opentext™

ArcSight SOAR CE 24.2 (v3.11)

Integration Plug-In Development Guide

opentext™

Contents

Description

This guide walks through the key concepts and structure of ArcSight SOAR integration plug-ins and provides step-by-step descriptions with code samples for both REST-API and SSH-based integrations.

Integration Plug-In Concepts/Glossary

Integration

An integration is the definition of plug-in and configuration for communicating with a third-party system to perform action and enrichment activities (for example, “Azure Active Directory”).

Capability

A capability is an action or enrichment command that is available for an integration (for example, “Get User Details”, “Disable User”). ArcSight SOAR integrations support two types of capabilities: “Action” and “Enrichment”.

Action

An action is a specific type of capability for giving a command to the remote system (for example, “Block IP”, “Disable User”, “Quarantine Host”).

Enrichment

Enrichment is a type of capability for retrieving additional data from the remote system to give more context to analysis (for example, “Get User Details”, “Query IP Reputation”).

Parameters

Parameters are used by capabilities as command arguments. Action and enrichment capabilities can get parameters from both case scope and from user input.

Duplicate Control

Action capabilities have support to check if the same action was taken before to avoid the duplicate entries/errors such as checking whether an IP address is blocked on the Firewall previously.

Rollback

The rollback mechanism ensures that the action taken is removed from the remote integration after a period that you specify. You can tell SOAR that an IP will be blocked on the firewall for a week. SOAR tracks the rollback time and removes the block from firewall automatically.

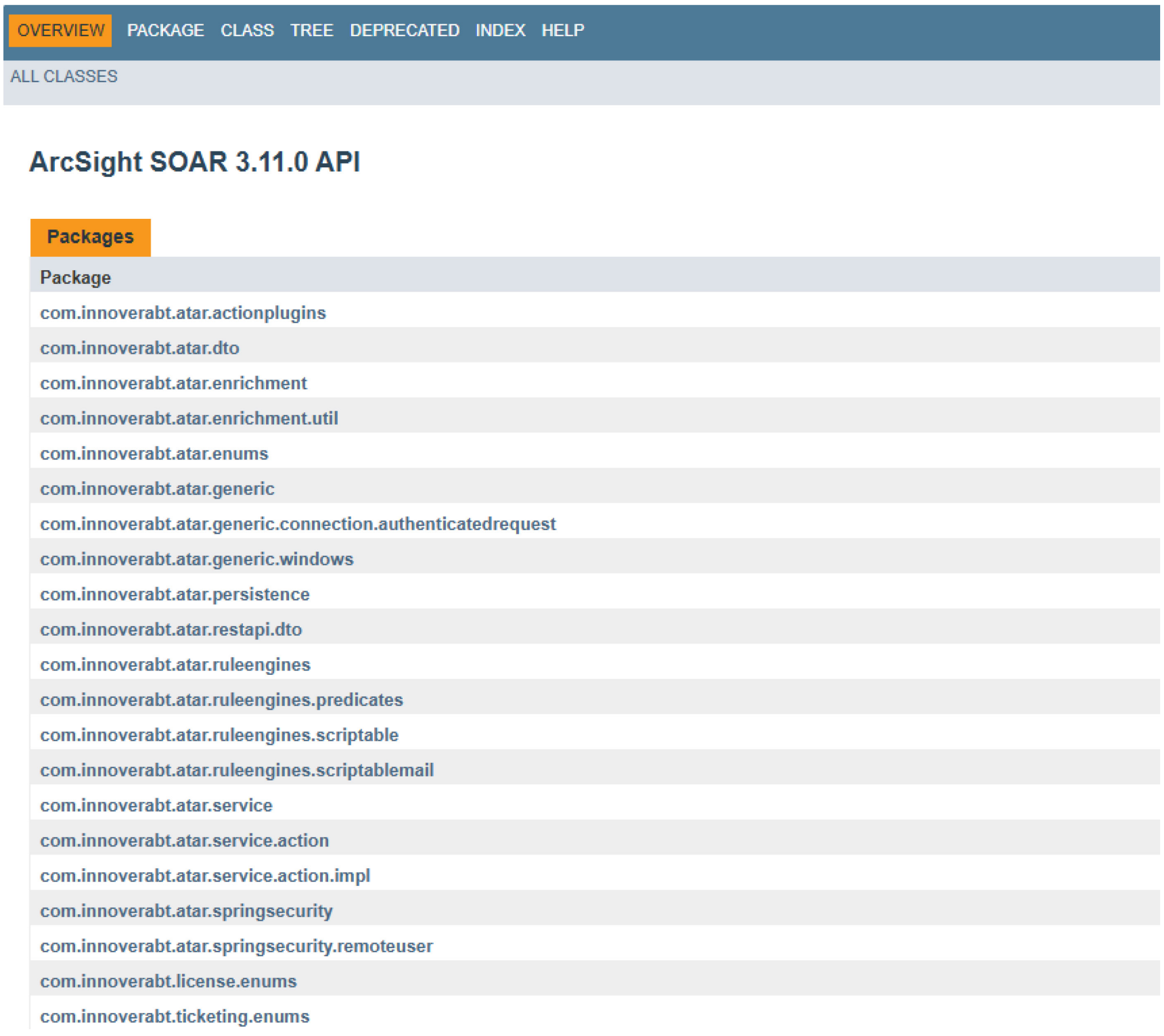
Scope

Scope is the collection of items (artifacts in other words) related to a case, such as username, IP address, MAC Address, Hostname, URL, etc.

Environment

Scripting API and Documentation

SOAR plug-ins support Jython compatibility. SOAR scripting API methods you need while developing integration plug-ins for SOAR are described in the ArcSight SOAR Scripting API which is accessible from the URL: https://<CDF_Master_IP_or_FQDN>/soar/js-api-doc/index.html?tenant=default



ArcSight SOAR 3.11.0 API

Packages

Package
com.innoverabt.atar.actionplugins
com.innoverabt.atar.dto
com.innoverabt.atar.enrichment
com.innoverabt.atar.enrichment.util
com.innoverabt.atar.enums
com.innoverabt.atar.generic
com.innoverabt.atar.generic.connection.authenticatedrequest
com.innoverabt.atar.generic.windows
com.innoverabt.atar.persistence
com.innoverabt.atar.restapi.dto
com.innoverabt.atar.ruleengines
com.innoverabt.atar.ruleengines.predicates
com.innoverabt.atar.ruleengines.scriptable
com.innoverabt.atar.ruleengines.scriptablemail
com.innoverabt.atar.service
com.innoverabt.atar.service.action
com.innoverabt.atar.service.action.impl
com.innoverabt.atar.springsecurity
com.innoverabt.atar.springsecurity.remoteuser
com.innoverabt.license.enums
com.innoverabt.ticketing.enums

The ArcSight SOAR Scripting API contains all the packages SOAR provides for scripting of automation bits, email parsers, plug-ins, etc. Packages that can be used in plug-ins are:

- com.innoverabt.atar.actionplugins
- com.innoverabt.atar.enrichment
- com.innoverabt.atar.enrichment.util
- com.innoverabt.atar.generic
- com.innoverabt.atar.generic.connection.authenticatedrequest
- com.innoverabt.atar.service.action

Important Methods to Know

ArcSight SOAR scripting API provides an extensive set of methods, but it is crucial to understand and practice the usage of the following ones for writing integration plug-ins.

