



UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale in Ingegneria Informatica

***A search engine on information
delivered by OSINT sources to
support Vulnerability Assessment***

Anno Accademico 2020/2021

Relatore

Ch.mo prof. Simon Pietro Romano

Correlatore

Pietro Petrella, Accenture Application Security

Candidato

Antonio Forte

matr. M63000845

Al mio Dio.

Contents

Contents	ii
List of Figures	v
List of Tables	vi
Code Listings	vii
Acronyms	viii
Abstract	xi
1 Vulnerability Assessment: Basic Concepts	1
1.1 Asset	1
1.2 Common Platform Enumeration (CPE)	2
1.3 Vulnerability	6
1.4 Common Vulnerabilities and Exposures (CVE)	8
1.5 Common Weakness Enumeration (CWE)	9
1.6 Common Vulnerability Scoring System (CVSS)	11
1.7 Severity	12
1.8 Exploit	13

2	Vulnerability Assessment: State of the Art	14
2.1	What is Vulnerability Assessment?	14
2.1.1	Vulnerability Assessment vs Penetration Testing	16
2.2	Vulnerability Assessment Process	17
2.3	Vulnerability Assessment Types	19
2.3.1	Host Assessment	20
2.3.2	Network Assessment	21
2.4	Vulnerability Assessment Approaches	22
2.4.1	Administrative Approach	22
2.4.2	Outsider Approach	24
2.5	Vulnerability Assessment Search Engines	25
2.5.1	NVD Search	25
2.5.2	CVE List Search	28
2.5.3	CVE Details Search	29
2.5.4	Vulmon Search	30
2.5.5	EDB Search	32
2.5.6	VulDB Search	33
2.5.7	Vulnerability & Exploit Database Search . . .	35
2.5.8	Tenable CVE Search	37
3	WISE: Architecture	39
3.1	High Level Architecture	39
3.2	Architecture Components	41
3.2.1	Data Store Components	41
3.2.2	Interface Components	43
3.2.3	Search Engine Core	44

3.2.4	Reporting Results Components	46
4	VISE: Implementation View	48
4.1	Microservices Configuration	48
4.1.1	The docker-compose File	49
4.2	The Search Functionality	52
4.2.1	Activity Diagram	53
4.2.2	Code Focus	55
5	VISE: User's Manual	61
5.1	Installation & Configuration	61
5.2	Example of use	64
6	Conclusions and Future Work	73
	Bibliography	75
	Sitography	77
	Acknowledgements	82

List of Figures

1.1	Principal types of IT assets [3]	3
1.2	CPE 2.3 stack [4]	4
3.1	WISE: High level architecture	40
4.1	Search function: Activity diagram	54
5.1	WISE: Home page	66
5.2	WISE: Form page	67
5.3	WISE: Searching card page	67
5.4	WISE: Searching card sample	68
5.5	WISE: Results page	68
5.6	WISE: Vulnerability summary dashboard	69
5.7	WISE: Vulnerability technical description dashboard	70
5.8	WISE: Exploit view dashboard	71
5.9	WISE: CLI results report	72

List of Tables

1.1	Severity - Base Score range mappings	13
-----	--	----

Code Listings

4.1	The docker-compose.yml file	50
4.2	The searchCPEs() definition	56
4.3	The search_CPEs() definition	56
4.4	The search_CVEs_from_single_limit() definition . .	57
4.5	The search_CVEs_from_interval() definition	58
4.6	The search_CVEs() definition	59
5.1	Install python3-pip & python3-venv	62
5.2	Install Flask	62
5.3	Install ExploitDB	63
5.4	Install Docker	63
5.5	Index cpe-index & cve-index in Elasticsearch	64
5.6	Start VISE	65
5.7	Start a new search from CLI	72

Acronyms

API Application Programming Interface.

CERT Computer Emergency Response Team.

CLI Command Line Interface.

CNA CVE Numbering Authority.

CPE Common Platform Enumeration.

CSV Comma-Separated Values.

CVE Common Vulnerabilities and Exposures.

CVE ID CVE IDentifier.

CVSS Common Vulnerability Scoring System.

CWE Common Weakness Enumeration.

CWE ID CWE IDentifier.

DoS Denial of Service.

EDB Exploit Database.

ELK Elasticsearch, Logstash, Kibana.

GUI Graphical User Interface.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

IAVM Information Assurance Vulnerability Management.

IDS Intrusion Detection System.

IETF Internet Engineering Task Force.

IP Internet Protocol.

ISO/IEC JTC 1 Joint Technical Committee of the International Organization for Standardization and the International Electrotechnical Commission.

IT Information Technology.

JSON JavaScript Object Notation.

NIST National Institute of Standards and Technology.

NVD National Vulnerability Database.

OpenVAS Open Vulnerability Assessment Scanner.

OSINT Open Source INTelligence.

OVAL Open Vulnerability and Assessment Language.

PoC Proof of Concept.

RFC Request for Comments.

RSS Real Simple Syndication.

SIG FIRST Special Interest Groups.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

US-CERT United States Computer Emergency Readiness Team.

VAPT Vulnerability Assessment and Penetration Testing.

WISE Vulnerability Information Search Engine.

VulDB Vulnerability Database.

WFN Well-Formed CPE Name.

WSGI Web Server Gateway Interface.

XML Extensible Markup Language.

XSS Cross-Site Scripting.

YAML YAML Ain't Markup Language.

Abstract

Today we have reached such a level of humus between our lives and information technology that it is practically unthinkable to be able to lead one's life doing completely without information systems. This is the reason why these assets become the main target of attack by those who are preparing to commit acts of crime in the online world. Those involved in Cyber Defence know that to deal with the so-called cyber pirates, a reactive approach focused, for example, on the use of anti-virus and firewalls is no longer enough, but proactive approaches are also needed. In this context, tools for the management and assessment of vulnerabilities are placed, which look for weaknesses in Information Technology (IT) systems so that, being known, they can be appropriately managed in order to be able to mitigate or avert any future attack that founds its own base on exploiting those vulnerabilities.

In the context of vulnerability assessment, Vulnerability Information Search Engine (VISE) [1] was born. In the first place, VISE deals with aggregating heterogeneous information on vulnerabilities received from Open Source INTelligence (OSINT) sources, after which given as

input a list of IT systems, software and packages it is able to perform an efficient search for information on vulnerabilities associated with the aforementioned and automatically present highly structured reports (even at different levels of detail).

To understand the context in which we move we will first have an introductory chapter necessary to address basic vulnerability assessment concepts to which we will refer extensively during our discussion.

In the second chapter, we will focus on understanding what is the state of the art regarding the world of vulnerability assessment, and then we will focus our attention on the main search engines that exist in support of vulnerability assessment.

In the third and fourth chapters we will respectively make a focus on the design and implementation view of VISE.

In the last chapter an example of operation is presented in order to evaluate the potential of the proposed solution. The description is carried out with the aid of screenshots.

Chapter 1

Vulnerability Assessment: Basic Concepts

In this chapter we introduce the most fundamental concepts of vulnerability assessment. We will cover concepts such as: asset, Common Platform Enumeration (CPE), vulnerability, Common Vulnerabilities and Exposures (CVE), Common Weakness Enumeration (CWE), Common Vulnerability Scoring System (CVSS), severity and exploit.

1.1 Asset

One problem we have in the world of vulnerability assessment is that, as we'll see, its first phase consists in identifying and cataloging all enterprises assets. The basic problem is that even in the literature, standards, frameworks, and regulations there is no clear definition of

what an asset is. Aware of this problem, in the paper [2] the authors propose a conceptual, generic working definition for IT assets to assist with asset identification in the identification phase of risk analysis/assessment.

The definition of IT asset they give is:

"a resource (definite or evolving) controlled (or controllable) by an entity/organization having tangible and intangible parts (such as hardware and/or software and/or data/information and/or derivatives and/or other parts) with a definable cost or value or impact level within the digital channel that has the potential to bring or produce economic benefit or liability (risk, reputation loss, and so on) to an organization by reason of its information value processing, transmitting or storing capability, and/or performing of any other IT and/or business function(s)."

As for the classification of asset types, the principal ones are shown in Figure 1.1, taken from the ISO/IEC 19770-1:2017(en) standard by the Joint Technical Committee of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC 1).

1.2 Common Platform Enumeration (CPE)

CPE is a standardized method to identify classes (and not specific instances) of enterprise's computing assets, be them: applications, op-

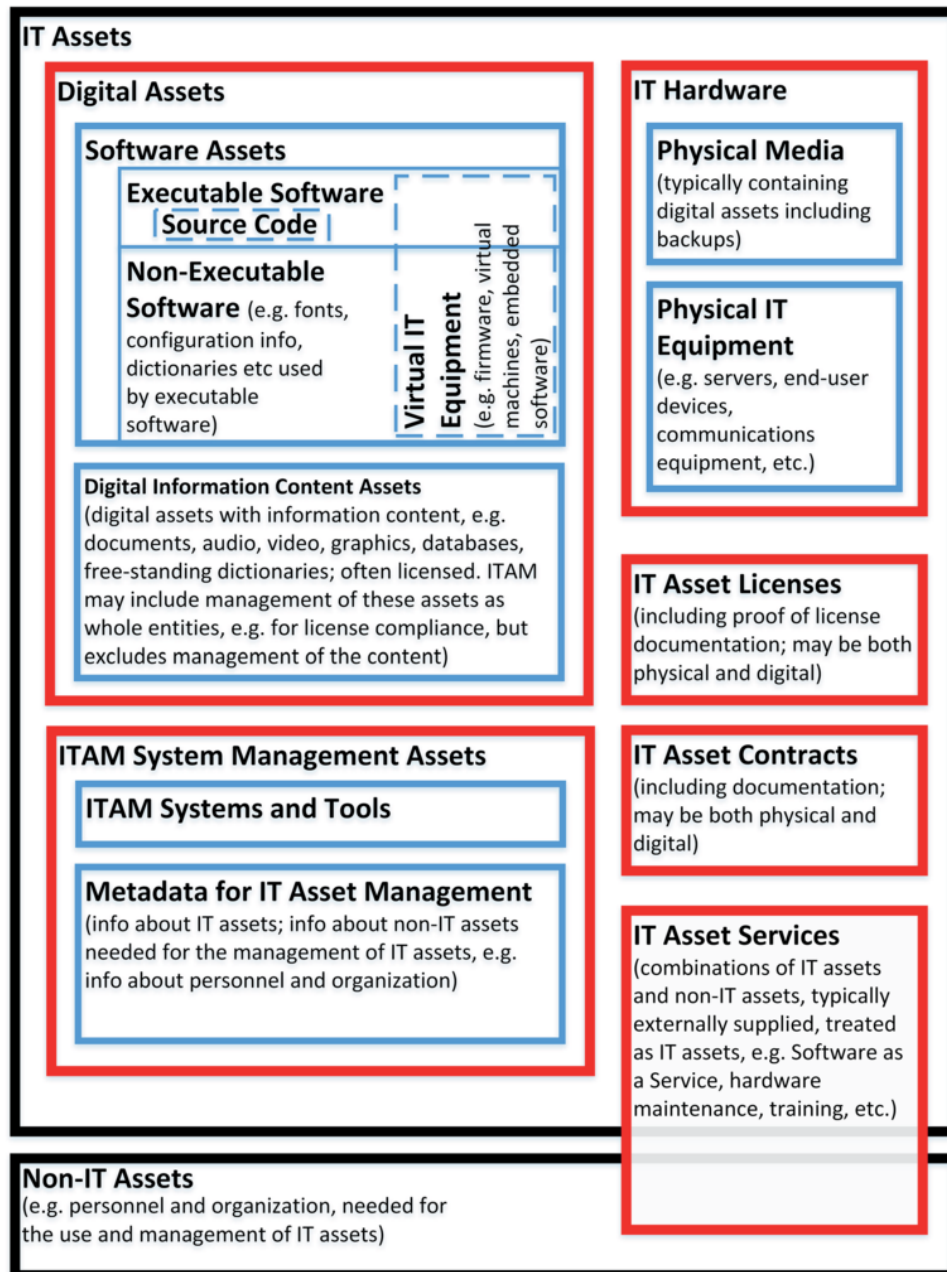


Figure 1.1: Principal types of IT assets [3]

erating systems, or hardware devices. CPE can be used as a standardized information for making fully or partially automated decisions regarding the assets.

