



# Incident Response Platform Integrations

# Carbon Black Protection Integration V1.0.2

Release Date: April 2019

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

This guide describes the Carbon Black Protection Integration.

Overview

This integration consists of 12 functions which call various APIs to perform different actions such as retrieving approval request details, updating approval requests, deleting files, etc. It also contains a polling component to create incidents in the Resilient platform that correspond to approval requests in Carbon Black Protection. The rest of this guild will go more in depth into the capabilities and example workflows they are used in. Please note, to access all features of this integration, Carbon Black Protection v8.1 is required.

Installation

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

* Resilient platform is version 30 or later.
* You have a Resilient account to use for the integrations. This can be any account that has the permission to view and modify administrator and customization settings, and read and update incidents. You need to know the account username and password.
* You have access to a Resilient integration server. An *integration server* is the system that you use to deploy integration packages to the Resilient platform. See the [Resilient Integration Server Guide (PDF)](https://github.com/ibmresilient/resilient-reference/blob/master/developer_guides/Integration%20Server%20Guide.pdf) for more information.

Install the Python components

The functions package contains Python components that are called by the Resilient platform to execute the functions during your workflows. These components run in the Resilient Circuits integration framework.

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

Complete the following steps to install the Python components:

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

sudo pip install --upgrade pip

sudo pip install --upgrade setuptools

sudo pip install --upgrade resilient-circuits

1. To install the package, you must first unzip it then install the package as follows:

sudo pip install --upgrade fn\_cb\_protection-1.0.2.tar.gz

Configure the Python components

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

Complete the following steps to configure and run the integration:

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

sudo su - integration

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

resilient-circuits config -c

or

resilient-circuits config -u

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

[fn\_cb\_protection]  
# Name or IP address of your CbProtect server

server=10.200.1.1

# Access token issued by the CbProtect administrator

token= XXXX-XXXX-XXXX-XXXX

# If your CbProtect server has a self-signed TLS certificate, you cannot verify it:

# verify\_cert=false

# Interval (seconds) for automatic escalation of approval requests, set 0 to disable

# Suggest 300 as a starting point, which will check CbProtect every 5 minutes

escalation\_interval=0

# Optional: query for which requests to escalate; default is to escalate all open approval requests

# escalation\_query=resolution:0

# Optional: path to a custom template file for the escalated incident

# template\_tile=/usr/integration/bit9\_escalation.jinja

# Optional: set this to only escalate a single request ID, e.g. when testing a custom template

# test\_single\_request=999

Add Passwords to your keystore (optional)

If the function contains passwords or other authentication values, the Resilient package includes a utility to add all of the keystore-based values from your app.config file to your system's compatible keystore system. Once you have created the keys in your app.config file, run res-keyring and you are prompted to create the secure values to store.

res-keyring

Configuration file: /Users/kexample/.resilient/app.config

Secrets are stored with 'keyring.backends.OS\_X'

[resilient] password: <not set>

Enter new value (or <ENTER> to leave unchanged):

Deploy customizations to the Resilient platform

The package contains function definitions that you can use in workflows, and includes example workflows and rules that show how to use these functions.