Method	**Description**
actionCapability()	Action capability definition
actionParam()	Action parameters coming from outside of the case scope
addAlertScopeItemProperty()	Sets scope item property value
addScopeItem()	Adds new scope items to case scope with given Role and Category
enrichmentCapability()	Enrichment capability definition
enrichmentData()	Data returned as enrichment result. It is used to add enrichment result to case timeline
enrichmentParams()	Enrichment parameters coming from outside of the case scope
failEnrichment()	Fails the enrichment with a given message
getCapability()	Action/Enrichment Capability Name
getCredential().getName()	Credential name used in integration configuration
getDevice()	Returns device object
getDeviceConfig()	Reads the parameter value from integration configuration
getEnrichmentParam()	Enrichment parameter's value
getParameter()	Action parameter's value
scopedActionParam()	Action parameters coming from the case scope
scopedEnrichmentParam()	Enrichment parameters coming from the case scope
SSH Specific Methods	
connectSSH()	Initiates SSH connection
readUntil()	Reads the stream from connection until a given string is detected. It supports both exact string match and regular expressions

readLine()	Reads the stream from connection. The returned data can be used to match for a condition later
sendLine()	Send command to be executed
sendPassword()	Send password during the session if needed, such as increasing privilege
HTTP Specific Methods	
atar.http()	Builds HTTP request
proxy()	Uses the web proxy defined in integration configuration (if exists)
method()	Sets HTTP method to be used in request
urlConfigurer()	Builds the URL based on integration address
withPath()	Adds additional path to the end of the integration URL
configure()	Configures/Finalizes URL configuration and authentication provider configuration
withQueryParam()	Sends query parameters as part of the URL
body()	Sets request body to be sent
header()	Sets the request header
customHeaderAuthenticationProviderConfigurer()	Configures Authentication header using credential fields
basicAuthenticationProviderConfigurer()	Configures Basic Authentication header using credential fields
ignoreCertErrors()	Ignores SSL handshake errors because of invalid certificates
execute()	Executes the request built
response()	Response returned as result of the request made
statusCode()	HTTP status code of the response
isSuccessful()	Returns True if the request is executed successfully

Authentication and Working with Credentials

Credentials

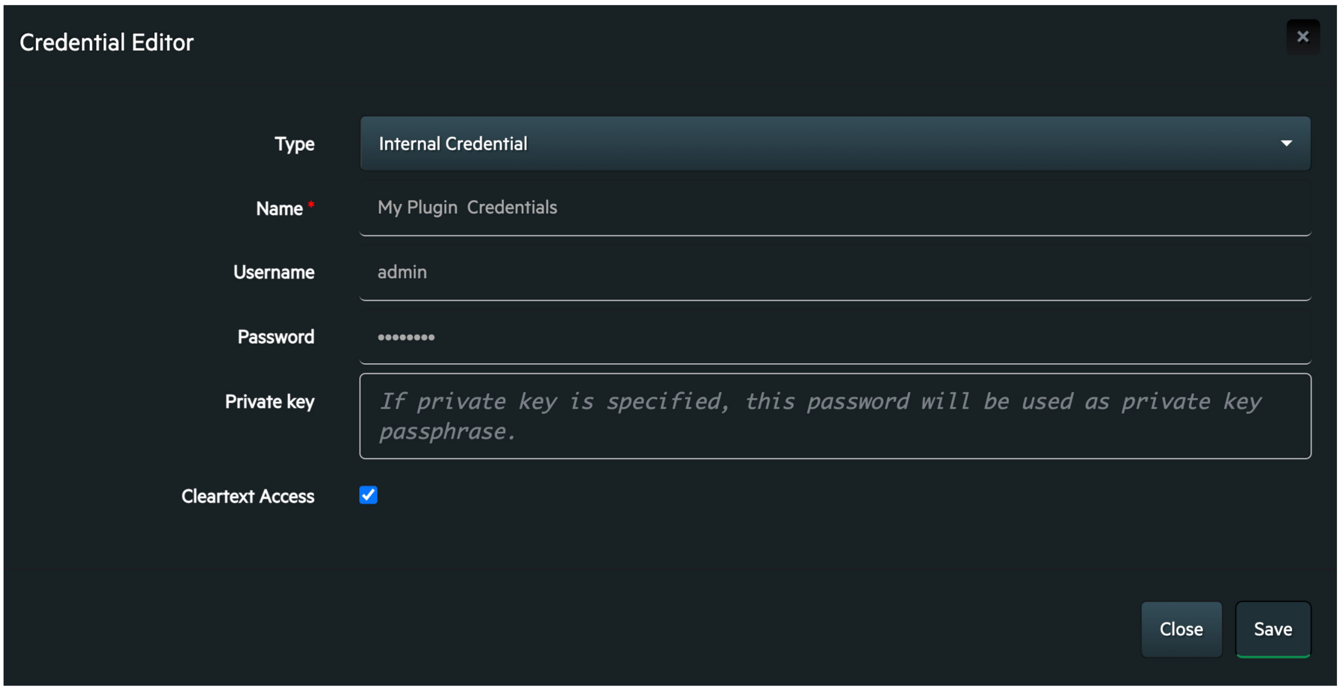
ArcSight SOAR uses credentials defined in integration configuration while authenticating the remote service. Credentials stored on SOAR contain the following 3 fields:

- USERNAME
- PASSWORD
- PRIVATE_KEY

You can also use “PASSWORD” and “PRIVATE_KEY” fields for storing API Keys, Client IDs and Client Secrets, etc.

Clear Text Access to Credentials

If SOAR's built-in authentication mechanisms don't work, and you need to implement an authentication mechanism, you may need to allow clear text access to the credential in credential definition as shown:


Credential Editor

Type: Internal Credential

Name *: My Plugin Credentials

Username: admin

Password: ••••••••

Private key: *If private key is specified, this password will be used as private key passphrase.*

Cleartext Access: ✓

You can access credential fields as:

```
credential_name = device.getCredential().getName()
credential = atar.cleartextCredentialDto(credential_name)
username = str(credential.getUsername())
passwd = str(credential.getPassword())
```

Authentication

SOAR provides the following classes and their methods to authenticate to remote HTTP Services (integrations) for various purposes:

- BasicAuthenticationProvider
- CustomHeaderAuthenticationProvider
- FormDataAuthenticationProvider
- ReplaceBodyAuthenticationProvider
- UrlParameterAuthenticationProvider

The methods and their usage are described below.

opentext™

Basic Authentication

The **BasicAuthenticationProvider** class lets you to manage basic authentication in HTTP requests. It reads the username and password fields of the credential given.

Sample code snippet:

```
request = atar.http() \
   .urlConfigurer(device) \
   .configure() \
   .basicAuthenticationProviderConfigurer(device.credential) \
   .username(AuthenticationCapableField.USERNAME) \
   .password(AuthenticationCapableField.PASSWORD) \
   .configure() \
   .method("GET")
```

Custom Header Authentication

The **CustomHeaderAuthenticationProvider** class lets you to authenticate to a remote service using an authentication header, typically with an API Key/Token, such as:

"API-Key: b7cc12f15d9666d354a493c102615172"

Sample code snippet:

```
request = atar.http() \
   .urlConfigurer(device) \
   .configure() \
   .customHeaderAuthenticationProviderConfigurer(device.credential) \
   .header("API-Key", AuthenticationCapableField.PRIVATE_KEY) \
   .configure() \
   .method("GET")
```

Form Authentication

The **FormAuthenticationProvider** class allows you to send credentials as form data.

Sample code snippet:

```
request = atar.http() \
   .urlConfigurer(device) \
   .configure() \
   .formDataAuthenticationProviderConfigurer(device.credential) \
   .addFormParameter("User", AuthenticationCapableField.USERNAME) \
   .addFormParameter("Password", AuthenticationCapableField.PASSWORD) \
   .configure() \
   .method("POST")
```

Replace Body Authentication

The **ReplaceBodyAuthenticationProvider** lets you to replace placeholder strings in the request body with credential fields.

