

## **Gestor de Risco aplicado à área de cibersegurança**

**DAVID PÓVOAS RESENDE**

novembro de 2022

# RISK MANAGEMENT APPLICATION FOR THE CYBERSECURITY FIELD

David Póvoas Resende

Departamento de Engenharia Eletrotécnica  
Mestrado em Engenharia Eletrotécnica e de Computadores  
Área de Especialização em Sistemas Autónomos

**2022**



Relatório elaborado para satisfação parcial dos requisitos da Unidade Curricular de  
Tese/Dissertação do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: David Póvoas, Nº 1200473, [1200473@isep.ipp.pt](mailto:1200473@isep.ipp.pt)

Orientação científica: Professora Doutora Isabel Praça, [icp@isep.ipp.pt](mailto:icp@isep.ipp.pt)

Coorientação científica: Professor Doutor Lino Figueiredo, [lbf@isep.ipp.pt](mailto:lbf@isep.ipp.pt)



Departamento de Engenharia Eletrotécnica  
Mestrado em Engenharia Eletrotécnica e de Computadores  
Área de Especialização em Sistemas Autónomos

**2022**



To my lovely girlfriend Agata who had the patience and the knowledge to help and support me in every single moment, and who I could have not completed my studies without.

À minha querida mãe que sempre me apoiou durante todo o meu percurso e me deu força para prosseguir com os estudos.

## *Acknowledgement*

Firstly, I would like to show my appreciation to the thesis' supervisor, Professora Isabel Cecília Correia da Silva Praça Gomes Pereira, who accompanied me through all the research steps and for the availability and engagement when discussing the research path.

Secondly, I would like to extend my appreciation to my girlfriend, my family and friends who kindly supported me in the research both technically and emotionally.

Finally, I would like to express my gratitude to Instituto Superior de Engenharia do Porto - Politécnico do Porto (ISEP), for giving me the opportunity to complete my Master's Degree.





## Resumo

No cenário moderno de gestão de riscos de segurança cibernética, uma verdade desconfortável é clara: a gestão de riscos cibernéticos numa empresa, de forma a manter arquiteturas e sistemas seguros e em conformidade, está mais difícil do que nunca. Esta gestão passa por um processo contínuo de identificação, análise, avaliação e tratamento das ameaças de segurança cibernética.

Quando se trata de gestores de riscos, geralmente segue-se um processo de quatro etapas, começando com a identificação do risco. Em seguida, o risco é avaliado com base na probabilidade de ameaças que exploram essas vulnerabilidades e o potencial impacto. Os riscos são priorizados e categorizados dependendo da estratégia de mitigação existente, na terceira etapa. Por fim, a quarta etapa, monitorização, é estruturada para a resposta ao risco num ambiente em constante mudança.

Esta tese tem como objetivo o desenvolvimento de uma aplicação de gestão de risco de vulnerabilidades dos *assets* encontrados numa topologia de rede. Esta aplicação web tem por base a *framework* Flask e o uso da ferramenta *open-source* Nmap, para a realização da deteção dos *assets* e todos os serviços que estes incluem. Para a deteção das vulnerabilidades a aplicação conta com uma ligação através de duas APIs, uma para o repositório NVD e outra para o repositório VulDB de forma a identificar as vulnerabilidades existentes de cada serviço encontrado. Toda a informação encontrada é guardada numa base de dados com base em SQLite.

De notar que o uso do Nmap é proibido por Lei (109/2009) mas se autorizada de forma evidenciada com permissão das partes envolvidas, pode ser usado.

Os testes efetuados utilizam a ferramenta VirtualBox para simular virtualmente a rede de um hospital virtual criado num outro projeto. Os resultados são por fim detalhados num relatório através da aplicação web.

Este projeto conseguiu de forma bem sucedida o desenvolvimento de um gestor de risco funcional através de uma aplicação web capaz de mapear uma rede e encontrar os *assets* com vulnerabilidades. Igualmente bem-sucedida foi a implementação da detecção de vulnerabilidades através de repositórios externos. Por fim esta tese implementou com sucesso uma comparação entre *scans* de forma a descobrir quais vulnerabilidades foram corrigidas ou que novas vulnerabilidades possam existir em determinados *assets*.

Contudo não foi possível uma implementação com sucesso desta aplicação num projeto já existente usando *React*. Igualmente não foi realizada uma forma automatizada da realização de *scans*. Por último, devido aos recursos disponíveis, a rede hospitalar virtual foi bastante reduzida.

## *Palavras-Chave*

Cybersecurity, Application, Risk Management, Flask, API, NVD, VulDB

## *Abstract*

In the modern scenario of cybersecurity, one uncomfortable truth is clear, the risk management of a company and/or institution in order to keep all its systems and information secure, is harder than ever. This management goes through a continuous cycle of identification, analysis, evaluation, and treatment of the daily threats. Usually risk management follows four steps, starting by the identification of the risk, then the evaluation of it with the probability of actors exploiting any existing vulnerability and the consequent impact. Given this analysis, the risks are prioritized and categorized depending on the mitigation strategy in place, and finally the last step is the monitorization, in other words, the structure that answers to the risk in an ever-changing environment.

This Thesis has as objectives the development of a Risk Management web application that scans all the assets of a given network. This web application uses the framework Flask for its development and the open-source tool Nmap for the asset scanning and all the services running on each live host. For the detection of vulnerabilities, the application has a connection to the repository NVD through an API, and to the repository VulDB through another API, in order to identify all the existing vulnerabilities associated with the services found during the scan. All this information is stored on a SQLite database.

According to the Portuguese law (109/2009), the use of the tool Nmap is strictly forbidden but can be authorized for use given the proper permission from the involved parties.

The experiments use the virtualization software VirtualBox to simulate a network of a virtual Hospital that was already created in another project. All the results are in the end available as a report through the web application.

This project was able to develop a functional risk management web application, capable of scan a network in order to find the vulnerable assets.

Equally successful was the implementation of the vulnerability detection through the use of external vulnerability databases. Finally, this thesis successfully implemented a comparison between scans to discover which vulnerabilities have been corrected and which ones appear as new in specific assets.

However it was not possible to integrate in a successful way this application to an already existing project using React. Equally not accomplished was an automated way to schedule periodic scans. Finally, given the available resources, the hospital virtual network was largely reduced.

## *Keywords*

Cybersecurity, Application, Risk Management, Flask, API, NVD, VulDB

# Index

ACKNOWLEDGEMENT .....	I
RESUMO .....	III
PALAVRAS-CHAVE .....	IV
ABSTRACT.....	V
KEYWORDS.....	VI
INDEX .....	VII
LIST OF FIGURES .....	IX
LIST OF TABLES .....	XIII
LIST OF SOURCE CODE .....	XIV
LIST OF ACRONYMS.....	XV
<b>1. INTRODUCTION .....</b>	<b>1</b>
1.1. CONTEXT .....	1
1.2. OBJECTIVES .....	2
1.3. DOCUMENT STRUCTURE .....	3
<b>2. STATE OF THE ART .....</b>	<b>5</b>
2.1. WHAT IS A SOFTWARE VULNERABILITY? .....	5
2.2. HOW VULNERABILITIES ARE REPORTED .....	5
2.3. CVE .....	6
2.4. CVSS.....	7
2.5. WHAT ARE CYBER THREATS? .....	8
2.6. WHAT IS CYBERSECURITY RISK MANAGEMENT? .....	9
2.7. CYBER RISK MANAGEMENT FRAMEWORKS.....	12
2.8. TECHNOLOGIES .....	29
2.8.1. <i>Python</i> .....	29
2.8.2. <i>Flask</i> .....	30
2.8.3. <i>Django</i> .....	31
2.8.4. <i>Comparison between Flask and Django</i> .....	32
2.9. VULNERABILITY MANAGER .....	33
2.9.1. <i>Nmap</i> .....	33
2.10. NETWORK SIMULATION .....	34
2.10.1. <i>VirtualBox</i> .....	34
2.10.2. <i>GNS3</i> .....	37

<b>3.</b>	<b>VULNERABILITY MANAGEMENT .....</b>	<b>39</b>
3.1.	INTRODUCTION .....	39
3.2.	THE USE OF NMAP .....	41
3.3.	HOST DISCOVERY .....	43
3.4.	PORT SCANNER .....	45
3.5.	OS DETECTION .....	46
3.6.	SERVICE AND VERSION DETECTION.....	47
3.7.	SSL/TLS DETECTION .....	49
3.8.	METHOD .....	50
3.9.	API .....	51
3.10.	SSL/TLS VULNERABILITIES .....	60
<b>4.</b>	<b>DATABASE .....</b>	<b>63</b>
<b>5.</b>	<b>APPLICATION .....</b>	<b>71</b>
5.1.	DASHBOARD.....	74
5.2.	SCAN .....	74
5.3.	SCAN HISTORY .....	76
5.4.	REPORT .....	84
5.5.	SCAN COMPARATION .....	87
<b>6.</b>	<b>TESTS AND RESULTS.....</b>	<b>89</b>
6.1.	PHASE ONE.....	90
6.2.	PHASE TWO .....	92
6.3.	PHASE THREE .....	98
6.4.	LIMITATION.....	105
<b>7.</b>	<b>CONCLUSIONS .....</b>	<b>107</b>
7.1.	OBJECTIVES ACHIEVED.....	107
7.2.	LIMITATIONS .....	108
7.3.	FUTURE WORK .....	109
7.4.	FINAL CONSIDERATIONS.....	110
	BIBLIOGRAPHY.....	111

# List of Figures

FIGURE 1: RISK MANAGEMENT FRAMEWORK [14]	13
FIGURE 2: ISO/IEC 27005 MODEL [17]	16
FIGURE 3: FAIR FRAMEWORK MODEL [29]	18
FIGURE 4: OCTAVE METHODOLOGY [17]	21
FIGURE 5: BOWTIE METHODOLOGY [17]	23
FIGURE 6: BOWTIE METHOD [21]	25
FIGURE 7: BOWTIE METHODOLOGY [35]	26
FIGURE 8: FLASK [38]	30
FIGURE 9: DJANGO [39]	31
FIGURE 10: NMAP [40]	33
FIGURE 11: VIRTUALBOX [41]	34
FIGURE 12: NAT NETWORK MODE [42]	36
FIGURE 13: HOST-ONLY NETWORK MODE [42]	36
FIGURE 14: GNS3 NETWORK TOPOLOGY [44]	37
FIGURE 15: VULNERABILITY DETECTION PROCESS	51
FIGURE 16: API JSON SCHEMA - 1	53
FIGURE 17: API JSON SCHEMA - 2	54
FIGURE 18: API JSON SCHEMA - 3	55
FIGURE 19: VULDB API JSON SCHEMA	59
FIGURE 20: DATABASE OVERVIEW	64
FIGURE 21: TABLE SCANNUMBER	65
FIGURE 22: TABLE HOSTDISCOVERY	65
FIGURE 23: TABLE SERVICESCAN	66
FIGURE 24: TABLE SSLDISCOVERY	66
FIGURE 25: TABLE CVESCAN	67
FIGURE 26: DATABASE LINKS	69
FIGURE 27: HOSTDISCOVERY TABLE EXAMPLE	69
FIGURE 28: SERVICESCAN TABLE EXAMPLE	70
FIGURE 29: DIAGRAMA DA APLICAÇÃO	73
FIGURE 30: DASHBOARD	74
FIGURE 31: SCAN INPUT OPTIONS	75
FIGURE 32: NEW BASIC SCAN	75
FIGURE 33: NEW FULL SCAN	76

FIGURE 34: NEW CUSTOM SCAN .....	76
FIGURE 35: HISTORY OF SCANS .....	77
FIGURE 36: HOSTS DISCOVERED .....	79
FIGURE 37: VULNERABILITIES FOUND - 1.....	79
FIGURE 38: VULNERABILITIES FOUND - 2.....	80
FIGURE 39: VULNERABILITIES FOUND - 3.....	80
FIGURE 40: INFORMATIVE CONTENT .....	81
FIGURE 41: INFORMATIVE SERVICES PAGE.....	81
FIGURE 42: INFORMATIVE PORTS PAGE .....	82
FIGURE 43: INFORMATIVE OPERATING SYSTEMS PAGE.....	82
FIGURE 44: INFORMATIVE SSL/TLS PAGE .....	82
FIGURE 45: SCAN REPORT - 1 .....	85
FIGURE 46: SCAN REPORT - 2 .....	86
FIGURE 47: SCAN COMPARATION .....	87
FIGURE 48: SCAN COMPARATION RESULTS .....	88
FIGURE 49: CORRECTED VULNERABILITIES .....	88
FIGURE 50: NEW VULNERABILITIES .....	88
FIGURE 51: OWASP VIRTUAL MACHINE.....	90
FIGURE 52: METASPLOIT VIRTUAL MACHINE.....	91
FIGURE 53: VIRTUAL MACHINE SETUP.....	91
FIGURE 54: VIRTUALBOX NAT NETWORK.....	92
FIGURE 55: FIRST NETWORK SETUP [55].....	93
FIGURE 56: MACHINES ON THE FIRST SETUP .....	93
FIGURE 57: SSH CONNECTION.....	94
FIGURE 58: INTERNAL NETWORK.....	95
FIGURE 59: VIRTUAL MACHINE CONFIGURATION .....	96
FIGURE 60: ROUTER SETUP .....	96
FIGURE 61: PING AND TRACE OTHER VIRTUAL MACHINES.....	97
FIGURE 62: LACK OF VISUAL ADAPTER .....	98
FIGURE 63: ETH1 ADAPTER.....	98
FIGURE 64: NETWORK ADAPTER CONFIGURATION.....	98
FIGURE 65: VIRTUAL HOSPITAL [58].....	99
FIGURE 66: PHASE TWO NETWORK WITH ADDITIONAL HOSTS .....	100
FIGURE 67: SCAN RESULTS HISTORY – FIRST NETWORK.....	100
FIGURE 68: SCAN RESULTS HOSTS – FIRST NETWORK .....	101
FIGURE 69: FIRST NETWORK SCAN REPORT .....	101
FIGURE 70: PHASE THREE NETWORK .....	102



FIGURE 71: SCAN RESULTS HISTORY – SECOND NETWORK.....	103
FIGURE 72: SCAN REPORT – SECOND NETWORK.....	103
FIGURE 73: COMPARATION OF API RESULTS.....	105



# List of Tables

TABLE 1: RISK APPROACHES .....	11
TABLE 2: RISK MANAGEMENT FRAMEWORKS .....	12
TABLE 3: NIST SP 800-37 FRAMEWORK .....	14
TABLE 4: NIST RMF [27] [28] .....	15
TABLE 5: ISSO 27005 .....	17
TABLE 6: FAIR FRAMEWORK .....	20
TABLE 7: OCTAVE .....	22
TABLE 8: EBIOS .....	24
TABLE 9: BOWTIE .....	27
TABLE 10: RISK MANAGEMENT METHODS, RA – RISK ASSESSMENT, RM – RISK MANAGEMENT RQ – RISK QUANTIFICATION .	28
TABLE 11: FLASK VS DJANGO.....	32
TABLE 12: VIRTUALBOX MODES [29] .....	35
TABLE 13: COMPUTER SPECIFICATION .....	89
TABLE 14: COMPARATION OF SCAN TIME .....	104

## List of Source Code

LISTING 1: PROCESS OF NMAP SCAN IN CODE.....	43
LISTING 2: NMAP HOST DISCOVERY SCAN .....	43
LISTING 3: HOST FOUND XML FILE.....	44
LISTING 4: CONVERT TO NMAP FORMAT .....	45
LISTING 5: NMAP PORT SCAN.....	45
LISTING 6: TOP 1000 PORTS XML FILE .....	46
LISTING 7: NMAP OS DETECTION FUNCTION.....	47
LISTING 8: SERVICE AND VERSION DETECTION - CODE.....	48
LISTING 9: VARIOUS UPDATES AND INSERTS INTO THE DATABASE.....	48
LISTING 10: SSL/TLS XML FILES CREATION .....	49
LISTING 11: HOST AND PORT INFORMATION EXTRACTION FROM XML FILE .....	50
LISTING 12: NMAP SSL/TLS DETECTION FUNCTION.....	50
LISTING 13: : API REQUEST .....	56
LISTING 14: NVD API FUNCTION .....	57
LISTING 15: VULDB API REQUEST .....	58
LISTING 16: VULDB API FUNCTION .....	60
LISTING 17: SSL/TLS CERTIFICATE CHECK .....	61
LISTING 18: SSL/TLS PROTOCOL VERSION CHECK .....	61
LISTING 19: FLASK INITIATING .....	71
LISTING 20: FLASK ROUTES.....	78
LISTING 21: INFORMATIVE WEBPAGES – CODE .....	83
LISTING 22: SCAN REPORT CODE .....	84

## *List of Acronyms*

**ACK** – Acknowledging

**API** – Application Programming Interface

**CIA** – Confidentiality, Integrity and Availability

**CPE** – Common Platform Enumeration

**CSF** – Cyber Security Framework

**CVE** – Common Vulnerabilities and Exposures

**CVSS** – Common vulnerability Scoring System

**EOF** – End of file

**ICMP** – Internet Control Message Protocol

**NVD** – National Vulnerability Database

**OS** – Operating System

**RM** – Risk Management

**RMF** – Risk Management Framework

**ROI** – Return of Investment

**SSL** – Secure Sockets Layer

**SYN** – Synchronize Sequence Number

**TCP** – Transmission Control Protocol

**TCP/IP** – Transmission Control Protocol/Internet Protocol

**TEDI** – Thesis / Dissertation

**TLS** – Transport Layer Security

**UDP** – User Datagram Protocol

**WSGI** – Web Server Gateway Interface

# 1. INTRODUCTION

Within the scope of the Curricular Unit Tese / Dissertação Mestrado de Engenharia Eletrotécnica e de Computadores – Sistemas Autónomos (TEDI) , this project aims to expand and consolidate the self-taught knowledge attained by the student and apply it to a real situation. All the research carried out, as well as all the necessary steps to the completion of this Thesis, are covered in this document.

The introductory chapter will be responsible for detailing the context of this Thesis as well as the problem interpretation.

## 1.1. CONTEXT

This Thesis development emerged by the advice of the main teacher coordinator who proposed the creation of a Risk Management application for the cybersecurity field. Despite the different background of the student, the proposal was gladly accepted and extended by the same. As such, this project took an extra time due to the long-needed research and development of some of the steps. Initially there was another implementation to be added, the combination of this project to an already existing one.

This second, a React Application Programming Interface (API), was supposed to welcome this Risk Management and converge it into one bigger project of Vulnerability detection with a graphical user's interface. However, the merge of these two projects with the use of React, was beyond the scope of the student given the time left for the Thesis completion. As such, a decision was made not to combine both projects at the present time, although a continuation could be done and indeed create the complete project.

## 1.2. OBJECTIVES

In the modern scenario of cybersecurity, one uncomfortable truth is clear, the risk management of a company and/or institution in order to keep all its systems and information secure, is harder than ever. This management goes through a continuous cycle of identification, analysis, evaluation, and treatment of the daily threats. In these types of risk managements, generally four steps are followed. Starting by the identification of the risk, then the evaluation of it with the probability of actors exploiting any existing vulnerability and the consequent impact. Given this analysis, the risks are prioritized and categorized depending on the mitigation strategy in place, and finally the last step is the monitorization, in other words, the structure that answers to the risk in an ever-changing environment.

This Thesis has as objectives the development of a Risk Management application that scans all the assets of a given network and detects any possible vulnerability with the connection to known vulnerabilities databases. The scan itself is done using the popular open-source tool Nmap, running each individual scan through a python script containing the module subprocess. A different xml file is generated for every phase of the scan, being then used to gather the necessary information to store on the SQLite database.

Through the developed Flask web application, it's possible to choose different scan types, a basic scan, a full scan or a customised scan. The main difference between them is the number of ports checked. All the other information such as, hosts found, services, operating system, SSL/TLS and vulnerabilities, remain the same. The validation of detected vulnerabilities is done through APIs to the National Vulnerability Database (NVD) and the Vulnerability Database (VulDB) using the Common Platform Enumeration (CPE). The web application displays all the information about the vulnerabilities found in



a dashboard filtered by date, the possibility to revisit past scans and the creation of a new scan. It is also possible to download a PDF report of each individual scan.

The testing will be carried out in a simulated network previously designed for a virtual hospital, using the virtualization software VirtualBox, with the creation of multiple virtual machines.

### 1.3. DOCUMENT STRUCTURE

There are eight chapters in this document. To better present the information to the reader, each chapter has its subsequent sections. As a result of this, the document is divided into the following sections: Introduction, State of the Art, Vulnerability Scanner, Vulnerability Detection, Database, Application, Tests and Results and the Conclusions.

The first part introduces the project, it's contextualization and motivation, followed by the objectives proposed and finally the documented structure.

The State of the Art informs the reader of the technologies and methods used in the present and helps familiarize with certain terms and concepts.

The vulnerability scanner chapter dives deep into what the many phases of a scan and the type of information extracted from it. Introduces the use of Nmap, a widely used open-source tool for network scanning, on the project and explains the code and logic used to perform the scan.

The vulnerability detection chapter explains the process behind detecting any vulnerabilities found from the hosts discovered on the network and how using an API to the National Vulnerability Database and another to the Vulnerability Database, it's possible to determine if a host can or not be exploitable and the dangers associated with it.

The fifth chapter elaborates on the usage of the database, all the tables created and the links between them. As an essential part of project, the database is what allows for all the connectivity between the scans performed and the display on the web application.

The application chapter details the development using Flask and the different pages created, with all the options and interactions possible. The user will be able to visualize the presentation of the web application and the information gathered in a graphic way.

The tests and results show the creation of the virtual network of the hospital and the application placed to a test of finding all possible assets in the hospital network and determine any vulnerability associated with them. An explanation of the usage of VirtualBox for this purpose is also detailed.

The final chapter, which is represented by the conclusions, addresses the objectives that were achieved regarding the project, and the implementations and milestones present on this thesis. Following this, the constraints and limitations are also mentioned and finally the possible future works and additions to this project, with the final considerations summarized by the author.

## 2. STATE OF THE ART

### 2.1. WHAT IS A SOFTWARE VULNERABILITY?

A software vulnerability can be classified as any weakness in the codebase that can be exploited for a multiple of reasons [1] [2]. The sources of their vulnerabilities can be a result a variety of reasons, some of the most common are coding mistakes, faulty logic, inadequate validation mechanisms, lack of protection against buffer overflows, as well as unprotected APIs and issues in third-party libraries.

Independent from the source, each vulnerability presents a risk to users and organizations. Unless they are discovered and patched or fixed in software updates, perpetrators can, depending on the risk, exploit them to steal data, deploy and spread malware, cause outages or damage systems.

### 2.2. HOW VULNERABILITIES ARE REPORTED

The way in which vulnerabilities are reported will depend on the type of software they are discovered on as well as the type of vulnerability they appear to be.

Also, the perceived importance of the vulnerability will always be affected by the finder in the way it is reported.

Generally, vulnerabilities are found and reported by penetration testers, security researchers, and users themselves.

When vulnerabilities are minor or can be easily fixed, these issues are more likely to go unreported. On the other hand, a severe issue can also go unreported if discovered by a black hat researcher or cybercriminal.

In the case a vulnerability is found in a proprietary software, there are two options: either to report directly to the vendor, or report to a third-party oversight organization, such as MITRE [3]. In open-source software, it may be reported to the project manager or the community.

In the scenario where the vulnerability is reported to a corporation like MITRE, the organization confirms with different sources the correct assignment of the vulnerability ID number and the vendor or project manager is notified [4]. The same vendor or project manager has a 30 to 90 days window to develop a patch for the issue before the information is made public.

With new services and application emerging every single day and being used by different companies, institutions and governments, it makes the effort to monitor all software vulnerabilities dramatically huge and unrealistic. For this reason, from a viewpoint of intelligent data processing, software vulnerability analysis uses enumeration to build a database of vulnerabilities and also the infrastructure of other analysis stages by providing the findings through verified observations [5]. On this area, a lot of work was done to unify the vulnerability enumeration and Common Vulnerabilities and Exposures (CVE) was founded by MITRE [6].

## 2.3. CVE

Usually, the description of vulnerabilities is full of vague concepts by different security communities, with different authorities managing security vulnerabilities with different numbering systems. Meaning that, when conducting a study in a vulnerability in detail, a collection of descriptions from different trusted sources should be made to carefully not fall into misled and vague concepts.

Vulnerability enumeration emerged inside the software corporations to manage the required security of their products. Some famous corporations have built vulnerability databases with published alert information, such as, Microsoft's Security Bulletins [7], HP's Security

Bulletins [8], and IBM's ISS X-Force Database [9]. The security communities and research institutes which may be sponsored by governments, foundations, or corporations such as BugTraq [10], US-CERT's Technical Cyber Security Alerts [11], and so on, publish and discuss the vulnerabilities collected or submitted by various sources. These collection systems are the typical enumeration databases which focus on specific software products security issues, which sometimes can create duplication and confusion in different vulnerability databases.

An effort to unify all these vulnerability enumerations began in 1999, creating the Common Vulnerabilities and Exposures (CVE) by MITRE. CVE is intended to provide the standard for information security and a unified directory for prevailing software vulnerabilities, sharing data across different and separate vulnerability repositories. Although CVE is not a full set of all existing vulnerabilities, it collects from a variety of data sources, obtaining a notable success and with over 60.000 items, it became an actual real industry and academic vulnerability enumeration standard.

## 2.4. CVSS

The Common vulnerability Scoring System (CVSS) is a set of free, open standards that allows for security and IT professionals, testers, and software developers with a standardized process for assessing vulnerabilities. The CVSS can be used for assessing the threat level of each vulnerability and consequently prioritize mitigation measures accordingly.

These standards are maintained by the Forum of Incident Response and Security Teams (FIRST), a non-profit security organization. A scale from 0.0 to 10.0 is used, where 10.0 represents the highest severity [12].

The scoring is based on a combination of several subsets of scores. The only actual requirement for the categorization of a vulnerability using CVSS is the completion of the base score components. Nevertheless, it is recommended the inclusion of temporal scores and environmental metrics for a more accurate evaluation by the reporters.

## 2.5. WHAT ARE CYBER THREATS?

The term cyber threat generally applies to any vector that can be exploited in order to breach security, cause damage to the organization, or exfiltrate data.

Common threat categories facing modern organizations include:

**Adversarial threats** – including third-part vendors, trusted insiders, insider threats, established hacker collectives, privileged insiders, suppliers, corporate and nation-states. This category includes malicious software (malware) created by any of these entities as well. With specialized tooling and trained security staff by establishing a security operations center (SOC), large organizations can mitigate these threats.

**System failure** – when a system fails, it may cause data loss and even lead to a disruption in business continuity. Timely support, up-to-date backups, redundancy measures and running high-quality equipment is an imperative procedure in critical systems.

**Human error** – any user may accidentally download malware or get tricked by social engineering schemes like phishing campaigns. A storage misconfiguration may expose sensitive data. In order to prevent and mitigate these threats, an employee training program should be established as well as enforcing strong security controls.

Key threat vectors that affect many organizations are:

- Unauthorized access – which may be the result of malicious attackers, malware, and employee error.
- Misuse of information by authorized users – an insider threat may misuse information by altering, deleting, or using data without authorization.
- Data leaks – threat actors or for example cloud misconfiguration may lead to leaks of personally identifiable information (PII) and other types of sensitive data.
- Loss of data – poorly configured replication and backup processes may lead to data loss or accidental deletion.
- Service disruption – downtime may cause reputational damage and revenue losses. It may be accidental, or the result of denial of service (DoS) attack.

## 2.6. WHAT IS CYBERSECURITY RISK MANAGEMENT?

Cybersecurity risk management is a strategic approach to prioritizing threats. The implementation of risk management practices by organizations should ensure the most critical threats are handled in a timely manner. This helps identify, analyze, evaluate, and address threats based on the potential impact each threat can bring [13].

A good risk management strategy acknowledges that organizations cannot eliminate all system vulnerabilities or block all cyber-attacks. Establishing a cybersecurity risk management initiative helps organizations attend first to the most critical flaws, threat trends, and attacks [14] [15].

It involves the identification of cyber-attacks that may negatively impact the assets. The organization is required to determine the likelihood of the occurrence of these attacks and define the impact each attack may incur. To understand the main goal of risk management, it's fundamental to emphasize the concepts and the process employed.

Generally, risk should be described by 3 components [16]:

- The probability of occurrence;
- Vulnerabilities of the infrastructure;
- Impacts of the successful threat.

The process of risk management can be divided into four stages [17]:

- Identifying risk – evaluate the organization's environment in order to identify current or potential risks that could affect business operations.
- Assess risk – analyzing identified risks to see how likely they are to impact the organization, and what the impact could be.
- Control risk – define methods, procedures, technologies, or other measures that can help the organization mitigate the risks.
- Monitoring – evaluating, on an ongoing basis, how effective controls are at mitigating risks, and adding or adjusting controls as needed.

