IBM Resilient

Security Orchestration, Automation and Response Platform

INTEGRATION SERVER GUIDE v32

Resilient Security Orchestration, Automation and Response Platform Integration Server Guide

Platform Version	Publication	Notes
32.3	May 2019	Corrected Resilient Circuits configuration procedure. Added more useful procedures.
32.3	May 2019	Added procedures, including keyring. Corrected systemctl command syntax error.
32.0	January 2019	Initial publication.

Table of Contents

1. Introduction

This guide provides the procedures to create and maintain a Resilient *integration server*, which is the system that you use to deploy extensions to the Resilient Security, Orchestration and Response (SOAR) Platform. This guide also provides the deployment procedures.

You can also use the Resilient integration server to write your own functions and custom actions, or implement custom threat services as described in the *Function Developer's Guide*, *Custom Action Developer's Guide* and Custom Threat Service Guide (PDF), respectively.

NOTE: As of V32.3, the name of the Resilient Incident Response Platform changed to Resilient Security, Orchestration and Response (SOAR) Platform. This guide supports all versions of the Resilient platform from V30.0 to V32.*x*.

1.1. Extensions and integrations

A Resilient *extension* is a software package that extends the functionality of the Resilient platform. It can contain one or more Resilient components and external code components. An external code component can access and return external data, interact or integrate with other security systems, or be a utility that performs a specific action.

There are various types of extensions, including functions and custom actions. The difference between custom actions and functions is where they send their results. A custom action can populate a custom field or data table within the Resilient platform. A function returns the results to the workflow that invoked the function, which allows the workflow to act upon the results.

An extension is also called an integration when its purpose is to communicate with another system.

IBM Resilient has a number of extensions in the form of functions and custom actions that you can download, customize and deploy.

1.2. Supported extensions

The following types of extensions require a Resilient integration server:

- Python-based, such as Resilient functions and some custom actions. The Resilient platform can be in an on-premises or cloud configuration.
- Java-based custom actions. The Resilient platform can be in an on-premises or cloud configuration.
- Custom threat services. The Resilient platform must be on-premises.

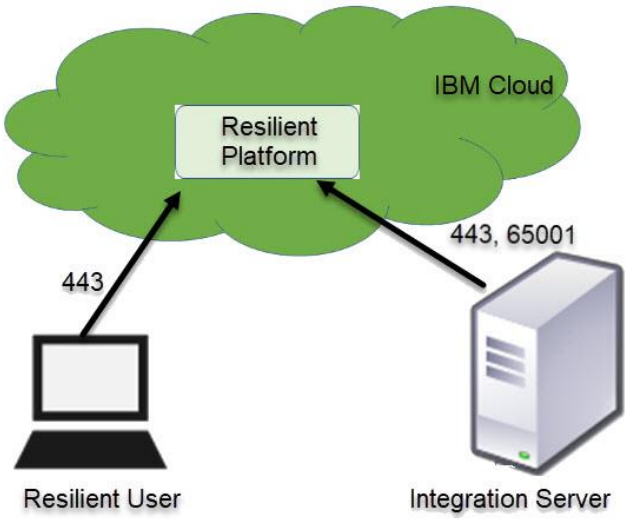
There are also extensions in the form of plug-ins, such as for Splunk and QRadar, which escalate incidents to the Resilient platform using the REST APIs. You do not need a Resilient integration server for these plug-ins.

If you have a package in a .res format, this means it contain components, such as scripts, workflows, and custom fields. You import these components into your Resilient platform using the import feature described in the S*ystem Administrator Guide*.

2. Architecture

The Resilient integration server must be able to access the Resilient platform and the hosts of the third-party applications that are integrated with the Resilient platform. The Resilient platform can reside in your own environment (known as an on-premises configuration) or within the IBM cloud (also known as a SaaS configuration).

The following network diagram is an example of a Resilient integration server used for python-based extensions and a Resilient platform in a cloud configuration. Resilient users access the platform from their web browsers.



If you have a Resilient platform in your environment, the Resilient integration server can be the same system as the one hosting the Resilient platform; however, IBM Resilient recommends that you keep them on separate systems for the following reasons:

- Security. Although all apps from the Community App Exchange are code scanned and tested, it is a security measure to not allow apps access to the command line of the Resilient platform host.
- Performance. Although the Resilient extensions are resource efficient, running them on the Resilient host could have impact on the available resources, especially if your extensions are heavily using in-memory processing.
- Access. The administrator of the Resilient integration server maintains, installs, and configures extensions. This administrator does not necessarily require access to the Resilient platform.

