Splunk Add-on for Defender

Introduction

This Add-on was designed to handle a REST API based interaction with the defender service, in SPL with two interactive generating custom commands:

defenderstatus:

```
| defenderstatus computername=<computer name target>
```

defenderscan:

```
| defenderscan computername=<computer name target> fullscan=<True|False>
```

It was redesigned and repackaged for the purpose of the Splunk Cloud migration, with the following evolutions:

- The custom commands need to be executed on Splunk Cloud, but relayed to another Splunk instance running on-premise
- The Splunk Enterprise on-premise instance (a Heavy Forwarder) then receives the incoming query, and performs the action effectively
- In addition, a least privileges approach was implemented to avoid having to grant some security concerning Splunk capabilities
- To achieve this, we use REST API endpoints associated with custom capabilities, as well as a RBAC approach to grant capabilities as needed
- Finally, the application was completely redesigned to rely on the Splunk UCC framework

References

Splunk UCC framework and ucc-gen:

- https://splunk.github.io/addonfactory-ucc-generator
- https://github.com/splunk/addonfactory-ucc-generator

Requirements

To build the package and maintain evolutions over time, the following requirements must be met:

- A Linux or Mac OS based development environment
- Python3 must be available
- pip3 must be installed
- The ucc-gen command must be installed
- Internet connectivity for pip to retrieves packages as needed

Installing ucc-gen

Installing ucc-gen is straightforward:

```
python3 -m "pip3 install splunk-add-on-ucc-framework"
```

Application structure and build

The application development structure is the following:

```
build
output
package
globalConfig.json
version.json
```

build and output directories

The build directory contains a simple build.py script which can be executed to orchestrate the generation or the final package.

To generate a new build:

```
cd build
python3 build.py
```

To generate a new build and retain the extracted directory in the output directory:

```
cd build
python3 build.py --keep
```

To update the application version, you need to update the version.json which is then taken into account automatically:

```
{
"version": "1.0.4",
"appID": "splunk_start_defender"
}
```

The Python build.py script does the following:

- It retrieves the current application version in the file version.json
- It prepares and cleans if necessary previous execution and artefacts in the output directory
- It calls the ucc-gen command including the version requirement
- It packages the generated directory and content into a tgz file
- It writes a version.txt, a build.txt and a release-sha256.txt containing the unique sha256 sum of the generated tgz file

The tgz file represents the final Splunk application, which should be submitted for deployment where needed.

The content of the output directory should be excluded from the Git repository, which is done via a .gitignore:

```
# output
output/*
!output/README.txt
```

As well, some Python related artefacts should be excluded in the build directory:

```
# Py build
build/libs/__pycache__
```

package directory

The `package` directory is the Splunk application structure content, any directory and its content will be picked up automatically by ucc-gen and included into the application release.

It should contain at the minimum:

```
bin
default
lib
static
app.manifest
```

bin directory The bin directory would contain executable scripts and binaries, in the case of this Addon, we have the following:

`get_defender_status.py`

- The Splunk generating custom command corresponding to the the SPL command `| defenderstatus`

`splunk_start_manager_rest_handler.py`

- This Python script contains the REST endpoints which are going to be exposed by splunkd

`start_defender.py`

- The Splunk generating custom command corresponding to the the SPL command `| defenderscan`

default directory In default, we have the following configuration files:

`app.conf`

The usual Splunk app.conf, note that it **not needed to maintain versioning manually**, the ucc-gen will do it automatically.

Therefore, when creating a new release, updating the app.conf is not necessary.

`authorize.conf`

We will describe further the content of this file in the next section regarding the least privileges approach, but in short it contains:

- The custom capability
- The role definitions which inherit and enable the capability as needed

`commands.conf`

Defines the custom commands and the associated Python files.

`props.conf`

In the context of this Add-on, it is important to highlight that Python scripts (custom commands and API endpoints) generate logs effectively.

The best practice is to manage explicitly the source ingestion definition to allow a proper parsing at index time and search time.

`restmap.conf`

The restmap.conf file contains the definition for the API endpoints to be taken into account by splunk and exposed by splunkd.

ucc-gen generates itself a first version of the restmap.conf configuration file as it does need API endpoints too.

These API endpoints are called when performing the configuration of the Add-on via the configuration page ucc-gen generates, based on the `globalConfig.json` file.