Nowadays, two versions of CPE exist.

The current version of CPE is 2.3. It is defined through a set of

specifications in a stack-based model, shown in Figure 1.2:

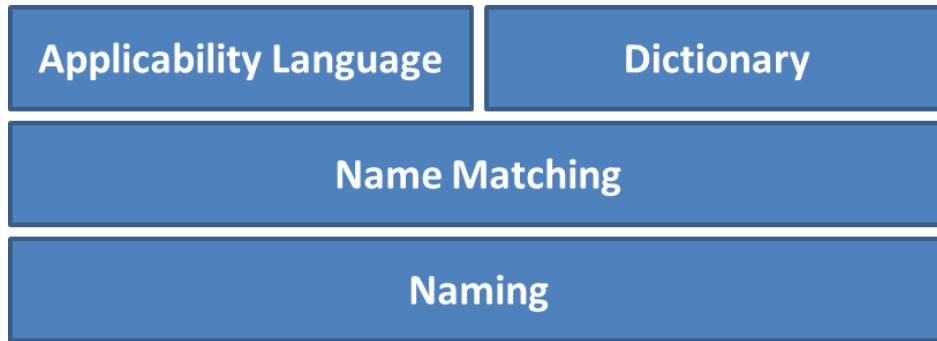


Figure 1.2: CPE 2.3 stack [4]

For our purposes, the *Dictionary* and *Naming* layers are of greatest interest.

The first defines a standardized method for creating and managing CPE dictionaries. A dictionary is a repository of well-formed CPE names and associated metadata for them. Each CPE name in the dictionary identifies a single class of IT products in the world.

On the contrary the *Naming* layer defines standardized methods for assigning names to IT product classes. One of them is Well-Formed CPE Name (WFN), the foundational logical construct of CPE.

WFNs are lists of attribute-value pairs enclosed in square brackets, prefixed with the string "*wfn*":

$$wfn:[a1=v1, a2=v2, ..., an=vn]$$

An attribute-value pair is a tuple $a=v$ in which a is an alphanumeric label and represents an attribute of some entity, and v is his assigned value. The only attributes permitted in a WFN attribute-value pair are: *part*, *vendor*, *product*, *version*, *update*, *edition*, *language*,

sw_edition, target_sw, target_hw, other.

The *Naming* layer also defines procedures for binding WFNs to machine-readable encodings and unbinding them back to WFNs.

The currently supported bindings are:

- Uniform Resource Identifier (URI) binding, that is included in both CPE 2.2 and also CPE 2.3 for backward compatibility
- formatted string binding, that has a different syntax than the URI binding and supports additional product attributes

We focus on formatted string binding because it is the innovation of CPE 2.3 version and it is more flexible with respect to URI binding because it relaxes the requirements that typically apply to URIs. For these reasons formatted string binding is also what we use in VISE to identify product classes.

The formatted string binds the attributes in a WFN in a fixed order prefixed with the string "*cpe:2.3:*":

*cpe:2.3:part:vendor:product:version:update:edition:language:
sw_edition:target_sw:target_hw:other*

An example of formatted string binding is shown below:

cpe:2.3:a:1password:1password:-::-:~*:~*:android:~*:~**

In a formatted string binding, the *** represents the logical value *ANY*, and the *-* represents the logical value *NA*. [5]

1.3 Vulnerability

Regarding vulnerabilities, there are different definitions provided by several standards. By way of example, we report below the definition given in Request for Comments (RFC) 4949 [6] by the Internet Engineering Task Force (IETF):

"a flaw or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's security policy."

Therefore a vulnerability can be introduced at one of the following phases of the asset life cycle:

- design or specification phase: for example the asset was not designed from the foundation to be secure (security by design)
- implementation phase
- operational and management phase: vulnerability is introduced in the deployment phase

As for the classification of vulnerability types, in ISO/IEC 27005:2008 standard [7] they are classified according to the asset class they are related to (for completeness, the corresponding asset classification of Figure 1.1 in round brackets):

- hardware (IT Hardware):
 - susceptibility to humidity or dust

- susceptibility to unprotected storage
- age-based wear that causes failure
- over-heating
- software (Software Assets):
 - insufficient testing
 - insecure coding
 - lack of audit trail
 - design flaw
- network (Physical IT Equipment):
 - unprotected communication lines (e.g., lack of cryptography)
 - insecure network architecture
- personnel (Non-IT Assets):
 - inadequate recruiting process
 - inadequate security awareness
 - insider threat
- physical site (Non-IT Assets):
 - area subject to natural disasters (e.g., floods, earthquakes)
 - interruption to power sources

- organizational (Non-IT Assets):
 - lack of regular audits
 - lack of continuity plans
 - lack of security

1.4 Common Vulnerabilities and Exposures (CVE)

As the official reference itself by MITRE Corporation (MITRE) points [8], the term CVE alone is ambiguous. So, MITRE gives the following definitions that associate an additional word to the aforementioned term to guarantee a specific meaning:

- CVE Program, that is the international effort to identify, define, and catalog publicly disclosed cybersecurity vulnerabilities
- CVE Identifier (CVE ID), that is, a unique alphanumeric identifier assigned by the CVE Program. Each identifier is associated to a specific vulnerability. The explicit and punctual identification of vulnerabilities, obtained thanks to CVE IDs, allows the discussion and sharing of this information, without ambiguity
- CVE Numbering Authority (CNA), that is an organization participating in the CVE Program responsible for the assignment

of CVE IDs to vulnerabilities, and for publishing information about the single vulnerability in the associated CVE Record. Each CNA has a specific set of hardware, software, or services for which it is responsible for vulnerability identification and publishing

- CVE Record, that is the descriptive data about a vulnerability associated with a CVE ID provided by a CNA
- CVE List, that is the catalog of all CVE Records. The CVE List feeds National Vulnerability Database (NVD) that, starting from this information, adds others (e.g. severity score, and impact ratings) to increase the level of knowledge of these vulnerabilities [9]

1.5 Common Weakness Enumeration (CWE)

CWE is a community-developed list of CWE entries. Nowadays, each entry is a specific type of software or hardware weakness (support for hardware weaknesses was added in 2020).

An helpful feature of CWE List is the possibility of being able to approach it through distinct viewpoints. Each viewpoint can be useful for a specific need.

CWE is currently maintained by MITRE and its main goal is to stop vulnerabilities before products are delivered. It makes so by ap-

propriately educating software and hardware architects, designers and programmers [10].

All individual CWEs are held within a hierarchical structure. In particular, CWEs located at higher levels of the structure provide a broad overview of a vulnerability type and can have many CWE children associated with them, CWEs at deeper levels in the structure provide a finer granularity and usually have fewer or no CWE children.

A single CWE entry includes the following information [11]:

- CWE Identifier number/name of the weakness type
- description of the type
- alternate terms for the weakness
- description of the applicable platforms (languages, operating systems, architectures, and technologies in which the weakness could appear)
- description of how the weakness may be introduced
- likelihood of exploit for the weakness
- description of the consequences of the exploit
- potential mitigations
- node relationship information

- source taxonomies
- code samples for the languages/architectures
- CVE IDs of vulnerabilities for which that type of weakness exists
- references

1.6 Common Vulnerability Scoring System (CVSS)

CVSS is owned and managed by FIRST Special Interest Groups (SIG), a group of representatives from a broad range of industry sectors, from banking and finance to technology and academia.

CVSS is meant to capture the principal characteristics of a vulnerability and produce a numerical score reflecting its severity to help organizations properly assess and prioritize their vulnerability management processes.

CVSS is based on three main metric groups:

- *Base metric group* represents the innate characteristics of a vulnerability that are constant over time and across user environments
- *Temporal metric group* reflects the characteristics of a vulnerability that may change over time but not across user environments.

For example, the presence of a simple-to-use exploit kit would increase the CVSS score, while the creation of an official patch would decrease it

- *Environmental metric group* represents the characteristics of a vulnerability that are specific to a particular user's environment. They are essential to consider because base and temporal metrics alone can give a distorted picture of the real risks.

The *Base metric group* produces a score ranging from 0 to 10, which can then be modified by scoring the temporal and environmental metrics [12].

Currently, the existing CVSS standards are v2.0 and v3.x (the latest version is 3.1). In VISE we refer to CVSS v3.1 that is the latest standard supported by NVD too.

NVD only supports CVSS Base Scores but not the Temporal and Environmental ones.

1.7 Severity

To increase readability, scores are mapped to qualitative severity rankings.

Table 1.1 shows the NVD mapping between the qualitative severity rankings and the CVSS v3.0 Base Score Ranges [13].

As an example, a CVSS Base Score of 7.0 has an associated severity

rating of High.

Severity	Base Score Range
None	0.0
Low	0.1-3.9
Medium	4.0-6.9
High	7.0-8.9
Critical	9.0-10.0

Table 1.1: Severity - Base Score range mappings

1.8 Exploit

An exploit is a piece of code, a chunk of data that takes advantage of a vulnerability to cause unwanted behavior on a computer software and/or hardware.

A possible high-level classification for exploits takes into account how the exploit communicates with the vulnerable resource [14]. Doing so, we can distinguish:

- remote exploit, that takes advantage of the network to carry out the attack and does not require a previous access to the system to exploit the security vulnerability
- local exploit, that requires prior access to the vulnerable system and its execution usually increases the privileges of the person running the exploit on the target system

The process of attacking a vulnerability is called exploiting.

Chapter 2

Vulnerability Assessment: State of the Art

2.1 What is Vulnerability Assessment?

Vulnerability assessment is the process of defining, identifying, classifying, prioritizing, and reporting the vulnerabilities present in corporate assets (e.g., applications, computer systems, or network infrastructures) [15].

There are several ways to protect enterprise assets, some of them are reactive approaches (e.g., virus detection, firewalls, Intrusion Detection System (IDS)), others are proactive approaches (e.g., vulnerability assessment). Therefore, it is immediately clear the importance of vulnerability assessment, because being a proactive approach, it permits to identify and remedy weaknesses before they can be exploited.

According to Computer Emergency Response Team (CERT) of Carnegie Mellon University *"99% of network intrusions result from exploitation of known vulnerabilities or configuration errors where countermeasures were available"*.

For this reason, performing periodic vulnerability assessment is an essential process to protect Confidentiality, Integrity, and Availability (CIA) of enterprise assets.

However, what do we mean by *"periodic vulnerability assessment"*? It is what we typically call Vulnerability Management. To understand this, we refer to the difference between vulnerability assessment and vulnerability management marked by Hitachi Systems Security in [16].

According to Hitachi, vulnerability management is *"an ongoing, comprehensive process or program that aims at managing an organization's vulnerabilities in a holistic and continuous manner"*.

It hasn't got start and end dates because it is a continuous cyclical process that aims at managing (identifying, classifying, prioritizing, remedying, and mitigating) vulnerabilities in the long run.

Instead, vulnerability assessment is a one-time project with a defined start and end date.

So we can think to vulnerability assessment as an embedded process in what we name vulnerability management that consists in repeat vulnerability assessment process repeatedly over time. This is essential because network and host configurations change in time (e.g., new

software or network elements are installed and this, with a certain probability, introduce new weaknesses).