### **Risk identification:**

The starting point of a Risk Management (RM) involves identifying the organization environment, its IT infrastructure, and the limits of the RM study [18]. This stage covers:

- System characterization – Identifying the boundaries of the targeted organizations, components, and resources of relevance.
- Assets identification – Identifies the crucial components of the organization in order to conduct a vulnerability examination. The assets can either be the business type, including information and processes or IT type, which are components of the information system that support the other assets.
- Security objective determination – Describes the security objectives for business assets, which mostly rely on confidentiality, integrity and availability (CIA).

### **Risk analysis:**

The goal of risk analysis is to assess the probability of an oncoming threat and its magnitude [19]. Thus, it's in this process that the components of risk are distinguished:

- Threats identification – Helps recognize the existence of risks.
- Vulnerabilities identification – Identifies security weaknesses or flaws that may be exploitable.

There are several approaches to assess risk components. They can be categorized as:

- Quantitative – uses several variables in order to describe the probability of an event occurring and the consequent loss [20] .
- Qualitative – uses descriptive variables to express the same probability and the consequences of risk [21].

The choice of the ideal approach is often based on the availability and the type of data.



Table 1: Risk approaches

Risk analysis approaches	Advantages	Disadvantages
Quantitative	+ Highly detailed	- Requires too much time and resources
	+ Helps calculate the cost of effectiveness of the RM process	- The impact precision seems unclear and requires a qualitative interpretation
Qualitative	+ Favorable for immediate improvement	- Inaccurate result in most cases
	+ Does not require an accurate calculation of the asset value and the cost of monitoring	- Making a cost effectiveness of the RM process is often difficult

### Risk mitigation:

After the risk analysis, a response strategy is needed. In this stage, there are four primary options for dealing with risks:

- Risk avoidance – Eliminating the exposure of assets or the source of risks.
- Risk limitation – Implementing appropriate controls limiting the exposure to risks.
- Risk Transference – Outsources security services, transferring the risk responsibility to them.
- Risk acceptance – Develops a risk mitigation plan having accepted the existing risks.

### Risk Monitoring:

Finally, after the implementation of a chosen risk mitigation plan, risk monitoring provides assurance that the current security controls are implemented in an effective way.

## 2.7. CYBER RISK MANAGEMENT FRAMEWORKS

There are several cyber risk management frameworks, each providing standards that organizations can use to identify and mitigate risks. Security leaders and senior managers use these frameworks to assess and improve the security posture of the organization [22].

These types of frameworks can help organizations effectively assess, mitigate, monitor risks and define security processes and procedures to address them.

Currently there are three types of cybersecurity frameworks, as explained by F. Kim [23]:

Table 2: Risk Management Frameworks

Control Frameworks	Program Frameworks	Risk Frameworks
Provides a baseline set of controls	Assesses the state of the security program	Identifies, measures, and quantifies the risk
Develops a basic strategy for the security team	Builds a comprehensive security program	Prioritizes security activities
Prioritizes control implementation	Simplifies the communication between the security team and the business leaders	Defines the key process steps to assess and/or manages the risk

A control framework, using an analogy, is similar to the need for having a good understanding of the language and the vocabulary before writing a book. In this case, various control frameworks describe security controls that can be implemented. NIST 800-53 [24] is an example of a control framework that lists in categories multiple security controls that can and/or should be implemented. On the other hand, a program framework, and continuing with the analogy, is a style guide that serves as reference point for any writing done by the author. ISO 27001 [25] is a program framework that introduces policies, processes, and procedures for a more robust security program. Finally, a risk management allows the prioritization of the security implementation and manages the risk according to the company/institution policy. Follows a list of commonly used cyber risk management frameworks [17]:

## NIST CSF/ NIST RMF (NIST 800-37)

The National Institute of Standards and Technology Cyber Security Framework (NIST CSF) is a popular framework [26]. Providing a comprehensive set of best practices that standardize risk management, it defines a map of activities and outcomes related to the core functions of cybersecurity risk management – protect, detect, identify, respond, and recover. It is meant to be applied to any organization looking to build a cybersecurity program.

NIST CSF, has a Program Framework, being created as a result of collaboration between the government and the private sector, serving has a high-level, strategic view of an organization management of cybersecurity risk. However, the framework complements and does not replace an organization risk management program. Thus, the CSF does not intend to replace the Risk Management Framework (RMF).

NIST RMF (NIST 800-37) has been mandatory for use by federal agencies and organizations that need to handle federal data and information.

The main differences between NIST CSF and NIST RMF reside on the first being a broader framework and including standards and guidelines for describing their target state for cybersecurity, identify and prioritize opportunities for improvement, assess the progress to achieve the desired target state and communicate among internal and external stakeholders about the risks. The second focuses more on the risk management side.

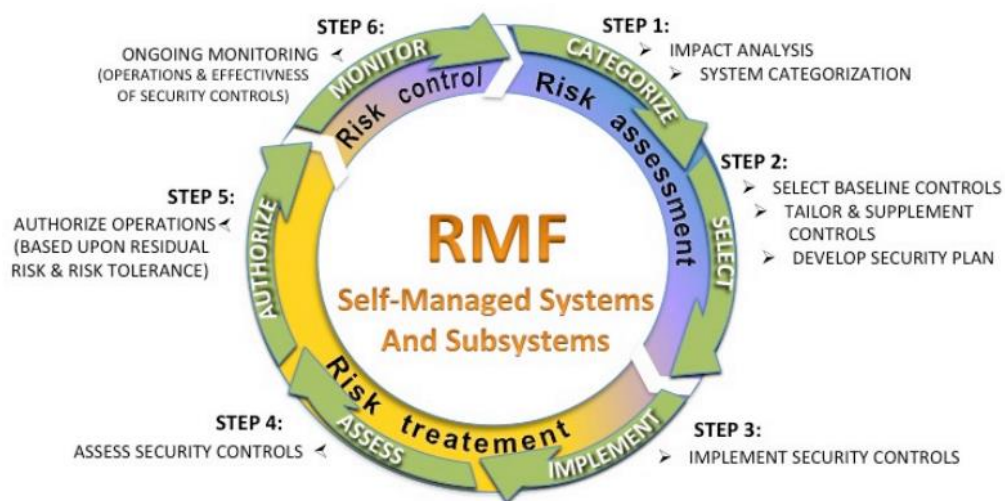


Figure 1: Risk Management Framework [14]

Table 3: NIST SP 800-37 Framework

<b>Risk Management activities</b>	<b>NIST SP 800-37 RMF Steps</b>	<b>RISK Management Framework description</b>
<b><i>Risk assessment</i></b>	1. Categorize	Defines the environment, CIA value, impact analysis, describes the system and the information processed, stored, and transmitted.
	2. Select	The type of security controls appropriate based on the categorization, organization assessment of risk and local conditions.
<b><i>Risk treatment</i></b>	3. Implement	Use security controls, describing how they are employed within the system and their effects on the environment
	4. Assess	Evaluate the effectiveness of the security controls, their correct implementation, and their quality.
	5. Authorize	Consideration by the top management if the risks are within the acceptance level, identifying how much risk is still present, deciding, and authorizing changes.
<b><i>Risk control</i></b>	6. Monitor	Ongoing monitoring and assessment schedule for security controls in order to measure their effectiveness, documenting the system and operation adjustments, including an impact analysis of changes made.

Table 4: NIST RMF [27] [28]

<b>NIST RMF</b>	
<b>Advantages</b>	<b>Disadvantages</b>
<ul style="list-style-type: none"> <li>+ Allows for a choice of quantitative, qualitative, or semi-quantitative methods of scoring vulnerabilities and risks.</li> </ul>	<ul style="list-style-type: none"> <li>- The key aspect of the system categorization is the so called ‘high-water mark’ approach. Where if a system confidentiality is assessed at high, then the availability and integrity need to also be set at ‘high’ level, incurring expenses for unnecessary security measures. Resulting in under categorization systems by the agencies.</li> </ul>
<ul style="list-style-type: none"> <li>+ Well documented</li> </ul>	
<ul style="list-style-type: none"> <li>+ The NIST framework is valuable in assessing cyber risk and managing cyber risks. The most advanced in terms of disaster and recovery planning</li> </ul>	<ul style="list-style-type: none"> <li>- Requires a detailed understanding of the standards, confusing and extensive use of acronyms.</li> </ul>
	<ul style="list-style-type: none"> <li>- Not an automated tool and does not contain an impact assessment model</li> </ul>

## ISO 27001/27005/31000

The International Organization for Standardization (ISO) created the standard **ISO/IEC 27001** in partnership with the International Electrotechnical Commission (IEC). It is widely regarded and used as the default risk management framework in most organizations. This cybersecurity framework offers a certifiable set of standards defined to systematically manage risks posed by information systems. Organizations can also use the **ISO 31000** standard, which provides guidelines for enterprise risk management or **ISO/IEC 27005**, which provides guidelines and techniques to managers in order to implement and manage information security risks. The **ISO/IEC 27005**, more focused on the RM, builds on the knowledge concepts, models, processes, and terminologies of **ISO/IEC 27001**.

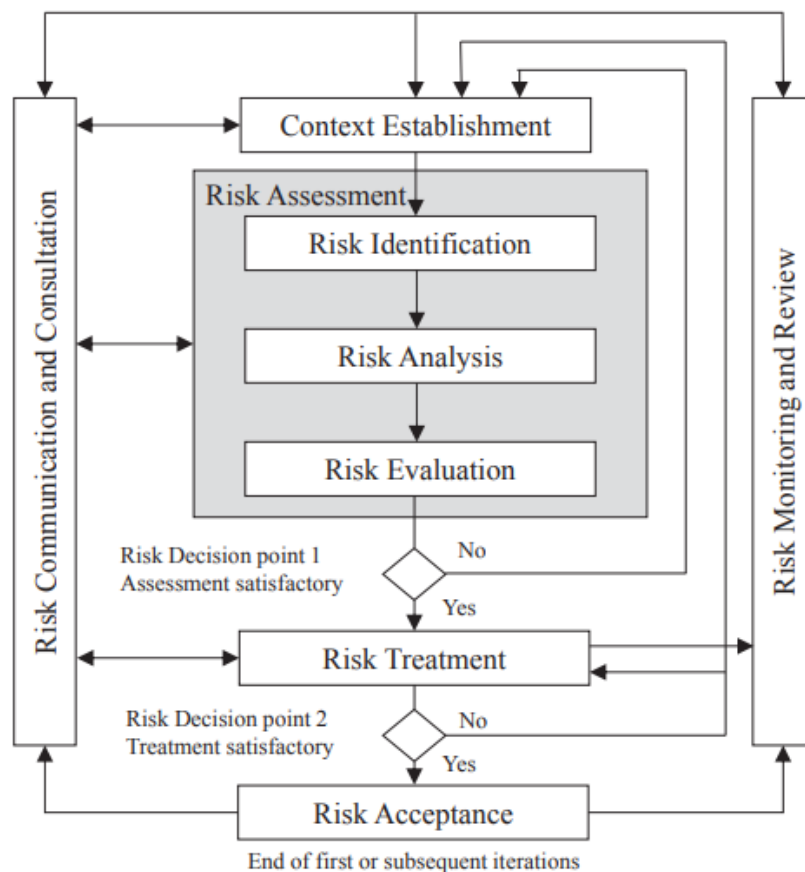


Figure 2: ISO/IEC 27005 model [17]

The first step consists of the context establishment, which includes determining the organization objectives, outlining the scope and boundaries of information security risk

management, specifying the basic criteria, among others. Following comes the risk assessment, consisting of 3 different steps: risk identification, risk analysis, and risk evaluation. On the first one, the assets and their owners are identified. Along with this, occurs the identification of the threats to those identified assets, the existing and planned controls, the exploitable vulnerabilities, and the record of incident scenarios with related impact to those identified assets. The risk analysis takes either a qualitative or quantitative approach to assess the consequences and the likelihood of occurrence of incidents. Finally on the risk evaluation, the risks identified previously are prioritized according to the risk evaluation criteria about the scenarios that lead to those risks. With positive results from the risk assessment, the risk treatment options are selected based on the outcome of risk assessment. The risks are retained when the level of risks fulfills the risk acceptance criteria. The risk communication and consultation step guarantee the exchange of information between the decision-maker and other stakeholders throughout the risk management process. Additionally, risk and their factors, such as, impacts, value of assets, vulnerabilities, threats, and the likelihood of occurrence, need to be monitored and reviewed to identify and corrected at an early stage, corresponding to the risk monitoring and review.

Table 5: ISSO 27005

<b>ISO 27005</b>	
<b>Advantages</b>	<b>Disadvantages</b>
+ Provides the most ROI amongst the common CSF.	- Is based on voluntary shared knowledge and is consensus based. Requires a level of compulsory compliance.
+ Better suited for commercial companies because of its inclusion of rigorous documentation.	- Do not ensure the effectiveness of measures implemented, only their existence
+ Provides standards for cyber risk and disaster recovery	- Huge documentation work

## FAIR Framework

The Factor Analysis of Information Risk (FAIR) framework is defined for the purpose of helping enterprises measure, analyze, and understand information risks. The goal is to guide enterprises through the process of making well-informed decisions when creating cybersecurity best practices.

The FAIR risk model evaluates the factors that make up IT risk and assesses their interaction and impact on each other, breaking each risk into basic building blocks. Taking these elements, it assigns mathematically a dollar value in order to measure the financial risk.

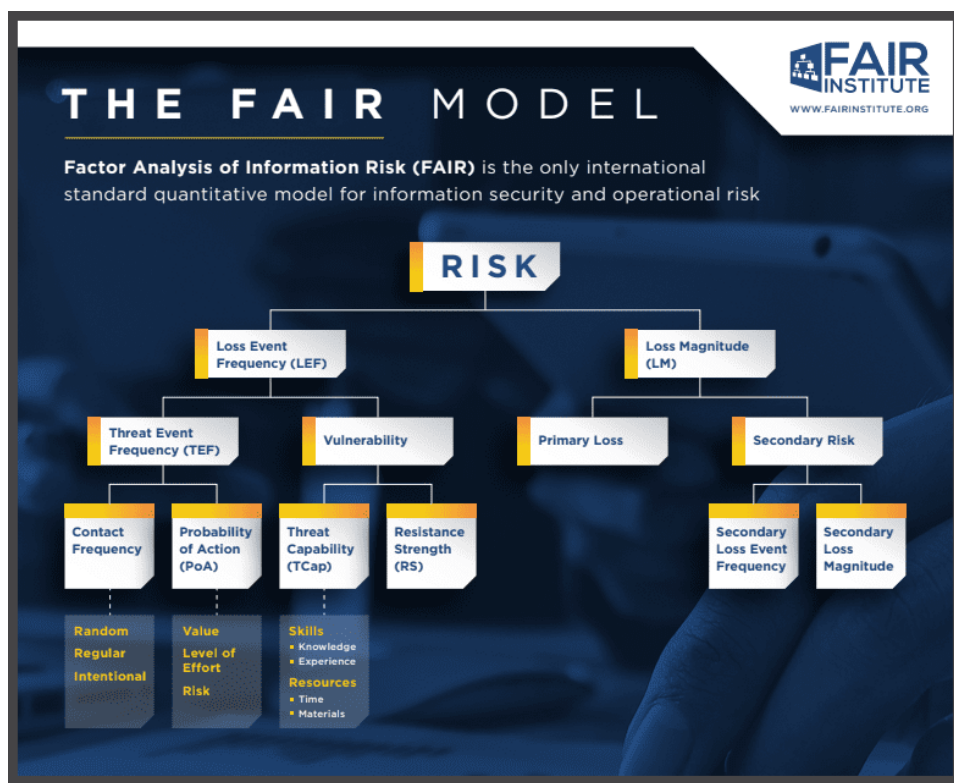


Figure 3: Fair Framework model [29]

The definition of each top block is as follows:

- Risk – The possible frequency and magnitude of future loss.
- Loss Event Frequency – The frequency, within a given timeframe, that the loss is expected to occur.
- Loss Magnitude – The consequent value of the loss from a given risk.



When running an analysis, the FAIR model has three ways to look at it, the first one is the Loss Event Frequency, related to the Threat Event Prevention:

- Threat Event Frequency – The frequency, within a given timeframe, that threat agents are anticipated to act in a way that could result in loss.
- Contact Frequency – The type of controls that could cause less loss events, the success of the control enhancement.
- Probability of Action – The decrease of action by the threat actor given the new control.

The implementation of these blocks would ultimately reduce the threat event frequency and potentially the overall loss event frequency.

The next look goes over vulnerability management:

- Vulnerability – The probability that a threat event will be transformed into a loss event.
- Threat Capability – The level of force capable of being applied by the threat agent.
- Resistance Strength – Used to measure how difficult it is for a threat actor to inflict harm.

It's in this section that it's given a bigger focus on the controls that would reduce the likelihood that a threat event will turn into a loss event.

The final look, the detection and response are overview:

- Loss Magnitude – consequent value of the loss from a given risk.
- Primary Loss – deals with the higher levels of loss created by a threat actor
- Secondary Risk – deals with the lower levels of loss created by a threat actor (example, a data breach, however the data is encrypted, deeming it unusable to the threat actor, creating an internal secondary line of defense)

Everything that could change the potential losses would be modeled in this stage.

The FAIR Risk Assessment Methodology respects the following steps:

1. Identify the scenario components:
  - Identifying the asset at risk
  - Identifying the threat under consideration
2. Evaluate Loss Event Frequency (LEF):
  - Estimate the probable threat Event Frequency (TEF)
  - Estimate the Threat Capability (TCAP)
  - Estimate Control Strength (CS)
  - Derive Vulnerability (Vuln)
  - Derive Loss Event Frequency (LEF)
3. Evaluate Probable Loss Magnitude (PLM):
  - Estimate worst-case loss
  - Estimate Probable Loss Magnitude (PLM)
4. Derive and articulate risk:
  - Derive and articulate risk

Table 6: FAIR Framework

<b>FAIR Framework</b>	
<b>Advantages</b>	<b>Disadvantages</b>
+ Provides the most ROI amongst the common CSF	- Difficult to use, not well documented, lack of access to current information and how methodology is applied
+ Complementary to existing risk frameworks, opportunity for developing a standardization reference architecture	- Depends on computational engine for calculating risk and commercial product RiskLens for analyzing complex risks
+ Promotes a quantitative, risk based, acceptable level of loss exposure	- Requires a tightly defined taxonomy to function

# OCTAVE

**OCTAVE** “Operationally Critical Threat, Asset, and Vulnerability Evaluation” is a risk-based assessment focused on organization’s information security objectives. It was originally developed by the Software Engineering Institute at Carnegie Mellon University, seeking to establish a information-protection decision based on risks, respecting the CIA of critical information technology assets.

This method uses a three-phase approach to examine organizational and technological issues and thus defines a global and comprehensive image of the organization’s information security needs.

The first phase builds an asset-based threat profile, dealing with context analysis in terms of the dependency of business processes of the information system. The remaining phases identify vulnerabilities in progressive series so that security strategies and action plans are developed.

Aside from being a well-structured method and providing detailed audit analysis, it requires a certain expertise from the technical team in order to obtain the upmost effectiveness of the used tools.

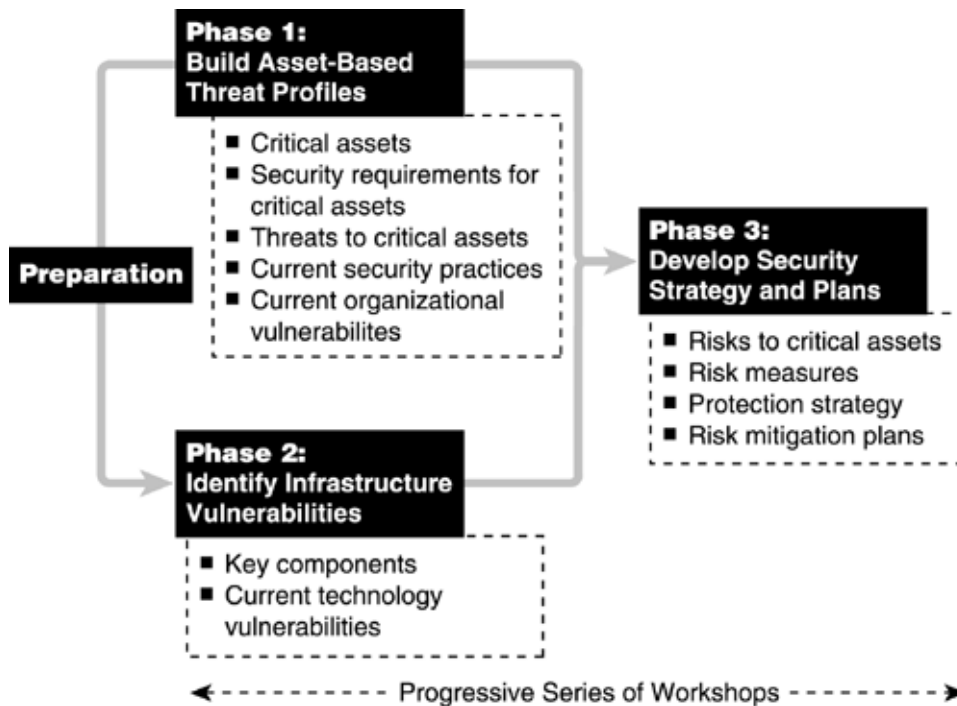


Figure 4: Octave methodology [17]

Table 7: Octave

<b>OCTAVE</b>	
<b>Advantages</b>	<b>Disadvantages</b>
+ Well documented with plenty of tools available	- Does not provide risk monitoring
+ The risk is assessed from an operational point of view	- Although it takes into consideration the mitigation and acceptance of the risk, the avoidance is not considered
+ Can be applied to investigate and categorize recovery impact areas	- No quantification method for calculating the required level of recovery
+ Aimed at companies with limited resources, free and can be used as a foundation for risk-assessment	- Complex and takes time to understand, does not provide a support for financial modelling

## EBIOS

“Expression of needs and identification of security objectives” (EBIOS) is a comprehensive technique dedicated for Information Security risk assessment, using suitable security measures, which allows for assessing the risks or information systems [30]. Originally developed by the French government, nowadays EBIOS is used both in public and in the private sector, in France and abroad, being compliant with major IT security standards.

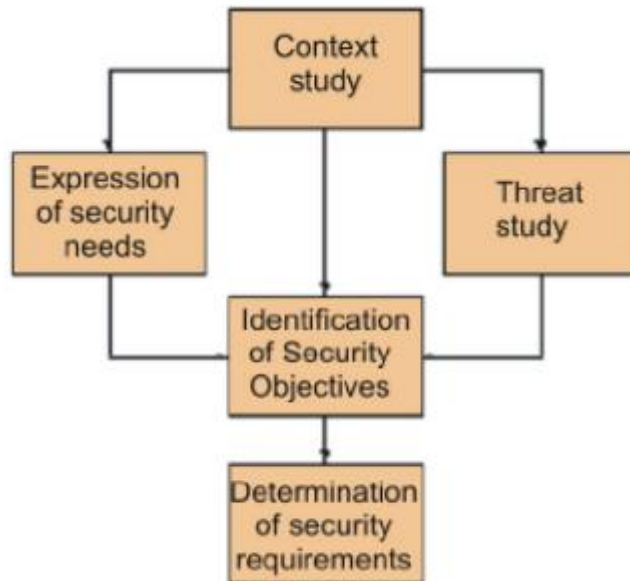


Figure 5: Bowtie methodology [17]

It primarily consists of 5 phases, the first one deals with context analysis in terms of the stage in which business processes depend on the information system. The second phase identifies and estimates the security needs and eventual sources of threats. Followed by the third phase, in which the risk assessment concerning the identification and the estimation of the scenarios that can cause a potential threat is carried out. The studied system has the risks highlighted by confronting the feared events to threat scenarios on the fourth stage. And finally, the security measures to be implemented are specified and evaluated [31] [32] [33].

Table 8: EBIOS

<b>EBIOS</b>	
<b>Advantages</b>	<b>Disadvantages</b>
<ul style="list-style-type: none"> <li>+ Easily adaptable to the context of each organization, allowing them to be adjusted to their methodological tools and habits</li> </ul>	<ul style="list-style-type: none"> <li>- Does not provide immediate solutions or even recommendations to security problems</li> </ul>
<ul style="list-style-type: none"> <li>+ Gradually improves the content by reusing each module</li> </ul>	<ul style="list-style-type: none"> <li>- Does not provide risk monitoring</li> </ul>
	<ul style="list-style-type: none"> <li>- Generates a large amount of information</li> </ul>

## Bowtie

Bowtie approach is a qualitative method incorporating management system techniques. The theory in this approach can be found in the “Swiss cheese model”. The Royal Dutch/Shell Group was the first major company to integrate the full methodology into their business practices, being credited with developing the technique which is widely used in the present. The bowtie became popular as a structured method in order to assess risk where a quantitative approach is not desirable or possible. Essentially, this method is applied to hazard identification, operational system and procedures to eliminate the hazard or reduce its frequency of occurrence [34].



Figure 6: Bowtie method [21]

The Bowtie process requires a continuous identification of hazards and effects, assessment of the risks associated and the specification of the control and recovery measures, which must be kept and maintained in place. It follows the steps:

- Identify the bowtie hazard – There are two items for this purpose, the hazard itself, and the event that will occur. The hazard has the potential to cause harm, in different forms and levels, while the event is the undesired event at the end of the fault tree and at the beginning of an event tree. Example: The release of a toxic substance (hazard) is caused by a structural failure (event).
- Assess the threats – The threats are located at the far-left side of the diagram and are related to something that will potentially cause the release of the identified hazard (in the example before, it may be a fire or explosion).

- Assess the consequences – The consequences are located at the far-right side of the diagram and are related to the threats that may lead to the top event (in the example before, it may be environmental pollution).
- Control – Control is related to the protective measure that is put in place in order to prevent the threats from releasing a hazard. In the diagram they belong between the threat and the hazard. The controls can be to prevent threats, prevent consequences or threats to the controls itself, in order to reduce the risk to a level low enough to be considered acceptable.
- Recover – These types of control are located between the hazard and the consequences, being technical, operational, and organizational measures needed to limit the chain of consequences arising from an event. Recovery controls are usually systems to detect and decrease incident occurrences.
- Identify threats to the controls – Although the control is meant to protect and prevent threats, there is the possibility that these same controls are susceptible to threats that may exploit or override them and thus increase the risk. On the diagram they are located under and off to the side of the control.
- Identify the controls for the threats to the controls – After knowing which threats may affect the controls in place, new controls to prevent these threats need to be put in place, ensuring the functionality of the first controls implemented.

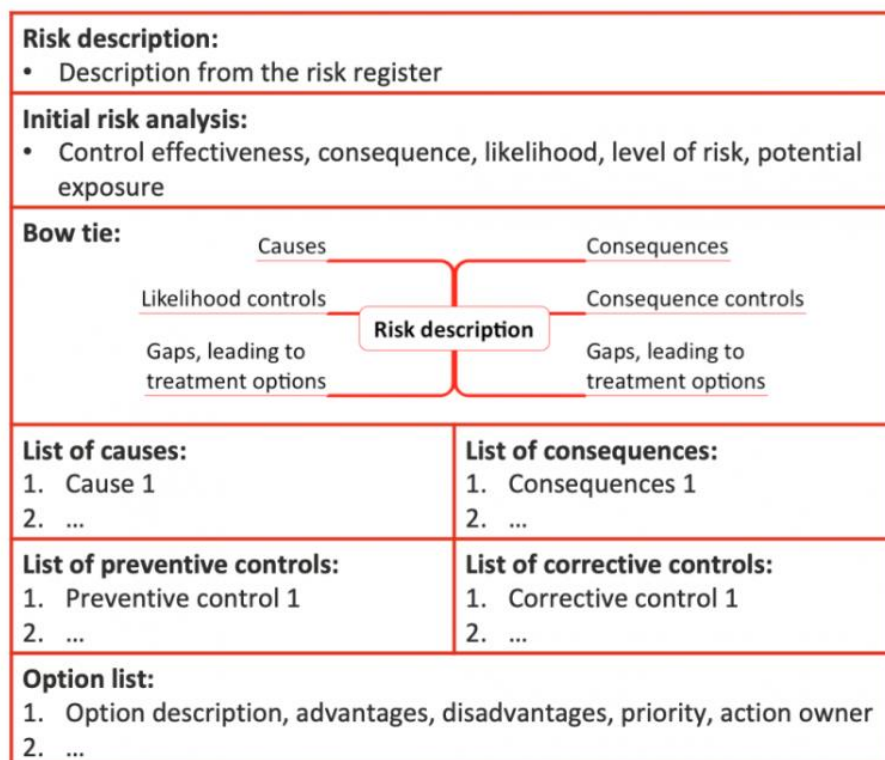


Figure 7: Bowtie methodology [35]



Table 9: Bowtie

<b>Bowtie</b>	
<b>Advantages</b>	<b>Disadvantages</b>
<p>+ Simple to read and understand</p>	<p>- Does not provide quantitative assessment of acceptability of risks unless linked to event tree analysis or fault tree analysis</p>
<p>+ On the left side of the diagram the full range of initiating events and the way they combine and escalate are clearly shown</p>	<p>- No standards exist therefore there are a range of different and subtle representations of bowtie diagrams</p>
<p>+ On the right side of the diagram the different possible consequences and outcomes are well defined</p>	

The following table (Table 10) represents a comparison between all the methods seen so far in terms of their type of analysis, monetary requirements, their main focus has a method, the need for an additional software and the difficulty of use/implementation. All the methods were found to be qualitative so there is nothing to compare on the first column. The second and third column unfortunately already exclude two methods, OCTAVE and Fair, since a monetary amount is needed to purchase the license for their software. Some of the other methods don't have information about the cost because they are most guides and standards and not exactly a product that can be installed and used as well.

