Splunk® SOAR (On-premises)
Build Playbooks with the Playbook Editor 5.4.0

Generated: 11/04/2022 7:13 am

Table of Contents

Table of Contents

Introduction to Splunk SOAR (On-premises) Playbooks

Use playbooks to automate analyst workflows in Splunk SOAR (On-premises)

Create a playbook in Splunk SOAR (On-premises) to automate security workflows so that analysts can spend more time performing analysis and investigation. The playbook editor provides a visual platform for creating playbooks without having to write code.

To define a workflow that you want to automate, link together a series of actions that are provided by apps. An app is third-party software integrated with Splunk SOAR (On-premises). For example, you can integrate MaxMind as an app, which provides a `geolocate ip` action, or integrate Okta as app to provide actions such as `set password` or `enable user`. The actions available for use in your playbooks are determined by the apps integrated with Splunk SOAR (On-premises).

After you create and save a playbook in Splunk SOAR (On-premises), you can run playbooks when performing these tasks in Splunk SOAR (On-premises):

- Triaging or investigating cases as an analyst
- Creating or adding a case to Investigation
- Configuring playbooks to run automatically directly from the playbook editor

The playbook editor has a minimum supported screen size of 1200px.

Python 3.9 impacts on apps: You must upgrade all custom apps to be compatible with with Python 3.9. If you don't, those apps might not run in the Python 3.9 environment. Existing Python 3.6 playbooks continue to work in the new Python 3.9 environment. If you use the terms "async" or "await" as names of variables, functions, or other pieces of code in your playbooks, a SyntaxError results. Rename anything named "async" or "await" in your playbooks.

Use playbooks to automate analyst workflows in Splunk SOAR (On-premises)

Create a playbook in Splunk SOAR (On-premises) to automate security workflows so that analysts can spend more time performing analysis and investigation. The playbook editor provides a visual platform for creating playbooks without having to write code.

To define a workflow that you want to automate, link together a series of actions that are provided by apps. An app is third-party software integrated with Splunk SOAR (On-premises). For example, you can integrate MaxMind as an app, which provides a `geolocate ip` action, or integrate Okta as app to provide actions such as `set password` or `enable user`. The actions available for use in your playbooks are determined by the apps integrated with Splunk SOAR (On-premises).

After you create and save a playbook in Splunk SOAR (On-premises), you can run playbooks when performing these tasks in Splunk SOAR (On-premises):

- Triaging or investigating cases as an analyst
- Creating or adding a case to Investigation
- Configuring playbooks to run automatically directly from the playbook editor

The playbook editor has a minimum supported screen size of 1200px.

Python 3.9 impacts on apps: You must upgrade all custom apps to be compatible with with Python 3.9. If you don't, those apps might not run in the Python 3.9 environment. Existing Python 3.6 playbooks continue to work in the new Python 3.9 environment. If you use the terms "async" or "await" as names of variables, functions, or other pieces of code in your playbooks, a SyntaxError results. Rename anything named "async" or "await" in your playbooks.

Use the Playbook Editor to create and view playbooks to automate analyst workflows

Create a new playbook in Splunk SOAR (On-premises)

Perform the following tasks to create a new playbook in Splunk SOAR (On-premises):

1. Click the menu bar, then select **Playbooks**.
2. Click **+ Playbook** to create a new playbook.
3. Select either the **Automation** or **Input** type playbook. Select an automation playbook to run a playbook automatically based on triggers. Automation playbooks can also have outputs, and can be used as sub-playbooks. Select an input playbook to accept configured inputs to run, and provide outputs. Input playbooks can only be used as sub-playbooks, and can't be triggered automatically as an independent playbook.

The **Start** and **End** blocks are pre-populated on the editor. All playbooks must start with the **Start** block. Regardless if playbooks end with the **End** block, the end/on_finish function is always called at the end of a playbook's execution.

Specify a name for the playbook.

- Playbooks in the same repository cannot have the same name. Playbooks in different repositories can have the same name.
- As a best practice, do not use personally identifiable information in the names of playbooks.

Once you have created your playbook, you can click the auto-arrange playbook icon to align the blocks.

Use the zoom to fit icon, or click the icons with the plus and minus signs to zoom in or zoom out. For keyboard shortcuts, see Use keyboard shortcuts in the playbook editor.

Next, see Add a new block to your Splunk SOAR (On-premises) playbook for instructions on how to add a new block and begin constructing your playbook.

Add outputs to Automation and Input playbooks

You can add outputs to both Automation and Input playbooks. Automation playbooks can be run both independently and as a sub-playbook. Input playbooks can only be run as a sub-playbook. Outputs will be available to use by the parent playbook that calls a sub-playbook with outputs. To add outputs to a playbook, follow these steps:

1. Create either an Automation or Input playbook. See "Create a new playbook in Splunk SOAR (On-premises)".
2. Click the **End** block to access the output configuration panel.
3. Enter a name for the output in the **Output Variable Name** field. The name can only contain A-Z, a-z, 0-9, spaces, or underscores. The name must be a valid Python identifier and cannot start with a zero.
4. (Optional) Enter help text or a description in the **HelpText/Description** field. This appears as help text on the playbook listing page and when selecting a playbook to run as a sub-playbook.
5. (Optional) Click the **Output** field and search for and select an **Output** datapath from the list. Click **Enter** to go to the next result or use the ⌄ ⌃ icons to navigate results. You can also expand or collapse the lists by using the icons. You can add multiple output datapaths per output.
6. (Optional) Select a **Data Type** for the output. If you select a data type, downstream blocks can filter on data type to know whether the output is compatible or not. The **Data Type** automatically populates based on the first output datapath you selected.

7. (Optional) Create a custom datapath if the datapath you need isn't available. When you add a custom datapath, it is only available for the block you add it to. To see an example of a custom datapath, see Example: Add a custom datapath to a Splunk SOAR (On-premises) playbook block. To create a custom datapath, follow these steps:
    1. Hover over a datapath field title and click **+**.
    2. Enter the datapath name.
    3. Select either **Key** or **List** from the drop-down menu. Use **Key** to use one value, and use **List** to use a list of values. Using **List** adds a **.*** value to the datapath and it appears as <list_name [] > with datapaths nested below it in the datapath picker. To add more values to your **List**, click the **+** icon under the top value of the list.
    4. Click **Save**.
8. Click **Done**.
9. Click **Save**.
10. (Optional) Click **+** to add another output. You can add a maximum of 10 outputs per playbook.
11. Add a block to your Splunk SOAR (On-premises) playbook. If you choose to add a playbook block, and the playbook has outputs, the **Synchronous** switch must be on to access the outputs. For more information, see Add a new block to your Splunk SOAR (On-premises) playbook.
12. Enter a name for the playbook in the **Playbook Name** field.
13. Click **Save** and enter a comment about the playbook.

Once you save the playbook, it appears on the playbook listing page with the type and outputs listed.

Add inputs to an Input playbook

Use Input playbooks to pass data between playbooks and sub-playbooks. Input playbooks accept configured inputs to run, and can provide outputs. Input playbooks can only be used as sub-playbooks, and can't be triggered automatically as an independent playbook. As Input playbooks are only used as sub-playbooks, Input playbooks can be more prescriptive without having to accommodate for all types of data in the notable making playbooks easier to develop and reuse. To add inputs to an Input playbook, follow these steps:

1. Create an Input playbook. See "Create a new playbook in Splunk SOAR (On-premises)".
2. Click the **Start** block to access the input configuration panel.
3. Enter a name for the input in the **Input Variable Name** field. The name can only contain A-Z, a-z, 0-9, spaces, or underscores. Input variable names must be unique.
4. (Optional) Enter help text or a description in the **HelpText**/**Description** field. This appears as help text on the playbook listing page and when selecting an Input playbook to run as a sub-playbook.
5. (Optional) Select a **Data Type** value from the list. The **Data Type** value you set is used to filter data when assigning data to a configured input.
6. (Optional) Create a custom datapath if the datapath you need isn't available. When you add a custom datapath, it is only available for the block you add it to. To see an example of a custom datapath, see Example: Add a custom datapath to a Splunk SOAR (On-premises) playbook block. To create a custom datapath, follow these steps:
    1. Hover over a datapath field title and click **+**.
    2. Enter the datapath name.
    3. Select either **Key** or **List** from the drop-down menu. Use **Key** to use one value, and use **List** to use a list of values. Using **List** adds a **.*** value to the datapath and it appears as <list_name [] > with datapaths nested below it in the datapath picker. To add more values to your **List**, click the **+** icon under the top value of the list.
    4. Click **Save**.
7. (Optional) Click **+** to add another input. You can add a maximum of 10 inputs to an Input playbook.
8. Add a block to your Splunk SOAR (On-premises) playbook. For more information, see Add a new block to your Splunk SOAR (On-premises) playbook.
9. Once you have added a block, select **playbook inputs** in the datapath picker for the block, usually found in the

Select Parameter field, and then select the input you want this block to use.

10. Click **Save**.
11. Enter a name for the playbook in the **Playbook Name** field.
12. Click **Save** and enter a comment about the playbook.

Once you save the playbook, it appears on the playbook listing page with the type and inputs listed.

Use an Input playbook as a sub-playbook

Once you have created an Input playbook, you can run it as a sub-playbook from an Automation playbook to avoid having to copy and maintain code in different places.

1. Create an Automation playbook.
2. Drag and drop the half-circle icon attached to any existing block in the editor. Select a **Playbook** block from the menu that appears.
3. Click the **Input** tab and select the playbook you want to run from the drop-down list.
4. Click in the input fields and assign the inputs datapaths from the drop-down list. Search for the datapaths you want to use and click **Enter** to go to the next result or use the icons to navigate results. You can also expand or collapse the lists by using the icons. If you assigned a **Data Type**, such as "ip", when configuring your inputs, you can filter the list by datapaths that have a **Data Type** of "ip" and toggle the filtering on or off using the "filter on ip" switch.
5. (Optional) Click the **Info** tab to view information about the playbook including the name, description, inputs, and outputs associated with the playbook.
6. (Optional) Toggle the Synchronous switch on to make this playbook wait for the called playbook to complete running before continuing. If this switch is left off, the playbook finishes executing without waiting for the called playbook to complete and you won't be able to access the inputs.
7. (Optional) Add any additional blocks to the playbook.
8. Click **Save**.

For more information, see Run other playbooks inside your playbook in Splunk SOAR (On-premises).

Sub-playbooks can't be called from Input playbooks.

Show input and output run data

Once a playbook is executed, you can view the inputs and outputs for that playbook in the Investigation page or Activity panel.

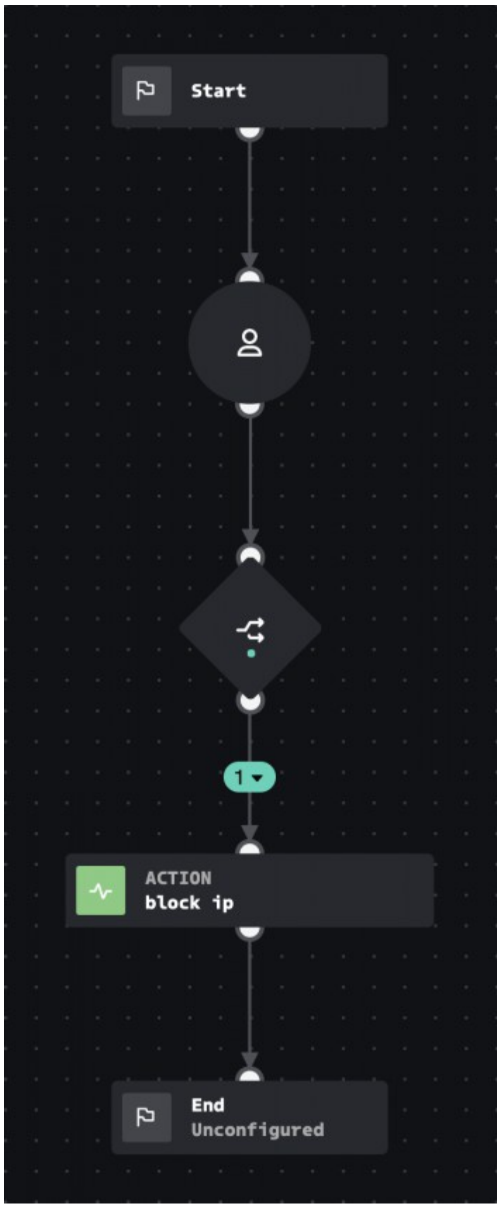
To see the input and output run data, complete the following steps:

1. Run a parent playbook with a sub-playbook that has inputs and outputs.
2. Click the playbook name in the Investigation page or Activity panel.
3. You'll see a "takeover" screen with information about the playbook run result, input, and output data

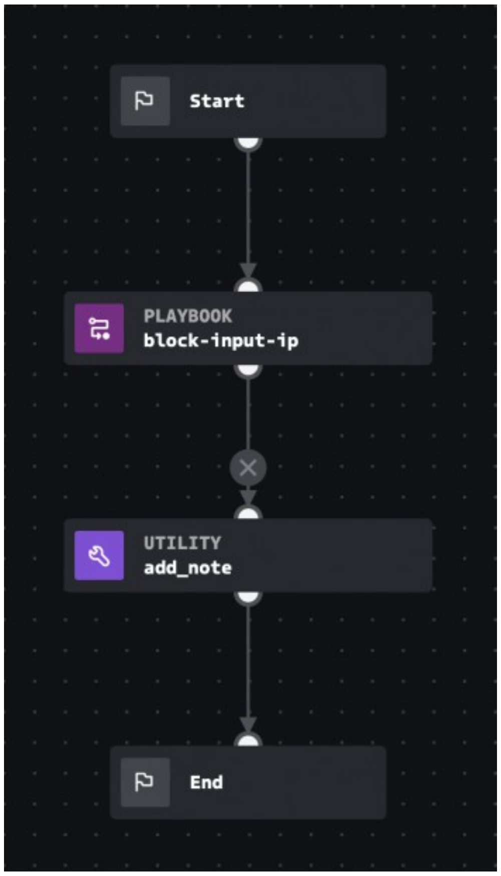
The results for a playbook without inputs and outputs still opens a takeover screen, but without any input and output results.

Example: Use inputs and outputs to block an IP address

Run an Input playbook as a sub-playbook to avoid having to copy and maintain code in different places. The following Input playbook uses an IP address as an input, and then a prompt block to ask a user whether to block the IP or not. A decision block is used next, where if the decision is to block the IP, then a block IP action block is used to block the IP and the playbook sets the status of the block IP action as an output.



In the following example, the Input playbook is used as a sub-playbook. The parent playbook passes the event src_ip datapath as an input to the sub-playbook, block-input-ip. The parent playbook then uses a utility block to add a note where the content of the note is the output of the block-input-ip playbook.



Add a new block to your Splunk SOAR (On-premises) playbook

To add a new block to a playbook:

1. Drag and drop the half-circle icon attached to any existing block in the editor. Select a block type from the menu that appears. Or, click on and drag and drop a block onto the editor from the list of block types.
2. Configure the block as needed. See the following table.
3. Click **Done** when you are finished configuring the block.
4. Connect your block either by dragging the half circle icon from a previous block to the half circle icon on your new block, or by dragging and dropping a new block onto an existing block. Each new block must be connected to a block before itself. For example if your playbook has a single action, it will connect to the **Start** block and the **End** block.

Playbook block type	Description
Action	Run an action provided by an app that is installed and configured in Splunk SOAR (On-premises). For example, you can use the MaxMind app to geolocate an IP address. See Add an action block to your Splunk SOAR (On-premises) playbook.
Playbook	Run an existing playbook inside your current playbook. See Run other playbooks inside your playbook in Splunk SOAR (On-premises).
Code	Process data with custom code. See Add custom code to your Splunk SOAR (On-premises) playbook with the code block.
Utility	Perform an action by making a utility call. See Set notable parameters in Splunk SOAR (On-premises) using the Utility block. The utility block is also where the custom functions live.
Filter	Filter the results of the previous block. For example, you can separate items that have a specific severity and perform a different set of actions on those items. See Use filters in your Splunk SOAR (On-premises) playbook to specify a subset of events before further processing.
Decision	Make a decision and perform different actions depending on the results of the previous block. For example, you can deny list all destination IPs that belong to a specific country. See Use decisions to send events to a specific downstream action in your Splunk SOAR (On-premises) playbook.
Format	Format the results of the previous block. For example, you can gather data, format that data in a specific way, and send an email. See Customize the format of your Splunk SOAR (On-premises) playbook content.
Prompt	Require a user to take action before proceeding to the next block. See Require user input using the Prompt block in your Splunk SOAR (On-premises) playbook.

Advanced settings

Follow these steps to configure advanced settings for a block.

To use Advanced settings, when configuring a block follow these steps:

1. Click **Advanced**.
2. Modify the advanced settings.

Setting	Block type	Description
Join Settings	Available for action, playbook, code, filter, decision, format, and prompt block types.	You can configure join settings when multiple incoming blocks that support the synchronous functionality are linked to any downstream block. All **Action**, **Prompt**, and **Manual Task** blocks run synchronously and playbooks can be toggled to run synchronously in the block configuration. See Run other playbooks inside your playbooks in Splunk SOAR (On-premises) for more information on the synchronous functionality.

Setting	Block type	Description
		Configure join settings from the downstream block. These settings determine whether or not you wait to execute the next block until the required upstream blocks finish running. Click the **required** checkbox if the action in the upstream block must be completed before this downstream block is run. The **required** checkbox is enabled by default.
Scope	Available for action, playbook, code, filter, utility, decision, format, and prompt block types.	Configure scope to determine how the artifact data passed into a block's API is collected. Collection occurs in the context of the current playbook. Setting the scope advanced setting on a playbook block doesn't change the scope of a child playbook. In child playbooks, scope only affects the collected artifact data that is passed in as inputs to the child playbook and the collection occurs before the child playbook is run. • Default: The artifact data for the block uses the same scope as the playbook. • New Artifacts: The artifact data for the block is collected for new events. • All Artifacts: The artifact data for the block is collected for all events. Specifying scope with Playbook and Utility blocks: • Playbook blocks: Scope is only relevant for Input playbooks, sometimes known as data playbooks. Scope is not relevant — and you cannot specify scope — for Automation playbooks. • Utility blocks: Scope is only relevant for utility blocks that have a datapath input. Scope is not relevant — and you cannot specify scope — when the utility block has a text input.
Action Settings	Available for action blocks.	Configure the action settings that a user must perform. Action settings are only available from an action block. • Reviewer: Select a user or group that must approve this action before the action runs. If you select a group or role, any user in that role can approve the action. • Delay Timer: Set a delay in minutes before the action runs. A clock icon is visible on the action block to show that a delay is configured.
Case-sensitive	Available for decision and filter blocks.	Select if you want the conditions evaluation to be case-sensitive, or case-insensitive. The default is case-sensitive.
Delimiter	Available for prompt and format blocks.	Specify an alternate separator to use when joining parameter values that result in a list together. The default separator is ",".
Drop None	Available for prompt and format blocks.	Select whether or not you want to drop the "None" values from the resulting lists of parameters. By default, the "None" values are included.