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

resilient-circuits customize

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

Run the integration framework

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

resilient-circuits run

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

Configure Resilient Circuits for restart

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

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

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

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

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

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

[Install]  
WantedBy=multi-user.target

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

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

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

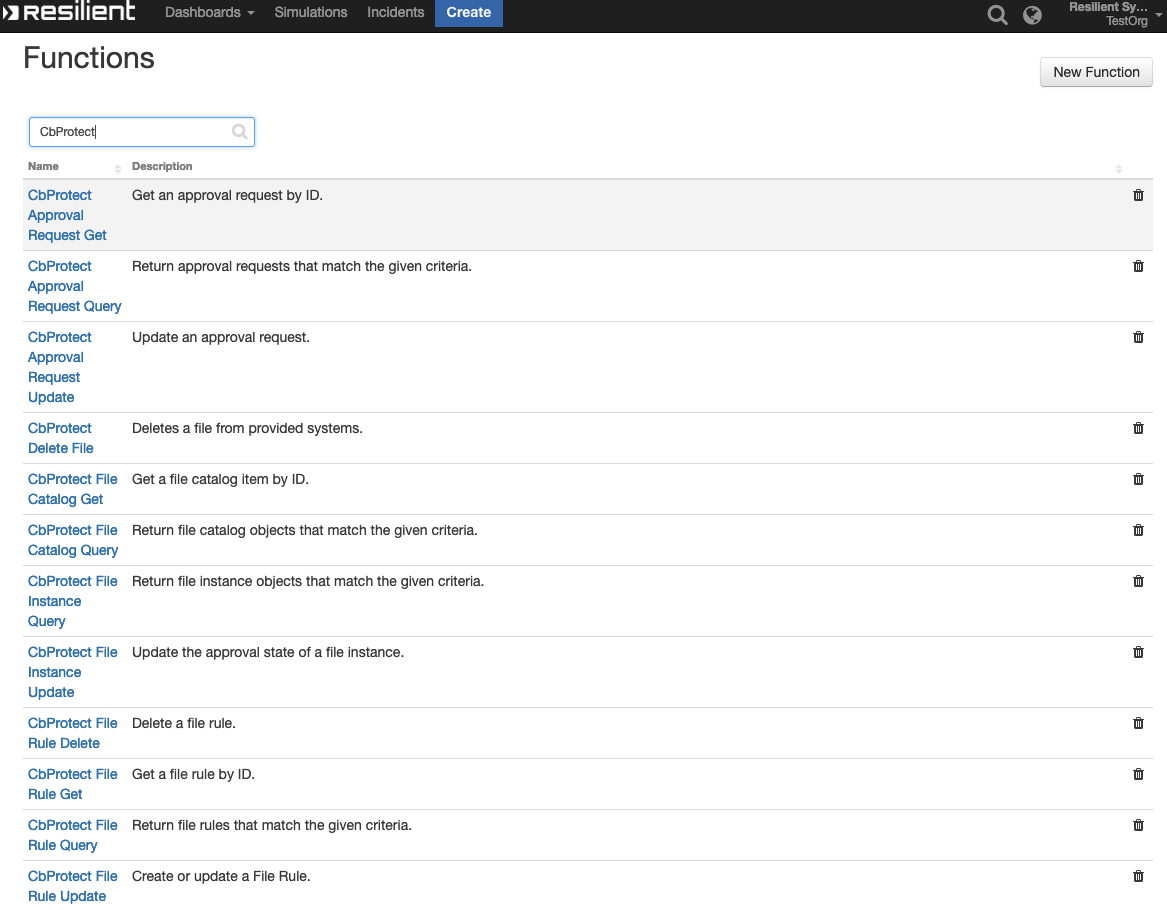
sudo systemctl resilient\_circuits [start|stop|restart|status]