For the main focus of the methods, the objective is to use a method that is better qualified or better targeting the RM area. On this regard, there is the OCTAVE, NIST SP800 and ISO 27005. The software support is associated with the software cost, in which those methods that have a price tag or are free to use, will require the specified software support.

Lastly the difficulty of use and implementation is distributed equally through the table, having some methods an easier learning curve, like OCTAVE, NIST SP800 and Bowtie, while the others require a longer time or knowledge to be applied. As a result of this comparison and taken into account the monetary part as well as the focus of this project, two options stand out, NIST P800 and ISOO 27005. After a careful consideration of the industry standards, it was decided that this thesis will rely on the framework ISO 27005, which provides a broad range of guidelines and an industry standard recognized anywhere.

Table 10: Risk Management Methods, RA – Risk Assessment, RM – Risk Management RQ – Risk Quantification

<b>Methods</b>	<b>Type of analysis</b>	<b>Support cost</b>	<b>Software cost</b>	<b>Focus</b>	<b>Software support</b>	<b>Ease of Use</b>
<b>EBIOS</b>	Qualitative	Free	Free	RA	EBIOS tool	-
<b>OCTAVE</b>	Qualitative	Free	1300 €	RA/RM	Resolver Ballot	+
<b>NIST SP800</b>	Qualitative	Free	N/A	RM	N/A	+
<b>ISO 27005</b>	Qualitative	Free	N/A	RA/RM	N/A	-
<b>Fair</b>	Qualitative	Free	€€	RQ/RA	Risk Lens	-
<b>Bowtie</b>	Qualitative	Free	N/A	RA	N/A	+

## 2.8. TECHNOLOGIES

For the development of the application, a first thought was given to a standard desktop application using python, however, to integrate certain features and allow the users to have a better control and management that the tool can provide, a web application was chosen instead. Another critical point is based on the proposition that this risk management application can be used in a critical infrastructure or large company, in which installing an application in every device can be problematic and time consuming, as some applications require but to whom this thesis will not further elaborate about, thus relying on a local web application where the management can be done anywhere on the network as long as given the necessary credentials.

### 2.8.1. PYTHON

Created in 1989 by Guido Van, Python has become one of the most popular high-level general programming languages [36]. It is an interpreted dynamically typed language; hence, it does not need any separate compiling time, it is instead compiled to byte code and executed instantaneously. One of the main reasons why it became so popular is the simplicity and code readability. Because of its object-oriented architecture, it is not only the language for just scripting anymore, as it can be used for nearly every situation, although it may have some drawbacks when compared to other languages in certain conditions. Some of the big areas of Python are web and big data and machine learning. Thus, for this project it will be used in the web development of the application [37].

## 2.8.2. FLASK

Designed to create a web application in a short amount of time, Flask can also be called a micro-framework, Web Server Gateway Interface (WSGI) application framework. It's flexible enough to be used either for pure backend or frontend if required. In terms of frontend, it provides full request object, routing system for endpoints, cache controls etc.



Figure 8: Flask [38]

### **The advantages of Flask are as follows:**

- Higher compatibility with latest technologies
- Easier to use for simple cases
- High scalability for simple applications
- Easy routing URL
- Easy Database integration
- Small core
- Minimal yet powerful platform

### **Disadvantages of Flask:**

- Higher maintenance cost for more complex systems
- Async may be problematic
- Lack of Object-relational mapping (ORM)
- Setting up a large project requires some previous knowledge of the framework

### 2.8.3. DJANGO

Introduced earlier than Flask, in 2005 by Adrian Holovaty and Simon Willison, Django had the goal of developing applications in a fast way using Python. As such, it offers a standard method for fast and effective website development. Enabling the process to be smooth and timesaving while having a full kit of tools and a large documentation to support the developer.



Figure 9: Django [39]

**The advantages of Django are as follows:**

- Easy to set up and run
- Allows end-to-end application testing
- Offers built-in authentication system
- A complete stack of tools
- Data modelled with Python classes
- Provides an easy-to-use interface for various administrative activities
- Offers bigger support and has a bigger community compared to Django

**Disadvantages of Django:**

- High dependence on Django ORM. Broad knowledge required
- Fewer design decisions and components
- Larger code size
- High learning curve
- Allows only to handle a single request per time
- A higher entry point for simple solutions

## 2.8.4. COMPARATION BETWEEN FLASK AND DJANGO

Being Flask younger, it was created for rapid development, allowing the user to require a smaller learning curve, when compared to Django, with API support, the use of multiple types of databases

Table 11: Flask vs Django

Flask	Django
Created in 2010	Created in 2005
WSGI framework	Full Stack Web Framework
Provides support for API	Does not have any support for API
Allows the use of multiple types of databases	Doesn't offer multiple types of databases
Does not offer dynamic HTML pages	Offers dynamic HTML pages
Flask template engine Jinja2 is based on Django's template engine	It comes with built-in template engine that saves a lot of development time
Flask is more open-ended, and developers may or may not follow the best practices	Django framework ensures developers use best practices as everything has a template
Flask doesn't not have a feature to handle administration tasks	Django comes with a ready-to-use admin framework

In conclusion, after a comparison between the two frameworks, Flask provides a faster and easier development without a preexisting extensive knowledge and a flexible kit for the front and backend of the application, along with some previous knowledge in Flask development, this framework will be the one used to develop the risk management application.

## 2.9. VULNERABILITY MANAGER

The most common and used cybersecurity tools, such as firewalls and antivirus software are reactive tools, designed to manage attacks as they occur. On the other hand, vulnerability management software proactively looks for weaknesses by scanning, identifying vulnerabilities in the network and if possible, providing remediation suggestions to mitigate the potential threat. In addition, some vulnerability management tools can assign threat levels, based risk score, helping to prioritize the most significant issues.

Using vulnerability scanners, the whole network or specific hosts can be scanned for security holes, missing patches or other type of vulnerabilities. Some of existing vulnerability managers are Nessus and OpenVAS, with different scanning techniques, offering a graphical user interface and allowing for different sectors of the company to access and understand the risk they may be involved in.

### 2.9.1. NMAP

Nmap, or Network Mapper, is a free and open source and powerful tool used for vulnerability checking, port scanning, service discovery, operating system detection and network mapping. Created in 1997, it remains one of the most used tools in cybersecurity. The heart of this tool is port scanning, in which the users designate a list of targets on a network without giving specific details.

The depth of each scan is also controllable, with different performance options, types of scans and techniques to allow for the maximum accuracy possible.

The techniques used for the application will be compared with Nmap for a performance and accuracy evaluation.



Figure 10: Nmap [40]

## 2.10. NETWORK SIMULATION

As the thesis relies on a risk management application, to be able to test the scanning mechanisms a network setup is required. However, as it is not possible to set up a physical network with a few hosts with different operating systems and vulnerabilities, a virtual network is needed instead. VirtualBox will be the main software to simulate machines with different vulnerabilities on a network, while GNS3 will be used to visually simulate the network that was set up for the tests.

### 2.10.1. VIRTUALBOX

VirtualBox is an open-source software for virtualizing. It acts as a hypervisor, creating a VM (Virtual Machine) where the user can run another OS (Operating System).

The operating system in which VirtualBox runs is called the “host” OS, while the one running in the VM is called the “guest” OS. In the configuration the user can specify how many CPU cores, how much RAM and disk space should be devoted to the VM.



Figure 11: VirtualBox [41]

One of the central ideas behind hardware virtualization is the possibility to use virtual machines in almost all cases where physical computers can be used. For this, VM's must be able to connect to physical and virtual networks with their own virtual network adapter. VirtualBox provides multiple network modes to be able to connect to different networks or allow for different configurations.

Each VM can use up to eight virtual network adapters, or also referred to eight network interface controllers (NIC). Each adaptor is attached to one network mode, from which the user can choose. This makes possible for each network adapter to have a separate configuration and operate in different modes.



Table 12: VirtualBox modes [29]

	VM <-> VM	VM -> Host	VM <- Host	VM -> LAN	VM <- LAN
Not attached	-	-	-	-	-
NAT	-	+	Port Forward	+	Port Forward
NAT Network	+	+	Port Forward	+	Port Forward
Bridged	+	+	+	+	+
Internal Network	+	-	-	-	-
Host-only	+	+	+	-	-

The comparison of VirtualBox network modes shows how each mode can communicate with other virtual machines, the host machine and the local area network (LAN).

As a network simulation is needed, the only two modes that will be covered are the NAT Network and the Host-only.

### NAT Network

Like the NAT mode used in a router configuration, this mode allows for multiple virtual machines to communicate between each other via the network. The virtual machines can also access other hosts in the physical network and external networks including the internet. However, any machine from an external network as well as those from the physical network to which the host machine is connected, are not allowed to access the virtual machine. As well as the host cannot access the guest machine, unless port forwarding is used.

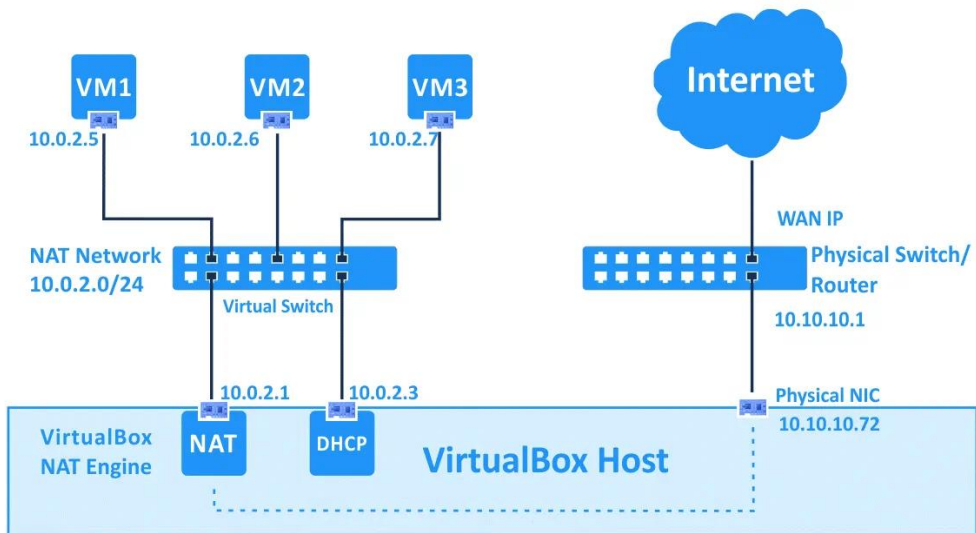


Figure 12: NAT Network Mode [42]

### Host-only Adapter

The primary focus of this mode is for the communication between a host and the guests, while a guest can only communicate with another one if connected also to the host-only network.

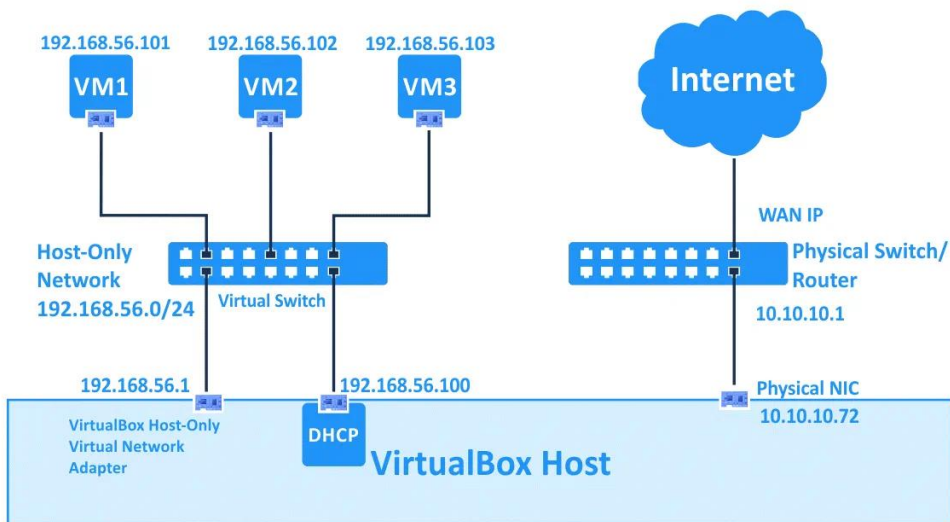


Figure 13: Host-Only Network Mode [42]

## 2.10.2. GNS3

GNS3 is a popular open-source emulator used to configure, test and troubleshoot virtual and real networks, allowing to run a small topology consisting of a few devices or one that have many devices hosted on multiple servers or even hosted in the cloud. GNS3 supports multiple switching options, multiple vendor environments, supports VirtualBox and has a large and active community [43].

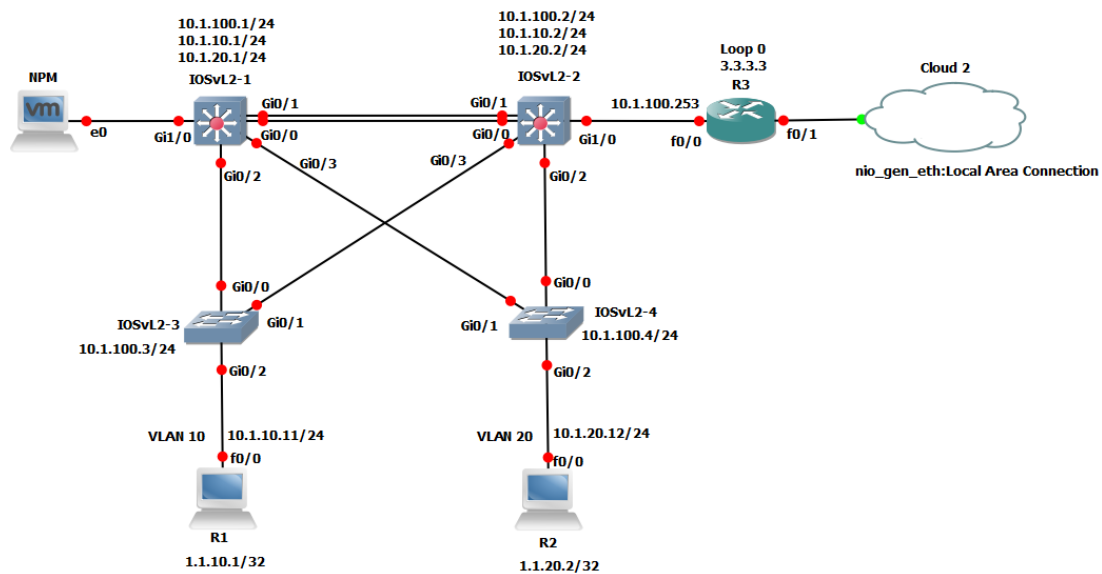


Figure 14: GNS3 network topology [44]

The client part and his graphical user interface (GUI), allow for a visual representation as seen on the Figure 14 where a network comprised of a router, switchers, cloud to connect to the local area network and a few hosts, are illustrated.



# 3. VULNERABILITY MANAGEMENT

## 3.1. INTRODUCTION

In today's times, many critical and high-level vulnerabilities are discovered every day in various products. In order to mitigate these vulnerabilities, companies and corporations are required to create procedures that can identify, analyse and mitigate those vulnerabilities in real time.

As mentioned above, a vulnerability is a weakness present in a device or system that, if left alone, will be exposed to a possibility of being exploited or attacked. A vulnerability scanning is the process that investigates, identifies and detects such vulnerabilities, using either manual or automated tools and techniques.

A vulnerability scan is not to be confused with a penetration test, where the former is a preliminary step to identify the hosts, subsequently the ports and services used for later to be used in a penetration test. All the information retrieved from a vulnerability scan, such as, the open ports in a server or the type of vulnerability that a server is susceptible to, will be the inputs for a penetration tester to use and perform the necessary actions to try to access that server and show the degree of compromise.

The usual flow of procedures can be divided into three phases, the host discover, port scanning and service and version detection.

## **Host Discovery**

As the name suggests, the first phase, host discovery, represents the initial moment where the mechanism or tool used will try to detect and distinguish in a specific network all the hosts that are up. This procedure is necessary to reduce the time of a network scanning, since attempting to scan the ports of all the hosts regarding their status, will add unnecessary time to the vulnerability scan.

## **Port Scanner**

Following the list of live hosts, a detection of the open ports for each live host or a specific one is conducted. For the detection of an open port in a particular host, a communication between that host on that port and the machine performing the scan must be established or partially established.

This procedure differs from protocol to protocol, as an example, TCP and UDP use distinct patterns and communications to conclude if a port is open or not.

## **Service and Version Detection**

Detecting services and the software version on a target device is critical to determine potential security holes or vulnerabilities belonging to outdated or specific software versions. Each commercial service or product can have a huge number of versions and its highly likely that some of them will have overseen vulnerabilities. The important is to detect them by running routine checks with known or the latest exploits found.

## **SSL/TLS Detection**

SSL (Secure Socket Layer) and TLS (Transport Layer Security) are cryptographic protocols used to secure web communications against any unauthorized tampering. They are used to establish a secure connection between clients and servers across the internet and ensure that the information relayed is encrypted and kept its integrity. SSL was the antecedent of TLS, with TLS currently being in its third iteration, TLS 1.3. Despite this, SSL continues to be used most of the time as well as TLS. Secure connections will indicate their secure status by presenting HTTPS (Hypertext Transfer Protocol Secure) in the URL bar of any web browser, contrary to the simple HTTP (Hypertext Transfer Protocol).

## 3.2. THE USE OF NMAP

Although the use of personal made scripts to do a vulnerability scan of a network was possible, the comparison between a personal made script and Nmap in terms of performance, speed, options available and type of output is largely visible. Specially in more demanding parts like service and version detection of a software. With this conclusion, the scripts used were changed to a whole adaptation of Nmap in this project.

Given the robustness of Nmap development, a scan can go from a very simple one with basic techniques, to an advance scan containing complex and hard to detect techniques. Follows an explanation of the different scan types given by Nmap:

### **TCP Scan**

This scan is generally used to check and complete a three-way handshake between the machine performing the scan and the chosen target(s) system(s). TCP stands for Transmission Control Protocol, which is a communication process largely used between devices over the internet, providing a reliable communication with both sides being able to acknowledge the reception of the data and making sure that even if the data was damaged during transmission, it will inform of it and a retry is possible to happen so that all the data is sent and received without any kind of loss. The three-way handshake, as the name suggests, is a mechanism that three segments are exchanged between both parties to ensure a reliable TCP connection to get established. First the client sends a segment with SYN (Synchronize Sequence Number) to inform the server that a certain client is trying to start a communication, and which will be the sequence number it should start the segments with, ensuring an established connection. Second, the server responds to that same client with a SYN-ACK segment, acknowledging (ACK) that the initial segment of the client was received with success and which sequence number (SYN) it will start the segments with, in other terms, which ports both client and server will use to communicate during this session. Finally, the client receives the SYN-ACK from the server and sends back the last acknowledgement (ACK) that it received with success the previous server segment. After this, the normal communication can be established, and the data can be transmitted and received between the two parties successfully [45].

## **SYN Scan**

SYN scanning, also known as half-open scanning, is often used by malicious actors to determine the state of a communication port without the need to establish a full connection.

The idea behind it is like a normal port scanning, which the user attempts to set up a TCP/IP (Transmission Control Protocol/Internet Protocol) connection with a specific server at every existing port by sending a SYN packet, almost like initiating a three-way handshake. However, if the server replies with an ACK response or even a SYN/ACK one from any port, meaning the port is open, the user sends an RST (reset) packet instead of the common ACK packet. This will cause the server to assume there was a communication error and a connection was not established with the client [46].

## **ACK Scan**

A TCP ACK scan technique uses packets with the before mentioned ACK, is different than the previous scanning techniques, as it does not determine open ports but rather is used to map out firewall rulesets, in order to check if the host is protected by some kind of filtering system. The user sends an ACK probe packet with a random sequence number, if there is no response it means the port is filtered, on the other hand if an RST response is received, the port is closed [47].

## **ICMP Ping**

ICMP (Internet Control Message Protocol) ping also known as ICMP sweep is a basic network scanning technique used to determine which hosts are live from a given range of IP addresses. Contrary to a single ping that tells whether one host exists on the network, a ping sweep using ICMP echo requests sent to different hosts. If an address is live, it will return an ICMP echo reply, telling the user that the host is live [45].

The normal use of Nmap would require the typing of the IP addresses and the chosen options on the command line in order to perform the scan, but since this project used a web application, the main python script initiates and runs by itself the Nmap without any need for manual typing.



### 3.3. HOST DISCOVERY

Given a network range or a certain number of IPs, obtained from the web application interface, the host discovery phase starts. The target is first sent to the function `sendTcpSyn()` (Listing 1 and Listing 2) along with the current working directory of a process. This way, the xml file to be outputted, will be saved on the same location of this script. The name of the output file is stated, followed by the Nmap command which contain the Nmap location, the chosen target, and the following options:

- **-PS21, 22, 23, 25, 80, 113, 443**: TCP SYN Ping to the following port list
- **-PA80, 113, 443**: TCP ACK Ping to the following port list
- **-n**: Do not do DNS resolution
- **-sn**: ICMP echo request
- **-T4**: speed template which tells Nmap how quickly to perform the scan.
- **-vv**: with verbose enabled the open ports are printed in interactive mode as they are discovered.
- **-oX**: output the results into xml format

```
target = '10.0.2.0/24'
print("Beginning tcp syn...\n")
sendTcpSyn(target, os.getcwd())

print("Beginning xml parse...\n")
hosts = parseDiscoverXml('tcp_syn_host_discovery.xml')

print("XML host discovery done.\n")
```

Listing 1: Process of Nmap scan in code

```
def sendTcpSyn(target, out_xml):
    out_xml = os.path.join(out_xml, 'tcp_syn_host_discovery.xml')
    nmap_cmd = f"/usr/bin/nmap {target} -PS21,22,23,25,80,113,443 -PA80,113,443 -n -sn -T4 -vv -oX {out_xml}"
    sub_args = shlex.split(nmap_cmd)
    subprocess.Popen(sub_args, stdout=subprocess.PIPE, stderr=subprocess.PIPE).communicate()
    makeInvokerOwner(out_xml)
```

Listing 2: Nmap host discovery scan

Next, the commands are split as arguments to be used by the function `subprocess.Popen()`, taking in the previous Nmap arguments and launches a “child” process. This allows to make a call to Nmap as well as manage the output effectively.

The output xml file is illustrated on Listing 3 (representing only the first 9 hosts), as it's visible, the information given has to do with the Nmap script, the location of the file and the most important part, the hosts IPv4, their status and the response given.

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nmaprun>
<?xml-stylesheet href="file:///usr/bin/./share/nmap/nmap.xsl" type="text/xsl"?>
<!-- Nmap 7.92 scan initiated Wed Sep 21 10:07:55 2022 as: /usr/bin/nmap -PS21,22,23,25,80,113,443 -
    PA80,113,443 -n -sn -T4 -vv -oX /home/kali/Desktop/tcp_syn_host_discovery.xml 10.0.2.0/24 -->
<nmaprun scanner="nmap" args="/usr/bin/nmap -PS21,22,23,25,80,113,443 -PA80,113,443 -n -sn -T4 -vv -oX /home/kali/Desktop/tcp_syn_host_discovery.xml 10.0.2.0/24"
    start="1663769275" startstr="Wed Sep 21 10:07:55 2022" version="7.92" xmloutputversion="1.05">
<verbose level="2"/>
<debugging level="0"/>
<taskbegin task="ARP Ping Scan" time="1663769275"/>
<taskend task="ARP Ping Scan" time="1663769277" extrainfo="255 total hosts"/>
<host-><status state="down" reason="no-response" reason_ttl="0"/>
<address addr="10.0.2.0" addrtype="ipv4"/>
</host>
<host-><status state="up" reason="arp-response" reason_ttl="0"/>
<address addr="10.0.2.1" addrtype="ipv4"/>
<address addr="52:54:00:12:35:00" addrtype="mac" vendor="QEMU virtual NIC"/>
<hostnames>
</hostnames>
<times srtt="264" rttvar="5000" to="100000"/>
</host>
<host-><status state="up" reason="arp-response" reason_ttl="0"/>
<address addr="10.0.2.2" addrtype="ipv4"/>
<address addr="52:54:00:12:35:00" addrtype="mac" vendor="QEMU virtual NIC"/>
<hostnames>
</hostnames>
<times srtt="217" rttvar="5000" to="100000"/>
</host>
<host-><status state="up" reason="arp-response" reason_ttl="0"/>
<address addr="10.0.2.3" addrtype="ipv4"/>
<address addr="08:00:27:37:D3:B8" addrtype="mac" vendor="Oracle VirtualBox virtual NIC"/>
<hostnames>
</hostnames>
<times srtt="177" rttvar="5000" to="100000"/>
</host>
<host-><status state="up" reason="arp-response" reason_ttl="0"/>
<address addr="10.0.2.4" addrtype="ipv4"/>
<address addr="08:00:27:6D:3A:D6" addrtype="mac" vendor="Oracle VirtualBox virtual NIC"/>
<hostnames>
</hostnames>
<times srtt="262" rttvar="5000" to="100000"/>
</host>
<host-><status state="up" reason="arp-response" reason_ttl="0"/>
<address addr="10.0.2.5" addrtype="ipv4"/>
<address addr="08:00:27:22:FF:4A" addrtype="mac" vendor="Oracle VirtualBox virtual NIC"/>
<hostnames>
</hostnames>
<times srtt="270" rttvar="1244" to="100000"/>
</host>
<host-><status state="down" reason="no-response" reason_ttl="0"/>
<address addr="10.0.2.6" addrtype="ipv4"/>
</host>
<host-><status state="down" reason="no-response" reason_ttl="0"/>
<address addr="10.0.2.8" addrtype="ipv4"/>

```

Listing 3: Host found xml file

From this file, it's extracted the information mentioned before using the python module *xml.etree.ElementTree*, which is a simple and effective API for parsing and creating XML data. At the top, the file is parsed and assigned to the variable *xml\_tree\_echo*, followed by the function *getroot()*, to return the root element for this tree. Having the tree initialized, now the xml structure can be accessed and using an iteration over the subelements (also called "children") in the root, it's possible to extract the attributes and consequently the value inside them.

In the case of any host being up, they are added to the database table *HostDiscovery* by the function *insert\_content()*. At this step the information is also appended to the list *get\_host\_number[]*, to be later used in the SSL/TLS check and Service scan.

## 3.4. PORT SCANNER

With the hosts discovered by previous scan, the variable *targets* now takes the different hosts found using the function *convertToNmapTarget()* (Listing 4) and which takes the hosts and creates a list of them all. This step is important for a smooth process in the following scans. The port scan finally begins with the use of the function *tcpSynPortScan()* (Listing 5)

This function will scan the top 1000 most common ports. The name of the output file is stated, followed by the Nmap command which contain the Nmap location, the chosen target, and the following options:

- **--top-ports 1000**: scans the top x ports specified
- **-n**: Do not do DNS resolution
- **-Pn**: This option will skip the host discovery stage
- **-sS**: TCP SYN Scan
- **--min-parallelism 100**: Specify the minimum number of parallel port scans
- **--min-rate 64**: Specify the sending rate of packets per second
- **-sV**: Detects the version of the service used
- **--version-intensity 6**: Intensity level representing the probes used in version detection.
- **--script banner**: Collect details of the target regarding services running on its open ports.
- (-T4, -vv and -oX were stated before)

```
def convertToNmapTarget(hosts):
    hosts = list(dict.fromkeys(hosts))
    return " ".join(hosts)
```

Listing 4: Convert to Nmap format