3. Prerequisites

The following sections describe the system and network requirements. Specific extensions may have additional requirements, as described in the extension's documentation.

3.1. Resilient integration server

The following lists the requirements for the system hosting the integration server.

- Operating system:
  - Red Hat Enterprise or Centos Linux 7.4.
  - Windows (Windows Enterprise Server 2016 or later is recommended)
- For Python-based integrations, Python 2.7.*x* (where *x* is 9 or later), or 3.6.*x* (where *x* is 4 or later). Make sure the Python installation includes pip. On a Linux system, you can verify the version of Python using the following command:

  ```
  python --version
  ```

- Minimum of 5GB free disk space.
- Minimum of 8 GB RAM.
- Dedicated operating system account for running integrations. The Resilient Circuits components run as an unprivileged user, typically named **integration**.
- Text editor, such as nano, to edit the configuration file.
- Access to the Resilient platform and various ports as described in Network configuration.

The resources required by the Resilient integration server are variable due to the requirements of the extensions installed. Some extensions that operate on files in memory may have additional memory requirements. Extensions that perform decryption tasks may need more CPU. Therefore, you may need to increase those resources.

3.2. Resilient platform

The following lists the prerequisites for the Resilient platform:

- Version 30 or later. Some extensions may require a later version.
- Licensed for Resilient Action Module. If using custom threat services, the platform must also be licensed for the Resilient Security Module. You can verify this by running the following command on the Resilient platform:

  ```
  resutil license
  ```

  Dedicated Resilient user account to use as the API user. With most integrations, the account must have the permission to view and edit incidents, and view and modify administrator and customization settings. You obtain the account credentials from the Resilient system administrator.

IBM Resilient recommends that you use a Resilient platform in a test environment to create the function, message destination, rules, workflows and other components needed for your extension. This can be a separate Resilient platform or a Resilient organization within your platform dedicated for testing. Once tested, you can deploy the extension into any Resilient platform that is at the same or later version as your test platform.

3.3. Network configuration

If the Resilient platform is beyond a firewall, such as in a cloud configuration, configure the firewall to allow the Resilient integration server access to the following ports:

- 443. Required for the extension to connect to Resilient data using the REST API.
- 65000. Only if supporting Java-based extensions. The Resilient integration server communicates with port 65000 of the platform using ActiveMQ OpenWire.
- 65001. Only if supporting Python-based extensions. All Resilient functions are Python-based. The Resilient integration server communicates with port 65001 of the platform using the STOMP messaging protocol.
- 9000 (default). Only if supporting custom threat feeds, which is not applicable for a Resilient platform in a cloud configuration. The platform must be in your environment. The Resilient platform communicates with the Resilient integration server over HTTPS on a port that you choose (by default on port 9000).

Port 9000 is an “inbound-only” connection to the Resilient integration server, where the other connections are “outbound-only” from the integration server to the Resilient platform.

4. Installation

To create a Resilient integration server, install the Resilient Circuits framework which includes its dependent modules.

4.1. Downloading Resilient Circuits

If your Resilient integration server is not connected to the Internet, you need to download the Resilient Circuits package from IBM Resilient GitHub.

The following is an example transfer procedure for a Linux system.

```
mkdir ~/package_name_offline
pip download package_name -d "~/some_package_offline"
tar -cvfz package_name_offline.tar package_name_offline
```

Transfer the package to the Resilient integration server then perform the following:

```
tar -xzvf package_name.tar
cd some_directory
pip install package_name-x.x.x-py2.py3-x-x.whl -f ./ --no-index
```

4.2. Installing on a Linux system

Install and configure the Resilient Circuits framework as follows:

1. Use ssh to access the command line interface on the Resilient integration server.
2. If you do not have an OS user for the service, create one now. To create an OS user called integration on RHEL Linux, use the following command:

   ```
   sudo adduser integration --home /home/integration
   ```
3. Install Resilient Circuits using the following command. This command also installs its dependent modules.

   ```
   pip install --upgrade resilient-circuits
   ```
4. Verify that the Python modules, resilient and resilient-circuits, are installed.