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

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

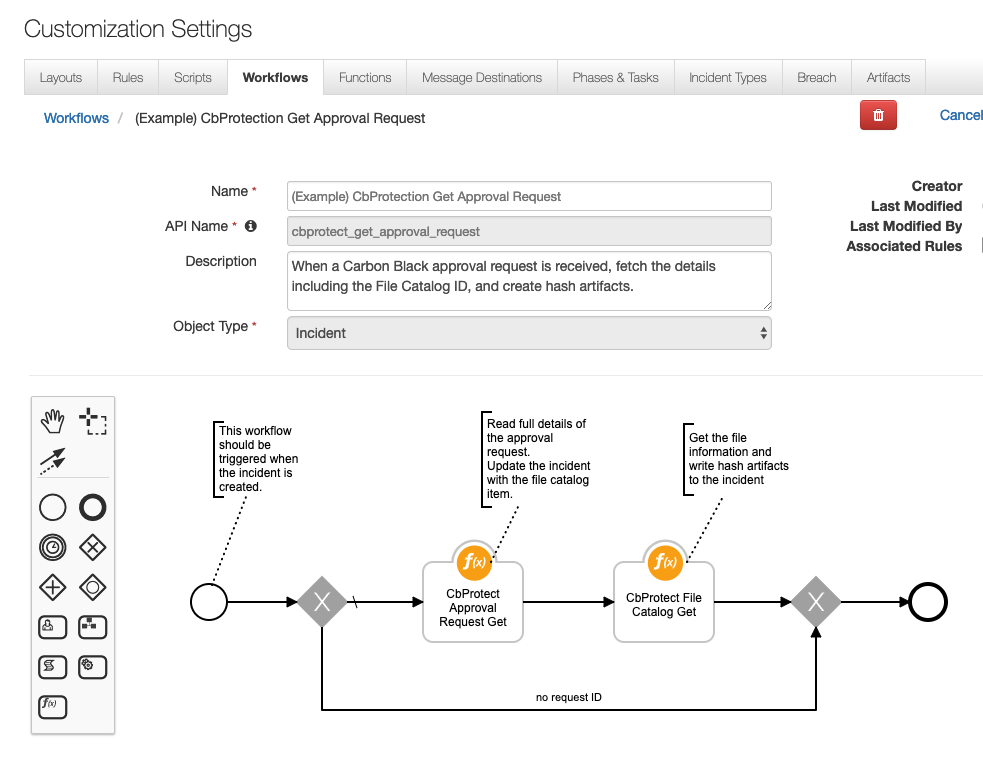
Function Descriptions

Once the function package deploys the function(s), you can view them in the Resilient platform Functions tab, as shown below. The package also includes example workflows and rules that show how the functions can be used. You can copy these workflows and rules for your own needs.



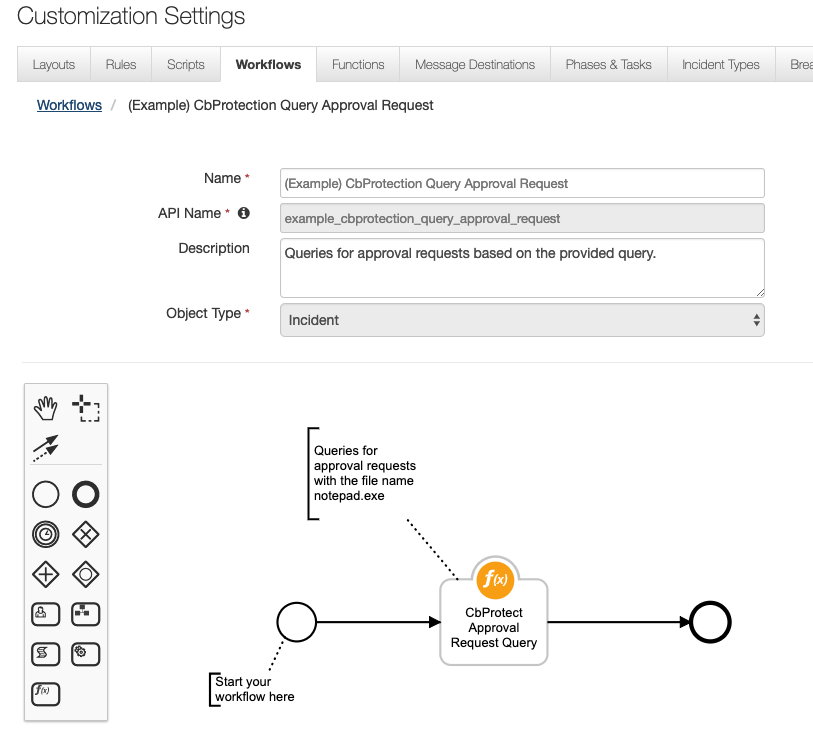
bit9\_approval\_request\_get: Cbprotect Approval Request Get

Given an approval request’s ID, the function returns the details of the approval request. The function takes one input bit9\_approval-request\_id which is a number. An example of this function used in a workflow can be seen below in workflow (Example) CbProtection Get Approval request.

