H SCAPL Plugin

H.1 Introduction

Security assessment for Computing APL (SCAPL) is an open-source framework with a user interface for automating particular tasks in the scope of security assessment of software products in support of the Software Approved Products List Automation Framework (SAPLAFv1). Its main goal is to make users spare as much time as possible in researching the required information for efficiently assessing products and to be able to make a Software Approved Products List (APL).

Data Scheme Its design relies on a **flexible** data scheme allowing to define what is called a **Data Item (DI)**. Each DI is related to an **orchestrable** task (either manual, interactive, systematized or automated) relying on an **external project** that provides a specific result for the assessment process.

Plugin Architecture In order to be implemented, such a **DI** must be associated to a **plugin** that is to be included in SCAPL's infrastructure such that it can be referenced with a command inside the item.

Purpose of this Document This document shows an example of establishment of such a plugin implementing a particular DI, in the Search Engine component of SCAPL (*scapl-search*), presented hereafter.

Data Item Known Vulnerabilities
Component *scapl-search* (SCAPL's Search Engine)
Plugin CVE-Search
Purpose Search for CVE's[1] of a product given its CPE[2]

H.2 External Project Analysis

This section presents the main information about CVE-Search so that it can be figured out how it works and how it could be plugged to SCAPL.

[1] Common Vulnerabilities and Exposures
[2] Common Platform Enumeration

GitHub Reference `https://github.com/cve-search/cve-search`

H.2.1 System Requirements

CVE-Search *is a tool to import CVE and CPE into a MongoDB to facilitate search and processing of CVEs*, is written in Python and uses various Python packages such as PyMongo (client for MongoDB), Flask (Web server), Redis (caching), ...

Hence, it requires :

- Ubuntu (natively owning the Python interpreter and its package manager, PIP)
- MongoDB (a NoSQL database management system)
- Git (in order to clone the project from GitHub)

The Git project contains `requirements.txt`, a text file containing the list of the required Python packages that can easily be installed with PIP. The installation procedure is straightforward and available on the main page of the project on GitHub. The update procedure for feeding MongoDB from various sources is also clearly explained.

H.2.2 Structure

The structure of the project can be consulted on GitHub but the main folders are the followings :

- `bin` : This contains the tools for querying MongoDB, owning the data of interest.
- `sbin` : This contains some tools for managing the database on MongoDB.
- `etc` : This contains a sample of configuration file owning all the settings for every component of the project.
- *lib* : This contains the libraries of the project, namely a module to parse a configuration file.
- `web` : This contains the embedded Web server for easily accessing the CVE's from the database.

The configuration file `configuration.ini` to be created from the template in `etc` can be used to tune the IP address of the Web server of CVE-Search so that it can be used by SCAPL to consult CVE's once the list of suggestions is given back from *scapl-search* to *scapl-frontend*.

H.2.3 Interfaces

Querying Tool This is `search.py` in the `bin` folder. It allows to search for CVE's based on some interesting criteria. The following output shows a sample help text of the tool with only the options of interest.

```
$ bin/search.py
usage: search.py [-h] [-p P] [-f F] [-c C] [-o O] [-l] [-n] [-r] [-a] [-v V]
[...]
optional arguments:
  -h, --help  show this help message and exit
  -p P        S = search product, e.g. o:microsoft:windows_7
  -f F        F = free text search in vulnerability summary
  -c C        search one or more CVE-ID
  -o O        O = output format [csv|html|json|xml|cveid]
  -l          sort in descending mode
[...]
```

Web Server This is `index.py` in the `web` folder. It runs a Web server application relying on Flask. The following output shows the first logging lines when starting the Web server.

```
$ web/index.py
 * Running on http://configure_IP_address:configured_port/
     (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger pin code: 148-039-599
```

H.2.4 Input/Ouptut

So, regarding the found interface, the input and output are relatively straightforward. In the following listings, italic words designate parameters and lines of interest are in bold.

Input : Simple command call

```
bin/search.py -p product_cpe -o json -l
```

product_cpe is for example `o:microsoft:windows_7`.

