merlin cyber

How to Develop Integrations with Swimlane

By Chris Calavas

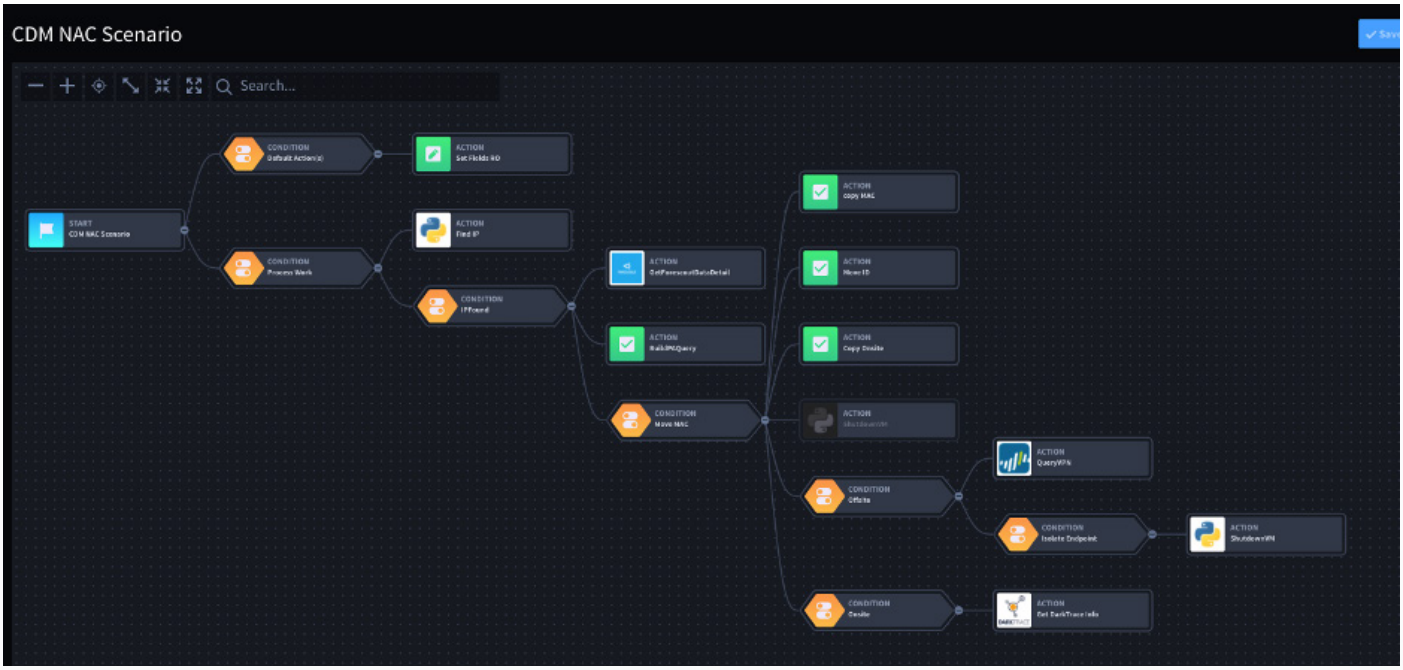
Solution Brief

Many security tools grant access to their analytics engine and data repositories through API interfaces. These interfaces allow users to access these disparate data sources and combine them so that engineers can glean better insight. Swimlane, a powerful security orchestration, automation, and response (SOAR) platform, can leverage these APIs using Python and other languages to integrate tools to force multiply their intended capabilities. This solution brief:

- Gives you a crash course on what to consider when designing within Swimlane
- Describes how to set up an essential task from integration as well as a primary customized task utilizing Python
- Includes code examples to demonstrate the process of integration task development and discusses some advanced techniques for processing multiple entry results

A Crash Course in Swimlane

Swimlane provides an orchestration framework that leverages data to coordinate actions based upon one or more events. It uses a one-to-many condition/action pair strategy for evaluation and execution purposes, unlike other security tools that leverage dependent state machines for deterministic model evaluation. The power that comes from this strategy results in time savings that translate to cost savings through automation.



