Resilient example email message parsing script

This package consists of the GenericEmailScript.res Resilient configuration file which contains an example email parsing script and a rule to run it automatically.

Installation instructions

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

- Resilient platform is version 32 or later.
- You have a Resilient account to use for the installation. 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.

Procedure

1. Log on to the Resilient platform using a suitable account.
2. Navigate to **Administrator Settings**.
3. Select the **Organization** tab.
4. Select the **Import** link.
5. Select the **+ Import settings** button.
6. Select the **GenericEmailParser.res** file from the installation bundle.
7. Select **Open**.
8. Select **Proceed**.

Result

After installing, the Resilient platform has a new Python script called "Generic email script" and a new rule called "Process email message". The rule runs the script when it is triggered by a new email message being received by the Resilient platform. The script is intended to perform generic email parsing on newly created email message objects. It performs the following:

- Checks if an existing incident exists whose title reflects the email message received.
    - If so, it associates the email message with the existing incident.
    - If not, it:
        - Creates a new incident with a suitable title.
        - Associates the email message with the new incident.
        - Adds the email message's subject as an artifact to the new incident.
        - Sets the incident's reporter field to be the email address that sent the message.
- Parses the email body text looking for URLs, IP addresses and file hashes. After filtering out invalid and whitelisted values, it adds the remaining data to the incident as artifacts.
- Adds non-inline email message attachments to the incident.

Configuration

The incident owner

New incidents need an owner, either an individual identified by their email address or a group name. In the provided script, every incident is owned by the user admin@co3sys.com. This should be changed to reflect your Resilient platform. For example, to change the owner to l1@businessname.com, locate line 507 of the script:

```
# The new incident owner
newIncidentOwner = "admin@co3sys.com"
```

Edit the line:

```
# The new incident owner
newIncidentOwner = "l1@businessname.com"
```

Whitelisting

There are two categories of whitelist - IP address and URL domain - which are configured by altering data in the script.

Variable Name	Line number	Purpose
`ipV4WhiteList`	214	General IP v4 whitelist
`ipV6WhiteList`	232	General IP v6 whitelist
`customIPv4WhiteList`	257	Additional customer-specific IP v4 whitelist
`customIPv6WhiteList`	258	Additional customer-specific IP v6 whitelist
`domainWhiteList`	261	General URL domain whitelist
`customDomainWhiteList`	264	Additional customer-specific URL domain whitelist

IP address whitelists

The IP address whitelists are divided into separate IPv4 and IPv6 lists. These lists apply to the IP addresses retrieved by pattern matching in the body of the email message. If an IP address appears on a whitelist then it is not be added as an artifact to the incident. There are standard whitelists which are expected to be the same for all customers, and customer-specific whitelists which are used in addition to the standard lists. The customer-specific lists are kept separate so that there are four whitelist variables in the script.

Additions to the whitelist should be made to `customIPv4WhiteList` and `customIPv6WhiteList`.

There are two categories of IP whitelist entry, CIDR (Classless Inter-Domain Routing) and IPRange. For example, in IP V4, IBM owns the "9" class A network. You may want to also whitelist an IP range, such as 12.0.0.1 - 12.5.5.5. To add this criterion to the whitelist we would add the following to customIPv4WhiteList:

```
CIDR("9.0.0.0/8"),
IPRange("12.0.0.1-12.5.5.5")
```

You may also want to whitelist an explicit IP address, such as 13.13.13.13. This would be specified by:

```
CIDR("13.13.13.13")
```

IP v6 whitelists operate similarly. For example to whitelist a V6 CIDR aaaa::/16 we would add CIDR("aaaa::/16") to customIPv6WhiteList. For example:

```
  # Customer-specific IP address whitelists
  # Add entries to these lists to whitelist the entries without disrupting
the standard set above
  customIPv4WhiteList = []
  customIPv6WhiteList = []
```

should become:

```
  # Customer-specific IP address whitelists
  # Add entries to these lists to whitelist the entries without disrupting
the standard set above
  customIPv4WhiteList = [ CIDR("9.0.0.0/8"),
                          IPRange("12.0.0.1-12.5.5.5"),
                          CIDR("13.13.13.13") ]
  customIPv6WhiteList = [ CIDR("aaaa::/16") ]
```