`searchbnf.conf`

Describes the custom command options and syntax, allowing autocompletion in Splunk.

`server.conf`

This file is needed in our context as we manage via ucc-gen objects like:

- preferences (called settings)
- accounts (an account references credentials, secrets and per account settings)

ucc-gen by itself does not generate (yet) the SHC replication stanza, this is required for the settings to be properly replicated amongst the SHC members.

Without these settings, the configuration would work on a given SHC node, but not on others.

`splunk_start_defender_settings.conf`

In this file, we can preset default values for the options we made available to the Add-on via ucc-gen and the globalConfig.json.

For instance, we can preset the logging level.

`web.conf`

This file is required here to expose the REST API endpoints via Splunk Web too.

lib directory `requirements.txt`

This file is a pypi file format which is used by ucc-gen to automatically retrieve the required Python libraries for the Add-on to operate.

As a minimal version, it would contain:

```
splunktaucclib>=5.0.4
```

You can add any additional libraries that would be required to be imported by a Python logic.

Note that some libraries can be problematic if these include compiled binaries for a certain type of processor architecture.

In such a case, you can as well include manually the libraries in the lib directory, which will be included in the final package automatically.

`lib_splunk_start_defender.py`

In this Python file, we store various utility Python functions which will be imported by the custom commands, as well as by the REST API endpoints themselves.

`rest_handler.py`

This Python file is a REST API wrapper, it is used for various purposes to facilitate the management of our REST API endpoints, such as orchestrating the extraction of Metadata and organising the endpoints structure and output format.

Least privileges approach

A typical issue with Splunk custom command and other types of advanced development techniques for Splunk relies on the fact that these require to grant users with high privileges capabilities.

In between, the following capability are problematic:

- `admin_allobjects`
- `list_settings`
- `list_storage_password`

The worst is with no doubts the last, which allows a user with some levels of knowledge to expose any Splunk credentials that are stored in the Splunk secure credentials store, and this in a clear text format. (!)

To avoid this issue, we can leverage API REST endpoints with escalated system wide privileges, associating these with capabilities and RBAC for a full and secured control compliant with best security practices:

- When the user requests the execution of the custom command, a Python function is called, which itself is stored in a Python file stored in the `lib` directory.
- The Python function performs a REST call locally, using the splunkd_uri provided as part of the self metadata by Splunk
- The endpoints can allow be accessed by users which own the right `capability`
- If the user owns the capability, the REST API endpoints executes and establishes a Splunk Python SDK service using the system wide token.
- The REST API returns required information, settings and credentials in a programmatic manner back to the requesting Python function
- The rest of the logic can be executed as needed, without the need from any further privileges

Using these techniques, users can be granted the right to use the custom commands, even with very low and limited privileges.

Exposing the REST API endpoint with passSystemAuth

The first requirement is to expose REST API endpoints with the `passSystemAuth` option and associates this with one or more capabilities:

```
[script:trackme_rest_handler_secmanager]
match                   = /splunk_start_defender/manager
script                  = splunk_start_manager_rest_handler.py
scripttype              = persist
handler                 = splunk_start_manager_rest_handler.SplunkStartDefender_v1
output_modes            = json
passPayload             = true
passSystemAuth          = true
capability              = splunkstartdefender
python.version = python3
```

When exposed this way, the REST API endpoint will automatically benefit from a special authentication token, in addition with the user based session token:

- `system_authtoken`

This special special authentication token is parsed and made available via the `rest_handler.py` Python wrapper:

```
#
# system auth
#
```

```
# If passSystemAuth = True, add system_authtoken
try:
    system_authtoken = args['system_authtoken']
except Exception as e:
    system_authtoken = None
```

When establishing the Python SDK service, the REST API endpoints functions stored in `splunk_start_manager_rest_handler.py` (in bin directory) use the system level token, rather than the user level token requesting the action:

```
# Get service
service = client.connect(
    owner="nobody",
    app="splunk_start_defender",
    port=request_info.connection_listening_port,
    token=request_info.system_authtoken
)
```

This way, the action is performed with system level privileges rather than limited user privileges.

Lastly, the endpoint is protected via capability:

```
capability            = splunkstartdefender
```

This capability is defined in the file `authorize.conf`, granted to a builtin role which is part of the application.