Add an action block to your Splunk SOAR (On-premises) playbook

Perform the following steps to add an **Action** block to a playbook.

1. Drag and drop the half-circle icon attached to any existing block in the editor. Select an **Action** block from the menu that appears. Actions available to you in the playbook editor are determined by the apps that are installed and configured on Splunk SOAR (On-premises). See Add and configure apps and assets to provide actions in Splunk SOAR (On-premises).
2. Select the action you want to configure, or enter an action name in the search field if you don't see the desired action listed.
3. (Optional) You can also filter the list of actions by action type. Select **By App** or **By Action**. Click **By App** to view a list of configured apps, and then select an available action provided by the selected app.
4. Select a configuration that you want to run the action on. In some cases, you may have multiple configurations for a specific app. For example, your environment may have multiple networks separated by firewalls, which would require you to configure one instance of a specific app for each network.

5. Select the field on which you want to perform the action with the configuration. For example, an IPS event may have fields like **sourceAddress** and **destinationAddress** and the attack signature. When a notable is created in Splunk SOAR (On-premises), it has an artifact with fields for the **sourceAddress** and **destinationAddress** from the event. Search for one of these fields to perform the action on. Click **Enter** to go to the next result or use the icons to navigate results. You can also expand or collapse the lists by using the icons.
6. (Optional) Create a custom datapath if the datapath you need isn't available. When you add a custom datapath, it is only available for the block you add it to. To create a custom datapath, follow these steps:
    1. Hover over a datapath field title and click **+**.
    2. Enter the datapath name.
    3. Select either **Key** or **List** from the drop-down menu. Use **Key** to use one value, and use **List** to use a list of values. Using **List** adds a **.*** value to the datapath and it appears as <list_name [] > with datapaths nested below it in the datapath picker. To add more values to your **List**, click the **+** icon under the top value of the list.
    4. Click **Save**.
7. Click **Done**.
8. Click **Save**.
9. Enter a comment about this action.

You can also configure **Advanced** settings for an **Action** block. You can use **Join Settings**, **Scope**, and **Action Settings** in an **Action** block. For more information on these settings, see Advanced settings.

Example: Add a custom datapath to a Splunk SOAR (On-premises) playbook block

You might want to create a custom datapath if the datapath you need isn't available. This can happen when running actions with dynamic results. For example, if you execute a "run query" action on the Splunk app in Splunk SOAR (On-premises), the action result output includes a dynamic list of fields that are defined as part of the query that was run. These fields don't appear in the data path selector, however they can be added by creating a custom datapath. In this instance, if the name of the action result output wasn't available, you can create the custom datapath `action_result.data.*.hostname`. To create this custom datapath, follow these steps:

1. Add an action block to your playbook by dragging and dropping the half-circle icon attached to any existing block in the editor. Select an **Action** block from the menu that appears.
2. Search for and select the action **run query** on the Splunk app in the **By Action** tab in the block.
3. In the **Configure** tab for the block, enter the SPL query `host="web_application"` to run a run query action on the Splunk app in Splunk SOAR (On-premises).
4. Click on the block you want to add the custom datapath to and select the datapath `run_query_1`. Once you select this, you see that the hostname isn't available even though it was visible when running your query in Splunk.
5. Add the custom datapath `hostname` under `data []` . Custom datapaths only appear in the block they were added in:
    1. Hover over the data field title and click **+**.
    2. Enter the datapath name `hostname` as a **Key**.
    3. Click **Save**.

The custom datapath appears in the list under `data []` as `hostname`.

Run other playbooks inside your playbook in Splunk SOAR (On-premises)

You can configure your playbook to run another existing playbook. Call one playbook from another playbook to avoid having to copy and maintain code in different places. You can call a playbook from another playbook any number of times, up to 10 levels of recursion. For example, playbook A can be called by playbook B, which can be called by playbook C, and so on, for a maximum of 10 nested levels. However, there is no limit to the number of playbooks which can call playbook A, provided the calls are within 10 levels of recursion.

To configure your playbook to run another playbook:

1. Drag and drop the half-circle icon attached to any existing block in the editor. Select a **Playbook** block from the menu that appears.
2. In the **Playbook** field, select the playbook you want to run from the drop-down list. When you hover over a playbook, the repository the playbook is in, description, and parent playbooks, if any, are listed. You can select playbooks from the **All Playbooks**, **Automation**, or **Input** type playbook categories.
3. (Optional) If you selected an **Input** type playbook, you can assign the inputs datapaths from the list. For more information on using an **Input** playbook as a sub-playbook, see Use an Input playbook as a sub-playbook.
4. (Optional) Toggle the **Synchronous** switch on to make this playbook wait for the called playbook to complete running before continuing. If this switch is left off, the playbook finishes executing without waiting for the called playbook to complete.

You can also configure **Advanced** settings for a **Playbook** block. You can use **Join Settings** and **Scope** with a playbook block. For more information on these settings, see Advanced settings.

Playbooks differ from action blocks in the following ways:

- The playbook continues to downstream blocks regardless of whether the called playbook is successful.
- The called playbook doesn't return any values that are used in downstream blocks.
- The called playbook doesn't determine the data set, and it operates on the container data with the scope inherited from the caller.
- The called playbook runs independently from the caller. If you wire a series of playbooks to run, they are processed in parallel if the **Synchronous** switch is left off. See Determine your playbook flow in Splunk SOAR (On-premises).

If you use the Scope advanced setting on a playbook block, it won't change the scope of a child playbook. Scope only affects the collected artifact data that is passed in as inputs to the child playbook and the collection occurs before the child playbook is run.

Add custom code to your Splunk SOAR (On-premises) playbook with the code block

Add custom Python code to a **Code** block. **Code** blocks enable you to expand the kinds of processing performed in a playbook, such as adding custom input parameters and output variables.

Add a code block to your playbook

Perform the following steps to add a **Code** block to a playbook.

1. Drag and drop the half-circle icon attached to any existing block in the editor. Select a **Code** block from the menu that appears.
2. Configure input parameters and output variables. See Add input parameters to a code block and Add output variables to a code block.
3. Click the **Python Playbook Editor** to open it and add your custom code. See Use the Python Playbook Editor to add custom code.

4. Click **Done**.

Add input parameters to a code block

Input parameters represent a data path. You can set a data path from any valid blocks upstream, artifact data, and container data.

To create or remove an input parameter, perform the following steps:

1. Click the **+ Input Parameter** icon to add an input parameter. The index of parameters starts at zero.
2. Click in the **Select Parameter** box to set the properties for the input parameter. You can select between artifact and event properties and can search in the search box for a specific property. Click **Enter** to go to the next result or use the ⌄ ⌃ icons to navigate results. You can also expand or collapse the lists by using the icons.

Add output variables to a code block

Output variables are usable as inputs in other downstream blocks, such as **Action**, **Utility**, **Filter**, **Decision**, **Format** and **Prompt** blocks. The name of an output variable becomes `<block_name>__<variable_name>` in the auto-generated section of the playbook code. Give your output variables clear and meaningful names in your custom code so that you can distinguish them from one another.

Follow these steps to add an output variable:

1. Click **+ Output Variable** to add an output variable.
2. Type a name to set the name for, or rename the output variable.

The following example shows both custom code and how outputs are saved:

```
def format_login(action=None, success=None, notable=None, results=None, handle=None,
filtered_artifacts=None, filtered_results=None, **kwargs):
    oar.debug("format_login() called")

    format_login__login_table = None

    ################################################################################
    ## Custom Code Start
    ################################################################################

    # format the output into JIRA's markup language for rendering a table
    format_login__login_table = "|| output of '/usr/bin/last -a' ||\n"
    last_lines = get_user_1_result_item_0[0].split('\n')
            for line in last_lines:
                      format_login__login_table += "| {} |\n".format(line)

        oar.debug("table of logins for jira:")
        oar.debug(format_login__login_table)

    ################################################################################
    ## Custom Code End
    ################################################################################

    oar.save_run_data(key="format_login:login_table", value=json.dumps(format_login__login_table))

    return
```

Use custom names to easily identify and arrange your code blocks

You might want to set a custom name for a block to help you distinguish between blocks.

To set a custom name for the **Code** block, follow these steps:

1. Click the **Info** tab from the configuration panel of the **Code** block.
2. Enter a name in the **Custom Name** box.
    1. As a best practice, do not use personally identifiable information in the names of code blocks.
    2. Custom names can use uppercase and lowercase letters A-Z, numbers 0-9, and underscores.
    3. Custom names can be up to 50 characters long.
    4. Setting or changing a custom name changes that custom name in all data paths that use it, including any generated and custom code.
3. Enter a **Description** in the **Description (code comment)** box to act as a description of your code.
4. Enter a note in the **Notes (block tooltip)** box to act as a tooltip for the **Code** block.

You can also configure **Advanced** settings for a **Code** block. You can use **Join Settings** and **Scope**, in a **Code** block. For more information on these settings, see Advanced settings.

Use the Python Playbook Editor to add custom code

You can use the **Python Playbook Editor** to add custom code to any existing block types. To add custom code to a block, follow these steps:

1. Click on or create a block to open the configuration panel.
2. Click **Python Playbook Editor**.
3. Enter your custom code.

If you add or edit code outside of the **Custom Code Start** and **Custom Code End** sections, the configuration panel for that block is disabled.

Example: Use a custom function to process multiple artifacts and build a parameter list

The following example shows a custom function used to process multiple artifacts in order to build a parameter list.

```
def dbsearch(action=None, success=None, notable=None, results=None, handle=None, filtered_artifacts=None,
filtered_results=None, **kwargs):
    oar.debug("dbsearch() called")

    ################################################################################
    ## Custom Code Start
    ################################################################################

    # Write your custom code here...
    customernamestr = name_value
    parameters = []

    # Loop over the notable event data structure (a list of lists, with each inner list of length three)
    for messagestr, start_time, artifact_id in notable_event_data:
        startdatestr = start_time.split(' ')[0]
        starttimestr = start_time.split(' ')[1]

        # Build the SQL
        if 'groupName:' in messagestr:
            hostgroupstr = messagestr.split('ï¼ ')[0].replace('groupName:', '')
            hoststr = messagestr.split('ï¼ ')[1].split(':')[1]
            sqlstr = "select COUNT(*) as cnt from schedule where customer = '"+ customernamestr +"' and
startdate <= '"+ startdatetimestr +"' and enddate >= '"+ startdatetimestr +"' and ("
            for group in hostgroupstr.split('/'):
                sqlstr = sqlstr + "kyoten like '%"+ group +"%' or reason like '%"+ group +"%' or "
            sqlstr = sqlstr + "kyoten like '%"+ hoststr +"%' or reason like '%"+ hoststr +"%')"

        else:
            hoststr = messagestr.split(':')[0]
            sqlstr = "select COUNT(*) as cnt from schedule where customer = '"+ customernamestr +"' and
 (kyoten like '%"+ hoststr +"%' or reason like '%"+ hoststr +"%') and startdate <= '"+ startdatetimestr +"'
and enddate >= '"+ startdatetimestr +"'"

        # Update the parameter list
        # There should be one parameter per item in the container_data variable
        # There should be one item in the container_data variable per artifact
        # Thus, there should be one parameter per artifact
        parameters.append({
            'query': sqlstr,
            'format_vars': "",
            'no_commit': False,
        })

    oar.act("run query", parameters=parameters, connector_configs=['mysql'], callback=filter_2,
name="SearchDB")

    ################################################################################
    ## Custom Code End
    ################################################################################

    return
```

Add additional functionality to your playbook in Splunk SOAR (On-premises) using the Utility block

This feature is currently in beta.

Use the **Utility** block to expand the functionality of your playbooks in Splunk SOAR (On-premises). You can use custom functions and APIs from the **Utility** block. Custom functions enable you to use your Python skills to expand the kinds of processing performed in a playbook, such as applying string transformations, parsing a raw data input, or calling a third party Python module. Custom functions can also interact with the REST API in a customizable way. You can share custom functions across your team and across multiple playbooks to increase collaboration and efficiency.

Configure a utility block

To configure a **Utility** block, follow these steps:

1. Drag and drop the half-circle icon attached to any existing block in the editor. Select a **Utility** block from the menu that appears.
2. Select whether to use a Custom Function or API utility.

Expand playbook functionality with the Custom Function utility

The following prerequisites are needed for using a custom function.

- To use custom functions from the Utility block, you must be using a custom function from a local or community repository, or you must have already created a custom function. See the following documentation for details:

  - Create a custom function
  - Add an input parameter to a custom function
  - Add an output variable to a custom function
  - Using draft mode with custom functions.
  - Delete a custom function

- To create custom functions, you must have **Edit Code** permissions, which can be configured by an Administrator in **Administration > User Management > Roles and Permissions**. For more information on the **Edit Code** permission, see Add a role to Splunk SOAR (On-premises) in the *Administer Splunk SOAR (On-premises)* manual.

If you selected a Custom Function, complete the following steps:

1. Click in the search bar to display all of your repositories.
2. Click the repository your custom function is saved to and either search for your custom function, or select it from the list.
3. (Optional) Once you have selected a custom function, you can configure the value of the input parameters.
    1. (Conditional) To configure the value of the input parameters, click the **>** icon to set the properties.
    2. (Conditional) Create a custom datapath for your input parameters if the datapath you need isn't available. When you add a custom datapath, it is only available for the block you add it to. To see an example of a custom datapath, see Example: Add a custom datapath to a Splunk SOAR (On-premises) playbook block. To create a custom datapath, follow these steps:
        1. Hover over a datapath field title and click **+**.
        2. Enter the datapath name.
        3. Select either **Key** or **List** from the drop-down menu. Use **Key** to use one value, and use **List** to use a list of values. Using **List** adds a **.*** value to the datapath and it appears as <list_name [] > with datapaths nested below it in the datapath picker. To add more values to your **List**, click the **+** icon under the top value of the list.
    3. Click **Save**.

Set parameters with the API utility

Use the **Utility** block API to set parameters of the container it's running in. For example, you can use a utility call from the **Utility** block to set the severity of a container.

If you selected an API, select the utility property you want to set. The following table summarizes the properties that you can set.

Property	Description
add comment	Add a comment to the container. You can either supply a variable or a static string in the input.
add to list	One of two API calls that doesn't operate directly on the container itself. The `add list` property takes two parameters: the list that you want to add to, and the data you are adding. If the list doesn't exist, it is created by Splunk SOAR (On-premises). You can point the data field to a variable by selecting from the drop-down menu or you can type in a fixed string.
add note	Add a note to the container.
add tag	Add a tag to the container.
promote to case	Promote the container to a case.
pin	Pin data to the summary tab in the container. This property takes the following parameters:

Property	Description
	• Message • Data • Pin Type • Pin Color • Name
remove list	One of two API calls that doesn't operate directly on the container. The `remove list` property takes a list name as the single parameter, and deletes that list when it has run.
remove tag	Remove a tag from the container.
set label	Set the label of the container. The drop-down lists all of the labels available on your Splunk SOAR (On-premises) instance.
set owner	Set the owner of the container.
set sensitivity	Set the sensitivity of the container.
set severity	Set the severity of the container.
set status	Set the status of the container, such as closed.

Finish editing the playbook

When you are finished editing your playbook, do the following:

1. Click **Save** to enter your desired settings and playbook name.
2. Once you have selected a utility, configure the datapaths. Search for the datapath you want to use. Click **Enter** to go to the next result or use the ⌄ ⌃ icons to navigate results. You can also expand or collapse the lists by using the icons. To create a custom datapath, see Example: Add a custom datapath to a Splunk SOAR (On-premises) playbook block.
3. Click **Done**.

You can configure multiple utility calls in any utility block. For example, you can set the label, severity, and status of a container using one utility block.

Use filters in your Splunk SOAR (On-premises) playbook to specify a subset of artifacts before further processing

Create conditions in a **Filter** block to gather a subset of artifacts. Only the artifacts matching the specified condition are passed along to downstream blocks for processing. This is useful when you want to remove artifacts that are not needed in the flow if the playbook, or you need to separate artifacts because they require different blocks for processing. For example:

- If an IP address comes from North Korea or Turkey, you can block it.
- If an IP address comes from North America, you can perform an IP reputation lookup.
- If an IP address falls in the 192.168.10.* range, you can to grant access to the user.

Options in a condition aren't related to each other and you can perform multiple actions on an IP address. For example, both the second and third conditions in the example could be true, as an IP address in the 192.168.10.* range could both come from North America and be an internal user who was granted access. The filtered data set is given a named result in the format `name="filter_1:condition_1"` and passed to the next block for processing.

Create a Filter block in your playbook

To create a filter, perform the following tasks:

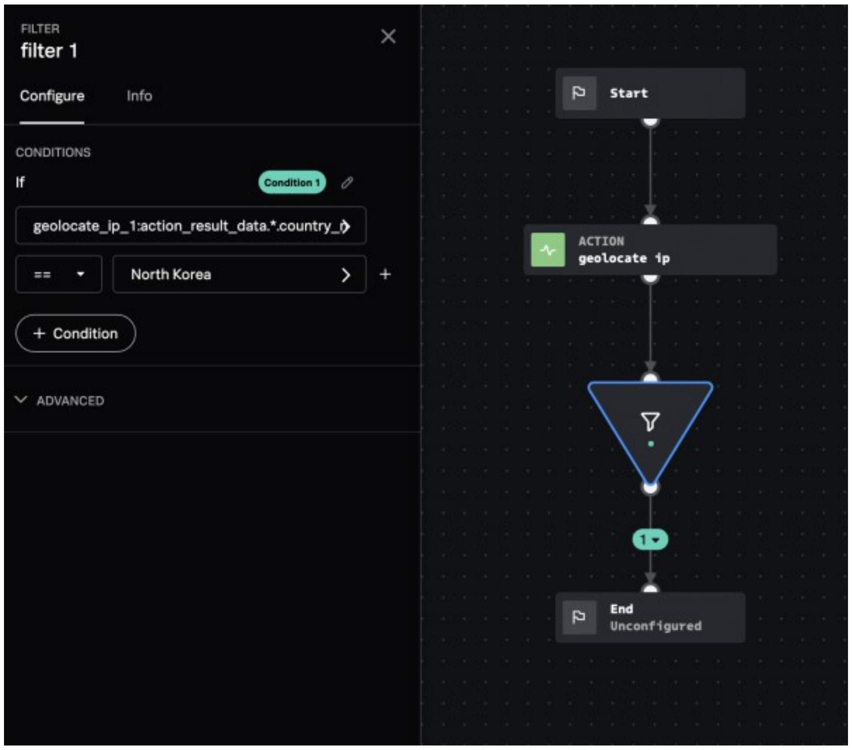
1. Drag and drop the half-circle icon attached to any existing block in the editor. Select a **Filter** block from the menu that appears.
2. Click the **Select Parameter** field and search for and select the parameter you want to filter on. Parameters are made available to the **Filter** block by upstream blocks. Click **Enter** to go to the next result or use the ⌄ ⌃ icons to navigate results. You can also expand or collapse the lists by using the icons.
3. (Optional) Create a custom datapath if the datapath you need isn't available. When you add a custom datapath, it is only available for the block you add it to. To see an example of a custom datapath, see Example: Add a custom datapath to a Splunk SOAR (On-premises) playbook block. To create a custom datapath, follow these steps:
    1. Hover over a datapath field title and click **+**.
    2. Enter the datapath name.
    3. Select either **Key** or **List** from the drop-down menu. Use **Key** to use one value, and use **List** to use a list of values. Using **List** adds a **.*** value to the datapath and it appears as <list_name [] > with datapaths nested below it in the datapath picker. To add more values to your **List**, click the **+** icon under the top value of the list.
    4. Click **Save**.
4. Click in the **==** field and select an operator for the filter.
5. Click the **Select Value** field and search for and select the value you want to match. See Example of creating a filter for an example of how these fields all work together.
6. (Optional) Click **+ Condition** to create another matching condition for the filter. You can have a maximum of five conditions per **Filter** block. Each condition will have its own downstream path.
7. Click **Done**.