```
def tcpSynPortScan(target, out_xml):
    print("im here")
    out_xml = os.path.join(out_xml, 'top_1000_portscan.xml')
    nmap_cmd = f"/usr/bin/nmap {target} --top-ports 1000 -n -Pn -sS -T4 --min-parallelism 100 --min-rate 64 -sV --version-intensity 6 --script banner -T4 -vv -oX {out_xml}"
    sub_args = shlex.split(nmap_cmd)
    subprocess.Popen(sub_args, stdout=subprocess.PIPE, stderr=subprocess.PIPE).communicate()
    makeInvokerOwner(out_xml)
```

Listing 5: Nmap port scan

```
-host starttime="1663769277" endtime="1663769312"-->status state="up" reason="arp-response" reason_ttl="0"/>
<address addr="88:00:27:60:3A:D6" addrtype="mac" vendor="Oracle VirtualBox virtual NIC"/>
</hostnames>
</hostnames>
<ports>
<extrareasons reason="reset" count="991" proto="tcp" ports="1,3,4,6,7,9,10,17,18,21,23,26,30,32,33,37,42,43,49,53,70,79,81,85,88,99,99,100,100,109,111,113,110,125,135,144,146,161,163,179,199,211,212,222,254,256,259,264,289,301,366,311,340,360,386,407,416,417,455,477,444,450,464,465,461,497,589,512,515,524,541,543,545,548,554,555,563,587,593,616,617,625,631,636,646,648,666,669,683,687,691,700,705,711,714,720,722,726,749,765,777,783,787,809,801,889,643,673,889,889,889,890,891,912,913,912,981,987,990,992,993,995,999,1002,1007,1009,1011,1021,1100,1102,1104,1108,1110,1114,1117,1119,1121,1126,1130,1132,1137,1139,1141,1145,1147,1149,1151,1152,1154,1163,1166,1168,1174,1175,1183,1185,1187,1192,1198,1199,1201,1213,1216,1218,1223,1234,1236,1244,1247,1248,1259,1271,1272,1277,1287,1289,1298,1301,1309,1311,1322,1326,1334,1352,1417,1433,1434,1443,1453,1461,1494,1508,1501,1503,1521,1524,1533,1556,1589,1593,1594,1600,1641,1658,1666,1687,1688,1700,1713,1721,1724,1755,1761,1762,1763,1801,1805,1812,1839,1840,1892,1894,1895,1896,1914,1935,1947,1971,1972,1974,1984,1998,2010,2013,2024,2022,2039,2033,2045,2048,2049,2065,2085,2099,2100,2103,2105,2107,2111,2110,2121,2126,2135,2144,2160,181,2179,2179,2191,2191,2196,2200,2222,2221,2269,2288,2391,2323,2366,2381,2383,2393,2394,2399,2401,2429,2500,2522,2525,2557,2601,2602,2604,2605,2607,2608,2638,2701,2702,2710,2717,2718,2725,2800,2809,2811,2869,2875,2909,2910,2920,2967,2968,999,3000,901,3003,3005,3007,3011,3013,3017,3039,3031,3052,3071,3077,3128,3160,3211,3221,3268,3261,3269,3269,3283,3300,3301,3306,3322,3325,3333,3351,3367,3369,3372,3389,3399,3404,3476,3483,3517,3527,3546,3551,3568,3659,3699,3703,3717,766,3784,3809,3801,3899,3914,3926,3929,3951,3989,3971,3978,3989,3989,3995,3914,3918,3920,3945,3971,3989,3995,3999,4000,4006,4045,4111,4125,4126,4129,4224,4242,4279,4321,4343,4443,4446,4449,4550,4567,4662,4648,4899,4900,4999,5000,5002,5004,899,5039,5033,5059,5061,5084,5060,5081,5088,5087,5100,5102,5120,5190,5208,5214,5221,5222,5225,5226,5269,5288,5296,5357,5405,5414,5431,5432,5440,5508,5510,5544,5550,5555,5560,5566,5631,5633,5666,5678,5679,5718,5739,5800,5802,5810,5811,5815,822,2523,2529,2539,2582,2677,3100,3904,5006,5907,5910,5911,5915,5922,5923,5930,5952,5959,5963,5967,5989,5998,6007,6009,6025,6059,6100,6111,6106,6112,6113,6123,6129,6156,6240,6309,6502,6510,6543,6247,6535,6567,6300,6646,6566,6569,6589,6623,6699,770,6788,6799,6792,6839,6881,6901,6969,7000,7002,7004,7007,7019,7025,7078,7100,7103,7106,7200,7201,7402,7435,7443,7496,7512,7625,7627,7676,7741,7777,7778,7808,7911,7928,7921,7937,7938,7999,8002,8007,8011,8021,8022,8031,8042,8045,8082,8090,893,8099,8100,8108,8101,8152,8194,8208,8222,8254,8298,8292,8308,8333,8383,8400,8402,8443,8588,8600,8640,8651,8652,8654,8781,8800,8873,8888,8899,8994,9000,9003,9009,9011,9040,9050,9071,9080,9081,9090,9091,9099,9103,9110,9111,9200,9207,9220,200,9415,9418,9445,9500,9502,9503,9525,9575,9599,9595,9610,9666,9676,9678,9690,9900,9917,9928,9943,9944,9968,9990,9990,9991,9992,9943,9944,9968,9990,9990,10009,10010,10012,10024,10025,10002,10100,10215,10242,10566,10016,10017,10021,10026,10028,10029,10070,11110,11111,11967,12000,12174,12265,12345,13456,13722,13782,13783,14000,14238,14441,14442,15000,15002,15004,15660,15742,16000,16001,16012,16016,16018,16080,16113,16992,16993,17877,17888,18040,18101,18088,19101,19283,19315,19350,19780,19801,19842,20000,20005,30003,20021,20022,20028,21571,22039,23502,24444,24800,25734,25735,26214,27000,27352,27353,27355,27356,27715,28281,30000,30718,30951,31038,31337,32768,32785,33354,33899,34571,34573,35500,38292,40193,40011,41511,42518,44176,44442,44443,44501,45100,48880,42152,49311,49163,49165,49167,49175,49176,49400,49999,50003,50006,50300,50309,50500,50636,30000,51103,51493,52073,52022,52048,52069,54045,54320,52655,55056,55355,55600,56727,56730,57204,57797,58000,60620,60443,61532,61900,62070,6331,64623,64680,65000,65129,65389"/>
</extrareasons>
<script protocol="tcp" portid="22">state state="open" reason="syn-ack" reason_ttl="64"/><service name="ssh" product="OpenSSH" version="5.3p1 Debian 3ubuntu4" extrainfo="(Ubuntu Linux; protocol 2.0)" ostype="Linux" method="probed" conf="10">
<cpes/cpe:/a:openssh:openssh:5.3p1/><cpes/cpe:/o:linux:linux_kernel/><cpes/cpe:/service:script id="banner" output="SSH-2.0-OpenSSH_5.3p1 Debian 3ubuntu4"/></port>
<script protocol="tcp" portid="80">state state="open" reason="syn-ack" reason_ttl="64"/><service name="http" product="Apache HTTP" version="2.2.14" extrainfo="(Ubuntu) mod_mono/2.4.3 PHP/5.3.2-Iubuntu04_38 with Suhosin-Patch proxy_html/3.0.1 mod_python/3.1 python/2.6.5 mod_ssl/2.2.14 OpenSSL/0.9.8k Mxusion Postgresql/4.0.38 mod_perl/2.0.4 Perl/v5.10.1" method="probed" conf="10"><cpes/cpe:/a:apache:http_server:2.2.14/><cpes/cpe:/service:port/>
<script protocol="tcp" portid="139">state state="open" reason="syn-ack" reason_ttl="64"/><service name="metbios-smb" product="Samba smb" version="3.X - 4.X" extrainfo="workgroup: WORKGROUP" method="probed" conf="10"><cpes/cpe:/a:samba:samba/><cpes/cpe:/service:port/>
<script protocol="tcp" portid="143">state state="open" reason="syn-ack" reason_ttl="64"/><service name="imap" product="Courier-Imap" extrainfo="(released 2008)" method="probed" conf="10"/><script id="banner" output=" OK (CAPABILITY) IMP4rev1 UIDPLUS CHILDREN NAMESPACE THREAD=ORWKA;DEREDSUBJECT THREAD=REFERENCES SORT QUOTA IDLE ACL ACL2=UNION) Courier-Imap ready. Copyright 1998-2008 Double Precision, Inc. See COPYING for&#x27; distribution information."/></port>
<script protocol="tcp" portid="443">state state="open" reason="syn-ack" reason_ttl="64"/><service name="http" product="Apache HTTP" version="2.2.14" extrainfo="(Ubuntu) mod_mono/2.4.3 PHP/5.3.2-Iubuntu04_38 with Suhosin-Patch proxy_html/3.0.1 mod_python/3.1 python/2.6.5 mod_ssl/2.2.14 OpenSSL/0.9.8k Mxusion Postgresql/4.0.38 mod_perl/2.0.4 Perl/v5.10.1" tunnel="ssl" method="probed" conf="10"><cpes/cpe:/a:apache:http_server:2.2.14/><cpes/cpe:/service:port/>
<script protocol="tcp" portid="445">state state="open" reason="syn-ack" reason_ttl="64"/><service name="metbios-smb" product="Samba smb" version="3.X - 4.X" extrainfo="workgroup: WORKGROUP" method="probed" conf="10"><cpes/cpe:/a:samba:samba/><cpes/cpe:/service:port/>
<script protocol="tcp" portid="5001">state state="open" reason="syn-ack" reason_ttl="64"/><service name="java-object" product="Java Object Serialization" method="probed" conf="10"/><script id="banner" output="\VAC\ED\X60\X65"/></port>
<script protocol="tcp" portid="8080">state state="open" reason="syn-ack" reason_ttl="64"/><service name="http" product="Apache Tomcat/Coyote JSP engine" version="1.1" method="probed" conf="10"><cpes/cpe:/a:apache:coyote_http_connector:1.1/></service:port/>
<script protocol="tcp" portid="8081">state state="open" reason="syn-ack" reason_ttl="64"/><service name="http" product="Jetty" version="6.1.25" method="probed" conf="10"><cpes/cpe:/a:mozilla:jetty:6.1.25/><cpes/cpe:/service:port/>
</ports>
<files frite="126" rttvar="10" to="1000000"/>
</host>
<host starttime="1663769277" endtime="1663769312"-->status state="up" reason="arp-response" reason_ttl="0"/>
<address addr="88:00:27:2F:FA:4A" addrtype="mac" vendor="Oracle VirtualBox virtual NIC"/>
</hostnames>
</hostnames>
```

Listing 6: Top 1000 ports xml file

### 3.5. OS DETECTION

Another important task to obtain is the detection of the operating system present on each live host. Although this detection is not 100% correct, for the most part it detects well the OS. This is done by the function `osScan()` (Listing 7) The name of the output file is stated, followed by the Nmap command which contains the name of the operating system, the chosen target, and the following options:

- **-O**: Detection of the operating system.
- **(-n, -Pn, -T4, --min-parallelism 100, --min-rate 64, -vv and -oX** were stated before)

After the file is generated, a *for loop* iterates through the xml root to find the different IP addresses and consequently the name of the operating system. In the case it's not possible to identify, the space is left as "Unknown". All the information is appended to the list `os_name[]` to later be added to the database.

```

def osScan(targets, out_xml):
    out_xml = os.path.join(out_xml, f'osdetection.xml')
    nmap_cmd = f"/usr/bin/nmap {targets} -n -Pn -O -T4 --min-parallelism 100 --min-rate 64 -vv -oX {out_xml}"
    sub_args = shlex.split(nmap_cmd)
    subprocess.Popen(sub_args, stdout=subprocess.PIPE, stderr=subprocess.PIPE).communicate()
    makeInvokerOwner(out_xml)

```

Listing 7: Nmap OS Detection function

## 3.6. SERVICE AND VERSION DETECTION

During the scan of the open ports (file `top_1000_portscan.xml`), the Nmap script also detected the service running on that port and the consequently version of it. From this file it's extracted all the information needed using a *for loop*. The same way as before, it's used the root to iterate through all the necessary attributes, in this case a new iteration is needed because inside each host element, there is the IPv4 address and the MAC address. The elements inside the element *address* need to be accessed by an iteration, in order to find the subelements and their corresponding attributes.

First and foremost, the initial attribute needed is the "addr" which contains the actual IPv4 and MAC of the host, if the attribute is different than MAC, then the second part of extracting information begins. Both *states* and *ports* variables hold the state of the port and the port number, respectively. To access the subelements inside the element *ports*, a new iteration is needed in which this time it will iterate through the range of the existing ports. Retrieving the port ID, the protocol and service present. A few variables are initiated that will hold in each iteration the strings containing the cpe name, service name, service product and service version. Once more, inside each available port, an iteration will try to find any existing service. Retrieving any value inside the attributes stated before. For the cpe case, the script will try to find any existing element called 'cpe' and through its own iteration, gather the text inside of it (Listing 8).

As per the end of each ports iteration, all the variables are added to the database with the use of the function *insert\_content\_service()*, with two variables *get\_host\_number[]* and *os\_name[]* being lists which contain the previous spoken host to be used as foreign key in the database and the corresponding OS name also previously mentioned, respectively.

During the iteration, the table *HostDiscovery* was also update with the total number of vulnerabilities found in each individual port. The first table *ScanNumber* is equally updated with the total number of vulnerabilities found on the entire scan.

```

for host in xml_root.echo.findall('host'):
    for address in host.findall('address'):
        #echo_ip = host.find('address').get('addr')
        echo_ip = address.attrib['addr']
        if(address.attrib['addrtype']!='mac'):
            data = {
                'address': echo_ip,
                'ports': []
            }
            states = host.findall('ports/port/state')
            ports = host.findall('ports/port')
            for i in range(len(ports)):
                port_id = ports[i].attrib['portid']
                protocol = ports[i].attrib['protocol']
                services = ports[i].findall('service')
                #cpe_list = []
                cpe_new = ""
                service_name = ""
                service_product = ""
                service_version = ""
                for service in services:
                    for key in ['name', 'product', 'version']:
                        if key in service.attrib:
                            if key == 'name':
                                service_name = service.attrib['name']
                            elif key == 'product':
                                service_product = service.attrib['product']
                            elif key == 'version':
                                service_version = service.attrib['version']
                cpes = service.findall('cpe')
                for cpe in cpes:
                    #cpe_list.append(cpe.text)
                    cpe_new = cpe.text
                data['ports'].append({
                    'port_id': port_id,
                    'protocol': protocol,
                    'service_name': service_name,
                    'service_product': service_product,
                    'service_version': service_version,
                    'cpes': cpe_new
                })

```

Listing 8: Service and version detection - code

```

get_service_name.append(insert_content_service(conn, (get_host_number[count],echo_ip, os_name[count_new_os],
                                                    port_id, protocol, service_name, service_product, service_version, cpe_new)))

echo_real_ip = echo_ip
conn.commit()

parsed_data.append(data)
if cpe_new=="":
    count_new +=1
else:
    cve_numb = cve_scan(cpe_new,get_service_name[count_new], conn)
    vuln_number_total = vuln_number_total + cve_numb
    vuln_number_ip = vuln_number_ip + cve_numb

    count_new +=1
    #count_new +=count_new

else:
    update_content_host_discovery(conn, (vuln_number_ip, echo_ip, scan))
    conn.commit()

count +=1

count_new_os += 1
#update_content_host_discovery(conn, (vuln_number_ip, echo_real_ip))
#conn.commit()
vuln_number_ip = 0
update_content_scan_number(conn, (vuln_number_total,scan))
conn.commit()

```

Listing 9: Various updates and inserts into the database

## 3.7. SSL/TLS DETECTION

With the conclusion of the port scan for each host, it's possible to detect if any port running a web service has SSL/TLS. First, the xml file containing the port scan is taken to the variable `in_ssl_xml` to be sent to the `parseDiscoverPorts()` function (Listing 11). This function takes the xml file and retrieves the hosts and ports found (Listing 10). With this information a *for loop* will extract each target and corresponding open ports and begin the SSL/TLS cipher and certificate scan. The two functions, illustrated on the Listing 12, shows the Nmap commands used for this detection. Besides previously mentioned commands, the only new command in each function is, `ssl-enum-ciphers` used to initiate SSLv3/TLS connections and record if the host accepts it or not, and `ssl-cert` which retrieves a server SSL certificate.

Although both functions integrate and complete the same table on the database, they are run separately because they both create a separate file. The certificate function will store on the database the information about the IP, the port, the name and issuer and the starting and expiring date of the certificate. While the cipher function stores on the database the protocol used for that specific service. They both generate xml files.

Finally, two more functions are called to extract the information from the previously generated xml files.

```
print("Beginning SSL scan...\n")

in_ssl_xml = 'top_1000_portscan.xml'
targets_ssl = parseDiscoverPorts(in_ssl_xml)

for target in targets_ssl:
    element = target.split()
    target_ip = element[0]
    target_ports = element[1]
    print(f'Scanning: {target_ip} against ports {target_ports}')
    sslCipherScan(target_ip, target_ports, os.getcwd())
    sslCertScan(target_ip, target_ports, os.getcwd())

ssl_certs(conn, get_host_number, get_scan_number)
ssl_ciphers(conn, get_host_number)
```

Listing 10: SSL/TLS xml files creation

```

def parseDiscoverPorts(in_xml):
    results = []
    port_list = ''
    xml_tree = ET.parse(in_xml)
    xml_root = xml_tree.getroot()
    for host in xml_root.findall('host'):
        ip = host.find('address').get('addr')
        ports = host.findall('ports')[0].findall('port')
        for port in ports:
            state = port.find('state').get('state')
            if state == 'open':
                port_list += port.get('portid') + ','
        port_list = port_list.rstrip(',')
        if port_list:
            results.append(f"{ip} {port_list}")
        port_list = ''
    return results

```

Listing 11: Host and Port information extraction from xml file

```

def sslCipherScan(target_ip, target_ports, out_xml):
    out_xml = os.path.join(out_xml, f'{target_ip}_ssl_ciphers.xml')
    nmap_cmd = f'usr/bin/nmap {target_ip} -p {target_ports} -n -Pn --script ssl-enum-ciphers -T4 -vv -oX {out_xml}'
    sub_args = shlex.split(nmap_cmd)
    subprocess.Popen(sub_args, stdout=subprocess.PIPE, stderr=subprocess.PIPE).communicate()

def sslCertScan(target_ip, target_ports, out_xml):
    out_xml = os.path.join(out_xml, f'{target_ip}_ssl_certs.xml')
    nmap_cmd = f'usr/bin/nmap {target_ip} -p {target_ports} -n -Pn --script ssl-cert -T4 -vv -oX {out_xml}'
    sub_args = shlex.split(nmap_cmd)
    subprocess.Popen(sub_args, stdout=subprocess.PIPE, stderr=subprocess.PIPE).communicate()

```

Listing 12: Nmap SSL/TLS Detection function

## 3.8. METHOD

With all the data gathered until this point, including the live hosts, their ports and services available, it's time to detect if there are any vulnerabilities related to them. To do this, a few ideas were proposed. Initially, given the service name and version, a script would search on the NVD database for any vulnerability associated with it, however this option was soon discontinued because not only it would return too many false positives but also the constant query for vulnerabilities in specific products online, is not the best security practice. With the idea of security in mind, a few open-source tools were tested to handle the searching part on the database. However, most of them were standalone tools with not an easy way to integrate into a Flask application.



With this in mind, a new approach was made, instead of searching for a service name and obtaining the CVE ID and the rest of the information, the NMAP script already outputs the translation of the service name and version to the corresponding cpe string.

The cpe string, also allows for search on the NVD, returning any vulnerabilities affecting that specific product, reducing the number of false positives. To ensure the security of this query, the use of an API was needed. With the correct implementation of the NVD, a new vulnerability database was added to the application, VulDB.

As seen on the Figure 15 the process begins with the port scan, followed by the corresponding cpe string extracted from it. The cpe string is added to the database and consequently used to search for any match on the NVD API and VulDB API. The result of it is a JSON file that is used to extract the necessary information of all the vulnerabilities found and finally added to the database to be displayed on the web application. This processes of extracting from the JSON file is explained in more detail on the chapter 4.2.

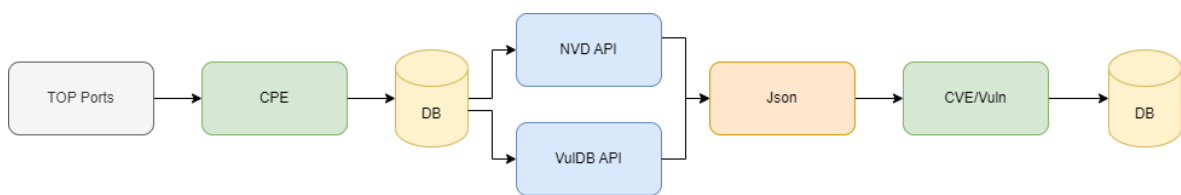


Figure 15: Vulnerability detection process

## 3.9. API

API stands for Application Programming Interface, meaning a set of definitions and protocols in order to build and integrate application software [48].

An API works in a way to let a product or a service to communicate with other products and services without having to know how they were implemented. This allows for a flexibility when creating tools or products because it can extract information from different sources in a secure and easy way, shifting most of the work to the actual development of that specific tool. APIs are basically a simplified way to connect an infrastructure through cloud-native app development, which also allows for data sharing between customers and external users.

A good example of a popular API is Google Maps API, allowing for the integration into other applications of google maps without requiring anything but a simple API communication.

Web APIs usually use HTTP for request messages and provide a definition of the response messages structure. These response messages are normally in the form of an XML or JSON file, as they present data in a way that is easy to be used and manipulated for other apps.

## **NVD API**

In this application to understand and identify the existence of vulnerabilities, it was used the CVE API [49] to easily retrieve information on the collection of CVE found by the application. NVD detains one of the biggest collections of CVE, more than 190000, for this reason, NVD was chosen to be the main database to check the vulnerability found. All the requests to the API use the method HTTP GET and each one has a unique base URL, with the possibility of adding parameters to the URL query in order to filters those same requests for specific information. They function is a similar way to the filters/parameters found on the NVD CVE search page.

As many public APIs available, there is a limit to how many requests it's possible to make in a given time. The public rate limit without any API Key is 5 requests in a 30 second window while with an API Key this limit is upgraded to 50 requests in the same time period. This shoes the usefulness and the need for an API Key, however it is still recommended that the application using the API sleeps for some seconds between the requests, to be sure no legitimate requests get denied. In order to request an API Key an account had to be made in the NVD website so that the API Key would be associated with the account created.

Searching for a specific cpe through the NVD API, returns the follow schema in the JSON format:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "JSON Schema for NVD CVE Applicability Statement CPE Data Feed version 1.0",
  "id": "https://scap.nist.gov/schema/nvd/feed/1.0/nvd_cpematch_feed_json_1.0.schema",
  "definitions": {
    "def_cpe_name": {
      "description": "CPE name",
      "type": "object",
      "properties": {
        "cpe22Uri": {
          "type": "string"
        },
        "cpe23Uri": {
          "type": "string"
        }
      }
    },
    "required": [
      "cpe23Uri"
    ]
  },
  "def_cpe_match": {
    "description": "CPE match string or range",
    "type": "object",
    "properties": {
      "vulnerable": {
        "type": "boolean"
      },
      "cpe22Uri": {
        "type": "string"
      },
      "cpe23Uri": {
        "type": "string"
      },
      "versionStartExcluding": {
        "type": "string"
      },
      "versionStartIncluding": {
        "type": "string"
      },
      "versionEndExcluding": {
        "type": "string"
      },
      "versionEndIncluding": {
        "type": "string"
      },
      "cpe_name": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/def_cpe_name"
        }
      }
    }
  }
},
```

Figure 16: API JSON schema - 1

```

"required": [
  "vulnerable",
  "cpe23Uri"
]
},
"def_node": {
  "description": "Defines a node or sub-node in an NVD applicability statement.",
  "properties": {
    "operator": {"type": "string"},
    "negate": {"type": "boolean"},
    "children": {
      "type": "array",
      "items": {"$ref": "#/definitions/def_node"}
    },
    "cpe_match": {
      "type": "array",
      "items": {"$ref": "#/definitions/def_cpe_match"}
    }
  }
},
"def_configurations": {
  "description": "Defines the set of product configurations for a NVD applicability statement.",
  "properties": {
    "CVE_data_version": {"type": "string"},
    "nodes": {
      "type": "array",
      "items": {"$ref": "#/definitions/def_node"}
    }
  },
  "required": [
    "CVE_data_version"
  ]
},
"def_subscore": {
  "description": "CVSS subscore.",
  "type": "number",
  "minimum": 0,
  "maximum": 10
},
"def_impact": {
  "description": "Impact scores for a vulnerability as found on NVD.",
  "type": "object",
  "properties": {
    "baseMetricV3": {
      "description": "CVSS V3.x score.",
      "type": "object",
      "properties": {
        "cvssV3": {"$ref": "cvss-v3.x_beta.json"},
        "exploitabilityScore": {"$ref": "#/definitions/def_subscore"},
        "impactScore": {"$ref": "#/definitions/def_subscore"}
      }
    }
  }
},

```

Figure 17: API JSON schema - 2

```

    "baseMetricV2": {
      "description": "CVSS V2.0 score.",
      "type": "object",
      "properties": {
        "cvssV2": {"$ref": "cvss-v2.0_beta.json"},
        "severity": {"type": "string"},
        "exploitabilityScore": {"$ref": "#/definitions/def_subscore"},
        "impactScore": {"$ref": "#/definitions/def_subscore"},
        "acInsufInfo": {"type": "boolean"},
        "obtainAllPrivilege": {"type": "boolean"},
        "obtainUserPrivilege": {"type": "boolean"},
        "obtainOtherPrivilege": {"type": "boolean"},
        "userInteractionRequired": {"type": "boolean"}
      }
    },
    "def_cve_item": {
      "description": "Defines a vulnerability in the NVD data feed.",
      "properties": {
        "cve": {"$ref": "CVE_JSON_4.0_min_1.1_beta.schema"},
        "configurations": {"$ref": "#/definitions/def_configurations"},
        "impact": {"$ref": "#/definitions/def_impact"},
        "publishedDate": {"type": "string"},
        "lastModifiedDate": {"type": "string"}
      },
      "required": ["cve"]
    },
    "type": "object",
    "properties": {
      "CVE_data_type": {"type": "string"},
      "CVE_data_format": {"type": "string"},
      "CVE_data_version": {"type": "string"},
      "CVE_data_numberOfCVEs": {
        "description": "NVD adds number of CVE in this feed",
        "type": "string"
      },
      "CVE_data_timestamp": {
        "description": "NVD adds feed date timestamp",
        "type": "string"
      },
      "CVE_Items": {
        "description": "NVD feed array of CVE",
        "type": "array",
        "items": {"$ref": "#/definitions/def_cve_item"}
      }
    },
    "required": [
      "CVE_data_type",
      "CVE_data_format",
      "CVE_data_version",
      "CVE_Items"
    ]
  }
}

```

Figure 18: API JSON schema – 3

From all those parameters, the ones used for the application will be:

- baseMetricV2/baseMetricV3 – Gathering the information about the score for both versions.
- CVE-ID – Gathering the ID of the corresponding CVE.
- Current Description – Gathering the text information about the given vulnerability.

The data collected from those 4 parameters are enough for the whole base of detecting and presenting the vulnerabilities found to be built.

From that point on, the data is processed and stored in a database and consequently shown on the web application as a final display of the product. A typical API request to the NVD is shown on Listing 13 where *key* represents the API key and *cpe\_name*, the variable that holds the cpe to be searched on the NVD.

```
api = 'https://services.nvd.nist.gov/rest/json/cves/1.0?apiKey={key}&cpeMatchString={cpe_name}'
```

Listing 13: : API request

The process of API request and information extraction from the result is done on the function *cve\_scan*. This function accepts the cpe string, ex: *cpe:/a:apache:http\_server:2.2*, the service name, representing the connection between two databases, and the connection to the database.

The *uri* variable takes the *cpe\_name* and *key* (hidden) into the API format which will then be used in the GET request to the NVD. The response is saved and right after loaded in JSON format, enabling an easier way to navigate through the data received and retrieve the necessary parts.

Using a *for loop* to iterate through the property with the name “CVE\_Items”, which contains all the useful data for this application. Given that step, we set a few variables to retrieve the attributes of that property like, *cve\_id*, identifying the ID of each CVE found by the attributes `['cve']['CVE_data_meta']['ID']` and equally important, the description of each vulnerability by the attributes `['cve']['description']['description_data'][0]['value']`. The `[0]` makes sure it only retrieves the first description of that vulnerability, preventing any wrong description.

The next step retrieves the Base Metric for the CVSS version 3, to provide the Base score of each vulnerability found. This score will later be important for the classification of the vulnerability severity. In some cases (mostly the old vulnerabilities), there is no base score, which in that case, it is given an empty string to the variable *cvssv3\_base\_score*. The same is applied to the Base Metric for the CVSS version 2. Although the version 3 is the latest and most reliable version of severity classification, some CVEs still use and often times only have the version 2.

