

# Forward-Looking Statements



This presentation may contain forward-looking statements regarding future events, plans or the expected financial performance of our company, including our expectations regarding our products, technology, strategy, customers, markets, acquisitions and investments. These statements reflect management's current expectations, estimates and assumptions based on the information currently available to us. These forward-looking statements are not guarantees of future performance and involve significant risks, uncertainties and other factors that may cause our actual results, performance or achievements to be materially different from results, performance or achievements expressed or implied by the forward-looking statements contained in this presentation.

For additional information about factors that could cause actual results to differ materially from those described in the forward-looking statements made in this presentation, please refer to our periodic reports and other filings with the SEC, including the risk factors identified in our most recent quarterly reports on Form 10-Q and annual reports on Form 10-K, copies of which may be obtained by visiting the Splunk Investor Relations website at [www.investors.splunk.com](http://www.investors.splunk.com) or the SEC's website at [www.sec.gov](http://www.sec.gov). The forward-looking statements made in this presentation are made as of the time and date of this presentation. If reviewed after the initial presentation, even if made available by us, on our website or otherwise, it may not contain current or accurate information. We disclaim any obligation to update or revise any forward-looking statement based on new information, future events or otherwise, except as required by applicable law.

In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only and shall not be incorporated into any contract or other commitment. We undertake no obligation either to develop the features or functionalities described, in beta or in preview (used interchangeably), or to include any such feature or functionality in a future release.

Splunk, Splunk> and Turn Data Into Doing are trademarks and registered trademarks of Splunk Inc. in the United States and other countries. All other brand names, product names or trademarks belong to their respective owners. © 2023 Splunk Inc. All rights reserved.

# De-Escher-ing SOAR Playbook Development

Simplifying the Stairwell  
SEC1475C

**Mhike Funderburk**

Principal Security Engineer | UltraViolet Cyber

**Luke Summers**

SOAR Technical Lead | UltraViolet Cyber



splunk> .conf23

# De-Escher-ing SOAR Playbook Development

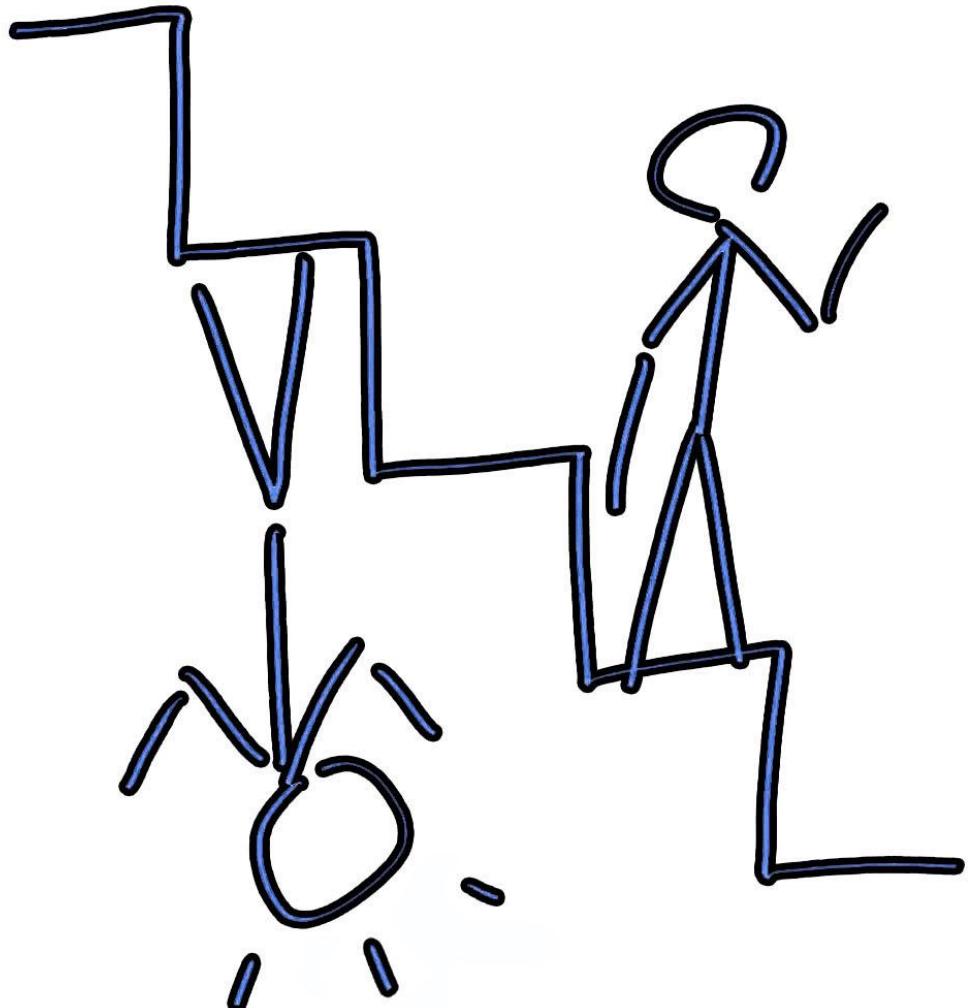
Simplifying the Stairwell  
SEC1475C

**Mhike Funderburk**

Principal Security Engineer | UltraViolet Cyber

**Luke Summers**

SOAR Technical Lead | UltraViolet Cyber



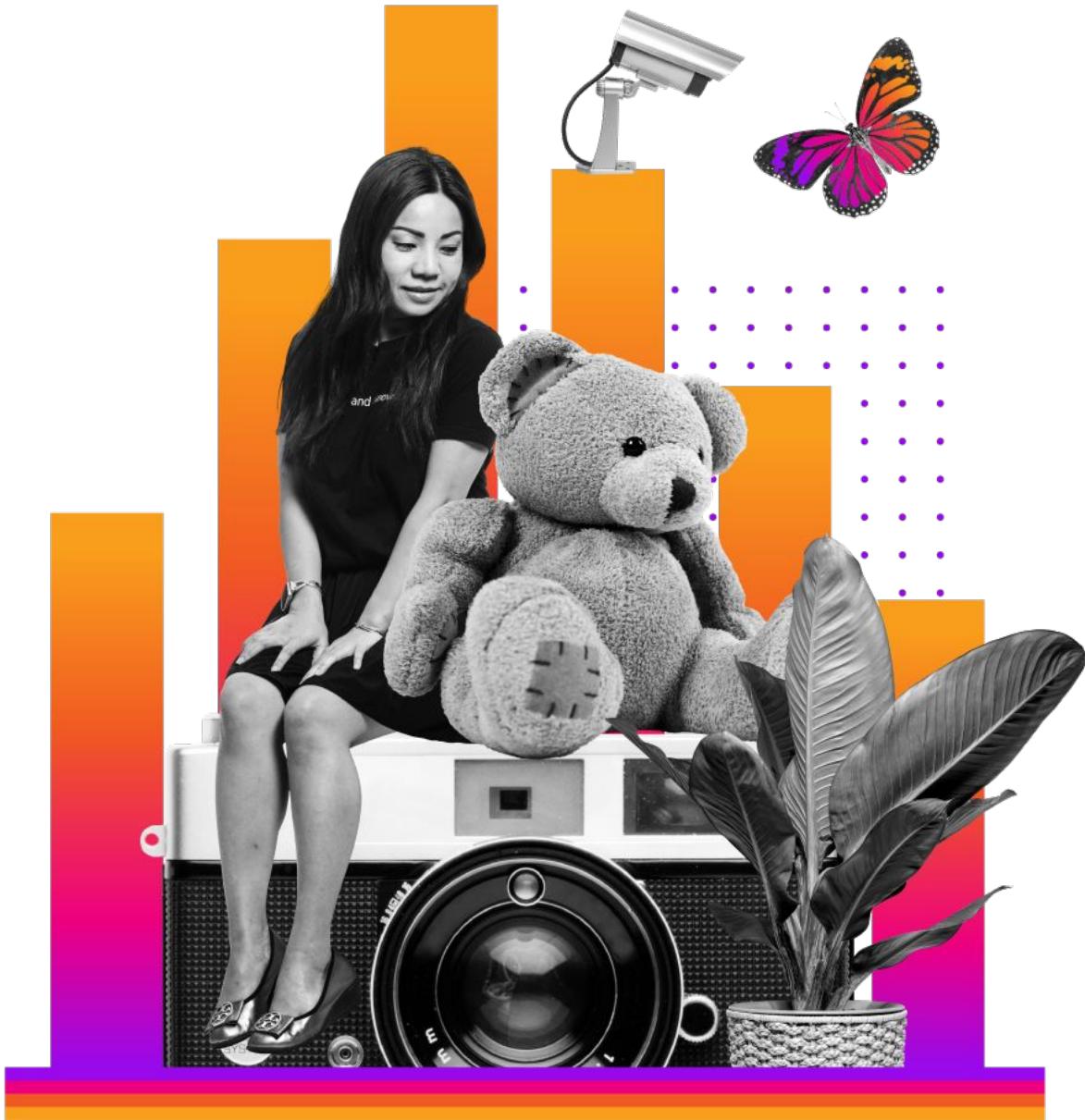


## Mhike Funderburk

Principal Security Engineer | UltraViolet Cyber

## Luke Summers

SOAR Technical Lead | UltraViolet Cyber



# Datapaths

Starting simple

splunk> .conf23

# Datapath Structure

1	event	Artifact 1	MEDIUM	admin
Name	Artifact 1		Start Time	May 7th at 7:46 pm
Label	event		Created	May 7th at 7:46 pm
Created by	admin		Type	N/A
Source ID	e7ce77f1-f5fe-4999-990d-8d8b978f6ce7	Severity	Medium	
Details				
src			192.168.0.1 ▾	

**artifact:\***  
**.cef.src**

Artifacts are JSON representations of events and data

**artifact:\***

Retrieves all artifacts in the container

**.cef**

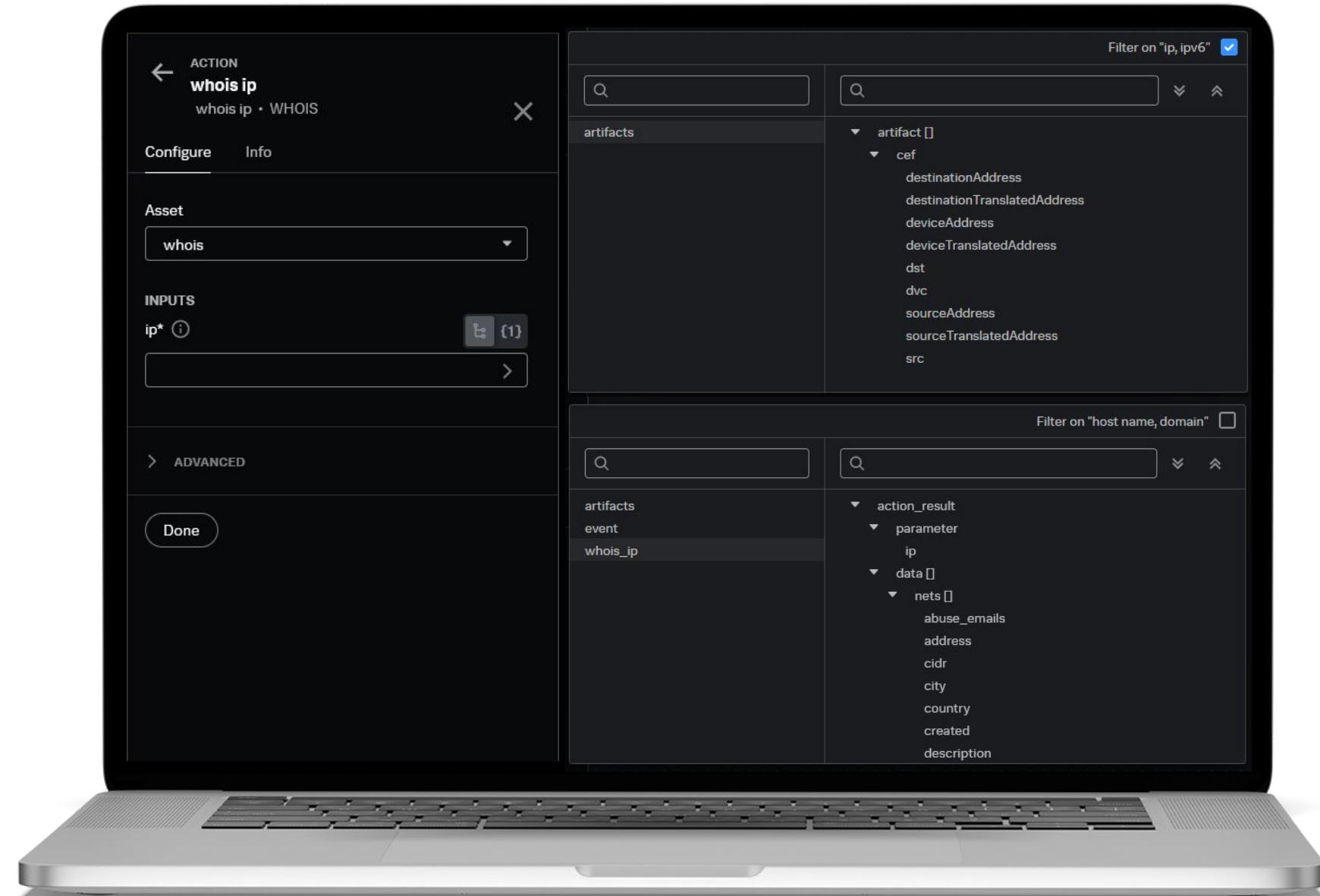
Literal name of the JSON dictionary containing the key-value pairs

**.src**

The name of the field which contains the value you want

# Selecting Datapaths

- Artifact field names based on the CEF naming convention
- Action result field names configured by the utilized app
- The options in this menu are not exhaustive



# Datapath Overrides

Some modifications to datapaths can get help get what you need. Others that look like they would work return null.

