SESAME
SECURE AND SAFE MULTI-ROBOT SYSTEMS

Project Number 101017258

D5.3 Tools for Automated Security Analysis of MRS and for Production of EDDIs

Version 1.0
30 June 2022
Final

Public Distribution

FORTH

Project Partners: **Aero41, ATB, AVL, Bonn-Rhein-Sieg University, Cyprus Civil Defence, Domaine Kox, FORTH, Fraunhofer IESE, KIOS, KUKA Assembly & Test, Locomotec, Luxsense, The Open Group, Technology Transfer Systems, University of Hull, University of Luxembourg, University of York**

PROJECT PARTNER CONTACT INFORMATION

Aero41 Frédéric Hemmeler Chemin de Mornex 3 1003 Lausanne Switzerland E-mail: frederic.hemmeler@aero41.ch	**ATB** Sebastian Scholze Wiener Strasse 1 28359 Bremen Germany E-mail: scholze@atb-bremen.de
AVL Martin Weinzerl Hans-List-Platz 1 8020 Graz Austria E-mail: martin.weinzerl@avl.com	**Bonn-Rhein-Sieg University** Nico Hochgeschwender Grantham-Allee 20 53757 Sankt Augustin Germany E-mail: nico.hochgeschwender@h-brs.de
Cyprus Civil Defence Eftychia Stokkou Cyprus Ministry of Interior 1453 Lefkosia Cyprus E-mail: estokkou@cd.moi.gov.cy	**Domaine Kox** Corinne Kox 6 Rue des Prés 5561 Remich Luxembourg E-mail: corinne@domainekox.lu
FORTH Sotiris Ioannidis N Plastira Str 100 70013 Heraklion Greece E-mail: sotiris@ics.forth.gr	**Fraunhofer IESE** Daniel Schneider Fraunhofer-Platz 1 67663 Kaiserslautern Germany E-mail: daniel.schneider@iese.fraunhofer.de
KIOS Maria Michael 1 Panepistimiou Avenue 2109 Aglatzia, Nicosia Cyprus E-mail: mmichael@ucy.ac.cy	**KUKA Assembly & Test** Michael Laackmann Uhthoffstrasse 1 28757 Bremen Germany E-mail: michael.laackmann@kuka.com
Locomotec Sebastian Blumenthal Bergiusstrasse 15 86199 Augsburg Germany E-mail: blumenthal@locomotec.com	**Luxsense** Gilles Rock 85-87 Parc d'Activités 8303 Luxembourg Luxembourg E-mail: gilles.rock@luxsense.lu
The Open Group Scott Hansen Rond Point Schuman 6, 5th Floor 1040 Brussels Belgium E-mail: s.hansen@opengroup.org	**Technology Transfer Systems** Paolo Pedrazzoli Via Francesco d'Ovidio, 3 20131 Milano Italy E-mail: pedrazzoli@ttsnetwork.com
University of Hull Yiannis Papadopoulos Cottingham Road Hull HU6 7TQ United Kingdom E-mail: y.i.papadopoulos@hull.ac.uk	**University of Luxembourg** Miguel Olivares Mendez 2 Avenue de l'Universite 4365 Esch-sur-Alzette Luxembourg E-mail: miguel.olivaresmendez@uni.lu
University of York Simos Gerasimou & Nicholas Matragkas Deramore Lane York YO10 5GH United Kingdom E-mail: simos.gerasimou@york.ac.uk nicholas.matragkas@york.ac.uk	

DOCUMENT CONTROL

Version	Status	Date
0.1	Initial draft with outline and first content	9 June 2022
0.2	First draft	11 June 2022
0.7	Ready for internal review	24 June 2022
0.8	Reviewed version	29 June 2022
1.0	Final QA version	30 June 2022

TABLE OF CONTENTS

TABLE OF FIGURES

EXECUTIVE SUMMARY

This deliverable describes the techniques and tools that are adopted towards the successful application of the SESAME security assessment methodology presented in D5.1. The adopted tools are intended to create system models able to incorporate information about the security status of a given system. The aforementioned models must be converted to ODE-compliant models for the generation of the runtime EDDIs.

The aim of such runtime EDDIs is the runtime dependability management. Along with the models, input from security monitoring tools is needed for the dependability management to take place.

The tools that are mentioned in this document are either opensource, with their code publicly available in free repositories, or custom that are going to become available at the project's repository. A short description of each follows:

- **cve-search** is a tool that imports CVE and CPE into a MongoDB, facilitating search and processing of CVEs. The main advantage of this tools is the fact that a local instance of CVE is created serving lookup requests.

- **RVD parser** is a custom JAVA tool that allows for querying the RVD database for robot vulnerabilities based on given product description or a CPE. Moreover, it facilitates querying RVD database for related CWEs given a specific CVE.

- **openVAS** is a generic scanner that can meticulously scan all ports on the target system for active services and provide a comprehensive report on the discovered assets, such as running software, specific version numbers etc. After that, it conducts attacks to the discovered services by using a plethora of known exploits and reports on the vulnerable ones.

- **w3af** is a scanner specifically focused on Web applications. It crawls the Web application under test, to detect possible injection points, and then it tries to exploit each one with the corresponding payloads.

- **CAPEC identifier** is a custom JAVA tool that allows for utilization of a local instance of the CAPEC catalogue to retrieve information about known attacks based on given known weaknesses.

- **Snort** is an intrusion detection system that performs real-time traffic analysis and packet logging. It uses sets of rules for the definition of malicious network activity. Packets that match against said rules generate alerts for users.

LIST OF ABBREVIATIONS

API	Application Programming Interface
CAPEC	Common Attack Pattern Enumeration and Classification
CIRCL	Computer Incident Response Centre Luxembourg
CPE	Common Platform Enumeration
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
EDDI	Executable Digital Dependability Identity
HTTP	Hypertext Transfer Protocol
MRS	Multi Robot System
ODE	Open Dependability Exchange
REST	REpresentational State Transfer
ROS	Robot Operating System
RVD	Robot Vulnerability Database
RVSS	Robot Vulnerability Scoring System
TCP	Transmission Control Protocol
URL	Uniform Resource Locator

1 INTRODUCTION

1.1 OVERVIEW

In D5.1 the SESAME security assessment concept and methodology was described. The main goal of the security assessment process is to gather all the necessary information for the description of the security state of a given system. The discovered vulnerabilities of the individual system components lead to the identification of potential attacks that an adversary could try to conduct, taking advantage of the existing system flaws.

With the extension of the ODE metamodel described in D4.2/D5.2, the aforementioned information can be stored and used for the creation of the Executable Digital Dependability Identity (EDDI) model-based artefact. EDDIs incorporate all the dependability information of a system. Although EDDIs are meant to have a runtime usage, the gathering of the included information is done also at design time with the use of different tools and techniques. These tools and techniques are described in this deliverable.

Additionally, the corresponding security monitoring system that will feed the runtime version of the EDDIs with events to trigger the individual processes, is also described.

1.2 DELIVERABLE STRUCTURE

The rest of the deliverable is structured as follows. In section 2, a high level overview of the SESAME security assessment process is given, divided in separated subsections. In each of these subsections, one of the subprocesses of security assessment is described and the corresponding tools that have been used or developed are mentioned. Section 3 focuses on the runtime aspect of the security assessment. The necessity of a runtime security monitoring tools is mentioned and the tool that was finally adopted is described. Information that are given for each tool are installation instructions, available APIs, and sample input and output. Section 4 presents a first integration of the described tools in the context of a test scenario. The test scenario presents an intermediate output of the security assessment process for a system with architecture that is very similar with three of the project use cases. Finally, section 6 concludes this deliverable.

2 THE SESAME SECURITY METHODOLOGY

2.1 PROCESSES OF THE SESAME SECURITY METHODOLOGY

The security assessment developed in the context of SESAME is strongly influenced by the threat modelling process and adopts its main principles, following a structured series of steps, which overlap in many cases with the highly structured process of threat modelling.

The next subsections include a high level description of the main processes of the SESAME security methodology, along with a description of the techniques and tools that are used.

2.1.1 Identification of vulnerabilities

The input of the identification of vulnerabilities process is the description of the target system in terms of architecture components, assets, entry points and trust boundaries. This description pinpoints all the deployed programs, libraries and services, triggering the process of requesting free catalogs and databases, populated with known vulnerabilities of the recognized software.

Common Vulnerabilities and Exposures (CVE)[1] is the first of such databases that is used for this purpose. CVE is a list of computer security flaws, cybersecurity vulnerabilities, and can be used for searching or incorporated into products and services for free. Each of these flaws is assigned an identifier called CVE-ID, which is used as a dependable way to uniquely recognise vulnerabilities. A more extended description of CVE list can be found in section 2.2.3 (Security knowledge repositories) of D5.1. However, since SESAME focuses on MRS, a robot-oriented vulnerability database would add more value to the proposed security assessment methodology. Robot Vulnerability Database (RVD) is such a database. RVD includes robot related vulnerabilities and bugs that are referred to software and hardware. The aim is to record and categorize robot related flaws. RVD is available at GitHub and offers tools that ease its management. Robot Vulnerability Scoring System (RVSS) is used for the rating of the included vulnerabilities. Once again, a more comprehensive description of RVD can be found in section 2.2.3 of D5.1.

2.1.1.1 Techniques and tools

The description of a system to be assessed can be done in two ways. According to the first one, system administrator can provide information about the system components by filling in forms and answering corresponding questionnaires. This way of gathering system information is useful for systems that are not fully implemented yet, being probably at the design phase. The output of this process is a full description of each involved system component, in the form of a standardized information such as this provided by Common Platform Enumeration (CPE)[2]. CPE is a standardized method of describing and identifying classes of applications, operating systems, and hardware devices present among assets of a given system. This standardized information can be used for creating requests to the available security knowledge repositories, asking for vulnerabilities, weaknesses or known attacks.