1: Swimlane creates a record, and two paths start: database insertion and a workflow analysis

Swimlane acts as a single-table database of information that integration tasks may act upon from a pure data retention perspective. The platform allows users to reference data from multiple applications to derive relationships between siloed applications. Actions can occur without data, but insertions within the Swimlane are what drive the automation workflow.

The most impactful event in Swimlane is a triggering event, like a record insertion, timed trigger, or a manual update, which assesses the data record as it exists and then processes the application workflow. The workflow's state conditions are considered from left to right, working down all paths simultaneously until reaching each path segment's end. It's important to note that workflow action can include updating record data and forking off another workflow evaluation process. The mechanism to re-evaluate a workflow path based upon the dynamic update of a record is a powerful feature since not every finite condition to consider must be determined upfront. Swimlane workflow designs need to embrace an iterative design pattern to determine if additional workflow paths should be added to accommodate the resulting action data. These actions include but are not limited to tasks encapsulated with Swimlane integrations.

Swimlane Integrations

Swimlane supports more than 250 pre-defined integrations that enable faster time to deployment. Developers can find these integrations in the Swimlane AppHub (https://apphub.swimlane.com/how-to).

These are the basic steps for getting data:

1. Install the integration of interest (download from Swimlane's AppHub)
2. Create an asset for the integration
   a. The essential criteria typically required are a URL, access information, and some other minor details
3. Create the task from the integration
   a. Choose the implemented task
   b. Give it a name
   c. Define how the task gets input data (via static field, record from Swimlane, etc.)
   d. Attach the asset to the task
   e. Define what fields from the output are necessary

The fundamental process to determine what the task can report on is to allow the output record to assign all data to new fields (where fields that look like data fields are changed to be data types). Using this method, you can determine what type of data is available via the task and then choose to modify the actual records included within the application.

Example of AppHub content: Darktrace plug-in

Darktrace is a network threat detection and analysis tool that uses ML techniques to determine threats within an environment based on asset behavior anomalies. Isolating a risk associated with one or more assets and corroborating that information with other integrations with Swimlane validates the accuracy or severity of the threat and determines the best course of action using a remediation playbook instrumented within Swimlane. Additionally, Darktrace verifies historical weaknesses among transient assets. Risk is identified through the vulnerability and through the frequency with which an asset breaches a threat model threshold over a specific time.

A practical Swimlane workflow using Darktrace Plug-in assesses the risk of an endpoint prior to admitting it on the network. For example, the endpoint can be blocked from connecting on the network if Darktrace determines that it has a threat model breach.

Third-party integrations can be powerful when used in combination, but they typically do not encapsulate the total robustness of the tool with which it connects with. To instrument even more capabilities, Swimlane allows for creating internal API connections using Python's powerful capabilities.

Integrations with Python

Besides pre-defined integrations in AppHub, Swimlane uses the Python scripting language to expand the breadth of integrations and actions available to the organization.

These are our recommended prerequisites:

1. A target application that supports a RESTful API
2. JSON (JavaScript Object Notation) experience
3. Some understanding of Python
4. PyCharm Community Edition (Developers should use an external development toolkit to debug before adding source code to Swimlane)