**

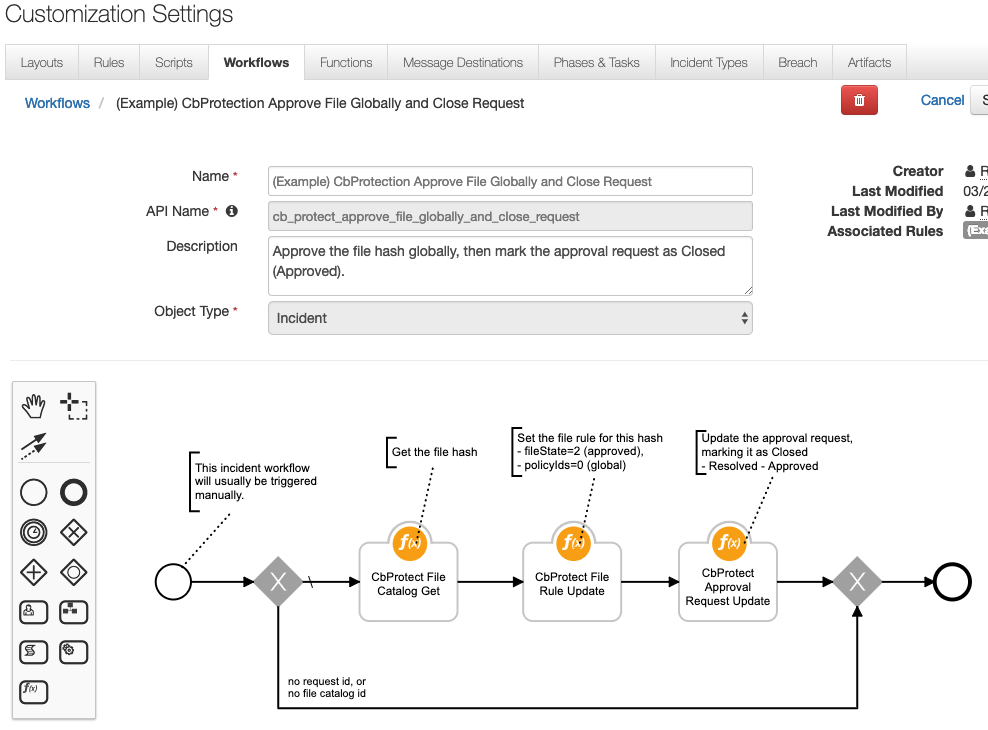
bit9\_approval\_request\_query: Cbprotect Approval Request Query

This function takes one input, bit9\_query, which is just a query sting and returns the approval requests which match the given query. An example of this can be found in the workflow (Example) CbProtection Query Approval Request below:

**

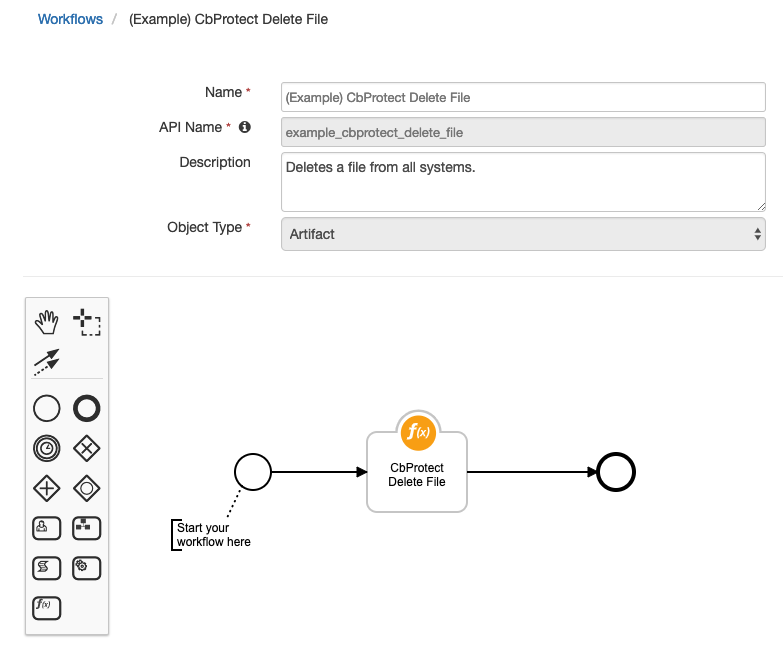
bit9\_approval\_request\_update: Cbprotect Approval Request Update

This function accepts inputs of a request ID, approval request resolution, comments, and status. With these it updates an approval request. An example of this can be found in the workflow below:

**

bit9\_file\_delete: Cbprotect Delete File

This function deletes a file from a specific computer or all computers using Carbon Black Protection. The inputs file action (which represents deleting by file hash or deleting by name) must be set in addition to computer ID (setting to 0 deletes from all computers), and depending on the action choosen either catalog ID, file hash, or file name. An example of this can be found in the workflow (Example) CbProtect Delete File below:

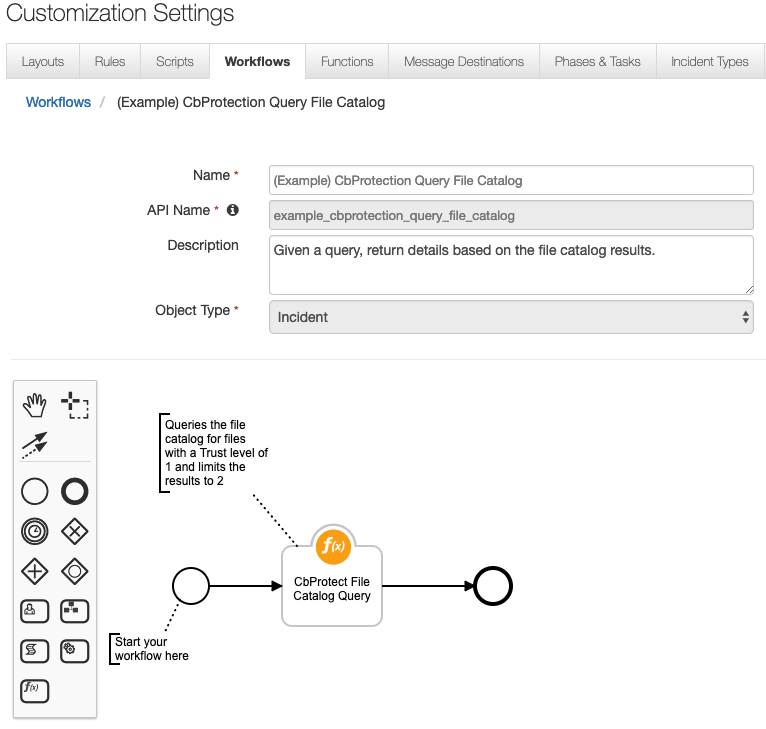
**

bit9\_file\_catalog\_get: Cbprotect File Catalog Get

Returns back file catalog details based on the catalog ID provided. An example of this can be found in the workflow shown earlier (Example) CbProtection Query Approval Request:

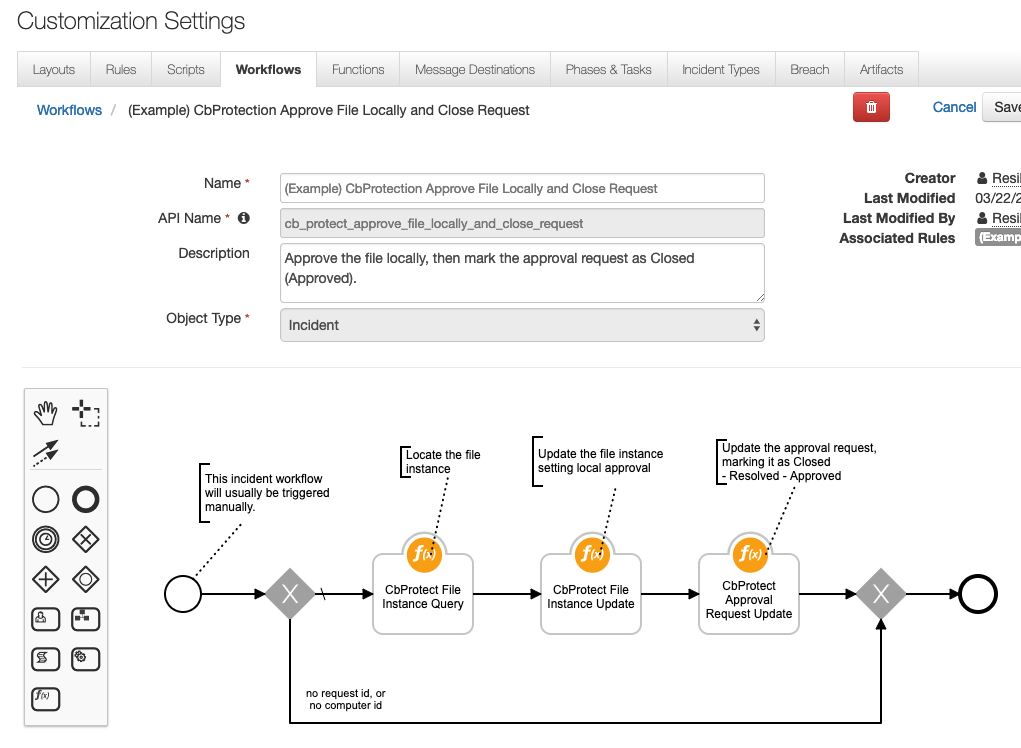
bit9\_file\_catalog\_query: Cbprotect File Catalog Query