[1] https://cve.mitre.org/index.html
[2] https://csrc.nist.gov/projects/security-content-automation-protocol/specifications/cpe

cve-search: cve-search is a tool that imports CVE and CPE into a MongoDB, facilitating search and processing of CVEs. The main advantage of this tools is the fact that a local instance of CVE is created serving lookup requests. In that way, direct requests to the public CVE databases are reduced. At the same time, local requests are served faster without exposing sensitive information to the internet. Among the cve-search offerings, are the followings: i) a back-end to store vulnerabilities and related information, ii) an intuitive web interface for search and managing vulnerabilities, iii) a series of tools to query the system and a web API interface. cve-search is used by many organizations including the public CVE services of Computer Incident Response Center Luxembourg (CIRCL). The source code is available on GitHub[3]. A whole community maintains it including CIRCL.

The installation, in a containerized form, of cve-search includes the following steps (Listing 1):

```
~/# git clone https://github.com/cve-search/CVE-Search-Docker.git

~/# cd CVE-Search-Docker

~/# sudo docker-compose up
```

Listing 1: cve-search installation commands

As it can be seen, the first command clones the code from GitHub and then the docker-compose up command is used for the creation and start of the corresponding container. Figure 1 depicts the first part of the container creation, where the redis image is pulled and used. Figure 2 shows the second part, where mongodb is deployed. Finally, Figure 3 shows the first steps of installing cve-search, where the python image is pulled.

```
manos@manos-OptiPlex-755:~/cve-search/CVE-Search-Docker$ sudo docker-compose up
Creating network "cvesearchdocker_frontend" with the default driver
Creating network "cvesearchdocker_backend" with the default driver
Building redis
Step 1/10 : FROM redis:latest
latest: Pulling from library/redis
f7a1c6dad281: Pull complete
c5f81eaec564: Pull complete
2be237d3defa: Pull complete
1640a11de2e5: Pull complete
9138edee9512: Pull complete
c62664237d8c: Pull complete
Digest: sha256:feb846600a248be6e6afbad39bf5b91afaef1de5524fd85b9b53839d0fd4af96
Status: Downloaded newer image for redis:latest
 ---> 0e403e3816e8
Step 2/10 : ENV APP /install
 ---> Running in 4826e6ee32bc
Removing intermediate container 4826e6ee32bc
 ---> 36f45f79fc33
Step 3/10 : RUN mkdir $APP
 ---> Running in 9c6c76f26b4a
Removing intermediate container 9c6c76f26b4a
 ---> bb3368962baa
```

Figure 1: cve-search installation - Building redis

[3] https://github.com/cve-search/cve-search

```
Building mongo
Step 1/12 : FROM mongo:4.4
4.4: Pulling from library/mongo
7c3b88808835: Pull complete
48a403577c28: Pull complete
76bbb7dc9013: Pull complete
e81b1e5a386b: Pull complete
7bdc1db49ec9: Pull complete
9a57632a8051: Pull complete
5bdd30561f84: Pull complete
a0c8b1ee700e: Pull complete
18023a840b71: Pull complete
e13aa570e4f6: Pull complete
Digest: sha256:cedca14375e629bb97a5caf5fd3d725f5bcc887b1b5143f89de32244d730cd58
Status: Downloaded newer image for mongo:4.4
 ---> 0450bd78d2c1
Step 2/12 : ENV APP /install
 ---> Running in 1aad0160cbff
Removing intermediate container 1aad0160cbff
 ---> 8b98dbebc31c
Step 3/12 : RUN mkdir $APP
 ---> Running in 342c2de3cd44
Removing intermediate container 342c2de3cd44
 ---> 8976aaa36e41
```

Figure 2: cve-search installation - Building mongo

```
Building cve_search
Step 1/17 : FROM python:latest
latest: Pulling from library/python
e4d61adff207: Pull complete
4ff1945c672b: Pull complete
ff5b10aec998: Pull complete
12de8c754e45: Pull complete
ada1762e7602: Pull complete
2f2b2e030155: Pull complete
02df93743040: Pull complete
9ecc1601d27c: Pull complete
1d814fdfc8ad: Pull complete
Digest: sha256:9d0a30e885d3ef26369279246298706ca7291620718fafbe5a1a2713bd911bf1
Status: Downloaded newer image for python:latest
 ---> 33ceb4320f06
Step 2/17 : ENV APP /app
 ---> Running in ea4125c142e3
Removing intermediate container ea4125c142e3
 ---> 7baf55faf57d
Step 3/17 : RUN mkdir $APP
 ---> Running in 95267d2c9923
Removing intermediate container 95267d2c9923
 ---> cc85eaeb87b9
```

Figure 3: cve-search installation - Building cve_search

After the installation, the `./bin/search.py` script can be used for querying the local instance of CVE. The corresponding help page is depicted in Listing 2.

```
usage: search.py [-h] [-q Q] [-p P [P ...]] [--only-if-vulnerable] [--
strict_vendor_product] [--lax] [-f F] [-c C] [-o O] [-l] [-n] [-r] [-
a] [-v V] [-s S] [-t T] [-i I]

Search for vulnerabilities in the National Vulnerability DB. Data from
http://nvd.nist.org.

options:
  -h, --help        show this help message and exit
  -q Q              Q = search pip requirements file for CVEs, e.g.
dep/myreq.txt
  -p P [P ...]      S = search one or more products, e.g.
```

```
o:microsoft:windows_7 or o:cisco:ios:12.1 or o:microsoft:windows_7
o:cisco:ios:12.1. Add --only-if-vulnerable if only vulnerabilities
that directly affect the product are wanted.
  --only-if-vulnerable  With this option, "-p" will only return vul-
nerabilities directly assigned to the product. I.e. it
                        will not consider "windows_7" if it is only
mentioned as affected OS in an adobe:reader
                        vulnerability.
  --strict_vendor_product
                        With this option, a strict vendor product
search is executed. The values in "-p" should be formatted
                        as vendor:product, e.g. microsoft:windows_7
  --lax                 Strict search for software version is disa-
bled. Note that this option only support product
                        description with numerical values only (of the
form cisco:ios:1.2.3)
  -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
  -n                    lookup complete cpe (Common Platform Enumera-
tion) name for vulnerable configuration
  -r                    lookup ranking of vulnerable configuration
  -a                    Lookup CAPEC for related CWE weaknesses
  -v V                  vendor name to lookup in reference URLs
  -s S                  search in summary text
  -t T                  search in last n day
  -i I                  Limit output to n elements (default: unlim-
ited)
```

Listing 2: cve-search help page

As we can see, there are different ways to form a request asking for vulnerabilities.

- Request returning vulnerabilities directly assigned to a specific product (`./bin/search.py -p microsoft:windows_7 -a -o json`).
- Request returning vulnerabilities based on text search in the vulnerability summary (`./bin/search.py -f "robotic simulator" -a -o json`).
- Request for a specific CVE ID (`./bin/search.py -c CVE-2010-3333`).
- Request the last 2 CVE entries in atom format (`./bin/dump_last.py -f atom -l 2`).

RVD custom parser: cve-search is a great tool for searching and processing vulnerabilities from the CVE catalogue. However, we need an additional tools that will offer the same functionality for the RVD database. RVD comes with a set of tools for the management of the database entries and is available as an opensource project at GitHub[4]. The installation steps are presented in Listing 3.

[4] https://github.com/aliasrobotics/RVD

```
~/# git clone https://github.com/aliasrobotics/RVD.git

~/# sudo python3 setup.py build

~/# sudo pip install - -upgrade requests

~/# sudo pip install - -upgrade zope.inteface

~/# sudo python3 setup.py install
```

Listing 3: RVD installation steps

After the installation described above, the "rvd" command is available for managing the populated RVD database.

Although the RVD GitHub project by Alias Robotics does come with its own access tools, they do not offer the functionality we need, such as querying the database for robot vulnerabilities based on given product description or a CPE. Another useful functionality would be the ability to query the database for related CWEs given a specific CVE.

Towards the desired functionality described in the previous paragraph, a custom RVD parser has been created. The RVD installation offers the "`rvd list --dump --label vulnerability`" command that returns all the RVD database entries, which are labeled as vulnerabilities. An example of such an entry is depicted in Listing 4. The provided information for each robot vulnerability include related CVEs and CWEs, affected systems, severity scores (RVSS, CVSS), exploitation and mitigation descriptions.

```
id: 3337
title: Service DoS through arbitrary pointer dereferencing on KUKA
simulator
type: vulnerability
description: "Visual Components (owned by KUKA) is a robotic simulator
that allows simulating factories and robots in order toimprove plan-
ning and decision-making processes. … Accordingly, a DoS in the simu-
lation might have higher repercusions, dependingon the Industrial Con-
trol System (ICS) ICS infrastructure."
cwe: CWE-248
cve: CVE-2020-10292
keywords:
- KUKA, RMS sentinel LM, Visual Components, DoS
system: Visual Components Network License Server 2.0.8
vendor: KUKA Roboter GmbH, Visual Components
severity:
  rvss-score: 6.1
  rvss-vector: RVSS:1.0/AV:IN/AC:L/PR:N/UI:N/S:U/Y:Z/C:N/I:L/A:H/H:N
  severity-description: High
  cvss-score: 8.2
  cvss-vector: CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:H
links:
- https://cwe.mitre.org/data/definitions/248.html
- https://www.visualcomponents.com/products/downloads/
- https://www.visualcomponents.com/products/visual-components/
flaw:
  phase: runtime-operation
  specificity: subject-specific
  architectural-location: application-specific
```

```
  application: Visual Components, RMS sentinel LM
  subsystem: simulation
  package: null
  languages: null
  date-detected: null
  detected-by: Sharon Brizinov (Claroty)
  detected-by-method: testing-dynamic
  date-reported: 2020-10-27
  reported-by: Sharon Brizinov (Claroty)
  reported-by-relationship: security researcher
  issue: https://gitlab.com/aliasrobotics/offensive/rvd/flaws/-
/issues/712
  reproducibility: always
  trace: null
  reproduction: null
  reproduction-image: null
exploitation:
  description: |
    To exploit this vulnerability the attacker needs to have network
access to the license server (either because
    it's exposed or because the internal network has been compromised.
Cause is related to the number of requested
    strings to merge, which is not correlated to the number of strings
provided, and so arbitrary pointers from the
    stack are popped out and dereferenced. This results with an un-
caught Access Violation exception which terminates
    the program. PoC available constructs a response reply to
featureInfoToFile with is a mismatch between the
    number of strings to merge and the requested amount leading to an
Access Violation exception and terminating the
    program. See alurity's robotsploit/exploits/kuka/rms exploits.
  exploitation-image: Not available
  exploitation-vector: null
  exploitation-recipe:
    networks:
    - network:
      - driver: bridge
      - name: kuka-simulation
      - subnet: 14.0.0.0/24
    vms:
      - vm:
        - name: vm1
        - path: $(pwd)/vms/visualcomponents_2.0.8
        - network: kuka-simulation
        - ip: 14.0.0.4
    containers:
    - container:
      - name: attacker
      - modules:
        - base: regis-
try.gitlab.com/aliasrobotics/offensive/alurity/alurity:latest
        - volume: regis-
try.gitlab.com/aliasrobotics/offensive/alurity/expl_robosploit/expl_ro
bosploit:latest
        - volume: regis-
try.gitlab.com/aliasrobotics/offensive/alurity/deve_atom:latest
        - volume: regis-
try.gitlab.com/aliasrobotics/offensive/alurity/reco_nmap:latest
        - volume: regis-
try.gitlab.com/aliasrobotics/offensive/alurity/expl_icssploit:latest
        - volume: regis-
```

```
try.gitlab.com/aliasrobotics/offensive/alurity/expl_metasploit:latest
        - volume: regis-
try.gitlab.com/aliasrobotics/offensive/alurity/fore_wireshark:latest
        - network: kuka-simulation
mitigation:
  description: |
    Do not launch Visual Components while connected to local or wide
area networks. Contain the simulation through
    virtualization.
  pull-request: null
  date-mitigation: null
```