Sample Code snippet:

```
request_body = {
   "id": 1,
   "method": "exec",
   "params": [
      {
         "data": {
            "passwd": "PASSWORD",
            "user": "USERNAME"
         },
         "url": "/sys/login/user"
      }
   ]
}

request = atar.http() \
   .urlConfigurer(device) \
   .configure() \
   .body(json.dumps(request_body)) \
   .replaceBodyAuthenticationProviderConfigurer(device.credential) \
   .replace("USERNAME", AuthenticationCapableField.USERNAME) \
   .replace("PASSWORD", AuthenticationCapableField.PASSWORD) \
   .configure() \
   .method("POST")
```

URL Parameter Authentication

The **URLParamaterAuthenticationProvider** lets you to send authentication credentials as URL parameters.

Sample code snippet:

```
request = atar.http() \
   .urlConfigurer(device) \
   .configure() \
   .urlParameterAuthenticationProviderConfigurer(device.credential) \
   .addParameter("User", AuthenticationCapableField.USERNAME) \
   .addParameter("Password", AuthenticationCapableField.PASSWORD) \
   .configure() \
   .method("GET")
```

Building and Executing HTTP Requests

The atar.http() method is used to build and execute HTTP requests on integrations. In general, HTTP requests contain:

- HTTP method
- Request Headers

- Query Parameters
- Request Body
- Endpoint URL

The following sample code can be used to build and handle HTTP requests based on integration URL.

```
def execute_http_request(method, request_headers=None, query_parameters=None, request_body=None, endpoint=None):

    log("Starting {} request to {} endpoint with query parameters {} with additional headers {}".format(method, endpoint, query_parameters, request_headers))

    request = atar.http() \
        .proxy(device) \
        .method(method) \
        .customHeaderAuthenticationProviderConfigurer(device.credential) \
        .header("Authorization", AuthenticationCapableField.PRIVATE_KEY) \
        .configure() \
        .urlConfigurer(device) \
        .withPath(endpoint)

    if query_parameters:
        for key, value in query_parameters.iteritems():
            request = request \
                .withQueryParam(str(key), str(value))

    if request_body:
        request \
            .configure() \
            .body(request_body)

    if device.ignoreSSLCertificationErrors:
        request \
            .configure() \
            .ignoreCertErrors()

    if request_headers:
        for key, value in request_headers.iteritems():
            request \
                .configure() \
                .header(str(key), str(value))

    request = request.configure()

    try:
        request = request.execute()

    except:
        log("HTTP Request Failed with Status Code:{}".format(request.statusCode()))
        raise Exception("HTTP Request Failed with Status Code:{} Response Body:{}".format(request.statusCode(), request.response()))
    else:
        log("HTTP Response gathered and returned")
        return request
```

In some cases, you may also need to send additional HTTP requests to a URL other than the integration URL. For example, the authentication flow may require you to authenticate through a different server. In such cases you can use **HTTPUrlBuilder()** and **setUrlBuilder()** methods.

```
url_builder = HttpURLBuilder(auth_url)
request = atar.http() \
   .proxy(device) \
   .method('POST') \
   .setUrlBuilder(url_builder)
```

Displaying Enrichment Results

ArcSight SOAR supports displaying enrichments results on case timeline in tabular form. It is done in three steps:

1. Get the response
2. Prepare data from response
3. Return prepared data with enrichmentData()

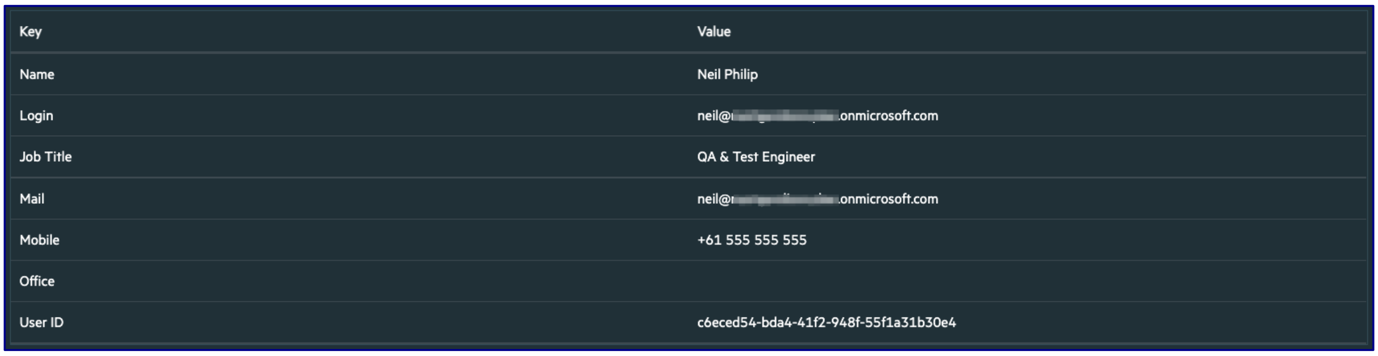
The following code snippet can be used to display a result as a Key-Value pair. It is suitable for displaying results of an enrichment that returns one entry:

```
def get_user():
   scope_item = atar.getEnrichmentParam("USER")[0].scopeItem.value
   response = execute_http_request('GET', None, None, None, '/users/' + scope_item).response()
   pretty_response = get_user_details(response)
   return atar.enrichmentData(capability.getCapability(), "JSON", json.dumps(pretty_response))

def get_user_details(response):
   user = json.loads(response)

   display_name = user['displayName']
   job_title = user['jobTitle']
   user_principal = user['userPrincipalName']
   user_id = user['id']
   mail = user['mail']
   mobile = user['mobilePhone']
   office_location = user['officeLocation']

   pretty_response = {
      "columns": ["Key", "Value"],
      "data": [["Name", display_name],
            ["Login", user_principal],
            ["Job Title", job_title],
            ["Mail", mail],
            ["Mobile", mobile],
            ["Office", office_location],
            ["User ID", user_id]
            ]
   }
   return pretty_response
```



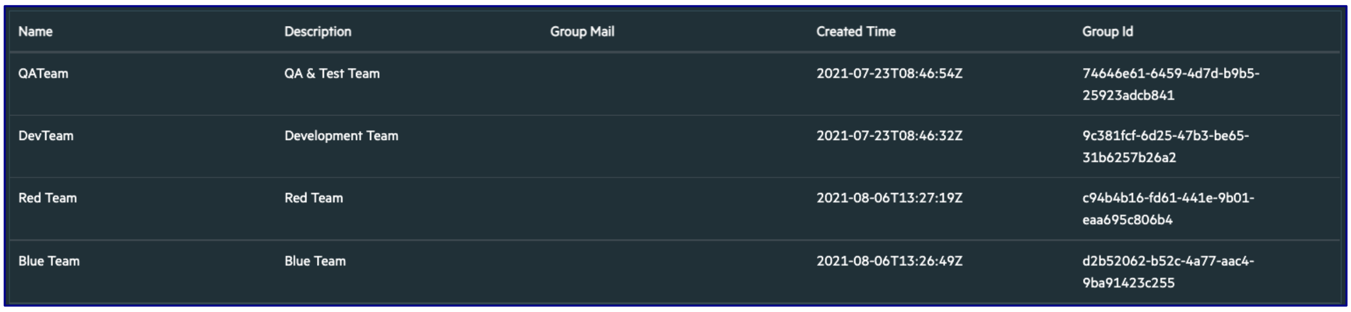
Key	Value
Name	Neil Philip
Login	neil@[redacted].onmicrosoft.com
Job Title	QA & Test Engineer
Mail	neil@[redacted].onmicrosoft.com
Mobile	+61 555 555 555
Office	
User ID	c6eced54-bda4-41f2-948f-55f1a31b30e4