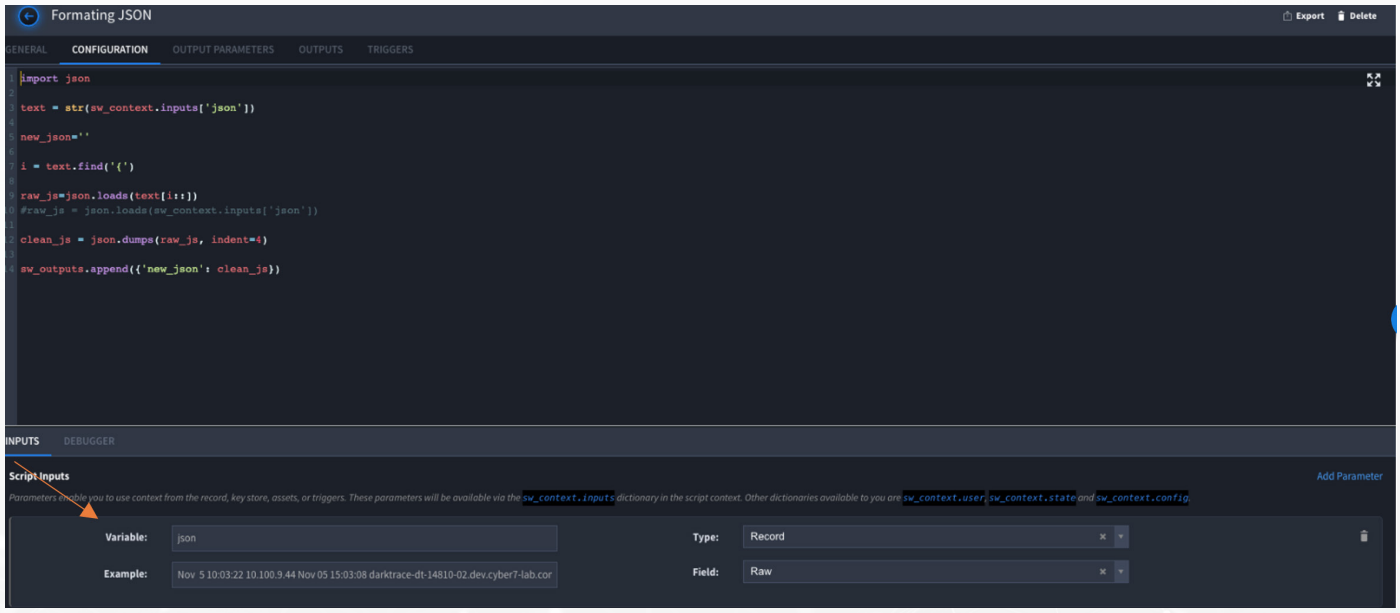
With the first three basic requirements, you can build almost any integration. Start by defining your key-value pairs, which will house the private URL, username password, JSON Web Token, etc., and which you will need to access the API. You will then use these keypairs in conjunction with some input directives to build an API GET command to get data from your service.

Our First Example

A very useful tool within Swimlane is to be able to parse JSON. Most of the data constructs are either key-value pairs of JSON raw data, so it becomes a very useful tool to develop. The following will use JSON Parsing as the basis for demonstrating how to create a custom task. Additional features a developer should consider expanding upon would include searching through and extracting elements from within the JSON structure.

Defining a task requires the identification of a data source. We will need to define a variable name that we can reference within our python task and link it to that data source. The process of linking a Swimlane application to a customized python task is done in the Task's Configuration tab. The input variable is defined in the section located on the bottom portion of the screen.

In the image below, the Inputs tab is the area of focus where a parameter has been added for use. The variable in this example is named "json". This is the reference that the code supporting the task can use as a source of data. The second part of this association is what data is linked to this variable named "json." The right side of the inputs section indicate that the "Type" of variable source is a "Record", which means that a field within this application will be used, and the field is literally named "Raw". That is the actual field name within our application.



2: Inputs section of the Configuration tab

Below, the Python variable "text" will receive data from the input parameter specified in the Inputs section. The Python function "sw_context" is the Swimlane function that provides data transfer between the application and integrations. The method "inputs" defines the direction of flow of data, in this case, into the task and uses the variable name "json" as the reference.

text = str(sw_context.inputs['json'])