Listing 4: Example robot vulnerability from the RVD database

The whole set of available robot vulnerabilities is the input for our custom parser. A set of Java classes has been created for storing and managing the information provided for the incoming robot vulnerabilities (Figure 4). The main class is called "RvdVulnerability", while four more subclasses are needed, called "Severity", "Exploitation", "Flaw", and "Mitigation".

A rest API has been created for the RVD parser to update the local version of the RVD database. The corresponding code is depicted in Listing 5. As it can be seen, the API endpoint is http://ipAddress:port/rvdinsert and the anticipated body of the request is a list of RVD vulnerabilities. The structure of the RvdVulnerability class can be seen in Figure 4. The @PostMapping annotation ensures that HTTP POST requests are mapped onto a specific handler method, the rvdInsert in this case. The request is expected to have a body in application/json format. The @RequestBody annotation enables the automatic deserialization of the request body onto a Java object. What follows the mapping of the request body to the corresponding Java instance, is the insertion of the latter into the rvdVulnerabilities array list in the file system. This API endpoint allows for the regular update of the RVD local version to include any newly added vulnerabilities.

```
@PostMapping("/rvdinsert")
public String rvdInsert(@RequestBody ArrayList<RvdVulnerability>
rvdJson) {
   //Generate the rvd database (rvdVulnerabilities) with the input
from the rvdjson array

   …

   System.out.println("Local RVD Repository has been updated");
   return "rvdresult";
}
```

Listing 5: REST API for the update of the local version of the RVD database

Figure 4: RVD Java classes of the custom RVD parser

Another exposed REST API is utilized for querying the local RVD database instance for CWEs based on a given CVE. Listing 6 depicts the corresponding code.

```
@PostMapping(value = "/searchwithcve")
public String searchWithCve(@RequestBody String cveId) {

   ArrayList<String> cweFilteredArrayList = new ArrayList<>();
   for (int i = 0; i <rvdVulnerabilities.size() ; i++) {

      if (rvdVulnerabilities.get(i).cve.equals(cveId)){
         cweFilteredArrayList.add(rvdVulnerabilities.get(i).cwe);

      }
   }
   …
```

```
    return "rvdresult";
}
```

Listing 6: REST API for querying for CWEs based on a given CVE

The API endpoint is http://ipAddress:port/searchwithcve and the anticipated body of the request is a CVE-ID. The @PostMapping annotation ensures that HTTP POST requests are mapped onto the searchWithCve handler method. What follows is the collection of all the related CWEs of the vulnerability with the given CVE-ID.

The implementation of the described custom RVD parser is an ongoing work that will be continued the coming months of the project lifetime. More functionality will be added, such as the ability to search for vulnerabilities using the name and version of each software present in the system in question.

A parser searches said vulnerability directories and, using the name and version of each software present in the system in question, spots the associated vulnerabilities. Each of those vulnerabilities are uniquely identified by the CVE identifiers (CVE-IDs). A list of such CVE-IDs is the output of this process. The said vulnerability directories are constantly updated with information regarding newly discovered vulnerabilities. Inherently the approach followed here is also not static as it will be in sync with the updated directories.

OpenVAS:

However, system description can be also offered as functionality by a set of automated tools, scanners that can scan given network and/or subnetworks for available services and then use open vulnerability databases to discover known vulnerabilities. OpenVAS, and w3af are two choices. The advantage of the usage of such scanning tools is that they can reveal services that are running in devices, which are part of the target system, and the provider of the system information may not be aware of. On the other hand, their disadvantage is that they can be used only after the system is up and running, otherwise the vulnerability scanning tools cannot produce an output.

The use of multiple scanners is preferred as their variety of auditing techniques, attack payloads etc. can collectively detect a plethora of different vulnerabilities and security malpractices. For the current implementation we have decided to include two such scanners, OpenVAS and w3af, whose capabilities we describe in the following paragraphs. It should be mentioned that while there is currently support for these two scanners, the modular design of the identification of vulnerabilities process allows for the easy addition of even more such tools.

Open Vulnerability Assessment Scanner (OpenVAS) is a generic scanner that offers several capabilities and considers a significant number of different vulnerabilities. Its core strength is that it can meticulously scan all ports on the target system for active services and provide a comprehensive report on the discovered assets, such as running software, specific version numbers etc. Furthermore, OpenVAS is able to conduct attacks to the discovered services by using a plethora of known exploits and reporting on the vulnerable ones by providing a high-level description of each vulnerability and the CVE's assigned CVSS score and severity level. Another advanced capability of OpenVAS is that it already makes use of wrappers for other vulnerability scanners (e.g.,

Nmap, wapiti) and leverages them to enhance its coverage, as well as number and type of detected vulnerabilities. Finally, OpenVAS offers a set of predefined configurations that cover the most common scanning scenarios, including fast, fast ultimate, deep and deep ultimate scans. One last feature of OpenVAS is the addition of custom configurations through its administrator dashboard.

The following steps have to be followed for the installation of OpenVAS, in a containerized form (Listing 7):

```
~/# git clone https://github.com/mikesplain/openvas-docker.git

~/# sudo docker run -d -p 443:443 --name openvas mikesplain/openvas
```

Listing 7: OpenVAS installation commands

As it can be seen, the first command clones the code from the corresponding GitHub project and then the docker run command is used for the creation and start of the OpenVAS container. By doing so, the individual OpenVAS components will be installed and become available. The whole installation, gsad, will be running on port 443. The OpenVAS Scanner (openvassd) will be running on TCP Port 9391 and the OpenVAS Manager (openvasmd) on TCP port 9390. Finally, the redis-server will be running on TCP 6379. The OpenVAS web interface is available in the browser, which shows the login screen for the Greenbone Security Assistant (Figure 5).

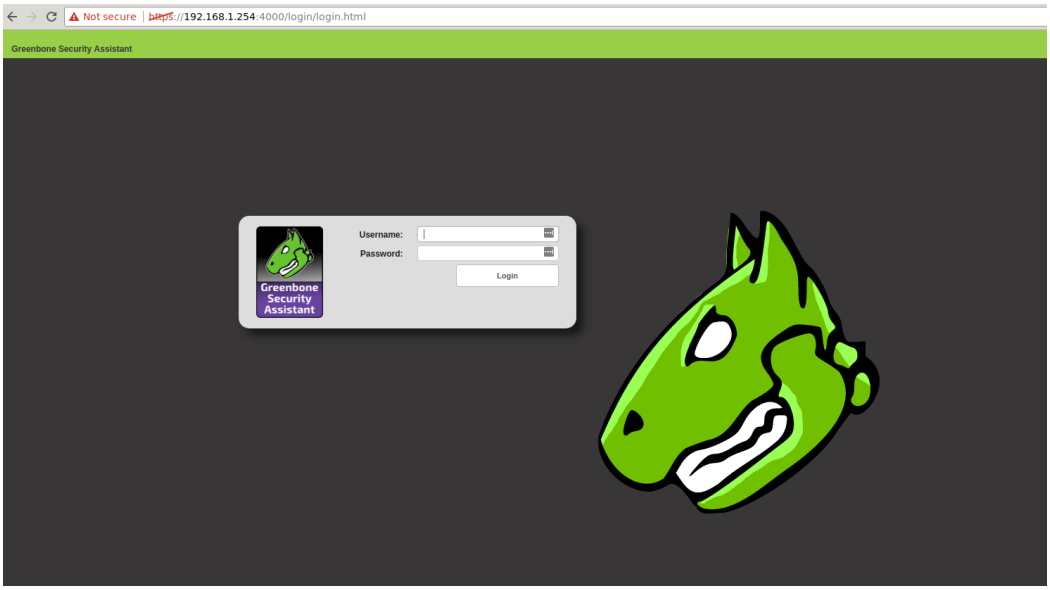

Figure 5: OpenVAS web interface

NVT plugins are .nasl files created using the NASL code, a scripting language is a legacy of its original Nessus code base. NASL stands for Nessus Attack Scripting Language developed in 1998. On a typical OpenVAS server there will be tens of thousands of such files used by the OpenVAS scanner service and performing vulnerability checks. Regular updating the local instance of the available NVT plugins is a good practice. The following commands should be executed:

```
~/# openvas-nvt-sync
```

```
~/# kill $pid_of_openvassd

~/# kill $pid_of_openvasmd

~/# openvasmd –rebuild

~/# openvasmd

~/# openvassd
```

Listing 8: Commands for update NVT plugins

According to Listing 8, the first command updates the NVT's on the system, while the rest rebuild the NVT cache of the OpenVAS manager by stopping and starting the it.