- artifact:\*.cef.**my\_field**
  - Normal Operation



# Datapath Overrides

Some modifications to datapaths can get help get what you need. Others that look like they would work return null.

- `artifact:*.cef.my_field`
  - Normal Operation
- `artifact:*.cef`
  - Get the entire cef dictionary for the artifacts



# Datapath Overrides

Some modifications to datapaths can get help get what you need. Others that look like they would work return null.

- `artifact:*.cef.my_field`
  - Normal Operation
- `artifact:*.cef`
  - Get the entire cef dictionary for the artifacts
- `artifact:*`
  - Invalid datapath



# Datapath Overrides

Some modifications to datapaths can get help get what you need. Others that look like they would work return null.



- `artifact:*.cef.my_field`
  - Normal Operation
- `artifact:*.cef`
  - Get the entire cef dictionary for the artifacts
- `artifact:*`
  - Invalid datapath
- `my_action_1:action_result.data.my_field`
  - Normal Operation
- `my_action_1:action_result.data`
  - Get the entire data dictionary from the action result
- `my_action_1:action_result`
  - Invalid datapath. Returns a null list

## test\_datapaths Function

[Copy](#)

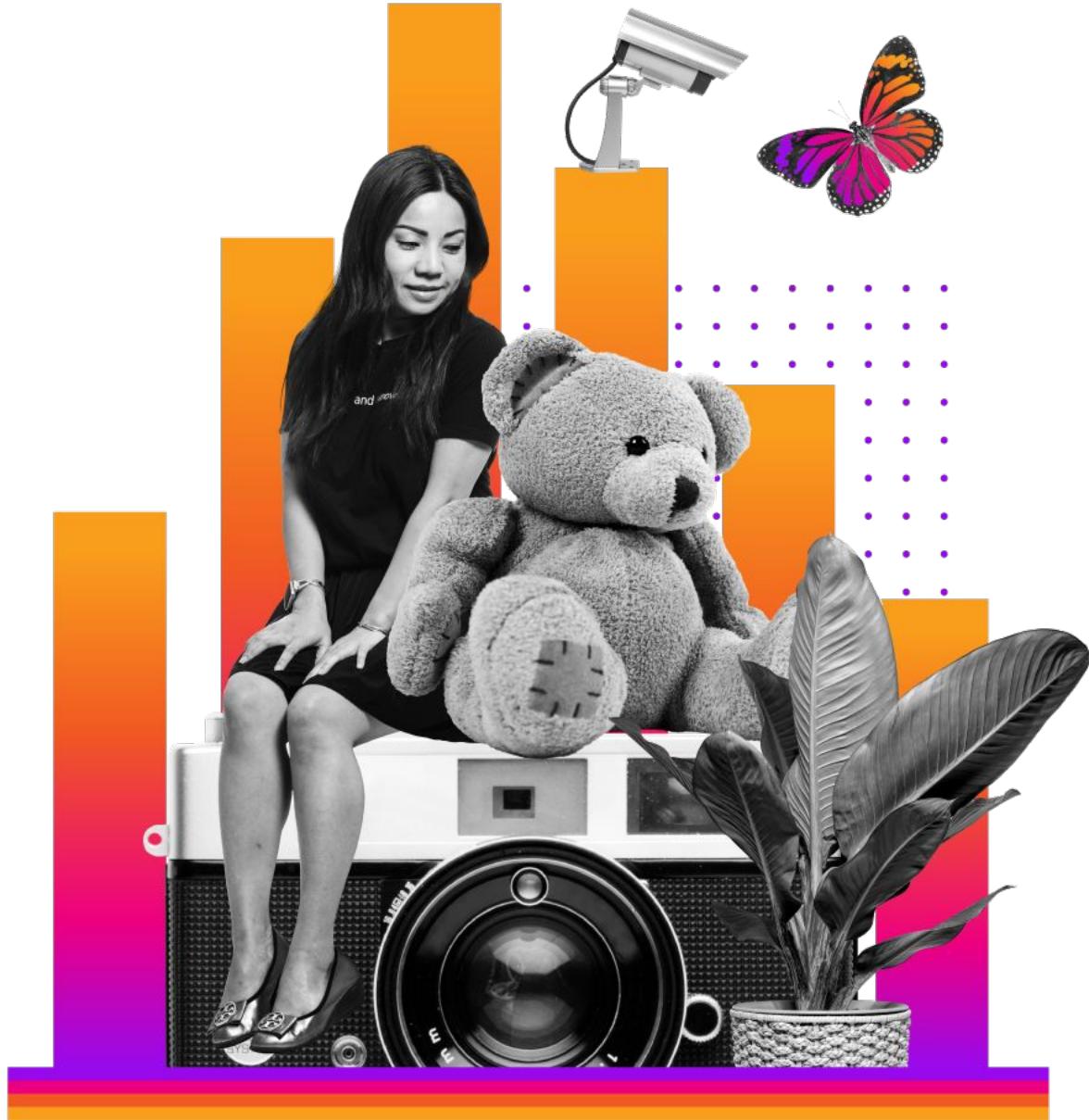
```
@phantom.playbook_block()
def test_datapaths(action=None, success=None, container=None, results=None, handle=None, filtered_artifacts=None):
    phantom.debug("test_datapaths() called")

    container_artifact_data = phantom.collect2(container=container, datapath=["artifact:*.cef.my_field"])

    container_artifact_cef_item_0 = [item[0] for item in container_artifact_data]

    container_artifact_data = phantom.collect2(container=container, datapath=["artifact:*.cef"])
    container_artifact_header_item_0 = [item[0] for item in container_artifact_data]

    input_parameter_0 = "artifact:*
```



# Decisions & Filters

Ramping up a little

splunk> .conf23

# What's The Difference?

Don't they both contain if statement operators?

## Decision Blocks

---

- Works like an if...else statement
- Only executes one branch
- Has no effect on the data

## Filter Blocks

---

- Functions as back-to-back if statements
- Any and all branches can be executed
- Returns a subset of the given data

ARTIFACTS (3)				
ID	LABEL	NAME	SEVERITY	CREATED BY
3	event	Artifact 3	MEDIUM	admin
	Name	Artifact 3	Start Time	May 7th at 7:46 pm
	Label	event	Created	May 7th at 7:46 pm
	Created by	admin	Type	N/A
	Source ID	996f0c12-cfe9-4fb5-a7ca-53b065c4669f	Severity	Medium
	Details			
	destinationPort		80 ▾	
2	event	Artifact 2	MEDIUM	admin
	Name	Artifact 2	Start Time	May 7th at 7:46 pm
	Label	event	Created	May 7th at 7:46 pm
	Created by	admin	Type	N/A
	Source ID	942b77db-6592-405d-bd16-1aba9da646e0	Severity	Medium
	Details			
	sourceUserName		user3@company.com ▾	
1	event	Artifact 1	MEDIUM	admin
	Name	Artifact 1	Start Time	May 7th at 7:46 pm
	Label	event	Created	May 7th at 7:46 pm
	Created by	admin	Type	N/A
	Source ID	e7ce77f1-f5fe-4999-990d-8d8b978f6ce7	Severity	Medium
	Details			
	destinationUserName		[user1@company.com, "user2@company.com"] ▾	
	src		192.168.0.1 ▾	

I Want the Src  
→ artifact:\*.cef.src



A [ '192.168.0.1' ]

B [ None ]

C [ '192.168.0.1', None, None ]

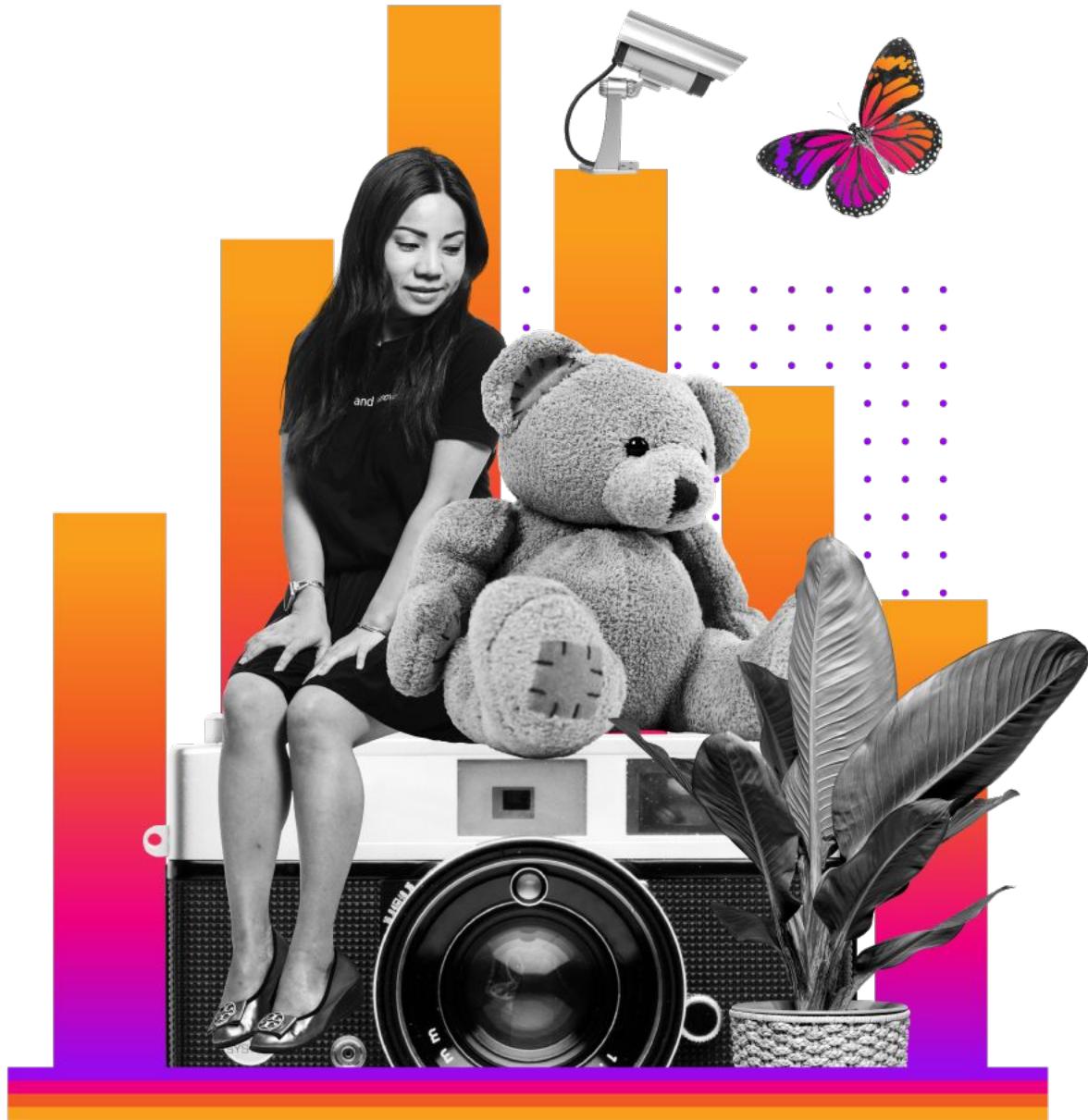
D [ None, None, None ]

# Filter Example

- Filter blocks return a subset of the given data
- Compare the field to a blank input
- If you want less data than you have:  
Filter Blocks

The screenshot shows the Splunk Filter configuration interface. A modal window titled "src filter" is open, showing a single condition: "artifact:\*.cef.src != <Select Value>". The "Done" button is visible at the bottom. Below the configuration is a preview window displaying log entries. One entry is highlighted with a red box, showing the result of the filter application.

```
May 31, 14:26:48 : phantom.collect2(): called for datapath['filtered-data:src_filter:condition_1:artifact:*.cef.src'], scope: None and
May 31, 14:26:48 : phantom.get_run_data() called for key filtered-data:src_filter:condition_1
May 31, 14:26:48 : phantom.collect2(): Classified datapaths as [<DatapathClassification.FILTERED_ARTIFACT: 3>]
May 31, 14:26:48 : phantom.collect(): called with datapath as LIST of paths, scope='all' and limit=2000. Found 1 TOTAL artifacts
May 31, 14:26:48 : phantom.collect2(): called for datapath['artifact:*.cef.src'], scope: None and filter_artifacts: None
May 31, 14:26:48 : phantom.collect2(): Classified datapaths as [<DatapathClassification.ARTIFACT: 1>]
May 31, 14:26:48 : phantom.collect(): called with datapath as LIST of paths, scope='all' and limit=2000. Found 3 TOTAL artifacts
May 31, 14:26:48 : artifact:*.cef.src [192.168.0.1, None, None]
May 31, 14:26:48 : filtered-data:src_filter:condition_1:artifact:*.cef.src [192.168.0.1]
May 31, 14:26:48 : metrics: Playbook_id:140, run_id:41, container: 1, function: on_start. TIME_TAKEN 8ms
May 31, 14:26:48 : No actions or custom functions were executed
```