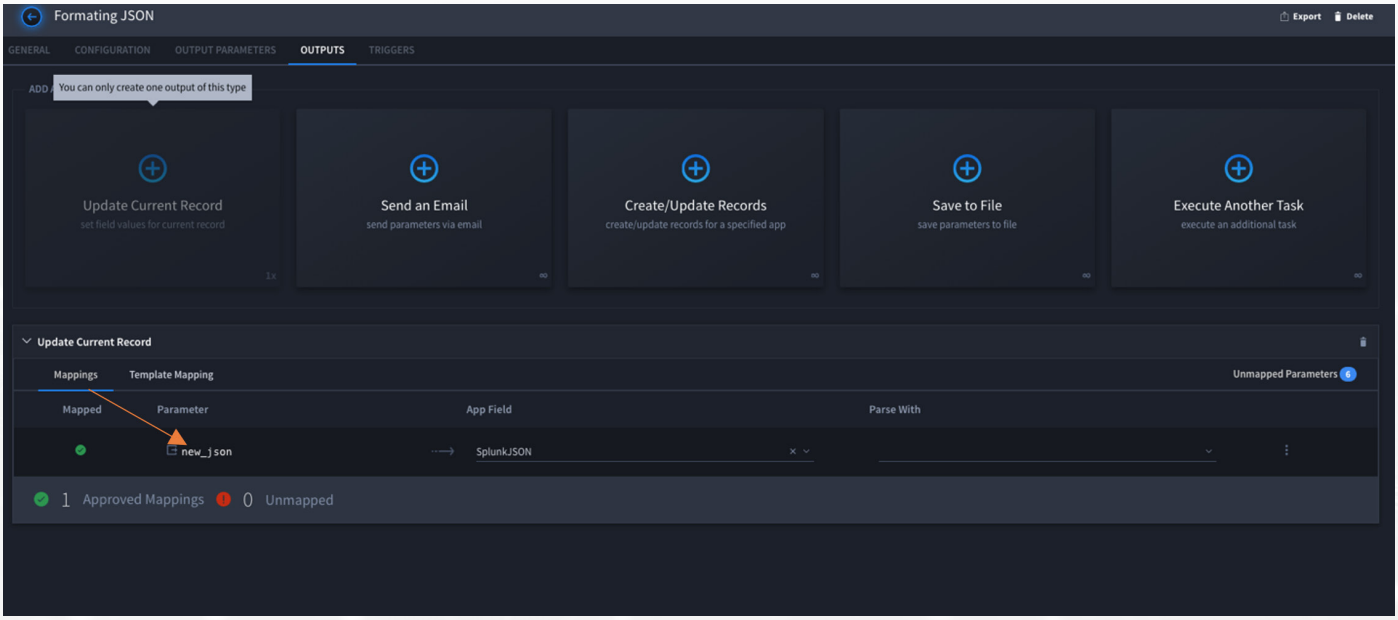
The following code snippet parses the JSON input and formats it for presentation value.

```
#Prepare the extraction process by removing any extraneous data and set the internal cursor to '{'
i = text.find('{')
#Export the rest of the data from the '{' to the end of the input string
raw_js=json.loads(text[i::])

#Format the json by setting the indent to 4 spaces
clean_js = json.dumps(raw_js, indent=4)
```

Similar to the way that the sw_context. Inputs did, we have sw_outputs.append pushes from this Python task back to the application.

```
sw_outputs.append({'new_json': clean_js})
```

After the code has been tested and the output returns the expected values, a final linkage must be made to define where the output of the task will be stored. This association is done in the "Outputs" tab. The following screen demonstrates that we are using the "new_json" Swimlane output parameter and associating it with a field named "SplunkJSON."



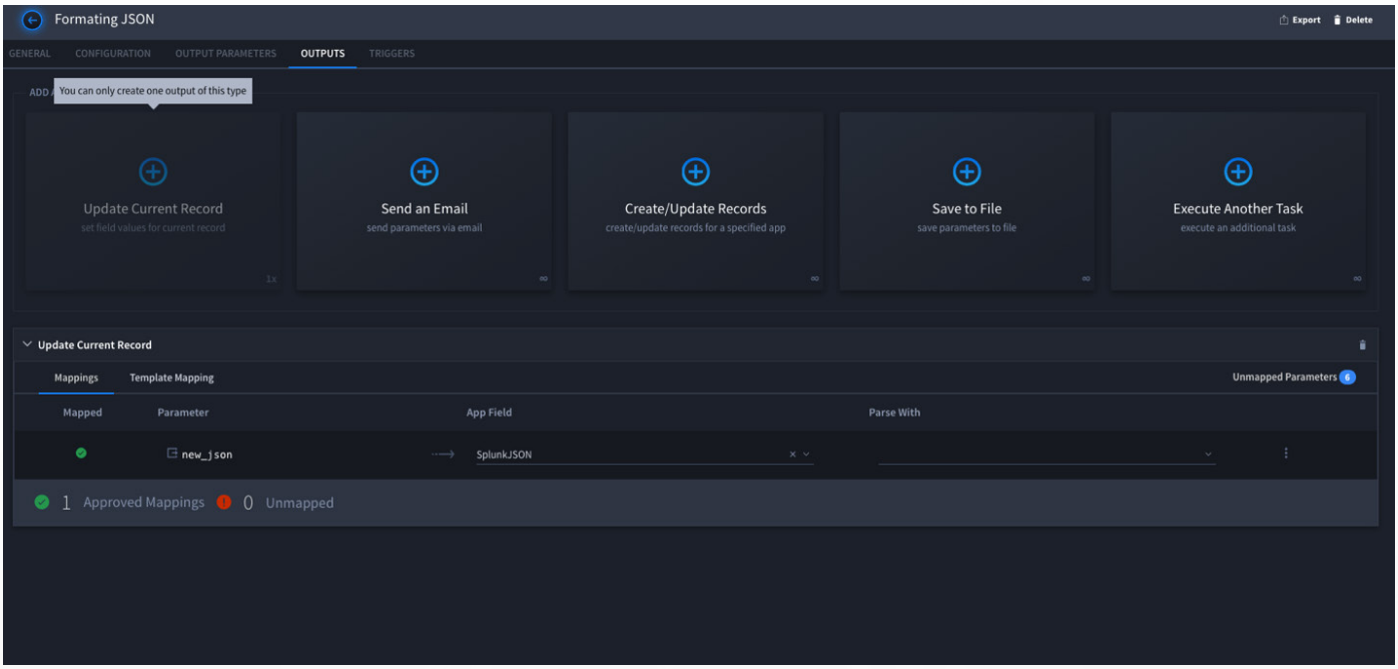
3: Example of output variable assignment in a Python task

