IBM Resilient

Incident Response Platform Integrations

Utility Function V1.0.3

Release Date: October 2018

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 Utility Function.

Overview

The Utility Functions integration package contains several useful workflow functions for common automation and integration activities in Resilient.

This document describes each utility function, how to configure it in custom workflows, and any additional customization options.

Licensed Materials – Property of IBM

© Copyright IBM Corp. 2010, 2018. All Rights Reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Installation

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

- Resilient platform is version 30 or later.
- Resilient platform is connected to the internet.
- 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 the command line of the Resilient appliance, which hosts the Resilient platform; or to a separate integration server where you will deploy and run the functions code. If using a separate integration server, you must install Python version 2.7."x", where "x" is 10 or later, and "pip". (The Resilient appliance is preconfigured with a suitable version of Python.)

Install the Python components

The utility functions package contains Python components that will be 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.

Ensure that the environment is up to date,

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

To install the package, if downloaded from the AppExchange as a .zip file, first unpack it and then run:

```
sudo pip install --upgrade fn_utilities-1.0.3.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.

Perform the following to configure and run the integration:

1. Using sudo, become the integration user.

   ```
   sudo su - integration
   ```

2. 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
   ```

3. Edit the resilient-circuits configuration file.

   a. In the [resilient] section, ensure that you provide all the information needed to connect to the Resilient platform.

   b. In the [fn_utilities] section, edit the settings as required. For details on the 'shell' commands, see the Utilities: Shell Command section later in this guide.

Import customizations into 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.

Install these customizations to the Resilient platform with the following command:

```
resilient-circuits customize
```

Answer the prompts to import functions, message destinations, workflows and rules.

Run the integration framework

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.

For normal operation, resilient-circuits must run continuously. The recommended way to do this is to configure 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.

The unit file should be named 'resilient_circuits.service':

```
sudo vi /etc/systemd/system/resilient_circuits.service
```

The contents:

```
[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
```

Ensure that the service unit file is correctly permissioned:

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

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

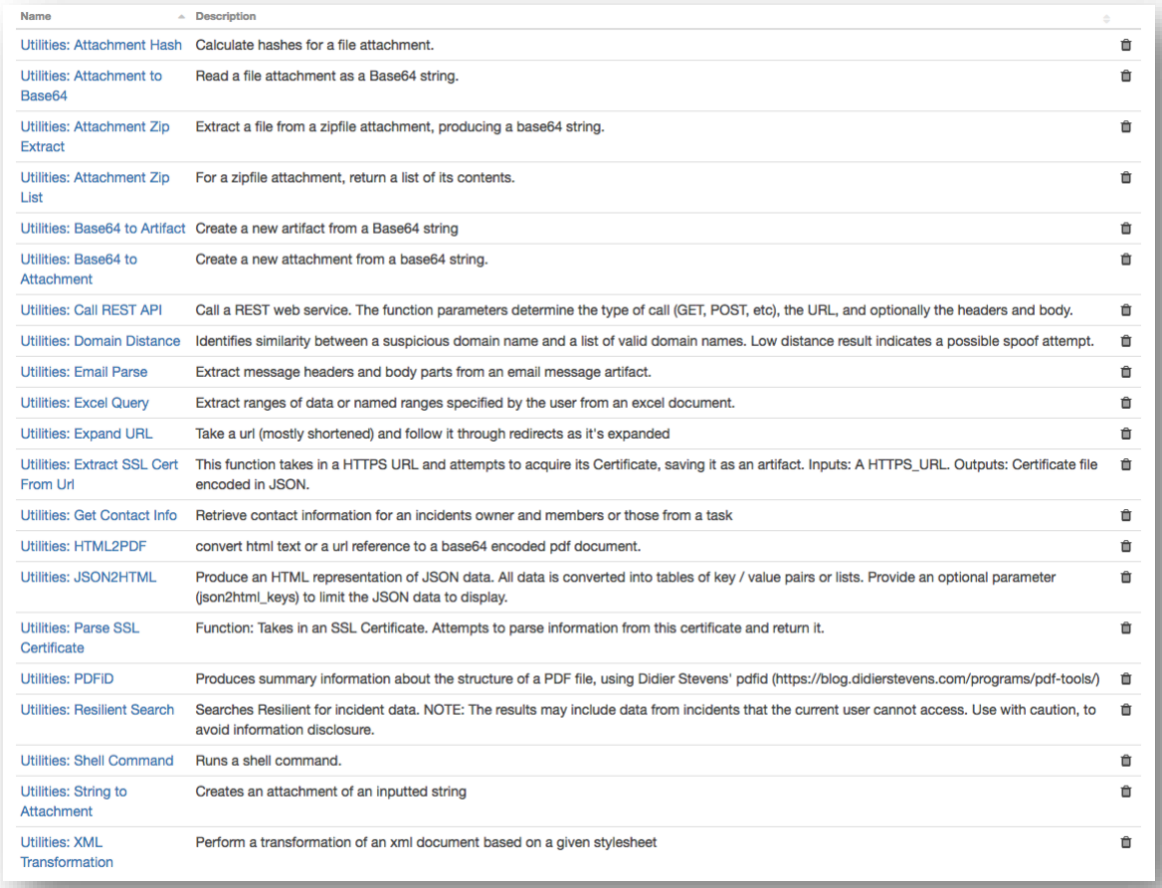
```
sudo systemctl resilient_circuits [start|stop|restart|status]
```