You can also configure **Advanced** settings for a **Filter** block. You can use the **Case-sensitive** check box to select if you want the conditions evaluation to be case-sensitive, or case-insensitive. The default is case-sensitive. For more information on other **Advanced** settings, see Advanced settings.

Additionally, you can click the **Info** tab to create a custom name for the block, add a description for the block, and add a tooltip to the block. See Use custom names.

Example of creating a filter

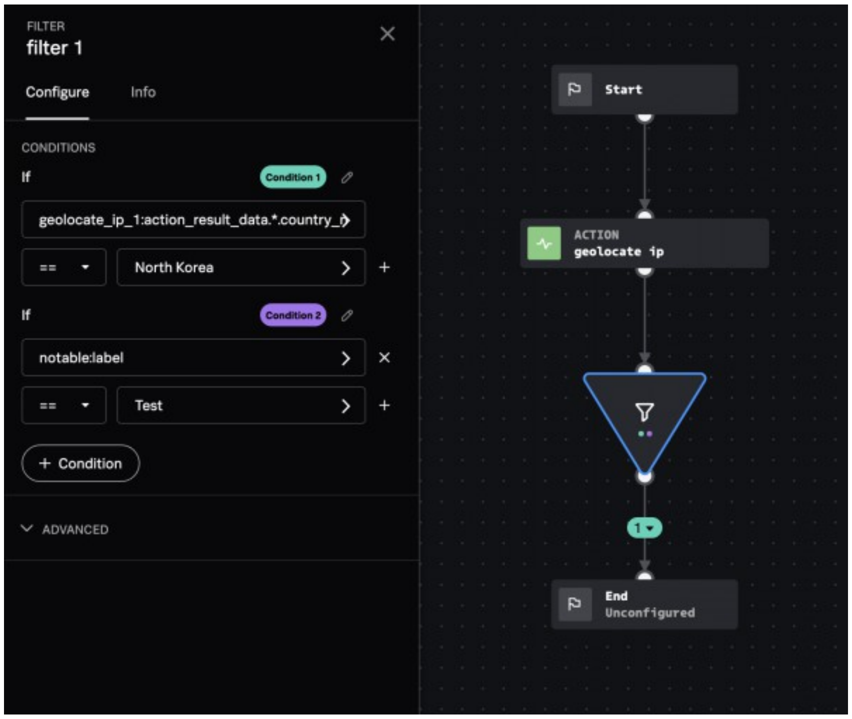
In the following example, you can create a filter to perform a `geolocate ip` action on a source IP address and block any IP addresses from North Korea.



1. Configure a `geolocate ip` **Action** block in the playbook editor. For more information on configuring an **Action** block, see Add an action block to your Splunk SOAR (On-premises) playbook.
2. Drag and drop the half-circle icon attached to the **Action** block in the editor. Select a **Filter** block from the menu that appears.
3. Click the **Select Parameter** field and select **geolocate_ip_1**.
4. Select **geolocate_ip_1:action_result_data.*.country_name**. Leave **==** as the operator, and type **North Korea** in the **Select Value** field.

Example of creating a filter with multiple conditions

You can create multiple rows within a condition or multiple conditions.

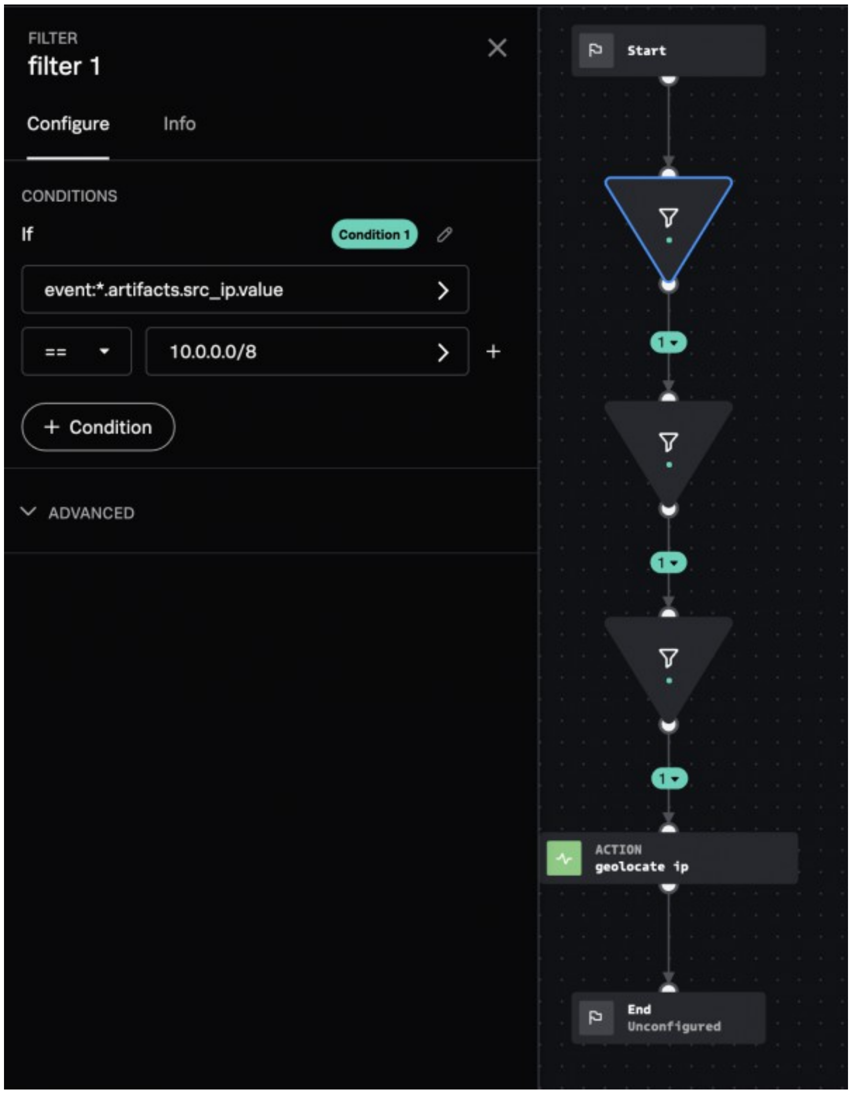


1. Click **+ Condition** to create a second set of filter conditions, which also adds a second output point on the **Filter** block.
2. In the **Select Parameter** field, select **artifacts** and then **label**.
3. Select **==** as the operator, and enter **Test** in the **Select Value** field.

Multiple conditions within a filter block are independent of each other. The results of Condition 1 don't play into the set of inputs for Condition 2. In this example, Condition 1 uses the result from the `geolocate ip` action, while Condition 2 uses a property of the artifact. Each condition has its own color to make it easier to identify the separate downstream actions; green marks the path on Condition 1, and purple marks the path of Condition 2. If you want to edit the Condition 1 and 2 labels, click the pencil icon next to the label names.

Example of filter chaining

You can also chain multiple filter blocks together to obtain a more specific set of data.


FILTER
filter 1

Configure Info

CONDITIONS

If Condition 1

event:*.artifacts.src_ip.value

== 10.0.0.0/8

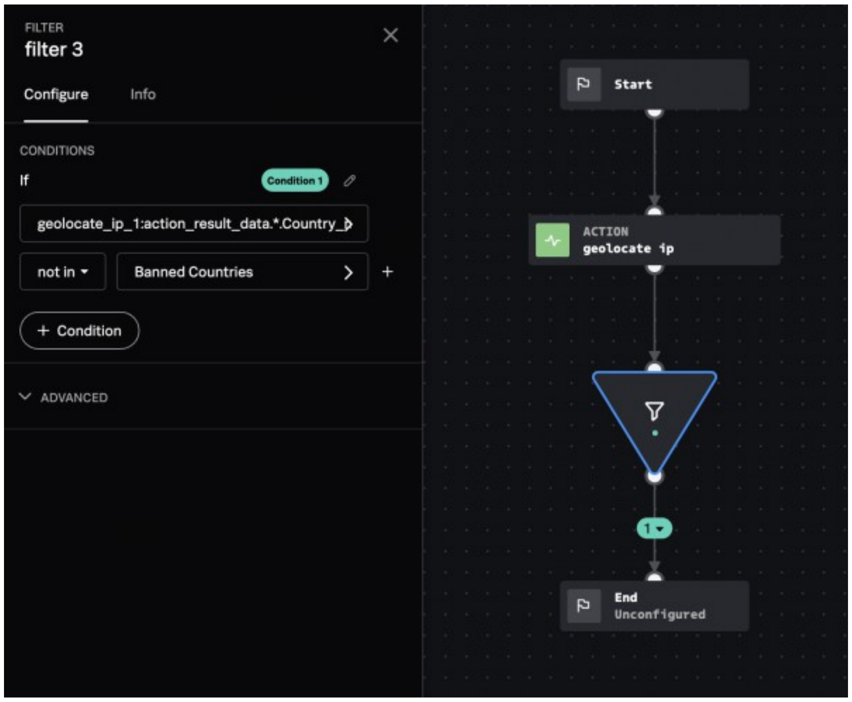
For example, to filter out RFC1918 addresses (10.x.x.x, 172.16.x.x-172.31.x.x, and 192.168.x.x), and then perform a `geolocate ip` action on the remaining addresses, perform the following steps:

1. Drag and drop the half-circle icon attached to any existing block in the editor. Select a **Filter** block from the menu that appears.
2. In the **Select Parameter** field, select **event > src_ip**.
3. Enter **10.0.0.0/8** in the **Select Value** field.
4. Create a second filter for the 172.16.0.0/12 IP addresses. This filter uses the filtered results from the previous block.
5. Create a third filter for the 192.168.0.0/12 IP addresses. This filter uses the filtered results from the previous block.
6. Create the `geolocate ip` action block on the remaining IP addresses.

If a filter block eliminates all variables while filtering, the downstream action can't run.

Example of using a custom list in a filter

You can use custom lists in your **Filter** blocks to simplify checking against a fixed set of items. For example, instead of checking the source country of an IP address to see if it is North Korea, you can define a list of countries in a custom list, then check the IP address against all of the countries in the list. See Create custom lists for use in Splunk SOAR (On-premises) playbook comparisons for more information about how to create and maintain custom lists.



FILTER
filter 3

CONDITIONS
If — Condition 1
geolocate_ip_1:action_result_data.*.Country_
not in — Banned Countries

In this example, use a custom list named **Banned Countries** in a filter by performing the following tasks:

1. Configure a `geolocate ip` action in the playbook editor.
2. Add a **Filter** block.
3. Click the **Select Parameter** field and select **geolocate_ip_1**.
4. Select **geolocate_ip_1:action_result_data.*.country_name**.
5. Use "in" or "not in" as the operator.
6. Click in the **Select Value** field, select **custom lists**, then select **Banned_Countries**.

The country name of the source IP address is checked against the countries defined in the **Banned Countries** custom list.

The matching that occurs between artifact data and custom list items is exact matching and is case sensitive with no partial matches.

Use decisions to send artifacts to a specific downstream action in your Splunk SOAR (On-premises) playbook

Use a **Decision** block to change the flow of artifacts by performing an If, Else If, or Else functions. When an artifact meets a True condition, it is passed downstream to the corresponding block in the playbook flow. If none of the Decision block conditions are met, the playbook run fails.

The first time an artifact meets a condition, it is passed along to the corresponding downstream block. The artifact is no longer available for evaluation by other Else If or Else statements, and cannot be passed to other downstream blocks. You can only perform one action an artifact based on the condition that is matched first.

In this example, IP addresses found in artifacts are compared against some specific IP address ranges:

- If an IP address is in the 192.168.* range, perform an IP reputation lookup, ELSE IF
- The IP address is in the 172.16.* range, grant access to the user, ELSE
- If the IP address doesn't fall into either of the previous categories, perform a `geolocate ip` action.

Unlike Filter blocks, no named datasets are created for reference later on in the playbook.

Create a Decision block in your playbook

To create a decision block, perform the following tasks:

1. Drag and drop the half-circle icon attached to any existing block in the editor. Select a **Decision** block from the menu that appears.
2. Click the **Select Parameter** field and search for and select the parameter you want to compare. Click **Enter** to go to the next result or use the ⌄ ⌃ icons to navigate results. You can also expand or collapse the lists by using the icons. Parameters are made available to the **Decision** block by upstream blocks. To create a custom datapath, see Example: Add a custom datapath to a Splunk SOAR (On-premises) playbook block.
3. Click in the **==** field and select an operator for the decision.
4. Click the **Select Value** field and select the value you want to match. See Example of creating decisions for multiple downstream actions for an example of how these fields all work together.
5. (Optional) Click **+ Else If** to create another matching condition for the decision.
6. Click **Else** to create the final branch for the decision.

7. Click **Done**.

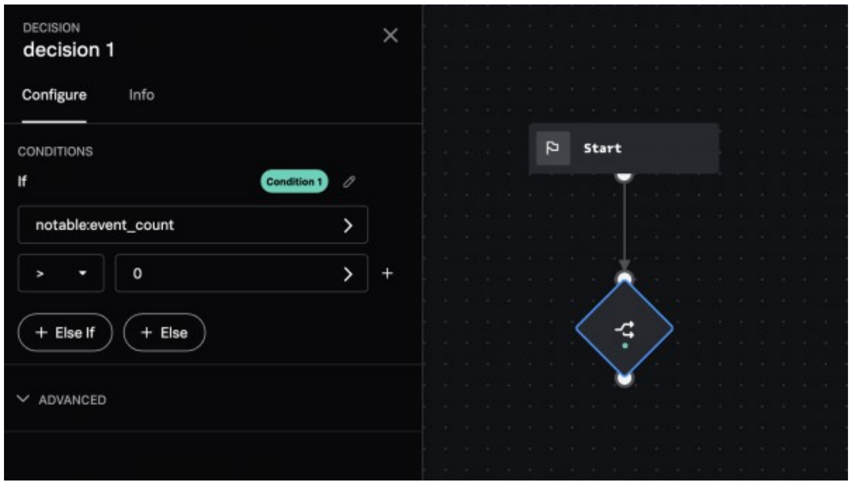
You can also configure **Advanced** settings for a decision block. You can use the **Case-sensitive** check box to select if you want the conditions evaluation to be case-sensitive, or case-insensitive. The default is case-sensitive. For more information on other **Advanced** settings, see Advanced settings.

Additionally, you can click the **Info** tab to create a custom name for the block, add a description for the block, and add a tooltip to the block. See Use custom names.

Example of creating decisions for multiple downstream actions

Decision blocks control the program flow based on comparisons of artifact data, container properties, date functions, and action results. Create `if` and `else if` conditions to branch to multiple downstream blocks as a results of the comparisons.

In the following example, start with a **Decision** block that checks to see if any artifacts are in the container.

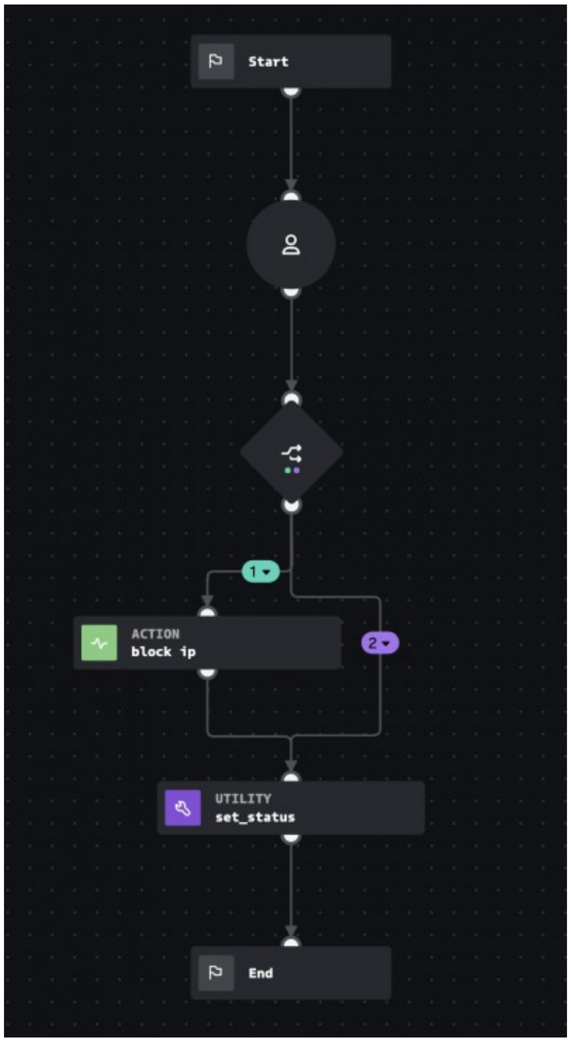


1. Drag and drop the half-circle icon attached to the start block in the editor. Select a **Decision** block from the menu that appears.
2. In the **If** field, click the **Select Parameter** field and select a parameter to evaluate. You can choose from the properties provided by the container, artifact data, and date and time options.
3. Select **container properties** from the list of options, and then click **artifact_count** as the property you want to evaluate.
4. Select **>** as the operator, and enter **0** in the **Select Value** field.