As well, this capability needs to be explicitly enabled for the `sc_admin` role for Splunk Cloud to allow granting this role:

```
# authorize.conf

#
# capabilities
#

# only roles with this capability can access to the splunk_start_defender API endpoints, and
[capability::splunkstartdefender]

#
# roles
#

# users members of this role, or roles inheriting this roles can use the app

[role_splunk_start_defender]

# Minimal import
importRoles = user
```

```
# capabilities
splunkstartdefender = enabled

# This is required for Splunk Cloud
[role_sc_admin]
splunkstartdefender = enabled
```

Finally, the custom commands calls a Python function which itself performs the REST call locally to splunkd, using the Python requests module:

```
# get account
try:
    # get account
    account_conf = splunk_start_defender_get_account(self._metadata.searchinfo.session_key,

except Exception as e:
    raise Exception(str(e))
```

This function, stored in `lib/lib_splunk_start_defender.py` performs:

```
# get system wide conf with least privilege approach
def splunk_start_defender_get_conf(session_key, splunkd_uri):
    """
    Retrieve settings.
    """

    # Ensure splunkd_uri starts with "https://"
    if not splunkd_uri.startswith("https://"):
        splunkd_uri = f"https://{splunkd_uri}"

    # Build header and target URL
    headers = CaseInsensitiveDict()
    headers["Authorization"] = f"Splunk {session_key}"
    target_url = f"{splunkd_uri}/services/splunk_start_defender/manager/splunk_start_defende

    # Create a requests session for better performance
    session = requests.Session()
    session.headers.update(headers)

    try:
        # Use a context manager to handle the request
        with session.get(target_url, verify=False) as response:
            if response.ok:
                logging.debug(f"Success retrieving conf, data=\"{response}\"")
                response_json = response.json()
                return response_json
            else:
                error_message = f"Failed to retrieve conf, status_code={response.status_code
                logging.error(error_message)
                raise Exception(error_message)

        except Exception as e:
            error_message = f"Failed to retrieve conf, exception=\"{str(e)}\""
            logging.error(error_message)
            raise Exception(error_message)
```

The same technique applies to other endpoints, for instance to retrieve the credentials (tokens) stored in the Splunk secure credential store, and make these available in a secured fashion to the rest of the Python process.

Application behaviour: relaying Splunk Cloud originating request to a relay server

A key of the Add-on behaviour relies on the fact that these API actions need to operate from a SAS based service (Splunk Cloud) and be executed against an on-premise service that cannot be made available from Splunk Cloud directly.

For this to operate, the following workflow was developed:

- On Splunk Cloud, the application is deployed accordingly
- A configuration item, `instance_role`, instructs the Addon where it operates, valid options are `splunk_cloud`, `splunk_relay`
- The Addon is deployed to an on-premise Splunk Enterprise Heavy Forwarder, which can reach the Defender API service
- The outgoing traffic from Splunk Cloud to the on-premise Heavy Forwarder API service is allow listed via ACS (HTTPS over 8089)
- The Addon on the on-premise side is configured with `instance_role` as `splunk_relay`

When the custom command is executed:

- On Splunk Cloud, a granted user executed the defender custom command
- As it is operating in `instance_role=splunk_cloud`, the custom command performs a REST call to an API endpoint part of the application on the Splunk Heavy Forwarder
- The Splunk Enterprise on-premise instance running the Addon with `instance_role=splunk_relay` receives the request on its listening endpoint
- It operates the REST API call to the Defender service, and returns the response in return
- The custom command executed on Splunk Cloud receives the response and returns it to the requester user

Authentication and privileges:

On Splunk Cloud:

- As explained in the previous section, the requesting user can only execute the custom command successfully if he owns the required capability
- If granted, the Python logic retrieves the stored in the account for `relay_url` and `relay_token`
- The `relay_url` represents the access to the Heavy Forwarder endpoint, in the format `https://<ip_address|fqdn>:<port>`
- The `relay_token` is a Splunk bearer token created on the Splunk on-premise Heavy Forwarder, which itself owns the required capability, the token value is configured on the Splunk Cloud account configuration