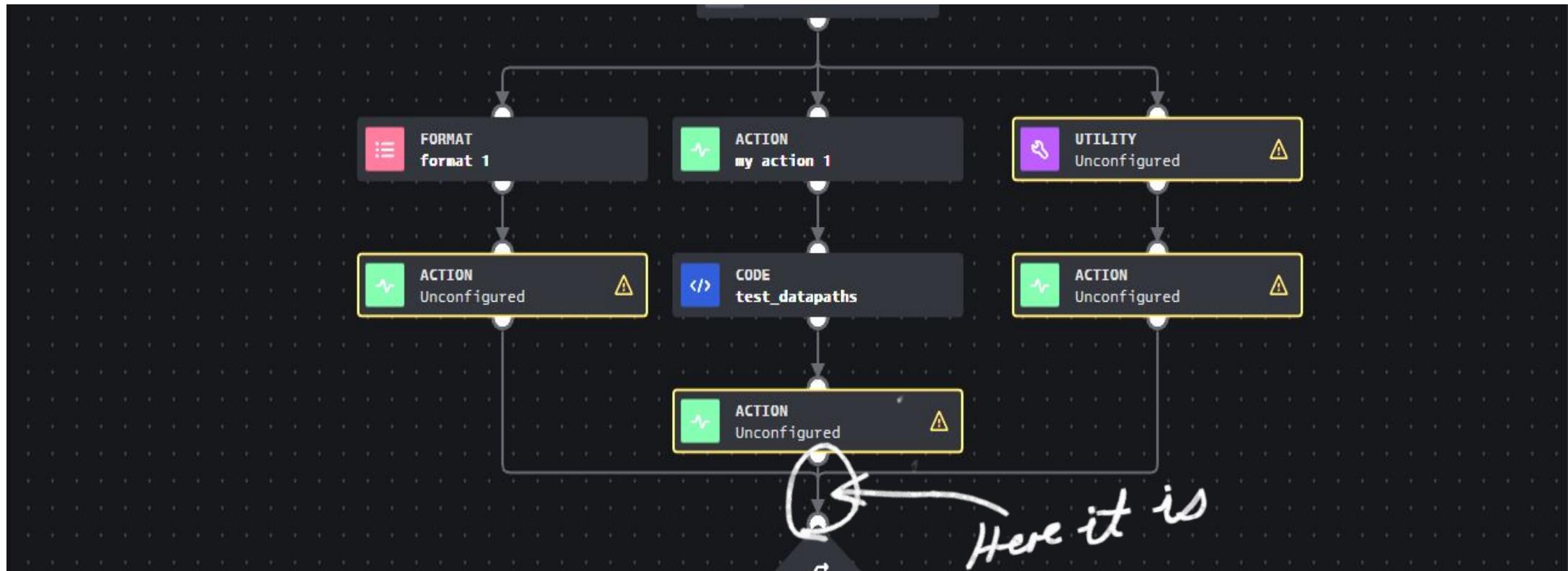
# The Joy of Joins

\*Spoiler Alert\* not a joy

splunk> .conf23

# Find the Join

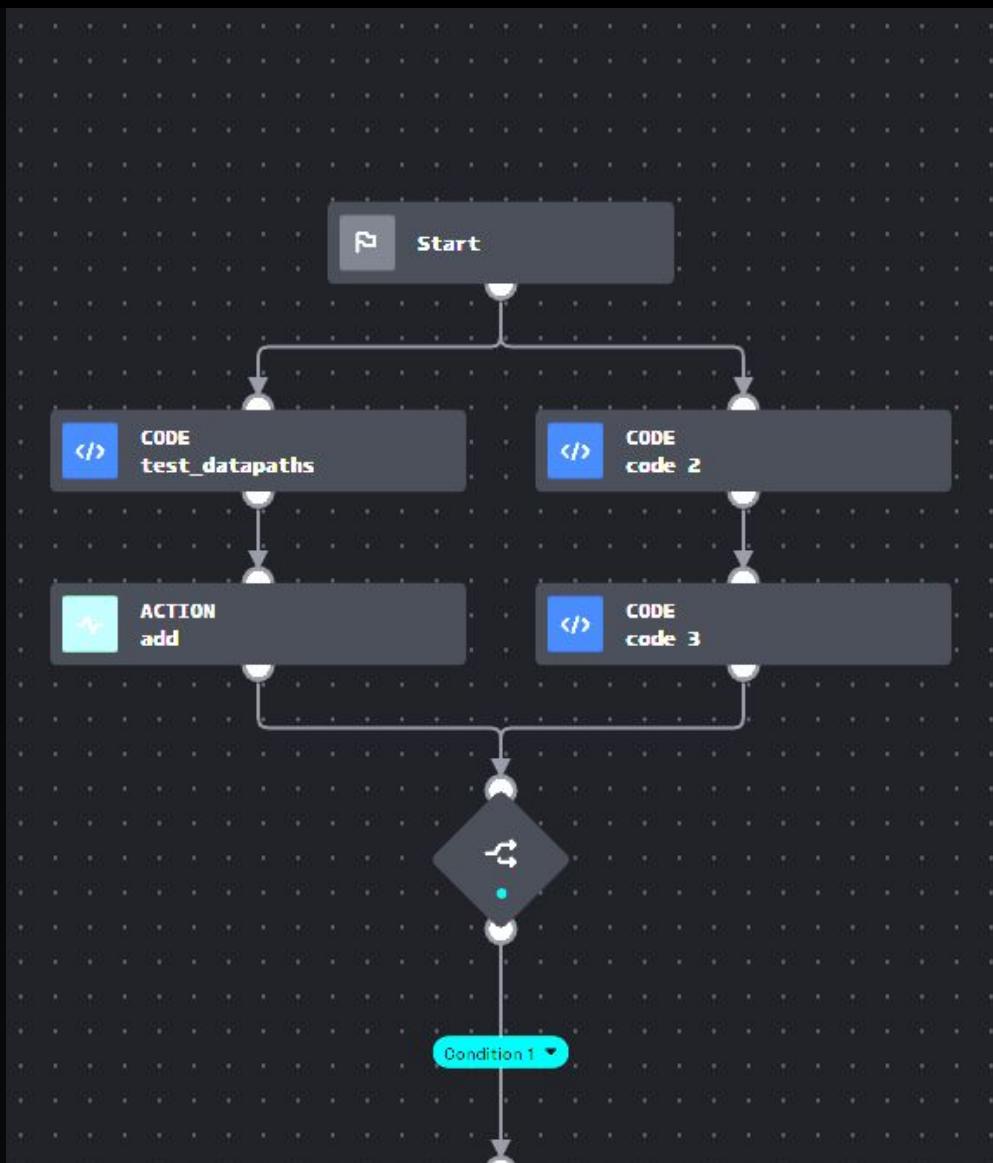
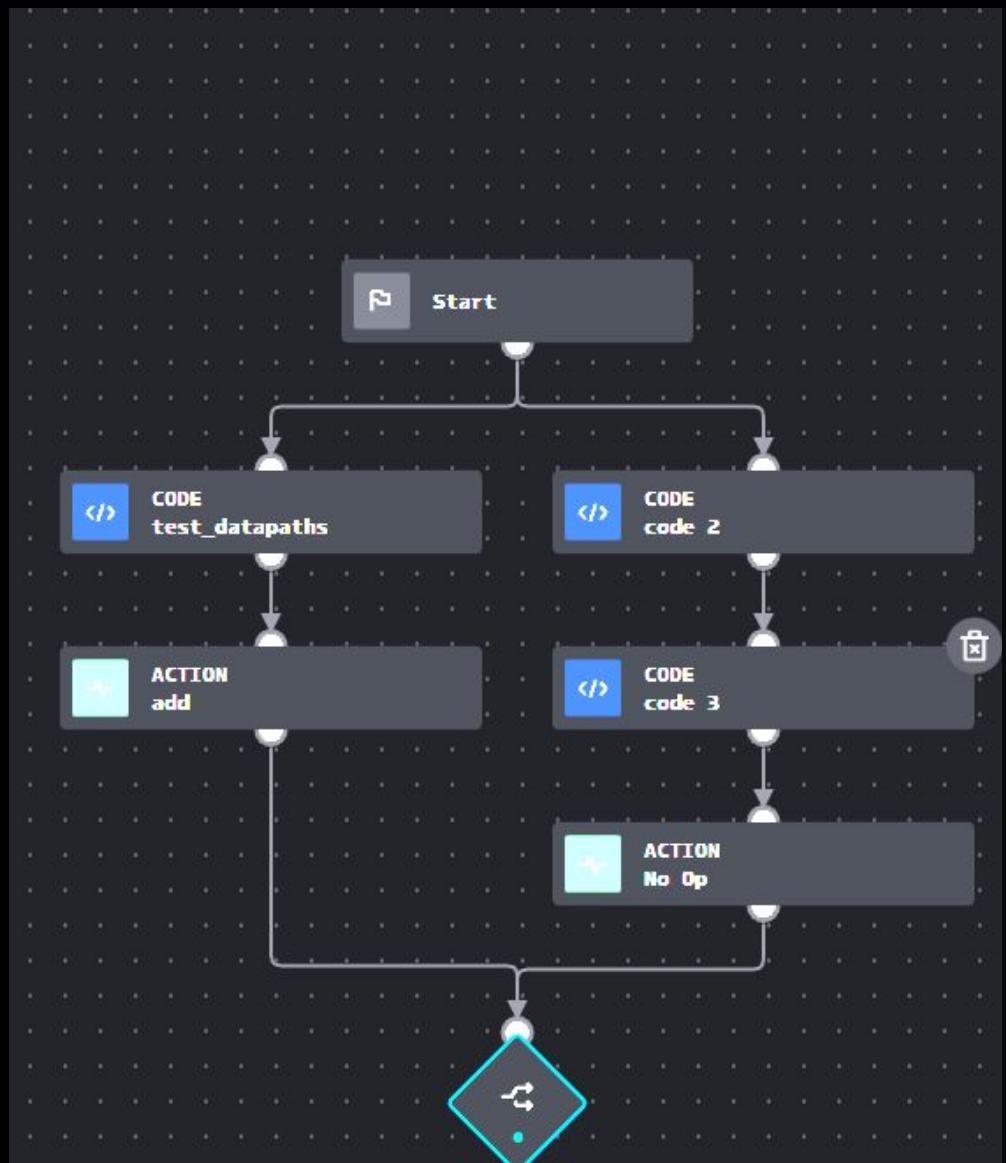
And then try not to use them