Returns back file catalogs and their details from a provided query string. An example of this can be found in the workflow below (Example) CbProtection Query File Catalog:



bit9\_file\_instance\_query: Cbprotect File Instance Query

Returns file file instance objects that match the given criteria from the inputs. An example of this can be found in the workflow below (Example) CbProtection Approve File Locally and Close Request:

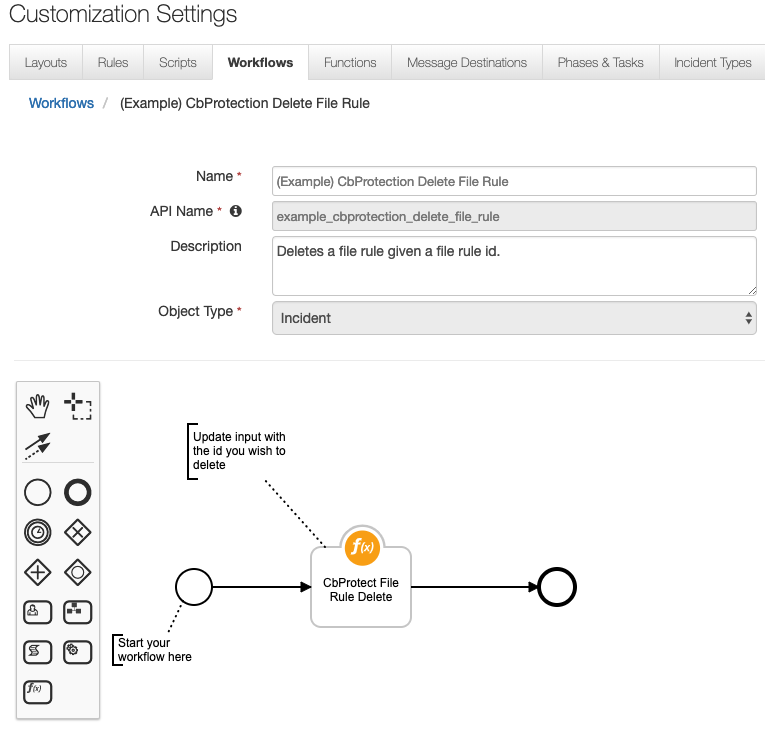


bit9\_file\_instance\_update: Cbprotect File Instance Update

Updates a file instance’s local approval/banned setting. This function has inputs for the file instance ID and the local state (ie: approved = 2). An example of function can be seen in the (Example) CbProtection Approve File Locally and Close Request workflow above:

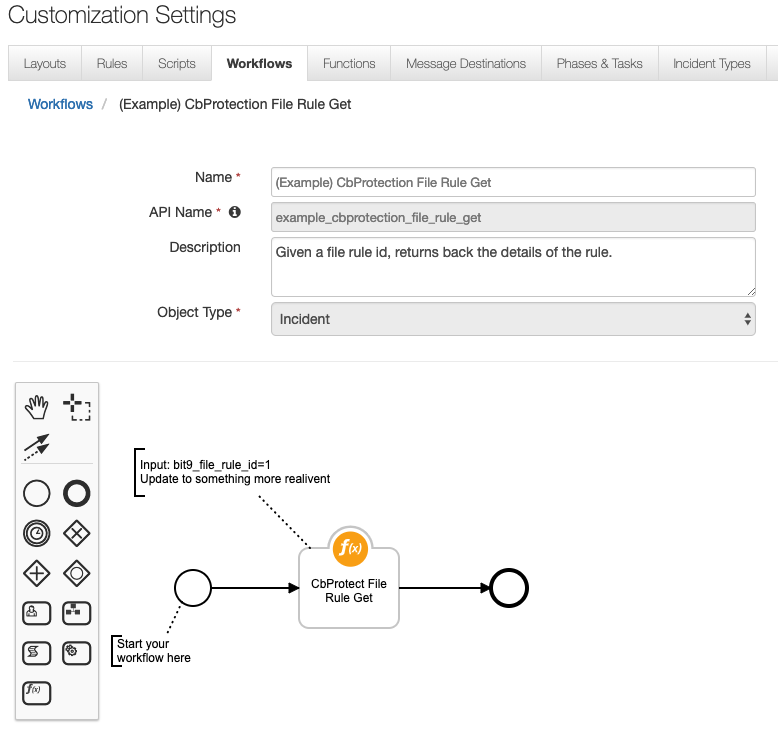
bit9\_file\_rule\_delete: Cbprotect File Rule Delete