A very useful feature of OpenVAS is its command line tool (omp), especially towards the automation of OpenVAS scans. The omp command has a large number of options. The help page of the command is depicted in Listing 9.

```
root@localhost:~# omp –help

Usage:
  omp [OPTION...] - OpenVAS OMP Command Line Interface

Help Options:
  -?, --help         Show help options

Application Options:
  -h, --host=        Connect to manager on host
  -p, --port=        Use port number
  -V, --version      Print version.
  -v, --verbose      Verbose messages (WARNING: may reveal passwords).
  --use-certs        Use client certificates to authenticate.
  --client-cert=     Client certificate. Default:
/usr/local/var/lib/openvas/CA/clientcert.pem
  --client-key=      Client key. Default:
/usr/local/var/lib/openvas/private/CA/clientkey.pem
  --client-ca-cert=  Client CA certificate. Default:
/usr/local/var/lib/openvas/CA/cacert.pem
  -u, --username=    OMP username
  -w, --password=    OMP password
  --config-file=     Configuration file for connection parameters.
  -P, --prompt       Prompt to exit.
  -O, --get-omp-version     Print OMP version.
  -n, --name=        Name for create-task.
  -C, --create-task  Create a task.
  -m, --comment=     Comment for create-task.
  -c, --config=      Config for create-task.
  -t, --target=      Target for create-task.
  -E, --delete-report  Delete one or more reports.
  -D, --delete-task    Delete one or more tasks.
  -R, --get-report     Get report of one task.
  -F, --get-report-formats  Get report formats. (OMP 2.0 only)
  -f, --format=        Format for get-report.
  --filter=            Filter string for get-report
  -G, --get-tasks      Get status of one, many or all tasks.
  -g, --get-configs    Get configs.
  -T, --get-targets    Get targets.
  -i, --pretty-print   In combination with -X, pretty print the re-
```

```
sponse.
  -S, --start-task     Start one or more tasks.
  -M, --modify-task    Modify a task.
  --ping               Ping OMP server
  --timeout=           Wait  seconds for OMP ping response
  --file               Add text in stdin as file on task.
  -X, --xml=        XML command (e.g. "").  "-" to read from stdin.
  --send-file=      Replace SENDFILE in xml with base64 of file.
  --details         Enable detailed view.
```

Listing 9: OpenVAS command line tool (omp) - help page

W3af:

Web Application audit and attack framework (W3af) is a scanner specifically focused on Web applications and tests for different types of Web-based vulnerabilities. Some of these vulnerabilities are reflected and stored XSS, SQL injections, cross-site request forgery, and remote/local file inclusions. The first step is crawling the Web application under test in order to detect possible injection points and then w3af tries to exploit each one with the corresponding payloads. As OpenVAS does, w3af offers a set of predefined configurations, such as performing a fast or a full scan with all modules enabled. The predefined configurations are:

- Bruteforce: Test default and commonly used credentials.
- Audit high risk: Identify only high risk vulnerabilities, e.g., SQL injections, OS commanding, etc.
- Full scan: Perform a scan with all auditing plugins enabled.
- OWASP Top 10: Perform a scan focusing on OWASP's top 10 most common vulnerabilities.
- Fast scan: Perform a scan, using only the fastest auditing plugins.
- Infrastructure: Fingerprint the remote Web infrastructure.

The installation steps are presented in Listing 10. The first command clones the source code form the corresponding GitHub project. After navigating to the right folder, the w3af_dependency_install.sh script installs all the dependencies that are necessary for the w3af_console command to run.

```
~/# git clone https://github.com/andresriancho/w3af.git

~/# cd w3af/

~/# ./w3af_console

~/# ./tmp/w3af_dependency_install.sh
```

Listing 10: W3af installation commands

The console user interface of w3af allows for framework and plugin settings configuration, scan launching and exploitation of a vulnerability. The help page depicted in Listing 11 shows the main menu commands.

```
w3af>>> help
|-------------------------------------------------------------------|
| start          | Start the scan.                                  |
| plugins        | Enable and configure plugins.                    |
| exploit        | Exploit the vulnerability.                       |
| profiles       | List and use scan profiles.                      |
| cleanup        | Cleanup before starting a new scan.              |
|-------------------------------------------------------------------|
| help           | Display help. Issuing: help [command] , prints   |
|                | more specific help about "command"               |
| version        | Show w3af version information.                   |
| keys           | Display key shortcuts.                           |
|-------------------------------------------------------------------|
| http-settings  | Configure the HTTP settings of the framework.    |
| misc-settings  | Configure w3af misc settings.                    |
| target         | Configure the target URL.                        |
|-------------------------------------------------------------------|
| back           | Go to the previous menu.                         |
| exit           | Exit w3af.                                       |
|-------------------------------------------------------------------|
| kb             | Browse the vulnerabilities stored in the         |
|                | Knowledge Base                                   |
|-------------------------------------------------------------------|
```

Listing 11: w3af help page

Similarly to OpenVAS, w3af uses plugins to achieve different goal such as finding new vulnerabilities, identifying new URLs and writing these to different file types. There are categories of plugins such as audit, crawl, bruteforce, grep, and auth, to name a few. Their configuration can be done through the console user interface. To enable the xss and sqli plugins, for example, the following command is used (Listing 12).

```
w3af/plugins>>>  audit xss, sqli
```

Listing 12: Enable xss and sqli w3af plugins command

After the configuration of the plugins, a scan may start. The corresponding commands are depicted in Listing 13. As we can see, the target of the scan is defined and then the start command is used to launch the scan.

```
w3af>>> target

w3af/config:target>>> set target http://localhost/

w3af/config:target>>> back

w3af>>> start
```

Listing 13: w3af start scan command

2.1.2 Identification of potential attacks

It is already described in the previous section, the identification of vulnerabilities can be conducted either by using the corresponding search tools (cve-search and custom RVD parser) of the two major security knowledge repositories (CVE and RVD), or by running scanner tools (OpenVAS and w3af) that scan the target system for available

services. In both cases, the output of the identification of vulnerabilities is a set of CVE-IDs, which serves as input to the next step of the security assessment, the identification of the potential attacks to the system.

The intermediate step between the identified system vulnerabilities and the discovery of the potential attacks that an adversary could try to conduct towards the system in question is the Common Weakness Enumeration (CWE). CWE catalog, a list of software and hardware weakness types and its role towards the discovery of the potential system attacks is shown in Figure 6.

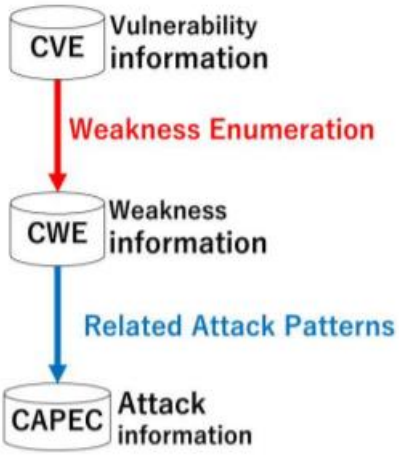

Figure 6: Discovering potential attacks from known vulnerabilities[5]

"Weakness Enumeration" is one of the fields in the description of a given vulnerability, entry of the CVE repository. In this field a list of all the related to a specific vulnerability weakness types are provided, in the form of CWE-IDs.

Additionally, the description of every weakness type (CWE-ID) includes a field called "Related Attack Patterns", presenting the attack patterns used for the exploitation of the corresponding weakness, in the form of CAPEC-IDs. A CAPEC-ID is a unique identifier for a given attack, and can be used for creating requests to the Common Attack Pattern Enumeration and Classification (CAPEC) catalogue. In this way, it is possible to trace a list of CAPEC-IDs from a single CVE-ID.

2.1.2.1 Techniques and tools

Two different tools can be used for the identification of potential attacks, based on the source of the identified vulnerabilities.

cve-search: cve-search is already described in the previous subsection. However, it is also mentioned here since it can be requested for known attacks related to a provided CVE-ID or a specific product (software/hardware).

[5] Kenta Kanakogi, Hironori Washizaki, Yoshiaki Fukazawa, Shinpei Ogata, Takao Okubo, Take-hisa Kato, Hideyuki Kanuka, Atsuo Hazeyama, and Nobukazu Yoshioka. Tracing CAPEC at-tack patterns from CVE vulnerability information using natural language processing technique. In Proceedings of the 54th Hawaii International Conference on System Sciences, page 6996, 2021.