At last, all the data collected is added to the database using the function `insert_content_cve` with all the used variables, along with the initial one, `get_service_name`. As stated before, this variable will connect two databases, since the current one holds all the vulnerabilities information, it is needed a connection using a foreign key to the previous database that holds the information regarding the IPs found, their open ports, services detected and corresponding cpe string. Without it, it would not be possible to associate each cpe string generated from the service running on a specific port, to all the potential vulnerabilities related to that service and version of it.

Given that commit to the database, a counting of the total number of vulnerabilities found for that cpe is counted and returned, as we can see on Listing 14, with the commit done, a second source of vulnerabilities is used by sending a list of all the CVE-IDs found to the function `cve_scan_vulndb()`, along with the current cpe string, the table id (`service_name`) and the total number of vulnerabilities found so far. The function returns the number of vulnerabilities found on the new API and finally the `sleep()` function is called to give sometime between the API requests, to not exceed any limits.

```
def cve_scan(cpe_new, get_service_name, conn):
    cpe_name = cpe_new
    get_service_name = get_service_name
    vuln_number_total = 0
    cve_list = []
    key = ''
    api = 'https://services.nvd.nist.gov/rest/json/cves/1.0?apiKey={key}&cpeMatchString={cpe_name}'
    uri = api.format(key=key, cpe_name=cpe_name)
    print("uri", uri)
    response = requests.get(uri)
    print("response", response)
    json_data = json.loads(response.text)

    vulnerabilities = json_data['result']['CVE_Items']
    for vuln in vulnerabilities:
        cve_id = vuln['cve']['CVE_data_meta']['ID'] # CVE-Get ID
        current_description = vuln['cve']['description']['description_data'][0]['value'] #Get Current Description
        cwe_id = vuln['cve']['problemtype']['problemtype_data'][0]['description'][0]['value'] # CVE-Get ID

        #Get BaseScore and VectorString if you have CVSS v3 information
        if 'baseMetricV3' in vuln['impact']:
            cvssv3_base_score = vuln['impact']['baseMetricV3']['cvssV3']['baseScore']
            cvssv3_base_severity = vuln['impact']['baseMetricV3']['cvssV3']['baseSeverity']
            cvssv3_vector_string = vuln['impact']['baseMetricV3']['cvssV3']['vectorString']
        else:
            cvssv3_base_score = ""
            cvssv3_vector_string = ""
            cvssv3_base_severity = ""

        if 'baseMetricV2' in vuln['impact']:
            #Get BaseScore and VectorString for CVSS v2
            cvssv2_base_score = vuln['impact']['baseMetricV2']['cvssV2']['baseScore']
            cvssv2_severity = vuln['impact']['baseMetricV2']['severity']
            cvssv2_vector_string = vuln['impact']['baseMetricV2']['cvssV2']['vectorString']
        else:
            cvssv2_base_score = ""
            cvssv2_vector_string = ""
            cvssv2_severity = ""
        cve_list.append(cve_id)
        insert_content_cve(conn, ((get_service_name, cpe_name, cve_id, cvssv2_base_score, cvssv2_severity, cvssv3_base_score, cvssv3_base_severity, current_description)))
        conn.commit()
        vuln_number_total += 1
    more_vuln = cve_scan_vulndb(cve_list, cpe_new, get_service_name, conn, vuln_number_total)
    vuln_number_total = vuln_number_total + more_vuln
    time.sleep(10)
    return vuln_number_total
```

Listing 14: NVD API Function

## VulDB API

For a more complete vision of the existing vulnerabilities, a second database was used to validate and add extra vulnerabilities to the application. VulDB stands for Vulnerability Database and is one of the biggest vulnerability databases, providing important sources for people responsible for vulnerability management, vulnerability handling, cyber threat intelligence, among others.

VulDB also allows for the access of the data through an API, although more limited in terms of free access than the NVD API. Data can be requested by HTTP requests with a JSON response, each individual request uses at least 1 credit, a maximum of 50 credits are given each day (reset of credits) and there should not be more than 10 requests per minute, otherwise the API response may be blocked. To hold more credits, an upgrade must be purchased to do so. Contrary to the other API, this one requires an API key for every access, otherwise no data can be retrieved [50].

The authentication data is sent as HTTP POST, instead of HTTP GET and all other parameters must be added as headers, such as the *fields*, the *cpe\_name* and the *Apikey*.

The parameter *fields*, expands the results of the API, since by default the details are given in a limited way, with less variables. It is possible to change for a full detail more, but it requires extra credits for each request. Because of this, a third option was used, requesting the limited details but adding 3 important variables that were not added by default, like the summary of the vulnerability, and both base score for the CVSS version 2 and 3.

```
key =
fields = 'entry_summary,vulnerability_cvss2_vuldb_basescore,vulnerability_cvss3_vuldb_basescore'
url = 'https://vuldb.com/?api'
postData = {'search': cpe_name, 'fields': fields}

headers = {'X-VulDB-ApiKey': key}

response = requests.post(url, headers=headers, data=postData)
json_data = json.loads(response.content)
```

Listing 15: VulDB API request



Searching for a specific cpe through the VulDB API, returns the follow schema in the JSON format:

```
"result": [
  {
    "entry": {
      "id": "122889",
      "title": "Apache HTTP Server up to 2.2.31/2.4.23 mod_userdir HTTP Response Splitting crlf injection",
      "summary": "A vulnerability, which was classified as critical, was found in Apache HTTP Server up to 2.2.31/2.4.23 (Web Server). This affects an unknown code of the component mod_userdir. Upgrading to version 2.2.32 or 2.4.25 eliminates this vulnerability. A possible mitigation has been published 3 weeks after the disclosure of the vulnerability.",
      "timestamp": {
        "create": "1534387997",
        "change": "1584285111"
      }
    },
    "vulnerability": {
      "risk": {
        "value": "2",
        "name": "medium"
      },
      "cvss2": {
        "vuldb": {
          "basescore": "6.8"
        }
      },
      "cvss3": {
        "vuldb": {
          "basescore": "7.3"
        }
      }
    },
    "advisory": {
      "date": "1534197600"
    },
    "source": {
      "cve": {
        "id": "CVE-2016-4975"
      }
    }
  }
],
```

Figure 19: VulDB API JSON schema

From all those parameters, the ones used for the application will be:

- **Summary** – Gathering the text information about the given vulnerability
- **Cvssv2/Cvssv3** – Gathering the information about the score for both versions.
- **CVE-ID** – Gathering the ID of the corresponding CVE.

Contrary to before, the severity level is not gathered from the API, because only three extra parameters can be attached to the request, and while those 3 chosen cannot be replicated, the severity level can be easily calculated depending on the score number. The data collected from those 3 parameters allow for the whole base of detecting and presenting the vulnerabilities found to be built. From that point on, the data is processed and stored in a database and consequently shown on the web application as a final display of the product.

Following the API request, the function tries to get the results from it, in case there is no, meaning there are no vulnerabilities for that specific cpe string, the code is skipped through the exception. In the case there is indeed results found, a *for loop* is used to iterate through the JSON result and obtain all the parameters previously described. For the severity level of each version, it is attributed the level depending on the extracted base score. Finally, all the data is stored into the database (Listing 16).

```

def cve_scan_vulndb(cve_list, cpe_new, get_service_name, conn, vuln_number_total):
    cpe_name = cpe_new
    key =
    fields = 'entry_summary,vulnerability_cvss2_vuldb_basescore,vulnerability_cvss3_vuldb_basescore'
    url = 'https://vuldb.com/?api'
    postData = {'search': cpe_name, 'fields': fields}
    headers = {'X-VulDB-APIKey': key}
    response = requests.post(url, headers=headers, data=postData)
    json_data = json.loads(response.content)
    try:
        vulnerabilities = json_data['result']
        for vuln in vulnerabilities:
            cve_id = vuln['source']['cve']['id'] # CVE-Get ID
            current_description = vuln['entry']['summary'] #Get Current Description
            #Get BaseScore and VectorString if you have CVSS v3 information
            if 'cvss3' in vuln['vulnerability']:
                cvssv3_base_score = vuln['vulnerability']['cvss3']['vuldb']['basescore']
                if cvssv3_base_score >= '0.1' and cvssv3_base_score <= '3.9':
                    cvssv3_base_severity = "LOW"
                elif cvssv3_base_score >= '4.0' and cvssv3_base_score <= '6.9':
                    cvssv3_base_severity = "MEDIUM"
                elif cvssv3_base_score >= '7.0' and cvssv3_base_score <= '8.9':
                    cvssv3_base_severity = "HIGH"
                elif cvssv3_base_score >= '9.0' and cvssv3_base_score <= '10.0':
                    cvssv3_base_severity = "CRITICAL"
            else:
                cvssv3_base_score = ""
                cvssv3_base_severity = ""
            if 'cvss2' in vuln['vulnerability']:
                #Get BaseScore and VectorString for CVSS v2
                cvssv2_base_score = vuln['vulnerability']['cvss2']['vuldb']['basescore']
                if cvssv2_base_score >= '0.1' and cvssv2_base_score <= '3.9':
                    cvssv2_severity = "LOW"
                elif cvssv2_base_score >= '4.0' and cvssv2_base_score <= '6.9':
                    cvssv2_severity = "MEDIUM"
                elif cvssv2_base_score >= '7.0' and cvssv2_base_score <= '8.9':
                    cvssv2_severity = "HIGH"
                elif cvssv2_base_score >= '9.0' and cvssv2_base_score <= '10.0':
                    cvssv2_severity = "CRITICAL"
            else:
                cvssv2_base_score = ""
                cvssv2_severity = ""
            if cve_id in cve_list:
                pass
            else:
                insert_content_cve(conn, ((get_service_name,cpe_name, cve_id, cvssv2_base_score, cvssv2_severity, cvssv3_base_score, cvssv3_base_severity, current_description)))
                conn.commit()
                vuln_number_total += 1
    except:
        pass
    return vuln_number_total

```

Listing 16: VulDB API function

## 3.10. SSL/TLS VULNERABILITIES

As described before on the previous chapter, a scan to potential SSL/TLS information is executed. With the resulting xml files, two functions *ssl\_certs()* and *ssl\_ciphers()* will extract the important information on both xml files for each host with SSL/TLS related content. The first function *ssl\_certs()* (Listing 17) will iterate through all the found files with an ending extension of “\_ssl\_certs.xml” and analyse it to retrieve the IP Address, the port, the certificate name and the issuer and the beginning and ending of the certificate.

Given this its necessary to add to the database table *SSLDiscovery* all the corresponding information, along with the foreign key retrieved from the *HostDiscovery* table and at the same time it has to match the correct port of that host on the table *ServiceScan* and finally match the identical foreign key from the *CVEScan* table. All this allows for the correct showing on the website of the vulnerability that may occur from the SSL/TLS outdated protocol or expired certificate date.

Still on Listing 17, the table SSLDiscovery is also populated with the possible vulnerability from the expired certificate date by comparing the data with the present date. To avoid any mismatch of the files, a condition checks if the corresponding IP of the file is on the current scan in the database. The corresponding severity and score of the expired date vulnerability was taken from the official plug in of the Nessus software from Tenable [51]. A flag is also placed as “true” on the database to more easily be spotted a vulnerability regarding SSL/TLS. The two arguments left empty correspond to the protocol version and the flag regarding the protocol vulnerability.

```

primary_key_host = get_host_id_ssl(conn, get_scan_number, ip)
today_date = datetime.date.today()
primary_key_scan = get_host_id_cve(conn, get_host_number, ip, port_id)
if not primary_key_scan and not primary_key_host:
    pass
else:
    if(cert_end < str(today_date)):
        insert_content_ssl(conn, (primary_key_host[0], ip, port_id, '', host_common_name, issuer_common_name, cert_start, cert_end, 'true', ''))
        conn.commit()
        insert_content_cve(conn, (primary_key_scan[0], '', 'SSL/TLS', '5.0', 'MEDIUM', '5.3', 'MEDIUM', ('The remote servers SSL certificate on port ' + port_id +
        conn.commit()
    else:
        insert_content_ssl(conn, (primary_key_host[0], ip, port_id, '', host_common_name, issuer_common_name, cert_start, cert_end, 'false', ''))
        conn.commit()

```

Listing 17: SSL/TLS certificate check

For the function `ssl_ciphers()`, Listing 18, the process is similar to the other function, except the ending part which compares the protocol version with a known vulnerable and old version, the TLSv1.0. With the retrieved data, a condition will check whether or not the protocol is vulnerable and will update the database with the corresponding flag and the score, severity and description based on official plug in of the Nessus software from Tenable [52].

```

def ssl_ciphers(conn, get_host_number):
    # Protocol Report Lists
    tls_v10_report = []
    tls_v11_report = []
    ssl_v3_report = []
    primary_key_scan_list = []
    # File Path
    in_path = '/home/kali/Desktop/application'
    # Cycle through each XML file
    for file in os.listdir(in_path):
        # Load each XML file
        if file.endswith('.ssl_ciphers.xml'):
            in_xml = os.path.join(in_path, file)
            xml_tree = ET.parse(in_xml)
            xml_root = xml_tree.getroot()
            # Cycle through each host
            for host in xml_root.findall('host'):
                ip = host.find('address').get('addr')
                ports_element = host.findall('ports')
                port_element = ports_element[0].findall('port')
                # Cycle through each port
                for scanned_port in port_element:
                    port_id = scanned_port.get('portid')
                    script_element = scanned_port.find('script')
                    if script_element:
                        # Cycle through each protocol
                        protocol_element = script_element.findall('table')
                        for tls_protocol in protocol_element:
                            protocol_version = tls_protocol.attrib['key']
                            # Stage data for TLSv1.0 Table
                            if protocol_version in 'TLSv1.0':
                                flagged_tls_v10 += protocol_version + ','
                            # Stage data for TLSv1.1 Table
                            if protocol_version in 'TLSv1.1':
                                flagged_tls_v11 += protocol_version + ','
                            # Stage data for SSLv3 Table
                            if protocol_version in 'SSLv3':
                                flagged_ssl_v3 += protocol_version + ','
                    primary_key_host = get_host_id(conn, get_host_number, port_id, ip)
                    primary_key_scan = get_host_id_cve(conn, get_host_number, ip, port_id)
                    if not primary_key_scan and not primary_key_host:
                        pass
                    else:
                        if (protocol_version == 'TLSv1.0'):
                            update_content_ssl(conn, (protocol_version, 'true', primary_key_host[0], ip, port_id))
                            conn.commit()
                            insert_content_cve(conn, (primary_key_scan[0], '', 'SSL/TLS', '6.1', 'MEDIUM', '6.5', 'MEDIUM', ('The remote server on port ' + port_id +
                            conn.commit()
                        else:
                            update_content_ssl(conn, (protocol_version, 'false', primary_key_host[0], ip, port_id))
                            conn.commit()

```

Listing 18: SSL/TLS protocol version check



# 4. DATABASE

For the display of the information into the web application, a database with multiple tables was created to provide a bridge between the data gathered and the presentation of it. SQLite, a library that implements a SQL database engine, was chosen for the database creation, given that is an embedded, server-less relational database management system, does not require prior configuration, and can easily integrate into Flask.

The Figure 20 shows the database and the connection between each table, using *Primary Key* and *Foreign Key*. This was done to allow retrieving information from previous tables in order to cross data.

Primary key refers to a key that is unique for each record, as seen on Figure 21 It's associated with the column *id*, recording every new entry by a new consecutive number. It allows the identification of each row as a unique identifier.

Foreign key on the other hand is a column that is used to establish a link between the data in two different tables. As such, a second table associates each row to a specific *id* from the first table. This can be seen on Figure 22 where *scan\_id* is the foreign key from the table *HostDiscovery*, associated with the primary key *id* from the table *ScanNumber*.

Whenever a new scan is performed and the data saved on the table *ScanNumber* with an individual name from the column *scan*, an *id* is associated.

However, the information about the hosts found is saved in a second table. A link between the two tables is required so that each individual scan can link to all the hosts found for that one specific scan. This is accomplished by the foreign key *scan\_id*, on the table *HostDiscovery*, that has the same number as the primary key *id* from the table *ScanNumber*.

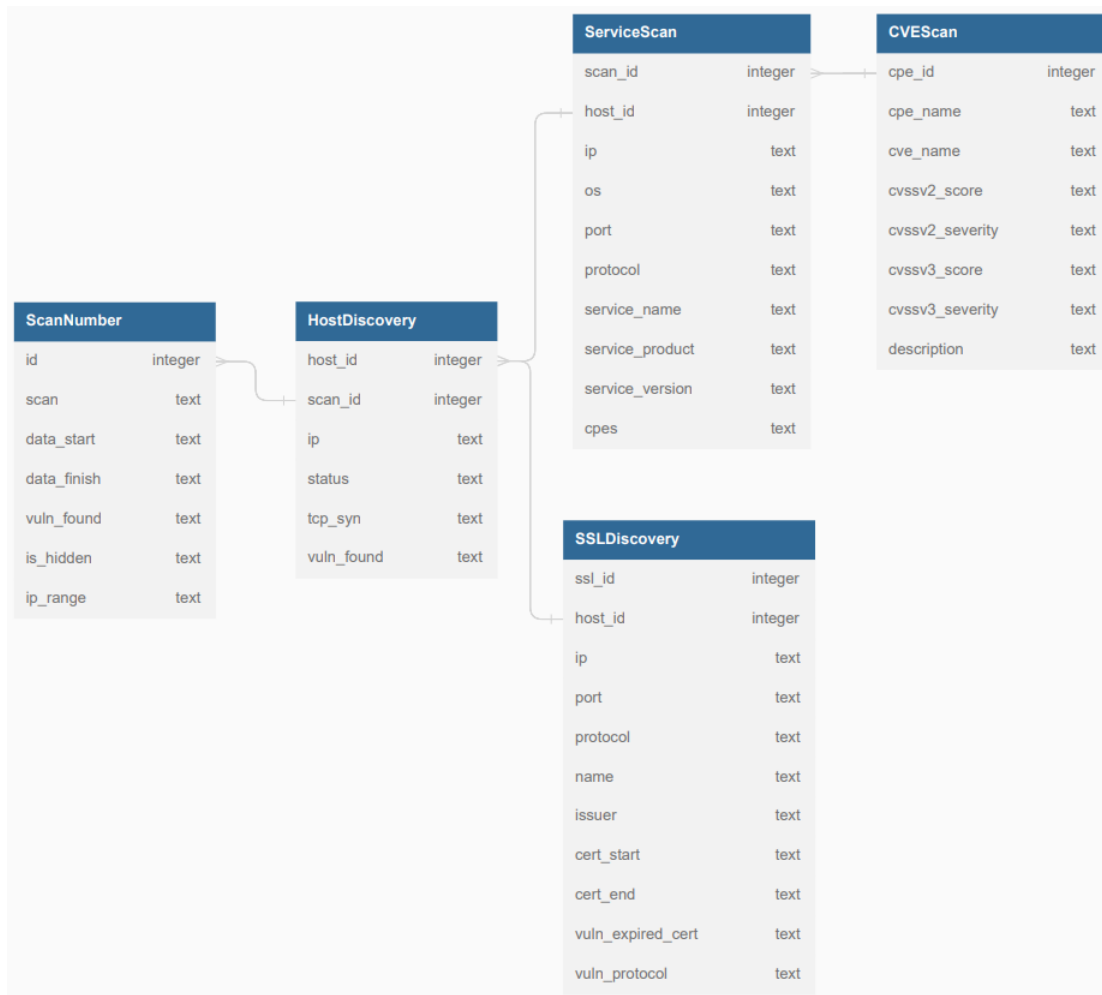


Figure 20: Database overview

The first table *ScanNumber*, gathers the information about each individual scan performed, its name, the date and time it started and ended, the total number of vulnerabilities found, a flag to show or not on the webpage and the IP range of the scan executed (Figure 21).

id	scan	date_start	date_finish	vuln_found	is_hidden	ip_range
Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1 testing_ssl	2022-10-23 14:31:34		810	False	192.168.1-3.0-255
2	2 testing_ssl	2022-10-23 14:33:39		810	False	192.168.1-3.0-255
3	3 testing_ssl_new	2022-10-23 14:52:30	2022-10-23 16:40:39	250	False	10.0.2.0/24
4	4 testing_ssl_new	2022-10-23 15:11:42	2022-10-23 16:40:39	250	False	10.0.2.0/24
5	5 testing_ssl_new	2022-10-23 15:18:01	2022-10-23 16:40:39	250	False	10.0.2.0/24
6	6 testing_ssl_new	2022-10-23 15:35:29	2022-10-23 16:40:39	250	False	10.0.2.0/24
7	7 testing_ssl_new	2022-10-23 15:44:26	2022-10-23 16:40:39	250	False	10.0.2.0/24
8	8 testing_ssl_new	2022-10-23 16:14:24	2022-10-23 16:40:39	250	False	10.0.2.0/24
9	9 testing_ssl_new	2022-10-23 16:27:26	2022-10-23 16:40:39	250	False	10.0.2.0/24
10	10 testing_ssl_new	2022-10-23 16:35:13	2022-10-23 16:40:39	250	False	10.0.2.0/24

Figure 21: Table ScanNumber

The second table *HostDiscovery*, gathers the information about each individual host found by the scan. The *host\_id* represents the primary key, *scan\_id* the foreign key, *ip* the IPv4 of the detected host, *status* indicating that the host is indeed live, *tcp\_syn* is the response given by the host, and *vuln\_found*, is the number of vulnerabilities found on that one host (Figure 22).

host_id	scan_id	ip	status	tcp_syn	vuln_found
Filter	Filter	Filter	Filter	Filter	Filter
1	1	10.0.2.1	NULL	up	0
2	2	10.0.2.2	NULL	up	40
3	3	10.0.2.3	NULL	up	0
4	4	10.0.2.4	NULL	up	102
5	5	10.0.2.7	NULL	up	0
6	6	2 10.0.2.1	up	arp-response	0
7	7	2 10.0.2.2	up	arp-response	40
8	8	2 10.0.2.3	up	arp-response	0
9	9	2 10.0.2.4	up	arp-response	102
10	10	2 10.0.2.5	up	arp-response	0
11	11	2 10.0.2.7	up	localhost-response	0
12	12	3 10.0.2.1	up	arp-response	0
13	13	3 10.0.2.2	up	arp-response	40
14	14	3 10.0.2.3	up	arp-response	0
15	15	3 10.0.2.4	up	arp-response	102

Figure 22: Table HostDiscovery

The third table *ServiceScan*, gathers the information about each individual host found by the scan. The *scan\_id* is the primary key, *host\_id* the foreign key, *ip* the host IPv4, *os* the Operating System detected, *port* the ports found for that IP Address, *protocol* is the protocol that the port is associated with, *service\_name* the name of the service found running on that port, *service\_product* the product associated with that service found, *service\_version* the version of the service found and, *cpes* the associated cpe to that service found. This table is linked to the *HostDiscovery* table by the foreign key *host\_id* to the primary key *host\_id* of the table *HostDiscovery*, associated each host found to all the possible ports that are active (Figure 23).

scan_id	host_id	ip	os	port	protocol	service_name	service_product	service_version	cpes
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	10.0.2.1	Grandstream GXP1105 VoIP phone	53	tcp	tcpwrapped			
2	2	10.0.2.2	Grandstream GXP1105 VoIP phone	135	tcp	msrpc	Microsoft Windows RPC		cpe:/o:microsoft:windows
3	3	2 10.0.2.2	Grandstream GXP1105 VoIP phone	445	tcp	microsoft-ds			
4	4	4 10.0.2.4	Linux 2.6.17 - 2.6.36	22	tcp	ssh	OpenSSH	5.3p1 Debian 3ubuntu4	cpe:/o:linux:linux_kernel
5	5	4 10.0.2.4	Linux 2.6.17 - 2.6.36	80	tcp	http	Apache httpd	2.2.14	cpe:/a:apache:http_server:2.2.14
6	6	4 10.0.2.4	Linux 2.6.17 - 2.6.36	139	tcp	netbios-ssn	Samba smbd	3.X - 4.X	cpe:/a:samba:samba
7	7	4 10.0.2.4	Linux 2.6.17 - 2.6.36	143	tcp	imap	Courier Imapd		
8	8	4 10.0.2.4	Linux 2.6.17 - 2.6.36	443	tcp	http	Apache httpd	2.2.14	cpe:/a:apache:http_server:2.2.14
9	9	4 10.0.2.4	Linux 2.6.17 - 2.6.36	445	tcp	netbios-ssn	Samba smbd	3.X - 4.X	cpe:/a:samba:samba
10	10	4 10.0.2.4	Linux 2.6.17 - 2.6.36	5001	tcp	java-object	Java Object Serialization		
11	11	4 10.0.2.4	Linux 2.6.17 - 2.6.36	8080	tcp	http	Apache Tomcat/Coyote JSP engine	1.1	cpe:/a:apache:coyote_http_connector:1.1
12	12	4 10.0.2.4	Linux 2.6.17 - 2.6.36	8081	tcp	http	Jetty	6.1.25	cpe:/a:mortbay:jetty:6.1.25

Figure 23: Table ServiceScan

The fourth table *SSLDiscovery* gathers the information about each SSL/TLS certificate found in all the ports of each host, and the protocol version. As the *ServiceScan*, this table also uses a foreign key associated with the primary key of the table *HostDiscovery*, in order to associate any specific host of each scan, to the discovery of the SSL/TLS certificates and the protocol version. Two flags also indicate the existence of vulnerabilities (Figure 24).

ss_id	host_id	ip	port	protocol	name	issuer	cert_start	cert_end	vuln_expired_cert	vuln_protocol
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	15 10.0.2.5	25	TLSv1.0	ubuntu804-base.localdomain	ubuntu804-base.localdomain	2010-03-17	2010-04-16	true	true
2	2	15 10.0.2.5	5432	TLSv1.0	ubuntu804-base.localdomain	ubuntu804-base.localdomain	2010-03-17	2010-04-16	true	true
3	3	14 10.0.2.4	443	TLSv1.0	owaspbwa	owaspbwa	2013-01-02	2022-12-31	false	true
4	4	21 10.0.2.5	25	TLSv1.0	ubuntu804-base.localdomain	ubuntu804-base.localdomain	2010-03-17	2010-04-16	true	true
5	5	21 10.0.2.5	5432	TLSv1.0	ubuntu804-base.localdomain	ubuntu804-base.localdomain	2010-03-17	2010-04-16	true	true
6	6	20 10.0.2.4	443	TLSv1.0	owaspbwa	owaspbwa	2013-01-02	2022-12-31	false	true
7	7	27 10.0.2.5	25	TLSv1.0	ubuntu804-base.localdomain	ubuntu804-base.localdomain	2010-03-17	2010-04-16	true	true
8	8	27 10.0.2.5	5432	TLSv1.0	ubuntu804-base.localdomain	ubuntu804-base.localdomain	2010-03-17	2010-04-16	true	true
9	9	26 10.0.2.4	443	TLSv1.0	owaspbwa	owaspbwa	2013-01-02	2022-12-31	false	true

Figure 24: Table SSLDiscovery



The fifth and last table *CVEScan*, gathers the information about each vulnerability found. For all the ports of all the hosts. The foreign key *cpe\_id* is linked with the primary key from the table *ServiceScan* to relate each vulnerability found to a specific port, as seen on the Figure 25. This table also holds each cpe string, the corresponding CVE-ID of the vulnerability, the CVSSV score for the version 2 and 3 and their respective severity and finally the description of the vulnerability. As it can be seen, sometimes there is only the severity and the score in one of the versions. This will be approached in more detail on the chapter 6.

cpe_id	cpe_name	cve_name	cvssv2_score	cvssv2_severity	cvssv3_score	cvssv3_severity	description
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
52	5 cpe:/a:apache:http_server:2.2.14	CVE-2014-0118	4.3	MEDIUM			The deflate_in_filter function in mod_deflate.c in the ...
53	5 cpe:/a:apache:http_server:2.2.14	CVE-2014-0226	6.8	MEDIUM			Race condition in the mod_status module in the Apache ...
54	5 cpe:/a:apache:http_server:2.2.14	CVE-2016-5387	6.8	MEDIUM	8.1	HIGH	The Apache HTTP Server through 2.4.23 follows RFC 38...
55	5 cpe:/a:apache:http_server:2.2.14	CVE-2016-8743	5.0	MEDIUM	7.5	HIGH	Apache HTTP Server, in all releases prior to 2.2.32 and ...
56	5 cpe:/a:apache:http_server:2.2.14	CVE-2022-22721	5.8	MEDIUM	9.1	CRITICAL	If LimitXMLRequestBody is set to allow request bodies ...
57	5 cpe:/a:apache:http_server:2.2.14	CVE-2022-28614	5.0	MEDIUM	5.3	MEDIUM	The ap_rwrite() function in Apache HTTP Server 2.4.53 ...
58	5 cpe:/a:apache:http_server:2.2.14	CVE-2022-29404	5.0	MEDIUM	7.5	HIGH	In Apache HTTP Server 2.4.53 and earlier, a malicious ...
59	5 cpe:/a:apache:http_server:2.2.14	CVE-2022-28615	6.4	MEDIUM	9.1	CRITICAL	Apache HTTP Server 2.4.53 and earlier may crash or ...
60	5 cpe:/a:apache:http_server:2.2.14	CVE-2022-31813	7.5	HIGH	9.8	CRITICAL	Apache HTTP Server 2.4.53 and earlier may not send th...
61	6 cpe:/a:samba:samba	CVE-2022-32743			7.5	HIGH	Samba does not validate the Validated-DNS-Host-Name ...
62	6 cpe:/a:samba:samba	CVE-2022-1615			5.5	MEDIUM	In Samba, GnuTLS gnutls_md() can fail and give ...
63	6 cpe:/a:samba:samba	CVE-2022-0336			8.8	HIGH	The Samba AD DC includes checks when adding service...
64	6 cpe:/a:samba:samba	CVE-2009-2906	4.0	MEDIUM			smbd in Samba 3.0 before 3.0.37, 3.2 before 3.2.15, 3.3...
65	6 cpe:/a:samba:samba	CVE-2018-1050	3.3	LOW	4.3	MEDIUM	All versions of Samba from 4.0.0 onwards are vulnerabl...
66	6 cpe:/a:samba:samba	CVE-2012-6150	3.6	LOW			The winbind_name_list_to_sid_string_list function in ...
67	6 cpe:/a:samba:samba	CVE-2020-14323	2.1	LOW	5.5	MEDIUM	A null pointer dereference flaw was found in samba's ...
68	6 cpe:/a:samba:samba	CVE-2014-0178	3.5	LOW			Samba 3.6.6 through 3.6.23, 4.0.x before 4.0.18, and ...
69	6 cpe:/a:samba:samba	CVE-2013-4475	4.0	MEDIUM			Samba 3.2.x through 3.6.x before 3.6.20, 4.0.x before ...
70	6 cpe:/a:samba:samba	CVE-2022-32742			4.3	MEDIUM	A flaw was found in Samba. Some SMB1 write requests ...
71	6 cpe:/a:samba:samba	CVE-2022-2031			8.8	HIGH	A flaw was found in Samba. The security vulnerability ...
72	6 cpe:/a:samba:samba	CVE-2022-32746			5.4	MEDIUM	A flaw was found in the Samba AD LDAP server. The AD ...
73	6 cpe:/a:samba:samba	CVE-2015-7540	5.0	MEDIUM	7.5	HIGH	The LDAP server in the AD domain controller in Samba 4...
74	6 cpe:/a:samba:samba	CVE-2017-15275	5.0	MEDIUM	7.5	HIGH	Samba before 4.7.3 might allow remote attackers to ...
75	6 cpe:/a:samba:samba	CVE-2017-12150	5.8	MEDIUM	7.4	HIGH	It was found that samba before 4.4.16, 4.5.x before ...