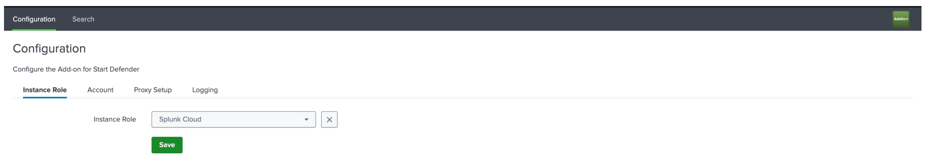
On the Splunk Relay:

- The Splunk relay, the On-premise Heavy Forwarder, has the associated account configured in the Add-on with `circ_url` and `circ_token`
- `circ_url` is the target Defender service in the format `https://<ip_address|fqdn>:<port>`
- `circ_token` is a bearer token providing access to the Defender API service
- When the endpoint is executed on the Splunk relay, upon a request originating from the custom command executed on Splunk Cloud, the endpoint retrieves the `circ_url` and `circ_token` equally and execute the requested action
- Finally, it returns the HTTP response which is interpreted by the Splunk Cloud side Python logic and returned to the user

Configuration

On Splunk Cloud

Once the Add-on has been installed in Splunk Cloud, access to the configuration page:



Configure the Add-on for Start Defender

Instance Role: Splunk Cloud

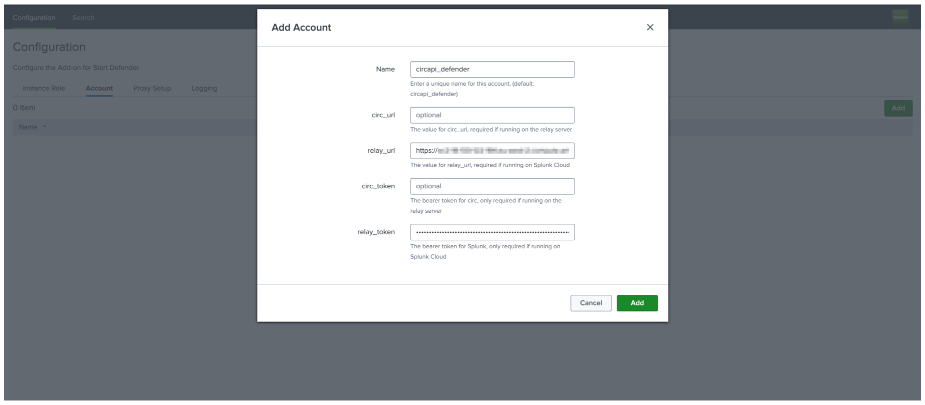
Figure 1: screen1

In Splunk Cloud, the instance role must be set to “Splunk Cloud”. (this is the default behaviour for the Addon)

Then, access to the account configuration, an account should be created as:

For the Splunk Cloud configuration, the following items are required:

- `account name`: the custom command expects a default of circapi_defender (if a different account name is configured, the account must be explicitly mentioned while calling the custom command)
- `relay_url`: The Splunk relay, in the format `https://fqdn:port`
- `relay_token`: The Splunk bearer token created on the Splunk relay, the account must have the capability `splunkstartdefender`



Name: circapi_defender
Enter a unique name for this account. (default: circapi_defender)

circ_url: optional
The value for circ_url, required if running on the relay server

relay_url: https://[redacted]
The value for relay_url, required if running on Splunk Cloud

circ_token: optional
The bearer token for circ, only required if running on the relay server

relay_token: ••
The bearer token for Splunk, only required if running on Splunk Cloud

Figure 2: screen2

On the Splunk Relay

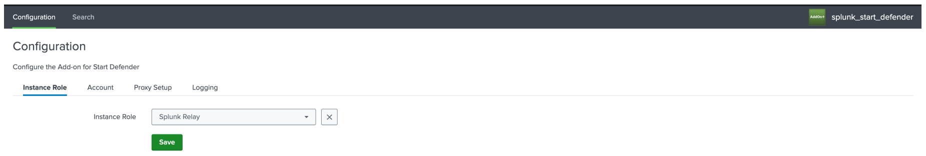
Once installed, on the Splunk Relay, the instance role must be set to "Splunk Relay":

Then, the corresponding account must be created, under the same name as created on Splunk Cloud, the following information are required:

- `account name`: the custom command expects a default of circapi_defender (if a different account name is configured, the account must be explicitly mentioned while calling the custom command)
- `circ_url`: The Defender API url, in the format `https://fqdn:port`
- `circ_token`: The Defender API bearer token