These basic components are what is necessary to build a complete custom Python task – Swimlane Inputs, code, and Swimlane outputs. The number of tasks a developer can create is limitless. It is important to note that the larger the task, the more potential a performance bottleneck can be observed, for this customized task has the possibility of being executed many times within an application.

Below, the complete set of code used in the above example is provided in its entirety - parsing JSON data (generic):

```
##. For demonstration, use only
##. Merlin International – ccalavas@merlincyber.com 2020

import requests
import JSON

#These input parameters are passed via either Keystore, static, or field values
#The common methods are through using the sw_context object
text = str(sw_context.inputs['json'])

#Prepare a new variable
new_json=''

#Prepare the extraction process by removing any extraneous data and set the internal cursor to '{'
i = text.find('{')
#Export the rest of the data from the '{' to the end of the input string
raw_js=json.loads(text[i::])

#Format the json by setting the indent to 4 spaces
clean_js = json.dumps(raw_js, indent=4)

#present back to Swimlane the final output, using the sw_outputs object and the append method
sw_outputs.append({'new_json': clean_js})
```

Code overview - This code prepares the output to be placed into a JSON key-value pair structure that can be passed back to the Swimlane application database. Actual association to fields, however, does not happen automatically, and there are a few steps to perform to do this. The first step would be to define the inputs at the beginning of this Python module, and Swimlane outputs fields that will populate records with the data retrieved. This then can be used to populate a current record or create new records.



4: Output field selection mapping

In this example, we have one output field that we can then set up to write and update records. We can also automatically detect these fields in the task's output parameters tab to associate the task output with Swimlane record fields.

Designing Applications with Advanced Record Retrieval

Some API retrievals will generate multiple records. It might be advisable to create a separate application which only retrieves numerous records and then uses a reference field to tie them back to the main application. This is true if the relationship is one-to-many. The technique to make the cross-application match requires using reference fields which can then be used to tie, for example, IP Address, between the two applications.

Other than this type of one-to-many relationship, previously unassociated data can now also be combined by using reference data and appropriate key fields. This powerful feature can tie assets to historical archives of network events and ITSM records, thus providing a robust history of a device's history and how its health is reflected within the current environment.

Final Thoughts

Swimlane is a powerful SOAR tool that is highly customizable in both its workflow and content integration capabilities. AppHub is a great source of integrations and community-based dialog to get a developer to build great applications for their organization. In recent years, more tools provide interfaces that provide RESTful (Web) connectivity, making them more accessible and available to integrate with. Swimlane empowers the developer to create highly customizable applications in a short amount of time, making Swimlane an invaluable tool in the organization.

About The Author

Chris Calavas is Vice President of IoT/Cyber Solutions for Merlin Cyber. He has more than 25 years of cybersecurity engineering experience developing solutions for both commercial and government customers. Chris holds a Master of Science in IT Management degree from George Mason University.