Condition 1 corresponds with the green dot on the decision block. All data is passed on to the next block.

Example of creating decisions with multiple statements

You can create more complex decision blocks with up to five statements. The following example shows a decision based on user input through a prompt block. The prompt block asked a yes/no question on whether to block all IP addresses in the container or not. The decision block checks if the answer is "yes", and if so, all IPs are blocked and the status is set to resolved. Or, if the answer is "no", the status is simply set as resolved and no IP addresses are blocked.


Start
1
ACTION
block ip
2
UTILITY
set_status
End

Each subsequent statement and downstream block is color coded: green marks the path of the **If** statement, and purple marks the path of the **Else** statement. Each statement has its own and only one downstream block.

Using decision and filter blocks together

When using decision and filter blocks, the best flow for creating a playbook using these blocks is to use a decision block, then a filter block. A decision block does not create a subset of data like a filter block does and a decision block stops evaluating conditions after the first matched condition. A filter creates a subset of data for each matched condition. For example, you can use a decision block to find out if you have at least one IP address, then use a filter block to create a subset of IPs to do an IP reputation check in a subsequent action.

Customize the format of your Splunk SOAR (On-premises) playbook content

Use the **Format** block to craft custom strings and messages from various objects.

You might consider using a **Format** block to put together the body text for creating a ticket or sending an email. Imagine you have a playbook set to run on new artifacts that does a basic lookup of source IP address artifacts. You want to take the results of that lookup, format the results, and send the information as an email. You craft your playbook so that the action results are available to the format block. See Use custom names.

To configure a format block, perform the following steps:

1. Drag and drop the half-circle icon attached to any existing block in the editor. Select a **Format** block from the menu that appears.
2. Configure the template parameter variables by clicking in the **Parameters** field and then searching for the parameter you want to use. Click **Enter** to go to the next result or use the icons to navigate results. You can also expand or collapse the lists by using the icons.The first variable is identified as `{0}`, the next as `{1}`, and so on. You can select data from any upstream block.
3. In the **Template** field, craft a message using the variables you define.

You can also configure **Advanced** settings for a format block. Use the **Delimiter** box to specify an alternate separator to use when joining parameters that result in a list together. The default separator is ",". Use the **Drop None** checkbox to select whether or not you want to drop the "None" values from the resulting lists of parameters. By default, the "None" values are included. For more information on other **Advanced** settings, see Advanced settings.

Additionally, you can click the **Info** tab to create a custom name for the block, add a description for the block, and add a tooltip to the block.

Example of defining a template

This example defines a template in the following manner:

```
IP address: {0}
IP address country: {1}
IP address reputation: {2}
```

The message returned as a result of this template looks like the following:

```
IP address: 1.2.3.4
IP address country: United States
IP address reputation: Malicious
```

If multiple events are picked up at the same time, you see the following message:

```
IP address: 1.2.3.4, 10.11.12.13
IP address country: United States, Turkey
IP address reputation: Malicious
```

You can wrap `%%` around a formatting block to make each set of values output on its own line. For example:

```
%%
The IP address {0} originates from {1}.
%%
```

Wrapping `%%` around a formatting block produces results like the following:

```
The IP address 1.2.3.4 originates from United States.
The IP address 10.11.12.13 originates from Turkey.
```

Example of using the Python str.format() function to create more advanced templates

More complicated formatting is supported using all the capabilities of the Python `str.format()` function. The first section of this template demonstrates double curly brackets to support curly bracket escaping. This is particularly important as if you don't use double curly brackets with curly bracket escaping, unintended errors can occur. For example, if you wanted to input JSON strings, you would need to escape the literal using double curly brackets or the system returns an invalid token.

The second section in this template demonstrates automatic escaping of backslashes and quotes, and centered string alignment with a custom filler character:

```
JSON Formatting:
{{"notable_id":"{0}","event_count":{1}}}

Backslashes and quotes are escaped:
"ls /Applications/System \Preferences.app/"

Python formatting is supported:
|{0:-^50}|{1:-^50}|
|{2:-^50}|{3:-^{3}{2}{1}{3}{2}{1}50}|
```

The template produces output as in the following example:

```
JSON Formatting:
{"notable_id":  "1004", "event_count": 10}

Backslashes and quotes are escaped:
"ls /Applications/System \Preferences.app/"

Python formatting is supported:
|-----------------------1004-----------------------|------------------------10------------------------|
|----------------------events----------------------|-------Zeus infection on HQ finance server------|
```

Require user input using the Prompt block in your Splunk SOAR (On-premises) playbook

Use a **Prompt** block in your playbook to send a message to a user or group that they must acknowledge.

To configure a prompt, perform the following tasks:

1. Drag and drop the half-circle icon attached to any existing block in the editor. Select a **Prompt** block from the menu that appears.
2. Select a **User or Role** from the drop-down list to approve the prompt. If the task is assigned to a group of users, the first user to approve it kickstarts the playbook run.
3. In the **Message** box, craft a meaningful message so the users receiving the message understand what actions they must take.
4. (Optional) Click **+ Message Parameter** to search for and add a message parameter. Click **Enter** to go to the next result or use the ⌄ ⌃ icons to navigate results. You can also expand or collapse the lists by using the icons. Adding a message parameter creates an input with a paired parameter that displays in the **Message** box.
5. Click **+ Question** and enter a question to ask the approver in the **Question 1** box. Click **+ Question** to add additional questions.
6. From the **Responses** drop-down list, choose the type of response to the question that is required to complete the task.
7. From the **Required response time** field, choose the response time in minutes.
8. Click **Done**.

To learn more about how to format a **Prompt** message, see Customize the format of your Splunk SOAR (On-premises) playbook content.

You can also configure **Advanced** settings for a prompt block. Use the **Delimiter** box to specify an alternate separator to use when joining parameters that result in a list together. The default separator is ",". Use the **Drop None** checkbox to select whether or not you want to drop the "None" values from the resulting lists of parameters. By default, the "None" values are included. For more information on other **Advanced** settings, see Advanced settings.

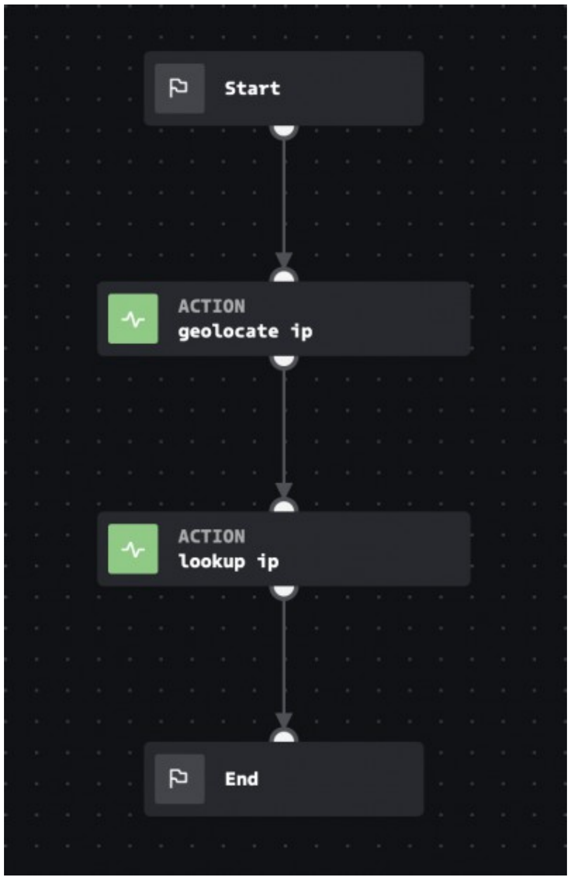
Additionally, you can click the **Info** tab to create a custom name for the block, add a description for the block, and add a tooltip to the block. See Use custom names.

Determine your playbook flow in Splunk SOAR (On-premises)

The order in which you arrange the blocks and lines in your playbook determine the playbook flow.

Process playbook blocks serially

Serial processing means playbook blocks are performed in the order they are arranged.


Start
ACTION
geolocate ip
ACTION
lookup ip
End

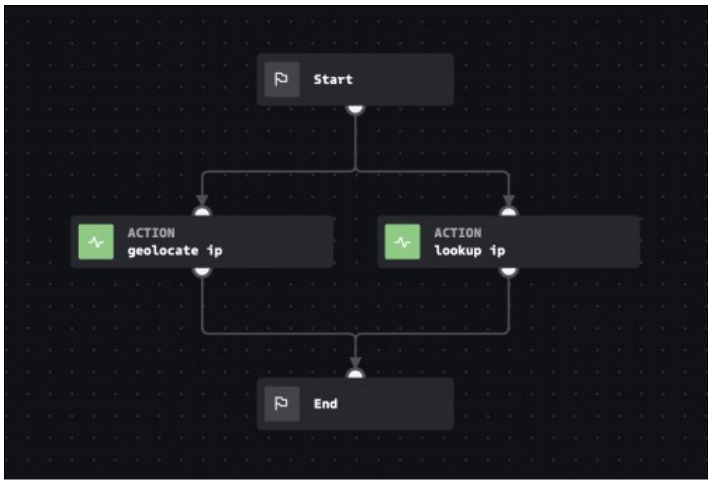
In this example, the blocks perform as described:

1. A `geolocate ip` is performed on a source IP address.
2. When the `geolocate ip` action is finished, a `lookup ip` performs.

Use serial processing when there must be a specific order to the operations, such as when a downstream block depends on the results from an upstream block.

Processing playbook blocks in parallel

You can also wire blocks to process in parallel, as shown in the following example.



In this case, the `geolocate ip` and `lookup ip` actions perform simultaneously, and either action can finish first. You can wire blocks in this manner when you have no dependencies on the completion of either block, or if there are no dependencies between the blocks themselves.

Save a playbook so Splunk SOAR (On-premises) can access it

You must save a playbook before Splunk SOAR (On-premises) can access it. If errors exist in the playbook or the underlying code, an error message is displayed with additional information. If there are any parent playbooks associated with the playbook, a message will appear listing the playbooks the save might affect. You can't save a playbook until you resolve all errors.

Save a playbook for the first time

Perform the following tasks to save a playbook for the first time.

1. Click **Save** to begin saving a playbook.
2. Enter a brief comment for the playbook, such as a description of what the playbook does.
3. (Optional) Click **Reuse this comment until the editor is reloaded** if you want to use the same comment each time you save the playbook during this editing session.
4. Click **Save** to save the playbook.

Save a copy of a playbook

Perform the following tasks to create a copy of a playbook.

1. Open any existing playbook for editing.
2. Click the vertical ellipsis icon and click **Save As**.
3. Enter a new name for the playbook.
4. Enter a brief comment for the playbook's revision history, such as a description of what has changed.
5. (Optional) Click **Reuse this comment until the editor is reloaded** if you want to use the same comment each time you save the playbook during this editing session.
6. Click **Save**.

Export and import a playbook

Perform the following tasks to export a playbook. The playbook is exported as a JSON file to your local filesystem.

1. Open any existing playbook for editing.
2. Click the vertical ellipsis icon and click **Export**.

Perform the following tasks to import and create a new playbook.

1. From the Splunk SOAR (On-premises) main menu, select **Playbooks**.
2. Click the **Import playbook** icon.
3. Select a target repository from the **source to update** drop-down menu.
4. Drag and drop your .tgz playbook file that was exported previously, or click **click here** and select the playbook on your file system.
5. Click **Upload**.

Use keyboard shortcuts in the playbook editor

When using the playbook editor, press **Command+?** while the **Settings** panel is open to see a list of keyboard shortcuts. The following keyboard shortcuts can be used in the playbook editor.

Description	Shortcut
Add block panel	Command+B
Add action block	Command+1
Add playbook block	Command+2
Add code block	Command+3
Add utility block	Command+4
Add filter block	Command+5
Add decision block	Command+6
Add format block	Command+7
Add prompt block	Command+8
Auto arrange playbook	Command+Shift+F

Description	Shortcut
Zoom to fit playbook	Command+Shift+Z
Toggle Python editor	Command+E
Toggle debugger	Command+D
Save	Command+S
Toggle settings	Command+K
Show the list of hotkeys	Command+?
Zoom in	Command+Shift++
Zoom out	Command+Shift+-

If you are using a Windows or Linux machine, replace Command with Control in the keyboard shortcuts.

Use the Classic Playbook Editor to create and view playbooks to automate analyst workflows

Create a new playbook in Splunk SOAR (On-premises) using the classic playbook editor

Perform the following tasks to open the classic playbook editor and create a new playbook in Splunk SOAR (On-premises):

1. From the **Home** menu, select **Playbooks**.
2. Expand the **+ Playbook** button and click **+ Classic Playbook**. The classic playbook editor opens in a new tab in your browser. The **Start** and **End** blocks are populated on the editor. All playbooks must start with the **Start** block and end with the **End** block.
3. Specify a name for the playbook.

Playbooks in the same folder cannot have the same name. Playbooks in different folders can have the same name. Click the plus and minus icons to zoom in or zoom out.

Go to Add a new block to your Splunk SOAR (On-premises) playbook using the classic playbook editor to learn how to add a new block and begin building your playbook.

Organize your playbook library in Splunk SOAR (On-premises)

Organizing your playbooks can help you quickly assess the purpose of each playbook. The following is a list of recommendations on how to name and organize your playbooks.

- Create concise names for your playbooks. In certain views, there is a limited width for displaying the playbook execution history.
- Use the category field in the playbook to specify the most important attribute of the playbook for organizational purposes. For example, the team that owns the playbook, the environment the playbook runs in, or the use case of the playbook. Using this field helps you quickly assess the purpose of each playbook.
- Use the name of the playbook as a high-level description of what the playbook does. You can use the description field of the playbook to add more detailed information.
- Use tags to track other attributes of the playbook such as the team member who is responsible for it, the stability level, or the priority.

The playbook listing page provides information on the apps, actions, assets, and playbooks used, so there is no need to include this type of information in your playbook name. Attempting to include this information in the playbook name can cause issues as the metadata will change over time and you will have to update your playbook names and calls to sub playbooks.

Add a new block to your Splunk SOAR (On-premises) playbook using the classic playbook editor

To add a new block to a playbook, drag the half-circle icon attached to any block on the canvas. Release your mouse to create a new empty block connected to the originating block with an arrow.

When you place a new block on the editor, a set of playbook types appears for you to select:

Playbook type	Description
Action	Run an action provided by an app that is installed and configured in Splunk SOAR (On-premises). For example, you can use the MaxMind connector to geolocate an IP address. See Add an Action block to a Splunk SOAR (On-premises) playbook using the classic playbook editor.
Playbook	Run an existing playbook inside your current playbook. See Run other Splunk SOAR (On-premises) playbooks inside your playbook using the classic playbook editor.
API	Perform an action by making an API call. See Set container parameters in Splunk SOAR (On-premises) using the API block.
Filter	Filter the results of the previous block. For example, you can separate items that have a specific severity and perform a different set of actions on those items. See Use filters to separate Splunk SOAR (On-premises) artifacts before further processing with the classic playbook editor.
Decision	Make a decision and perform different actions depending on the results of the previous block. For example, you can blacklist all destination IPs that belong to a specific country. See Use decisions to send Splunk SOAR (On-premises) artifacts to a specific downstream action with the classic playbook editor.
Format	Format the results of the previous block. For example, you can gather data, format that data in a specific way, and send an email. Customize the format of your Splunk SOAR (On-premises) playbook content using the classic playbook editor.
Prompt	Require a user to take action before proceeding to the next block. See Require user input to continue running the Splunk SOAR (On-premises) playbook using the classic playbook editor.
Manual Task	Send a message to a Splunk SOAR (On-premises) user or group that must be acknowledged. See Require user input to continue running the Splunk SOAR (On-premises) playbook using the classic playbook editor.
Custom Function	Add custom Python code to your playbook to expand the kinds of processing that are performed by the playbook. Add custom code to your Splunk SOAR (On-premises) playbook with the Custom Function block using the classic playbook editor.
Legacy Custom Function	Legacy custom functions are the custom functions that were introduced for playbooks in Splunk Phantom version 4.2. Add custom code to your Splunk SOAR (On-premises) Playbook with the Legacy Custom Function block using the classic playbook editor. Legacy custom functions are supported for users transitioning from Splunk Phantom to Splunk SOAR (On-premises). Legacy custom functions should be converted to the newer custom function type. For information on converting legacy custom functions to new custom functions, see Convert legacy custom functions to new custom functions.

Advanced settings

Follow these steps to configure advanced settings for a block.

To use Advanced settings, when configuring a block follow these steps:

1. Click **Advanced**.
2. Modify the advanced settings.

Setting	Block type	Description
Join Settings	Available for action, playbook, filter, decision, format, and prompt block types.	You can configure join settings when multiple incoming blocks that support the synchronous functionality are linked to any downstream block. All **Action**, **Prompt**, and **Manual Task** blocks run synchronously and playbooks can be toggled to run synchronously in the block configuration. See Run other playbooks inside your playbooks in Splunk SOAR (On-premises) for more information on the synchronous functionality.

Setting	Block type	Description
		Configure join settings from the downstream block. These settings determine whether or not you wait to execute the next block until the required upstream blocks finish running. Click the **required** checkbox if the action in the upstream block must be completed before this downstream block is run. The **required** checkbox is enabled by default.
Scope	Available for action, playbook, filter, decision, format, and prompt block types.	Configure scope to determine how the artifact data passed into a block's API is collected. Collection occurs in the context of the current playbook. Setting the scope advanced setting on a playbook block doesn't change the scope of a child playbook. Scope only affects the collected artifact data that is passed in as inputs to the child playbook and the collection occurs before the child playbook is run. • Default: The artifact data for the block uses the same scope as the playbook. • New Artifacts: The artifact data for the block is collected for new events. • All Artifacts: The artifact data for the block is collected for all events.
Action Settings	Available for action blocks.	Configure the action settings that a user must perform. Action settings are only available from an action block. • Reviewer: Select a user or group that must approve this action before the action runs. If you select a group or role, any user in that role can approve the action. • Delay Timer: Set a delay in minutes before the action runs. A clock icon is visible on the action block to show that a delay is configured.
Case-sensitive	Available for decision and filter blocks.	Select if you want the conditions evaluation to be case-sensitive, or case-insensitive. The default is case-sensitive.
Delimiter	Available for prompt and format blocks.	Specify an alternate separator to use when joining parameter values that result in a list together. The default separator is ",".
Drop None Values	Available for prompt and format blocks.	Select whether or not you want to drop the "None" values from the resulting lists of parameters. By default, the "None" values are included.
Re-fetch Container Data	Available for API blocks.	Select this option to fetch updated container data. The default state is checked. If you uncheck the checkbox to use the original cached container data, it is less expensive.