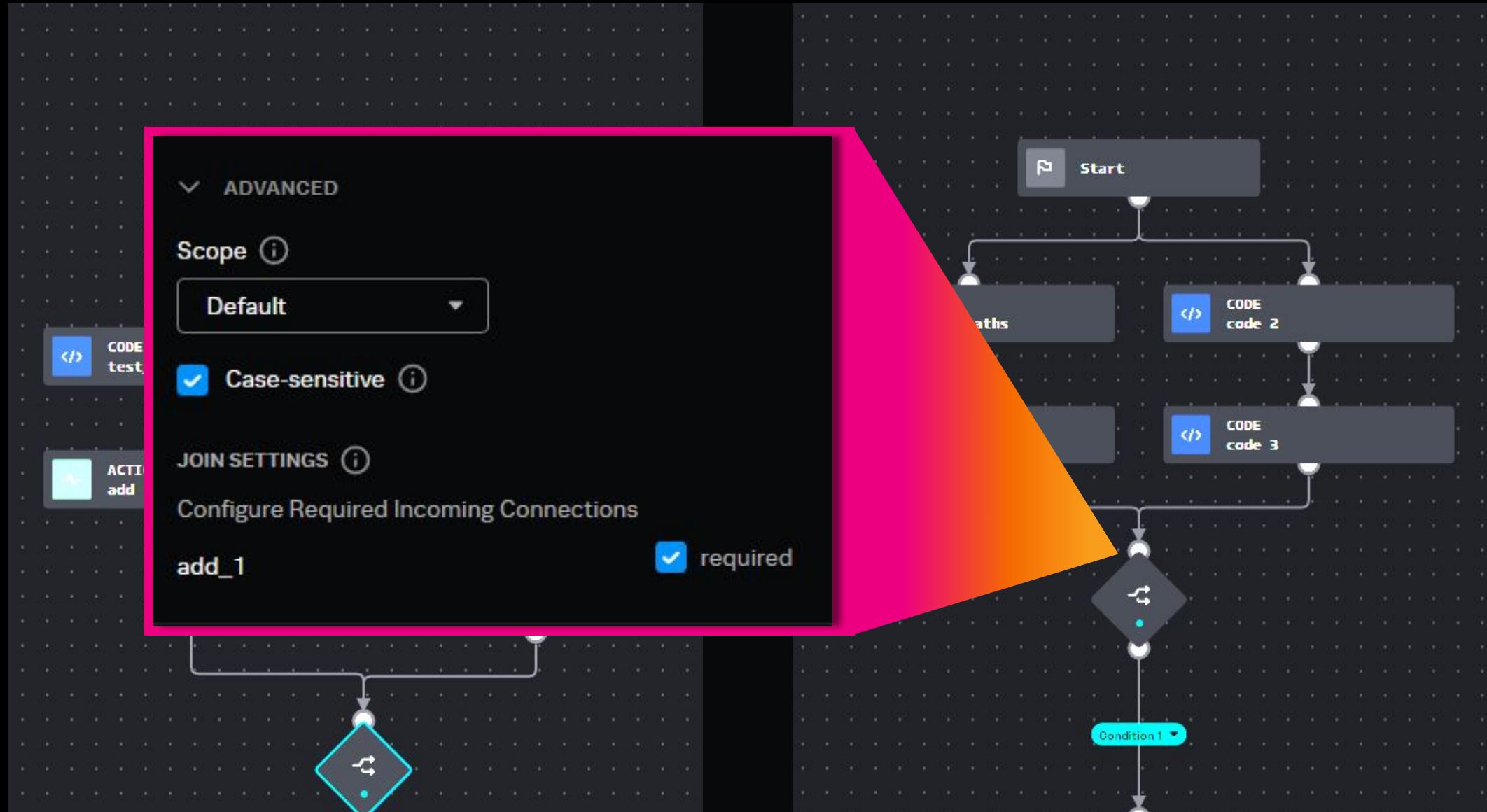


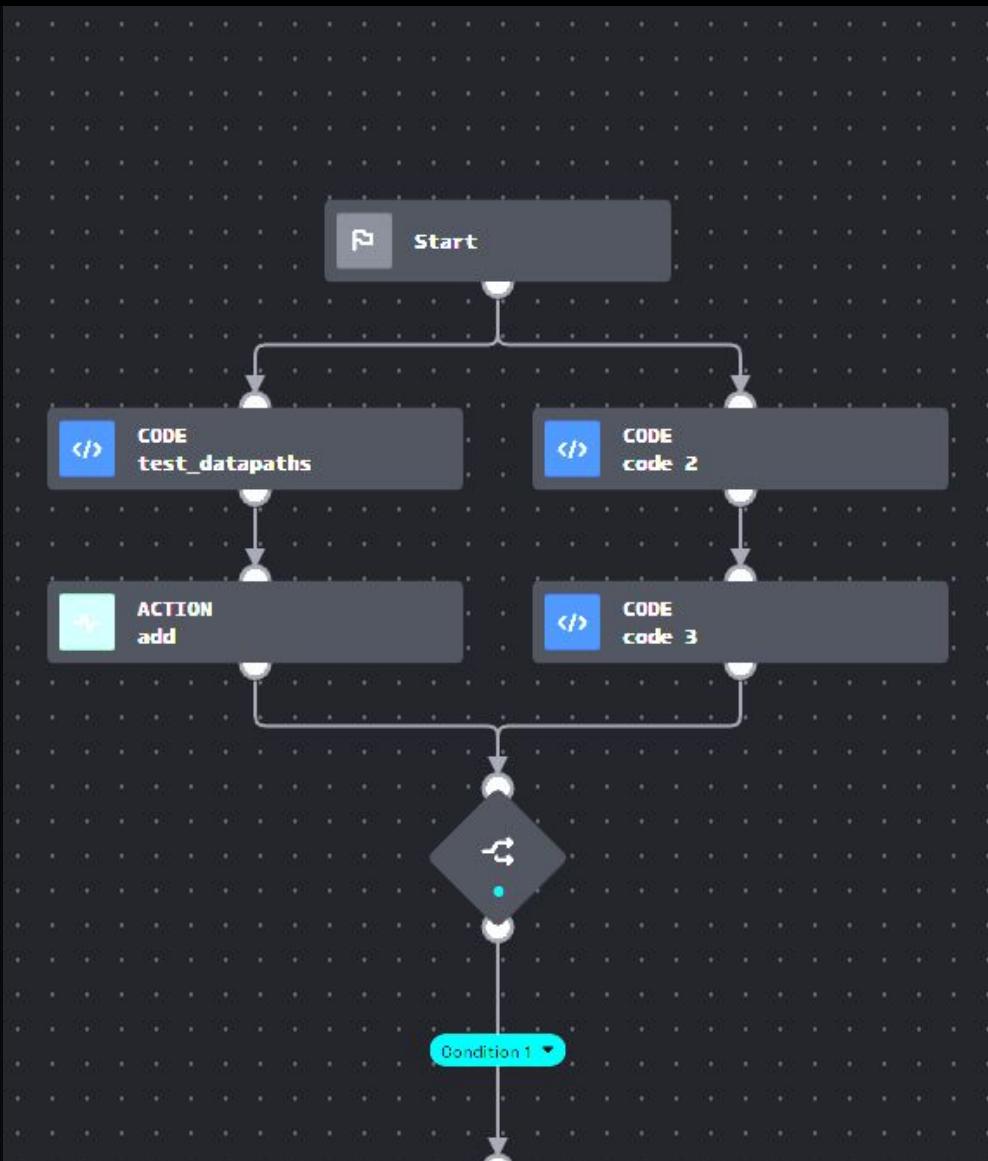
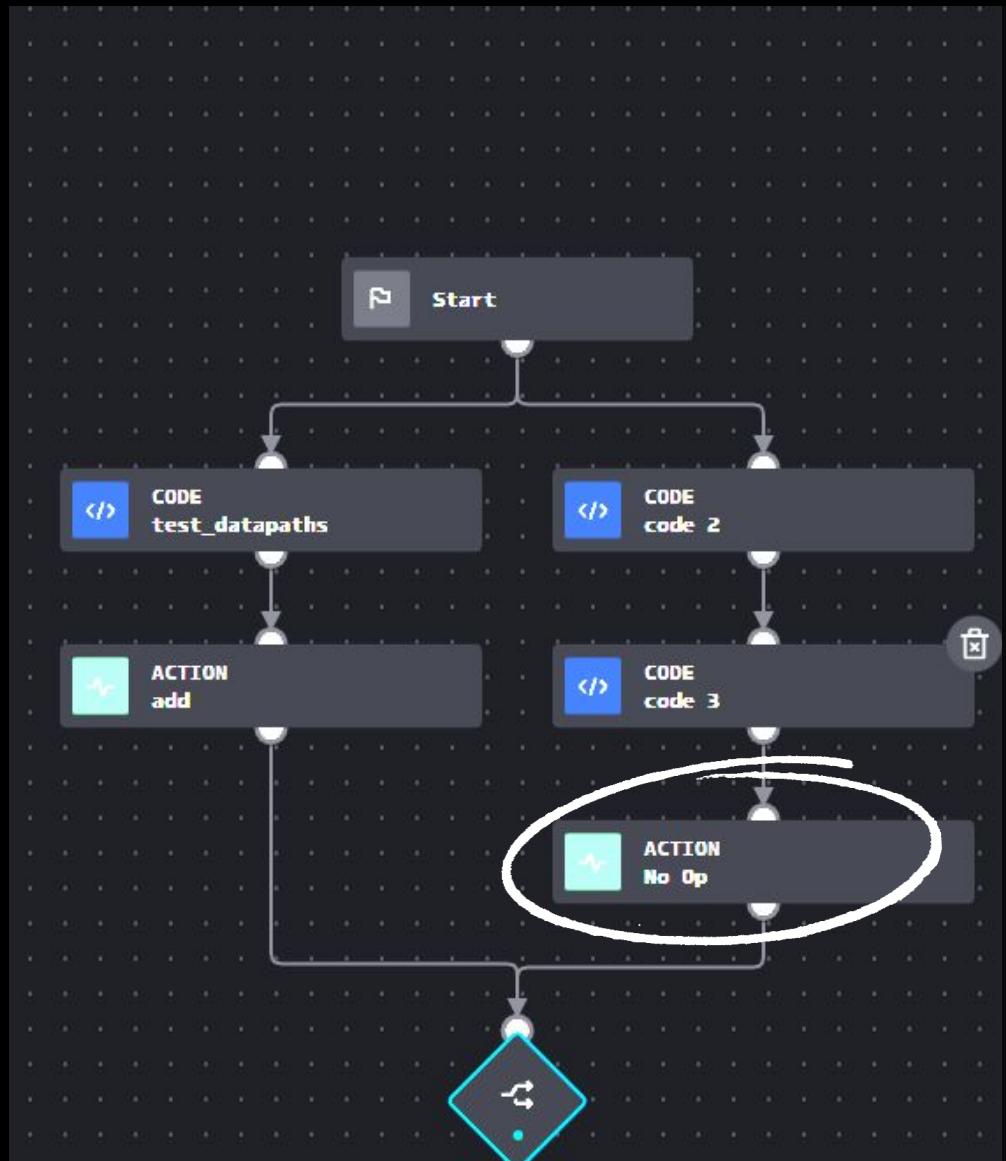
# Configuring Joins