Log files for systemd and the resilient-circuits service can be viewed through the journalctl command:

```
sudo journalctl -u resilient_circuits --since "2 hours ago"
```

Function Descriptions

Once the utility package deploys the functions, 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 and modify these workflows and rules for your own needs.



Name	Description
Utilities: Attachment Hash	Calculate hashes for a file attachment.
Utilities: Attachment to Base64	Read a file attachment as a Base64 string.
Utilities: Attachment Zip Extract	Extract a file from a zipfile attachment, producing a base64 string.
Utilities: Attachment Zip List	For a zipfile attachment, return a list of its contents.
Utilities: Base64 to Artifact	Create a new artifact from a Base64 string
Utilities: Base64 to Attachment	Create a new attachment from a base64 string.
Utilities: Call REST API	Call a REST web service. The function parameters determine the type of call (GET, POST, etc), the URL, and optionally the headers and body.
Utilities: Domain Distance	Identifies similarity between a suspicious domain name and a list of valid domain names. Low distance result indicates a possible spoof attempt.
Utilities: Email Parse	Extract message headers and body parts from an email message artifact.
Utilities: Excel Query	Extract ranges of data or named ranges specified by the user from an excel document.
Utilities: Expand URL	Take a url (mostly shortened) and follow it through redirects as it's expanded
Utilities: Extract SSL Cert From Url	This function takes in a HTTPS URL and attempts to acquire its Certificate, saving it as an artifact. Inputs: A HTTPS_URL. Outputs: Certificate file encoded in JSON.
Utilities: Get Contact Info	Retrieve contact information for an incidents owner and members or those from a task
Utilities: HTML2PDF	convert html text or a url reference to a base64 encoded pdf document.
Utilities: JSON2HTML	Produce an HTML representation of JSON data. All data is converted into tables of key / value pairs or lists. Provide an optional parameter (json2html_keys) to limit the JSON data to display.
Utilities: Parse SSL Certificate	Function: Takes in an SSL Certificate. Attempts to parse information from this certificate and return it.
Utilities: PDFiD	Produces summary information about the structure of a PDF file, using Didier Stevens' pdfid (https://blog.didierstevens.com/programs/pdf-tools/)
Utilities: Resilient Search	Searches Resilient for incident data. NOTE: The results may include data from incidents that the current user cannot access. Use with caution, to avoid information disclosure.
Utilities: Shell Command	Runs a shell command.
Utilities: String to Attachment	Creates an attachment of an inputted string
Utilities: XML Transformation	Perform a transformation of an xml document based on a given stylesheet

The following sections describe each function.