   ```
   pip list
   ```
5. Auto-generate the app.config file as follows. This creates a directory, **.resilient**, in your home directory with a file in it called app.config, which is the default and preferred option. The Resilient Circuits configurations are maintained in the app.config file.

   ```
   resilient-circuits config -c
   ```

   The output of the command shows the directory where it installed the config file. By default, this directory is:

   ```
   /home/integration/.resilient/app.config
   ```

   If you require the configuration file to be in a different location or have a different name, you need to store the full path to the environment variable, APP_CONFIG_FILE.

   ```
   resilient-circuits config -c /path/to/<filename>.config
   ```
6. Open the app.config file in your text editor. If using nano, the command would be:

   ```
   nano /home/integration/.resilient/app.config
   ```

7. Replace the contents with the following settings. Your actual path names may be different.

```
[resilient]
host=localhost
port=443
email=resilient_account@example.com
password=ResilientPassword
org=Dev
# componentsdir=/home/resadmin/.resilient/components
logdir=/home/resadmin/.resilient
logfile=app.log
loglevel=INFO
```

   Enter the actual **email** and **password** of the Resilient user account for the integration server,

   Use the actual Resilient organization name for the **org** name. It is not needed if the Resilient account does not belong to more than one Resilient organization.

   See Updating the Configuration File for a detailed description of all the app.config settings, especially **cafile** if your Resilient platform does not have a valid certificate.

8. Save the file.

9. Test your installation by running the following command:

```
resilient-circuits run
```

   Resilient Circuits starts, loads its components, and continues to run until interrupted. If it stops immediately with an error message, check your configuration values and retry.

You are ready to download and deploy extension packages.

4.3. Configuring 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 RHEL system, this is done using a systemd unit files to define services. The configuration file defines the following properties:

- OS user account to use.
- Directory from where it should run.
- Any required environment variables.
- Command to run the integrations, such as resilient-circuits run.
- Dependencies.

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
```

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

```
[Unit]
Description=Resilient-Circuits 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
```

   NOTE: If you are installing Resilient Circuits on the same system as the Resilient platform (not recommended), you need to add the following lines in the [Unit] section after Description:

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

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

```
sudo chmod 664 /etc/systemd/system/resilient_circuits.service
```

4. Reload and enable the new service:

```
sudo systemctl daemon-reload
sudo systemctl enable resilient_circuits.service
```

You can use the systemctl command to manually start, stop, restart and return status on the service:

```
sudo systemctl [start|stop|restart|status] resilient_circuits
```

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"
```

4.4. Installing on a Windows system

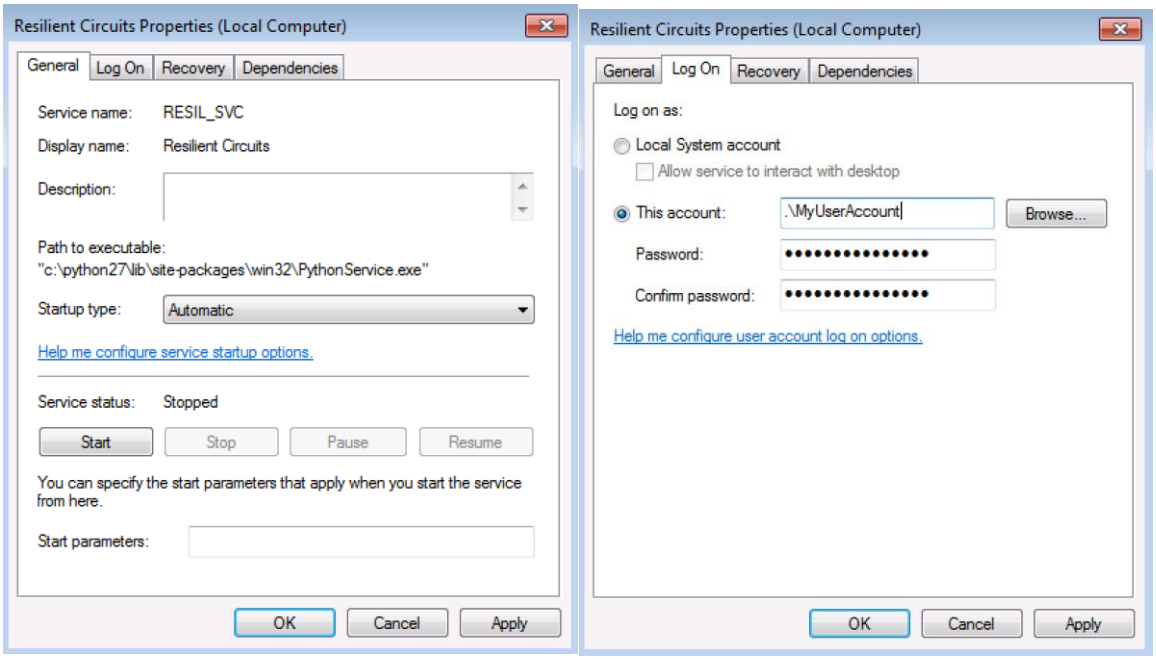
Resilient Circuits can be configured to run as a service on Windows. It requires the pywin32 library, which should be downloaded from sourceforge. Instructions for downloading and installing the correct package are at the bottom of the sourceforge web page and must be followed carefully.