For example, the command depicted in Listing 14 requests the CVE repository for all the known vulnerabilities related with Microsoft Windows 7 operating system.

```
~/# ./bin/search.py -p microsoft:windows_7 -a -o json
```

Listing 14: cve-search command for vulnerabilities related to

An simplified example of the information that cve-search returns for each of the related vulnerabilities is shown in Listing 15. This example includes just one vulnerability related to Windows 7. As it can be seen, in the JSON formatted response, there is an element called "capec" with information such as id, name, related capecs, related weaknesses, solutions, severity, etc.


```
{
    "Modified": "2017-09-19 01:31:00",
    "Published": "2011-03-03 20:00:00",
    "access": {
        "authentication": "NONE",
        "complexity": "HIGH",
        "vector": "NETWORK"
    },
    "assigner": "product-security@apple.com",
    "capec": [
        {
        "execution_flow": {
            "1": {
                "Description": "[Identify target application]...",
                "Phase": "Explore",
                "Techniques": []
            },
            "2": {
                "Description": "[Find injection vector] The ...",
                "Phase": "Experiment",
                "Techniques": ["Provide large input to a ..."]
            },
            "3": {
                "Description": "[Craft overflow content] The ...",
                "Phase": "Experiment",
                "Techniques": ["Create malicious shellcode ..."]
            "4": {
                "Description": "[Overflow the buffer] Using the...",
                "Phase": "Exploit",
                "Techniques": []
            }
        },
        "id": "8",
        "loa": "High",
        "name": "Buffer Overflow in an API Call",
        "prerequisites": "The target host exposes an API ...",
```

```
        "related_capecs": ["100"],
        "related_weakness": ["118", "119", "120", "20", "680", "697",
"733", "74"],
        "solutions": "Use a language or compiler that ... ",
        "summary": "This attack targets libraries or ...",
        "taxonomy": {},
        "typical_severity": "High"
    }
    ],
    "cvss": 7.6,
    "cvss-time": "2017-09-19 01:31:00",
    "cvss-vector": "AV:N/AC:H/Au:N/C:C/I:C/A:C",
    "cvss3": null,
    "cwe": "CWE-119",
    "exploitabilityScore": 4.9,
    "id": "CVE-2011-0112",
    "impact": {
        "availability": "COMPLETE",
        "confidentiality": "COMPLETE",
        "integrity": "COMPLETE"
    },
    "impactScore": 10.0,
    "last-modified": {
        "$date": 1505784660000
    },
    "products": ["itunes", "webkit"],
    "references": ["http://support.apple.com/kb/HT4554"],
    "summary": "WebKit, as used in Apple iTunes before 10.2 ...",
    "vendors": ["apple"],
    "vulnerable_configuration":
["cpe:2.3:a:apple:itunes:4.6.0:*:*:*:*:*:*:*"],
    "vulnerable_configuration_cpe_2_2": [],
    "vulnerable_configuration_stems": ["cpe:2.3:a:apple:itunes"],
    "vulnerable_product":
["cpe:2.3:a:apple:itunes:4.6.0:*:*:*:*:*:*:*"],
    "vulnerable_product_stems": ["cpe:2.3:a:apple:itunes"]
}
```


Listing 15: cve-search example vulnerability output

Custom CAPEC identifier: Once again, cve-search offers the functionality of querying the CVE repository. A tools with similar functionality is needed for the RVD database. A custom CAPEC identifier has been created for that purpose. Similarly to the RVD parser, described in the previous subsection, CAPEC identifier utilizes a local instance of the CAPEC catalogue for retrieving information about known attacks based on given known weaknesses. A set of Java classes has been created for storing all the information camming from the CAPEC repository (Figure 7). Due to the lack of space, only the classes able to store the main information of available for a known attack is presented in

Figure 7. As we can see, the CAPEC repository includes an "AttackPatternCatalog", which includes a list of "AttackPatterns". The "AttackPattern" class corresponds to the known vulnerabilities including a lot information such as related weaknesses, related attacks and proposed mitigation actions.

A REST API has been created for inserting the latest version of the CAPEC catalogue, creating a local instance. The corresponding code snippet is depicted in Listing 16.

```
@PostMapping(value = "/capecinsert")
public String capecInsert(@RequestBody Capec capecJson) {
   //Generate the capec database (capecs) with the input from the
capecJson object
   …

   System.out.println("Local CAPEC repository has been updated");
   return "rvdresult";
}
```

Listing 16: REST API for the update of the local version of the CAPEC database

As it can be seen, the API endpoint is http://ipAddress:port/capecinsert and the anticipated body of the request is a CAPEC object, which includes a list of known attacks. The structure of the CAPEC class can be seen in Figure 7. The @PostMapping annotation ensures that HTTP POST requests are mapped onto a specific handler method, the *capecInsert* in this case. The request is expected to have a body in application/json format. The @RequestBody annotation enables the automatic deserialization of the request body onto a Java object. What follows the mapping of the request body to the corresponding Java instance, is the insertion of the latter into the *capecs* array list in the file system. This API endpoint allows for the regular update of the CAPEC local version to include any newly added known attacks.

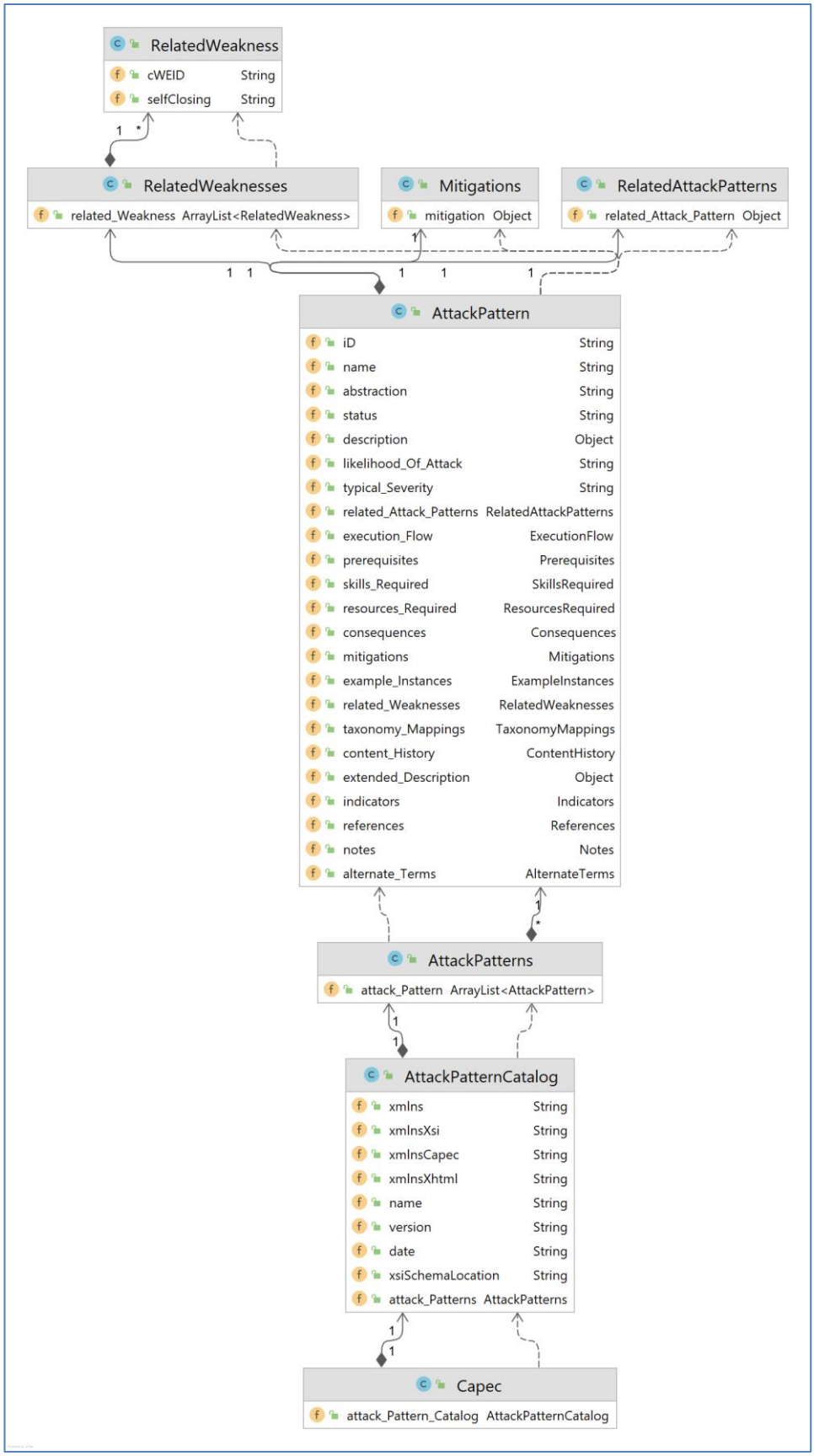

Figure 7: CAPEC classes of the custom CAPEC identifier

One of the desired functionalities of the CAPEC identifier is to be able to discover known attacks based on given CWEs. This list of CWEs is created by the RVD parser we have already described, and includes the known weaknesses related to a given vulnerability. A REST API is available for initiating the process of querying the local CAPEC catalogue instance, and is depicted in Listing 17.

```
@PostMapping(value = "/searchwithcwe")
public String searchWithCwe(@RequestBody ArrayList<String> cweIds) {

   ArrayList<AttackPattern> capecFilteredArraylist = new Ar-
rayList<>();
   //Iterate all cwe to find the capecs
   for (int i = 0; i < cweIds.size(); i++) {
      String tempCWE= cweIds.get(i).substring(4);
      for (int j = 0; j <capecs.size() ; j++) {
         AttackPattern tempCapec = capecs.get(j);
         for (int k = 0; k <temp-
Capec.related_Weaknesses.related_Weakness.size() ; k++) {
            RelatedWeakness tempRelatedWeak-
ness=tempCapec.related_Weaknesses.related_Weakness.get(k);
            if (tempRelatedWeakness.cWEID.equals(tempCWE)){
               capecFilteredArraylist.add(tempCapec);

            }
         }
      }
   }
   System.out.println("The following CAPECs have been identified as
potential attacks related to :" + cveId);
   for (int i = 0; i < capecFilteredArraylist.size(); i++) {
      System.out.print(" "+capecFilteredArraylist.get(i).iD);

   }

   return "rvdresult";
}
```

Listing 17: REST API for querying for CAPECs based on a given CWE list

The API endpoint is http://ipAddress:port/searchwithcwe . The @PostMapping annotation ensures that HTTP POST requests are mapped onto the corresponding "searchWithCwe" handler method. Regarding the logic, the CAPEC catalogue is searched against every incoming CWE. If there is a match with the related CWEs of a known attack, the known attack is stored in the "capecFilteredArraylist" list, creating the output.

The implementation of the CAPEC identifier, similarly to the RVD parser, is an on going work. New functionality will be added based on the emerged project needs of the following months.