Utilities: Attachment Hash

This function produces hashes of a file attached to an incident. Provide the incident ID and attachment ID as input, and the output includes md5, sha1, sha256 and other hashes of the file content. Those hashes can then be used as artifacts or in other parts of your workflows.

Utilities: Attachment Zip List

This function reads a ZIP file attached to an incident, and produces lists of the zip file contents. A first list contains the file paths. A second list contains detailed information about each file including its path, size, and other attributes.

Utilities: Attachment Zip Extract

This function reads a ZIP file attached to an incident, and extracts one of the files. Provide the file path as input to the function. For flexibility, the file contents are not directly attached to the incident, but are returned to your workflow as a base64-encoded string. That string can then be used as input to subsequent functions that might write it as a file attachment, as a malware sample artifact, or in other ways.

Utilities: Attachment to Base64

This function reads a file attachment in the incident, and produces a base64-encoded string with the file attachment content. This content can then be used in combination with other workflow functions to create an artifact, a new file attachment, or to analyze the contents using various tools.

Utilities: Base64 to Artifact

This function creates a new file artifact in the incident, using the base64-encoded string that your workflow provides as input. Other function inputs allow you to specify the artifact type (log file, malware sample, and so on) and description.

Utilities: Base64 to Attachment

This function creates a new file attachment in the incident, using the base64-encoded string that your workflow provides as input.

Utilities: Call REST API

This function calls a REST web service. It supports the standard REST methods: GET, HEAD, POST, PUT, DELETE and OPTIONS.

The function parameters determine the type of call, the URL, and optionally the headers and body. The results include the text or structured (JSON) result from the web service, and additional information including the elapsed time.

Utilities: Domain Distance

This function calculates the similarity between a suspicious domain name (or other string) and a list of valid reference domain names (such as a list of domains owned by your organization).

The comparison takes account of confusable Unicode characters such as similar-looking Greek, Cyrillic and Latin letters. It then measures the similarity using the “damerau levenshtein distance”, which counts the number of changes required from one string to another. Low distance results indicate a possible spoof attempt.

Utilities: Email Parse

This function extracts message headers and body parts from an email message in EML format. Provide the email message content as a base64-encoded string, for example the output from the “Attachment to Base64” function.

The results including subject, sender and recipients, other headers, and other content can then be added to your incident as artifacts or notes, or used in other ways.

Utilities: Expand URL

This function takes any URL, particularly shortened URLs, and following the path redirects to its destination. The results return each URL which are added to a new artifact.

Utilities: Extract Excel Query

This function takes in three inputs to specify the attachment and two strings – one listing Excel style ranges (such as “Example Sheet”!A1:B2, or ‘“Sheet1”!A1:B2 “Ex2.”!A1’) and a list of comma separated names of defined ranges – that specify the data to be extracted from the provided attachment of Excel format. The function uses a Python library called openpyxl (http://openpyxl.readthedocs.io/en/stable/) to interface with Excel files.

The output data format:

```
{
  "titles": [List of String names of the sheets],
  "sheets": {
    "_keys": ["Sheet Name", "Other sheet", ...],
    "Sheet Name": {
      "_keys": ["A1:B5", "Other range"],
      "A1:B5": list of rows,
      "Other range": [[row0], [...], [...]]
    },
    "Other sheet": {}
  },
  "defined_names": {
    "_keys": ["defined range 1"],
    "defined range 1": {
      The name structure as in "sheets”, but with an extra defined_names
layer
    }
  }
}
```

The “_keys” list in every dictionary object can be used to iterate over the dictionaries, which is otherwise impossible in a post-process script. The topmost layer does not have _keys because the user always knows if sheets or defined_names are included, based on whether they requested it.