2.1.1 Vulnerability Assessment vs Penetration Testing

At the end of the whole assessment process, a detailed report is generated which can be used as an input to the penetration testing process. Penetration testing, also known as ethical hacking, consists in the legitimate use of tools and techniques typically used by hackers to search for vulnerabilities in computer systems, networks, or web applications. By performing the penetration test, it is possible to verify which vulnerabilities are actually present in the system and which are false positives.

Typically, when the goal is to find vulnerabilities, we rely on a process known as Vulnerability Assessment and Penetration Testing (VAPT) in which we try to combine the benefits of both techniques [17]:

The advantages of vulnerability assessment are:

- several open source tools are available
- allows to identify the majority of vulnerabilities
- scanning can be automated thanks to the available tools
- it is easy to rerun it several times over time

However, vulnerability assessment also has several disadvantages: for example the results are affected by numerous false positives. Furthermore, it can undermine the systems under study compromising their normal functioning and often it fails to notice the latest vulnerabilities.

Instead, the advantages of penetration testing are:

- allows to test assets with the same tools and techniques that hackers use
- permits to validate vulnerabilities
- permits to determine at what depth vulnerabilities can be exploited.

However, penetration testing has several disadvantages too, for example it needs experts, it is costly, it is typically conducted in a limited time period and in some cases it reveals the source code to third party.

2.2 Vulnerability Assessment Process

Vulnerability assessment can be thought as a process consisting of the following steps [18]:

1. *Initial Assessment* consists in identifying and cataloging assets and assigning quantifiable value in term of risk and importance

for each of them. In this phase it's also important to understand by whom the asset can be accessed (e.g., administrators, authorized users, any member of company).

2. *System Baseline Definition* consists in obtaining as much information as possible about the IT environment (e.g, networks, Internet Protocol (IP) addresses, open ports, processes, services, operating systems version, etc.). Also, the default configuration, drivers and software installed on each asset should be known. Finally, is critical to gather known vulnerabilities and other relevant details about assets.
3. *Perform the Vulnerability Scan*: in this phase vulnerability scanners are used to identify existing vulnerabilities. Prior to starting the vulnerability scan, the best time and date to perform the scan should be chosen and the right policy should be set on scanner taking into account enterprise needs.
4. *Vulnerability Assessment Report Creation* consists in reporting. Reports at different levels of detail are preferable, typically a high-level summary is useful for management while a detailed and complete report is more useful for system administrator. A complete and detailed report should include the following information:

- the name of vulnerability

- the date of discovery
- the score, based on CVE databases
- a detailed description of the vulnerability
- details regarding the affected systems
- details regarding the process to correct the vulnerability.
Some known remedies for vulnerabilities are: installation of a patch, a change in network security policy, reconfiguration of software and so on
- a Proof of Concept (PoC) of the vulnerability for the system (if possible)
- a blank field for the owner of the vulnerability, the time it took to correct, the next revision and countermeasures

2.3 Vulnerability Assessment Types

Russ Rogers in his book *Nessus Network Auditing*, 2nd Edition [19] distinguishes between two types of assessments.

According to Rogers, we speak of host assessment when our goal is the security analysis of a specific system. In contrast, we speak of network assessment when our goal is the security analysis of an entire network at once.

Network assessments are by far the most common and the most complex type of vulnerability assessment.

2.3.1 Host Assessment

A host assessment looks for system-level vulnerabilities such as:

- insecure file permissions
- missing software patches
- non-compliant security policies
- application level bugs
- outright backdoors
- trojan horse installations

Because of its deep analysis, this type of assessment is typically used to monitor security of critical systems. However, it is avoided for other systems because of large time investment required to perform it and its no scalability.

To perform host assessments is required to utilize a set of specialized tools for the operating system and software packages that need to be installed on the target assets. The assessment software can run stand-alone or linked to a central system. Nowadays, many of the stand-alone tools have been replaced by agent-based systems that use a centralized reporting and management system. This transition is explained by a demand for scalable and deployable systems with a minimum of administrative effort.

Several products, mostly commercial, exist to conduct host assessment.

2.3.2 Network Assessment

Network assessment allows looking for vulnerabilities on any system connected to the network.

The following are the steps typically required to conduct a network vulnerability assessment:

1. locate all live systems on the network
2. detect open ports in these systems
3. determine network services in use
4. analyze those services for potential vulnerabilities

Network assessment is the only feasible method to assess the security of large, complex networks of heterogeneous systems. Furthermore, this process does not require any configuration changes on the systems being assessed.

Network assessments can be both scalable and efficient.

However, despite the many advantages in identifying network vulnerabilities, network assessments have numerous limitations:

- can disrupt normal operations
- use large amounts of bandwidth

- create large log files on the systems being assessed
- some vulnerabilities may not be detected due to the presence of firewalls

2.4 Vulnerability Assessment Approaches

In [19] Rogers also defines two possible approaches for vulnerability assessment:

1. administrative
2. outsider

What can typically happen is that different assessment tools can provide completely different results despite they are evaluating the same network. This is largely due to the fact that the tools likely use different approaches for vulnerability assessment. This is the reason many of the better assessment tools have migrated to a hybrid model that combines the best features of both approaches. Doing so, they typically provide results which are better than those provided by tools that only use one of the two approaches.

2.4.1 Administrative Approach

The administrative approach performs the assessment from the perspective of an authenticated system administrator.

Typically, the assessment tool which follows the administrative approach might require to be executed by an authenticated administrative user.

These credentials can be used to detect:

- insecure configuration settings
- missing patches
- potentially vulnerable client-side software (such as web browsers and e-mail clients)

Considering the fact that many viruses like worms and trojans exploit vulnerabilities in e-mail clients and web browser software, the administrative approach is really useful from this point of view.

An assessment tool using this approach can access the Registry of each system and determine whether the latest patches have been installed, whether the proper security settings have been applied, and in some cases, can determine also whether the system has already been successfully attacked.

However, the administrative approach is not a panacea, in fact these products often have some severe limitations as well. For example, in most cases they only check for the existence of a patch and not if the service is really enabled or configured. Doing so, it may happen that they report vulnerabilities in services that are not enabled or configured.

2.4.2 Outsider Approach

The outsider approach allows to take the perspective of an unauthenticated malicious user attempting to break into the enterprise network. This is fundamental because it permits to gather the same information that an intruder would see in a real-life attack.

For example, if a system is behind a firewall, only the exposed services will be tested.

This approach is different from the administrative approach that often focuses on missing patches and insecure configuration settings.

The outsider approach is the only technique that has a chance of returning accurate, consistent results about each discovered system when conducting a large-scale assessment against a network consisting of many different operating systems and network devices.

However, the outsider approach provides incomplete results as it only detects what is visible from the point in the network where the assessment was launched, but it might not report a vulnerable service that is bound to a different interface on the same system. This is an issue as who reviews the assessment report might not consider the network perspective when creating a list of remediation tasks for that system.

2.5 Vulnerability Assessment Search Engines

A large variety of tools exist to conduct vulnerability assessment. In this section we only focus on the main search engines in support of vulnerability assessment because they are the type of tool that comes closest to the work we present in this thesis.

2.5.1 NVD Search

NVD by National Institute of Standards and Technology (NIST) is probably the reference repository when it comes to recovering data for the automation of vulnerability management, security measurement, and compliance. The NVD includes databases of security checklist references, security-related software flaws, misconfigurations, product names, and impact metrics.

Specifically, as regards the search engine, the official reference provides two search features, respectively called:

1. *Vulnerability Search*
2. *CPE Search*

As for *Vulnerability Search* feature [20], it gives the possibility to refine the search by specifying among the following different options:

- *Search Type* allows to choose between a Basic or Advanced search.

The Advanced search option allows to specify with respect to the Basic one: CVE ID, Category (CWE), CPE Name, Vendor, Product, CVSS Metrics (the exclusive options are Version 3.x, Version2, All), Published Date Range, Last Modified Date Range

- *Results Type*: the exclusive options are Overview and Statistics
- *Keyword Search* is an `<input>` element of type *text* in which is possible to insert a product name, vendor name, CVE name, or an Open Vulnerability and Assessment Language (OVAL) query
- *Search Type* allows to define the target time range of the search (the exclusive options are All Time, Last 3 Months, Last 3 Years)
- *Contains HyperLinks* allows to specify if the results must contain links to United States Computer Emergency Readiness Team (US-CERT) Technical Alerts, US-CERT Vulnerability Notes, and OVAL Queries

The *Vulnerability Search* feature returns a list of vulnerabilities matching the searching criteria. For each vulnerability, the reported information is divided into the following sections:

- CVE ID
- Current Description
- Severity (in both CVSS Version 3.x and 2.0)

- References to Advisories, Solutions, and Tools
- Weakness Enumeration in terms of CWE IDentifier (CWE ID), CWE Name and Source
- known Affected Software Configurations
- Change History
- QUICK INFO in terms of CVE Dictionary Entry, NVD Published Date, NVD Last Modified and Source

Instead, as for the *CPE Search* feature [21], it can perform a keyword search or a CPE Name search.

It gives the possibility to refine the search by specifying among the following different options:

- *CPE Naming Format*: the exclusive options are 2.3 and 2.2
- *CPE Name or Keyword* is an *<input>* element of type *text* in which is possible to insert a CPE Name or Keyword
- *Include deprecated CPEs* allows to eventually specify the will to include deprecated CPEs

The *CPE Search* feature returns a list of CPEs matching the searching criteria. For each CPE, a list of associated vulnerabilities is provided. The information available for these last is the same as provided with *Vulnerability Search* feature.

2.5.2 CVE List Search

CVE List Search [22] is the search feature provided by MITRE that returns as result an individual CVE Record and/or a list of all CVE Records.

CVE List Search provides the following search options:

- *Search by CVE ID*: giving as input a CVE ID number returns its description
- *Search by keyword*: giving as input a keyword returns the official CVE Record for a known vulnerability
- *Search by specific keywords*: it is possible to give as input specific keywords, such as an application name
- *Search by multiple keywords*: giving as input multiple keywords separated by a space returns as result several CVE Records matching all specified keywords

CVE List Search returns a list of vulnerabilities matching the searching criteria. For each vulnerability, the reported information is divided into the following sections:

- CVE ID with explicit reference to NVD
- Description
- References

- Assigning CNA
- Date Record Created
- Phase (Legacy)
- Votes (Legacy)
- Comments (Legacy)
- Proposed (Legacy)

CVE is not meant to fix information, impact, classification, or other technical details, it is meant to link vulnerability databases and not to replace them [23].

2.5.3 CVE Details Search

[24] is an effort by Serkan Özkan that provides a web interface to CVE vulnerability data. All data are taken from Extensible Markup Language (XML) feeds provided by NVD.

The search types offered by CVE Details are:

- *Vendor Search*: it is possible to use % for a like query. For example if we write %soft%, vendors with the *soft* string in their names are returned. The maximum number of displayed results is 50

- *Product Search*: it is possible to use % for a like query. For example if we write %soft%, products with a *soft* string in their names are returned. The maximum number of displayed results is 50
- *Version Search* permits to search for exact vendor, product and version strings. The maximum number of displayed results is 100
- *Vulnerability Search* provides several options to refine the search. Visit [25] for more details
- *Search by Microsoft References*: it is possible to search for security vulnerabilities related to a specific Microsoft "Security Advisory", "Knowledge Base Article" or "Security Bulletin". To do so, in the form the Microsoft Reference ID must be specified

2.5.4 Vulmon Search

Vulmon Search [26] is a vulnerability search engine. It gives comprehensive vulnerability information through a very simple user interface.

It is based on a full-text database and allows to search by:

- CVE ID
- vulnerability score
- vulnerability type

- vendor
- product
- exploit
- operating system
- anything else related to vulnerabilities

Secondly, it permits to sort the results by *Relevance*, *Risk Score*, *Publish Date* or *Recent Activity*.