2.1.3 Generation of attack trees

The generation of attack trees is separated in two different processes. The first process creates relative small trees/graph of mostly two or three levels incorporating known attacks that have been identified as potential attacks for the target system. The hierarchical classification and the relationships among the attacks that populate the

CAPEC repository allow this first process to produce the aforementioned trees/graphs. The CAPEC hierarchical classification includes different types of relationships between two attack patterns, including *CanFollow* and *CanPrecede*. The former gives information about the attacks that may follow a given attack according to a specific attack pattern. The latter reveals attacks that could have been conducted before a given attack, opening the way for it. Following these relationships, we can create different graphs/trees, one for each possible known attack pattern. An example of such a graph, can be seen in Figure 8.

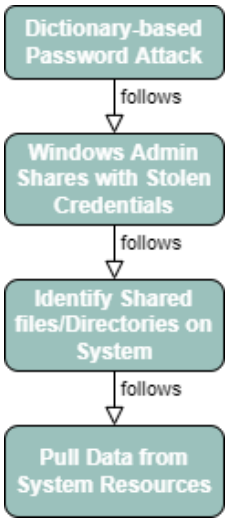

Figure 8: Example graph that can be produced utilizing the CanFollow relationship of CAPEC

This tree has been created based on the CanFollow relationship. The detailed description of this graph can be found in D5.1.

The second process utilizes attack tree templates for the creation of the final attack trees that will describe potential attack scenarios towards the target system.

The attack tree templates are predefined attack trees, whose root is the ultimate goal of the attacker and the leaves are different ways to achieve that goal. Between the goal and the leaves, one can find sub-goals, which describe achievements of the attacker that bring them closer to their goal. Between the goal and the leaves, one can find sub-goals, which describe achievements of the attacker that bring them closer to their goal.

An example template attack tree can be seen in Figure 9. This tree assumes that the robots of the target system make use of ROS messages to communicate with each other. A different template attack tree should be used is another communication protocol was used. The ultimate goal of the attacker in that tree is the robot to crash with a person. There are a lot sub-goals on the way such as to compromise the robot API, to compromise a robot in the target system or to publish arbitrary data to a ROS topic. The leaves are known attacks with CAPEC-IDs. If some or all of these known attacks that can be found in the template's leaves have also be identified as potential attacks by the previous processes, the template itself can be adopted as an attack/fault tree that describes attacks that could be conducted against the system in question.

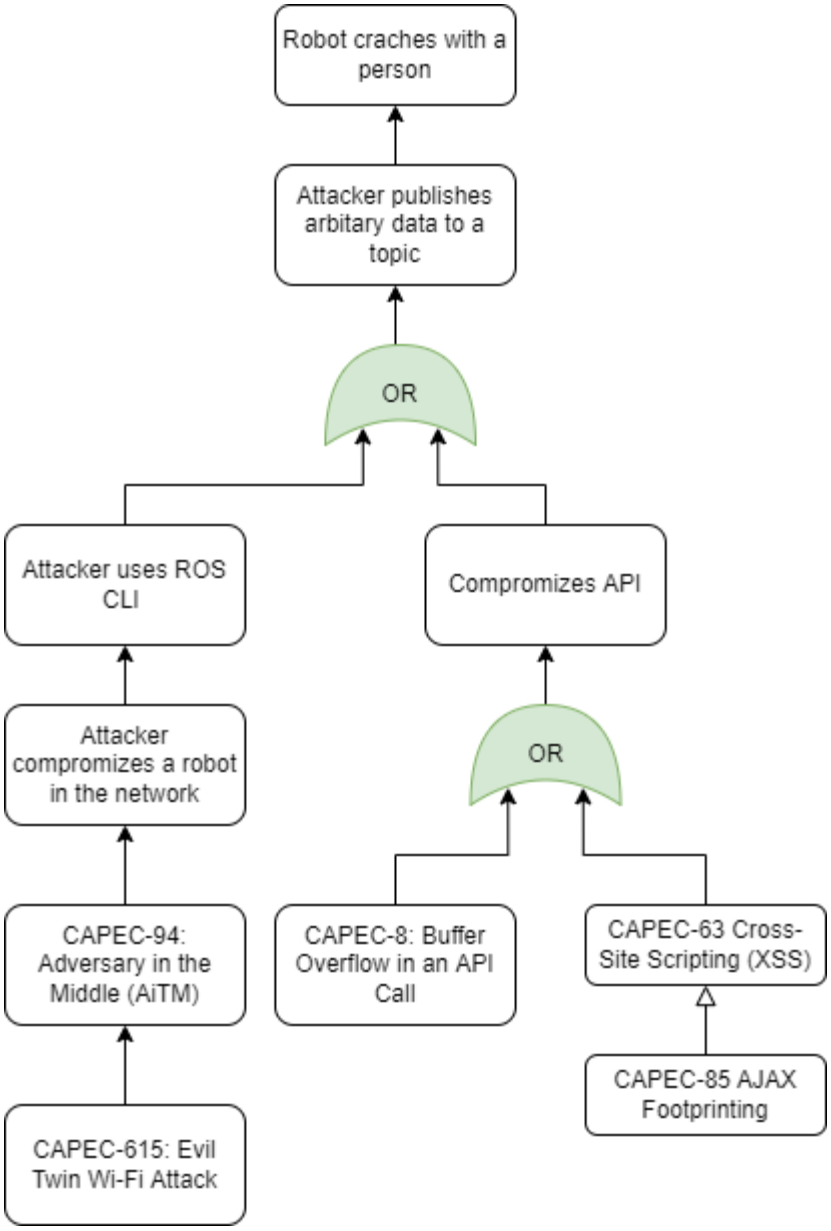

Figure 9: Example template attack tree

2.1.3.1 Techniques and tools

The tools that will be used in this part of the SESAME security assessment are all custom. They are in the initial phase of their implementation and will be reported in details in the next version of this deliverable.

3 RUNTIME SECURITY MONITORING

The SESAME security methodology described in the previous section, allows for the assessment of the security status for a target system, depicting potential attack scenarios in the form of attack/fault trees. These scenarios define the security level of the system in question in every step of a potential attack.

However, additional information is needed for a more complete description of the security status in a given point in time and the proposed mitigation actions. This information is about the actual threats that a system may face and is produced by a runtime security monitoring tool. There is a number of potential security monitoring tools, such as Intrusion Detection System (IDS), Anti-Virus and Breach detection systems (BDS) that can be used for that purpose.

3.1 INTRUSION DETECTION

An IDS is a security monitoring system that is able to detect suspicious activities and generates alerts when a detection occurs. Produced alerts can be communicated to a system administrator or an incident responder for further investigation and defining of mitigation actions.

IDSs can be implemented as software or hardware. There are three main types of IDS: host based, network based and application based. Host based IDSs are deployed on hosts and monitor incoming and outgoing traffic. Such hosts are systems that carry sensitive data, cannot be patched or have other reasons for extra security measures. Network based IDSs are placed in key points in networks, such as the gateway, and filter the traffic that is exchanged among the different devices of the network. Application based IDSs try to identify intrusions by filtering traffic on application specific protocols, such as SQL protocol.

Moreover, there are two different types of detection that is used by IDS: signature based and anomaly detection based. The former has an advantage regarding the known attacks, since they can be detected with great precision. The way it is done is by recognizing specific patterns in the headers or body of traffic packets. On the other hand, anomaly detection based IDSs are better at discovering unknown attacks. Said attacks alter the traffic making it different from the norm. Machine learning techniques, such as Tree classifiers, Bayesian Clustering, Deep Learning are used for the detection of unknown attacks.

3.1.1 Techniques and tools

The final choice to monitor the network incoming packets for malicious ones is an IDS. Regarding the type of detection, a signature based IDS is used, where specific patterns are recognized in the headers or body of traffic packets.

Our familiarity with Snort, the experience we have using it for the creation of components as part of other systems and its wide-adoption drove us to its selection. Such a tool will allow for the detection of attacks towards the system in question and the creation of corresponding alerts.

Snort is among the most known open source IDS. It performs real-time traffic analysis and packet logging. It uses sets of rules for the definition of malicious network activity.

Packets that match against said rules generate alerts for users. Generally, the Snort IDS has been in development since 1998 by Sourcefire. Nowadays, it has become a standard for IDSs and has been broadly used in research studies. It has a single-threaded architecture, which uses the TCP/IP stack to capture and inspect network packets (header and body). It makes use of a set of rules that define malicious network activity. Based on this set of rules Snort identifies malicious packets, and sends alerts to users. The corresponding rule language is so expressive that allows the description of the network traffic to be collected and the remedy action that follows when malicious packets are detected. Snort can be configured as a full network IPS solution that monitors network activity and detects and blocks potential attack vectors. Satisfied rules create alerts that are logged, and based on these alerts a report is created. Snort is well suited to fill lightweight IDS requirements. It takes a few minutes to be compiled, configured and set. The fact that there is no need for monitoring or administrative maintenance makes Snort ideal for integral part of many network security infrastructures.

Snort rules are divided into two logical sections, the rule header and the rule options as depicted in Figure 10. The rule header contains the rule's action, protocol, source and destination IP addresses and netmasks, and the source and destination ports information. On the other hand, rule option section contains alert messages and information on which parts of the packet should be inspected to determine if the rule action should be taken.

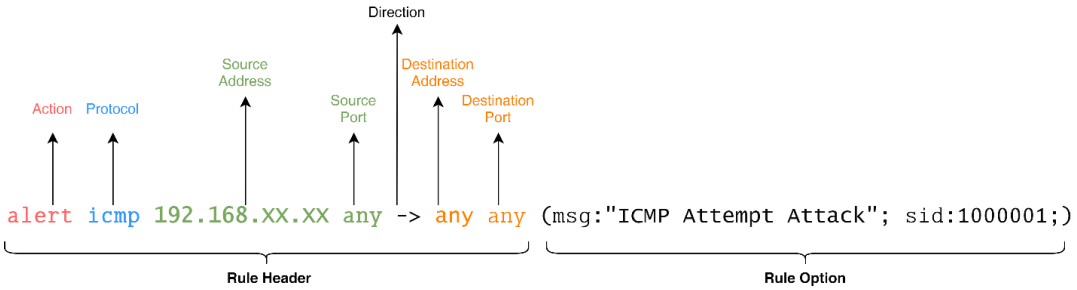

Figure 10: Sample Snort Rule

Regarding the Rule Header:

- Action: The action tells Snort what to do (alert, log, pass) when it finds a packet that matches the rule criteria.
- Protocol: The protocol references one of the four protocols that Snort currently analyses for suspicious behaviour – TCP, UDP, ICMP, and IP.
- Source Address and Port: This part of the Rule Header gives information about the IP address and port of the source.
- Direction: The direction operator indicates the orientation, or direction, of the traffic that the rule applies to. There is also a bidirectional operator (<>), which handy for recording/analysing both sides of a conversation.