Utilities: Extract SSL Cert from URL

This function takes in a HTTPS URL or DNS input, establishes a connection and then attempts to acquire the SSL certificate. If successful, the function then saves the certificate as an artifact of type ‘X509 Certificate File’. Works on most URLs including those with self-signed or expired certificates.

The output of this function is a string representation of the certificate saved in PEM format.

Utilities: Get Contact Info

This function collects contact information from an incident’s or a task’s owner and members which can be used by downstream functions. The information collected includes:

- first name
- last name
- display name
- title

- email address
- phone number
- cell number

Utilities: JSON2HTML

Produce an HTML representation of JSON data. All data is converted into tables of key / value pairs or lists. Provide an optional parameter (`json2html_keys`) to limit the JSON data to display. For the example below, specifying `key1.key2.key3` will only convert the JSON data associated with that key path.

The following JSON data produces the formatted html as:

```
{ "key1": {
    "key2": {
      "a": "a1",
      "b": "b1",
      "c": "c1",
      "key3": [
        "d",
        "e",
        "f"
      ]
    }
  }
}
```

key1	key2		
		a	a1
		b	b1
		c	c1
		key3	• d • e • f

Utilities: Parse SSL Certificate

This function produces the structured data from a provided SSL certificate. Three inputs are accepted by the function. There are 2 defined ways to use this function for parsing certificates.

Option 1 involves providing a JSON-encoded representation of a certificate. In this case the certificate input parameter should be this JSON string.

Option 2 involves providing a certificate file for parsing. When the rule is triggered on an artifact, both the incident_id for that incident and the artifact_id for the specified certificate file must be provided.

NOTE: The Parse SSL Certificate function expects a certificate of type PEM. If you require a way to get a PEM formatted certificate from a URL consider using this in conjunction with the Extract SSL Cert from URL function.

Utilities: PDFiD

Produces summary information about the structure of a PDF file, using Didier Stevens' pdfid (https://blog.didierstevens.com/programs/pdf-tools/). Provide the PDF file content as a base64-encoded string, for example the output from the “Attachment to Base64” function.

This function is useful in initial triage of suspicious email attachments and other files. It allows you to identify PDF documents that contain (for example) JavaScript or that execute an action when opened. PDFiD also handles name obfuscation. The combination of PDF automatic action and JavaScript makes a document very suspicious.

Utilities: Resilient Search

This function searches the Resilient platform for incident data according to the criteria specified, and returns the results to your workflow. It can be used to find incidents containing data that matches any string, or incidents currently assigned to a given user, or a very wide range of other search conditions.

> **NOTE**: The search results may include data from incidents that the current Resilient user (the person who triggered the workflow) cannot access. Often your Resilient users have the “Default” role that allows them to only see incidents where they are members. This function runs with the permissions of your integration account, which typically may have much wider access privileges. Use with caution, to avoid information disclosure.

Utilities: Shell Command

This function allows your workflows to execute shell-scripts locally or remotely, and return the result into the workflow. The results include the ‘stdout’ and ‘stderr’ streams, the return code, and information about the execution time. If the output of the shell script is JSON, it is returned as structured data. Results can then be added to the incident as file attachments, artifacts, data tables, or any other uses.

These functions can be run on any platform. If you install and run the resilient-circuits framework on Windows, this allows you to configure this function to run PowerShell scripts.

NOTE:

- Remote commands must specify a target Windows machine that has Windows Remote Management (WinRM) enabled. This can be done by running winrm qc in the remote computer’s command prompt.
- Remote shells have a max memory that may not be sufficient to run your script; to change this value you must set MaxMemoryPerShellMB.

For security, the list of available shell commands must be configured explicitly by the administrator. To do this, edit the [fn_utilities] section of the ‘app.config’ file.