As a search response, Vulmon returns a list of results. The single result, which refers to a specific vulnerability, is structured in the following sections:

- set of information on Base Score, Impact Score, and Exploitability Score (specified for both CVSS v3 and CVSS v2)
- Vulnerability Summary
- Most Upvoted Vulmon Research Post
- Vulnerability Trend to make cybersecurity researchers know if other of them talk about this vulnerability
- Vendor Advisories
- Exploits
- Mailing Lists

- Github Repositories
- Recent Articles
- References

Not necessarily all the sections above are present.

2.5.5 EDB Search

The Exploit Database (EDB) is a CVE compliant archive of public exploits and proof-of-concepts provided as a public service by Offensive Security to support penetration testers and vulnerability researchers.

The EDB Advanced Search [27] is the official search feature provided by Offensive Security to retrieve information from EDB.

EDB Advanced Search gives the possibility to refine the search by specifying among the following different options:

- Title of the exploit
- CVE ID of the exploited vulnerability
- exploit Type: the exclusive options are *dos*, *local*, *remote*, *shellcode*, *papers*, *webapps*
- target Platform (e.g. ARM, Linux, iOS, Windows)
- Port
- exploit Content

- Author of the exploit
- Tag (e.g. buffer overflow, Cross-Site Scripting (XSS), Denial of Service (DoS))

In addition, it is possible to further refine the search with *Verified*, *Has App* and *No Metasploit* checkboxes.

The *Verified* checkbox is used to show in results only exploits whose working has been verified.

The *Has App* checkbox is used to show in results only exploits for which a vulnerable application is available.

The *No Metasploit* checkbox is used to filter out exploits provided by Metasploit.

2.5.6 VulDB Search

Vulnerability Database (VulDB) is a community-driven vulnerability database containing all public security vulnerabilities about electronic products.

As regards VulDB Search feature [28], it provides two search options, respectively, called:

1. *Simple Search*
2. *Advanced Search*

The *Simple Search* consists of an `<input>` element of *text* type.

It also permits to use double quotes to define an exact CPE string or phrases to match products.

The *Advanced Search* permits to refine the search by specifying more details in the following search sections:

- Product: Vendor, Product, Version, Type
- Details: Component, File, Function, Argument
- Advisory: Advisory ID, Researcher, Organization
- Exploit: Developer, Programming Language
- Attack Tools: (e.g. Nessus ID, Open Vulnerability Assessment Scanner (OpenVAS) ID)
- Defense Tools: (e.g. Snort ID, Suricata ID)
- References: (e.g. CVE, OVAL, EDB)

As a search response, VulDB Search feature returns a list of vulnerabilities. For each vulnerability VulDB Search feature provides a great variety of information.

The main characteristic about VulDB is that it provides a wide variety of features not available by other vendors and services, for example:

- also exotic sources are included (e.g. social media, forums, Darknet)
- multiple risk rating indicators are provided:

- high-level risk rating (low, medium, high)
 - CVSSv2 and CVSSv3 Base and Temp scores by VulDB, Vendor, Researcher and NVD
 - unique CVSSv3 Meta score
 - unique exploit price calculation
 - unique cyber threat intelligence scoring
 - additional risk ratings by other sources
- open standards supported: CVSSv3, CVE, CWE, CPE, OVAL, Information Assurance Vulnerability Management (IAVM)
 - not a single Application Programming Interface (API) but several are provided: JavaScript Object Notation (JSON), XML, Comma-Separated Values (CSV), Real Simple Syndication (RSS)

However, both the number of consecutive searches that can be done and the amount of information provided in the results are limited by default. In order to use the full potential of VulDB Search feature a sign-up is required.

2.5.7 Vulnerability & Exploit Database Search

Vulnerability & Exploit Database is a repository of vulnerabilities and related exploits provided by Rapid7.

The vulnerabilities are also utilized by the Rapid7 commercial vulnerability management tool InsightVM, instead the exploits are all included in the Metasploit framework.

Vulnerability & Exploit Database Search feature [29] permits to search by specifying exclusively one of the following options:

- Type
- Metasploit Module
- Vulnerability

As a search response, Vulnerability & Exploit Database Search feature returns, depending on the chosen search option, respectively:

- a list of vulnerabilities or Metasploit modules matching the searching criteria
- a list of Metasploit modules matching the searching criteria
- a list of vulnerabilities matching the searching criteria

Each result of *Metasploit Module* type provides the following information:

- a Description of module
- Author(s) of module
- Platform

- Architectures
- Development in terms of source code and history
- Module Options

Each result of *Vulnerability* type provides the following information:

- a set of information in terms of Severity, CVSS, informational dates (Published, Created, Added and Modified)
- a Description of vulnerability
- a list of Solution(s)
- a list of References

2.5.8 Tenable CVE Search

Tenable CVE Search [30] is a search feature provided by Tenable.

It consists of an *<input>* element of *text* type and permits to order results by *Relevance*, *Newest* and *Updated*.

Tenable CVE Search also permits to refine search by filtering by:

- CVE ID
- Date Published
- Date Modified

- CVSS v2.0 Severity
- CVSS v3.0 Severity

As result, it provides a list of CVEs. For each of them the following information are provided:

- a Description on vulnerability
- more Details in terms of Source, Published and Updated dates and Type of CWE
- Risk Information in terms of Base Score, Vector, Impact Score, Exploitability Score and Severity for both CVSS v2.0 and CVSS v3.0
- References
- Vulnerable Software (in terms of CPEs) affected by the vulnerability

Chapter 3

VISE: Architecture

In this chapter, we examine the architecture of the search engine we present in this thesis. We will deal with the high level architecture of VISE and then we will deepen the description of the individual components.

3.1 High Level Architecture

In Figure 3.1 is shown the logical architecture of the proposed tool. The idea behind VISE is to create a search engine that is able to retrieve information on vulnerabilities and related exploits affecting the search input packages by aggregating and normalizing this information.

To make this possible, we need some components to previously store this information retrieved by OSINT sources.

In particular, we use Elasticsearch to store the data feeds retrieved

by NVD.

Instead, EDB is used as a database to retrieve information on exploits associated with vulnerabilities.

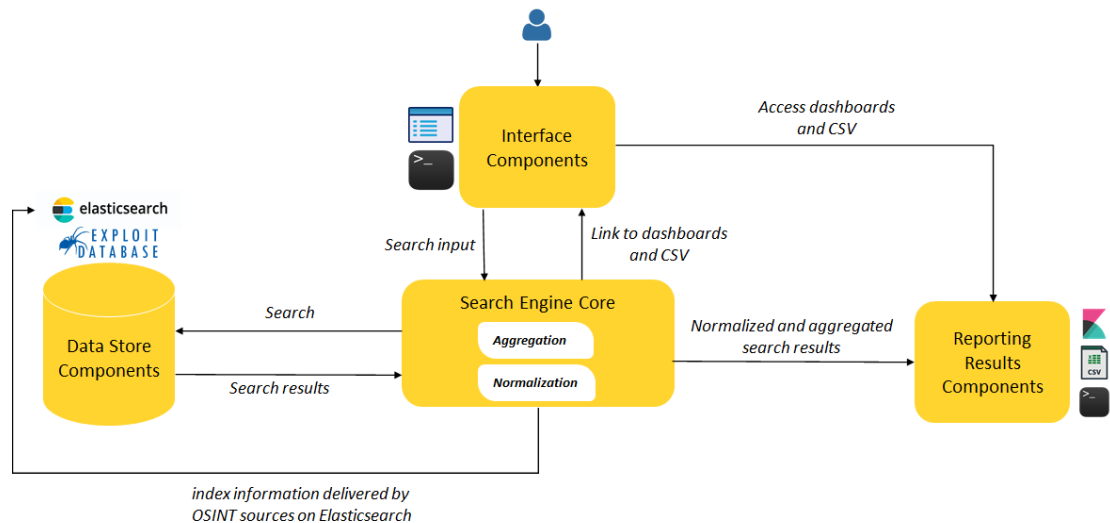


Figure 3.1: VISE: High level architecture

At the heart of the architecture we have the search engine core that is the engine of the architecture.

Its tasks are:

- the creation of indices on Elasticsearch for storing NVD feeds
- the search for vulnerabilities and exploits associated with search input packages
- the creation of dashboards on Kibana, the creation of a CSV file and the creation of a Command Line Interface (CLI) output for the results report

The user can interface with the tool thanks to two interfaces:

1. a web Graphical User Interface (GUI)
2. a CLI. In this case the user can invoke a new search from the CLI itself

In the case of search via web GUI, at the end of the search it is possible to find the links to the dashboards created on Kibana, and the link to the CSV file for the report of the results.

In the case of search via CLI, VISE also provides an output in the CLI itself.

3.2 Architecture Components

3.2.1 Data Store Components

Elasticsearch

As previously mentioned, our main information source is NVD data feeds [31]. In particular, in our work we refer to:

- CPE Match Feed
- CVE JSON Feeds

In subsection 4.2.1 we will see more in depth how this information can be combined to obtain search results.

This feeds are indexed on Elasticsearch, a distributed, free search engine for all types of data built on Apache Lucene.

Elasticsearch is the central component of the Elastic Stack, that is the Elasticsearch, Logstash, Kibana (ELK) Stack, but with more flexibility. Elastic Stack nowadays includes tools for data ingestion, enrichment, storage, analysis, visualization, and a set of agents known as Beats to send data to Elasticsearch [32].

The reasons that led us to choose Elasticsearch to store feeds are:

- it is fast at full-text search
- the latency from the time a document is indexed until it becomes searchable is very short, so in a context such as that of vulnerability assessment in which new vulnerabilities and products are discovered or released on a daily basis, it is important that the data are available as soon as possible to be searched
- it is distributed by nature thanks to the shards, the containers storing the redundant copies of the data in case of hardware failure
- it is also scalable thanks to the previous feature

Exploit Database

To retrieve information on exploits, we needed an existing repository of exploits, so we decided to refer EDB.

For more information on EDB, please refer to the 2.5.5 subsection. In particular, we have decided to give the possibility to use our tool

even when the connection to the Internet is not available.

This goal was achieved thanks to two project choices:

1. the use of NVD feeds (indexed on Elasticsearch) instead of the NVD API
2. using a local exploit database instead of a remote one

These two choices also increase the search performance.

The possibility of having a local exploit database has been pursued thanks to SearchSploit [33].

It is a command line search tool for EDB that also allows to take a local copy of Exploit Database.

3.2.2 Interface Components

As previously said, the user can start a new search either by using a Web GUI or by invoking a search from the CLI.

Web GUI

With the aim of reducing the heterogeneity of the used tools for the realization of VISE, we decided to opt for Flask for the realization of the web GUI.

Flask is lightweight Web Server Gateway Interface (WSGI) web application framework [34].

It is one of the most popular Python web application frameworks, so we opt for it since the heart of the tool we propose is entirely in Python.

Flask is a wrapper around:

- Werkzeug: a utility library for the Python programming language, and in particular, a toolkit for WSGI applications. Werkzeug allows to manage software objects for request, response, and utility functions.
- Jinja2: one of the most used template engines for Python programming language that handles templates in a sandbox.

Regarding the search goal, the web GUI makes available two options to define the search input:

1. manually filling a form
2. uploading a searching card

In section 5.2 we details these two options.

Another goal of web GUI is to make available the links to Kibana dashboards and to the CSV file that are the outputs provided by the search engine core.

3.2.3 Search Engine Core

As for the search engine core, it was developed almost entirely in Python and from scratch.

It consists of four main modules:

1. *json-parse.py*: it is part of a bigger job which is the repository *cve-analysis* [35] that we use as is in our work. The repository as a whole provides scripts to download the CVE JSON Feeds by NVD and then to index this information on Elasticsearch. The indexing is realized thanks to the *jsonparse.py* module.
2. *cpe-match_indexing.py*: it was created from scratch and does indexing in Elasticsearch of CPE Match Feed by NVD.
3. *queries.py*: it contains the search logic of the tool which consists in crossing the information present in CVE JSON Feeds and CPE Match Feed, previously uploaded to Elasticsearch.
4. *create_dashboards.py*: it deals with the creation of dashboards containing normalized and aggregated search results.