Add an Action block to a Splunk SOAR (On-premises) playbook using the classic playbook editor

Perform the following steps to add an **Action** block to a playbook:

1. Drag the half-circle icon attached to any existing block in the editor.
2. Select **Action** from the list of block types. Actions available to you in the playbook editor are determined by the apps that are installed and configured in Splunk SOAR (On-premises).
3. Select the action you want to configure, or enter an action name in the search field if you don't see the desired action listed. You can also filter the list of actions by action type.
4. Select **investigate**, **generic**, **correct**, or **contain**.
5. Click **By App** to view a list of configured apps, and select an available action provided by the selected app.
6. Select an asset that you want to run the action on. An asset is a specific configuration or instance of an app. In some cases, you may have multiple configurations for a specific app. For example, your environment may have multiple networks separated by firewalls, which require you to configure one instance of a specific app for each network.
7. Select the field where you want to perform the asset. For example, an IPS event may have fields like **sourceAddress** and **destinationAddress** and the attack signature. When a container is created in Splunk SOAR (On-premises), it has an artifact with fields for the **sourceAddress** and **destinationAddress** from the event.

8. Select one of these fields to perform the action on.
9. Click **Save**.
10. Enter a comment about this action.

Configure linked parameters

Configure linked parameters in an **Action** block when you have multiple assets that share parameters with the same name. For example, you might have multiple assets configured that provide an action to create a ticket with a `subject` parameter. In this case, the word "linked" appears above the **subject** field, indicating that the field is linked to another field with the same name in a different asset. If you change the value here, the value for the field changes in all assets.

If you need to have the field take separate values, create separate action blocks.

Advanced settings

Follow these steps to configure advanced settings for an **Action** block:

1. Click **Advanced Settings**.
2. Select **General Settings**, **Action Settings**, or **Join Settings**.

Setting	Description
General Settings	Configure settings for this **Action** block. • Custom Name: The name for this action block. This name is visible in the playbook editor and also in Splunk SOAR (On-premises) wherever details about this action are visible. • Description: The **Description** field shows up as a code comment above the block definition. • Notes: The **Notes** field contents appear when you hover over the Note icon in the action block.
Action Settings	Configure the action settings that a user must perform. • Reviewer: Select a user or group that must approve this action before the action runs. If you select a group or role, any user in that role can approve the action. • Delay Timer: Set a delay in minutes before the action runs. A clock icon is visible on the action block to show that a delay is configured.
Join Settings	You can configure Join settings when you have two blocks with callbacks both calling the same downstream block. Block types with callbacks are **Action** and **Prompt**. Configure Join settings from the downstream block. Click the **required** checkbox if the action in the upstream block must be completed before this downstream block is run.

Use filters to separate Splunk SOAR (On-premises) artifacts before further processing with the classic playbook editor

Create conditions in a **Filter** block to separate a subset of artifacts. Only the artifacts matching the specified condition are passed along to downstream blocks for processing. This is useful when you want to remove artifacts that aren't needed in the flow of the playbook, or you need to separate artifacts because they require different blocks for processing. For example:

- If an IP address comes from North Korea or Turkey, you can block it.
- If an IP address comes from North America, you can perform an IP reputation lookup.
- If an IP address falls in the 192.168.10.* range, you can to grant access to the user.

Options in a filter aren't related to each other and you can perform multiple actions on an IP address. For example, both the second and third conditions in the example could be true, as an IP address in the 192.168.10.* range could both come from North America and be an internal user who was granted access. The filtered data set is given a named result in the format `name="filter_1:condition_1"` and passed to the next block for processing.

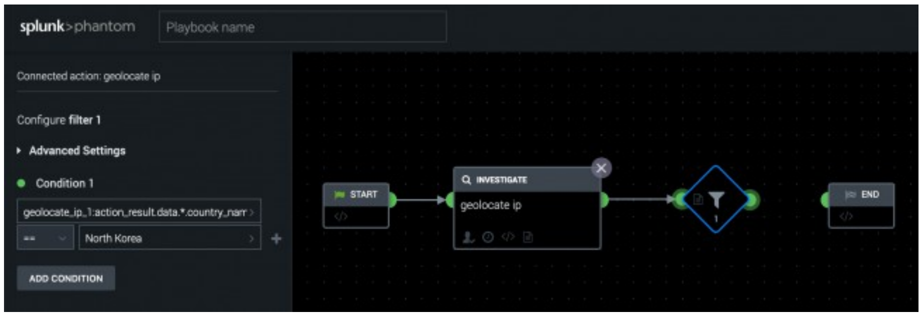
Create a Filter block in your playbook

To create a **Filter** block, perform the following tasks:

1. Create a new block in the classic playbook editor.
2. Select **Filter** from the list of block types.
3. Click the **Select Parameter** field and select the parameter you want to filter on. Parameters are made available to the **Filter** block by upstream blocks.
4. Click in the **==** field and select an operator for the filter.
5. Click the **Select Value** field and select the value you want to match. See Example of creating a filter for an example of how these fields work together.
6. (Optional) Click **+** to add parameters to this condition.
7. (Optional) Click **Add Condition** to create another matching condition for the filter. You can have a maximum of five conditions per **Filter** block. Each condition has its own downstream path.

Example of creating a filter

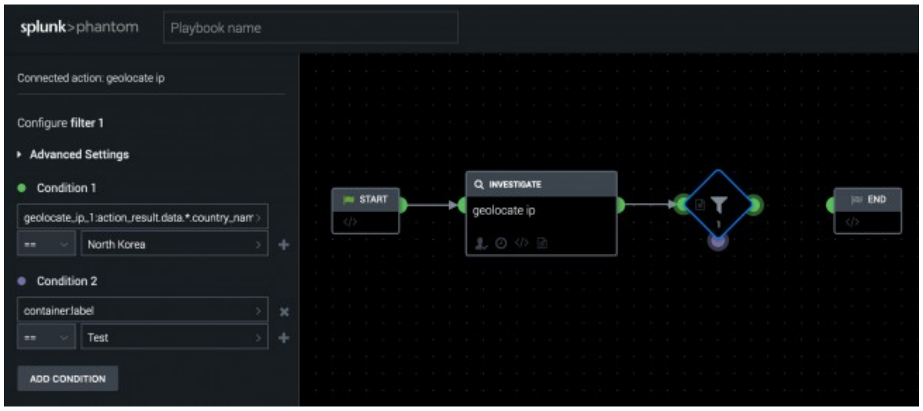
In the following example, you can create a filter to perform a `geolocate ip` action on a source IP address and block any IP addresses from North Korea. The following image shows the classic playbook editor in Splunk SOAR (On-premises).



1. Configure a `geolocate ip` action in the playbook editor.
2. Drag the half-circle icon on the `geolocate ip` action block in the editor.
3. Select **Filter** from the list of block types.
4. Click the **Select Parameter** field and select **geolocate_ip_1**.
5. Select **geolocate_ip_1:action_result_data.*.country_name**.
6. Leave **==** as the operator, and type **North Korea** in the **Select Value** field.

Example of creating a filter with multiple conditions

You can create multiple rows within a condition or multiple conditions. The following image shows an example of the classic playbook editor in Splunk SOAR (On-premises).



Connected action: geolocate ip

Configure filter 1

Advanced Settings

Condition 1

geolocate_ip_1:action_result.data.*.country_nam

== North Korea

Condition 2

container:label

== Test

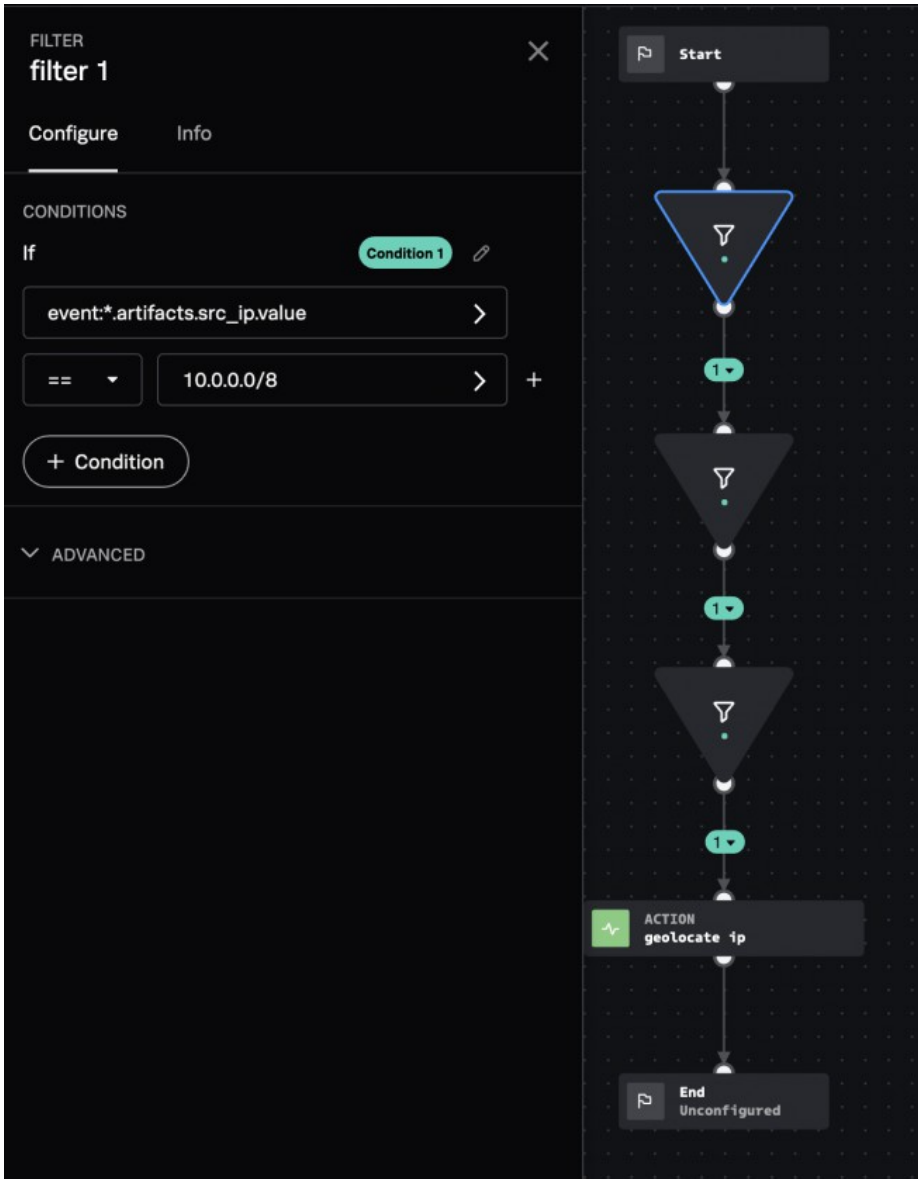
ADD CONDITION

1. Click **Add Condition** to create a second set of filter conditions, which also adds a second output point on the **Filter** block.
2. In the **Select Parameter** field, select **container properties** and choose **label**.
3. Select **==** as the operator, and enter **Test** in the **Select Value** field.

Multiple conditions within a filter block are independent of each other. The results of Condition 1 don't play into the set of inputs for Condition 2. In this example, Condition 1 uses the result from the `geolocate ip` action, while Condition 2 uses a property of the container. Each condition has its own color to make it easier to identify the separate downstream actions. Green marks the path of Condition 1 and purple marks the path of Condition 2.

Example of filter chaining

You can also chain multiple filter blocks together to obtain a more specific set of data. The following image shows a chain of filters in the classic playbook editor.


FILTER

filter 1

Configure Info

CONDITIONS

If Condition 1

event:*.artifacts.src_ip.value

== 10.0.0.0/8

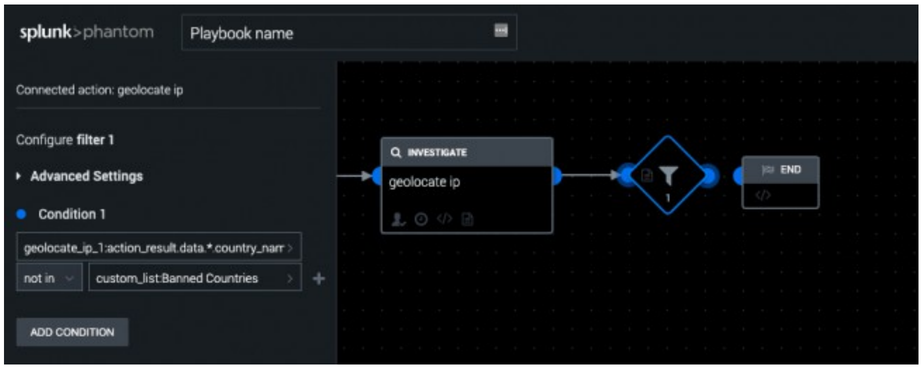
For example, to filter out RFC1918 addresses (10.x.x.x, 172.16.x.x-172.31.x.x, and 192.168.x.x) and perform a `geolocate ip` action on the remaining addresses, perform the following steps:

1. From the **Start** block, create new filter block.
2. In the **Select Parameter** field, select **event > src_ip**.
3. Enter **10.0.0.0/8** in the **Select Value** field.
4. Create a second filter for the 172.16.0.0/12 IP addresses. This filter uses the filtered results from the previous block.
5. Create a third filter for the 192.168.0.0/16 IP addresses. This filter uses the filtered results from the previous block.
6. Create the `geolocate ip` action block on the remaining IP addresses.

If a filter block eliminates all variables while filtering, the downstream action can't run.

Example of using a custom list in a filter

You can use custom lists in your **Filter** blocks to simplify checking against a fixed set of items. For example, instead of checking the source country of an IP address to see if it is North Korea, you can define a list of countries in a custom list, then check the IP address against all of the countries in the list. You can maintain custom lists using the Splunk SOAR (On-premises) web interface, REST API, and playbook API. See Create custom lists for use in Splunk SOAR (On-premises) playbooks in *Use Splunk SOAR (On-premises)*. The following image shows the classic playbook editor in Splunk SOAR (On-premises).



Connected action: geolocate ip

Configure filter 1

Condition 1

geolocate_ip_1:action_result.data.*.country_narr

not in

custom_list:Banned Countries

In this example, use a custom list named **Banned Countries** in a filter by performing the following tasks:

1. Configure a `geolocate ip` action in the playbook editor.
2. Drag the half-circle icon on the `geolocate ip` action block in the editor.
3. Select **Filter** from the list of block types.
4. Click the **Select Parameter** field and select **geolocate_ip_1**.
5. Select **geolocate_ip_1:action_result_data.*.country_name**.
6. Use "in" or "not in" as the operator.
7. Click in the **Select value** field and select **custom_list:Banned_Countries**.

The country name of the source IP address is checked against the countries defined in the **Banned Countries** custom list.

Use decisions to send Splunk SOAR (On-premises) artifacts to a specific downstream action with the classic playbook editor

Use a **Decision** block to change the flow of artifacts by performing IF, ELSE IF, or ELSE functions. When an artifact meets a True condition, it is passed downstream to the corresponding block in the playbook flow. If none of the Decision block conditions are met, the playbook run fails.

In this example, IP addresses found in artifacts are compared against some specific IP address ranges:

- If an IP address is in the 192.168.* range, perform an IP reputation lookup, ELSE IF
- The IP address is in the 172.16.* range, grant access to the user, ELSE
- If the IP address doesn't fall into either of the previous categories, perform a `geolocate ip` action.

The first time an artifact meets a condition, it is passed along to the corresponding downstream block. The artifact is no longer available for evaluation by other ELSE IF or ELSE statements, and cannot be passed to other downstream blocks. You can only perform one action on an artifact based on the condition that is matched first.

Unlike **Filter** blocks, no named datasets are created for reference later on in the playbook.

Create a Decision block in your playbook

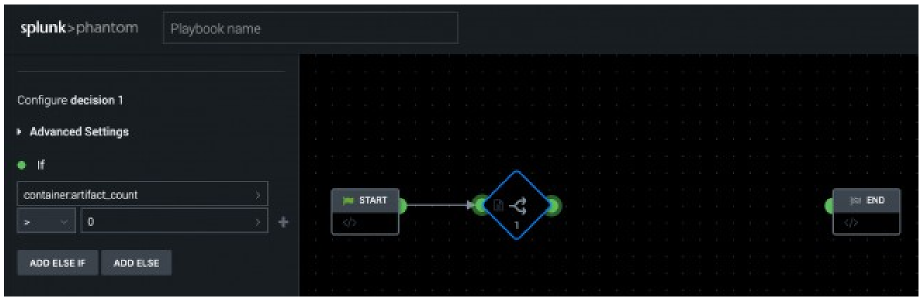
To create Decision block, perform the following tasks:

1. Drag the half-circle icon attached to any existing block in the editor.
2. Select **Decision** from the list of block types.
3. Click the **Select Parameter** field and select the parameter you want to compare. Parameters are made available to the **Decision** block by upstream blocks.
4. Click the **==** field and select an operator for the decision.
5. Click the **Select Value** field and select the value you want to match. See Example of creating decisions for multiple downstream actions for an example of how these fields all work together.
6. (Optional) Click **Add Else If** to create another matching condition for the decision.
7. Click **Add Else** to create the final branch for the decision.

Example of creating decisions for multiple downstream actions

Decision blocks control the program flow based on comparisons of artifact data, notable properties, date functions, and action results. Create `if` and `else if` conditions to branch to multiple downstream blocks as a results of the comparisons.

In the following example, start with a **Decision** block that checks to see if any artifacts are in the notable. The image shows a decision block in the playbook editor.

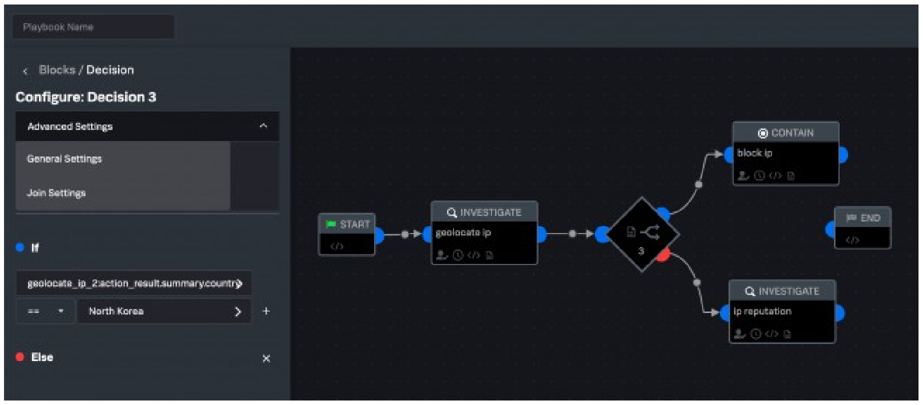


1. Drag the half-circle icon attached to any existing block in the editor.
2. Select **Decision** from the list of block types.
3. In the **If** field, click the **Select Parameter** field and select a parameter to evaluate. You can choose from the properties provided by the container, event data, date and time options, and custom lists.
4. Select **container properties** from the list of options, and then click **artifact_count** as the property you want to evaluate.
5. Select **>** as the operator, and enter **0** in the **Select Value** field.