Testing the custom command

Once the configuration has been performed, assuming the connectivity between Splunk Cloud and the Splunk Relay (HTTPS/8089) is operational, and other

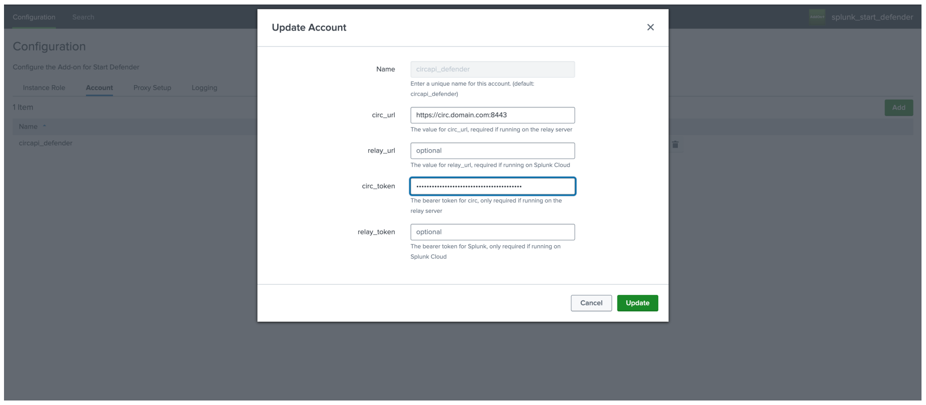


Configuration

Configure the Add-on for Start Defender

Instance Role: Splunk Relay

Figure 3: screen3



Name: circapi_defender

Enter a unique name for this account. (default: circapi_defender)

circ_url: https://circ.domain.com:8443

The value for circ_url, required if running on the relay server

relay_url: optional

The value for relay_url, required if running on Splunk Cloud

circ_token: ••

The bearer token for circ, only required if running on the relay server

relay_token: optional

The bearer token for Splunk, only required if running on Splunk Cloud

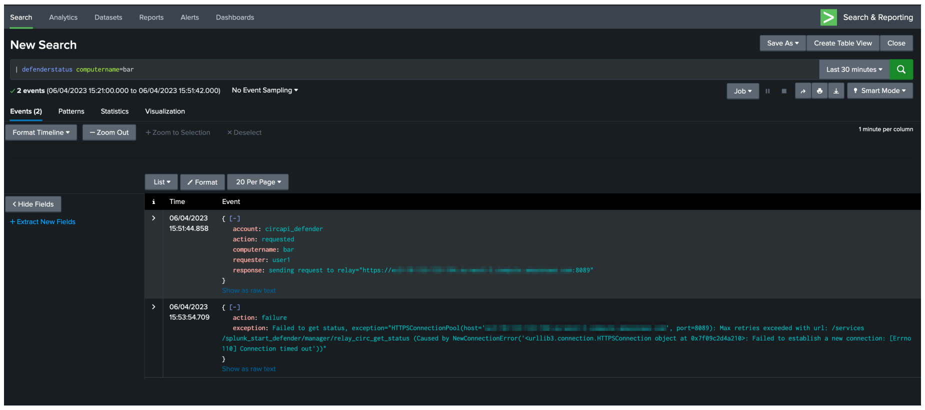
Figure 4: screen4

settings are valid, the operation will be performed as expected.

In the following example, and for the purposes of this documentation, there is real Defender API, however we can observe the following:

- The custom command call was accepted, our user has the permissions to access the associated endpoints
- The connectivity is operational, we can view the response from the Splunk Relay which attempted to perform the operation
- We can observe the proper handling of the exception, given that we tried to reach a non existing domain for Defender

Shall we fail to connect from Splunk Cloud to the Splunk Relay:



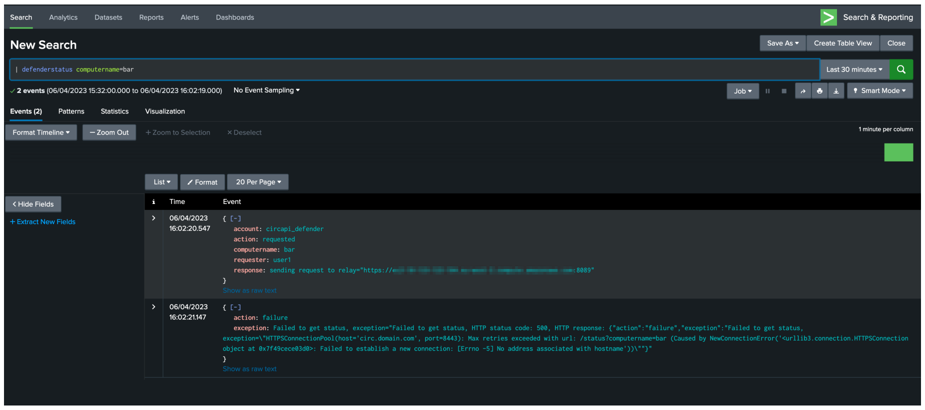
```
| defenderstatus computername=bar
```

✓ **2 events** (06/04/2023 15:21:00.000 to 06/04/2023 15:51:42.000) No Event Sampling

```
i  Time          Event
>  06/04/2023    { [-]
   15:51:44.858    account: circapi_defender
                   action: requested
                   computername: bar
                   requester: user1
                   response: sending request to relay="https://[redacted]:8089"
                 }
>  06/04/2023    { [-]
   15:53:54.709    action: failure
                   exception: Failed to get status, exception="HTTPSConnectionPool(host='[redacted]', port=8089): Max retries exceeded with url: /services
                 /splunk_start_defender/manager/relay_circ_get_status (Caused by NewConnectionError('<urllib3.connection.HTTPSConnection object at 0x7f09c2d4a210>: Failed to establish a new connection: [Errno
                 110] Connection timed out'))"
                 }
```

Figure 5: screen5

Shall the Splunk Relay be failing to reach the Defender API:



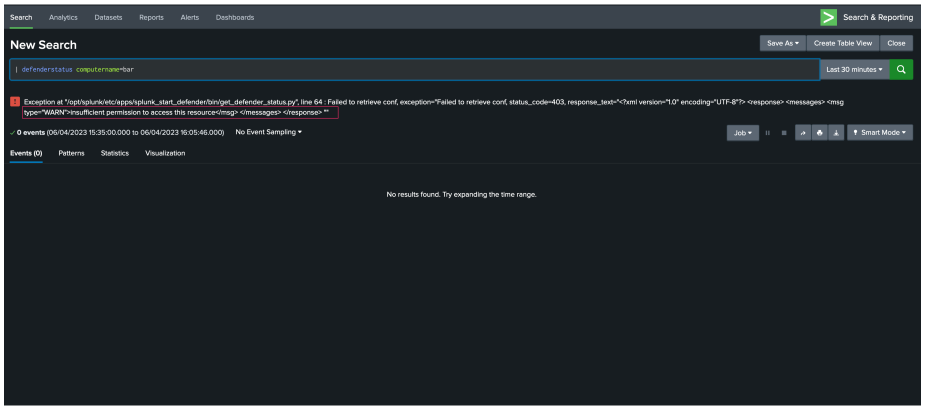
```
| defenderstatus computername=bar
```

✓ **2 events** (06/04/2023 15:32:00.000 to 06/04/2023 16:02:19.000) No Event Sampling

```
i  Time          Event
>  06/04/2023    { [-]
   16:02:20.547    account: circapi_defender
                   action: requested
                   computername: bar
                   requester: user1
                   response: sending request to relay="https://[redacted]:8089"
                 }
>  06/04/2023    { [-]
   16:02:21.147    action: failure
                   exception: Failed to get status, exception="Failed to get status, HTTP status code: 500, HTTP response: {"action":"failure","exception":"Failed to get status,
                 exception=\"HTTPSConnectionPool(host='circ.domain.com', port=8443): Max retries exceeded with url: /status?computername=bar (Caused by NewConnectionError('<urllib3.connection.HTTPSConnection
                 object at 0x7f49cece03d0>: Failed to establish a new connection: [Errno -5] No address associated with hostname'))\""}"
                 }
```

Figure 6: screen6

If the user is not allowed to access to the REST API (capability not granted):



```
| defenderstatus computername=bar
```

Exception at "/opt/splunk/etc/apps/splunk_start_defender/bin/get_defender_status.py", line 64 : Failed to retrieve conf, exception="Failed to retrieve conf, status_code=403, response_text="<?xml version="1.0" encoding="UTF-8"?> <response> <messages> <msg type="WARN">insufficient permission to access this resource</msg> </messages> </response> ""

✓ 0 events (06/04/2023 15:35:00.000 to 06/04/2023 16:05:46.000) No Event Sampling

No results found. Try expanding the time range.