As for the *json-parse.py*, *cpe-match_indexing.py* and *queries.py* modules we used the Python *Elasticsearch* library [36] that is the official low-level client for Elasticsearch in Python.

Instead, as for the *create_dashboards.py* module we used the Python *requests* library [37] which allows to make Hypertext Transfer Protocol (HTTP) requests (in our case dashboard creation requests) in a simplified way in Python.

3.2.4 Reporting Results Components

Kibana

To provide search results reports, we introduced in our architecture Kibana [38]. This is because Kibana is the official interface of Elasticsearch.

It is a free and open frontend application that has got searching, viewing, and visualizing capabilities for data indexed in Elasticsearch.

Kibana permits the visual analysis of data from one or more Elasticsearch indices. Data are visualized through pie charts, bar charts, tables, histograms, and maps.

A Kibana dashboard is a pane collecting these and other elements like graphs, metrics, and searches. It permits visualize and analyze data from multiple perspectives and enable users to drill down into the details.

For our purposes, for each submitted search, three dashboards are created on Kibana, respectively, named:

1. Vulnerability summary
2. Vulnerability technical description
3. Exploit view

In section 5.2 we delve into the content of these dashboards.

For simplicity, in our work we use Elasticsearch and Kibana in their Docker container versions.

To make so, we started from the GitHub repository *docker-elk* [39] but we made some changes to customize it for our work.

In subsection 4.1.1 we detail the docker-compose file that contains the configuration of these two services.

Chapter 4

VISE: Implementation View

In this chapter we go deep on some relevant aspects of our work.

In particular we detail:

1. the *docker-compose.yml* file for the configuration of services, networks and volumes when working with Docker
2. the *Search* function: to make the search function more understandable we report a detailed activity diagram and we make a focus on code.

4.1 Microservices Configuration

As mentioned in subsection 3.2.4, in our product we needed the deployment of Kibana and Elasticsearch. To do this we relied on Docker.

Docker is an open-source project that automates the deployment of applications within software containers, providing additional abstraction through Linux OS-level virtualization.

In practice we used the Kibana and Elasticsearch microservices in their version mapped on docker containers.

To define the configuration of these services, but also of the network on which these services are attached and of the volume used for persistence, we have defined a docker-compose file that we detail in the following subsection.

4.1.1 The docker-compose File

The Compose file is a YAML Ain't Markup Language (YAML) file defining services, networks, and volumes.

The Compose version we adopt is *3.2*.

First of all we define the *elasticsearch* and *kibana* services.

For both of them we want the automatic restart when they exit or when Docker restarts.

Then we define the configuration options that are applied at *build* time. In particular for both services we define the *context* and the *ELK_VERSION* whose value is *7.8.0*. Therefore we use Elasticsearch and Kibana in their *7.8.0* version.

Host ports *9200* and *9300* are respectively mapped on *elasticsearch* container's ports *9200* and *9300*, instead host port *5601* is mapped on

kibana container's port *5601*.

Both containers are attached to the *elk* network we define at the end of the docker-compose file.

Furthermore we also point out that Kibana *depend_on* Elasticsearch.

For *kibana* container we define the *bind* mount between *./kibana/-config/kibana.yml* file on the host machine and the */usr/share/kibana/-config/kibana.yml* file into the container.

For *elasticsearch* container we define the *bind* mount in *read_only* mode between *./elasticsearch/config/elasticsearch.yml* file on the host machine and the */usr/share/elasticsearch/config/elasticsearch.yml* file into the container.

Furthermore for *elasticsearch* container we also define the *volume* mounted on the */usr/share/elasticsearch/data* container path.

Finally, we set the maximum heap size to 3GB, we set *changeme* as *ELASTIC_PASSWORD* and with *discovery.type: single-node* we say *elasticsearch* container to elect itself a master and not join a cluster with any other node.

```
1 version: '3.2'
2
3 services:
4   elasticsearch:
5     restart: always
6     build:
7       context: elasticsearch/
```

```
8      args:
9          ELK_VERSION: $ELK_VERSION
10     volumes:
11         - type: bind
12           source: ./elasticsearch/config/elasticsearch.yml
13           target: /usr/share/elasticsearch/config/
14             elasticsearch.yml
15         read_only: true
16         - type: volume
17           source: elasticsearch
18           target: /usr/share/elasticsearch/data
19     ports:
20         - "9200:9200"
21         - "9300:9300"
22     environment:
23         ES_JAVA_OPTS: "-Xmx3g_-Xms3g"
24         ELASTIC_PASSWORD: changeme
25         discovery.type: single-node
26     networks:
27         - elk
28 kibana:
29     restart: always
30     build:
31         context: kibana/
32         args:
33             ELK_VERSION: $ELK_VERSION
34     volumes:
```



```
35     - type: bind
36     source: ./kibana/config/kibana.yml
37     target: /usr/share/kibana/config/kibana.yml
38     read_only: true
39   ports:
40     - "5601:5601"
41   networks:
42     - elk
43   depends_on:
44     - elasticsearch
45
46 networks:
47   elk:
48     driver: bridge
49
50 volumes:
51   elasticsearch:
```

Code Listing 4.1: The docker-compose.yml file

4.2 The Search Functionality

In this section we deepen the search functionality.

As a search input, we aspect user provide a list of packages for which he is interested to retrieve information about related vulnerabilities, their remediations, and exploits.

This information is the search output we provide in several formats

(graphical and textual).

4.2.1 Activity Diagram

In Figure 4.1 we provide the activity diagram describing the execution flow of a generic search.

First of all a user should submit a list of packages of which he is interested to retrieve information.

If not all packages have just been processed, a new package is picked up from the list and a new corresponding *cpe23Uri* is built.

A *cpe23Uri* is a formatted string binding used to identify one or more class of products.

As mentioned in subsection 3.2.1 a cross between the information in the CPE Match Feed and the CVE JSON Feeds is needed for our purposes.

This is because in CVE JSON Feeds the mapping between each CVE and affected products is typically made in terms of general *cpe23Uris*.

To avoid false positives and match in the right way the input package, we use CPE Match Feed.

CPE Match Feed let us to map the search input *cpe23Uri* with a list of CPEs.

If no CPE is returned we start processing a new package.

Instead, if one or more CPEs are returned we have that each CPE contains information on:

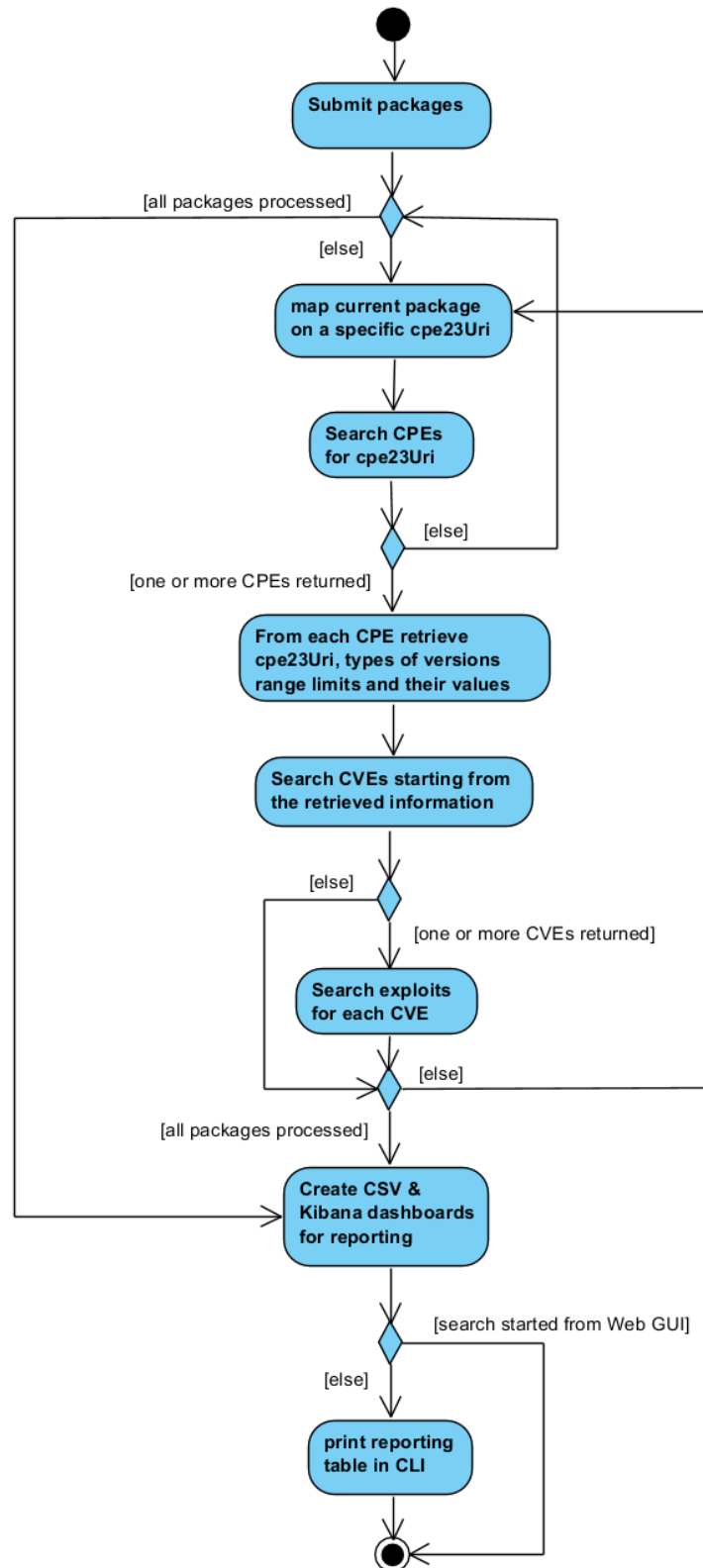


Figure 4.1: Search function: Activity diagram

1. a general *cpe23Uri*
2. the types of versions range limits (*versionStartIncluding*, *versionStartExcluding*, *versionEndIncluding*, *versionEndExcluding*)
3. the values of versions range limits (e.g. *2.0*, *1.27.0*)

This three information can be all used to retrieve CVEs from CVE JSON Feeds.

At this point if no CVE is returned but other packages to process exist we start processing a new package, else if one or more CVEs are returned we start searching related exploits.

When all packages have been processed a CSV and three Kibana dashboards are created to report results.

At the end, if the search has not been started from the Web GUI but from the CLI, a reporting table is printed in the CLI itself.

4.2.2 Code Focus

In this subsection we make a focus on the CPEs and CVEs search functions we invoke to realize the overall search functionality.