The blue circle next to the **If** section corresponds with the blue connector dot on the side of the decision block. All data is passed on to the next block.

Example of creating decisions with multiple statements

You can create more complex decision blocks with up to five statements. For example, you can perform a `geolocate ip` action on a source IP address and block the IP if the country is from North Korea. Otherwise, you can perform an `ip reputation` action on the IP address, as shown in the following screenshot:



Each subsequent statement and downstream block is color coded: blue marks the path of the **If** statement, and red marks the path of the **Else** statement. Each statement has its own and only one downstream block.

Customize the format of your Splunk SOAR (On-premises) playbook content using the classic playbook editor

Use the **Format** block to craft custom strings and messages from various objects.

You might consider using a **Format** block to put together the body text for creating a ticket or sending an email. Imagine you have a playbook set to run on new containers and artifacts that does a basic lookup of source IP address artifacts. You want to take the results of that lookup, format the results, and send the information as an email. You craft your playbook so that the action results are available to the format block.

To configure a **Format** block, perform the following steps:

1. Create a new block in the Classic Playbook Editor.
2. Select **Format** from the list of block types.
3. Configure the template parameter variables in the **Template Parameters** field. The first variable is identified as `{0}`, the next as `{1}`, and so on. You can select any event and container properties. You can also select data from any upstream block.
4. In the **Template** field, craft a message using the variables you define.

Example of defining a template

This example defines a template in the following manner:

```
IP address: {0}
IP address country: {1}
IP address reputation: {2}
```

The message returned as a result of this template looks like the following:

```
IP address: 1.2.3.4
IP address country: United States
IP address reputation: Malicious
```

If multiple events are picked up at the same time, you see the following message:

```
IP address: 1.2.3.4, 10.11.12.13
IP address country: United States, Turkey
IP address reputation: Malicious
```

You can wrap `%%` around a formatting block to make each set of values output on its own line. For example:

```
%%
The IP address {0} originates from {1}.
%%
```

Wrapping `%%` around a formatting block produces results like the following:

```
The IP address 1.2.3.4 originates from United States.
The IP address 10.11.12.13 originates from Turkey.
```

Example of using the Python str.format() function to create more advanced templates

More complicated formatting is supported using all the capabilities of the Python `str.format()` function. The following template demonstrates double curly brackets to support curly bracket escaping, automatic escaping of backslashes and quotes, and centered string alignment with a custom filler character:

```
JSON Formatting:
{{"notable_id":"{0}","event_count":{1}}}

Backslashes and quotes are escaped:
"ls /Applications/System \Preferences.app/"

Python formatting is supported:
|{0:-^50}|{1:-^50}|
|{2:-^50}|{3:-^{3}{2}{1}{3}{2}{1}50}|
```

The template produces output as in the following example:

```
JSON Formatting:
{"notable_id":  "1004", "event_count": 10}

Backslashes and quotes are escaped:
"ls /Applications/System \Preferences.app/"

Python formatting is supported:
|-----------------------1004-----------------------|-----------------------10-----------------------|
|----------------------events----------------------|-------Zeus infection on HQ finance server------|
```

If a new line is needed, use \n.

Require user input to continue running the Splunk SOAR (On-premises) playbook using the classic playbook editor

You can configure a task or prompt in your Splunk SOAR (On-premises) playbook that must be acknowledged by a user before further actions in the playbook are run. You can configure the following types of user input in a playbook:

- A manual task using a **Manual Task** block that must be acknowledged by a user.
- A prompt using a **Prompt** block that must be acknowledged by a user. You can configure a specific response type with a **Prompt** block.

Require user input using the Manual Task block in your playbook

Use a **Manual Task** block to send a message to a Splunk SOAR (On-premises) user or group that they must acknowledge. This is the same as manually running a task action from the **Investigation** menu.

To configure a manual task, perform the following tasks:

1. Drop a new block onto the playbook editor.
2. Click on the block, then select **Manual Task** from the block types.
3. Select an **Approver** from the drop-down list. If the task is assigned to a group of users, the first user to process it completes the task.
4. From the **Required response time** field, choose the response time in minutes.
5. In the **Message** box, craft a meaningful message so the users receiving the message understand what actions they must take.

Require user input using the Prompt block in your playbook

Use a **Prompt** block in your playbook to send a message to a user or group that they must acknowledge.

To configure a prompt, perform the following tasks:

1. Drop a new block onto the playbook editor.
2. Click on the block, and then select **Prompt** from the block types.
3. Select an **Approver** from the drop-down list. If the task is assigned to a group of users, the first user to process it completes the task.
4. From the **Required response time** field, choose the response time in minutes.
5. In the **Message** box, craft a meaningful message so the users receiving the message understand what actions they must take. Markdown is supported.
6. From the **Responses** drop-down list, choose the type of response required to complete the task. If the response type is **Message**, markdown is supported.

See https://guides.github.com/features/mastering-markdown/ for more information on the type of Markdown that can be used in the Message box.

Set container parameters in Splunk SOAR (On-premises) using the API block with the classic playbook editor

Use the **API** block to set parameters of the container it's running in. For example, you can use an API call to set the severity of a container.

Perform the following tasks to configure an **API** block:

1. Drop a new block onto the playbook editor.
2. Click on the block, and then select **API** from the block types.
3. Select the API property you want to set. The following table summarizes the properties that you can set:

Property	Description
label	The label of the container. The drop-down list shows all of the container labels currently available on your Splunk SOAR (On-premises) instance.
sensitivity	The sensitivity of the container.
severity	The severity of the container.
status	The status of the container, such as Resolved.
owner	The owner of the container.
add list	One of two API calls that doesn't operate directly on the container. The `add list` property takes two parameters: the list that you want to add to, and the data you are adding. If the list doesn't exist, it is created by Splunk SOAR (On-premises). You can point the data field to a variable by selecting from the properties, results, and artifacts, or you can type in a fixed string.
remove list	One of two API calls that doesn't operate directly on the container. The `remove list` property takes a list name as the single parameter, and deletes that list when it has run.
pin	Pin data to the heads-up display (HUD) in the container. This property takes the following parameters: ♦ Data ♦ Message ♦ Pin Type

Property	Description
	♦ Pin Style
add tag	The API call used to add a tag to the container.
remove tag	The API call to remove a tag from the container.
add comment	The API call used to add a comment to a container. You can either supply a variable or a static string in the input.
promote to case	The API call used to promote the container to a case. It takes a single parameter, the case template you can pick from a drop-down list.
add note	The API call used to add a note. It takes the parameters title, content, and note format. With the note format parameter, you can choose either HTML or Markdown.

You can configure multiple API calls in any API block. For example, you can set the label, severity, and status of a container using one API block.

4. Click **Save** to save the settings. A check mark appears next to the API calls that you configured.

Run other Splunk SOAR (On-premises) playbooks inside your playbook using the classic playbook editor

You can configure your playbook to run another existing playbook. Call one playbook from another playbook to avoid having to copy and maintain code in different places.

To configure your playbook to run another playbook:

1. Drop a new block onto the playbook editor.
2. Click on the block, then select **Playbook** from the block types.
3. In the **Playbook** field, select the playbook you want to run from the drop-down list.
4. (Optional) Toggle the **Synchronous** switch on to make this playbook wait for the called playbook to complete running before continuing.

Playbooks differ from action blocks in the following ways:

- The playbook continues to downstream blocks regardless of whether the called playbook is successful.
- The called playbook doesn't return any values that are used in downstream blocks.
- The called playbook doesn't determine the data set, and it operates on the container data with the scope inherited from the caller.
- The called playbook runs independently from the caller. If you wire a series of playbooks to run, they are processed in parallel if the **Synchronous** switch is left off. See Determine your Splunk SOAR (On-premises) playbook flow.

Add custom code to your Splunk SOAR (On-premises) playbook with the custom function block using the classic playbook editor

Use custom functions to expand the functionality of your playbooks in Splunk SOAR (On-premises). Custom functions enable you to use your Python skills to expand the kinds of processing performed in a playbook, such as applying string transformations, parsing a raw data input, or calling a third party Python module. Custom functions can also interact with the REST API in a customizable way. You can share custom functions across your team and across multiple playbooks to increase collaboration and efficiency. To create custom functions, you must have **Edit Code** permissions, which can be configured by an Administrator in **Administration > User Management > Roles and Permissions**. For more information on the **Edit Code** permission, see Add a role to Splunk SOAR (On-premises) in the *Administer Splunk SOAR (On-premises)* manual.

Create a custom function

To create a custom function, follow these steps:

1. Navigate to your Splunk SOAR (On-premises) instance.
2. From the **Home** menu, select **Playbooks**.
3. Click the **Custom Functions** tab.
4. Click the **+ Custom Function** button.
5. Create a name for your custom function. Once you save your custom function, the name can only be changed using the **Save As** button as playbooks reference custom functions by name.
6. (Optional) Add a description. Descriptions explain what the function is used for. Descriptions appear in both the custom function listing page and in the Classic Playbook Editor.
7. Add any desired custom Python code in the editor.
8. Click **Validate** to check if your Python code is valid.
9. Click **Save** and select a repository to save your custom function to.
   Clicking the Save button automatically validates your Python code. Add a commit message in the pop-up window about your function.
10. (Optional) After saving your custom function, you can make a copy of it.
    1. To make a copy, first click **Edit** and then click the arrow next to the **Save** button.
    2. Click **Save As** and create a name, select the repository you want your custom function saved to, and write a commit message.
    3. Press **Save** again to save it as a copy.
11. (Optional) After saving your custom function, you can click the more icon to view the documentation, export the function, view the audit log, or view the history of the function.

Functions, input parameters, and output variables must be named using valid Python identifiers: A-Z, a-z, 0-9, and underscores.

Add an input parameter to a custom function

Adding input parameters to your custom function is optional. Each input parameter can be populated with a data path or a literal string when calling the custom function from a playbook. You can set a data path from any valid blocks upstream, artifact data, and container data.

To add an input to a custom function, follow these steps:

1. From the New Custom Functions page, click **Add Input**.
2. Enter a **Variable*** name that is a valid Python identifier.
3. Select an input type. Select **List** to provide an entire collection. Selecting **List** causes the Classic Playbook Editor to generate code that passes the entire collection to the custom function without looping. For example, if there is a container with multiple artifacts, each with a destination address Common Event Format (CEF) field, and you configure a list type input, the custom function is called once and passes in a list of all destination addresses as shown:

```
 container_data_0 = phantom.collect2(container=container,
datapath=['artifact:*.cef.destinationAddress', 'artifact:*.id'])
```

```
 parameters = []
 container_data_0_0 = [item[0] for item in container_data_0]
 parameters.append({
     'list_type_input': container_data_0_0,
 })
```

Select **Item** to provide a single item from a collection. Selecting **Item** causes the classic playbook editor to generate code that loops over items in the collection, calling the custom function for each item. For example, if there is a container with multiple artifacts, each with a destination address CEF field, and you configure an item type input with a destination address of those artifacts, the custom function is called once per destination address as shown:

```
container_data_0 = phantom.collect2(container=container,
datapath=['artifact:*.cef.destinationAddress', 'artifact:*.id'])
parameters = []
for item0 in container_data_0:
    parameters.append({
       'item_type_input': item0[0],
    })
```

4. (Optional) Select a CEF data type. When you are populating an input, this CEF data type is the default filter limiting the values that are shown.
5. (Optional) Add a custom placeholder. The custom placeholder you create appears as the input placeholder in the classic playbook editor configuration panel.
6. (Optional) Add help text. Help text is a longer description that appears as a tooltip on the input in the classic playbook editor.
7. (Optional) Click **Add Input** to add another input. You can add a maximum of 10 inputs to a custom function.

A single custom function can have a mix of item and list type inputs. When the input types are mixed, the custom function loops over all the item type inputs and passes the list type inputs to every custom function call.

Add an output variable to a custom function

Adding output variables to your custom function is optional. Output variables are usable as inputs in other downstream blocks, such as **Action**, **API**, **Filter**, **Decision**, **Format**, **Prompt**, or other **Custom Function** blocks. A custom function output variable gets published as a data path in this format: `<block_name>:custom_function_result.data.<output_variable>`. Make sure to give your output variables clear and meaningful names in your custom code.

To add an output to a custom function, follow these steps:

1. Click **Add Output**
2. Enter a Data Path. Write the full path to your output.
3. Enter the output in the code in your output object.
4. Select a data type. The data type is the Common Event Format (CEF) field that you want to use.
5. Add a description. The description appears as help text in the classic playbook editor.
6. Click **Add Output** to add another output. You can add up to 10 outputs to a custom function.

Splunk Soar doesn't validate the output data paths. Because of this, you need to make sure the output data path is valid. It is possible to have valid data paths that are not configured in the UI.

Using draft mode with custom functions

Draft mode is a good way to save custom functions that you are in the process of creating. If you have created a function that isn't yet valid, it will automatically be saved in draft mode. You can only have one draft of an original custom function at a time. Use draft mode custom functions in playbooks to help test or debug the custom function.

You can use draft mode in the following ways:

1. Toggle draft mode on and off in the custom function editor.
2. When you make an error in the Python code and try to save it, an error message displays letting you know that you can only save it in draft mode. To save it as a draft, follow these steps:
    1. If it is your first time saving the custom function, select the repository where you want to save it.
    2. Enter a commit message.
    3. Click **Save as draft.** In draft mode, you can toggle between the original version and the draft version by clicking the **View original** or the **View draft** button.

Once you have finished making your edits, you can take the function out of draft mode by following these steps:

1. Toggle Draft Mode to **Off**.
2. Click **Save.** If you have any associated playbooks, a message appears alerting you about each of the playbooks using the draft custom function, or the custom function that you are overriding.
3. Click **Next.**
4. Enter a commit message.
5. Click **Save.**

Switching Draft Mode to Off deletes the draft copy of the custom function and removes all playbook associations. When you take a custom function out of Draft Mode, any playbooks using the draft version must be updated as the custom function they refer to no longer exists.

Delete a custom function

To delete a custom function, follow these steps:

1. Navigate to the Custom Functions listing page.
2. Check the box next to the custom function you want to delete.
3. Click **Delete.** If a custom function is being used in any playbooks, a warning displays listing all the associated playbooks for that custom function. You can switch active playbooks to inactive before confirming the deletion of a custom function.
4. Click **Delete.**

You can't bulk delete custom functions.

Add a custom function to a playbook

Once you have created a custom function, add it to a playbook by following these steps:

1. From the **Home** menu, click **Playbooks**.
2. Click the **+ Playbook** button.
3. To add a new block to a playbook, drag the half-circle icon attached to any block on the canvas. Release your mouse to create a new empty block connected to the originating block with an arrow.

4. Click the **Custom Function** button to add a custom function block.
5. Click in the search bar to display all of your repositories. Then, click the repository your custom function is saved to and either search for your custom function, or select it from the list.
6. (Optional) Test the draft version of your custom function by selecting it from the list and adding it to the playbook.
7. (Optional) Once you have selected a custom function, you can configure the value of the input parameters. Keyword argument datapaths are available to add as input parameters by clicking **keyword arguments** and then selecting an argument from the drop-down list. Keyword argument datapaths are only available for custom functions as they aren't valid datapaths and can't be passed to Splunk SOAR (On-premises) APIs. For more information see Understanding datapaths and Playbook automation API in the *Splunk SOAR (On-premises) Playbook API Reference*.
8. (Optional) Click the **+ New** button in this panel to navigate to the custom function editor and create a new custom function (link). Once you save a new custom function, you can select it from this menu.
9. (Optional) Click the **Edit** button to navigate to the custom function editor where you can make any needed edits. Once you save your edits, you need to reselect the custom function to use your changes.
10. (Optional) Click **Refresh List** to update the list of available custom functions.
11. When you are finished editing your playbook, click **Save** to enter your desired settings and playbook name.
12. Click **Save** again then choose a source control to save your playbook to and enter a comment about your playbook.

To learn more about editing Python code in the classic playbook editor, see View or edit the Python code in Splunk SOAR (On-premises) playbooks.

Once you have added a custom function to a playbook, you can see the associated playbooks used by a custom function in the custom function configuration panel.

Write useful custom functions

Structure custom function outputs like action results whenever possible. This means the preferred data structure should be a list of dictionaries, with a meaningful name for each field in the dictionary. For example, if you are converting strings from upper case to lower case in a custom function block named **lower_1**, your custom function might output something like the following:

```
{
        'items': [{"lower_case_string": "abcd"}, {"lower_case_string": "efgh"}]
        'total_lowered_count': 2
}
```

Based on the output you could access the data through the following datapaths. In the Custom Function datapaths column, for the full path to each datapath, it is prefixed with `lower_1:custom_function_result.data`. In the Collect2 results column, it shows what the results of using the full datapaths passed into a downstream phantom.collect2 call would look like.

Custom function datapaths	Collect2 results
`items`	`[[{"lower_case_string": "abcd"}, {"lower_case_string": "efgh"}]]`
`items.*.lower_case_string`	`["abcd", "efgh"]`
`items.0.lower_case_string`	`["abcd"]`
`total_lowered_count`	`[2]`

Custom function datapaths	Collect2 results
items.*.lower_case_string, total_lowered_count	[["abcd", 2], ["efgh", 2]]

Here's the custom function that produced the output above:

```
def lower(values=None, **kwargs):
    """
    Args:
        values

   Returns a JSON-serializable object that implements the configured data paths:
        items.*.lower_case_string (CEF type: *)
    """
    ############################ Custom Code Goes Below This Line #################################
    import json
    from phantom.rules import debug
â
    if values is None:
        values = []

    # Write your custom code here...
    items = []
    for value in values:
        item = None
        try:
            item = value.lower()
        except:
            pass
        finally:
            items.append({
                'lower_case_string': item
            })

    outputs = {
            'items': items,
            'total_lowered_count': len(items),
    }
â
    # Return a JSON-serializable object
    assert json.dumps(outputs)  # Will raise an exception if the :outputs: object is not JSON-serializable
    return outputs
```

Use datetime objects with custom functions

JSON is the primary language playbooks use to communicate. As such, values that aren't JSON encodable must be encoded into JSON encodable representations before they can be serialized to JSON. The playbook API uses Python's built in JSON module for encoding and decoding JSON. However, this module does not support datetime.datetime objects without a custom encoder or decoder.