Regarding the Rule Option:

- msg: The msg rule option tells the logging and alerting engine the message to print along with a packet dump or to an alert.
- sid: The sid identifies uniquely each Snort rule.
- classtype: The classtype keyword is used to categorize a rule as detecting an attack that is part of a more general type of attack class. Snort provides a default set of attack classes that are used by the default set of rules it provides (not shown in Figure 10).
- priority: The priority tag assigns a severity level to rules. A classtype rule assigns a default priority (defined by the config classification option) that may be overridden with a priority rule (not shown in Figure 10).

The installation of Snort includes a number of different steps, such as i) the installation of the required dependencies, ii) installation of Snort itself, iii) configuration of network interface cards, and iv) installation of rulesets.

Regarding the first step, the corresponding commands are depicted in Listing 18. The most important parts are the installation of Snort Data Acquisition library (LibDAQ) and the Tcmalloc, which will optimize memory allocation and provide better memory usage.

```
~/# sudo apt install build-essential libpcap-dev libpcre3-dev libnet1-
dev zlib1g-dev luajit hwloc libdnet-dev libdumbnet-dev bison flex
liblzma-dev openssl libssl-dev pkg-config libhwloc-dev cmake cpputest
libsqlite3-dev uuid-dev libcmocka-dev libnetfilter-queue-dev libmnl-
dev autotools-dev libluajit-5.1-dev libunwind-dev

~/# mkdir snort-source-files

~/# cd snort-source-files



~/# git clone https://github.com/snort3/libdaq.git

~/# cd libdaq

~/# ./bootstrap

~/# ./configure

~/# make

~/# make install



~/# cd ../

~/#  wget  https://github.com/gperftools/gperftools/releases/download/
gperftool s-2.9/gperftools-2.9.tar.gz
```

```
~/# tar xzf gperftools-2.9.tar.gz

~/# cd gperftools-2.9/

~/# ./configure

~/# make

~/# make install
```

Listing 18: Snort install dependencies commands

The commands for the installation of Snort itself are included in Listing 19. According to them, we need to clone Snort 3 official GitHub repository, configure and enable tcmalloc, and install Snort 3 using make and make install commands.

```
~/# git clone git://github.com/snortadmin/snort3.git

~/# snort3/

~/# ./configure_cmake.sh --prefix=/usr/local --enable-tcmalloc

~/# cd build

~/# make

~/# make install

~/# sudo ldconfig
```

Listing 19: Snort installation commands

The configuration of the network interface cards can be done with the commands in Listing 20. These commands enable the promiscuous mode of the interface that Snort listens to, and disable the Offloading interface to prevent the truncate of large packets.

```
~/# ip link set dev eh0 promisc on

~/# ethtool -k eth0 | grep receive-offload
```

Listing 20: Snort network interface card configuration commands

The last step of the Snort installation includes the installation of the Snort rules. Commands in Listing 21, show how the community rulesets of Snort can be installed. According to these commands, we need to create a directory for the rules, download the ruleset from the official download page, and extract the rules to the dedicated directory.

```
~/# mkdir /usr/local/etc/rules

~/#  wget  https://www.snort.org/downloads/community/snort3-community-
rules.tar.gz

~/# tar xzf snort3-community-rules.tar.gz -C /usr/local/etc/rules/
```

Listing 21: Snort community rulesets installation commands

4 INITIAL INTEGRATION – TEST SCENARIO

In this section, we will present an initial integration of the tools described in the previous subsections of this document. The integration of all the aforementioned tools will create an automated pipeline, which will lead to the assessment of the security status of a given system. The first pipeline functionality will be the identification of the system assets. The can be done either by the system administrator or by automated tools such as **openVAS** and **w3af**. The next functionality will be the identification of system vulnerabilities. **RVD parser** and **cve-search** are responsible for this task. The identification of vulnerabilities will then lead to the specification of potential attacks, a task undertaken by **CAPEC identifier**.

The target system consists of fleet of drones and a ground control station. Drones communicate, through a wireless connection, with the control station that controls their actions and renders their status. This system setup was chosen since it is very similar with some of the SESAME use cases. In the viticulture and the drone emergency surveillance cases, a number of drones communicate with the corresponding ground control station. The ground control station assigns tasks to the drones, such as a spraying root or the surveillance of a specific area. Moreover, data are also communicated from the drones back to the ground control station for their storage and visualization.

The communication that takes place among the system components makes use of the MAVLink protocol. Its design pattern is a combination of publish-subscribe and point-to-point for data streams and mission/parameter communication respectively. MAVLink faces security problems due to lack of confidentiality and authentication mechanisms. As a result, the drones exchange state and control messages with the control station through an unauthenticated, unencrypted channel.

The system administrator, knowing the security flaws of the MAVLink communication protocol, wants to use the SESAME security assessment process to identify potential attacks. Such attacks could be conducted by an adversary, taking advantage of the vulnerabilities that are related to the MAVLink protocol.

The first two actions that need to be taken are the creation of the local instances of the RVD and CAPEC repositories. As it was described in subsections 2.1.1.1 and 2.1.2.1 respectively, we download both these catalogues locally, to allow for faster requests.

The creation of the RVD local instance requires an HTTP POST request to the http://ipAddress:port/rvdinsert endpoint. The body of the request should include all the robot vulnerabilities mentioned in the RVD repository. Listing 22 depicts one of these vulnerabilities, which is associated with the MAVLink protocol.

```
[{
  "id": "3316",
  "title": "RVD#3316: No authentication in MAVLink protocol",
  "type": "vulnerability",
  "description": "The Micro Air Vehicle Link (MAVLink) protocol pre-
sents no authentication mechanism on its version 1.0 (nor authoriza-
tion) whichs leads to a variety of attacks including identity spoof-
ing, unauthorized access, PITM attacks and more. According to litera-
```

```
ture, version 2.0 optionally allows for package signing which miti-
gates this flaw. Another source mentions that MAVLink 2.0  only pro-
vides a simple authentication system based on HMAC. This implies that
the flying system overall should add the same symmetric key into all
devices of network. If not the case, this may cause a security issue,
that if one of the devices and its symmetric key are compromised, the
whole authentication system is not reliable.",
  "cwe": "CWE-306",
  "cve": "CVE-2020-10282",
  "keywords": "['MAVLink', 'v1.0', 'v2.0', 'PX4', 'Ardupilot']",
  "system": "MAVLink: v1.0",
  "vendor": "PX4",
  "severity": {
    "severity-description:critical": {}
  },
  "rvss-score:9.6": {},
  "rvss-vector:RVSS:1.0/AV:AN/AC:L/PR:N/UI:N/S:U/Y:T/C:H/I:H/A:H/H:U":
{},
  "cvss-score:9.8": {},
  "cvss-vector:CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H": {},
  "links": "['https://arxiv.org/abs/1906.10641',
'https://arxiv.org/abs/1905.00265',
'https://ieeexplore.ieee.org/document/8425627',
'https://www.researchgate.net/publication/335973981_Assessing_and_Expl
oiting_Security_Vulnerabilities_of_Unmanned_Aerial_Vehicles',
'https://link.springer.com/chapter/10.1007/978-981-13-8406-6_66',
'https://www.esat.kuleuven.be/cosic/publications/article-2667.pdf',
'https://www.usenix.org/conference/usenixsecurity19/presentation/kim',
'https://docs.google.com/document/d/1ETle6qQRcaNWAmpG2wz0oOpFKSF_bcTmY
MQvtTGI8ns/edit',
'https://docs.google.com/document/d/1upZ_KnEgK3Hk1j0DfSHl9AdKFMoSqkAQV
eK8LsngvEU/edit',
'https://docs.google.com/document/d/1XtbD0ORNkhZ8eKrsbSIZNLyg9sFRXMXbs
R2mp37KbIg/edit',
'https://github.com/PX4/Firmware/issues/13538#issuecomment-574281772',
'https://github.com/rligocki/Diploma_thesis_px4']",
  "flaw": {
    "phase": "unknown",
    "specificity": "subject-specific",
    "architectural-location": "platform code",
    "application": "Flying vehicles and/or others using MAVLink proto-
col.",
    "subsystem": "communication",
    "package": "N/A",
    "languages": "C, C++",
    "date-detected": "None",
    "detected-by": "None",
    "detected-by-method": "testing",
```

```
    "date-reported": "2020-06-30",
    "reported-by": "Victor Mayoral Vilches (Alias Robotics)",
    "reported-by-relationship": "security researcher",
    "issue": "https://github.com/aliasrobotics/RVD/issues/3316",
    "reproducibility": "always",
    "trace": "N/A",
    "reproduction": "N/A",
    "reproduction-image": "N/A"
  },
  "exploitation": {
    "description": "Not available",
    "exploitation-image": "Not available",
    "exploitation-vector": "Not available",
    "exploitation-recipe": ""
  },
  "mitigation": {
    "description": "MAVLink 2.0 includes signing capabilities which
mitigate this issue. Signatures seem to be optional for backwards com-
patibility and https://arxiv.org/abs/1906.10641 confirms this matter.
Proper mitigation should enforce signatures.",
    "pull-request": "N/A"
  }
}]
```


Listing 22: MAVLink robot vulnerability extracted from RVD database in JSON format

A plethora of information is made available for each vulnerability. One of the most important information is the related weaknesses, cwe in the listing. This piece of information will lead us to the potential attacks. The request for the creation of the local RVD instance inserts all the RVD vulnerabilities in the local memory of the RVD parser, and makes them available for parsing.