The following code snippet can be used to display a result in multiple columns. It is suitable for displaying results of an enrichment that returns more than one entry:

```
def list_groups():
    response = execute_http_request('GET', None, None, None, '/groups').response()
    pretty_response = list_groups_details(response)
    return atar.enrichmentData(capability.getCapability(), "JSON", json.dumps(pretty_response))


def list_groups_details(response):
    pretty_response_data = []

    groups = json.loads(response)['value']
    for group in groups:
        pretty_response_data.append(
            [group['displayName'],
             group['description'],
             group['mail'],
             group['createdDateTime'],
             group['id']]
        )
    pretty_response = {
        "columns": ["Name", "Description", "Group Mail", "Created Time", "Group ID"],
        "data": pretty_response_data
    }
    return pretty_response
```



Name	Description	Group Mail	Created Time	Group Id
QATeam	QA & Test Team		2021-07-23T08:46:54Z	74646e61-6459-4d7d-b9b5-25923adcb841
DevTeam	Development Team		2021-07-23T08:46:32Z	9c381fcf-6d25-47b3-be65-31b6257b26a2
Red Team	Red Team		2021-08-06T13:27:19Z	c94b4b16-fd61-441e-9b01-eaa695c806b4
Blue Team	Blue Team		2021-08-06T13:26:49Z	d2b52062-b52c-4a77-aac4-9ba91423c255

opentext™

Extending Case Scope with Enrichments

Based on your use-case, you may need to extend case scope with enrichment results by adding new scope items and/or modifying scope item properties.

Adding Scope Item to Scope

Based on the enrichment result, if you want to extend case scope by adding a new scope item, you can use **addScopeItem()**.

```
addScopeItem(java.lang.String value, java.lang.String scopeItemRole, java.lang.String scopeItemCategory)
```

The Scope Item role value can be one of the following:

- IMPACT
- OFFENDER
- RELATED

The Scope Item category value can be one of the following:

- COMPUTER_NAME
- EMAIL_ADDRESS
- FILEDATA
- FILENAME
- HASH
- HOST
- KEYWORD
- MAC_ADDRESS
- NETWORK_ADDRESS
- PROCESS
- UNKNOWN
- URL
- USERNAME

```
atar.addScopeItem('user@example.com', 'RELATED','EMAIL_ADDRESS')
```

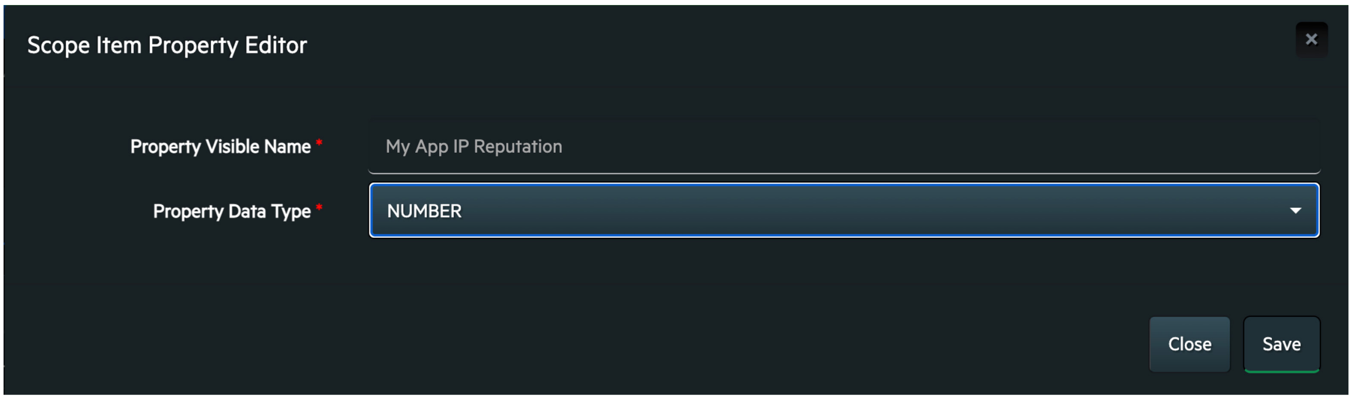
Setting Scope Item Property

ArcSight SOAR comes with more than 50 preconfigured scope item property definitions used by other integration plug-ins and supports the following data types for scope item property definitions:

- Boolean
- Json

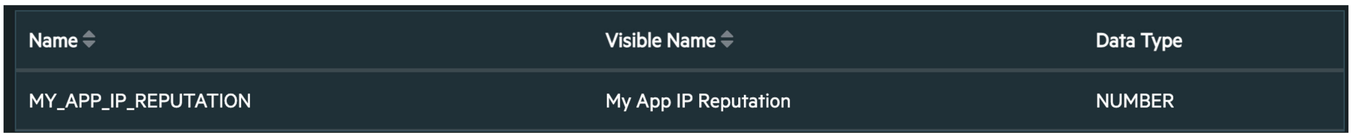
- Number
- Percentage
- Text

If you need to create a new scope item property, you can do it under the **Configuration/Scope Item Property** menu.


Scope Item Property Editor

Property Visible Name *: My App IP Reputation

Property Data Type *: NUMBER



Name	Visible Name	Data Type
MY_APP_IP_REPUTATION	My App IP Reputation	NUMBER

Within your code you can set or modify scope item property values by **addAlertScopeItemProperty()** method:

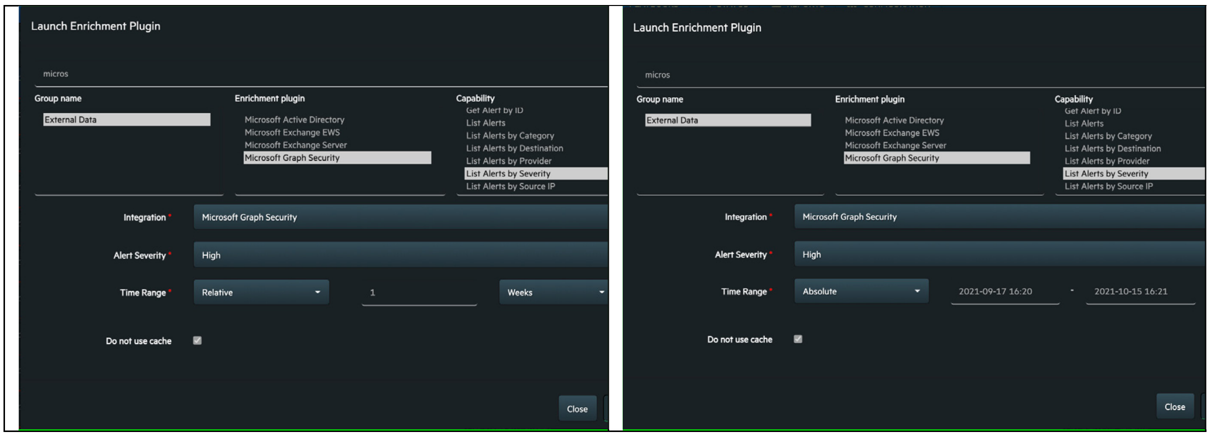
```
scope_item = atar.getEnrichmentParam("IP_ADDRESS")[0]
atar.addAlertScopeItemProperty(scope_item, 'MY_APP_IP_REPUTATION', '74')
```

Working With Time Ranges

In enrichment capabilities you sometimes need to provide a time range as a parameter, especially for limiting the scope while querying events or logs from a remote system. “TIME_RANGE” type of parameter allows selecting a Relative time range such as “last 5 days” or Absolute rime range: for example, 2021-08-14 15:10 – 2021-08-14 15:32.