To use datetime.datetime objects, always pass encoded data between blocks. These objects can be encoded as ISO strings. The following custom function shows how to decode a datetime.datetime object, modify the object, and re-encode the object.

```
def add_one_day(iso_formatted_datetime=None, **kwargs):
    """
    Adds one day to an iso formatted datetime.
    ...

    Args:
        iso_formatted_datetime: An ISO-formatted datetime, e.g. '2020-05-18T20:11:39.198140+00:00'


    Returns a JSON-serializable object that implements the configured data paths:
        iso_formatted_datetime (CEF type: <output_contains>): An ISO formatted datetime which shall be one
day later than the input ISO-formatted datetime.

    """
    ############################ Custom Code Goes Below This Line #################################
    import json
    from phantom.rules import debug

    from datetime import datetime, timedelta
    from pytz import timezone
    import dateutil.parser

    outputs = {}

    # Write your custom code here...
    utc = timezone('UTC')

    # Decode the ISO-formatted string into a native python datetime
    dt = dateutil.parser.parse(iso_formatted_datetime).replace(tzinfo=utc)

    # Add one day
    dt += timedelta(days=1)

    # Encode data before returning it
    augmented_iso_formatted_datetime = dt.isoformat()

    # Set the output
    outputs['iso_formatted_datetime'] = augmented_iso_formatted_datetime

    # Return a JSON-serializable object
    # This assertion would not pass without encoding the dt variable.
    assert json.dumps(outputs)  # Will raise an exception if the :outputs: object is not JSON-serializable
    return outputs
```

Playbook APIs supported from within a custom function

The following are lists of playbook APIs that are supported from within a custom function. For more information on playbook APIs, see Playbook automation API in the *Splunk SOAR (On-premises) Playbook API Reference* manual.

Playbook automation APIs

View a list of playbook automation APIs supported from within custom functions.

- debug
- error
- format
- render_template

Container automation APIs

View a list of container automation APIs supported from within custom functions.

- add_artifact
- add_note
- add_tags
- add_task
- close
- comment
- create_container
- delete_artifact
- delete_pin
- get_notes
- get_phase
- get_tags
- get_tasks
- merge
- pin
- promote
- remove_tags
- set_duetime
- set_label
- set_owner
- set_phase
- set_sensitivity
- set_severity
- set_status
- update
- update_pin

Data management automation APIs

View a list of data management automation APIs supported from within custom functions.

- add_list
- delete_from_list
- get_list
- get_run_data
- remove_list
- set_list

Data access automation APIs

View a list of data access automation APIs supported from within custom functions.

- collect
- collect_from_contains
- collect2
- get_action_results
- get_container
- get_custom_function_results
- get_filtered_data
- get_format_data
- parse_errors
- parse_success

Session automation APIs

View a list of session automation APIs supported from within custom functions.

- build_phantom_rest_url
- get_base_url
- get_phantom_home
- get_rest_base_url
- requests

Vault automation APIs

View a list of vault automation APIs supported from within custom functions.

- vault_add
- vault_delete
- vault_info

Network automation APIs

View a list of network automation APIs supported from within custom functions.

- address_in_network
- attacker_ips
- victim_ips

Use the Splunk SOAR (On-premises) REST API from within a custom function

In addition to using playbook APIs, you can also use the Splunk SOAR (On-premises) REST API from within a custom function. This is helpful for reading and writing objects in Splunk SOAR (On-premises) that are not accessible through the playbook APIs. For example, if you had multiple playbooks that always start with the same three blocks, you can set the status to **In Progress**, set the severity to **Needs Triage**, and add a comment to the container with the text: "automation started." You can then write a custom function named "setup" that performs the previously mentioned tasks, but replaces the three blocks with a single custom function block. This allows you to get the same playbook functionality with fewer blocks and gives you the ability to modify the status, severity, or comment value in a single place that then updates all of your playbooks. See the *REST API Reference for Splunk SOAR (On-premises)* to learn more about the capabilities of the Splunk SOAR (On-premises) REST API and the specific syntax needed for each type of request.

The Playbook API provides the following methods to call the REST API and to avoid storing credentials in the custom function. For more information on the Playbook API, see the *Splunk SOAR (On-premises) Playbook API Reference*.

Playbook API	Description
phantom.build_phantom_rest_url()	Combine the Splunk SOAR (On-premises) base URL and the specific resource path, such as /rest/artifact.
phantom.requests	Use methods such as get and post from the Python 'requests' package to perform HTTP requests. If you use `requests` you don't need to set the session by hand.

The following example combines these methods to query the tags of an indicator with a given ID.

```
def list_indicator_tags(indicator_id=None, **kwargs):
    """
    List the tags on the indicator with the given ID
    Args:
        indicator_id: The ID of the indicator to list the tags for

    Returns a JSON-serializable object that implements the configured data paths:
        tags: The tags associated with the given indicator
    """
    ############################ Custom Code Goes Below This Line #################################
    import json
    import phantom.rules as phantom
    outputs = {}

    # Validate the input
    if indicator_id is None:
        raise ValueError('indicator_id is a required parameter')

    # phantom.build_phantom_rest_url will join positional arguments like you'd expect (with URL encoding)
    indicator_tag_url = phantom.build_phantom_rest_url('indicator', indicator_id, 'tags')

    # Using phantom.requests ensures the correct headers for authentication
    response = phantom.requests.get(
        indicator_tag_url,
        verify=False,
    )
    phantom.debug("phantom returned status code {} with message {}".format(response.status_code,
response.text))

    # Get the tags from the HTTP response
    indicator_tag_list = response.json()['tags']
    phantom.debug("the following tags were found on the indicator: {}".format(indicator_tag_list))
    outputs['tags'] = indicator_tag_list

    # Return a JSON-serializable object
    assert json.dumps(outputs)  # Will raise an exception if the :outputs: object is not JSON-serializable
    return outputs
```

Use the REST API carefully if you are running queries across large data sets, such as containers, artifacts, indicators, and action results, or when you are working with a long-running Splunk SOAR (On-premises) instance with a lot of accrued data.

To improve the performance of these queries, you can apply strict filters, limit the page size of the query, restrict the query to objects created within a certain date range, or only query for a specific object detail instead of the entire object. To see the full set of available query modifiers, see Query for Data in the *REST API Reference for Splunk SOAR (On-premises) Manual.*

Use Python packages with custom functions

To write a custom function that uses a Python package that's not installed by default on your system, perform the following steps using the pip executable appropriate to the Python version of your custom function. These steps use pip3.

Using `pip3` is a shorthand for the SOAR system's currently running version within python3 like python3.9. If you need to use a specific version of python3 for your custom function, like python3.#, specify that specific version, like `pip3.#`.

1. Log in to Splunk SOAR (On-premises) using SSH as a user with sudo access. For more information, see Log in to Splunk SOAR (On-premises) using SSH.
2. (Optional) Determine if the needed Python packages are already installed on your system using `phenv pip3 freeze | grep -i <package_name>`.

3. Custom functions use the same Python version interpreter as playbooks, so set the user's permissions and add the package to the appropriate Python path by calling the following commands:
    1. `umask 0022`
    2. `sudo phenv pip3 install <package_name>`
4. Import the package into the custom function using `import <package_name>`.

Legacy custom functions versus new custom functions

Legacy custom functions are the custom functions that were introduced in Splunk Phantom version 4.2. See the following section for information on how to update your custom functions to the new version. In Splunk Phantom 4.9, legacy custom functions and new custom functions are both options in the classic playbook editor to make migration to the new custom functions as painless as possible. The main difference between the two versions is that the new custom functions support a library of source controlled custom functions that can each be easily reused and called from multiple playbooks, while legacy custom functions are not reusable and must be written once per custom function block in a playbook.

Convert legacy custom functions to new custom functions

Version 4.9 of Splunk Phantom added support for a library of custom functions that can each be called from multiple playbooks. When converting a legacy custom function to a new custom function, certain Splunk SOAR (On-premises) APIs can't be called from within a new custom function. If this occurs, use the custom code section of the custom function block code in the classic playbook editor to make those API calls. You can also use this section to override any of the `parameters` before the `phantom.custom_function` call is made. To see a complete list of APIs supported from within a custom function, see Playbook APIs supported from within a custom function.

To convert any of your legacy custom functions to the current version of custom functions, follow these steps:

1. Copy the legacy custom functions inputs and outputs using the custom function editor. If you're using an API that isn't supported from within a custom function, you can't call it from within the custom function, but you can call it from the custom code section available in the classic playbook editor in the custom function block code.
2. Select your input type. If you want your inputs to be the same as the legacy custom function inputs, configure them as list-type inputs. For more information on input types, see Add an input parameter to a custom function.
3. Copy your code from the playbook editor into the custom function editor, and change the input names to the names of your newly configured custom function inputs.
4. Structure your outputs into the output object that the custom function publishes.
5. Change any downstream blocks that use a legacy custom function output to take the new custom function data path.

Troubleshoot custom function result failures

Your custom function result will be marked as failed if it raises an uncaught exception. This is shown in the following custom function by the message `raise RuntimeError('Try again next time.')`.

```
def fail_fast():
    raise RuntimeError('Try again next time.')
```

Custom functions also produce a failed result when the input is not specified or the input denominator is 0, which causes a `ZeroDivisionError` as shown in the following function:

```
def divide(numerator=None, denominator=None):
    if numerator is None:
        raise ValueError('"numerator" is a required parameter')
    if denominator is None:
        raise ValueError('"denominator" is a required parameter')

    return {
        'result': numerator / denominator  # Raises a ZeroDivisionError if denominator is 0
    }
```

You can test if your custom function was successful or not by using a decision block in the playbook editor.

Custom function run failures

A custom function run contains a collection of intermediate custom function results. If some of the custom function results in the run fail, the overall custom function run will still succeed as long as some of the functions in the run succeed. The following call to `phantom.custom_function` shows a successful custom function run where the first three custom function results succeed, and the last custom function result failed:

```
phantom.custom_function('local/divide', parameters=[
    {'numerator': 8.0, 'denominator': 4},
    {'numerator': 8.0, 'denominator': 2},
    {'numerator': 8.0, 'denominator': 1},
    {'numerator': 8.0, 'denominator': 0},
])
```

The following example shows a custom function run that failed because each of the custom function results failed.

```
phantom.custom_function('local/divide', parameters=[
    {'numerator': 8.0, 'denominator': 0},
    {'numerator': 2.0, 'denominator': 0},
])
```

Keyboard shortcuts in the custom function editor

The following keyboard shortcuts can be used in the custom function editor.

Description	Shortcut
Save	Cmd+S
Edit mode	Cmd+E

If you are using a Windows machine, replace Cmd with Ctrl in the keyboard shortcuts.

Add custom code to your Splunk SOAR (On-premises) Playbook with the legacy custom function block using the classic playbook editor

Legacy custom function blocks are the custom functions that were introduced in Splunk Phantom version 4.2 and are supported for playbooks which use those blocks. A newer implementation was released in Splunk Phantom version 4.9 and should be used instead. For information on converting legacy custom functions to the current custom function implementation, see Convert legacy custom functions to new custom functions.

Add custom Python code to a **Custom Function** block. Custom functions enable you to expand the kinds of processing performed in a playbook, such as adding custom input parameters and output variables. To use a legacy custom function block, you must have **Edit Code** permissions, which can be configured in **Administration > User Management > Roles and Permissions**. For more information on the **Edit Code** permission, see Add a role to Splunk SOAR (On-premises) in the *Administer Splunk SOAR (On-premises)* manual.

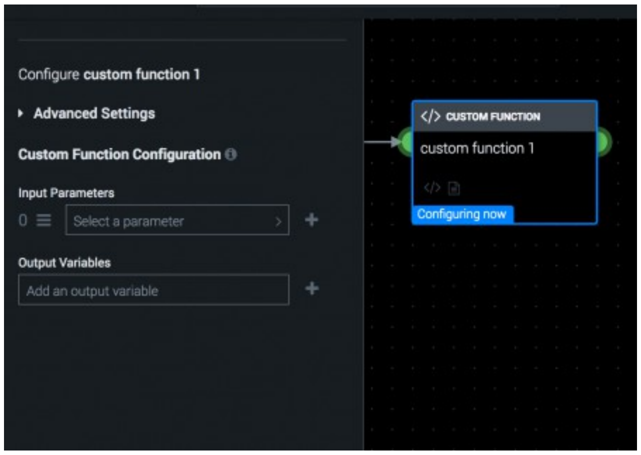
Functions, parameters, and output variables can all have custom names using the following characters:

- A-Z
- a-z
- 0-9
- Underscores

Setting or changing a custom name changes that custom name in all data paths that use it, including generated and custom code.

Set the name, tooltip, comments and other options for the block in the **Advanced Settings > General Settings**.

Click the **</>** icon to open the Python Playbook Editor to add your custom code:



Input parameters

Input parameters represent a data path. You can set a data path from any valid blocks upstream, artifact data, and container data.

To create or remove an input parameter, perform the following steps:

1. Click the **+** icon to add an input parameter. The index of parameters starts at zero.
2. Type a name to set the name for or rename the input parameter.
3. Click the **>** icon to set the properties for the input parameter.

You can delete the input parameter by clicking the **x** icon.

Output variables

Output variables are usable as inputs in other downstream blocks, such as **Action**, **API**, **Filter**, **Decision**, **Format**, **Prompt**, or other **Custom Function** blocks. The name of an output variable becomes `<block_name>__<variable_name>` in the auto-generated section of the playbook code. Give your output variables clear and meaningful names in your custom code.

Follow these steps to create or remove an output variable:

1. Click the **+** icon to add an output variable.
2. Type a name to set the name for, or rename the output variable.
3. Click the **x** icon to delete an output variable.

See the following example of a custom function:

```
def format_login(action=None, success=None, container=None, results=None, handle=None,
filtered_artifacts=None, filtered_results=None):
phantom.debug('format_login() called')
results_data_1 = phantom.collect2(container=container,
datapath=['get_login_history:action_result.data.*.output'], action_results=results)
results_item_1_0 = [item[0] for item in results_data_1]

format_login__login_table = None

################################################################################
## Custom Code Start
################################################################################

# format the output into Jira's markup language for rendering a table
format_login__login_table = "|| output of '/usr/bin/last -a' ||\n"
last_lines = results_item_1_0[0].split('\n')
for line in last_lines:
    format_login__login_table += "| {} |\n".format(line)

phantom.debug("table of logins for jira:")
phantom.debug(format_login__login_table)

################################################################################
## Custom Code End
################################################################################

phantom.save_run_data(key='format_login:login_table', value=json.dumps(format_login__login_table))
enrich_ticket_1(container=container)

return
```

Custom function example for processing multiple artifacts to build a parameter list

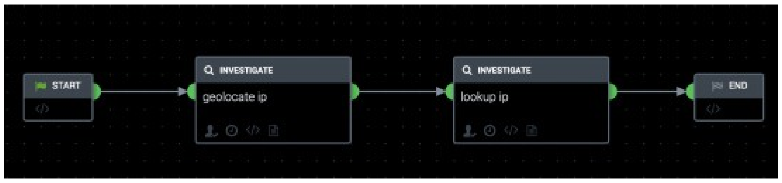
```
def DBSearch(action=None, success=None, container=None, results=None, handle=None, filtered_artifacts=None,
filtered_results=None):
    phantom.debug('SearchDB() called')
    name_value = container.get('name', None)
    container_data = phantom.collect2(container=container, datapath=['artifact:*.cef.message',
'artifact:*.cef.startTime', 'artifact:*.id'])
    container_item_0 = [item[0] for item in container_data]
    container_item_1 = [item[1] for item in container_data]

    ################################################################################
    ## Custom Code Start
    ################################################################################

    # Write your custom code here...
    customernamestr = name_value
    parameters = []
    # Loop over the container data structure (a list of lists, with each inner list of length three)
    for messagestr, start_time, artifact_id in container_data:
        startdatestr = start_time.split(' ')[0]
        starttimestr = start_time.split(' ')[1]

        # Build the SQL
        if 'groupName:' in messagestr:
            hostgroupstr = messagestr.split('ï¼ ')[0].replace('groupName:', '')
            hoststr = messagestr.split('ï¼ ')[1].split(':')[1]
            sqlstr = "select COUNT(*) as cnt from schedule where customer = '"+ customernamestr +"' and
startdate <= '"+ startdatetimestr +"' and enddate >= '"+ startdatetimestr +"' and ("
            for group in hostgroupstr.split('/'):
                sqlstr = sqlstr + "kyoten like '%"+ group +"%' or reason like '%"+ group +"%' or "
            sqlstr = sqlstr + "kyoten like '%"+ hoststr +"%' or reason like '%"+ hoststr +"%')"

        else:
            hoststr = messagestr.split(':')[0]
            sqlstr = "select COUNT(*) as cnt from schedule where customer = '"+ customernamestr +"' and
 (kyoten like '%"+ hoststr +"%' or reason like '%"+ hoststr +"%') and startdate <= '"+ startdatetimestr +"'
and enddate >= '"+ startdatetimestr +"'"

        # Update the parameter list
        # There should be one parameter per item in the container_data variable
        # There should be one item in the container_data variable per artifact
        # Thus, there should be one parameter per artifact
        parameters.append({
            'query': sqlstr,
            'format_vars': "",
            'no_commit': False,
        })

    phantom.act("run query", parameters=parameters, assets=['mysql'], callback=filter_2, name="SearchDB")

    ################################################################################
    ## Custom Code End
    ################################################################################
    filter_2(container=container)

    return
```

Determine your Splunk SOAR (On-premises) playbook flow using the classic playbook editor

The order in which you arrange the blocks and lines in your playbook determine the playbook flow.

Process playbook blocks serially

Serial processing means playbook blocks are performed in the order they are arranged, as shown in the following screenshot:



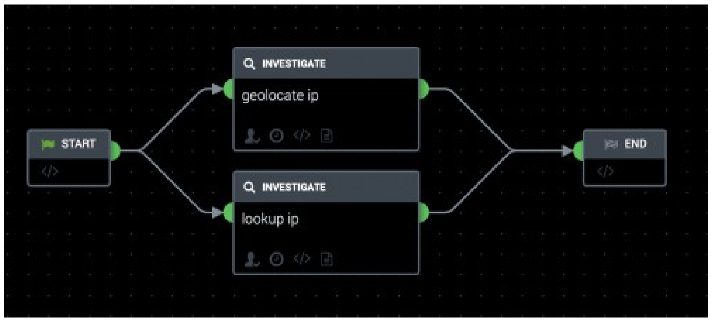
In this example, the blocks perform as described:

1. A `geolocate ip` is performed on a source IP address.
2. When the `geolocate ip` action is finished, a `lookup ip` performs.