URL domain whitelists

The domain whitelist applies to URLs found in the body of the email. If a whitelisted domain is discovered in a potential URL artifact, then it is not added to the incident. Domains can be added explicitly, such as mail.businessname.com, or using a wildcard, such as *.otherbusinessname.com. For example:

```
  # Customer-specific domain whitelist
  customDomainWhiteList = []
```

would become:

```
  # Customer-specific domain whitelist
  customDomainWhiteList = [ Domain("mail.businessname.com"),
                            Domain("*.otherbusinessname.com") ]
```

Extension and Customization

There are two approaches to customization of the mechanism:

- Running multiple scripts for the same email
- Modifying the supplied script

For a variety of reasons, adding more scripts is generally a better idea than adding more complexity to one script. The Resilient platform could be expected to ingest multiple categories of email messages from different integrations. Some of the processing of the email messages could be common, and some processing could be category- or integration-specific. Keeping the common processing in one script, and the specialized processing in others would allow a cleaner and more maintainable implementation.

Each script execution is run within defined computational quota limits - 5 seconds of execution time or 50,000 lines of Python executed. Regular Expression processing is performed by the ***re*** Python module, execution of which is considered part of the quota. It is possible to create a complex regular expression whose execution requires a great many lines of Python to be interpreted on a particular email message. Execution of many such complex regular expressions could overrun the 50,000 line limit.

Examples

Extending the solution to deal with Phishing reports

Scenario: Emails arriving in a particular mailbox reflect individuals forwarding suspected phishing messages. The scripts operating on these email messages should, in addition to the generic processing, record the reporter's email address as possibly having been the target of a phishing attack, and record the sender of the forwarded phishing email as suspicious.

A solution: Add the following script to the Resilient platform:

```
import re

def addArtifact(regex, artifactType, description):
  """This method adds new artifacts to the incident derived from matches
of the the regular expression
  parameter within the email body contents.
  Parameter "regex" - a regular expression to match against the email body
contents.
  Parameter "artifactType" - the type of the artifact(s).
  Parameter "description" - the description of the artifact(s).
  """
  dataList = set(re.findall(regex, emailmessage.body.content)) # Using a
set to enforce uniqueness
  if dataList is not None and len(dataList) > 0 :
    map(lambda theArtifact: incident.addArtifact(artifactType,
theArtifact, description), dataList)

###
# Mainline starts here
###

# Add "Phishing" as an incident type for the associated incident
incident.incident_type_ids.append("Phishing")
```

```
# Add the email sender information to the incident as the recipient of the
phishing attempt
reportingUserInfo = emailmessage.from.address
if emailmessage.from.name is not None:
  reportingUserInfo = u"{0} <{1}>".format(emailmessage.from.name,
emailmessage.from.address)
incident.addArtifact("Email Recipient", reportingUserInfo, "Recipient of
suspicious email")

# Extract email sender information on the assumption that a fishing email
is being forwarded
if not emailmessage.body.content is None:
  addArtifact(r"From: (.*)\n", "Email Sender", "Suspicious email sender")
  addArtifact(r"Reply-To: (.*)\n", "Email Sender", "Suspicious email
sender (Reply-To)")
```

Run the script as part of a rule that includes a condition that helps identify the email message as a phishing report. The script should run either as part of a multi-script rule that first runs the generic script, or as a separate rule that runs afterwards. It is important that the phishing-specific script should run after the generic script because the generic script causes the `incident` variable to be set, and the phishing-specific script expects this to have been done already.

Campaign identifier

Scenario: The customer wants to collect email messages related to the same campaign of attack to a single incident.

A solution:

1. Create a new incident custom field for the campaign signifier of type Text.
2. Copy the generic parsing script into a new script, where this script is used for the category of email messages in question.
3. Modify the mainline of the new script to create a value for the campaign signfier.
4. Use the signifier field for the incident search criteria instead of the incident title.
5. For new incidents, set the campaign signifier field to be the signifier value.
6. Modify the rules so that the new script runs instead of the generic script.