A sample capability definition with time range parameter is:

```
atar.enrichmentCapability("LIST_ALERTS_BY_SEVERITY", "List Alerts by Severity", "EXTERNAL_DATA", [
   atar.enrichmentParam("DEVICE", "Integration", "Device"),
   atar.enrichmentParam("SEVERITY", "Alert Severity", ["High", "Medium", "Low", "Informational", "Unknown"]),
   atar.enrichmentParam("TIME_RANGE", "Time Range", "String")
]),
```


Launch Enrichment Plugin

micros

Group name	Enrichment plugin	Capability
External Data	Microsoft Active Directory	Get Alert by ID
	Microsoft Exchange EWS	List Alerts
	Microsoft Exchange Server	List Alerts by Category
	Microsoft Graph Security	List Alerts by Destination
		List Alerts by Provider
		List Alerts by Severity
		List Alerts by Source IP

Integration *: Microsoft Graph Security

Alert Severity *: High

Time Range *: Relative 1 Weeks

Do not use cache ✓

Launch Enrichment Plugin

micros

Group name	Enrichment plugin	Capability
External Data	Microsoft Active Directory	Get Alert by ID
	Microsoft Exchange EWS	List Alerts
	Microsoft Exchange Server	List Alerts by Category
	Microsoft Graph Security	List Alerts by Destination
		List Alerts by Provider
		List Alerts by Severity
		List Alerts by Source IP

Integration *: Microsoft Graph Security

Alert Severity *: High

Time Range *: Absolute 2021-09-17 16:20 - 2021-10-15 16:21

Do not use cache ✓

To get the start and end of the time range in Zulu (UTC) time, you can use the following code:

```python
import time
import json

def get_time_range(time_range):
  time_range_json = json.loads(time_range)
  if time_range_json["timeRangeType"] == "RELATIVE_TIME_RANGE":
    time_range = atar.getStartEndDateFromTimeRange(time_range)
    start_s = time_range.startDate.getTime() / 1000 # in seconds
    end_s = time_range.endDate.getTime() / 1000 # in seconds

    start_time = time.strftime("%Y-%m-%dT%H:%M:%S.0Z", time.gmtime(start_s))
    end_time = time.strftime("%Y-%m-%dT%H:%M:%S.0Z", time.gmtime(end_s))
  else:
    start_time = time.strftime("%Y-%m-%dT%H:%M:%S.0Z", time.strptime(time_range_json["start"], "%Y-%m-%d %H:%M"))
    end_time = time.strftime("%Y-%m-%dT%H:%M:%S.0Z", time.strptime(time_range_json["finish"], "%Y-%m-%d %H:%M"))

  return start_time, end_time
```

Exception Handling

You can utilize Python's exception mechanism to handle errors. SOAR provides the following methods to help you handle exceptions in your code:

request.statusCode()

This method gives the HTTP status code of the response received.

```
public java.lang.Integer statusCode()
```

request.response()

This method gives the HTTP response body received.

```
public java.lang.String response()
```

request.responseHeader()

This method gives the HTTP response header received.

```
public java.lang.String responseHeader(java.lang.String header)
```

atar.failEnrichment()

This method fails the enrichment with a given message.

```
public void failEnrichment(java.lang.String message, java.lang.Throwable t)
```

Sample code snippets:

```
try:
   request = request.execute()
except:
   log("Authentication Failed with Status Code:{} Response Body:{}".format(request.statusCode(), request.response()))
   raise Exception( "Authentication Failed with Status Code:{} Response Body:{}".format(request.statusCode(), request.response()))
else:
   log("HTTP Request object gathered and returned")
   return request
```

```
if get_request.statusCode() == 200:
   return json.loads(get_request.response())
else:
   atar.failEnrichment("Unknown HTTP Response")
```

Debugging and Logging

When you test the plug-in integration, it is helpful to use logging within the plug-in code and enable HTTP debugging for REST API based integrations on SOAR. You can follow the log messages in the soar-web-app pod’s logs on ArcSight Platform.

A sample logging function you can use is:

```
def log(log_line):
    device = get_enrichment_parameter("DEVICE")
    device_name = 'Integration Name'
    logger.clear()
    label = '{}_{}'.format(device_name, device.id)
    logger.info(label, log_line)
    logger.printBuffer()
    logger.clear()
```

Enabling HTTP Debug Logging

To enable HTTP debug logs, you can edit the “**HTTPDebugLogEnabled**” parameter under the **Configuration / Parameters** menu. HTTP debug logs contain all the requests made and responses received.

Because HTTP debug logs may affect the logging performance in the long run, it is recommended to disable it after you finish.

Integration Plug-In Structure

An integration plug-in package consists of the following files:

- plugin-info.json
- <NAME>.device-meta.json
- <NAME>.SCRIPTABLE_ACTION.meta.json
- <NAME>.SCRIPTABLE_ACTION.py
- <NAME>.SCRIPTABLE_ENRICHMENT.meta.json
- <NAME>.SCRIPTABLE_ENRICHMENT.py

plugin-info.json

This is the Json configuration file to define the plug-in compatibility information. The supported parameters are:

Parameter	Description
version	SOAR script API version supported by this plug-in. Current version is **3.0** Note: The version is different than SOAR version itself.

Sample file:

```
{
  "version": "3.0"
}
```

<NAME>.device-meta.json

This is the Json configuration file to define parameter template used in plug-in integration definition. The supported parameters are:

Parameter	**Description**
name	Display name of the Integration. Example: *“name” : “Quotes App”*
url	Default value for the integration address. For SSH-based integrations only the IP address can be given. This can be changed while configuring integration on ArcSight SOAR. Example: *“url” : “https://www.example.com”*
username	Default value of username field in the credential configuration. This can be changed while configuring the Credential on ArcSight SOAR. Example: *“username”: “admin”*
password	Default value of password field in the credential configuration. This can be changed while configuring the Credential on ArcSight SOAR. You can use this field to store API keys, also. Example: *“password”: “myStr0n9.pAsSwooRd”*
key	Default value of private key field in the credential configuration. This can be changed while configuring the Credential on ArcSight SOAR. You can use this field to store API keys, also. Example: *“key” : “abcdefg1234567890”*
configurationTemplate	Default template for configuration parameters shown in integration configuration. Example: *"configurationTemplate": "#Tenant Id.\ntenant.id=11111111\n\nIntegration ID of the proxy integration to use when connecting to current integration.\n#proxy.id=123"*
ignoreSsl	Parameter for ignoring invalid SSL certificate errors in connection Example: *“ignoreSSL” : true*

Sample file:

```
{
  "name": "Quotes App",
  "url": "https://www.example.com",
  "configurationTemplate": "#Tenant Id.\ntenant.id=11111111\n",
  "password": "client_id",
  "key": "client_secret"
}
```

<NAME>.SCRIPTABLE_ACTION.meta.json

This is the Json configuration file to define plug-in ID and name references used for action script definition. The supported parameters are:

Parameter	Description
pluginId	Internal name reference for the integration/action plug-in. It is typically name given in <NAME>.device-meta.json file without space character. Example: *"pluginId" : "Quotes_App"*
pluginVisibleName	Visible name of the integration/action plug-in displayed under Customization Library. It is typically name given in <NAME>.device-meta.json file. Example: *"pluginVisibleName" : "Quotes App"*