The output of this request is shown in the terminal with a message that informs about the updated status of the RVD local repository (Figure 11).

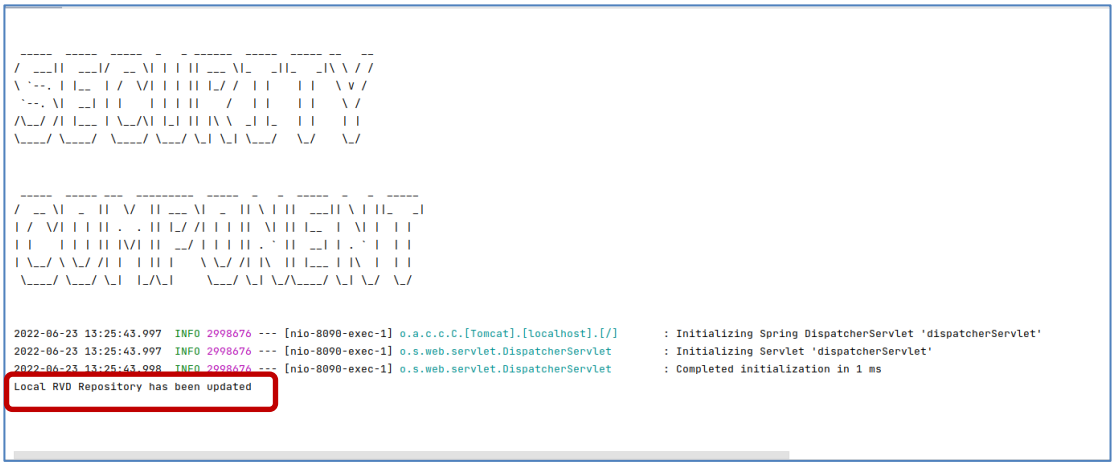

Figure 11: Response of the request for the creation of the local RVD repository

Regarding the CAPEC local instance creation, a corresponding request must be made to the http://ipAddress:port/capecinsert API endpoint. The body of this request must include all the known attacks that have been registered in the CAPEC repository. Listing 23 depicts, as an example, the CAPEC-12 known attack as it is included in the CAPEC repository. A list of such attack description is included in the body of the request.


```
{
    "Attack_Pattern_Catalog": {
        "-xmlns": "http://capec.mitre.org/capec-3",
        "-xmlns:xsi": "http://www.w3.org/2001/XMLSchema-instance",
        "-xmlns:capec": "http://capec.mitre.org/capec-3",
        "-xmlns:xhtml": "http://www.w3.org/1999/xhtml",
        "-Name": "CAPEC",
        "-Version": "3.7",
        "-Date": "2022-02-22",
        "-xsi:schemaLocation": "http://capec.mitre.org/capec-3
http://capec.mitre.org/data/xsd/ap_schema_v3.5.xsd",
        "Attack_Patterns": {
            "Attack_Pattern": [
                {
                    "-ID": "12",
                    "-Name": "Choosing Message Identifier",
                    "-Abstraction": "Standard",
                    "-Status": "Draft",
                    "Description": "This pattern of attack is ...
identifier to more a privileged one.",
                    "Likelihood_Of_Attack": "High",
                    "Typical_Severity": "High",
                    "Related_Attack_Patterns": {
                        "Related_Attack_Pattern": [
                            {
                                "-Nature": "PeerOf",
                                "-CAPEC_ID": "21",
                                "-self-closing": "true"
                            },
                            {
                                "-Nature": "ChildOf",
                                "-CAPEC_ID": "216",
                                "-self-closing": "true"
                            }
                        ]
                    },
                    "Execution_Flow": {},
                    "Prerequisites": {},
                    "Skills_Required": {},
                    "Resources_Required": {},
                    "Consequences": {},
```

```
                    "Mitigations": {},
                    "Example_Instances": {},
                    "Related_Weaknesses": {
                        "Related_Weakness": [
                            {
                                "-CWE_ID": "201",
                                "-self-closing": "true"
                            },
                            {
                                "-CWE_ID": "306",
                                "-self-closing": "true"
                            }
                        ]
                    },
                    "Content_History": {}
                }
            ]
        }
    }
}
```


Listing 23: CAPEC-12 known attack extracted from the CAPEC repository in JSON format

A lot of information has been missing from the listing above, due to lack of space. However, the important information for the CAPEC identifier is the "Related_Weaknesses" element. That element includes all the weaknesses (CWEs) that related to a specific attack. In the listing above, we see that CAPEC-12 is related with the weaknesses with ids 201 and 306.

The output of this request is shown in the terminal with a message that informs about the updated status of the CAPEC local repository (Figure 12).

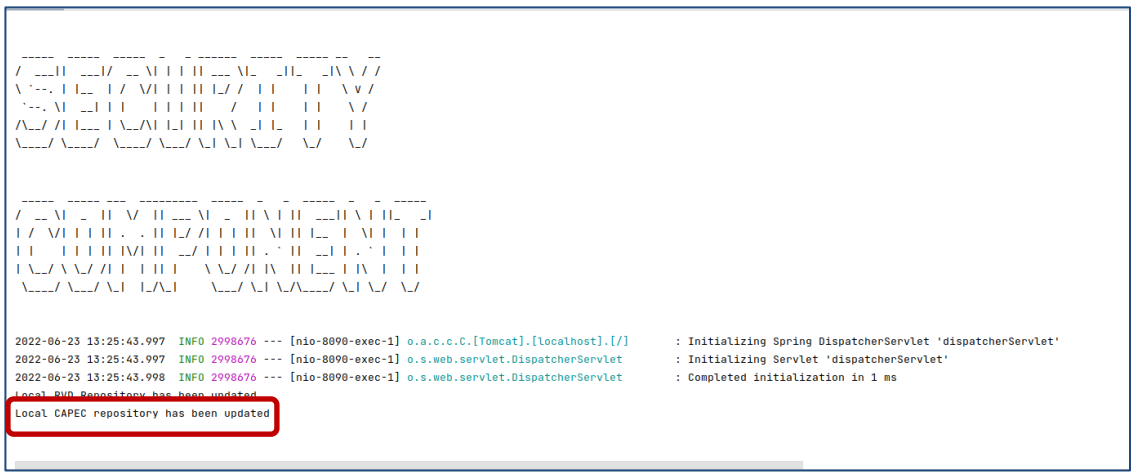

Figure 12: Response of the request for the creation of the local CAPEC repository

Now that both security repositories are available, RVD parser and CAPEC identifier can be used for the identification of the potential system attacks. The triggering of this

process is another HTTP request to the http://ipAddress:port/searchwithcve endpoint. We mentioned that, according to the test scenario, the system administrator wants to find all the potential attacks that can be conducted due to the MAVLink vulnerability. The body of the aforementioned request must include the CVE-ID of the MAVLink vulnerability, which is CVE-2020-10282.

As soon as the request reaches the API endpoint, the RVD parser searches the local instance of the RVD repository against the incoming CVE-ID, trying to find all the related weaknesses (CWEs). For each of the discovered weakness the CAPEC identifier parses all the known attacks in the local CAPEC repository and identifies those that are related with the specific CWE. The output is a list of all the discovered known attacks. Figure 13 depicts the response of the request. As it can be seen, known attacks with CAPEC-IDs 12, 166, 36, 62 are identified due to the CVE-2020-10282 vulnerability.

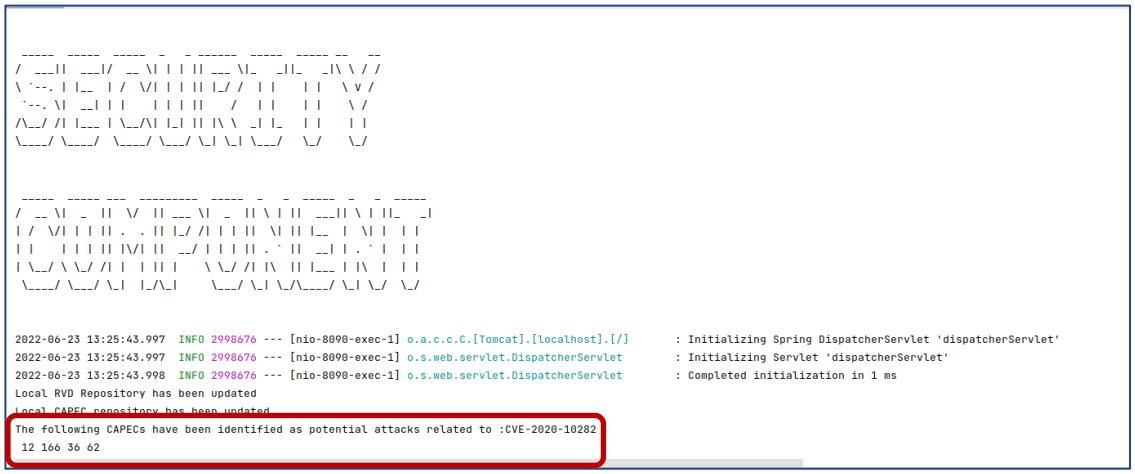

Figure 13: Response of the request for the known attacks related to a specific CVE-ID

5 CONCLUSIONS

This deliverable presents to the reader the techniques and tools that are adopted for the realization of the SESAME security assessment process. A set of opensource and custom tools is described, aiming to implement the functionalities of each of the individual subprocesses of the security assessment process. The described tools are mainly used at the design time, however, one of them is the authors choice for the runtime security monitoring system.

The custom tools are described in this deliverable are still under development since not all of their functionalities have been delivered. Moreover, additional features arise every time a new use case tries to adopt the proposed methodology. The intended functionality and all the new features will be described in the next version of this deliverable.

Moreover, intending to show that the presented tools are not only standalone solutions but they can be integrated to provide a more comprehensive outcome, a first integration of some of them is presented. The chosen testing scenario includes a system with architecture that is very similar with three of the project use cases. In that way, we intent to show that the proposed security assessment methodology can be materialized and offer tangible outcomes.