Output : JSON[3]

```
{
   "Modified":"datetime",
   "Published":"datetime",
   "access":{
      "authentication":"scheme",
      "complexity":"LOW|MEDIUM|HIGH",
      "vector":"NETWORK|..."
   },
   "cvss":"10.0",
   "cvss-time":"datetime",
   "id":"CVE-YYYY-NNNN",
   "impact":{
      "availability":"severity",
      "confidentiality":"severity",
      "integrity":"severity"
   },
   "references":[
      "hyperlink", ...
   ],
   "summary":"text",
   "vulnerable_configuration":[
      "cpe", ...
   ],
   "vulnerable_configuration_cpe_2_2":[
      "cpe", ...
   ]
}
```

Afterwards, there only remains to compute the URL of the CVE to the Web server by just appending server's URL and the found CVE ID, e.g. giving `http://192.168.1.158:5158/CVE-2016-0158`.

[3] JavaScript Object Notation

H.3 Integration in SCAPL

This section presents how the plugin must be created so that it can bind the project to SCAPL through the asynchronous tasking mechanism.

H.3.1 Configuration

1. **Web Server** : CVE-Search has a Web server for providing a view of the found CVE's ; it should be ensured that it is well configured with the right IP address and port and that it is running.
2. **Configuration Parser** : This will be necessary to get the IP address and port of the Web server in order to build the URL's for the CVE's.

H.3.2 Plugin Establishment

SCAPL's Search Engine has, at its root, a folder named `search` containing all the plugins, each in the form of a subfolder containing project's sources and, in addition :

1. `__runner__.py` : The template of *TaskRunner* that translates the parameters from the Data Item into an object owning, as its attributes, the dictionary of input arguments (`self.param`) of the launcher of the plugged project.
2. `run.py` : The plugin itself, implementing the `run()` method of the *TaskRunner* with the particular processing that depends on the output of the plugged project.

In this way, only `run.py` must be rewritten for each plugin. The following code gives the implementation of CVE-Search plugin's `run.py`.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import json
from __runner__ import TaskRunner
from lib.Config import Configuration
from subprocess import Popen, PIPE

@TaskRunner.bind(__file__, 'CVE-Search')
def run(self):
    cmd = "bin/search.py -o json -p {cpe} -l".format(cpe=self.param['kw'])
    p = Popen(cmd.split(" "), stdout=PIPE, stderr=PIPE)
    out, err = p.communicate()
    if len(out) == 0:
        self.logger.error(err)
    cves = []
    for line in out.decode('utf-8').split('\n'):
        if line == '': continue
        cve = json.loads(line)
        cves.append((cve['id'], cve['summary'], ))
    cves = sorted(cves, key=lambda x: x[0])[::-self.param['n']]
    addr = 'http://{host}:{port}/cve/'.format(
        host=Configuration.getFlaskHost(), port=Configuration.getFlaskPort())
    return [{'link': addr + cid, 'title': cid, 'text': summary} for cid, summary in cves]

if __name__ == '__main__':
    run()
```

The code block between `cmd = ...` and `return ...` aims to start an OS subprocess for launching `search.py` and to parse the result to keep only the `n` most recent CVE's. The result is returned in the form of a list of dictionaries containing the hyperlink to CVE-Search's Web server, the CVE ID as the title of the page result and CVE's summary as its description text.

H.3.3 Data Item Creation

At this point, there still remains to create a Data Item in the administration interface of SCAPL. This can be done by obviously connecting as an administrator, by going to the *Wizard Scheme Design* menu and by adding a new *Data Item (Search Engine)* with the following attributes :

- **Identification**
  - **Name** : Known Vulnerabilities
  - **Description** : Found known vulnerabilities from common security sources.
- **Action**
  - **API** : `cve-search/run.py -param "'kw':'keywords', 'n':suggestions"`
  - **Keywords** : `%cpe%`
  - **Max suggestions** : 10

API will provide the binding to the right plugin (that is, `run.py` of the right plugged project) to be started as an asynchronous task carried by AMQP with Celery through the RabbitMQ server to *scapl-search*. *Keywords* will retrieve the `%cpe%` attribute of the current APL task and forward it as the parameter `kw` for the plugged project. *Max suggestions* will then determine the parameter `n` for the maximum number of most recent CVE's to return.