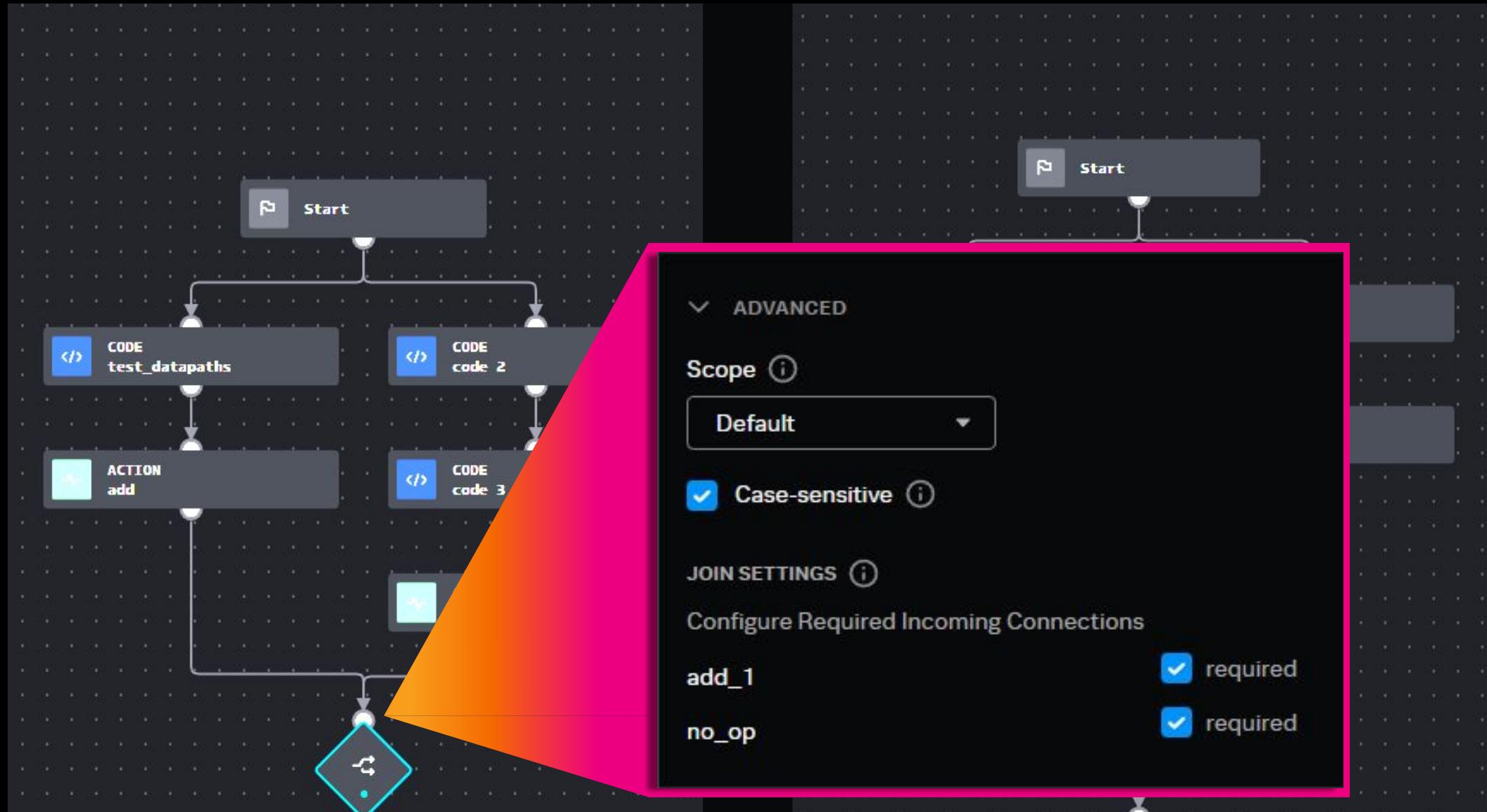
Everything is on

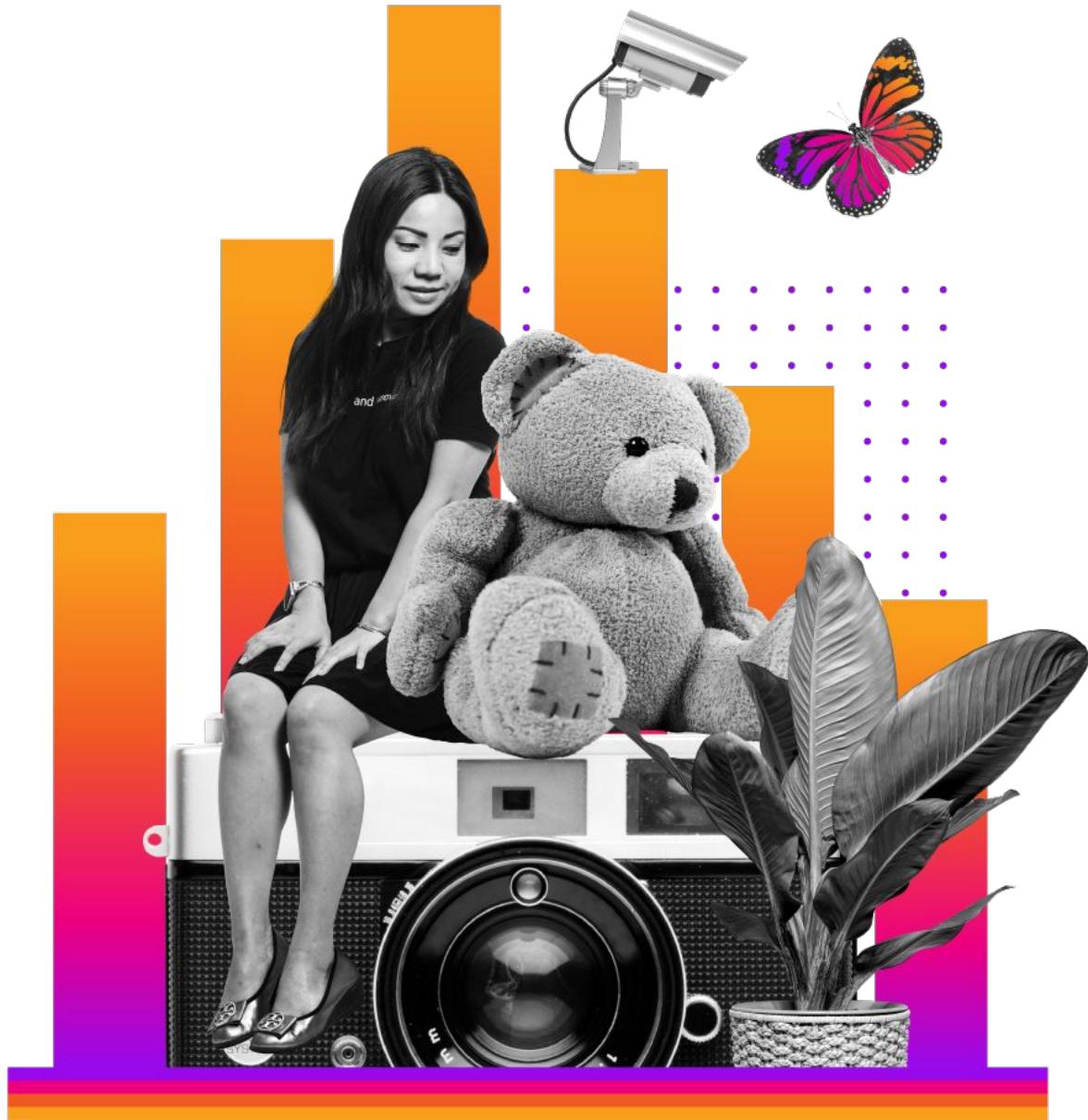












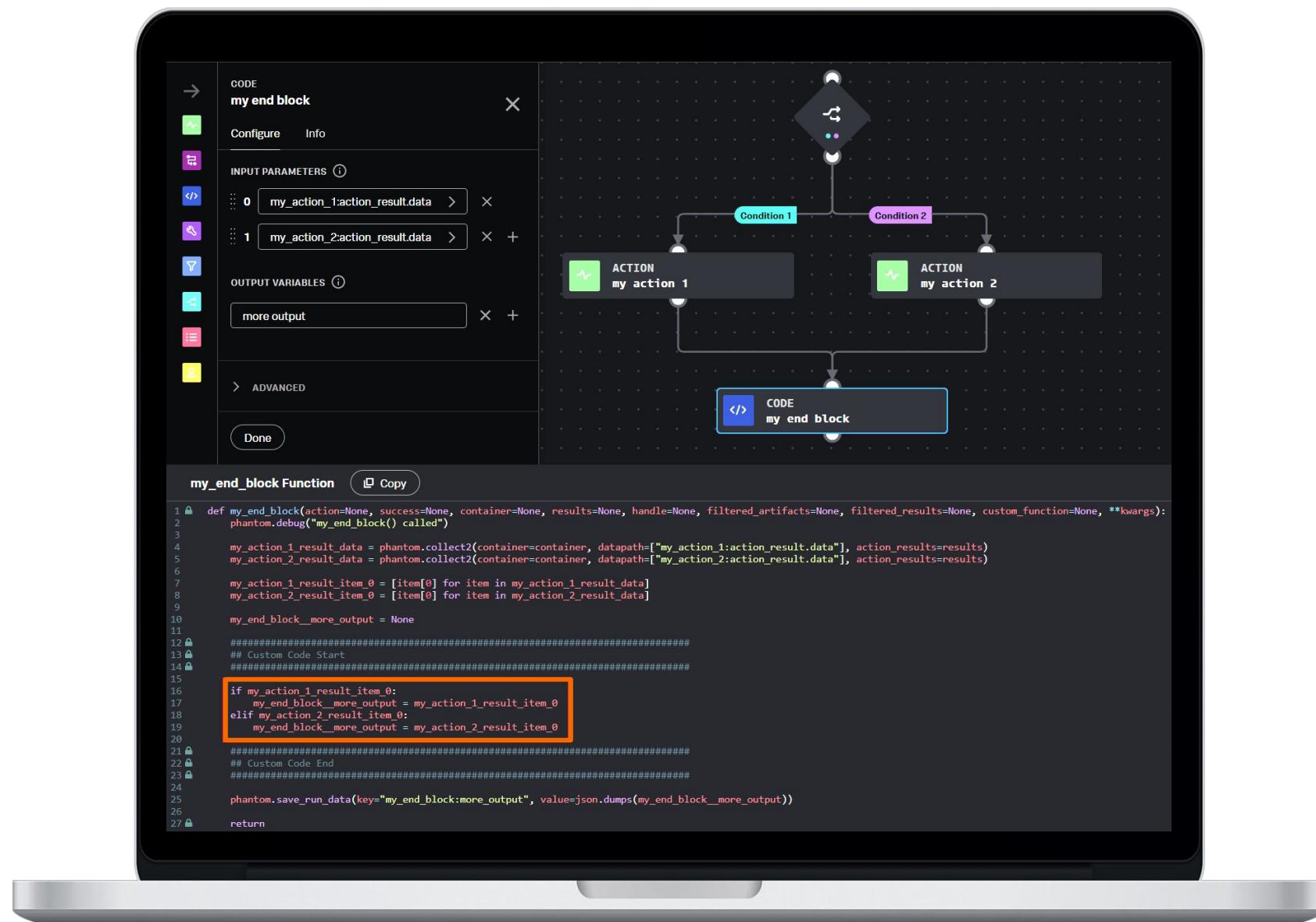
# Capturing Variables

Are we sure this is “no code”?

splunk> .conf23

# Global Variables?

- Global variables can be made in the global block
  - Can be accessed and modified in custom code blocks
  - Cannot be used as action inputs
  - Not recommended
- Using a code block to manually join the outputs of the two branches
  - Reasonable solution
  - Can be used for action inputs



The screenshot shows the SOAR platform's configuration interface for a custom action named "my end block".

**Configuration Panel:**

- CODE:** my end block
- Configure** and **Info** buttons
- INPUT PARAMETERS:** 0 items (my\_start\_code:custom\_function:r)
- OUTPUT VARIABLES:** more output
- ADVANCED** section
- Done** button

**Workflow Diagram:**

```

graph TD
    Start[CODE my start code] --> Path1[CODE path 1 code]
    Start --> Path2[CODE path 2 code]
    Path1 --> End[CODE my end block]
    Path2 --> End
  
```

**Code View:**

```

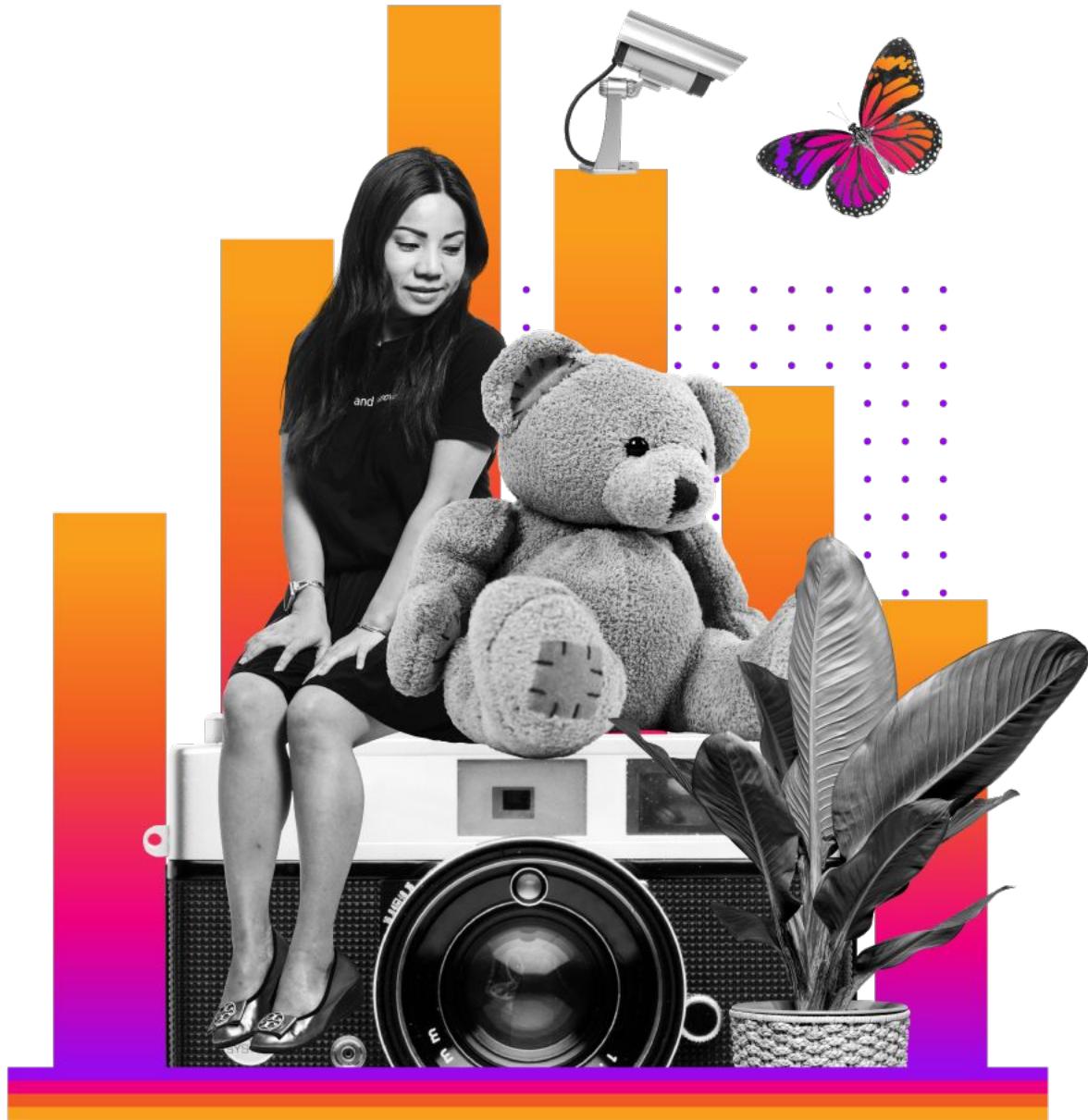
my_end_block Function [Copy]
1 def my_end_block(action=None, success=None, container=None, results=None, handle=None, filtered_artifacts=None, filtered_results=None, custom_function=None, **kwargs):
2     phantom.debug("my_end_block() called")
3
4     my_start_code_my_output = json.loads(phantom.get_run_data(key="my_start_code:my_output"))
5
6     my_end_block_more_output = None
7
8     #####
9     ## Custom Code Start
10 #####
11
12     # Write your custom code here...
13
14     #####
15     ## Custom Code End
16 #####
17
18     phantom.save_run_data(key="my_end_block:more_output", value=json.dumps(my_end_block_more_output))
19
20     return
  
```

The code highlights two specific lines of code in orange boxes:

- Line 4: `my_start_code_my_output = json.loads(phantom.get_run_data(key="my_start_code:my_output"))`
- Line 18: `phantom.save_run_data(key="my_end_block:more_output", value=json.dumps(my_end_block_more_output))`

# Global Vars the SOAR Way (Sort Of)

- `phantom.save_run_data( )`
  - Stores your data with the given key
- `phantom.get_run_data( )`
  - Retrieves your data from the given key
- Can also be used as the input parameter of a subsequent action



# Format Loops

Well this is just not intuitive

splunk> .conf23

# Format Input Looping

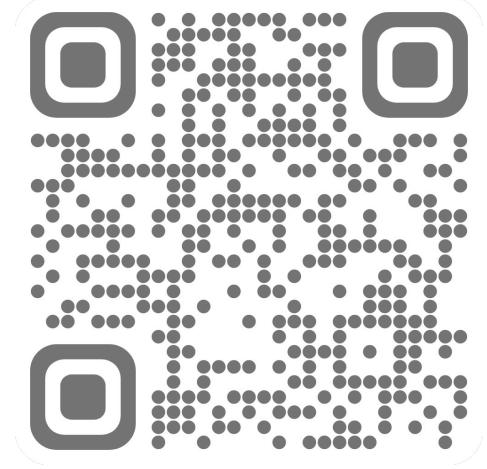
Making lists out of not lists

## Standard Format

---

Hello {0},  
The SOC has identified some  
suspicious activity using your  
enterprise credentials: {1}

As part of the response to this  
discovery, your account password is  
going to be reset etc etc blah blah  
blah it's going to be a bad day for  
you



# Format Input Looping

Making lists out of not lists

## Standard Format

---

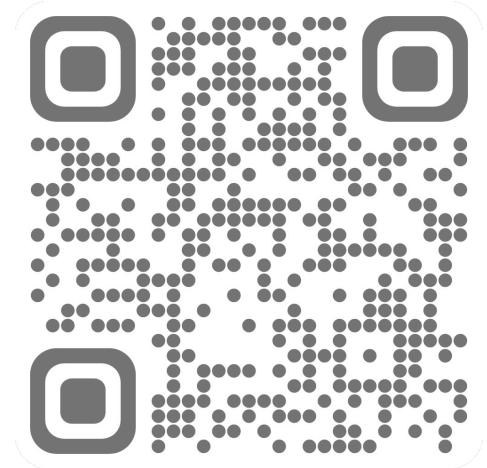
Hello {0},  
The SOC has identified some  
suspicious activity using your  
enterprise credentials: {1}

As part of the response to this  
discovery, your account password is  
going to be reset etc etc blah blah  
blah it's going to be a bad day for  
you

## Loop Format

---

%%  
{0}  
%%



# Format Input Looping

Making lists out of not lists

## Standard Format

---

Hello {0},  
The SOC has identified some  
suspicious activity using your  
enterprise credentials: {1}

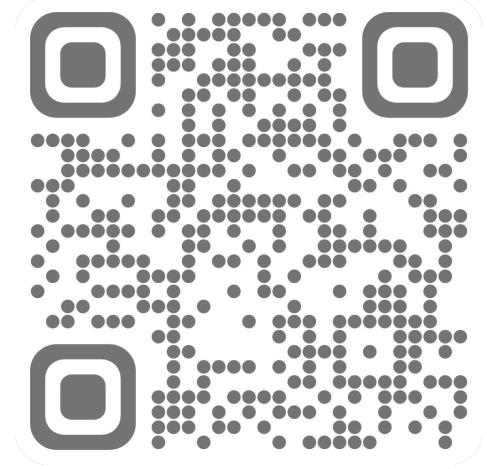
As part of the response to this  
discovery, your account password is  
going to be reset etc etc blah blah  
blah it's going to be a bad day for  
you

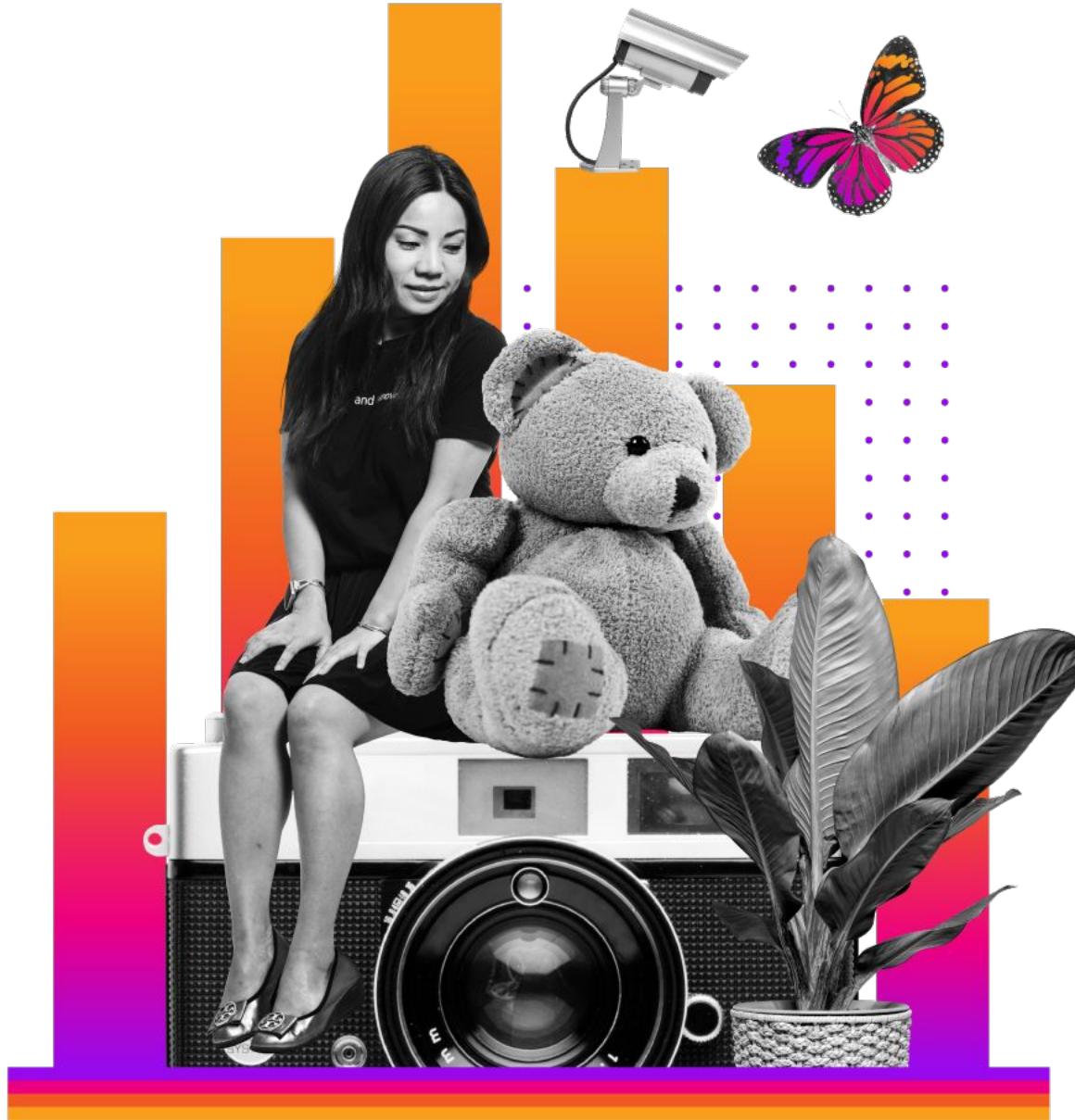
## Loop Format

---

Hello {0},  
The SOC has identified some  
suspicious activity using your  
enterprise credentials:  
%%  
Suspicious application: {1}  
%%

Please remove these items ASAP.





# Utility Functions

Wield the power of reusable code!

splunk> .conf23

# Inputs: List vs Item

Input variables need an input type

## List

- The utility is run just one time
- Useful for taking care of everything with code

Name  
my\_utility

Description (Optional)

**Input**

Variable\*  
my\_variable

Input Type List Item

## Item

- The utility is run once for each value
- Useful for taking a more SOAR-like approach

ID	Label	Name	Severity	Created By
5	event	Artifact 5	MEDIUM	admin
	Name	Artifact 5	Start Time	May 22nd at 10:15 pm
	Label	event	Created	May 22nd at 10:15 pm
	Created by	admin	Type	N/A
	Source ID	6cba0991-3a09-4766-88a5-d10e4d1d1923	Severity	Medium
Details				
data_example				123
4	event	Artifact 4	MEDIUM	admin
	Name	Artifact 4	Start Time	May 22nd at 10:10 pm
	Label	event	Created	May 22nd at 10:10 pm
	Created by	admin	Type	N/A
	Source ID	3ca5ffab-54e8-4f34-ad1b-155512d6f176	Severity	Medium
Details				
data_example				abc

1 ⓘ Test Scope ⓘ All Artifacts ⓘ Run as current user ⓘ

Copy Status: Done

```

May 30, 13:20:48: my_list_utility() called
May 30, 13:20:48: phantom.collect2(): called for datapath['filtered-data:filter_3:condition_1:artifact:*.cef.data_example','filtered-data:filter_3:condition_1:artifact:*.id'], scope: None and filter_artifacts: None
May 30, 13:20:48: phantom.get_run_data() called for key filtered-data:filter_3:condition_1
May 30, 13:20:48: phantom.get_run_data() called for key filtered-data:filter_3:condition_1
May 30, 13:20:48: phantom.collect2(): Classified datapaths as [-DatapathClassification:FILTERED_ARTIFACT:3->]
May 30, 13:20:48: phantom.collect(): called with datapath as LIST of paths, scope='all' and limit=2000. Found 2 TOTAL artifacts
May 30, 13:20:48: phantom.custom_function(): The custom function "local/my_utility_1" is being called with parameters: [['my_input': ['abc', '123']]]
May 30, 13:20:48: metrics: Playbook_id:131, run_id:33, container: 1, function: on_start. TIME_TAKEN 8ms
May 30, 13:20:48: Input Type: List
May 30, 13:20:48:
[
  "abc",
  "123"
]
May 30, 13:20:48: metrics: Playbook_id:131, run_id:33, container: 1, function: Execute Custom Function. TIME_TAKEN 35889104ms
May 30, 13:20:48: CustomFunctionRun with id=21 SUCCEEDED
May 30, 13:20:48: Completed Custom Function run: 'my_list_utility' (id: 21)
May 30, 13:20:48: calling cf my_list_utility's callback function 'my_item_utility()'
May 30, 13:20:48: my_item_utility() called
May 30, 13:20:48: phantom.collect2(): called for datapath['filtered-data:filter_3:condition_1:artifact:*.cef.data_example','filtered-data:filter_3:condition_1:artifact:*.id'], scope: None and filter_artifacts: None
May 30, 13:20:48: phantom.get_run_data() called for key filtered-data:filter_3:condition_1
May 30, 13:20:48: phantom.get_run_data() called for key filtered-data:filter_3:condition_1
May 30, 13:20:48: phantom.collect2(): Classified datapaths as [-DatapathClassification:FILTERED_ARTIFACT:3->]
May 30, 13:20:48: phantom.collect(): called with datapath as LIST of paths, scope='all' and limit=2000. Found 2 TOTAL artifacts
May 30, 13:20:48: phantom.custom_function(): The custom function "local/my_utility_2" is being called with parameters: [['my_input': 'abc'], {'my_input': '123'}]
May 30, 13:20:48: metrics: Playbook_id:131, run_id:33, container: 1, function: my_item_utility. TIME_TAKEN 45180904ms
May 30, 13:20:48: finished cf my_list_utility's callback function 'my_item_utility'
May 30, 13:20:48: Input Type: Item
May 30, 13:20:48: abc
May 30, 13:20:48: Input Type: Item
May 30, 13:20:48: 123
May 30, 13:20:48: metrics: Playbook_id:131, run_id:33, container: 1, function: Execute Custom Function. TIME_TAKEN 26647984ms
May 30, 13:20:48: CustomFunctionRun with id=22 SUCCEEDED
May 30, 13:20:48: Completed Custom Function run: 'my_item_utility' (id: 22)
May 30, 13:20:48: cf run 'my_item_utility' did not have any callback. The custom function is now marked completed
May 30, 13:20:48:

Playbook 'conf playbook' (playbook id: 131) executed (playbook run id: 33) on events 'test event'(container id: 1).
Playbook execution status is 'success'
Total actions executed: 0

May 30, 13:20:48: on_finish() called
May 30, 13:20:48: metrics: Playbook_id:131, run_id:33, container: 1, function: on_finish. TIME_TAKEN 5781272ms