Figure 25: Table CVEScan

In this chapter and the next one, the concept of connection between tables will be greatly used, as such, it's important to understand how it exactly works.

The main point of the display of information on the webpage or the extraction of data from the database passes through a link between the primary or foreign key across two or more tables. The Figure 26 shows the flow of connections between tables. It starts on the very first one, *ScanNumber* with only a primary key (*id*), which is linked to the foreign key (*scan\_id*) of the table *HostDiscovery*. The same table has also a primary key (*host\_id*) which is linked to the foreign key (*host\_id*) from the *ServiceScan* table and the foreign key (*host\_id*) from the *SSLDDiscovery* table.

This last table, is a last branch table, containing no other connections to any other table. The *ServiceScan* table has a primary key (*scan\_id*) which connects to the last table *CVEScan* foreign key (*cpe\_id*).

To better understand how the links are used, an example is shown on Figure 27. The application needs information from certain columns of the table *HostDiscovery*. The first step is to identify the scan in question. The primary key value from the *ScanNumber* table is taken and like this the foreign key of *HostDiscovery* is known.

Similar to the Figure 22, the foreign key column from this table repeats a lot the same number, this means that all those rows belong to one scan. Now having that information, the required data from the other columns can be taken or filtered.

This will also be useful for the webpages, since whenever a scan is seen in more detail, the value that will pass between pages will be the primary or foreign key of the tables. In the beginning the primary key of the table *ScanNumber* will be used to uniquely identify the data from each scan. Now imagining the information of the *ServiceScan* is required, the process starts the same way but then the primary key (*host\_id*) from the table *HostDiscovery* is used depending on which host the user wants to know more information about.

As seen on Figure 22 the primary key has a different value for each host but still belonging to the same foreign key value. Having this primary key of the specific host, the *ServiceScan* foreign key is also known. As seen on the Figure 23, if a host has a variety of services, the foreign key of this table will remain the same because the host is still the same one, and thus the information can be obtained or filtered (Figure 28).

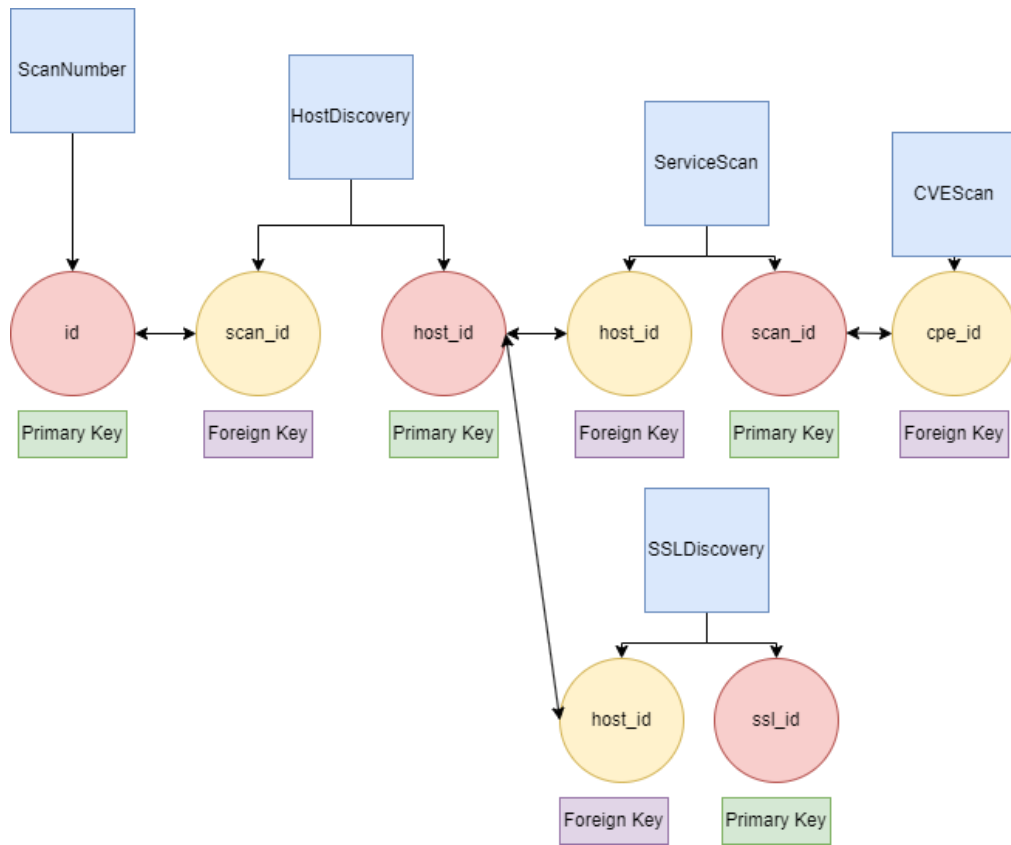


Figure 26: Database links

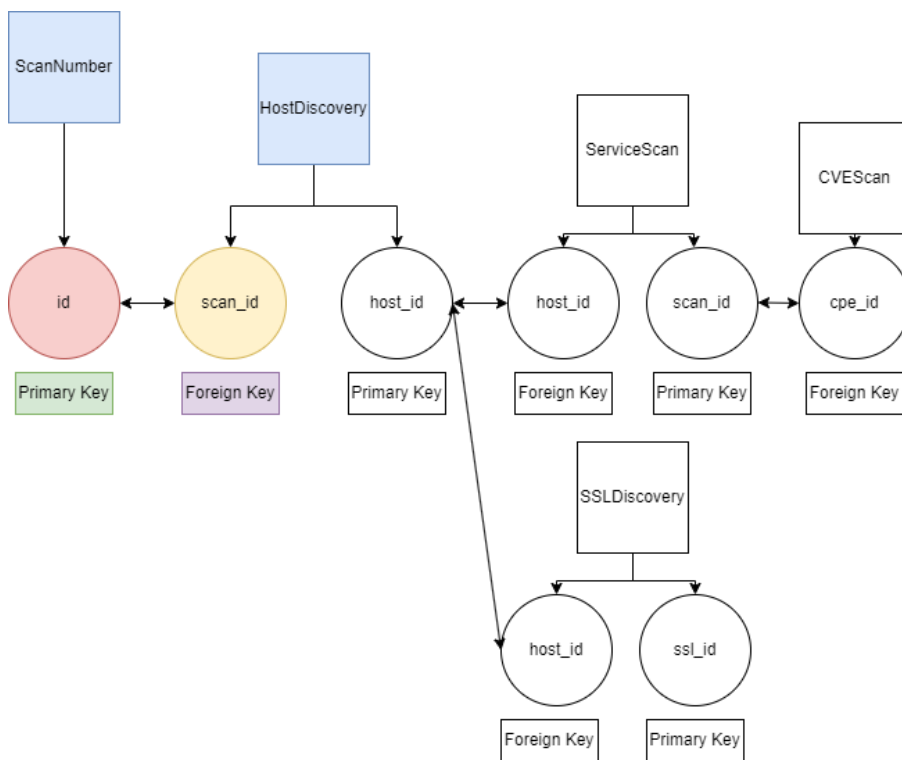


Figure 27: HostDiscovery table example

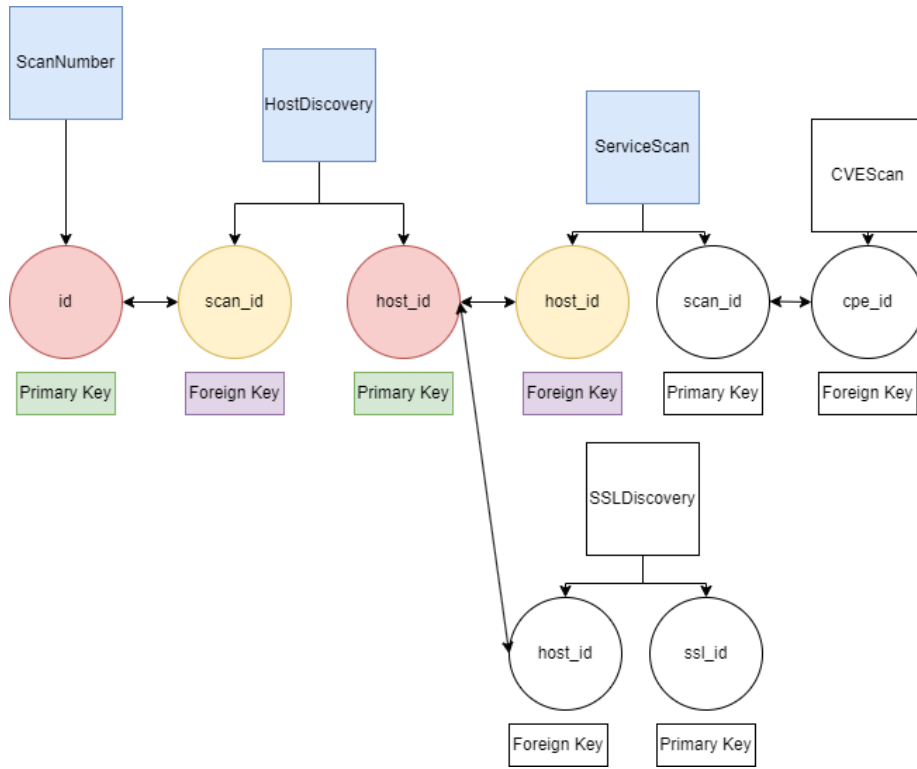


Figure 28: ServiceScan table example

# 5. APPLICATION

The application was created using the micro-framework Flask which allows for the creation of a web application in a straightforward way with the integration of a database (SQLite). The use of Flask demands a python file containing the instance of the Flask class, which will be the WSGI application, in this case the variable *app*. All the necessary libraries are imported for both flask and the database. Along with those, all the python scripts are also imported so they can be called whenever its needed (Listing 19).

To run the application the user simply has to run the command “export FLASK\_APP = r\_scan.py” to validate the main flask script and finally “flask run” to start the application in a localhost environment through the default port of 5000.

```
from flask import Flask, render_template, request, redirect, url_for
import full_scan_nmap, report_creation, basic_scan_nmap, custom_scan_nmap
from flask_sqlalchemy import SQLAlchemy
import sqlite3

app = Flask(__name__)

def get_db_connection():
    conn = sqlite3.connect('FullScan_14.db')
    conn.row_factory = sqlite3.Row
    return conn
```

Listing 19: Flask initiating

A good way to explain the way something works and is structured, is with an image of the overall process, as such, the Figure 29 illustrates the whole process this application takes to display to the user the vulnerabilities detected.

It starts by requesting the user a name of the scan and the IP range (in case of a custom scan, the port range is also requested). Submitting the arguments, the program will check whether a database exists, if the answer is negative, then the program creates a new database. On the other hand, if the answer is affirmative, the scan settings (meaning the starting date and the scan name) will be added to the database on the first table (*ScanNumber*). The next step is the host scan which will be executed through Nmap and generate an xml file. This file will have the important information extracted by the algorithm and added to the second table of the database (*HostDiscovery*).

Following the program, after the hosts have been discovered, the ports are now scanned to find any open port through Nmap commands. An xml file is generated but this time, it will not be automatically added to the database. The same situation happens with the OS discovery, using Nmap commands and generating an xml file.

With all this information, it's time to use the previously created xml files and extract the information regarding the existing service on the open ports. During this process, the OS xml is also gathered and both information's are used by the algorithm to store on the third table of the database (*ServiceScan*).

The service found generated a cpe string which is then used for finding vulnerabilities, this detection is done through two APIs, the NVD API and the VulDB API. With each vulnerability found on every specific host, the information is stored on the fourth table (*CVEScan*) on the database. In the case of no vulnerabilities have been found, the algorithm simply passed to the next host discovered and no data will be added to the table about the previous host.

With this stage completed, it's time to update some tables of the database. The first update is related with the total number of vulnerabilities found on each host and stored on the *HostDiscovery* table. The second update is the total global number of vulnerabilities found during this scan, stored on the table *ScanNumber*.

Following this, the final scan is executed, associated with the SSL/TLS discovery, using Nmap commands, an xml file is generated, and the information stored in the table *SSLDiscovery*. The final update is executed, which adds the ending time of the vulnerability scan to the table *ScanNumber*.

Finally, all the information is updated on the webpage and displayed to the user.

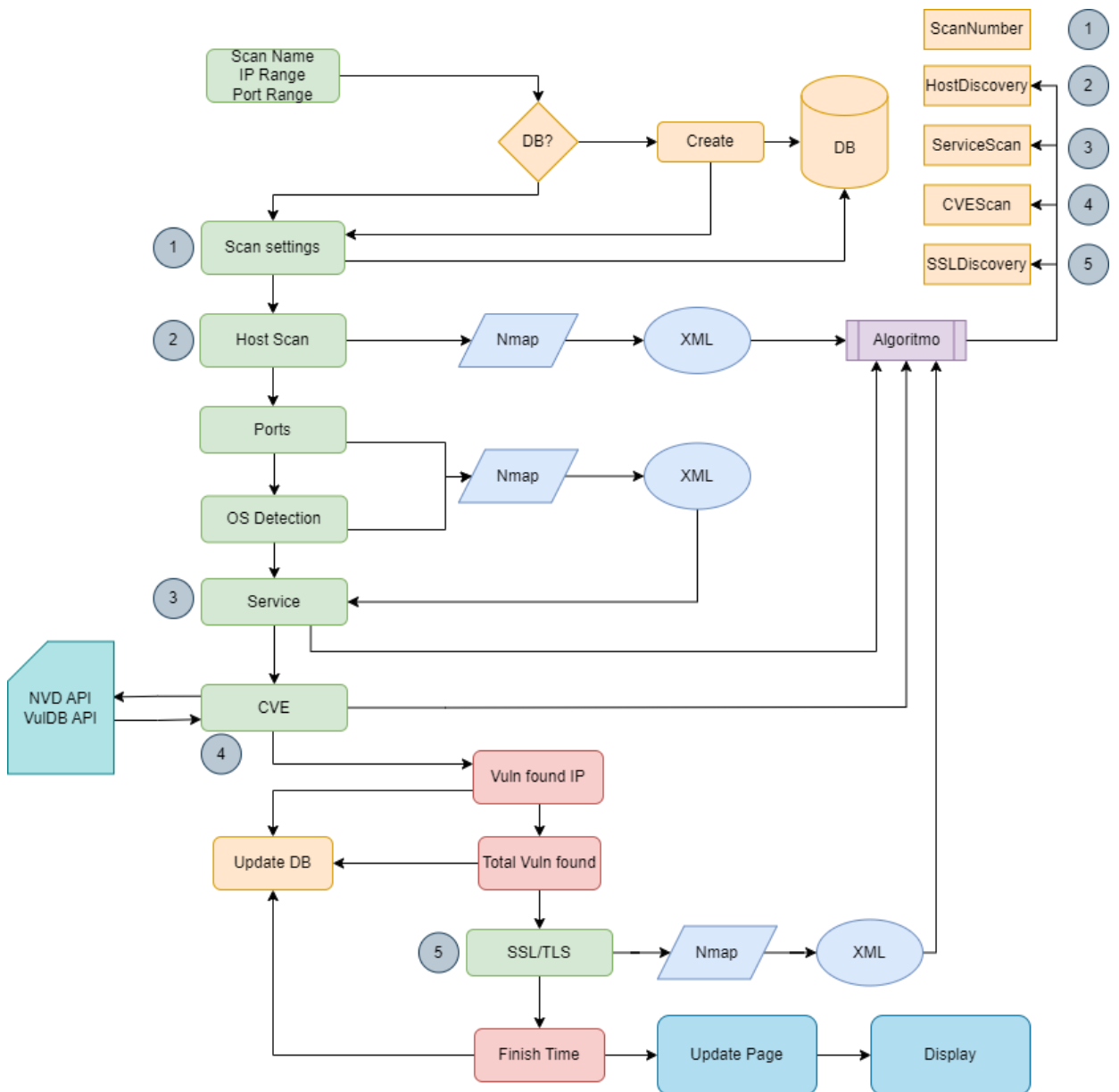


Figure 29: Diagrama da aplicação

## 5.1. DASHBOARD

The first page that will be seen by the user is the Dashboard page, containing information about the total number of scans executed, all the different hosts found, all the unique open ports found and the total number of vulnerabilities in all the scans.

There are three graphics that illustrate the percentage of each severity level of all the vulnerabilities found, the percentage of the operating systems found and finally the number of vulnerabilities found by each date containing scans.

The Dashboard itself is filtered by the date range input by the user. As default it shows the range of dates containing all the executed scans, but it can be changed to show only certain dates or even the scans done in a single day.

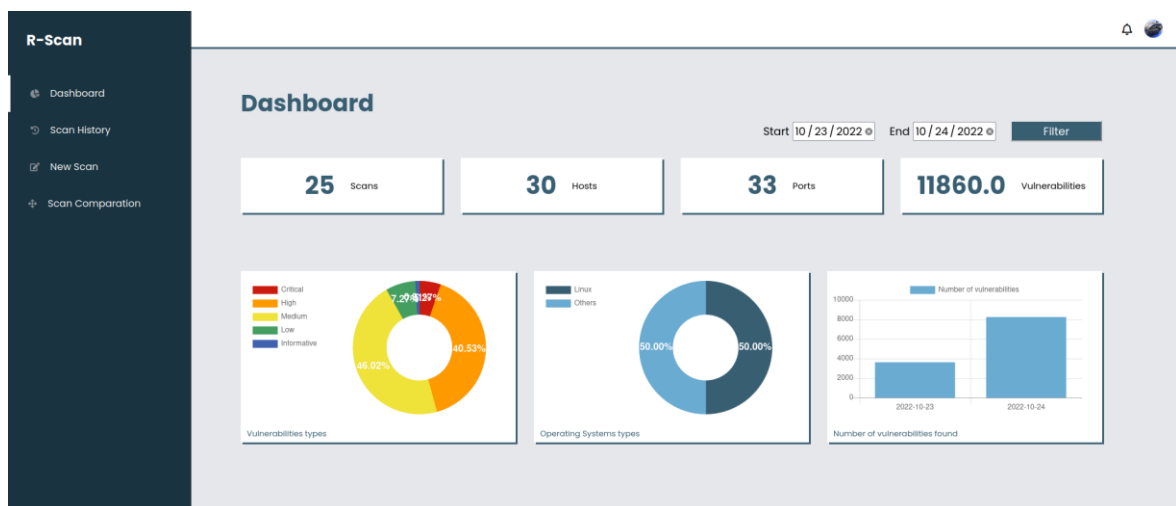


Figure 30: Dashboard

## 5.2. SCAN

For the main goal of this application, the user can and should start a new scan from the left side bar, as illustrated on the Figure 31. This will present 3 options, the first one is a basic scan that will find every available host on the network and the top 1000 open ports, not specifically the ports from 1 to 1000 but rather, the 1000 most known ports to be running in a machine.



The second is a full scan that will have the same characteristics as the previous one but this time it will attempt to check which open ports from the possible 65,535, allowing for a more complete scan but with the downside of taking a longer time to complete. Finally, the last option is the custom scan, again with the same characteristics as the previous ones but the port scan can be defined by the user using a range of ports to be checked.

After the user chooses one of the options, the page will redirect to the `/scanning_create` webpage (Figure 32, Figure 33 and Figure 34). In this page the information regarding the name of the scan and the IP Address range, on both basic and full scan, must be filled before the actual scan starts. On the case of the custom scan, the information about the port range must be also given, Figure 34.

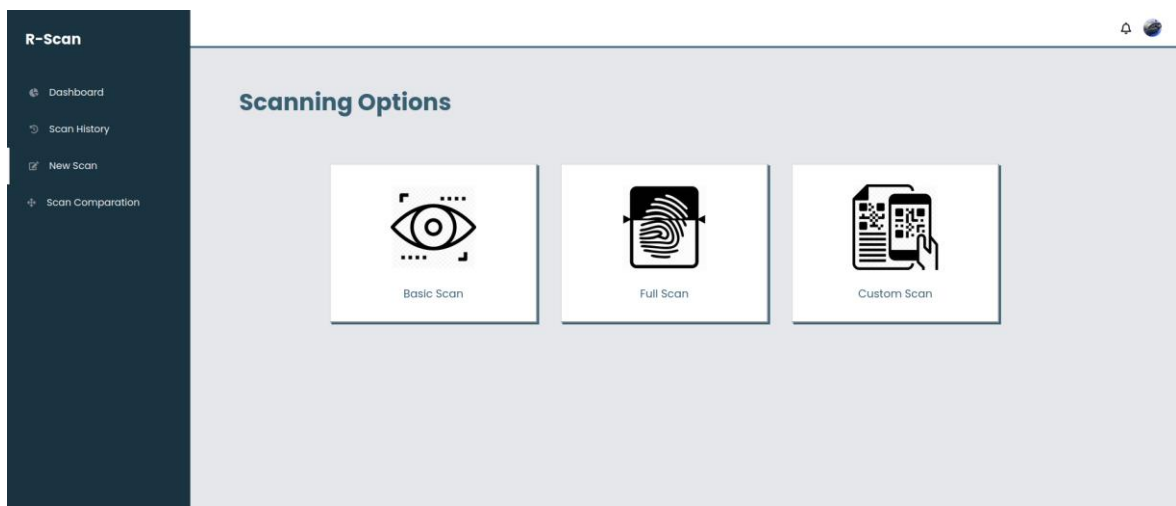


Figure 31: Scan Input options

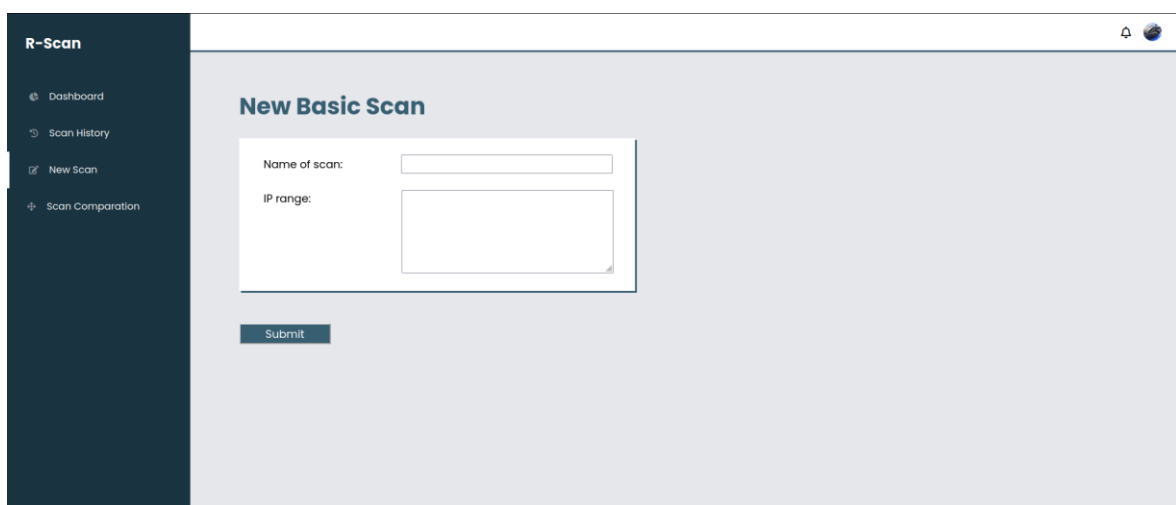


Figure 32: New basic scan

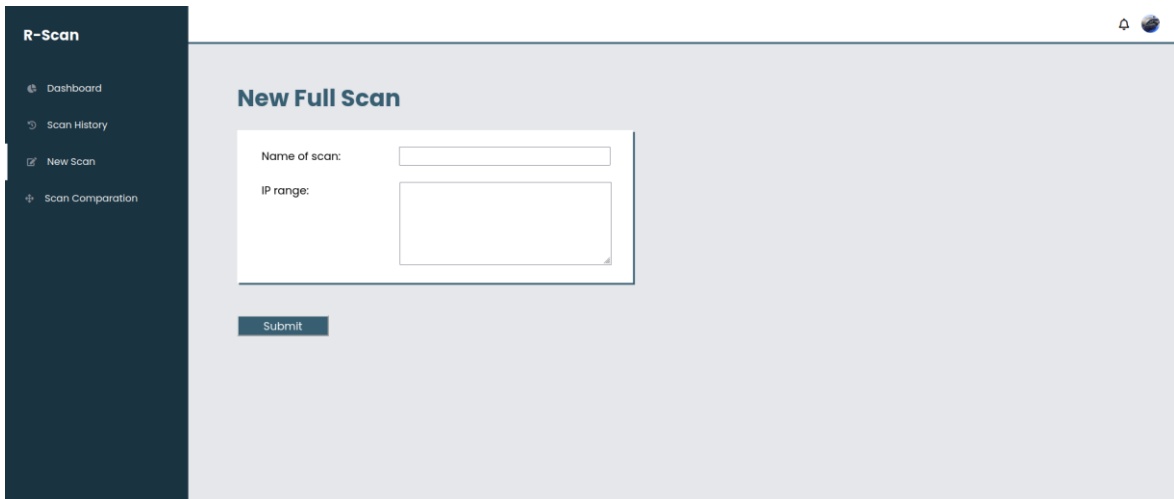


Figure 33: New full scan

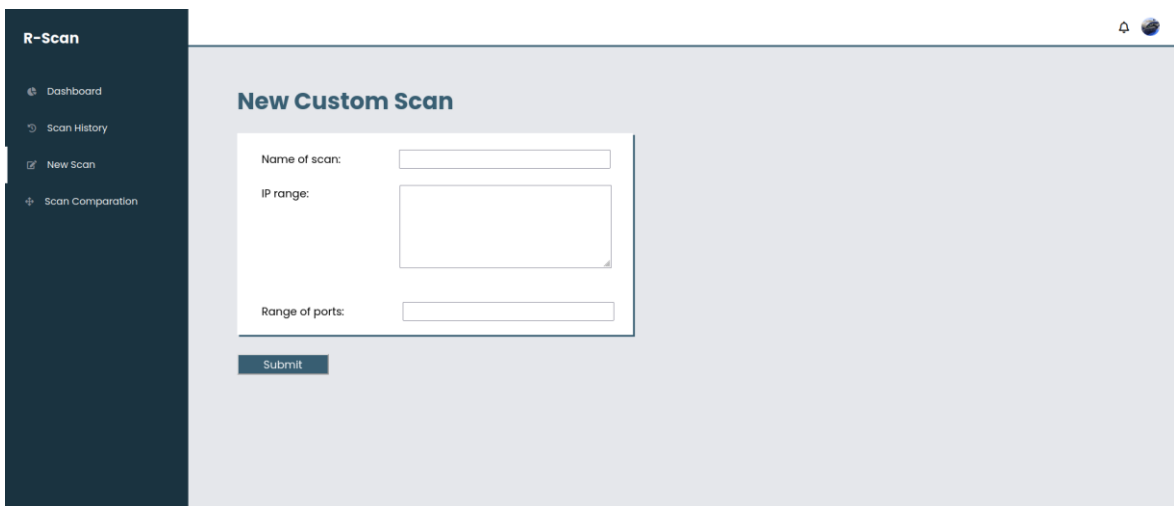


Figure 34: New custom scan

### 5.3. SCAN HISTORY

The scan history allows the user to check all the scans done so far. In this page the user can see the names given to each scan, their start and end time and the total number of vulnerabilities found. There are three actions the user can use, checking the individual details of each scan, giving a more in dept look at them, the creation of a report of the chosen scan and finally deleting the row from the history. The scans are ordered by the newest date to the oldest (Figure 35).