The *searchCPEs()* function, given as input a package in terms of *Product Name*, *Version Number*, *Vendor Name*, *Target Software* and *Product Type*, builds a *searched_cpe23Uri* that is a CPE 2.3 in its formatted string binding form and it passes this last to the *search_CPEs()* function.

```

1 def searchCPEs(package):
2     if package['Product Type'] in ["Application", "application", "a", "A"]:
3         package['Product Type'] = "a"
4     elif package['Product Type'] in ["OS", "os", "O", "o"]:
5         package['Product Type'] = "o"
6     elif package['Product Type'] in ["Hardware", "hardware", "h", "H"]:
7         package['Product Type'] = "h"
8     else:
9         package['Product Type'] = "a"
10
11     if (package['Vendor Name'] == ""):
12         package['Vendor Name'] = ".*"
13
14     if (package['Target Software'] == ""):
15         package['Target Software'] = ".*"
16
17     searched_cpe23Uri = "cpe:2.3:{0}:{1}:{2}:{3}::*:*:*:*:*:{4}::*:*".format(package['Product Type'], package['Vendor Name'], package['Product Name'], package['Version Number'], package['Target Software'])
18     cpes = search_CPEs(searched_cpe23Uri)
19
20     searched_cpe23Uri = searched_cpe23Uri.replace(".*", "")
21     return searched_cpe23Uri, cpes

```

Code Listing 4.2: The searchCPEs() definition

The *search_CPEs()* function returns a list of CPEs matching the input *searched_cpe23Uri*. To make so, it starts a new search in *cpe-index* indexed on Elasticsearch using the official low-level client for Elasticsearch.

```

1 def search_CPEs(searched_cpe23Uri):
2     es = Elasticsearch(hosts=[es_url])
3
4     res = es.search(index="cpe-index", body={
5         "query": {
6             "bool": {
7                 "should": [
8                     {
9                         "regexp": {
10                             "cpe23Uri.keyword": {
11                                 "value": searched_cpe23Uri,
12                                 "boost": 1.0
13                             }
14                         }
15                     },
16                     {
17                         "regexp": {
18                             "cpe_name.cpe23Uri.keyword": {
19                                 "value": searched_cpe23Uri,

```

```
20         "boost": 1.0
21     }
22 }
23 }
24 ]
25 }
26 }
27 }, size=1000)
28 cpes = res['hits']['hits']
29 return cpes
```

Code Listing 4.3: The search_CPEs() definition

At this point, for each retrieved CPE we start a new CVEs search by specifying the related *cpe23Uri*, the types of versions range limits and their values.

```
1 def search_CVEs_from_single_limit(cpe23Uri_to_submit, children = ""):
2     es = Elasticsearch(hosts=[es_url])
3
4     res = es.search(index="cve-index", body={
5         "query": {
6             "bool": {
7                 "must": [
8                     {
9                         "term": {
10                             "vuln.nodes{0}.cpe_match.cpe23Uri.keyword".format(children): {
11                                 "value": cpe23Uri_to_submit['cpe23Uri'],
12                                 "boost": 1.0
13                             }
14                         }
15                     },
16                     {
17                         "term": {
18                             "vuln.nodes{0}.cpe_match.{1}.keyword".format(children,
19 cpe23Uri_to_submit['version_types'][0]): {
20                                 "value": cpe23Uri_to_submit['version_types_values'][0],
21                                 "boost": 1.0
22                             }
23                         }
24                     }
25                 ]
26             }
27         }, size=1000)
28     cves = res['hits']['hits']
29     return cves
```

Code Listing 4.4: The search_CVEs_from_single_limit() definition

In particular if only one boundary is present in the CPE, we invoke the *search_CVEs_from_single_limit()* function. This last, given as input the *cpe23Uri_to_submit* dictionary, starts a new search in *cve-index* indexed on Elasticsearch using the official low-level client for Elasticsearch.

```

1 def search_CVEs_from_interval(cpe23Uri_to_submit, children = ""):
2     es = Elasticsearch(hosts=[es_url])
3
4     res = es.search(index="cve-index", body={
5         "query": {
6             "bool": {
7                 "must": [
8                     {
9                         "term": {
10                            "vuln.nodes{0}.cpe_match.cpe23Uri.keyword".format(children): {
11                                "value": cpe23Uri_to_submit['cpe23Uri'],
12                                "boost": 1.0
13                            }
14                        },
15                    ],
16                    {
17                        "term": {
18                            "vuln.nodes{0}.cpe_match.{1}.keyword".format(children,
19                                cpe23Uri_to_submit['version_types'][0]): {
20                                "value": cpe23Uri_to_submit['version_types_values'][0],
21                                "boost": 1.0
22                            }
23                        },
24                    ],
25                    {
26                        "term": {
27                            "vuln.nodes{0}.cpe_match.{1}.keyword".format(children,
28                                cpe23Uri_to_submit['version_types'][1]): {
29                                "value": cpe23Uri_to_submit['version_types_values'][1],
30                                "boost": 1.0
31                            }
32                        }
33                    ]
34                }
35            }, size=1000)
36     cves = res['hits']['hits']
37     return cves

```

Code Listing 4.5: The *search_CVEs_from_interval()* definition

The dictionary contains the *cpe23Uri*, the type of version range

limit, and its value.

The *search_CVEs_from_single_limit()* function returns a list of CVEs matching this three information.

This function is invoked two times to take in consideration also children field in *cve-index*.

```
1 def search_CVEs(cpe23Uri):
2     es = Elasticsearch(hosts=[es_url])
3
4     res = es.search(index="cve-index", body={
5         "query": {
6             "bool": {
7                 "should": [
8                     {
9                         "term": {
10                            "vuln.nodes.cpe_match.cpe23Uri.keyword": {
11                                "value": cpe23Uri,
12                                "boost": 1.0
13                            }
14                        }
15                    },
16                    {
17                        "term": {
18                            "vuln.nodes.children.cpe_match.cpe23Uri.keyword": {
19                                "value": cpe23Uri,
20                                "boost": 1.0
21                            }
22                        }
23                    }
24                ]
25            }
26        }, size=1000)
27     cves = res['hits']['hits']
28     return cves
```

Code Listing 4.6: The *search_CVEs()* definition

Instead, if two boudaries are present in the CPE, we invoke the *search_CVEs_from_interval()* function. This function differs from the previous one only for the number of types of versions range limits and related values. In this case, two types and values of versions range limits are specified in the *cpe23Uri_to_submit* dictionary.

Also in this case the function is invoked two times to take in consideration the children field in *cve-index*.

If no boundary are specified in the current CPE but a *cpe23Uri* that specifies a specific version is present, we invoke the *search_CVEs()* function that given the aforementioned *cpe23Uri* as input, returns the related CVEs.

In this case a single invocation of function it is enough to take in consideration also the children field in *cve-index*.

Chapter 5

WISE: User's Manual

In this chapter we see in more detail how to install, configure and utilize VISE.

5.1 Installation & Configuration

In this section we deal with defining the steps for installing the proposed tool. In fact for the tool to be usable, some requirements must be met.

The first requirement is to use as operating system a Linux distribution (preferably Ubuntu).

After that, since the tool is mostly developed in Python, we need to proceed with the installation of the Python package manager *python3-pip* which will allow us to install and manage additional libraries and dependencies, not present in the standard library, that we need for the

correct functioning of VISE.

At this point, before proceeding with the installation of Flask, we create a virtual environment. This will allow us to have more flexibility in managing Python libraries, so that any different Flask projects can use different versions of the same library without creating conflicts.

Therefore, to proceed with the installation of the *python3-pip* package manager and of the virtual environment we execute the following commands:

```
$ cd VISE/scripts/  
$ sudo ./installOthers.sh
```

Code Listing 5.1: Install python3-pip & python3-venv

The third step is installing Flask. This goal can be achieved by running the following commands:

```
$ cd ../VISE_app/  
$ source venv/bin/activate  
$ sudo ../scripts/installFlask.sh
```

Code Listing 5.2: Install Flask

Therefore once the virtual environment is activated, thanks to the *installFlask.sh* script, both Flask and Python needed libraries are installed.

The fourth step is to install the EDB.

The *searchsploit -u* command permits to keep updated the database, instead the *cve_searchsploit -u* command permits to update the

cve-edbid mapping.

```
$ sudo ../scripts/installExploitDB.sh  
$ searchsploit -u  
$ sudo cve_searchsploit -u
```

Code Listing 5.3: Install ExploitDB

In particular, the *installExploitDB.sh* script downloads the EDB locally, while the *searchsploit -u* and *sudo cve_searchsploit -u* commands respectively, allow to update the exploit database and the cve-edbid mapping.

cve_searchsploit [40] is a Python library that allows to search for exploits in the local EDB starting from the CVE for which these exploits have been created.

To complete the installation, we need to install Docker, which we use, as previously mentioned, for the deployment of the Elasticsearch and Kibana microservices. To do this, we run the following command:

```
$ sudo ../scripts/installDockerUbuntu.sh
```

Code Listing 5.4: Install Docker

In reality, in addition to proceeding with the installation of Docker, the aforementioned command also runs Elasticsearch and Kibana containers thanks to the *docker-compose up* command.

At this point, we only need one last configuration before starting the tool. This configuration consists in uploading to Elasticsearch the *cpe-index* and *cve-index* indices we mentioned in subsection 3.2.1. To

do this, we execute the following command:

```
$ sudo ../scripts/uploadIndices.sh
```

Code Listing 5.5: Index cpe-index & cve-index in Elasticsearch

In particular, the *uploadIndices.sh* script invokes four further scripts we detail below:

- *get-cve-json.sh*: which takes care of downloading the NVD CVE JSON Feeds we are interested in
- *update-es.sh*: which invokes the *json-parse.py* Python module (which we discussed in subsection 3.2.3) on each newly downloaded NVD CVE JSON Feed. The Python *json-parse.py* module actually takes care of creating the *cve-index*
- *download_cpe-match.sh*: which takes care of downloading the CPE Match Feed
- *cpe-match_indexing.py*: which is the Python module that takes care of the creation of the *cpe-index* related to the CPE Match Feed just downloaded

5.2 Example of use

Once the VISE tool is installed, the user can start the web application by executing the following commands (superuser's security privileges are required):

```
$ cd VISE/VISE_app/  
$ sudo ../scripts/startVISE.sh
```

Code Listing 5.6: Start VISE

In Figure 5.1 is shown the home page of the web application that is available at *http://localhost:5000/* after *startVISE.sh* script is executed.

The home page gives a brief about VISE that explains how it can be used.

In particular, from the web GUI we have two options to start a search:

1. filling a Form
2. uploading a Searching card

Suppose in the first place we want to start a new search by filling a form.

The form elements available to refine the search are:

- Package Name
- Version Number
- Vendor Name
- Target Software
- Product Type

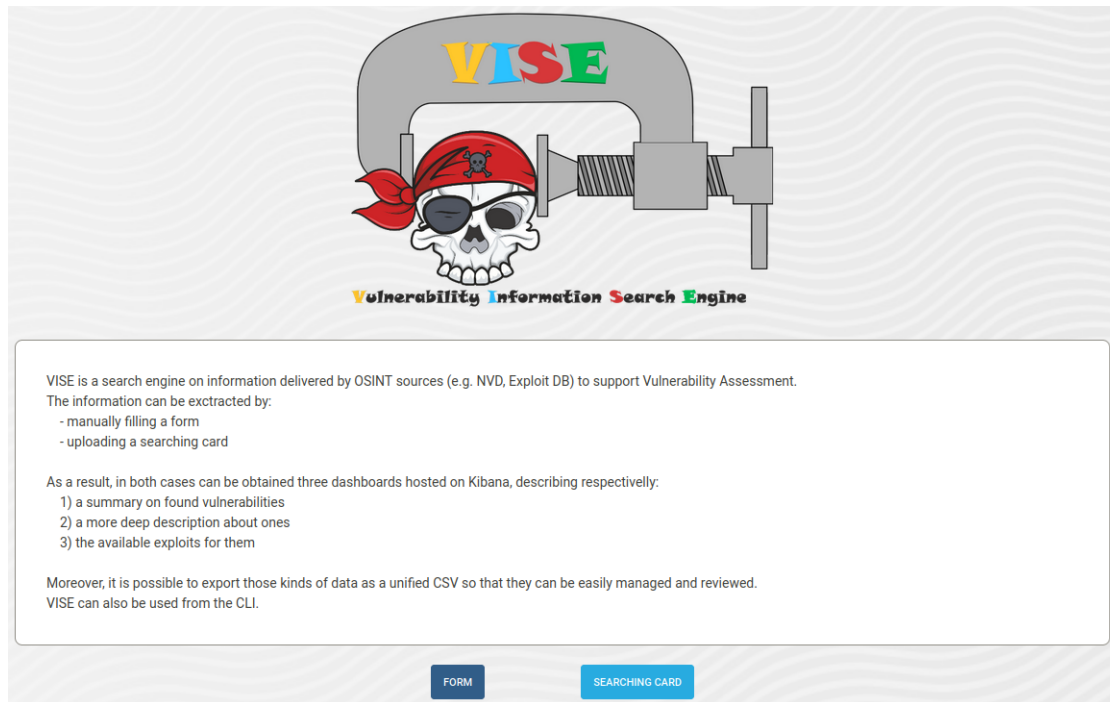


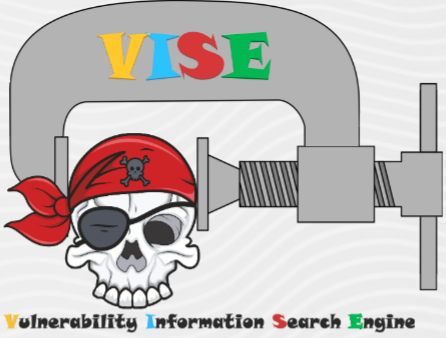
Figure 5.1: VISE: Home page

In Figure 5.2 we have a snapshot of the aforementioned form page in which we can notice that the table headers in green indicate the mandatory fields to be provided as input to the search.