Sample file:

```
{
  "pluginId" : "Quotes_App",
  "pluginVisibleName" : "Quotes App"
}
```

<NAME>.SCRIPTABLE_ACTION.py

This is the Python script file to define action capabilities of the integration. The supported methods are:

Function	Description
capabilities()	Method where the list of action capabilities and their parameters are defined. Listed capabilities are shown in Action launcher menu and playbook editor. **actionCapability** method is used to create the list. **scopedActionParam** method is used to get parameters from case scope. **actionParam** method is used to add parameters as user input. Basic structure of the capability definition is:

	public ActionPlug-InCapabilityDTO actionCapability (*java.lang.String* **capabilityName**, *java.lang.String* **visibleName**, *java.util.List<ActionParameterDTO>* **params**, *boolean* **avoidDuplicates**, *boolean* **isRollbackSupported**)
	
	Ref: Check **com.innoverabt.atar.actionplugins -> ScriptableActionPluginUtil** from API documentation
	
	Sample Code:
	
	`def capabilities():`
	`   return [`
	`      atar.actionCapability("DISABLE_USER", "Disable User", (`
	`         atar.scopedActionParam("USER", "User", ["USERNAME", "EMAIL_ADDRESS", "KEYWORD", "UNKNOWN"]),`
	`      ), False, True),`
	
	`      atar.actionCapability("ENABLE_USER", "Enable User", (`
	`         atar.scopedActionParam("USER", "User", ["USERNAME", "EMAIL_ADDRESS", "KEYWORD", "UNKNOWN"]),`
	`      ), False, True),`
	
	`      atar.actionCapability("ADD_USER_TO_GROUP", "Add User to Group", (`
	`         atar.scopedActionParam("USER", "User", ["USERNAME", "EMAIL_ADDRESS", "KEYWORD", "UNKNOWN"]),`
	`         atar.actionParam("GROUP_ID", "Group ID", "java.lang.String"),`
	`      ), False, False),`
	
	`   ]`

execute()	Method which handles the capability execution. Capability can be triggered by manual action request from the analyst or from the actions which are defined inside a playbook.
	
	getCapability method is used for getting the action capability called by analyst (within case UI) or playbook execution.
	
	Ref: Check **com.innoverabt.atar.service.action -> ActionExecutionRequest**.
	
	Sample Code:
	
	`def execute():`
	`   if action.capability.getCapability() == "DISABLE_USER":`
	`      return disable_user()`
	`   elif action.capability.getCapability() == "ENABLE_USER":`
	`      return enable_user()`
	`   elif action.capability.getCapability() == "ADD_USER_TO_GROUP":`
	`      return add_user_to_group()`
	`   else:`
	`      log("Unknown Capability:{}".format(action.capability.getCapability()))`
	`      return False`

rollback()	Method which handles rollback, reverting the execution of an action such as removing IP address from the firewall lists.
	The method can be triggered by analyst's manual rollback request or from the actions which are scheduled to rollback. The required methods same as **execute()**.
	
	Sample Code:
	
	`def rollback():`
	`    if action.capability.getCapability() == "ENABLE_USER":`
	`        return disable_user()`
	`    elif action.capability.getCapability() == "DISABLE_USER":`
	`        return enable_user()`
	`    else:`
	`        log("Unknown Capability:{}".format(action.capability.getCapability()))`
	`    return False`

check()	Method which handles periodic checking of integrations accessibility (also called availability). Successful check marks the integration as "Online"
	
	The scheduling of integration checks is controlled/managed by SOAR.
	
	Sample Code:
	
	`def check():`
	`    # Try to get user count to see if SOAR can login and execute queries`
	`    response = execute_http_request('GET', '/users/$count')`
	`    if response.statusCode() == 200:`
	`        return True`
	`    else:`
	`        return False`

maintain()	Method which handles the house keeping type of periodic command execution. Example usage would be:
	1. On a firewall integration, action capability adds IP addresses to object group list.
	2. However, because of network management policy, it is not allowed to initiate "install changes" command on firewall each time
	3. You can call "install changes" in maintain() block and schedule it for example to every 3 hours to meet the network management policy requirements.
	
	Sample Code:
	
	`def maintenance():`
	`    response = execute_http_request('GET', '/policy/install_changes')`
	`    if response.statusCode() == 200:`
	`        return True`
	`    else:`
	`        return False`

<NAME>.SCRIPTABLE_ENRICHMENT.meta.json

This is the Json configuration file to define plug-in ID and name references used for enrichment script definition. The supported parameters are:

Parameter	Description
pluginId	Internal name reference for the integration/enrichment plug-in. It is typically name given in <NAME>.device-meta.json file without space character. Example: *"pluginId" : "Quotes_App"*
pluginVisibleName	Visible name of the integration/enrichment plug-in displayed under Customization Library. It is typically name given in <NAME>.device-meta.json file. Example: *"pluginVisibleName" : "Quotes App"*

Sample file:

```
{
 "pluginId" : "Quotes_App",
 "pluginVisibleName" : "Quotes App"
}
```

<NAME>.SCRIPTABLE_ENRICHMENT.py

This is the Python script file to define enrichment capabilities of the integration. The supported methods are:

Function	Description
capabilities()	Method where the list of enrichment capabilities and their parameters are defined. Listed capabilities are shown in Enrichment launcher menu and playbook editor. **enrichmentCapability** method is used to create the list. **scopedEnrichmentParam** method is used to get parameters from case scope. **enrichmentParam** method is used to add parameters as user input. Basic structure of the capability definition is: public EnrichmentPlug-InCapabilityDTO enrichmentCapability (*java.lang.String* **capabilityName**, *java.lang.String* **visibleName**, *java.lang.String* **groupName**, *java.util.List*<EnrichmentPlug-InParameterDTO> **params**)

	Ref: Check **com.innoverabt.atar.enrichment -> ScriptableEnrichmentPlug-InUtil** from API documentation
	
	Sample Code:
	
	`def capabilities():`
	`   return [`
	`      atar.enrichmentCapability("LIST_USERS", "List Users", "EXTERNAL_DATA", [`
	`         atar.enrichmentParam("DEVICE", "Integration", "Device")`
	`      ]),`
	
	`      atar.enrichmentCapability("GET_USER_DETAILS", "Get User Details", "EXTERNAL_DATA", [`
	`         atar.enrichmentParam("DEVICE", "Integration", "Device"),`
	`         atar.scopedEnrichmentParam("USER", "User", "String",`
	`                        ["USERNAME", "EMAIL_ADDRESS"])`
	`      ])`
	`   ]`

enrich()	Method which handles the capability execution similar to action plug-in's execute() method. Capability can be triggered by manual enrichment request from the analyst or from the enrichments which are defined inside a playbook.
	
	getCapability method is used for getting the enrichment capability called by analyst (within case UI) or playbook execution.
	