SCAN	START	FINISH	VULNERABILITIES FOUND	Details	Report	Delete
testing_ssl_new	2022-10-23 16:35:13	2022-10-23 16:40:39	250	Details	Report	Delete
testing_ssl_new	2022-10-23 16:27:26	2022-10-23 16:40:39	250	Details	Report	Delete
testing_ssl_new	2022-10-23 16:14:24	2022-10-23 16:40:39	250	Details	Report	Delete
testing_ssl_new	2022-10-23 15:44:26	2022-10-23 16:40:39	250	Details	Report	Delete
testing_ssl_new	2022-10-23 15:35:29	2022-10-23 16:40:39	250	Details	Report	Delete
testing_ssl_new	2022-10-23 15:18:01	2022-10-23 16:40:39	250	Details	Report	Delete
testing_ssl_new	2022-10-23 15:11:42	2022-10-23 16:40:39	250	Details	Report	Delete
testing_ssl_new	2022-10-23 14:52:30	2022-10-23 16:40:39	250	Details	Report	Delete

Figure 35: History of scans

In the main python file, the route `/scanning_menu_history`, is the webpage of the scan history. The function for this route will take into consideration the chosen option of the new scan and in case there is a POST request, meaning, the button was submitted with the name of the scan, the IP range and sometimes the port range, it will save the values there placed and shortly after will be used to run the corresponding script. In the case of no POST request (Listing 20), only the normal access to the page, so there is only the part of the connection to the database and the extraction of all the scans not deleted from the page by date order.

The three possible actions are, the *details* button, from the route `/posting_hosts`, taking the user to the webpage with all the discovered hosts from that scan, taking into account the foreign key from the table *HostDiscovery*, which is the same as the primary key from the table *ScanNumber* (Figure 36). The second option is the report button, from the route `/scanning_menu_history/report`, this page will download a PDF file containing all the information regarding that specific scan, and again using the the primary key from the table *ScanNumber* to gather the information and to be used on the python report script. More in detail on chapter 6.4.

```

@app.route('/scanning_menu_history', methods = ['POST', 'GET'])
def scanning_menu_page():
    if request.method == "POST":
        scan_name = request.form.get("scan_name")
        ip_range = request.form.get("ip_range")
        port_range = request.form.get("nname")
        if request.form['scan_type'] == 'basic_scan':
            print('basic scan')
            basic_scan_nmap.main(scan_name, ip_range)
        elif request.form['scan_type'] == 'full_scan':
            print("full scan")
            full_scan_nmap.main(scan_name,ip_range)
        else:
            print("custom scan")
            custom_scan_nmap.main(scan_name,ip_range, port_range)

    conn = get_db_connection()
    posts = conn.execute("SELECT * FROM ScanNumber WHERE is_hidden = 'False' ORDER BY id desc").fetchall()
    conn.close()
    return render_template("scanning_menu_history.html", posts=posts)

@app.route('/scanning_menu_history/report/<string:ip>', methods = ['POST', 'GET'])
def scanning_menu_page_report(ip):
    report_creation.main(ip)
    conn = get_db_connection()
    posts = conn.execute("SELECT * FROM ScanNumber WHERE is_hidden = 'False' ORDER BY id desc").fetchall()
    conn.close()
    return render_template("scanning_menu_history.html", posts=posts)

@app.route('/scanning_menu_history/delete/<string:ip>', methods = ['POST', 'GET'])
def scanning_menu_page_delete(ip):
    conn = get_db_connection()
    sql = "UPDATE ScanNumber SET is_hidden = 'True' WHERE id = '" + ip + "'"
    cur = conn.cursor()
    cur.execute(sql)
    conn.commit()
    cur.close()

    posts = conn.execute("SELECT * FROM ScanNumber WHERE is_hidden = 'False' ORDER BY id desc").fetchall()
    conn.close()
    return render_template("scanning_menu_history.html", posts=posts)

@app.route('/posting_hosts/<string:ip>')
def post_hosts(ip):
    query = "SELECT * FROM HostDiscovery WHERE scan_id=" + ip
    conn = get_db_connection()
    posts = conn.execute(query).fetchall()
    conn.close()
    return render_template("post_hosts.html", posts=posts)

```

Listing 20: Flask routes

Having clicked on an individual scan, the user can see all the hosts and their individual vulnerabilities found. The button *Details* shows those vulnerabilities and associated information (Figure 36).

IP	STATE	VULNERABILITIES	
10.0.2.1	up	0	<a href="#">Details</a>
10.0.2.2	up	20	<a href="#">Details</a>
10.0.2.3	up	0	<a href="#">Details</a>
10.0.2.4	up	100	<a href="#">Details</a>
10.0.2.8	up	130	<a href="#">Details</a>
10.0.2.7	up	0	<a href="#">Details</a>

Figure 36: Hosts discovered

As an example of some vulnerabilities found, they are ordered by the base score level (10 being the highest). Along with the score level, the severity is also displayed, ranging from Critical to Informative and distinguished by colours, as illustrated on Figure 37, Figure 38 and Figure 39. Each vulnerability has also the description and the corresponding CVE-ID.

SCORE	SEVERITY	DESCRIPTION	NAME
9.8	CRITICAL	A carefully crafted request body can cause a buffer overflow in the mod_lua multipart parser (rparsebody()) called from Lua scripts). The Apache httpd team is not aware of an exploit for the vulnerability though it might be possible to craft one. This issue affects Apache HTTP Server 2.4.51 and earlier.	CVE-2021-44790
9.8	CRITICAL	ap_escape_quotes() may write beyond the end of a buffer when given malicious input. No included modules pass untrusted data to these functions, but third-party / external modules may. This issue affects Apache HTTP Server 2.4.48 and earlier.	CVE-2021-39275
9.8	CRITICAL	Apache HTTP Server 2.4.52 and earlier fails to close inbound connection when errors are encountered discarding the request body, exposing the server to HTTP Request Smuggling	CVE-2022-22720
9.0	CRITICAL	A crafted request uri-path can cause mod_proxy to forward the request to an origin server chosen by the remote user. This issue affects Apache HTTP Server 2.4.48 and earlier.	CVE-2021-40438
8.8	HIGH	Use after free in FedCM in Google Chrome prior to 104.0.5112.101 allowed a remote attacker to potentially exploit heap corruption via a crafted HTML page.	CVE-2022-2852
8.8	HIGH	A flaw was found in the way samba, as an Active Directory Domain Controller, is able to support an RODC (read-only domain controller). This would allow an RODC to print administrator tickets.	CVE-2020-25718
8.8	HIGH	The Samba AD DC includes checks when adding service principals names (SPNs) to an account to ensure that SPNs do not alias with those already in the database. Some of these checks are able to be bypassed if an account modification re-adds an SPN that was previously present on that account, such as one added when a computer is joined to a domain. An attacker who has the ability to write to an account can exploit this to perform a denial-of-service attack by adding an SPN that matches an existing service. Additionally, an attacker who can intercept traffic can impersonate existing services, resulting in a loss of	CVE-2022-0336

Figure 37: Vulnerabilities found - 1

R-Scan					
<ul style="list-style-type: none"> <li>Dashboard</li> <li>Scan History</li> <li>New Scan</li> <li>Scan Comparison</li> </ul>	7.5	HIGH	Samba before 4.7.3 might allow remote attackers to obtain sensitive information by leveraging failure of the server to clear allocated heap memory.	CVE-2017-15275	
	7.4	HIGH	It was found that samba before 4.4.16, 4.5.x before 4.5.14, and 4.6.x before 4.6.8 did not enforce "SMB signing" when certain configuration options were enabled. A remote attacker could launch a man-in-the-middle attack and retrieve information in plain-text.	CVE-2017-12150	
	7.2	HIGH	A flaw was found in the way Samba, as an Active Directory Domain Controller, implemented Kerberos name-based authentication. The Samba AD DC could become confused about the user a ticket represents if it did not strictly require a Kerberos PAC and always use the SIDs found within. The result could include total domain compromise.	CVE-2020-25719	
	7.1	HIGH	A vulnerability has been found in Linux Kernel and classified as problematic. This vulnerability affects the function inet6_stream_ops/inet6_dgram_ops of the component IPv6 Handler. The manipulation leads to race condition. It is recommended to apply a patch to fix this issue. VDB-211090 is the identifier assigned to this vulnerability.	CVE-2022-3567	
	7.1	HIGH	A vulnerability, which was classified as problematic, was found in Linux Kernel. This affects the function tcp_getsockopt/tcp_setsockopt of the component TCP Handler. The manipulation leads to race condition. It is recommended to apply a patch to fix this issue. The identifier VDB-211089 was assigned to this vulnerability.	CVE-2022-3566	
	6.9	MEDIUM	envvars (aka envvars-std) in the Apache HTTP Server before 2.4.2 places a zero-length directory name in the LD_LIBRARY_PATH, which allows local users to gain privileges via a Trojan horse DSO in the current working directory during execution of apachectl.	CVE-2012-0883	
	6.8	MEDIUM	Race condition in the mod_status module in the Apache HTTP Server before 2.4.10 allows remote attackers to cause a denial of service (heap-based buffer overflow), or possibly obtain sensitive credential information or execute arbitrary code, via a crafted request that triggers improper scoreboard handling within the status_handler function in modules/generators/mod_status.c and the lua_ap_scoreboard_worker function in modules/lua/lua_request.c.	CVE-2014-0226	
	6.5	MEDIUM	The remote server on port 443 offers deprecated TLS 1.0 protocol which can lead to weaknesses. Consider upgrading to TLSv2.0 or above.	SSL/TLS	
	6.3	MEDIUM	NVIDIA DCGM contains a vulnerability in nvhostengine, where a network user can cause detection of error conditions without action, which may lead to limited code execution, some denial of service, escalation of privileges, and limited impacts to both data confidentiality and integrity.	CVE-2022-21820	

Figure 38: Vulnerabilities found - 2

R-Scan					
<ul style="list-style-type: none"> <li>Dashboard</li> <li>Scan History</li> <li>New Scan</li> <li>Scan Comparison</li> </ul>			The client cannot control the area of the server memory written to the file (or printer).		
	4.0	MEDIUM	smbd in Samba 3.0 before 3.0.37, 3.2 before 3.2.15, 3.3 before 3.3.8, and 3.4 before 3.4.2 allows remote authenticated users to cause a denial of service (infinite loop) via an unanticipated oplock break notification reply packet.	CVE-2009-2906	
	4.0	MEDIUM	Samba 3.2.x through 3.6.x before 3.6.20, 4.0.x before 4.0.11, and 4.1.x before 4.11, when vfs_streams_depot or vfs_streams_xattr is enabled, allows remote attackers to bypass intended file restrictions by leveraging ACL differences between a file and an associated alternate data stream (ADS).	CVE-2013-4475	
	3.7	LOW	An attacker who is able to send and receive messages to an authoritative DNS server and who has knowledge of a valid TSIG key name may be able to circumvent TSIG authentication of AXFR requests via a carefully constructed request packet. A server that relies solely on TSIG keys for protection with no other ACL protection could be manipulated into providing an AXFR of a zone to an unauthorized recipient or accepting bogus NOTIFY packets. Affects BIND 9.4.0-9.8.8, 9.9.0-9.9.10-PI, 9.10.0-9.10.5-PI, 9.11.0-9.11-PI, 9.9.3-SI-9.9.10-S2, 9.10.5-SI-9.10.5-S2.	CVE-2017-3142	
	3.6	LOW	The winbind_name_list_to_sid_string_list function in nsswitch/pam_winbind.c in Samba through 4.12 handles invalid require_membership_of group names by accepting authentication by any user, which allows remote authenticated users to bypass intended access restrictions in opportunistic circumstances by leveraging an administrator's pam_winbind configuration-file mistake.	CVE-2012-6150	
	3.5	LOW	Samba 3.6.6 through 3.6.23, 4.0.x before 4.0.16, and 4.1.x before 4.1.8, when a certain vfs shadow copy configuration is enabled, does not properly initialize the SRV_SNAPSHOT_ARRAY response field, which allows remote authenticated users to obtain potentially sensitive information from process memory via a (1) FSCTL_GET_SHADOW_COPY_DATA or (2) FSCTL_SRV_ENUMERATE_SNAPSHOTS request.	CVE-2014-0178	
	2.6	LOW	CRLF injection vulnerability in the mod_negotiation module in the Apache HTTP Server 2.2.6 and earlier in the 2.2.x series, 2.0.61 and earlier in the 2.0.x series, and 1.3.39 and earlier in the 1.3.x series allows remote authenticated users to inject arbitrary HTTP headers and conduct HTTP response splitting attacks by uploading a file with a multi-line name containing HTTP header sequences and a file extension, which leads to injection within a (1) "406 Not Acceptable" or (2) "300 Multiple Choices" HTTP response when the extension is omitted in a request for the file.	CVE-2008-0456	
	2.6	LOW	The ap_proxy_ftp_handler function in modules/proxy/proxy_ftp.c in the mod_proxy_ftp module in the Apache HTTP Server 2.0.63 and 2.2.13 allows remote FTP servers to cause a denial of service (NULL pointer dereference and child process crash) via a malformed reply to an EPSV command.	CVE-2009-3094	

Figure 39: Vulnerabilities found - 3

Regarding the Informative rows, each serve a purpose of informing the user of the following:

- All the services found running on that host, Figure 41
- All the open ports found on that host, Figure 42
- The most likely operating system of that host Figure 43
- The SSL/TLS found on that host Figure 44

CVE	Severity	Description	Details
CVE-2009-2906	MEDIUM	smdb in Samba 3.0 before 3.0.37, 3.2 before 3.2.15, 3.3 before 3.3.8, and 3.4 before 3.4.2 allows remote authenticated users to cause a denial of service (infinite loop) via an unanticipated oplock break notification reply packet.	
CVE-2013-4475	MEDIUM	Samba 3.2.x through 3.6.x before 3.6.20, 4.0.x before 4.0.11, and 4.1.x before 4.1.1, when vfs_streams_depot or vfs_streams_xattr is enabled, allows remote attackers to bypass intended file restrictions by leveraging ACL differences between a file and an associated alternate data stream (ADS).	
CVE-2012-6150	LOW	The winbind_name_list_to_sid_string_list function in nsswitch/pam_winbind.c in Samba through 4.1.2 handles invalid require_membership_of group names by accepting authentication by any user, which allows remote authenticated users to bypass intended access restrictions in opportunistic circumstances by leveraging an administrator's pam_winbind configuration-file mistake.	
CVE-2014-0178	LOW	Samba 3.6.6 through 3.6.23, 4.0.x before 4.0.18, and 4.1.x before 4.1.8, when a certain vfs shadow copy configuration is enabled, does not properly initialize the SRV_SNAPSHOT_ARRAY response field, which allows remote authenticated users to obtain potentially sensitive information from process memory via a (1) FSCTL_GET_SHADOW_COPY_DATA or (2) FSCTL_SRV_ENUMERATE_SNAPSHOTS request.	
CVE-2021-43566	LOW	All versions of Samba prior to 4.13.16 are vulnerable to a malicious client using an SMB1 or NFS race to allow a directory to be created in an area of the server file system not exported under the share definition. Note that SMB1 has to be enabled, or the share also available via NFS in order for this attack to succeed.	
0.0	INFORMATIVE	Services Found	<a href="#">Details</a>
0.0	INFORMATIVE	Ports Found	<a href="#">Details</a>
0.0	INFORMATIVE	OS Detected	<a href="#">Details</a>
0.0	INFORMATIVE	SSL/TLS Detected	<a href="#">Details</a>

Figure 40: Informative content

IP	PORT	SERVICE	SERVICE NAME	SERVICE VERSION
10.0.2.8		ftp	vsftpd	2.3.4
10.0.2.8		ssh	OpenSSH	4.7pl Debian 8ubuntu1
10.0.2.8		telnet	Linux telnetd	
10.0.2.8		smtp	Postfix smtpd	
10.0.2.8		domain	ISC BIND	9.4.2
10.0.2.8		http	Apache httpd	2.2.8
10.0.2.8		rpcbind		
10.0.2.8		netbios-ssn	Samba smbd	3.X - 4.X

Figure 41: Informative Services page

IP	PORT
10.0.2.8	21
10.0.2.8	22
10.0.2.8	23
10.0.2.8	25
10.0.2.8	53
10.0.2.8	80
10.0.2.8	111
10.0.2.8	139
10.0.2.8	445

Figure 42: Informative Ports page

IP	OPERATING SYSTEM
10.0.2.8	Linux 2.6.9 - 2.6.33

Figure 43: Informative Operating Systems page

IP	PORT	PROTOCOL	NAME	ISSUER	CERT START	CERT END
10.0.2.8	25	TLSv1.0	ubuntu804-base.localdomain	ubuntu804-base.localdomain	2010-03-17	2010-04-16

Figure 44: Informative SSL/TLS page



Every route for the informative pages is very similar, containing the */informative* followed by the actual page purpose and finally the IP which corresponds to the primary key from the table *HostDiscovery* or the linked foreign key from the tables *ServiceScan* and *SSLDDiscovery*. This allows for selecting specific parts of those tables. Each function seen on the Listing 21 returns the associated webpage.

```
@app.route('/informative/Ports/<string:ip>')
def post_port(ip):
    print(ip)
    query = "SELECT * FROM ServiceScan WHERE host_id=" + ip
    conn = get_db_connection()
    post_port = conn.execute(query).fetchall()
    conn.close()
    return render_template('post_port.html', post_port=post_port)

@app.route('/informative/OS/<string:ip>')
def post_os(ip):
    print(ip)
    query = "SELECT DISTINCT ip,os FROM ServiceScan WHERE host_id=" + ip
    conn = get_db_connection()
    post_os = conn.execute(query).fetchall()
    conn.close()
    return render_template('post_os.html', post_os=post_os)

@app.route('/informative/TLS_SSL/<string:ip>')
def post_tls_ssl(ip):
    print(ip)
    query = "SELECT * FROM SSLDiscovery WHERE host_id=" + ip
    conn = get_db_connection()
    post_tls_ssl = conn.execute(query).fetchall()
    conn.close()
    return render_template('post_tls_ssl.html', post_tls_ssl=post_tls_ssl)

@app.route('/informative/Services/<string:ip>')
def post_service(ip):
    query = "SELECT DISTINCT ip,service_name,service_product,service_version FROM ServiceScan WHERE host_id="+ ip
    conn = get_db_connection()
    post_service = conn.execute(query).fetchall()
    conn.close()
    return render_template('post_service.html', post_service=post_service)
```

Listing 21: Informative webpages – code

## 5.4. REPORT

For each individual scan, a PDF report can be generated with all the relevant information. This conversion to PDF is achieved through the class FPDF, allowing a variety of customizable options for the creation of a PDF report. The python script *report\_creating()* is responsible for this conversion, in which the main function connects to the database in order to extract all the important information and stored on proper variables. Afterwards, an instance of the class is created and used to build the structure of the PDF, such as, the size of the page, font, looks and tables. The information is then added to the corresponding cell of the table, which sometimes requires a *for loop* for a bigger table with different rows. The Listing 22 demonstrates the code used.

```
def main(ip):
    db_file = 'FullScan_11.db'
    conn = create_connection(db_file)

    host_discovery = get_host_discovery(conn, ip)
    scan_data = get_scan_data(conn, ip)
    vuln_found = [row[0] for row in scan_data]
    date_start = [row[1] for row in scan_data]
    date_finish = [row[2] for row in scan_data]
    scan = [row[3] for row in scan_data]
    service = get_service(conn, ip)
    vuln_count = get_vulnerability_count(conn, ip)
    severities = get_severity(conn, ip)

    #Creating pdf
    pdf = FPDF()
    pdf.add_page()
    pdf.set_font("Arial", size=15)
    pdf.rect(5.0, 5.0, 200.0, 287.0)
    datetime = datetime.now().strftime("%d/%m/%Y %H:%M:%S")

    pdf.set_font("Arial", size=12)
    pdf.cell(190, 10, txt=datetime, ln=1, align='R')
    pdf.set_font("Arial", 'B', size=16)
    pdf.cell(200, 10, txt="Scan Report: " + scan[0], ln=1, align='C')
    pdf.set_font("Arial", size=12)

    pdf.cell(200, 10, txt="Start Date: " + str(date_start[0]), ln=1, align='L')
    pdf.cell(200, 10, txt="Finish Date: " + str(date_finish[0]), ln=3, align='L')
    pdf.cell(200, 10, txt="", ln=1, align='L')

    pdf.set_font("Arial", size=12)
    pdf.cell(200, 10, txt="Scan Results", ln=1, align='L')
    epw = pdf.w - 2 * pdf.l_margin
    col_width = epw / 4
    th = pdf.font_size
    name_scan_results = [('IP range', 'Hosts', 'Total Vulnerabilities')]
    name_scan_results_2 = [( '0', vuln count, vuln found[0])]
    name_scan_results = name_scan_results + name_scan_results_2
    for row in name_scan_results:
        for datum in row:
            pdf.cell(col_width, 2 * th, str(datum), border=1)

    pdf.ln(2 * th)
    pdf.rect(5.0, 5.0, 200.0, 287.0)

    pdf.set_font("Arial", size=16)
    pdf.cell(200, 10, txt="", ln=1, align='L')
    pdf.cell(200, 10, txt="Level of Vulnerabilities", ln=1, align='L')
    pdf.set_font("Arial", size=12)
```

Listing 22: Scan report code

The output of the report of the scan can be divided into a few sections. The first section demonstrates the date that the report was generated, the name of the scan itself and the beginning and ending of the scan. The second section shows the general result of the scan, represented by the IP Address range of the scan, the total number of hosts found and the total number of vulnerabilities. The third section represents the level of vulnerabilities found, in which the different levels of severity display their vulnerabilities. The fourth section demonstrates the hosts discovered and their individual vulnerability numbers. Finally, the fifth section shows the OS, ports and service name of each host discovered, as illustrated on the Figure 45 and Figure 46.

23/10/2022 16:46:31		
<b>Scan Report: testing_ssl_new</b>		
Start Date: 2022-10-23 16:35:13		
Finish Date: 2022-10-23 16:40:39		
Scan Results		
IP range	Hosts	Total Vulnerabilities
10.0.2.0/24	6	250
Level of Vulnerabilities		
Severity	Number	
Critical	14	
High	104	
Medium	118	
Low	17	
Informative	4	
Host Discovered		
Hosts	Vulnerabilities	
10.0.2.1	0	
10.0.2.2	20	
10.0.2.3	0	
10.0.2.4	100	
10.0.2.8	130	
10.0.2.7	0	

Figure 45: Scan report - 1

### Service data

Hosts	OS	Ports	Service Name
10.0.2.1	2N Helios IP VoIP doorbell	53	tcpwrapped

10.0.2.2	2N Helios IP VoIP doorbell	135	msrpc
10.0.2.2	2N Helios IP VoIP doorbell	445	microsoft-ds
10.0.2.4	Linux 2.6.17 - 2.6.36	22	ssh
10.0.2.4	Linux 2.6.17 - 2.6.36	80	http
10.0.2.4	Linux 2.6.17 - 2.6.36	139	netbios-ssn
10.0.2.4	Linux 2.6.17 - 2.6.36	143	imap
10.0.2.4	Linux 2.6.17 - 2.6.36	443	http
10.0.2.4	Linux 2.6.17 - 2.6.36	445	netbios-ssn
10.0.2.8	Linux 2.6.9 - 2.6.33	21	ftp
10.0.2.8	Linux 2.6.9 - 2.6.33	22	ssh
10.0.2.8	Linux 2.6.9 - 2.6.33	23	telnet
10.0.2.8	Linux 2.6.9 - 2.6.33	25	smtp
10.0.2.8	Linux 2.6.9 - 2.6.33	53	domain
10.0.2.8	Linux 2.6.9 - 2.6.33	80	http
10.0.2.8	Linux 2.6.9 - 2.6.33	111	rpcbind
10.0.2.8	Linux 2.6.9 - 2.6.33	139	netbios-ssn
10.0.2.8	Linux 2.6.9 - 2.6.33	445	netbios-ssn

Figure 46: Scan report - 2

## 5.5. SCAN COMPARATION

The last feature of the side bar is the scan comparison. It takes two previously done scans stored on the database and compares the hosts found. Depending on the hosts, the follow up information may or may not be displayed. In the example on the Figure 48, the hosts that are on both scans can be seen on the “Compared Hosts”, while the hosts on the oldest and not found on the new scan can be seen on the “Old Hosts” and, the hosts found on the new scan and not on the oldest, can be seen on the “New Hosts”.

The Figure 49 show the vulnerabilities that were corrected, by the CVE-ID and the Description. This means that a vulnerability was found on a specific host on the oldest scan that was not found on the newest scan in the same host, meaning most likely a correction/patch was made. In the case there was no corrections, the host simply shows the corresponding IP Address without any further information. On the Figure 50, new vulnerabilities are shown, meaning, vulnerabilities found on a specific host on the newest scan that were not present in the same host on the oldest scan. This comparison was made possible by using SQLite EXCEPT statement that filters the records based on the intersection of two SELECT statements, as well as the INTERSECT operator that returns the result if two or more datasets intersect from the SELECT statement.

