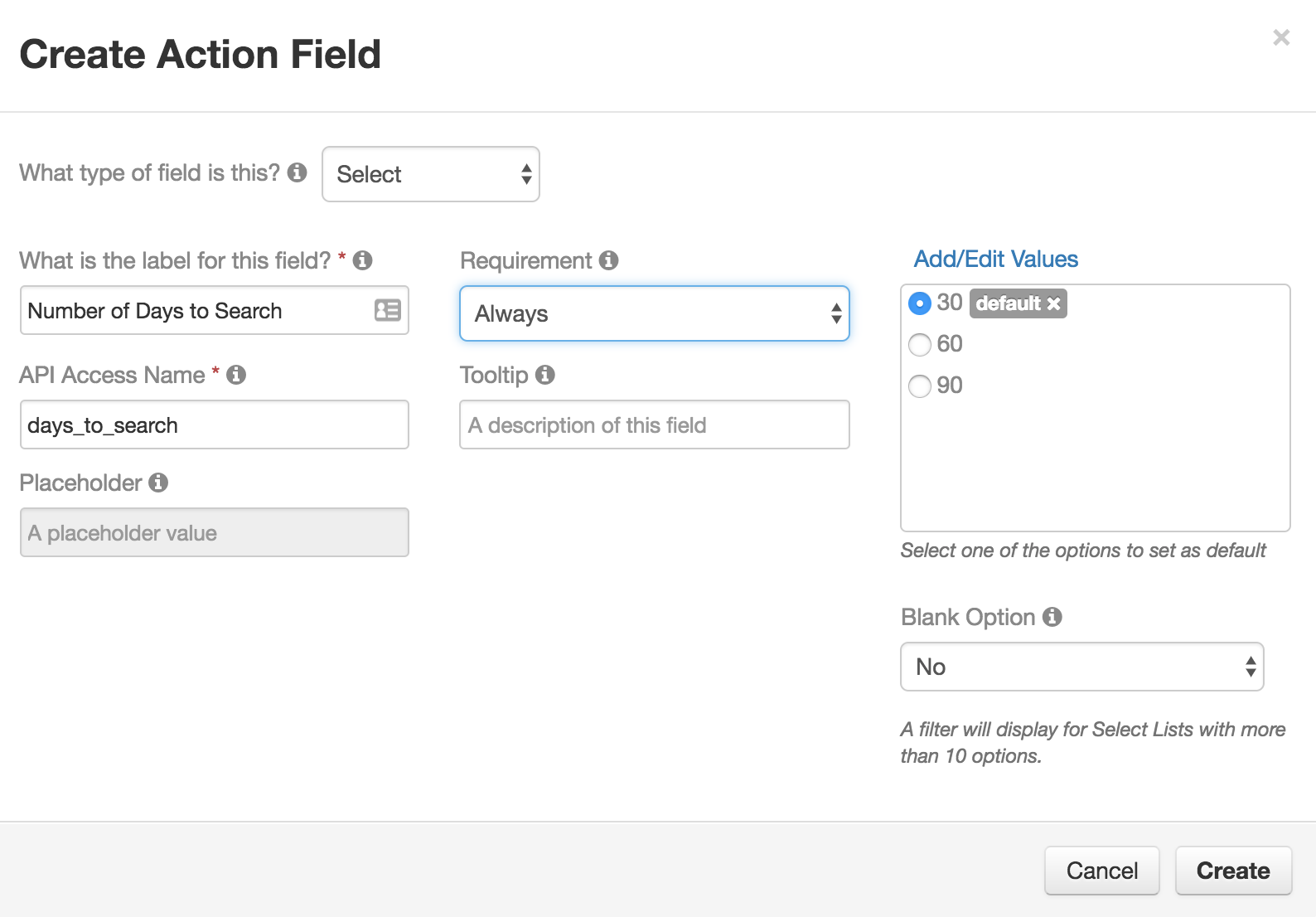


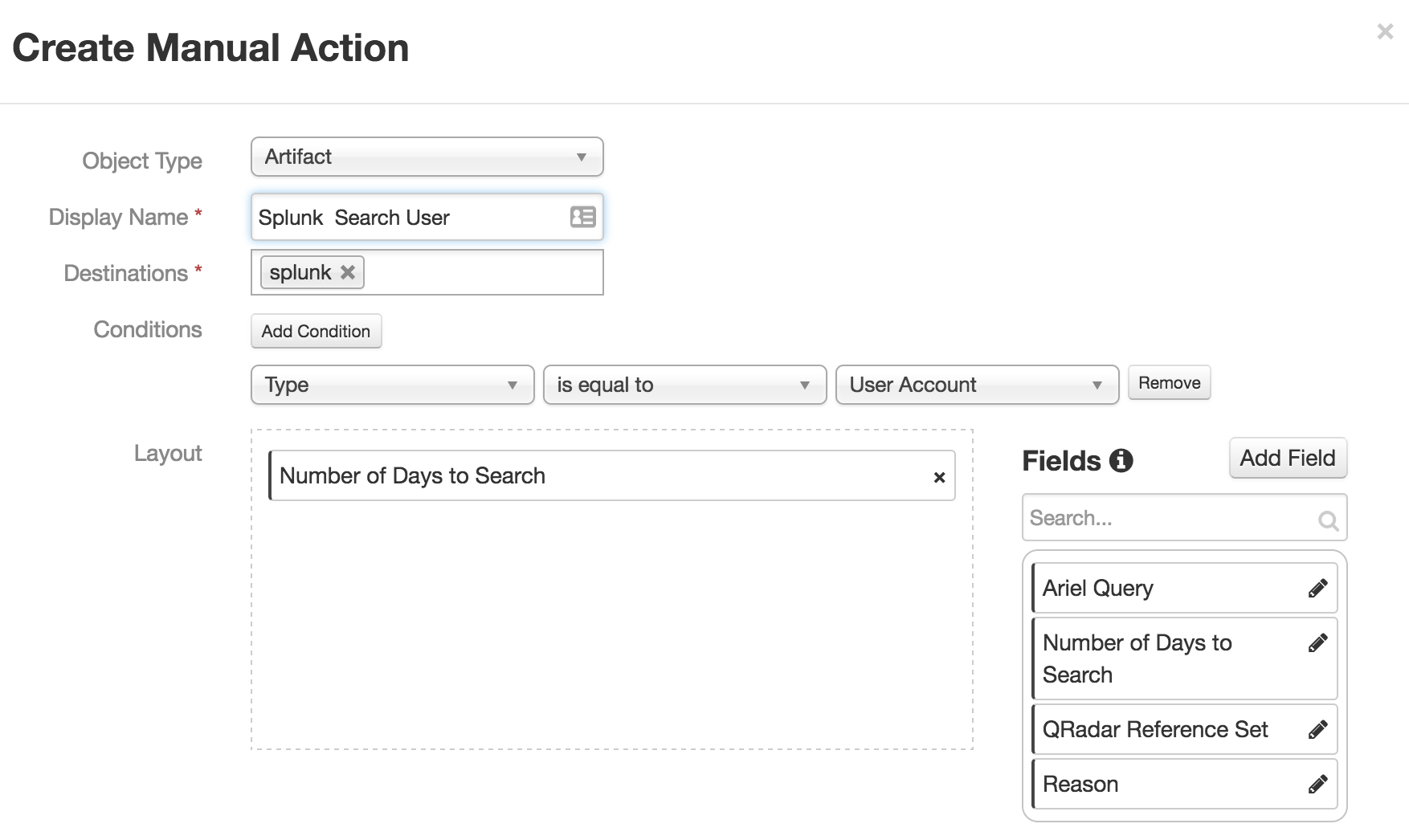
# Creating Search Actions

A search action will consist of an action in Resilient (either manual or automatic) and a corresponding query definition file in the directory specified in the app.config file.

## Create Action

A search action can be either manual or automatic. If a manual action is used, you can add any action fields that you need for the query, such as a start and stop time, a LIMIT value, etc. Use the “splunk” destination.





## Create Query Definition

The query definition file will need to have the same name as the “api name” of the action. For example, if an action’s displayed name is “My Action”, then the api name of it would be “my\_action”. It should be placed in the directory specified in your app.config file.

A query definition file is in Json format.

|  |
| --- |
| {  "comments": "<description or purpose of this query>",  "vars": {  "<var\_1>": "<string to be rendered for use in query or mapping>",  "<var\_2>": "<string to be rendered for use in query or mapping>"  },  "query": {  "expression": "<query to run>",  "limit": "<max results to retrieve>",  "extract\_results\_from": "results",  "additional\_queries": [  "<additional query file to run for this action",  "<additional query file to run for this action"  ]  },  "incident\_fields": {  "<field1>": "{{ row.<column 1>|js }}",  "<field2>": "{{ row.<column 2>|js }}"  },  "artifacts": {  "value": "{{ row.<column 3>|js }}",  "type": "<artifact type>",  "description": "<description>"  },  "datatables": [  {  "name": "<data\_table\_name>",  "row\_id": "{{row.id}}",  "keys": [  "<col1>",  "<col2>"  ],  "cells": {  "<col 1>": {  "value": "{{result.column1|js}}"  },  "<col 2>": {  "value": "{{artifact.value|js}}"  },  "<col 3>": {  "value": "{{result.column2|js}} {{result.column4|js}}"  }  }  }  ],  "attachment": {  "name": "Splunk\_File",  "keys": [  "<col 1>",  "<col 2>",  "<col 4>"  ],  "ext": "<file extension to use>"  }  } |

The **“comments”** value is optional. It is a place to record a description for the purpose or intended use of the query.