	Ref: Check **com.innoverabt.atar.enrichment -> ScriptableEnrichmentPlug-InUtil**
	
	Sample Code:
	
	`def enrich():`
	`   if capability.getCapability() == "LIST_USERS":`
	`      return list_users()`
	`   elif capability.getCapability() == "GET_USER_DETAILS":`
	`      return get_user_details()`
	`   else:`
	`      atar.failEnrichment("Unknown capability: " + capability.getCapability())`

Prepare and Upload Integration Plug-In

Preparing integration file for uploading onto SOAR is as simple as preparing a zip archive file as:

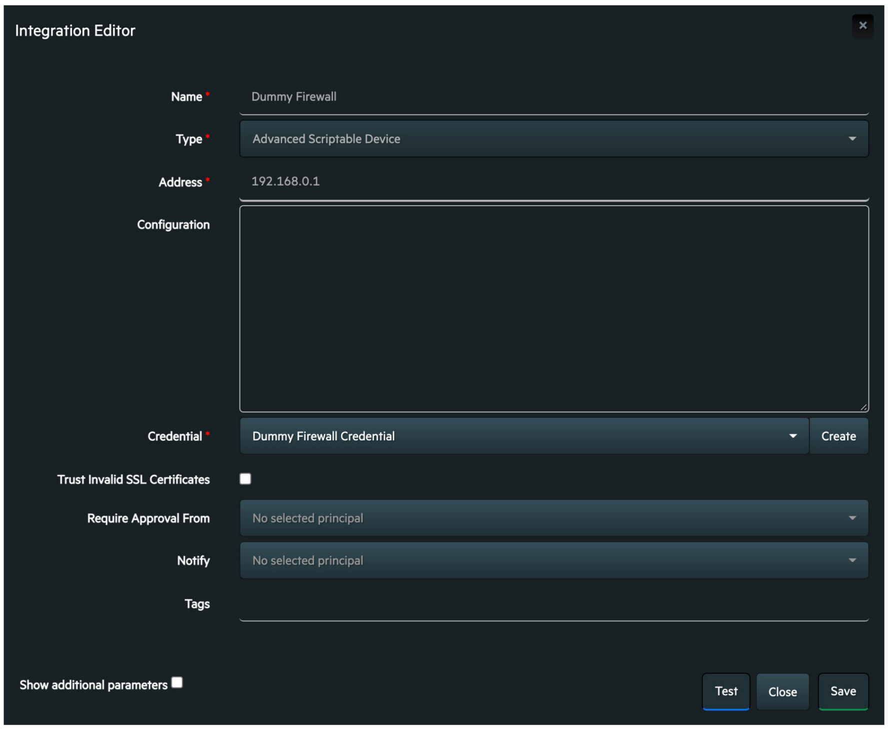
```
# zip <NAME>.zip plugin-info.json <NAME>.device-meta.json <NAME>.SCRIPTABLE_ACTION.meta.json  <NAME>.SCRIPTABLE_ACTION.py
<NAME>.SCRIPTABLE_ENRICHMENT.meta.json <NAME>.SCRIPTABLE_ENRICHMENT.py
```

To upload the plug-in file, navigate to "**Configuration / Integrations**" and upload the zip archive file you've created. ArcSight SOAR:

1. Automatically creates a credential set and populates the values as given in <NAME>.device-meta.json file. You can change the values after saving the configuration.
2. Copies plug-in code (<NAME>.SCRIPTABLE_ACTION.py and <NAME>.SCRIPTABLE_ENRICHMENT.py) under "**Configuration /Customization Library**".

Note: When you want to modify your integration plug-in code, you can change the contents of those 2 scripts.

3. Asks you to configure the integration on SOAR:



Integration Editor

Name *	Dummy Firewall
Type *	Advanced Scriptable Device
Address *	192.168.0.1
Configuration	
Credential *	Dummy Firewall Credential
Trust Invalid SSL Certificates	☐
Require Approval From	No selected principal
Notify	No selected principal
Tags	

Show additional parameters ☐

Sample SSH Plug-In

SSH-based integration plug-ins are generally used to run a series of commands on remote systems that do not support a REST API, such as network security devices or Linux/Unix machines. The aim of this plug-in is to automate the following command sequence for blocking an IP address on the firewall using SSH commands:

```
Dummy - FW> enable
Dummy - FW (enable)# config
Dummy - FW (config)# block 192.168.100.100
Dummy - FW (config)# commit-changes
Dummy - FW (config)# exit
Dummy - FW (enable)# exit
Dummy - FW> exit
Logged out.
```

plugin-info.json

```json
{
 "version": "3.0"
}
```

DummyFW.device-meta.json

```json
{
 "name": "Dummy Firewall",
 "url": "192.168.0.1",
 "username": "username",
 "password": "password"
}
```

DummyFW.SCRIPTABLE_ACTION.meta.json

```json
{
 "pluginId": "DummyFW",
 "pluginVisibleName": "Dummy Firewall"
}
```

DummyFW.SCRIPTABLE_ACTION.py

```python
from com.innoverabt.atar.service.action import ActionExecutionException
readTimeout = 10000

def capabilities():
    return [
        atar.actionCapability("BLOCK", "Block IP", (
            atar.scopedActionParam("IP_ADDRESS", "IP Address", ["NETWORK_ADDRESS"]),
        ), False, True),
    ]

def execute():
    if action.capability.getCapability() == "BLOCK":
        return execute_block_ip()
    else:
        raise ActionExecutionException('Unknown capability: ' + action.capability.getCapability(), False)


def rollback():
    log("Unknown Capability:{}".format(action.capability.getCapability()))
    return False


def check():
    connection = atar.connectSSH()
    result = connection.readUntil('Dummy - FW> ', readTimeout)
    log(result)
    if "Dummy - FW> " in str(result):
        return True

def maintain():
    return True

def get_action_parameter(parameter_name):
    return action.getParameter(parameter_name)
```

```
def execute_block_ip():
    ip_address = get_action_parameter('IP_ADDRESS')
    connection = atar.connectSSH()
    connection.readUntil('Dummy - FW> ', readTimeout)
    connection.sendLine('enable')
    connection.readUntil('Dummy - FW (enable)# ', readTimeout)
    connection.sendLine('config')
    connection.readUntil('Dummy - FW (config)# ', readTimeout)
    connection.sendLine('block ' + ip_address)
    connection.readUntil('Dummy - FW (config)# ', readTimeout)
    result = connection.sendLine('commit-changes')
    connection.readUntil('Dummy - FW (config)# ', readTimeout)
    connection.sendLine('exit')
    connection.readUntil('Dummy - FW (enable)# ', readTimeout)
    connection.sendLine('exit')
    connection.readUntil('Dummy - FW> ', readTimeout)
    connection.sendLine('exit')

    if str(result).find('Error') != -1 :
        logger.error(logger_label, str(result).replace('%', '-'))
        logger.printBuffer()
        return False

    return True

def log(log_line):
    device_name = 'DummyFW'
    logger.clear()
    label = '{}_{}'.format(device_name, device.id)
    logger.info(label, log_line)
    logger.printBuffer()
    logger.clear()
```

DummyFW.SCRIPTABLE_ENRICHMENT.meta.json

```
{
  "pluginId": "DummyFW",
  "pluginVisibleName": "Dummy Firewall"
}
```

DummyFW.SCRIPTABLE_ENRICHMENT.py

```
import datetime

def capabilities():
    return [
    ]

def enrich():

def get_enrichment_parameter(parameter_name):
    return atar.getEnrichmentParam(parameter_name)

def log(log_line):
    device = get_enrichment_parameter("DEVICE")
    device_name = 'DummyFW'
    logger.clear()
    label = '{}_{}'.format(device_name, device.id)
    logger.info(label, log_line)
    logger.printBuffer()
    logger.clear()
```

Sample HTTP/REST-API Plug-In

Most security platforms and products provide a REST API for integrating with other systems, and majority of the SOAR plug-ins utilize REST API methods provided by remote systems. The aim of this plug-in is to showcase action and enrichment capabilities over HTTP requests.

plugin-info.json

```
{
  "version": "3.0"
}
```

MyHTTPService.device-meta.json

```

{
  "name": "My HTTP Service",
  "url": "https://www.example.com",
  "configurationTemplate": "# Integration ID of the proxy integration to use when connecting to current integration.\n# If not provided,
ArcSight SOAR will try to use a direct connection.\n#proxy.id=123",
  "key": "api_token"
}
```