Figure 7: screen7

Logging and troubleshooting

The Add-on is designed to use best logging practices, 2 log files are generated in `$SPLUNK_HOME/var/log/splunk/` and automatically indexed in the corresponding Splunk environment.

This first relies on the `props.conf` definition part of the application:

```
[source::...splunk_start_manager_rest_api.log]
sourcetype = splunk_start_defender:rest
SHOULD_LINEMERGE=false
LINE_BREAKER=([\r\n]+)\d{4}-\d{2}-\d{2}\s\d{2}:\d{2}:\d{2}
CHARSET=UTF-8
TIME_PREFIX=^
TIME_FORMAT=%Y-%m-%d %H:%M:%S,%3N
TRUNCATE=0

[source::...splunk_start_defender.log]
sourcetype = splunk_start_defender:commands
SHOULD_LINEMERGE=false
LINE_BREAKER=([\r\n]+)\d{4}-\d{2}-\d{2}\s\d{2}:\d{2}:\d{2}
CHARSET=UTF-8
TIME_PREFIX=^
TIME_FORMAT=%Y-%m-%d %H:%M:%S,%3N
TRUNCATE=0
```

When running on Splunk Cloud, the logs can be searched using:

```
index=_internal sourcetype=splunk_start_defender:commands
```

When running on the Splunk relay, the logs be searched using:

```
index=_internal sourcetype=splunk_start_defender:rest
```

In both cases, the Add-on carefully logs:

- The requester information
- The detailed requested operation
- The API response, either from the Splunk relay (which itself contains the API Defender response) or from the Defender API

This can be found in the custom commands and REST API endpoints, such as:

```
# run call
yield_record = {
    '_time': time.time(),
    '_raw': {
        'action': 'requested',
        'requester': self._metadata.searchinfo.username,
        'computername': self.computername,
        'account': self.account,
        'response': 'sending request to relay=\"{}\"'.format(relay_url),
    },
}
logging.info(json.dumps(yield_record, indent=2))
yield yield_record
```

Observing unauthorised response:

Should a REST query be made against the endpoints without owning the capability, the following answer will be raised:

Example:

```
curl -k -H "Authorization: Splunk $token" -X GET https://$mytarget:8089/services/splunk_star

<?xml version="1.0" encoding="UTF-8"?>
<response>
<messages>
    <msg type="WARN">insufficient permission to access this resource</msg>
</messages>
</response>
```

If properly allowed, we would have got:

```
curl -k -H "Authorization: Splunk $token" -X GET https://$mytarget:8089/services/splunk_star

{
"logging": {
    "disabled": "0",
    "eai:appName": "splunk_start_defender",
    "eai:userName": "nobody",
    "loglevel": "INFO"
},
"proxy": {
    "disabled": "0",
    "eai:appName": "splunk_start_defender",
    "eai:userName": "nobody"
},
"role": {
    "disabled": "0",
    "eai:appName": "splunk_start_defender",
    "eai:userName": "nobody",
    "instance_role": "splunk_relay"
}
}
```

Endpoints reference

The following endpoints are available as part of this application:

get conf

- type: GET
- purpose: returns all Add-on configuration items and values in a JSON structured dictionary

curl example:

```
curl -k -H "Authorization: Splunk $token" -X GET https://$mytarget:8089/services/splunk_sta
```

get account

- type: POST
- purpose: returns the account configuration, depending on the `instance_role`, to be used by the custom commands and API endpoints in a programmatic fashion

curl example:

```
curl -k -H "Authorization: Splunk $token" -X POST https://$mytarget:8089/services/splunk_sta
```

run Defender get status action (relay)

- type: POST
- purpose: runs the Defender get status action in relay mode

curl example:

```
curl -k -H "Authorization: Splunk $token" -X POST https://$mytarget:8089/services/splunk_sta
```

run Defender scan action (relay)

- type: POST
- purpose: runs the Defender scan action in relay mode

curl example:

```
curl -k -H "Authorization: Splunk $token" -X POST https://$mytarget:8089/services/splunk_sta
```