Given a file rule ID, deletes the file rule from Carbon Black. An example of this function can be seen in the workflow (Example) CbProtection Delete File Rule below:

**

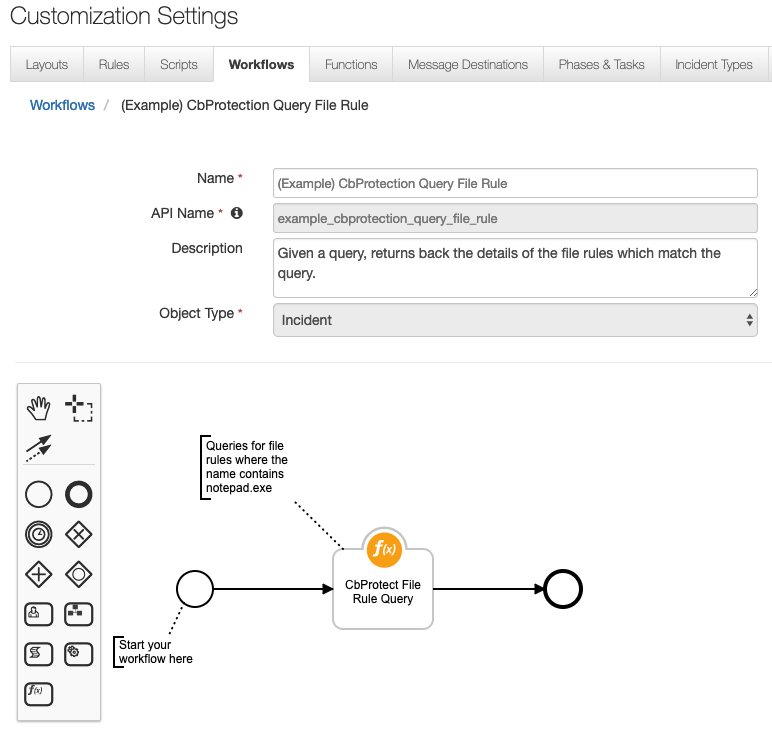
bit9\_file\_rule\_get: Cbprotect File Rule Get

Given a file rule ID, returns back the details of the file rule. An example of this function can be seen in the workflow (Example) CbProtection File Rule Get below:

**

bit9\_file\_rule\_query: Cbprotect File Rule Query

Given a query string, returns back the details of file rules which match the query. An example of this function can be seen in the workflow (Example) CbProtection Query File Rule:

**

bit9\_file\_rule\_update: Cbprotect File Rule Update

This function updates a file rule in Carbon Black based on the data passed as inputs. An example of this function can be seen in the workflow (Example) CbProtection Approve File Globally and Close Request shown earlier:

Carbon Black Protection Resilient Polling Component

This integration also contains a polling component to automatically escalate approval requests into the Resilient platform. To enable this feature the escalation\_interval variable in the app.config file must be set to an integer greater than 0. This integer represents the interval in number of seconds for the automatic escalation of approval requests. It is recommended to start at 300 which will check every 5 mins. Optional values can also be set such as escalation\_query, if set will escalate approval requests which match the query otherwise defaults to all open approval requests. In addition, template\_file can be set to the location of a custom jinja template file otherwise the default template file will be used. To create your own custom jinja file it is recommended to use the default one as a reference which can be found when expanding the package in the fn\_cb\_protection-<version#>/fn\_cb\_protection/data/ directory.

Resilient Platform Configuration

Please note, while some functionality of this integration should work on older versions of Carbon Black Protection, full support of this integration will only work on Carbon Black Protection v8.1 and up.

Troubleshooting

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

* Resilient Action Status

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

* Resilient Scripting Log

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

* Resilient Logs

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

* Resilient-Circuits

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

Support

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

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