```

Python Playbook Editor

# Utility Output Formatting

Standard vs Modified

1

**The “outputs” variable is a dict**

```
outputs = {}
```

1

**The “outputs” variable is a list**

```
outputs = []
```

# Utility Output Formatting

Standard vs Modified

1

**The “outputs” variable is a dict**

```
outputs = {}
```

2

**Keys and values added to outputs**

```
outputs['fruit'] = 'apple'
```

1

**The “outputs” variable is a list**

```
outputs = []
```

2

**Keys and values added to a dict**

```
outputs.append({'fruit': 'apple'})
```

# Utility Output Formatting

Standard vs Modified

1

**The “outputs” variable is a dict**

```
outputs = {}
```

2

**Keys and values added to outputs**

```
outputs['fruit'] = 'apple'
```

3

**Output paths are var names**

```
fruit
```

1

**The “outputs” variable is a list**

```
outputs = []
```

2

**Keys and values added to a dict**

```
outputs.append({'fruit': 'apple'})
```

3

**Output paths are \*. var names**

```
*.fruit
```

# Utility Output Formatting

Standard vs Modified

1

**The “outputs” variable is a dict**

```
outputs = {}
```

2

**Keys and values added to outputs**

```
outputs['fruit'] = 'apple'
```

3

**Output paths are var names**

```
fruit
```

4

**Datapaths point to the literal values**

```
'apple'
```

1

**The “outputs” variable is a list**

```
outputs = []
```

2

**Keys and values added to a dict**

```
outputs.append({'fruit': 'apple'})
```

3

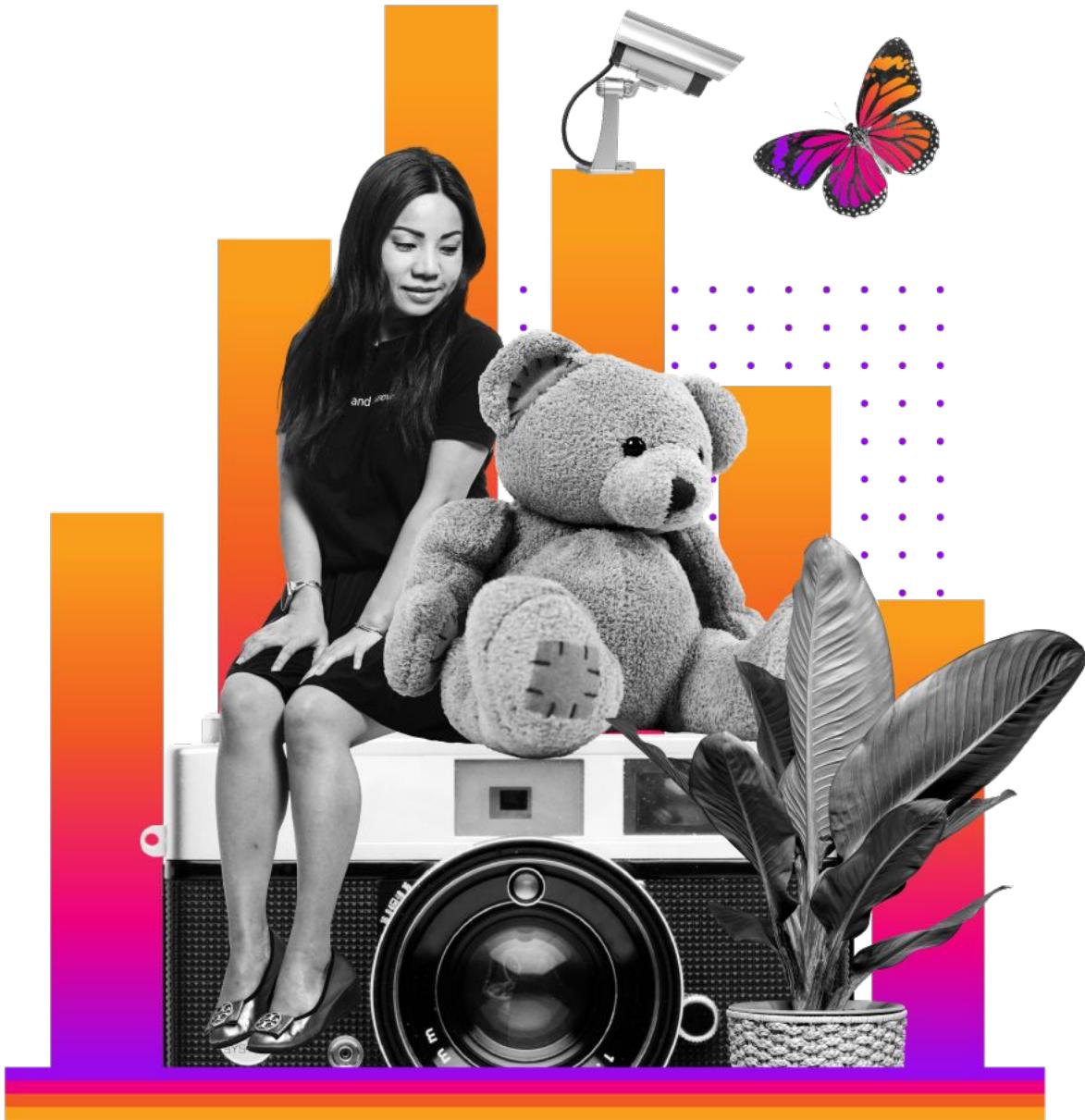
**Output paths are \*. var names**

```
*.fruit
```

4

**Datapaths point to list of values**

```
[ 'apple', 'banana', 'cherry', 'durian' ]
```



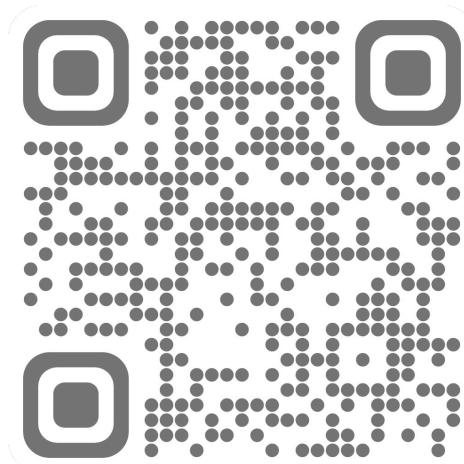
# Global Utilities

Let's get a little crazy

splunk> .conf23

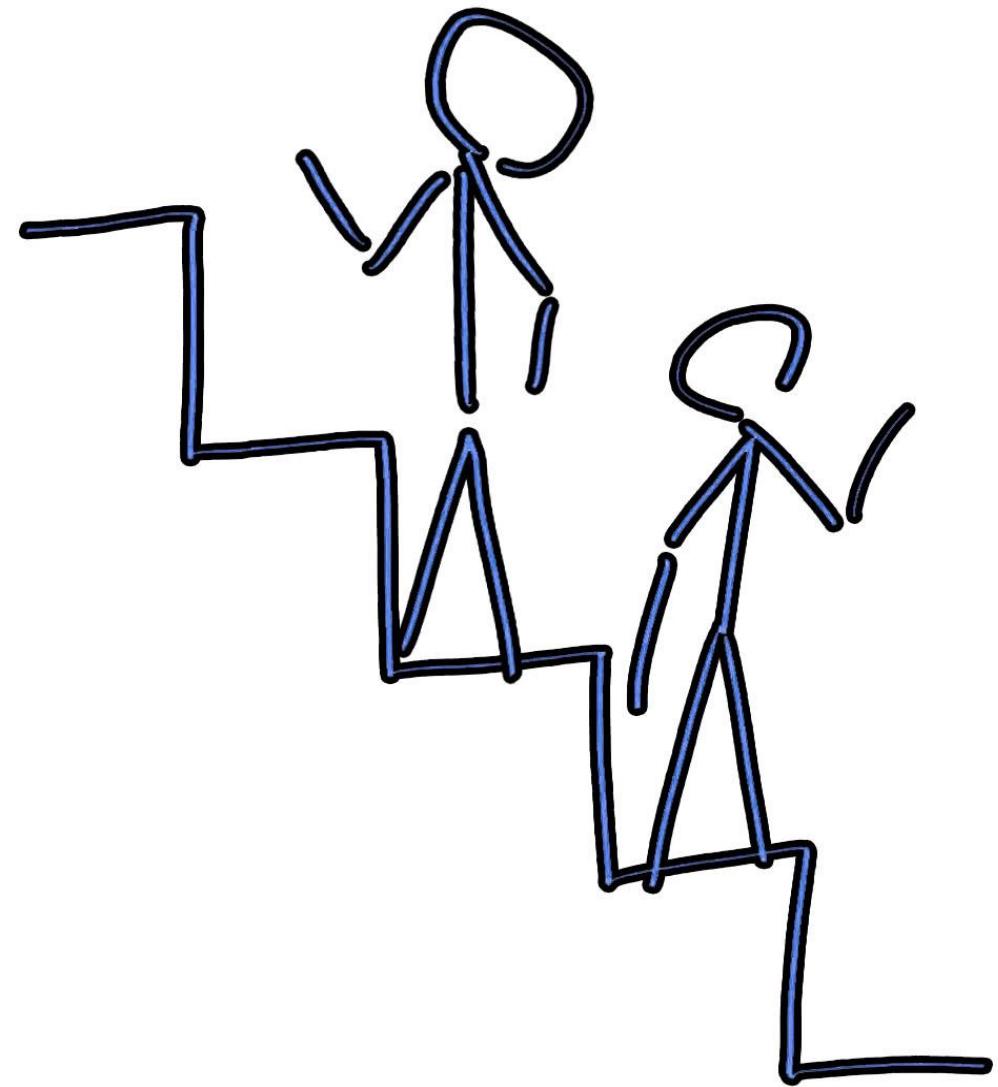
# Hijacking a Utility Block

- Don't cross the streams



The screenshot shows the Splunk SOAR interface. At the top, there's a header with the Splunk logo and the word "SOAR". Below it, a sub-header says "Edit Custom Functions" and "Draft Mode OFF". A "Name" field is filled with "Global\_Util". There's a "Description (Optional)" section with an empty text area. Under "Input", there's a "ADD INPUT" button. Under "Output", there's a "ADD OUTPUT" button. The main area contains a code editor with Python-like syntax:

```
1 def Global_Util(**kwargs):
2     """
3     Returns a JSON-serializable object that implements the configured data paths:
4     """
5     """
6     ##### Custom Code Goes Below This Line #####
7     return
8
9     import json
10    import phantom.rules as phantom
11    import ipaddress
12
13    # Reloads the Global_Util module to reflect new changes
14    def reload_module(module):
15        from importlib import reload
16        reloaded_module = reload(module)
17
18    # Returns the username of the user who executed the current playbook run
19    def current_playbook_run_username():
20        user_id = phantom.get_current_playbook_info()[0]['effective_user_id']
21        user_url = 'https://127.0.0.1/rest/ph_user/{0}'.format(user_id)
22        response = phantom.requests.get(user_url, verify=False)
23        if response.status_code == 200:
24            try:
25                content = json.loads(response.text)
26                return content['username']
27            except Exception as e:
28                return("Exception thrown while getting username: {}".format(e))
```

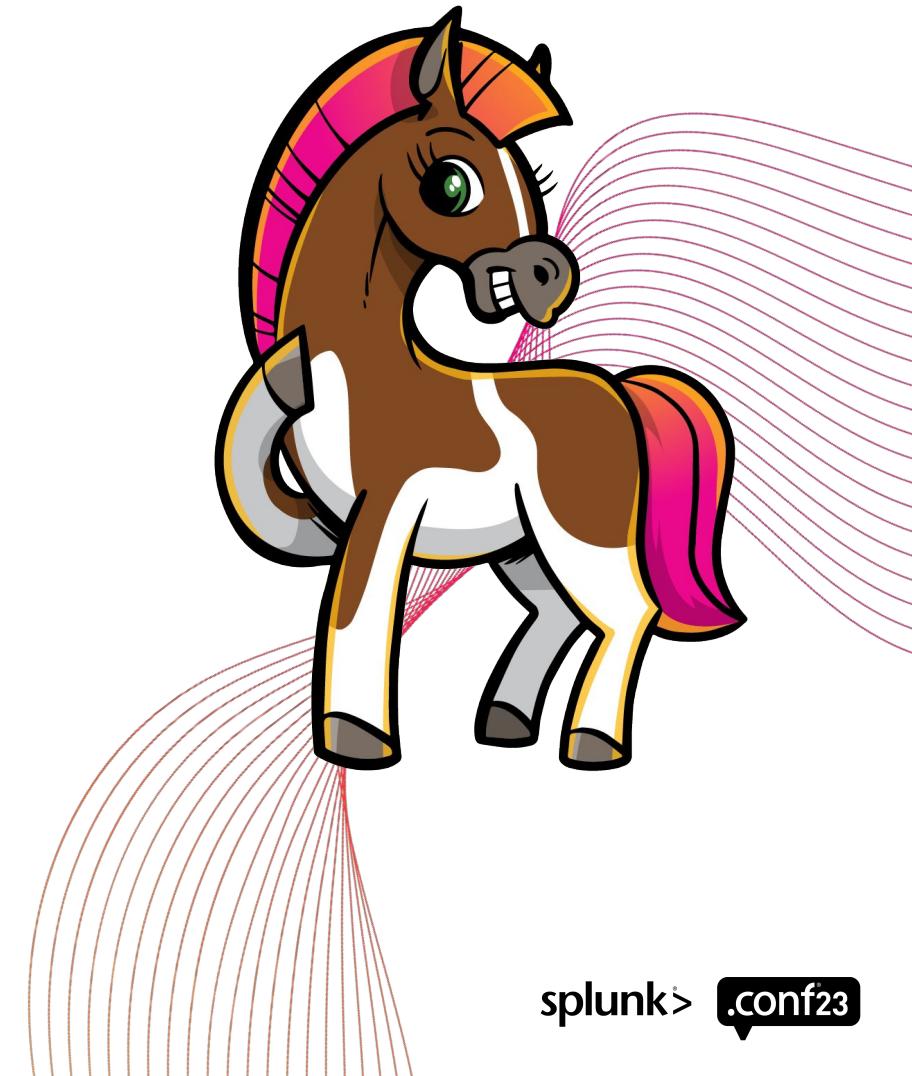
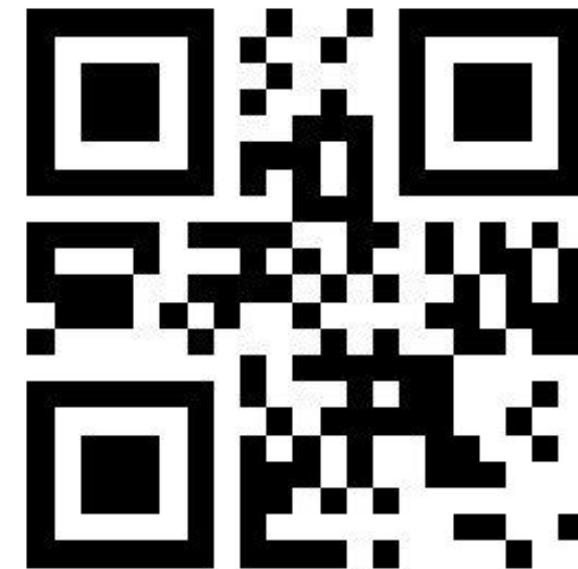