The **“expression”** is the query that will be run. You can use Jinja2 syntax to substitute in values from incident fields. For example, {{ incident.discovered\_date }}. Custom fields are accessed like {{ incident.properties.my\_custom\_field }}. Manual action fields are accessed like {{ properties.my\_action\_field }}. All incident values need to be valid Json in order to be submitted successfully to the Resilient REST API. Using the “js” filter as shown will ensure that any reserved characters are escaped appropriately. Another useful Jinja filter when building queries is “datetime”. It will take an epoch time in milliseconds and convert it to the Datetime stamp format that Splunk requires for time parameters.

The **“vars”** section is optional. It contains a mapping of variable names to string values. These variables will be rendered before the query is run, and so can be used when constructing the query. Include a variable in a query expression like “ search {{ var\_name }}”

The **“additional\_queries”** list is optional. If you want a single Resilient action to trigger multiple queries, then you can list those additional query definition files here.

The **“limit”** value is optional. It specifies the maximum number of search results to retrieve.

The **“extract\_results\_from”** should always be set to “results” for Splunk searches if it is included. If “extract\_results\_from” is included, then the data will be accessed from a member called “result” when you do the incident or artifact mapping. If extract\_results\_from is not included, then the entire search result list will be available all at once for mapping.

The **“incident\_fields”** section is a mapping specifying the incident fields that should be updated from the search result. The value for the field will use Jinja2 syntax to specify data from the query result. If a key was specified for “extract\_results\_from”, then the value will be rendered with a single object called “result” that contains all columns from the first row of data returned in the list. Any other rows returned will be ignored. If it was not included, then all of the objects from the result structure will be available when the value is rendered.

The **“artifacts”** section specifies how to create an artifact from the search results. Assuming an “extract\_results\_from” value was specified in the query definition, then the application will iterate over the list of rows and one artifact will be created for each row item. The columns accessed from an object called “result”.

The **“datatables”** section specifies a list of Resilient data tables for which you want to add or update rows from the search results.

The **“name”** is the name of the data table.

The **“row\_id”** should be included if this action should update the same data table row that the custom action was triggered from.

The **“keys”** list is optional. It indicates which columns uniquely distinguish a row in a data table. If keys is omitted, and a data table row is found that matches on the key fields, then that row will be updated. If keys is omitted, or no matching row is found, then a new row is added to the table.

The **“cells”** section is a mapping of data table column names to the value that should be used for creating or updating a row.

The **“attachment”** section specifies how to create a CSV file from the search results to upload to the incident as an attachment.

The **“name”** value will be combined with a timestamp to create the filename.

The **“keys”** list is optional and lists which columns in the search results to include in the CSV file. If excluded, all columns will be included. Including a keys list is more efficient even if you want all the columns as it eliminates the need to walk the result list twice to determine what the column names are.

The **“ext”** value is optional and lets you specify a file extension to use. If excluded, “.csv” is used for the filename.

## Examples

### Example 1

Grabs up to 5 results from a Splunk search and creates an artifact from each one. A custom incident field in Resilient called my\_custom\_property will be populated from the first result returned. It searches data going back n number of days, where n comes from the days\_to\_search action field.

This query definition would work for either an automatic or manual action.

|  |
| --- |
| {  "comments": "Check splunkd.log for artifact value",  "vars": {  "days": "{{ properties|value('days\_to\_search') }}"  },  "query": {  "expression": "search index=\_internal source = \/Applications\/Splunk\/var\/log\/splunk\/splunkd.log \"{{ artifact.value }}\" earliest=-{{ days }}d | fields component,message",  "limit": 5,  "extract\_results\_from": "results"  },  "incident\_fields": {  "my\_custom\_property": "{{ result.message|js }}"  },  "artifacts": {  "value": "{{ result.component|js }}",  "type": "String",  "description": "From Splunk {{ result.splunk\_server|js }}"  }  } |

### Example 2

Grabs up to 10 results from a Splunk search and create or update a data table row for each one. Each data table row will have a unique set of values for col1 and col2. Col3 will store the number of results from splunk for that combination. It searches data going back n number of days, where n comes from the days\_to\_search action field.

It also stores a subset of the fields returned as an attachment to the incident.

This query definition would work for either an automatic or manual action.

|  |
| --- |
| {  "comments": "An example with counts",  "query": {  "expression": "search index=\_internal source = \/Applications\/Splunk\/var\/log\/splunk\/splunkd.log {{ artifact.value }} earliest=-{{ properties|properties\_value(\"days\_to\_search\", type\_info) }}d | stats count by host, component",  "limit": 10,  "extract\_results\_from": "results"  },  "datatables": [  {  "name": "test\_table",  "keys": ["col2", "col1"],  "cells": {  "col1": {  "value": "{{artifact.value}}"  },  "col2": {  "value": "{{result.host|js}} - {{result.component|js}}"  },  "col3": {  "value": "{{result.count}}"  }  }  }  ] ,  "attachment": {  "name": "Splunk\_Result",  "keys": [  "host",  "component"  ]  }  } |