Furthermore, it is possible to iteratively add new rows to the table, where each row constitutes a package of which we are interested in receiving information about the vulnerabilities that affect it, the remedies to solve these vulnerabilities and the exploits available to take advantage of them.

As shown in the Figure 5.3 a user can alternatively decide to upload a spreadsheet file in which he has defined the packages for which he is interested in retrieving information.

To help the user to accomplish this task VISE also makes available



Vulnerability Information Search Engine

Package Name *

Version Number *

Vendor Name

Target Software

Product Type


* = Required fields.

ADD ROW

Package	Version	Vendor	Target Software	Product Type	Remove
docker	1.12.0	docker		Application	<input type="button" value="x"/>
mozilla	84.0	firefox	android	Application	<input type="button" value="x"/>
f5c_router_firmware	3.1.1.65150	360		OS	<input type="button" value="x"/>
bbpress	2.0.0	bbpress	wordpress	Application	<input type="button" value="x"/>
vcenter_server	6.7	vmware		Application	<input type="button" value="x"/>

SUBMIT

Figure 5.2: VISE: Form page



Vulnerability Information Search Engine

Import Searching Card

Click the button below to upload the searching card.

Searching Card sample

Click the button below to download a sample of a searching card.

DOWNLOAD

If you want to create your own searching card be careful, you must specify at least the green fields in the image. A correct example of a searching card is the following.

Product Name	Version Number	Vendor Name	Target Software	Product Type
curl	7.58.0	haxx		Application
network_monitor	5.4	10-strike		Application
easycreate	3.2.1	iscrypts		Application
f5c_router_firmware	3.1.1.65150	360		OS
intercan_web_security_virtual_appliance	6.3	trendmicro		Application
3cn220	2.0.22	3com		Hardware
firefox	84.0	mozilla	android	Application

Figure 5.3: VISE: Searching card page

the possibility to download a Searching Card sample, whose structure is shown in the Figure 5.4.

Product Name	Version Number	Vendor Name	Target Software	Product Type
curl	7.58.0	haxx		Application
network_monitor	5.4	10-strike		Application
easycreeate	3.2.1	iscripts		Application
f5c_router_firmware	3.1.1.65150	360		OS
interscan_web_security_virtual_appliance	6.5	trendmicro		Application
3cnj220	2.0.22	3com		Hardware
firefox	84.0	mozilla	android	Application

Figure 5.4: VISE: Searching card sample

An HyperText Markup Language (HTML) page is returned to the user once the search is completed. This page contains the links to the three dashboards we mentioned in subsection 3.2.4, and the link to the CSV file.

In Figure 5.5 the aforementioned page is represented.

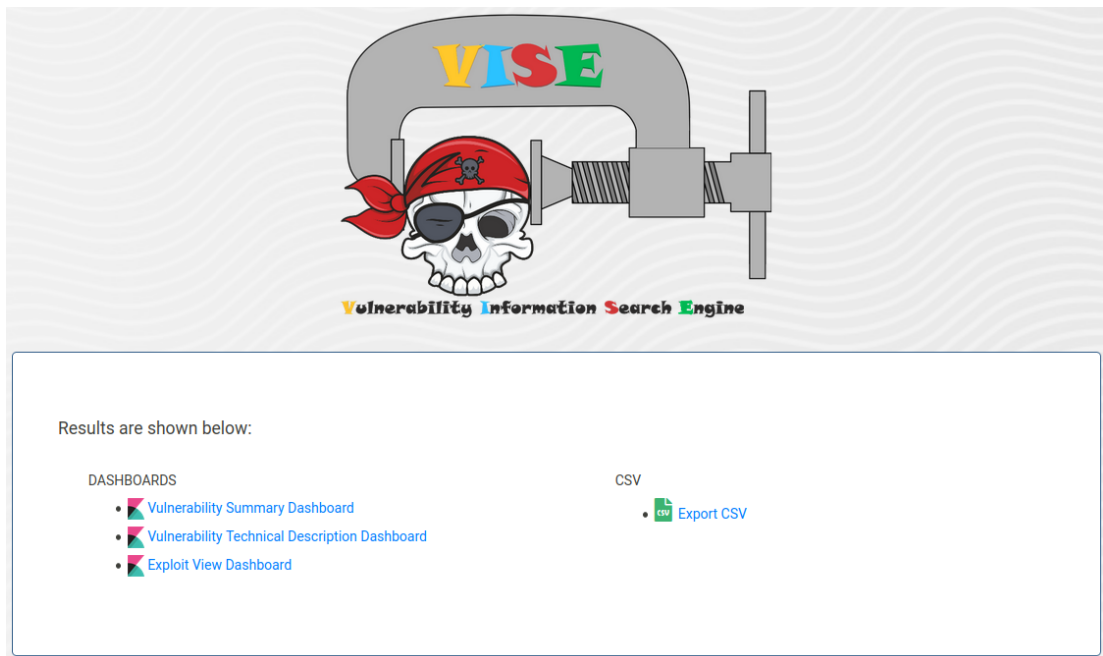


Figure 5.5: VISE: Results page

The idea behind dashboards is to offer several viewpoints about

the search results report.

The *Vulnerability summary* dashboard, Figure 5.6, gives an overview about the vulnerabilities associated with the search input packages. In particular, the first visualization shows for each CPE the total number of related CVEs and their severity in terms of CVSSv3 Base Score.

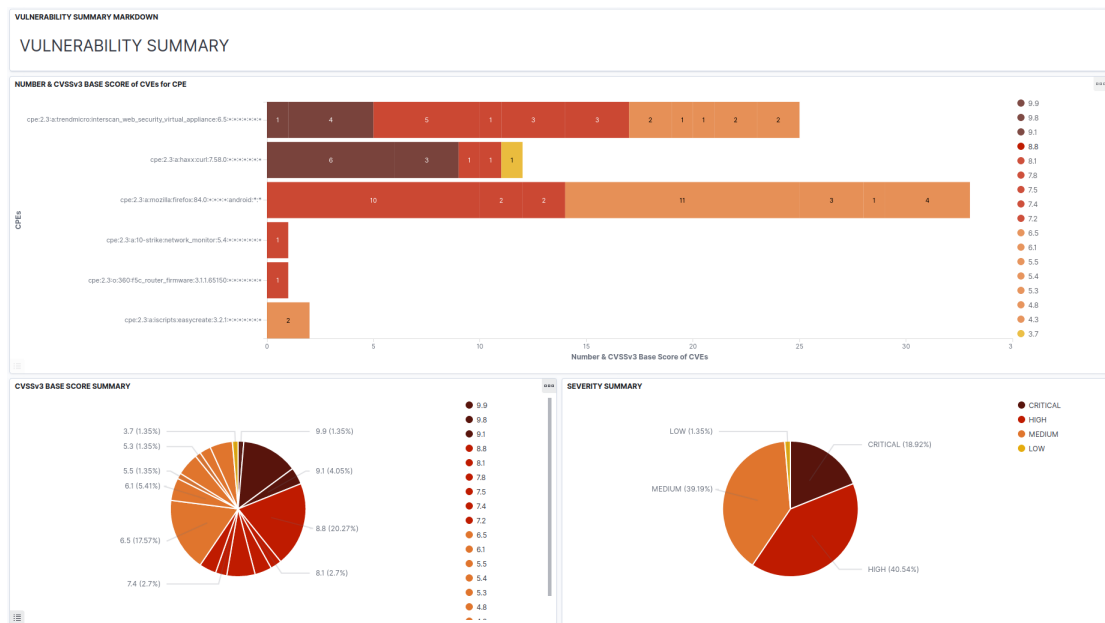


Figure 5.6: VISE: Vulnerability summary dashboard

The *CVSSv3 Base Score summary* visualization gives information regarding the percentage of vulnerabilities with a given level of severity in terms of CVSSv3 Base Score.

Finally, the *Severity summary* visualization shows information regarding the percentage of vulnerabilities with a given qualitative severity level (critical, high, medium, low).

The *Vulnerability technical description* dashboard, Figure 5.7, gives

a more detailed information about the vulnerabilities. In particular, the first visualization shows for each CVE its detailed description. This kind of information is very useful to begin to understand more closely the flaws in own systems.

The *Remediations & CWE for CVE* visualization adds other useful information for each CVE. In particular for each CVE we provide:

- Uniform Resource Locator (URL)s in which remedies for these vulnerabilities are described
- a CWE identifier number of the weakness type

VULNERABILITY TECHNICAL DESCRIPTION MARKDOWN		
VULNERABILITY TECHNICAL DESCRIPTION		
VULNERABILITY DESCRIPTION		
CVE-ID	DESCRIPTION	
CVE-2018-1000120	A buffer overflow exists in curl 7.12.3 to and including curl 7.58.0 in the FTP URL handling that allows an attacker to cause a denial of service or worse.	
CVE-2018-1000121	A NULL pointer dereference exists in curl 7.21.0 to and including curl 7.58.0 in the LDAP code that allows an attacker to cause a denial of service	
CVE-2018-1000122	A buffer over-read exists in curl 7.20.0 to and including curl 7.58.0 in the RTSP+RTP handling code that allows an attacker to cause a denial of service or information leakage	
Export: Raw Formatted		
REMEDIATIONS & CWE for CVE		
CVE-ID	REMEDIATIONS	CWE-ID
CVE-2016-9269	https://success.trendmicro.com/solution/1116672	CWE-264
CVE-2016-9314	https://success.trendmicro.com/solution/1116672	CWE-200
CVE-2016-9316	https://success.trendmicro.com/solution/1116672	CWE-79
CVE-2017-11396	https://success.trendmicro.com/solution/1117412	NVD-CWE-noinfo
CVE-2017-6338	https://success.trendmicro.com/solution/1116960	CWE-732
Export: Raw Formatted		

Figure 5.7: VISE: Vulnerability technical description dashboard

The last dashboard, Figure 5.8, provides a more attack-oriented view. In the first visualization, we detail the percentage of each CWE related to the discovered vulnerabilities. This information can help security analysts to have an overview about the types of most present vulnerabilities in their systems.