Simple examples of commands include:

```
# shell_command default commands (unix)

nslookup=nslookup "{{shell_param1}}"
dig=dig "{{shell_param1}}"
traceroute=traceroute -m 15 "{{shell_param1}}"
```

NOTE: The parameter values {{shell_param1}}, {{shell_param2}}, {{shell_param3}} may contain spaces, dashes and other characters. In your command configuration, they must be surrounded with double-quotes. Failure to properly quote your command parameters creates a security risk, since the parameter values usually come from artifacts and other untrusted data.
For remote powershell scripts the `shell_param1`, `shell_param2`, and `shell_param3` values map `$args[0]`, `$args[1]`, and `$args[2]` respectively in the Powershell script.

Shell commands with more complex parameters can be configured according to your needs. For example, if you use the Volatility forensics framework, example commands might include the configuration below. This allows you to fully integrate Volatility and other tools into your Resilient workflows.

In these examples, the first parameter is filename of the memory image, assuming $VOLATILITY_LOCATION is set in the environment (such as in the system unit configuration). The second parameter is the Volatility profile ("Win7SP0x64" etc).

```
imageinfo=python /path/to/vol.py -f "{{shell_param1}}" imageinfo --
output=json

kdbgscan=python /path/to/vol.py -f "{{shell_param1}}" --
profile="{{shell_param2}}" kdbgscan --output=json

psscan=python /pathto/vol.py -f "{{shell_param1}}" --
profile="{{shell_param2}}" psscan --output=json

dlllist=python /path/to/vol.py -f "{{shell_param1}}" --
profile="{{shell_param2}}" dlllist --output=json
```

To configure running scripts remotely, the user must make these changes to the config file:

- Specify acceptable powershell compatibile extensions of script files:
  `remote_powershell_extensions=ps1`
- Specify the transport authentication method
  `remote_auth_transport=ntlm`
- Specify remote commands in the config file wrapped in square brackets []
  `remote_command=[C:\remote_directory\remote_script.ps1]`
- Specify a remote computer in the config file to run the script wraped in parentheses ()
  `remote_computer=(username:password@server)`

These two configurations have specific syntax: remote commands must specify a remote directory wrapped in square brackets and remote computers must have the credentials wrapped in parentheses. The syntax to run the command from the workflow is then:
`remote_command:remote_computer` to run the specified remote command on the specified remote computer

Note that changes to ‘app.config’ are usually only available after the ‘resilient-circuits run’ process is restarted.

Some examples of remote commands include:

```
# Remote commands
remote_command1=[C:\scripts\remote_script.ps1]
remote_command2=[C:\scripts\another_script.ps1]

# Remote computers
remote_computer1=(domain\administrator:password@server1)
remote_computer2=(domain\admin:P@ssw0rd@server2)
```

These remote commands can then be run in the workflow using the syntax `remote_command:remote_computer` as the input for `shell_command`. Examples:

- `remote_command1:remote_computer1` runs `remote_command1` remotely on `remote_computer1`
- `remote_command2:remote_computer1` runs `remote_command2` remotely on `remote_computer1`
- `remote_command1:remote_computer2` runs `remote_command1` remotely on `remote_computer2`
- `remote_command2:remote_computer2` runs `remote_command2` remotely on `remote_computer2`

The function payload has these fields:

```
"commandline": the command that ran
"start": timestamp, epoch milliseconds
"end": timestamp, epoch milliseconds
"elapsed": milliseconds
"exitcode": nonzero indicates that the command failed
"stdout": text output from the command
"stderr": error text output from the command
"stdout_json": object parsed from JSON output from the command
"stderr_json": object parsed from JSON error output from the command
```

Utilities: String to Attachment

This function creates a new file (.txt) attachment in the incident or task from a string that your workflow provides as input.

Utilities: XML Transformation

This function transforms an XML document using a preexisting xsl stylesheet. The resulting content is returned.

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 Platform 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 creates progress information. Failures show up as errors and may contain python trace statements.

Support

For additional support, contact support@resilientsystems.com.

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