MyHTTPService.SCRIPTABLE_ACTION.meta.json

```
{
  "pluginId": "MyHTTPService",
  "pluginVisibleName": "My HTTP Service"
}
```

MyHTTPService.SCRIPTABLE_ACTION.py

```

import json

def capabilities():
    return [
        atar.actionCapability("CLEAR_USER_SESSIONS", "Clear User Sessions", (
            atar.scopedActionParam("USER", "Username", ["USERNAME", "EMAIL_ADDRESS"]),
        ), False, False),
    ]


def execute():
    if action.capability.getCapability() == "CLEAR_USER_SESSIONS":
        return clear_user_sessions()
    else:
        raise Exception('Unknown capability: ' + action.capability.getCapability())


def rollback():
    log("Unknown rollback capability:{}".format(action.capability.getCapability()))
    return False


def check():
    endpoint = '/api/v1/users/me'
    query_headers = query_parameters = []
    response = execute_http_request('GET', None, None, None, endpoint).response()
    if 'id' in json.loads(response):
        return True
    return False
```

```
def maintain():
    return False


def get_user_id(scope_item):
    endpoint = '/api/v1/users/{}'.format(scope_item)
    response = execute_http_request('GET', None, None, None, endpoint).response()

    if 'id' in json.loads(response):
        return json.loads(response)["id"]
    else:
        raise Exception('User not found: ', json.loads(response))


def clear_user_sessions():
    scope_item = get_action_parameter("USER")
    user_id = get_user_id(scope_item)
    endpoint = '/api/v1/users/{}/sessions'.format(user_id)
    response = execute_http_request('DELETE', None, None, None, endpoint)
    return response.isSuccessful()


def execute_http_request(method, request_headers=None, query_parameters=None, request_body=None, endpoint=None):

    log("Starting {} request to {} endpoint with query parameters {} with additional headers {}".format(method, endpoint, query_parameters,
request_headers))

    request = atar.http() \
        .proxy(device) \
        .method(method) \
        .customHeaderAuthenticationProviderConfigurer(device.credential) \
        .header("Authorization", AuthenticationCapableField.PRIVATE_KEY) \
        .configure() \
        .urlConfigurer(device) \
        .withPath(endpoint)

    if query_parameters:
        for key, value in query_parameters.iteritems():
            request = request \
                .withQueryParam(str(key), str(value))

    if request_body:
        request \
            .configure() \
            .body(request_body)

    if device.ignoreSSLCertificationErrors:
        request \
            .configure() \
            .ignoreCertErrors()

    if request_headers:
        for key, value in request_headers.iteritems():
            request \
                .configure() \
                .header(str(key), str(value))

    request = request.configure()

    try:
        request = request.execute()

    except:
        log("HTTP Request Failed with Status Code:{}".format(request.statusCode()))
        raise Exception("HTTP Request Failed with Status Code:{} Response Body:{}".format(request.statusCode(), request.response()))
    else:
        log("HTTP Response gathered and returned")
        return request

def get_action_parameter(parameter_name):
    return action.getParameter(parameter_name)

def log(log_line):
    device_name = device.getName()
    logger.clear()
    label = '{}_{}'.format(device_name, device.id)
    logger.info(label, log_line)
    logger.printBuffer()
    logger.clear()
```

MyHTTPService.SCRIPTABLE_ENRICHMENT.meta.json

```
{
  "pluginId": "MyHTTPService",
  "pluginVisibleName": "My HTTP Service"
}
```

MyHTTPService.SCRIPTABLE_ENRICHMENT.py

```
import json

def capabilities():
    return [
        atar.enrichmentCapability("GET_USER", "Get User Details",
                            "EXTERNAL_DATA", [
                                atar.enrichmentParam("DEVICE", "Integration", "Device"),
                                atar.scopedEnrichmentParam("USER", "User", "String",
                                                ["EMAIL_ADDRESS", "USERNAME"])
                            ])
    ]


def enrich():
    if capability.getCapability() == "GET_USER":
        return get_user()
    else:
        atar.failEnrichment("Unknown capability: " + capability.getCapability())


def get_user():
    scope_item = get_scoped_enrichment_parameter_value("USER")
    endpoint = '/api/v1/users/{}'.format(scope_item)
    enrichment_result = execute_http_request('GET', None, None, None, endpoint).response()
    pretty_response = prepare_get_user_pretty_response(enrichment_result)
    return atar.enrichmentData(capability.getCapability(), "JSON", json.dumps(pretty_response))


def prepare_get_user_pretty_response(enrichment_result):
    user = json.loads(enrichment_result)

    full_name = user["profile"]["firstName"] + " " + user["profile"]["lastName"]
    login = user["profile"]["login"]
    email = user["profile"]["email"]
    mobile = user["profile"]["mobilePhone"]
    created = user["created"]
    last_updated = user["lastUpdated"]
    last_login = user["lastLogin"]
    status = user["status"]
    id = user["id"]
    pretty_response = {
        "columns": ["Key", "Value"],
        "data": [["Full Name", full_name],
                 ["Login", login],
                 ["Email", email],
                 ["Mobile", mobile],
                 ["ID", id],
                 ["Status", status],
                 ["Last Login", last_login],
                 ["Created", created],
                 ["Last Updated", last_updated]
                 ]
    }
    return pretty_response


def execute_http_request(method, request_headers=None, query_parameters=None, request_body=None, endpoint=None):

    log("Starting {} request to {} endpoint with query parameters {} with additional headers {}".format(method, endpoint, query_parameters,
request_headers))

    device = get_enrichment_parameter("DEVICE")
    request = atar.http() \
        .proxy(device) \
        .method(method) \
        .customHeaderAuthenticationProviderConfigurer(device.credential) \
        .header("Authorization", AuthenticationCapableField.PRIVATE_KEY) \
        .configure() \
        .urlConfigurer(device) \
        .withPath(endpoint)

    if query_parameters:
        for key, value in query_parameters.iteritems():
            request = request \
                .withQueryParam(str(key), str(value))

    if request_body:
        request \
            .configure() \
            .body(request_body)

    if device.ignoreSSLCertificationErrors:
        request \
            .configure() \
            .ignoreCertErrors()

    if request_headers:
        for key, value in request_headers.iteritems():
            request \
                .configure() \
                .header(str(key), str(value))

    request = request.configure()

    try:
        request = request.execute()

    except:
        atar.failEnrichment(
            "HTTP Request Failed with Status Code:{} with Response Body:{}".format(
                request.statusCode(), request.response()))
    else:
        log("HTTP Response gathered and returned")
        return request

def get_scoped_enrichment_parameter_value(parameter_name):
    return atar.getEnrichmentParam(parameter_name)[0].getScopeItem().getValue()
```

```
def get_enrichment_parameter(parameter_name):
   return atar.getEnrichmentParam(parameter_name)


def log(log_line):
   device = get_enrichment_parameter("DEVICE")
   device_name = device.getName()
   logger.clear()
   label = '{}_{}'.format(device_name, device.id)
   logger.info(label, log_line)
   logger.printBuffer()
   logger.clear()
```

About OpenText

OpenText enables the digital world, creating a better way for organizations to work with information, on-premises or in the cloud. For more information about OpenText (NASDAQ/TSX: OTEX), visit opentext.com.

Connect with us:

OpenText CEO Mark Barrenechea's blog

Twitter | LinkedIn

Legal Notices

Open Text Corporation

275 Frank Tompa Drive, Waterloo, Ontario, Canada, N2L 0A1

Copyright Notice