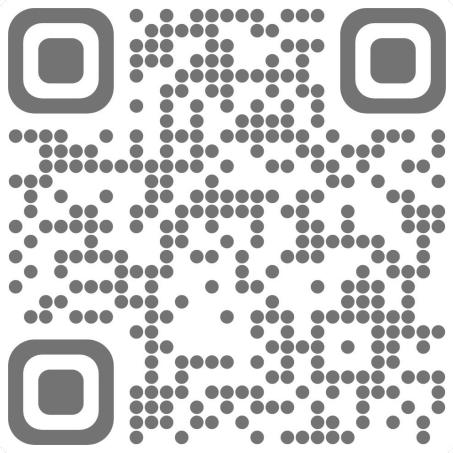
# .conf23 Learning Quest

Explore all .conf23  
has to offer!



**SCAN ME** in the  
.conf23 mobile  
app game



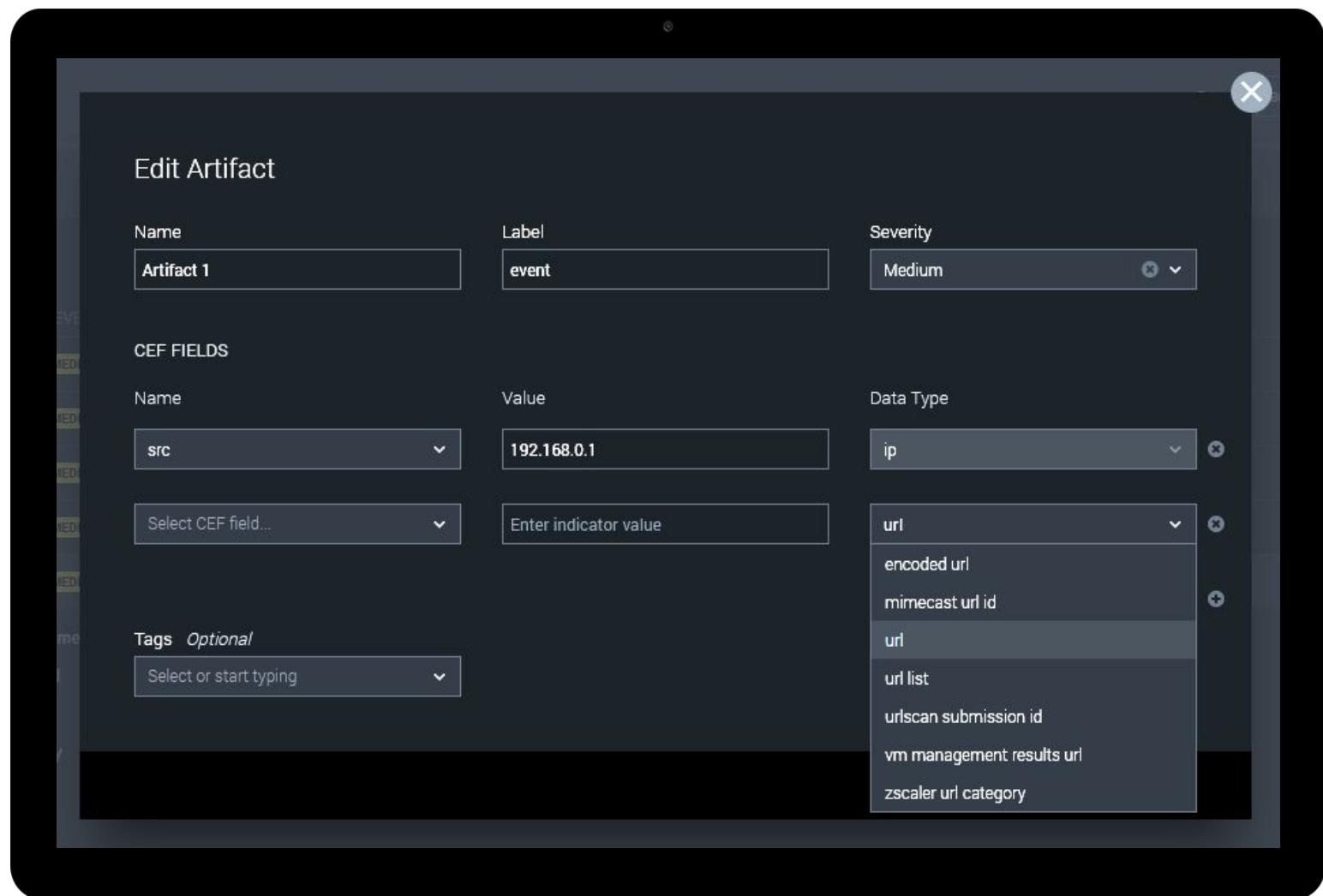


# Thank You



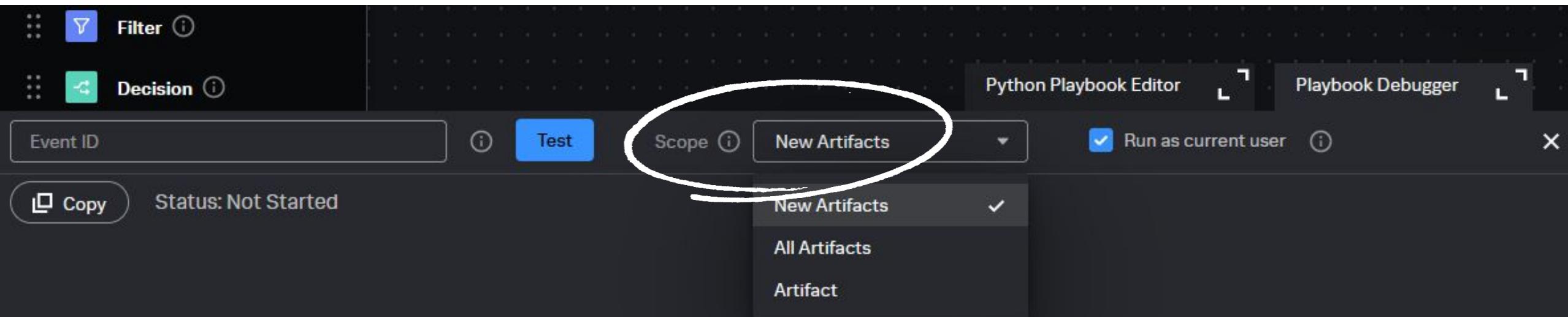
# Data Type

- Contextual Actions
  - Example: the drop-down arrow next to the field value in an artifact
- Associates a given field with app-configured action input types

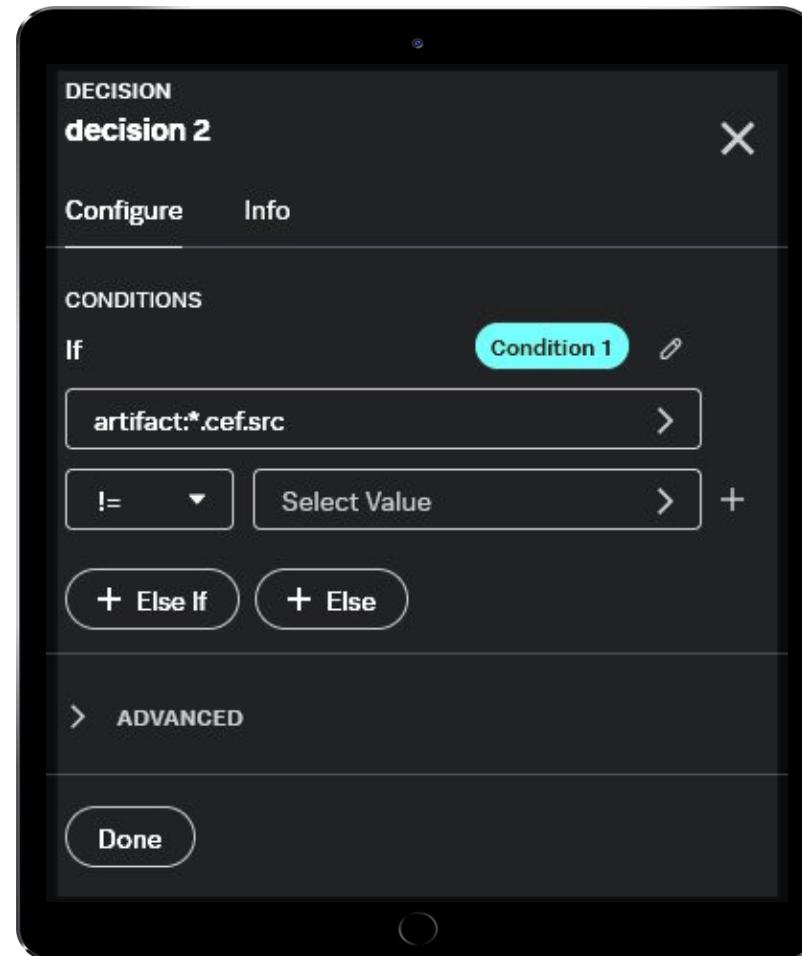


# Mind the Scope!

Why did your playbook only work the first time?



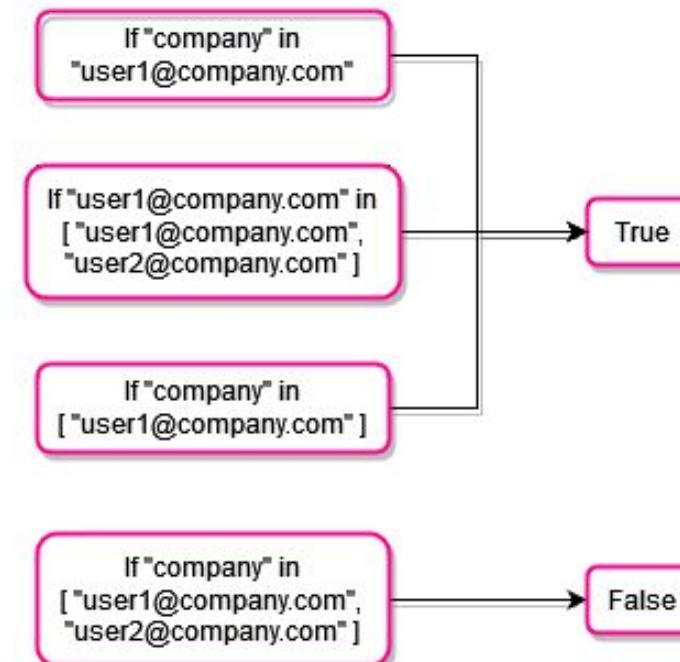
# Decision Examples



Proceed only if the given field exists

Be aware of how lists are processed

- Strings and Multi-element Lists are processed the same as they are in Python
- Splunk SOAR does you a favor with Single-element Lists (sometimes confusing)



Configure   Info   Stats

Asset

calculator

INPUTS

operand\_1\* ⓘ

artifact:\*.cef.box\_lengths >

operand\_2\* ⓘ

artifact:\*.cef.box\_widths >

> ADVANCED

# So You Would Think...

But it doesn't work like that

More than one multi-valued input in an action will nest those inputs instead of generating tuples

2x multi-valued inputs with 3 values each will execute this action 9 times

Technically your data will be there because every combination of fields will be executed

```
for length in artifact:*.cef.box_lengths:  
    for width in artifact:*.cef.box_widths:  
        RunAction(length, width)
```