Do not use the pypi/pip version of pywin32. Installation of the wrong version of the pywin32 library will likely result in a Resilient service that installs successfully but is unable to start.

Enter the following command:

```
resilient-circuits.exe service install
```

Once installed, you can update the service to start up automatically and run as a user account.



It is recommended that you log in as whichever user account the service will run as to generate the config file and confirm that the integration runs successfully with “resilient-circuits.exe run” before starting the service.

Commands to start, stop, and restart the service are provided as well.

```
resilient-circuits.exe service start
resilient-circuits.exe service stop
resilient-circuits.exe service restart
```

4.5. Updating your environment

When Resilient Circuits is updated, you can make sure you have the latest tools and Resilient Circuits framework by entering the following commands:

```
sudo pip install --upgrade pip
sudo pip install --upgrade setuptools
sudo pip install --upgrade resilient-circuits
```

5. Configuration

Once installed, you need to edit the configuration file, which defines essential configuration settings for all Resilient Circuits components running on the system. There is a [resilient] section in the configuration file that controls how the core Resilient Circuits and Resilient packages access the Resilient platform.

As you install extensions on the Resilient integration server, each extension creates its own section within the configuration file, as described later in this guide.

NOTE: If the APP_CONFIG_FILE environment variable is not set, Resilient Circuits looks for a file called "app.config" in the local directory where the run command is launched.

5.1. Editing the configuration file

Open the configuration file in the text-editor of your choice and update the [resilient] section with your Resilient platform host name/IP, credentials and the absolute path to the logs directory you created. The following table describes all the required and optional values that can be included in this section.

NOTE: If on a Windows system and you edit the file with Notepad, please ensure that you save it as type **All Files** to avoid a new extension being added to the filename, and use UTF-8 encoding.

Parameter	Description
logfile	Name of rotating logfile that is written to logdir. Default is app.log.
logdir	Path to directory to write log files. If not specified, program checks environment variable DEFAULT_LOG_DIR for path. If that is not set, then defaults to a directory called "log" located wherever Resilient Circuits is launched.
log_level	Level of log messages written to stdout and the logfile. Levels are: CRITICAL, ERROR, WARN, INFO (default), and DEBUG.
host	Required. IP or hostname for the Resilient platform.
org	Name of the Resilient organization. Required only if the user account is used with more than one Resilient organization.
email	Required. Resilient user account for authenticating to the Resilient platform. It is recommended that this account is dedicated to integrations.
password	Required. Password for the Resilient user account.
no_prompt_password	If set to False (default) and the "password" value is missing from this config file, the user is prompted for a password. If set to True, the user is not prompted.
stomp_port	Port number for STOMP. Default is 65001.
componentsdir	Path to directory containing additional Python modules. Resilient Circuits can load custom components from this directory. Typically this option is only used by integration developers.
noload	Optional. Comma-separated list of: • Installed components that should not be loaded. • Module names in the componentsdir that should not be loaded. Example: my_module, my_other_module, InstalledComponentX

Parameter	Description
proxy_host	IP or Host for Proxy to use for STOMP connection. By default, no proxy is used.
proxy_port	Port number for Proxy to use for STOMP connection. By default, no proxy is used.
proxy_user	Username for authentication to Proxy to use for STOMP connection. If a proxy_host is specified and no proxy_user specified, then assumed no authentication is required.
proxy_password	Password for authentication to Proxy to use for STOMP connection. Used in conjunction with proxy_user.
cafile	Path and file name of the PEM file to use as the list of trusted Certificate Authorities for SSL verification when the Resilient platform is using untrusted self-signed certificates. If not using a trusted certificate, **cafile** must be set to False. If there is a PEM file, use a second instance of **cafile** to set to True or False. If set to False, certificate verification is not performed and the PEM file is used. If set to True (default), allow only trusted certs.