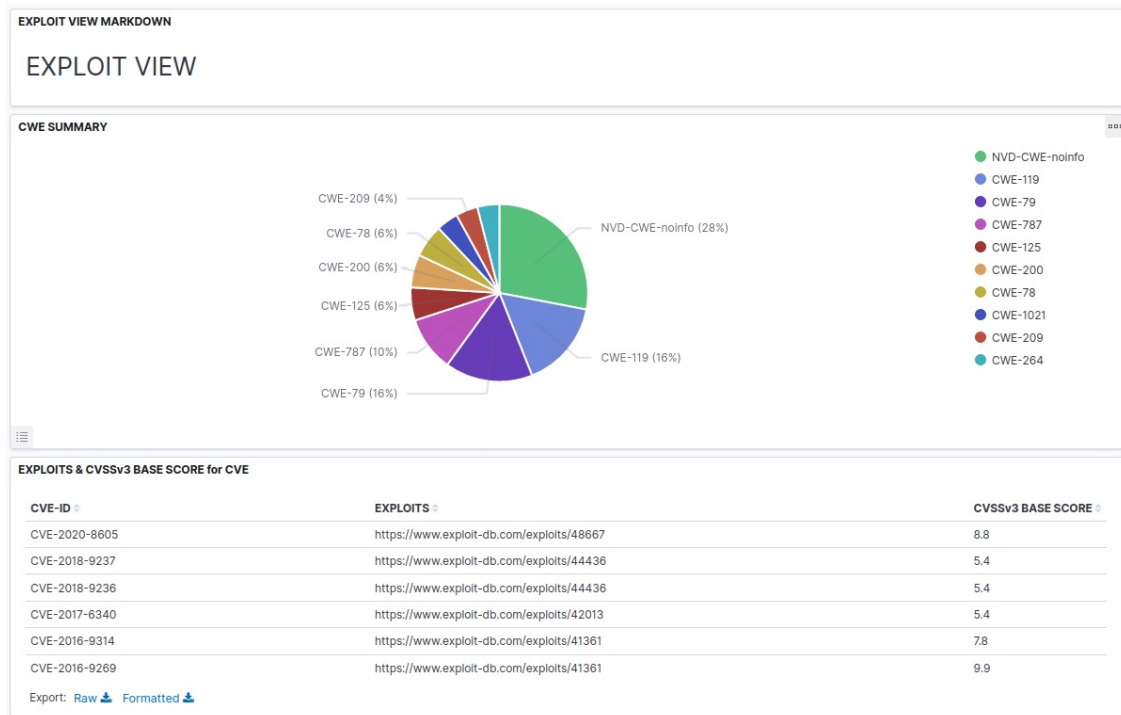


Figure 5.8: VISE: Exploit view dashboard

An additional visualization *Exploit & CVSSv3 Base Score for CVE* provides for each CVE the URLs in which related exploits are available and severity in terms of CVSSv3 Base Score.

The CSV file is a way to retrieve the same information presented in these three dashboards but in a standard format that is more useful of dashboards in some contexts (e.g. in all contexts in which we want to automate operations).

As mentioned in 3.1 we can also start a new search from the CLI.

In this case, the user should provide, as search input, a searching-Card in which he has defined the packages for whom he is interested to make the search.

To do this, the user can run the following command:

```
$ cd VISE/VISE_app/
$ sudo ./main.py static/assets/SearchingCards/SearchingCard
  .xlsx
```

Code Listing 5.7: Start a new search from CLI

The results are reported in the CLI itself as shown in Figure 5.9.

[illegible]

Figure 5.9: VISE: CLI results report

For readability, only the following information is reported in the CLI: *CPE*, *CVE ID*, *CVSSv3 Base Score*, *severity*, *description* and *URLs*.

In addition, links to the dashboards and CSV file are provided.

Chapter 6

Conclusions and Future Work

In this work, we presented a new open source tool to support vulnerability assessment. We have seen how it is possible to integrate heterogeneous information on vulnerabilities received from OSINT sources in order to facilitate the role of security experts who in this way can have all information they need in one place. This clearly allows to reduce the costs in terms of time required to retrieve information, which would typically require security experts to manually browse the web.

Surely, the possibility of having reports available at different levels of detail and in different formats (graphic and otherwise) makes this tool useful for experts of the different levels of a company hierarchy. In fact, a higher level view can be useful for management purposes, while a more detailed view is more useful for those doing a more technical

job.

Several future developments can be foreseen. Among these, the possibility of integrating a higher number of OSINT sources in addition to those already used. This goal can be achieved without too many obstacles thanks to the indexing process on Elasticsearch. For example, one might think of integrating information received from the dark web or from other online sources such as the repository [41] which aggregates all of the PoCs available on GitHub in a single place.

A further addition that might be envisaged is the introduction of a scheduler for the automatic update of the vulnerability database in order to stay up to date with the new daily released disclosures.

Bibliography

- [2] A. Kayode Adesemowo. “Towards a conceptual definition for IT assets through interrogating their nature and epistemic uncertainty”. In: *Computers & Security* 105 (2021), p. 102131. ISSN: 0167-4048. DOI: <https://doi.org/10.1016/j.cose.2020.102131>. URL: <https://www.sciencedirect.com/science/article/pii/S0167404820304041>.
- [3] ISO Central Secretary. *Information technology — IT asset management — Part 1: IT asset management systems — Requirements*. en. Standard ISO/IEC 19770-1:2017(en). Geneva, CH: International Organization for Standardization, 2017. URL: <https://www.iso.org/obp/ui/#iso:std:iso-iec:19770:-1:ed-3:v1:en>.
- [5] National Institute of Standards and Technology. *Common platform enumeration :naming specification version 2.3*. Tech. rep. Interagency Report (IR) 7695. Washington, D.C.: U.S. Department of Commerce, 2011. DOI: 10.6028/NIST.IR.7695.

- [6] Robert W. Shirey. *Internet Security Glossary, Version 2*. RFC 4949. Internet Engineering Task Force, Aug. 2007, pp. 1–365.
URL: <https://tools.ietf.org/rfc/rfc4949.txt>.
- [7] ISO Central Secretary. *Information technology — Security techniques — Information security risk management*. en. Standard ISO/IEC 27005:2008. Geneva, CH: International Organization for Standardization, 2008. URL: <https://www.iso.org/standard/42107.html>.
- [17] SACHIN Umrao, MANDEEP Kaur and GOVIND KUMAR Gupta. “Vulnerability assessment and penetration testing”. In: *International Journal of Computer & Communication Technology* 3.6-8 (2012), pp. 71–74.
- [19] Russ Rogers. *Nessus Network Auditing, 2nd Edition*. Syngress, 2011. ISBN: 9780080558653.

Sitography

- [1] *VISE (Vulnerability Information Search Engine)*. GitHub. Apr. 2020. URL: <https://github.com/antoniofortel1995/VISE> (visited on 17th Apr. 2021).
- [4] *Common Platform Enumeration (CPE)*. URL: <https://csrc.nist.gov/Projects/Security-Content-Automation-Protocol/Specifications/cpe> (visited on 17th Apr. 2021).
- [8] *A glossary of terms used by the CVE Program*. URL: https://cve.mitre.org/about/terminology.html#cve_record (visited on 19th Apr. 2021).
- [9] *CVE and NVD Relationship*. URL: https://cve.mitre.org/about/cve_and_nvd_relationship.html (visited on 19th Apr. 2021).
- [10] *About CWE*. URL: <https://cwe.mitre.org/about/index.html> (visited on 20th Apr. 2021).

- [11] *Frequently Asked Questions - What information is included in a CWE weakness entry?* URL: https://cwe.mitre.org/about/faq.html#information_in_weakness_entry (visited on 20th Apr. 2021).
- [12] *Common Vulnerability Scoring System v3.1: Specification Document.* URL: <https://www.first.org/cvss/v3.1/specification-document#1-1-Metrics> (visited on 20th Apr. 2021).
- [13] *Vulnerability Metrics: CVSS.* URL: <https://nvd.nist.gov/vuln-metrics/cvss> (visited on 20th Apr. 2021).
- [14] Wikipedia contributors. *Exploit (computer security)* — *Wikipedia, The Free Encyclopedia*. 2021. URL: [https://en.wikipedia.org/w/index.php?title=Exploit_\(computer_security\)&oldid=1018600115](https://en.wikipedia.org/w/index.php?title=Exploit_(computer_security)&oldid=1018600115) (visited on 20th Apr. 2021).
- [15] *Vulnerability assessment (vulnerability analysis).* URL: <https://searchsecurity.techtarget.com/definition/vulnerability-assessment-vulnerability-analysis> (visited on 22nd Apr. 2021).
- [16] *The Difference Between Vulnerability Assessments And Vulnerability Management.* URL: <https://hitachi-systems-security.com/the-difference-between-vulnerability-assessments-and-vulnerability-management/> (visited on 22nd Apr. 2021).

- [18] *A Step-By-Step Guide to Vulnerability Assessment*. URL: <https://securityintelligence.com/a-step-by-step-guide-to-vulnerability-assessment/> (visited on 24th Apr. 2021).
- [20] *Search Vulnerability Database*. URL: <https://nvd.nist.gov/vuln/search> (visited on 29th Apr. 2021).
- [21] *Search Common Platform Enumerations (CPE)*. URL: <https://nvd.nist.gov/products/cpe/search> (visited on 29th Apr. 2021).
- [22] *Search CVE List*. URL: https://cve.mitre.org/cve/search_cve_list.html (visited on 29th Apr. 2021).
- [23] *CVE List Search Tips*. URL: https://cve.mitre.org/find/search_tips.html (visited on 29th Apr. 2021).
- [24] *CVE Details Home*. URL: <https://www.cvedetails.com/> (visited on 29th Apr. 2021).
- [25] *CVE Details Vulnerability Search*. URL: <https://www.cvedetails.com/vulnerability-search.php> (visited on 29th Apr. 2021).
- [26] *Vulmon - Vulnerability Intelligence Search Engine*. URL: <https://vulmon.com/> (visited on 1st May 2021).
- [27] *Exploit Database Advanced Search*. URL: <https://www.exploit-db.com/search> (visited on 3rd May 2021).

- [28] *VulDB Search*. URL: <https://vuldb.com/?search> (visited on 4th May 2021).
- [29] *Vulnerability & Exploit Database*. URL: <https://www.rapid7.com/db/> (visited on 4th May 2021).
- [30] *Tenable CVE Search*. URL: <https://www.tenable.com/cve/search> (visited on 4th May 2021).
- [31] *NVD Data Feeds*. URL: <https://nvd.nist.gov/vuln/data-feeds> (visited on 7th May 2021).
- [32] *What is Elasticsearch?* URL: <https://www.elastic.co/what-is/elasticsearch> (visited on 7th May 2021).
- [33] *SearchSploit – The Manual*. URL: <https://www.exploit-db.com/searchsploit> (visited on 7th May 2021).
- [34] *Flask*. URL: <https://palletsprojects.com/p/flask/> (visited on 7th May 2021).
- [35] *cve-analysis*. GitHub. URL: <https://github.com/joshbressers/cve-analysis> (visited on 8th May 2021).
- [36] *elasticsearch Python library*. URL: <https://pypi.org/project/Elasticsearch/> (visited on 8th May 2021).
- [37] *requests Python library*. URL: <https://pypi.org/project/requests/> (visited on 8th May 2021).
- [38] *What is Kibana?* URL: <https://www.elastic.co/what-is/kibana> (visited on 7th May 2021).

- [39] *Elastic stack (ELK) on Docker*. GitHub. URL: <https://github.com/deviantony/docker-elk> (visited on 8th May 2021).
- [40] *cve-searchsploit*. URL: <https://pypi.org/project/cve-searchsploit/> (visited on 15th May 2021).
- [41] *PoC in GitHub*. URL: <https://github.com/nomi-sec/PoC-in-GitHub> (visited on 24th May 2021).

Acknowledgements

Innanzitutto, mi sembra doveroso ringraziare il Prof. Simon Pietro Romano per l'enorme umanità, professionalità e disponibilità che lo contraddistinguono.

Poi passerei a ringraziare la restante parte delle persone che hanno contribuito in varie forme alla realizzazione di questo lavoro, compreso il mio correlatore e l'intero team VISE.

Un ringraziamento speciale va alla mia famiglia per avermi sempre sostenuto e per l'enorme pazienza che hanno avuto con me durante tutto il percorso.

Infine, ringrazio tutti quelli che hanno creduto in me e che hanno contribuito in svariati modi al mio percorso di crescita professionale.