Use serial processing when the operations must happen in a specific order, such as when a downstream block depends on the results from an upstream block.

Processing playbook blocks in parallel

You can also wire blocks to process in parallel, as shown in the following example:



In this case, the `geolocate ip` and `lookup ip` actions perform simultaneously, and either action can finish first. You can wire blocks in this manner when you have no dependencies on the completion of either block, or if there are no dependencies between the blocks themselves.

Arranging blocks in a playbook

You can drag blocks around the canvas. Lines connected to boxes automatically arrange themselves when you move blocks around.

Hover over any playbook block and click the trash can icon to delete the block. The corresponding connecting arrow is also deleted.

Save a playbook so that Splunk SOAR (On-premises) can access it using the classic playbook editor

You must save a playbook before Splunk SOAR (On-premises) can use it. If the playbook has errors, an error message appears with additional information. You cannot save a playbook until you resolve all errors.

1. Click **Save** to begin saving a playbook.
2. Enter a brief description of the playbook.
3. (Optional) Click **Reuse this comment until the editor is reloaded** if you want to use the same description each time you save the playbook.
4. Click **Save** to finish saving the playbook.

Create a copy of an existing playbook

You can create a copy of any saved playbook by using the **Save As** option.

Perform the following tasks to create a copy of an existing playbook:

1. Click the down arrow next to the **Save** button and click **Save As**.
2. Enter a new name for the playbook.
3. Select the repository where you want to save the playbook.
4. (Optional) Enter comments or a description about the playbook.
5. Click **Save**.

Use keyboard shortcuts in the classic playbook editor

When using the classic playbook editor, press the **?** key to see a list of keyboard shortcuts. The following keyboard shortcuts can be used in the classic playbook editor.

Description	Shortcut
Save	Cmd+S
Save and comment	Cmd+Opt+S
Save as	Cmd+Shift+S
Run debugger	Cmd+D
Edit mode	Cmd+E

If you are using a Windows machine, replace Cmd with Ctrl in the keyboard shortcuts.

Manage playbooks and playbook settings

Create custom lists for use in Splunk SOAR (On-premises) playbook comparisons

A custom list is a collection of values that you can use in a playbook, such as a list of banned countries, or blocked or allowed IP addresses. Custom lists are used to save information in a visual format that can be used to make decisions or track information about playbooks. In your **Filter** and **Decision** blocks, compare parameters against all the values in a custom list, rather than having to configure each comparison in the playbook.

See Example of using a custom list in a filter for an example of how to use a custom list in a playbook.

Create a custom list in Splunk SOAR (On-premises)

Custom lists have a size limit of 2GB.

Perform the following steps to create a custom list in Splunk SOAR (On-premises):

1. From the **Home** menu, select **Playbooks**.
2. Select the **Custom Lists** tab.
3. Click **+ List** to create a new list.
4. Enter a name for the list.
5. Enter or paste the list values in the table using one value per cell. For example, you can create a list of banned countries, or blocked or allowed IP addresses. Right-click in a cell to add or remove rows and columns.
6. Click **Save**.

Import a custom list to Splunk SOAR (On-premises) using a CSV file

Imported custom list files have a size limit of 1MB.

Perform the following tasks to import a CSV file to be used as a custom list.

1. From the **Home** menu, select **Playbooks**.
2. Select the **Custom Lists** tab.
3. Click the **Import Custom List CSV** icon () to import a custom list as a CSV file.
4. Enter a name for the list.
5. Drag and drop your CSV file to the window, or click the window to locate the CSV file on your file system.
6. Click **Upload**.

View the list of configured playbooks in Splunk SOAR (On-premises)

The Playbooks table contains all currently available Splunk SOAR (On-premises) playbooks and significant metadata about those playbooks. Use the playbooks list to sort, filter, and manage your playbooks.

To view the Playbooks table, click the main menu in Splunk SOAR (On-premises), then click **Playbooks**.

From the Playbooks table page, you can perform any of the following tasks:

- Use the search field to find specific playbooks. Searches are case-insensitive and partial-word matches are supported. This search does not support booleans, such as AND, NOT, or OR.
- Click on column headers to sort the playbooks. The first click sorts in ascending order. The second will sort in descending order. For example, click the **Updated** column header. The icon changes to indicate whether the column is sorted in descending or ascending order. so that the playbooks with the most recent changes are listed at the top.
- By default, 10 playbooks are displayed at a time. Click the **Show 10** dropdown and select the number of playbooks you want displayed at a time.
- Click the **Reorder Active Playbooks** icon ( ) to change the order in which active playbooks are listed. See Reorder active playbooks in Splunk SOAR (On-premises).
- Click the **Import Playbook** icon ( ) to add a playbook to Splunk SOAR (On-premises). See Import a playbook to Splunk SOAR (On-premises).
- Click **+ Playbook** button to create a new playbook. The Playbook Editor will open in a new tab or window.

The columns in the Playbook table are described in the following table. By default, all available columns are visible. Scroll to the right as needed to view all available columns. Click the vertical ellipsis icon to select which columns to display.

Column	Description
Name	The name and description of the playbook.
Success	The number of times the playbook has run successfully.
Failed	The number of times the playbook did not finish running.
Label	The event label that the playbooks runs on. This value is configured as the **Operates on** field in the playbook settings. See Review playbook settings for a playbook in Splunk SOAR (On-premises).
Repo	The repository or folder where the playbook is saved.
Category	The playbook category. This value is configured in the **Category** field in the playbook settings. See Review playbook settings for a playbook in Splunk SOAR (On-premises).
Status	Indicates whether or not the playbook is active in Splunk Mission Control: • **Inactive** means the playbook is not active. • **Active** means the playbook is active. Only active playbooks can be run.
Mode	Either **Classic** or **Modern**. A Classic playbook uses the older version of the two playbook editors available in Splunk SOAR (On-premises), and a Modern playbook uses a newer version of the playbook editor in Splunk SOAR (On-premises). For details about the differences between them, see Choose between playbooks and classic playbooks in Splunk SOAR (On-premises).
Type	The type of user who created the playbook, either Automation or Input. Automation playbooks run automatically based on triggers. Input playbooks are used only as sub-playbooks, and are not automatically triggered as independent playbooks. The input type playbooks can't be run manually so they won't be visible if you use the **Run Playbook** button.
Python Version	The Python version used in the playbook.
Created	The date and time when the playbook was saved for the first time.
Updated	The date and time when the playbook was most recently saved.
Updated By	The name of the user who last updated the playbook.
Version	The playbook version number.
Tags	The tags associated with the playbook.

Column	Description
Apps Used	The apps that are used by this playbook.
Actions Used	The actions that are used by this playbook.
Sub-Playbooks Called	The sub-playbooks that are called by this playbook.

Export and import playbooks in Splunk SOAR (On-premises)

You can export, or download, and import, or upload, playbooks in Splunk SOAR (On-premises) to share them with other tenants.

Export a playbook to your local filesystem

Perform the following tasks to export a playbook from the playbooks table. The playbook is downloaded as a JSON file to your local filesystem.

1. From the **Home** menu, select **Playbooks**.
2. Select the checkbox for the playbook you want to export.
3. Click **Export**.

You can also export a playbook while you are editing the playbook in the playbook editor. In order to export a playbook, you must save it at least once. The export will download the latest saved version of the playbook.

Perform the following tasks to export a playbook from the classic playbook editor:

1. Make sure there is at least one saved version of the playbook.
2. In the playbook editor, click **Playbook Settings**.
3. Click **Export Playbook**.

Perform the following tasks to export a playbook from the modern playbook editor:

1. Make sure there is at least one saved version of the playbook.
2. In the playbook editor, click the more icon (⋮).
3. Click **Export**.

Import a playbook to Splunk SOAR (On-premises)

Perform the following tasks to import a playbook. The imported playbook is treated as a new playbook.

1. From the **Home** menu, select **Playbooks**.
2. Click the **Import Playbook** icon (⬆) to add a playbook to Splunk SOAR (On-premises).
3. Set the source control repository to which the playbook will be uploaded.
4. (Conditional) If you are uploading a playbook with the same name as an existing playbook, check the **Force Update** checkbox.
5. Drag and drop your .tgz compressed playbook file onto the **Select Playbook Archive** field, or click that field to browse to select the playbook archive from your file system.
6. Click **Upload**.

Missing configurations in imported playbooks

When you import a playbook from a different Splunk SOAR environment or from a different repository, such as the Splunk SOAR Community repository, one or more action block configurations for that playbook might not be found on your system. Playbooks will not run if they have one or more missing configurations.

When you open a playbook with missing configurations, a warning displays on the top of the screen.

To connect missing configurations for your imported playbook, follow these steps.

1. Click the link on the warning message to view the missing configurations.
2. For each of the missing configurations shown, use the list provided to select an appropriate asset configuration to link to that action block.
   - ♦ If multiple action blocks use the same missing asset configuration, when you create the link for the first action block in this step, the other mappings for the same asset configuration resolve automatically.
   - ♦ If you do not have an appropriate configuration installed, click the link provided to add a new configuration.
     When creating the new asset configuration, if you use the exact same name as the missing asset configuration, the mappings resolve automatically.
3. Click **Save**.

The warning continues to display until you have resolved all missing configurations.

Manage settings for a playbook in Splunk SOAR (On-premises)

After you've saved a playbook, you can manage the settings for a specific playbook.

1. In Splunk SOAR (On-premises), from the **Home** menu, select **Playbooks**.
2. Locate the playbook that you want to review settings for and click on the playbook name.
3. In the playbook editor, click **Playbook Settings**.

The following table describes the fields in the playbook settings. The fields you will see depend on whether you are configuring a classic or modern playbook, and whether your modern playbook is an input or automation playbook.

Field	Description	Available in classic playbook?	Available in modern automation playbook?	Available in modern input playbook?
Operates on	Select the event label or labels that this playbook runs on in the **Operates on** field. Most playbooks are designed to work with data in a particular category, and therefore a particular label for events. Every event in Splunk SOAR (On-premises) has a label associated with it, such as **Events** or **Email**. For more on labels in Splunk SOAR (On-premises), see Configure labels to apply to containers in *Administer Splunk SOAR (On-premises)*.	Yes	Yes	No
Category	Use categories to organize your playbooks. For example, you can create a **Production** category for playbooks that are ready to be marked active, and a **Test** category for playbooks that are under development.	Yes	Yes	Yes
Run as		Yes	Yes	No

Field	Description	Available in classic playbook?	Available in modern automation playbook?	Available in modern input playbook?
	The automation user Splunk SOAR (On-premises) used to run the playbook.			
Tags	Use tags to provide additional metadata to group playbooks together. You can create tags for playbook within the same category or across multiple categories.	Yes	Yes	Yes
Logging	Toggle this switch to turn on debug logging each time the playbook is run. This might be useful when create a new playbook. If the playbook has an error, you are able to see what the problem was using debug logging. Eventually, when the playbook works like you expect, turn logging off to save disk space.	Yes	Yes	Yes
Active	Toggle this switch to make the playbook run automatically on every new event or artifact that comes into Splunk SOAR (On-premises).	Yes	Yes	No
Safe Mode	Toggle this switch to put the playbook in read-only mode. Read and write permissions are defined by each connector in Splunk SOAR (On-premises). For example, in an LDAP connector, `get users` is a read-only action, while `reset password` is read-write. Specifying the Safe Mode setting applies only to the current playbook, and *not* to any child playbooks called by that playbook. Specify Safe Mode for child playbooks directly.	Yes	Yes	Yes
Draft Mode	Toggle this switch to save a draft of your playbook, even if your playbook is incomplete or has errors. Playbooks in draft mode can't be marked active.	Yes	Yes	Yes
Description	Enter a description for the playbook. The description becomes a triple-quoted comment in the playbook, and appears on the playbooks page.	Yes	Yes	Yes
Notes	Notes aren't visible anywhere else in Splunk SOAR (On-premises) and can only be viewed by editing the playbook.	Yes	Yes	Yes
Export Playbook	Export a playbook to download the current version of the playbook. This setting allows you to share playbooks with other users. You can import the file on the playbooks page.	Yes	No	No
Parent Playbooks	Expand the **Parent Playbooks** section to view or open the parent playbooks associated with this playbook. If there aren't any parent playbooks associated with this playbook, this section is not displayed.	No	Yes	Yes
View Keyboard Shortcuts	Click **View Keyboard Shortcuts** to see more information about keyboard shortcuts or to view the documentation.	No	Yes	Yes
Revision History	Click **Revision History** to view a playbook's revision history. • Click **View** to view a previous revision of the playbook. You can make edits, then save the edits as a new version.	Yes	Yes	Yes

Field	Description	Available in classic playbook?	Available in modern automation playbook?	Available in modern input playbook?
	• Click **Revert** to use the corresponding previous version of the playbook as the most current version.			
Audit Trail	Click **Audit Trail** to download a comma-separated value (CSV) file that shows the full audit trail of the playbook, including dates and times.	Yes	Yes	Yes
Docs	• Click the **Docs** link in the classic playbook editor to go to the documentation page for Splunk SOAR (On-premises). • Click the **View Documentation** link in the modern playbook editor to go to the documentation page for Splunk SOAR (On-premises).	Yes	Yes	Yes
Tenants	Select one or more tenants to run the playbook against the containers belonging to the selected tenants. Use an asterisk (*) to run the playbook on containers for all tenants. This field is only available in Splunk SOAR on-premises deployments.			

YesYesYes

Edit playbook properties in Splunk SOAR (On-premises)

You can edit the properties of one or more playbooks, such as its active state, logging mode, the labels operated on, categories, and tags.

To edit the properties of one or more playbook, perform the following steps:

1. From the **Home** menu in Splunk SOAR (On-premises), select **Playbooks**.
2. Select the playbooks for which you want to edit properties, then click **Edit**.
3. Make the desired changes,
4. Click **Save**.

For example, if you want to make a group of playbooks inactive, perform the following steps:

1. From the **Home** menu in Splunk SOAR (On-premises), select **Playbooks**.
2. Select the playbooks you want to make inactive, then click **Edit**.
3. In the **Active** field, select **Inactive**.
4. Click **Save**

Reorder active playbooks in Splunk SOAR (On-premises)

To change the order in which active playbooks are run in Splunk SOAR (On-premises), do the following:

1. From the **Home** menu in Splunk SOAR (On-premises), click **Playbooks**.
2. Click the **Reorder Active Playbooks** icon (🔄).
3. In the **Order** column, select the desired order of the playbooks, or drag and drop the playbooks to your desired

order.
4. Click **Done**.

Debug playbooks in Splunk SOAR (On-premises)

If you're having problems with your playbook and need to troubleshoot issues, run your playbook using the debugger.

To run your playbook using the debugger, the playbook must meet the following conditions:

- The playbook must be saved. Playbooks in edit mode can't be debugged.
- The playbook cannot be marked active.
- The playbook must have an event to run against. If there are dependencies on any artifacts as part of the event, the artifacts must also be present and must not have been previously used by this same version of the playbook.

You can access the playbook debugger using one of the following methods:

- Click the **Playbook Debugger** tab in the playbook editor.
- Use the **Cmd+D** or **Ctrl+D** keyboard shortcut. See Use keyboard shortcuts in the classic playbook editor.

Set **Scope** to define which artifacts are processed in the playbook run. You can set the scope to **New Artifacts** to process only artifacts defined since the playbook was last run, or **All Artifacts** to include all artifacts in the playbook run.

Each line in the debug content starts with a date time stamp. Log entries show which action is running. The parameter sent, such as inputs from earlier blocks or playbooks and message it received, and the outputs of each block are logged. The API call to `on_finish` represents a call to the End block. The playbook completes by logging a SUCCESS or FAILURE status.

Debug input playbooks

You can test the inputs of an input playbook using the debugger.

To test and debug an input playbook:

1. Open the playbook in the Visual Playbook Editor.
2. Open the debugger from the tab in the lower right corner of the Visual Playbook Editor.
3. In the top left corner of the debugger, click the adjustment bars icon.
4. Add values for the playbook's inputs.
5. Add the event id to test against.
6. Click **Test**.

The output of the debugger shows the execution of the playbook so you can see each of the blocks and the test inputs.

View or edit the Python code in Splunk SOAR (On-premises) playbooks

Click the **Python Playbook Editor** tab to view the underlying Python code for your playbook. The code for the entire playbook is shown by default. Click any block in your playbook to view the code for the selected block only.

See the following documentation for more information about Python code in your playbooks:

- Python Playbook Tutorial for Splunk SOAR (On-premises) overview in the *Python Playbook Tutorial for Splunk SOAR (On-premises)* manual.
- About Splunk SOAR (On-premises) playbook automation APIs in the *Python Playbook API Reference for Splunk SOAR (On-premises)* manual.

Manage your editing session

Use the icons in the Python Playbook Editor to manage your editing session.

Icon	Description
</>	View the Python code for the entire playbook. Using this icon is useful if you are viewing the Python code for a specific block on the canvas, and want to return to view the Python code for the entire playbook.
🌐	Add code that needs to be defined at the global level of the playbook, such as import statements for Python libraries.
⊡	View functions for blocks that have diverging or converging actions. The functions are explained in the following list: • **Block Function** is highlighted when viewing the Python code that is applicable to a single block • **Callback Function** is used to view to the block of code that is generated to split the output of the single block into multiple blocks. • **Join Function** is used to view the block of code that is generated to join the output of the multiple blocks into a single block.
⟲	Go back to the original version and discard all changes. If there are changes to revert, the button turns white when you hover over it.

How custom Python edits affect the visual playbook editor

When you see **Playbook Code** in the Python Playbook Editor, you are making changes affecting the whole playbook. When you begin to make edits, you are prompted to verify that you want to continue. If you continue, you will no longer be able to edit the playbook using the playbook editor. All changes to the playbook must be made by editing or adding Python code.

If you click a block in the playbook, your edits only disable the playbook editor for that block. The Python Playbook Editor changes from **Playbook Code** to the name of the Python function called in that block. You can continue to use the playbook editor to add, edit, or delete other blocks in the playbook. If you want to add another block downstream from the block you edited, you have to manually enter a Python function call for the next block, such as `phantom.act()`. The playbook editor doesn't generate Python code for any block containing custom edits.

When editing the Python code for a **Code** block, make your edits in the editable area in order for callback functions to work.

1. Create a **Code** block in the playbook editor. See Add custom code to your Splunk SOAR (On-premises) playbook with the code block.
2. Click **Python Playbook Editor**.
3. Click the **Code** block.
4. Write your custom code in the area with the `# Write your custom code here...` text.
   ```
   ################################################################################
   ## Custom Code Start
   ################################################################################

   # Write your custom code here...

   ################################################################################
   ## Custom Code End
   ################################################################################
   ```