5.2. Using encrypted keys (optional)

The Resilient integration server includes a utility to add all of the keyring-based values from your app.config file to your system's compatible keyring backend, such as a 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):
```

You can also use the utility to store passwords or other authentication values used by an extension package.

To pull configuration values from your KeyStore system into the app.config file, enter ^<*key*>. For example:

```
[resilient]
api_key_secret =^resilient_secret
```

To pull environmental variables from your KeyStore system in the app.config file, enter $<*key*>. For example:

```
[resilient]
api_key_secret =$resilient_secret
```

5.3. Using multiple configuration files (optional)

By default, every Resilient extension package you install uses the same configuration file. If you need multiple 'resilient-circuits' applications running with different configuration files, you can set the APP_CONFIG_FILE environment variable to a different path for each process. For Linux, use the following command:

```
resilient-circuits config -c /path/to/<filename>.config
```

5.4. Monitoring the config file for changes

You can configure Resilient Circuits to monitor the app.config file for changes. When it detects a change has been saved, it updates its connection to the Resilient platform and notifies all components of the change. To enable this option, install the "watchdog" package.

```
pip install watchdog
```

Now you can run with:

```
resilient-circuits run –r
```

Without the –r option, changes to the app.config file have no impact on a running instance of Resilient Circuits. Note that not all components currently handle the reload event and may continue using the previous configuration until Resilient Circuits is restarted.

6. Extension packages

Once Resilient Circuits is installed and running, you can install extension packages on your server.

Whenever you install a new package, it creates a new section in the app.config file. You need to update that section in accordance with the instructions that came with the extension package. Afterwards, you can deploy it to your Resilient platform for customization and testing. Also see the document provided with the extension for details on how to customize and use the extension once deployed.

6.1. Downloading extensions

You can download functions and other extension packages from the IBM Resilient Community or IBM X-Force App Exchange. Once downloaded, perform the following to install and configure the package on your Resilient integration server.

6.2. Installing a package

1. Use ssh to connect to your Resilient integration server.
2. Go to the folder where the package is located.
3. Install the package by first unzipping the file then using the following command:

   ```
   pip install --user <package_name>-x.x.x.tar.gz
   ```

   The `--user` ensures that the libraries are installed in the home directory of the user you are logged in as (integration) and avoids conflicts with other Python libraries in the system directory.
4. Verify that the component is installed using the following command.

   ```
   resilient-circuits list
   ```
5. After installing the package, run the following command to update app.config. This command adds a new section with default values in the app.config file.

   ```
   resilient-circuits config -u
   ```

   Alternately, you can specify which packages to update app.config.

   ```
   resilient-circuits config -u -l <package1> <package2>
   ```

   If using an alternate file location for your app.config file, you need to specify it when you update.

   ```
   resilient-circuits config -u /path/to/app.config
   ```
6. Follow the instructions in the component's documentation file to edit the app.config file. Depending on the requirements of the extension, you may need to modify the default values to fit your environment, such as credentials to a 3rd party system.

6.3. Upgrading a package

Use the same procedure as installing a package, except use the –U option in the install command:

```
pip install -U --user <package_name>-x.x.x.tar.gz
```

6.4. Deploying and testing

Use the following command to deploy the components in a package to the Resilient platform:

```
resilient-circuits customize
```

You are prompted deploy the components, such as functions, message destinations, workflows and rules. You can use the optional parameter, -y, in the command line to not be prompted.

Optionally, if using Resilient Circuits V31 or later and your package has multiple functions, you can specify which functions to deploy as follows.

```
resilient-circuits customize –l <function_name1> <function_name2>
```

7. 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.

8. Support

For additional support, contact support@resilientsystems.com.

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