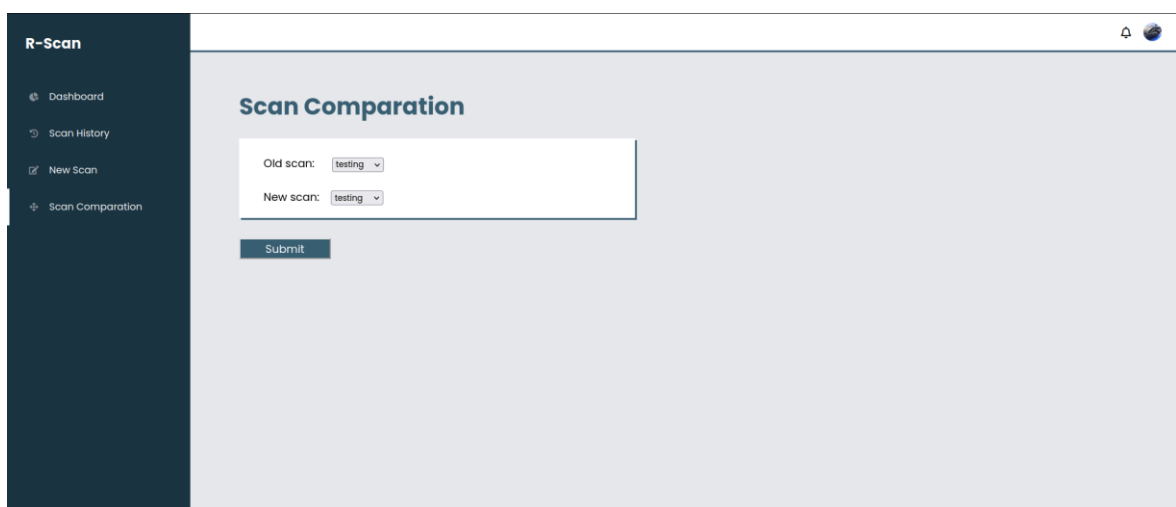


Figure 47: Scan comparison

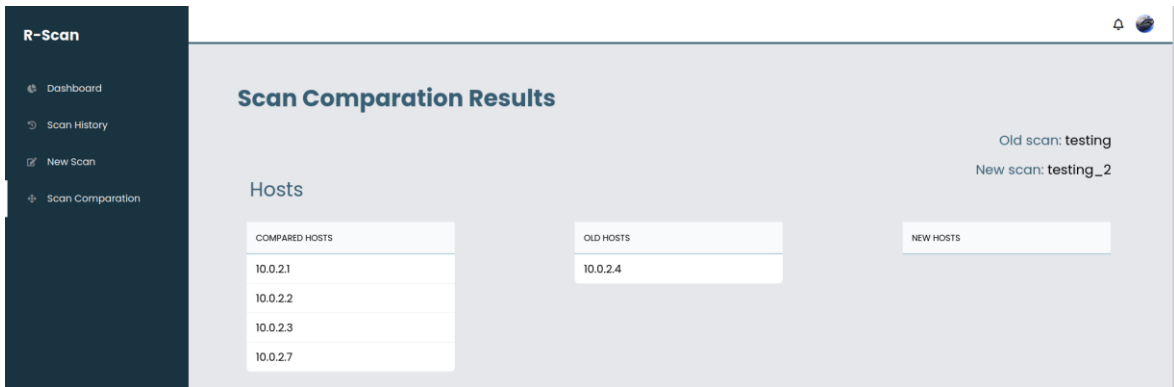


Figure 48: Scan comparison results

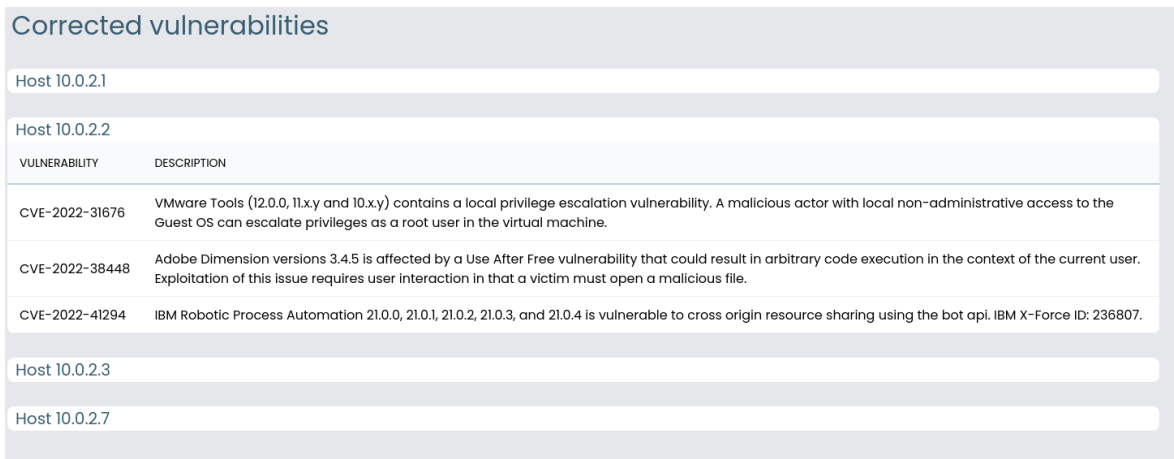


Figure 49: Corrected vulnerabilities

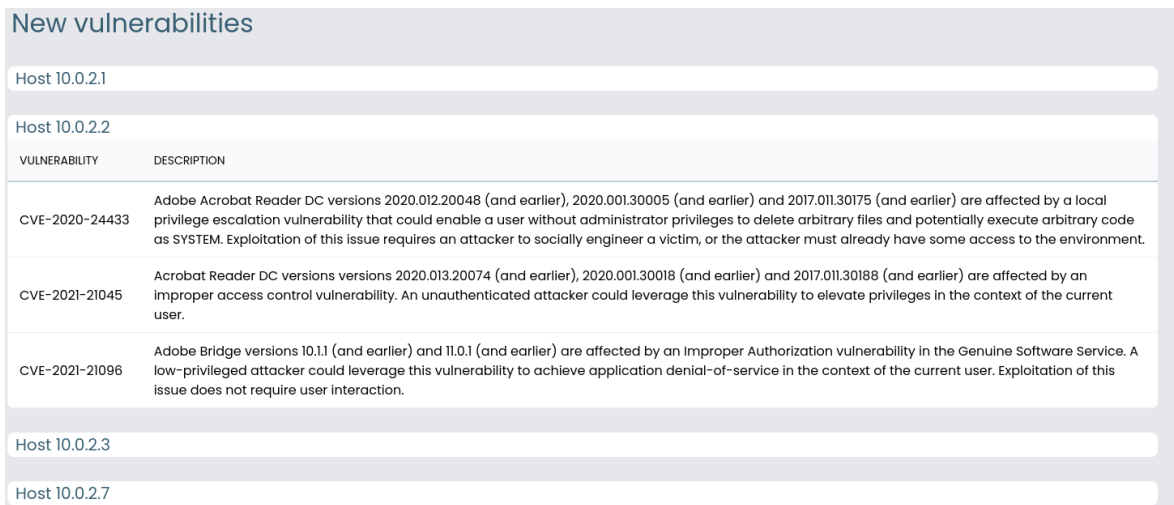


Figure 50: New vulnerabilities

# 6. TESTS AND RESULTS

The tests to simulate the virtual network and the execution of the application were carried out on a laptop with the following characteristics:

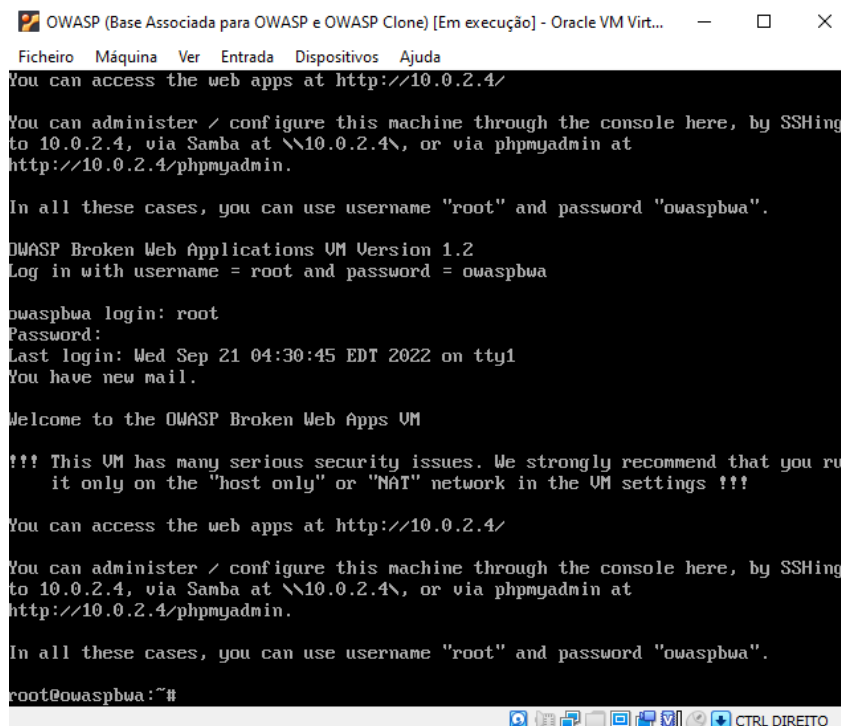
Table 13: Computer specification

NAME	LENOVO LEGION 5
RAM	16.0 GB
CPU	AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz
GPU	NVIDIA GeForce RTX 3060 Laptop GPU

## 6.1. PHASE ONE

For the initial experiments, a few virtual machines were used in VirtualBox for the development of the application and the corresponding testing of vulnerabilities found. The main machine uses the Kali Linux as operating system, which allows for a fast and robust performance while at the same time, holding a lot of the tools and libraries that were needed during the tests.

For the vulnerable machines, three were picked containing different environments and services. The first one OWASP Broken Web Applications Project, which is a collection of vulnerable web applications on a virtual machine [53]. This machine containing different vulnerable services, allow for a complete scan and a good testing ground for this application. It does not have the typical graphical interface, only a command line interface, preserving resources as it only needs a small amount of RAM and CPU usage to run properly, as seen on Figure 51 The second machine is Metasploitable 2, a virtual test environment to practice penetration testing and security research. As the previous machine, there is only the command line interface, creating a lightweight virtual environment full of vulnerabilities to be detected and tested (Figure 52) [54].



```
OWASP (Base Associada para OWASP e OWASP Clone) [Em execução] - Oracle VM Virt...
Ficheiro Máquina Ver Entrada Dispositivos Ajuda
You can access the web apps at http://10.0.2.4/

You can administer / configure this machine through the console here, by SSHing
to 10.0.2.4, via Samba at \\10.0.2.4\, or via phpmyadmin at
http://10.0.2.4/phpmyadmin.

In all these cases, you can use username "root" and password "owaspbwa".

OWASP Broken Web Applications VM Version 1.2
Log in with username = root and password = owaspbwa

owaspbwa login: root
Password:
Last login: Wed Sep 21 04:30:45 EDT 2022 on tty1
You have new mail.

Welcome to the OWASP Broken Web Apps VM

!!! This VM has many serious security issues. We strongly recommend that you run
it only on the "host only" or "NAT" network in the VM settings !!!

You can access the web apps at http://10.0.2.4/

You can administer / configure this machine through the console here, by SSHing
to 10.0.2.4, via Samba at \\10.0.2.4\, or via phpmyadmin at
http://10.0.2.4/phpmyadmin.

In all these cases, you can use username "root" and password "owaspbwa".

root@owaspbwa:~#
```

Figure 51: OWASP virtual machine



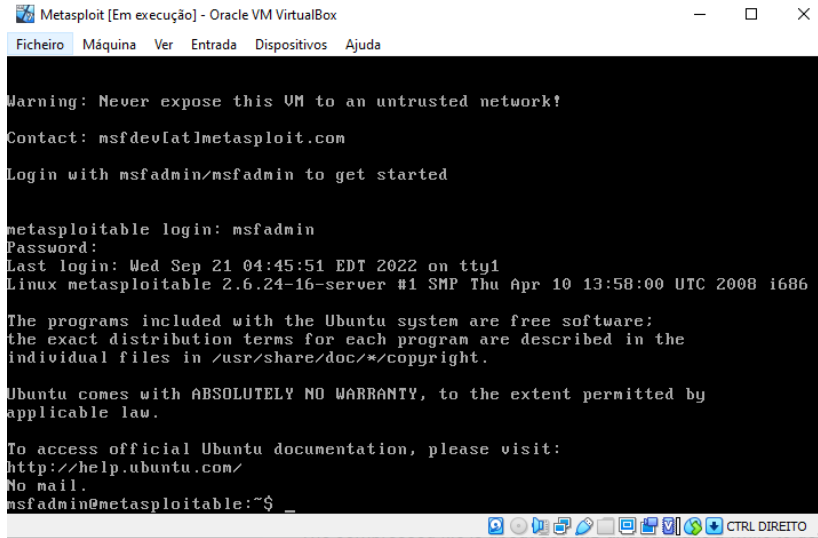


Figure 52: Metasploit virtual machine

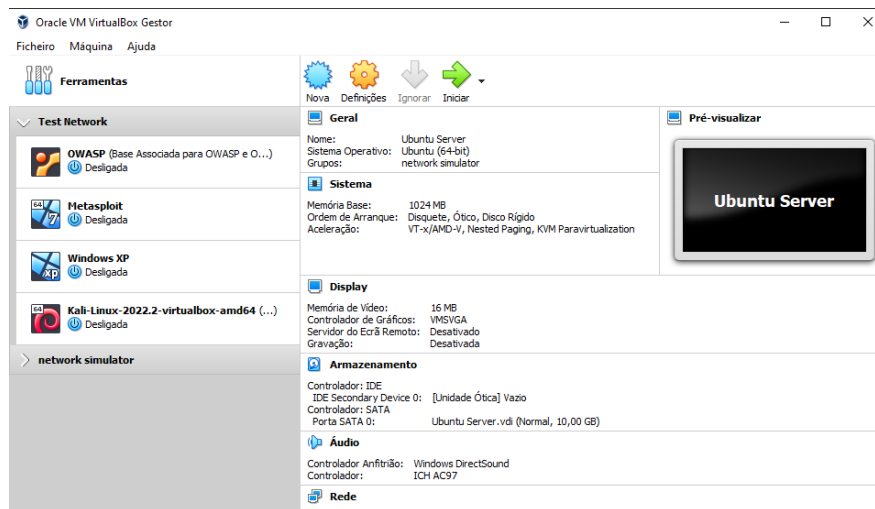


Figure 53: Virtual Machine setup

All the virtual machines can communicate between each other through a NAT Network, as stated on the State of the Art, this mode allows for multiple virtual machines to communicate between each other via the network. The virtual machines can also access other hosts in the physical network and external networks including the internet. However, any machine from an external network as well as those from the physical network to which the host machine is connected, are not allowed to access the virtual machine.

Using this method, the main machine (Kali Linux) takes the IP Address 10.0.2.7 while the OWASP and Metasploit take the IP Address, 10.0.2.4 and 10.0.2.5, respectively. This explains some of the figures seen in chapter 6 showing the application and some of the hosts found.

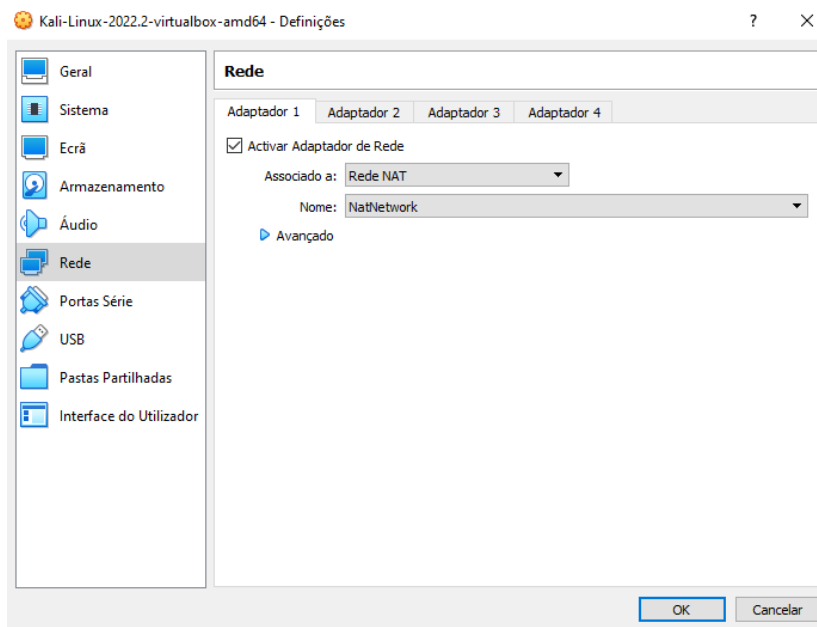


Figure 54: VirtualBox Nat Network

## 6.2. PHASE TWO

With the whole application working well and the correct showing and scanning of the vulnerabilities, it was time to pass to the next phase of experiments, the creation of the network. The idea behind the way to simulate a virtual network through VirtualBox was explored by Brian Linkletter on his blog [55]. A small network was first simulated, containing three routers connected between them and a computer connected to each individual router (Figure 55). All the routers contain four active interface adapters, one for the NAT connection to be able to communicate with the host machine and three others to allow communications between the routers (enp0s8, enp0s9 and enp0s10). The computers have two active interface adapters, one for the communication with the host and another to communicate with the respective router.

For the creation of the virtual machines, the second experiment used the Ubuntu Server 16.04 virtual environment and follow up clones of that the configured machine, to prevent having to install all the individual machines (Figure 56). In this case the clones use different MAC addresses to prevent wrong communications and use the linked clone that create a new differencing disk image based on the original virtual machine disk image [56].

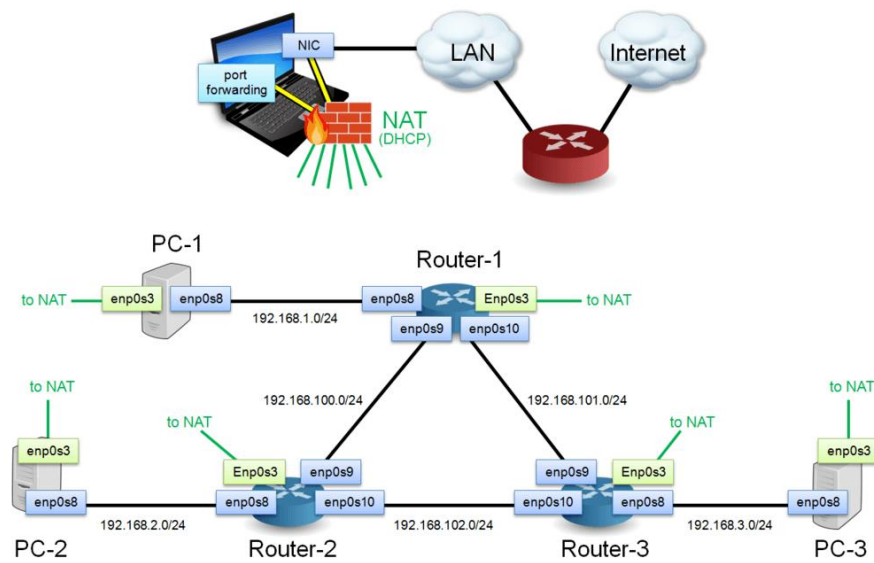


Figure 55: First network setup [55]

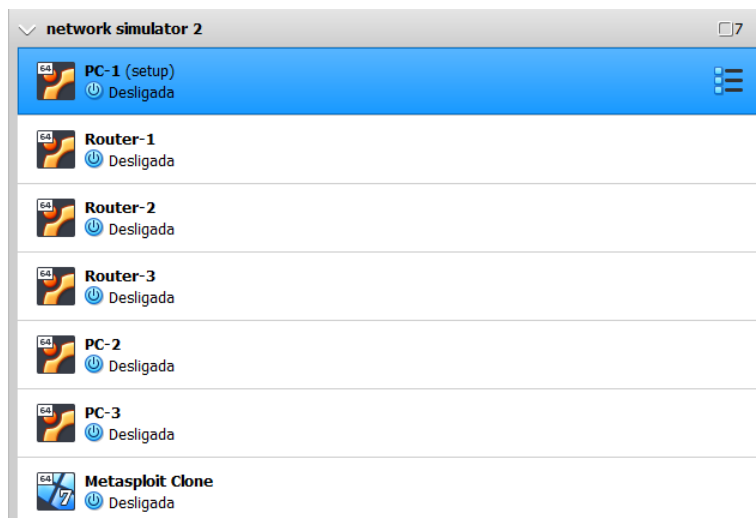


Figure 56: Machines on the first setup

With the six virtual machines installed, their interface network adapter was configured. On the first adapter, the NAT mode was chosen to allow a TCP port forwarding using a SSH communication with the host machine. This way, the setup and configuration of the network takes less time to accomplish. Each virtual machine communicates through the Port 22 (SSH), while at the same time a unique port on the host to establish the communication, Figure 57.



Figure 57: SSH connection

For the second adapter, the Internal Network mode was used so each virtual machine can communicate with each other only in a closed environment. In the case of each computer, there is only one adapter with the Internal Network mode, having every one of them a different name because they communicate directly only with the corresponding router. On the case of the routers, they have three adapters with the Internal Network mode, in which one communicates directly with one computer, and the two others communicate directly with each one of the other two routers.

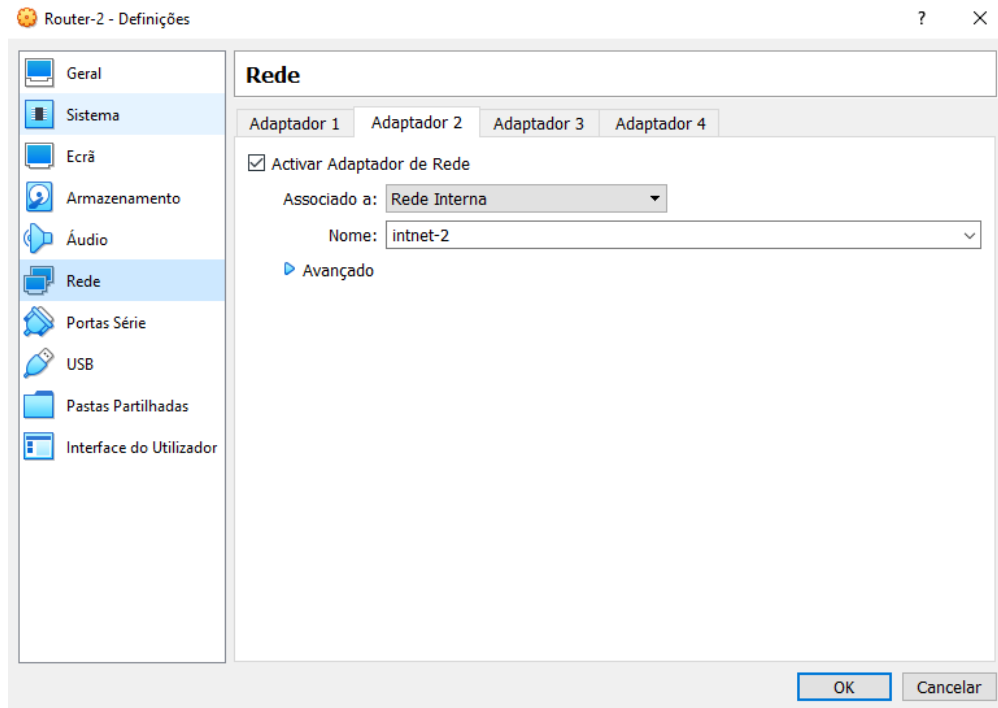


Figure 58: Internal Network

With the virtual machines running, the configuration of their network interface and routing protocols was done. Each computer has the previously defined network interface adapter (enp0s8) but without any configuration of IP Address. For this purpose, a block of input was redirected to the bash shell command line starting with the first line which tell the bash shell that a new block (EOF2) is starting. End-of-file (EOF) is an operating system condition that tell the computer when there are no more data to be read from the data source. In this case the EOF2 creates a new block of inputs but inside of it, there is an additional EOF block, thus the unusual adding of "2". The second, third and fourth line tell the computer the new assigned hostname name. The fifth line will input into the location */etc/network/interfaces* the configuration of the new network adapter. In this case of the Ubuntu version, it corresponds to enp0s8, and the new address will be 192.168.2.1 with a netmask of 255.255.255.0.

Finally, it adds the new to the 192.168.0.0/16 address and will connect directly to the router at the IP Address 192.168.2.254. With this addition, the network is restarted and the block ends (Figure 59). The same process is repeated for the other machines with the only difference being the IP Addresses of the machine and the corresponding router.

```

ubuntu@pc2:~$ bash <<EOF2
> sed -i 's/ubuntu/pc2/g' /etc/hostname
> sed -i 's/ubuntu/pc2/g' /etc/hosts
> hostname pc2
> cat >> /etc/network/interfaces << EOF
> auto enp0s8
> iface enp0s8 inet static
>     address 192.168.2.1
>     netmask 255.255.255.0
> up route add -net 192.168.0.0/16 gw 192.168.2.254 dev enp0s8
> EOF
> /etc/init.d/networking restart
> exit
> EOF2

```

Figure 59: Virtual machine configuration

In the case of the router, the process is slightly different. The first four lines start the same, the fifth update the machine and installs quagga, a network routing software that provides implementation of Open Shortest Path First, Routing Information Protocol and others [57]. This will allow the virtual machine “routers” to behave like routers. Follows a normal configuration of quagga in the machine and the setup of different network interfaces, in this case enp0s8, enp0s9 and enp0s10. The networks to communicate with each router and computer is also setup along with the IP Address of the machine.

```

ubuntu@router1:~$ bash <<EOF2
> sed -i 's/ubuntu/router1/g' /etc/hostname
> sed -i 's/ubuntu/router1/g' /etc/hosts
> hostname router1
> apt-get update
> apt-get install quagga quagga-doc traceroute
> cp /usr/share/doc/quagga/examples/zebra.conf.sample /etc/quagga/zebra.conf
> cp /usr/share/doc/quagga/examples/ospfd.conf.sample /etc/quagga/ospfd.conf
> chown quagga:quaggavty /etc/quagga/*.conf
> chmod 640 /etc/quagga/*.conf
> sed -i s'/zebra=no/zebra=yes/' /etc/quagga/daemons
> sed -i s'/ospfd=no/ospfd=yes/' /etc/quagga/daemons
> echo 'VTYSH_PAGER=more' >>/etc/environment
> echo 'export VTYSH_PAGER=more' >>/etc/bash.bashrc
> cat >> /etc/quagga/ospfd.conf << EOF
> interface enp0s8
> interface enp0s9
> interface enp0s10
> interface lo
> router ospf
>   passive-interface enp0s8
>   network 192.168.1.0/24 area 0.0.0.0
>   network 192.168.100.0/24 area 0.0.0.0
>   network 192.168.101.0/24 area 0.0.0.0
> line vty
> EOF
> cat >> /etc/quagga/zebra.conf << EOF
> interface enp0s8
> ip address 192.168.1.254/24
> ipv6 nd suppress-ra
> interface enp0s9
> ip address 192.168.100.1/24
> ipv6 nd suppress-ra
> interface enp0s10
> ip address 192.168.101.2/24
> ipv6 nd suppress-ra
> interface lo
> ip forwarding
> line vty
> EOF
> /etc/init.d/quagga start

```

Figure 60: Router setup

With all the machines setup and started, it's possible to ping and communicate between them, as seen on Figure 61. In this case the machine 192.168.3.3 will be used, further detailed on Figure 66.

```
ubuntu@pc2:~$ ping 192.168.3.3
PING 192.168.3.3 (192.168.3.3) 56(84) bytes of data.
64 bytes from 192.168.3.3: icmp_seq=1 ttl=62 time=4.31 ms
64 bytes from 192.168.3.3: icmp_seq=2 ttl=62 time=0.624 ms
^C
--- 192.168.3.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.624/2.467/4.310/1.843 ms
ubuntu@pc2:~$ traceroute 192.168.3.3
traceroute to 192.168.3.3 (192.168.3.3), 30 hops max, 60 byte packets
 1  192.168.2.254 (192.168.2.254)  0.459 ms  0.441 ms  0.433 ms
 2  192.168.102.1 (192.168.102.1)  0.401 ms  0.392 ms  0.338 ms
 3  192.168.3.3 (192.168.3.3)  78.763 ms  78.715 ms  78.708 ms
```

Figure 61: Ping and trace other virtual machines

The next step is the implementation of different operating systems into this configured network. Since the previous tests were done with the OWASP and Metasploit machines, one of each is added into the network. However, due to the type of distribution, the setup was slightly different. First the network interface had to be added on the VirtualBox, this allows for the configuration of an extra interface, but it does not show on the command line (Figure 62), the command “ifconfig eth1 up” must be used to associate with the open interface. Now that the interface is up, the configuration takes place on the “/etc/network/interfaces” where the IP Address is attributes and the communication with the corresponding router, Figure 63 and Figure 64.

```
msfadmin@metasploitable:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:d5:65:71
          inet addr:10.0.2.8  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fed5:6571/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1138 errors:0 dropped:0 overruns:0 frame:0
          TX packets:636 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:79399 (77.5 KB)  TX bytes:44651 (43.6 KB)
          Base address:0xd010  Memory:f0200000-f0220000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128  Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:134 errors:0 dropped:0 overruns:0 frame:0
          TX packets:134 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:40017 (39.0 KB)  TX bytes:40017 (39.0 KB)
```

Figure 62: Lack of visual adapter

```
eth1      Link encap:Ethernet  HWaddr 08:00:27:21:9e:ed
          inet6 addr: fe80::a00:27ff:fe21:9eed/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:238 (238.0 B)
          Base address:0xd040  Memory:f0820000-f0840000
```

Figure 63: Eth1 adapter

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet dhcp

auto eth1
iface eth1 inet static
    address 192.168.2.4
    netmask 255.255.255.0
up route add -net 192.168.0.0/16 gw 192.168.2.254 dev eth1
```

Figure 64: Network adapter configuration

## 6.3. PHASE THREE

After all the preparations in the previous phases, the third and last phase adds more machines into the existing network. The initial hospital network that was going to be simulated on a smaller scale, can be seen on the Figure 65. It belongs to a scientific paper titled “Intelligent cyberattack detection on SAFECARE virtual hospital\*” [58], developed by various members of the University, Instituto Superior de Engenharia do Porto - ISEP. The



first idea was to replicate at least four subnetworks of the hospital, however as it will be discussed later, it was not possible to run virtual machines past a certain number.

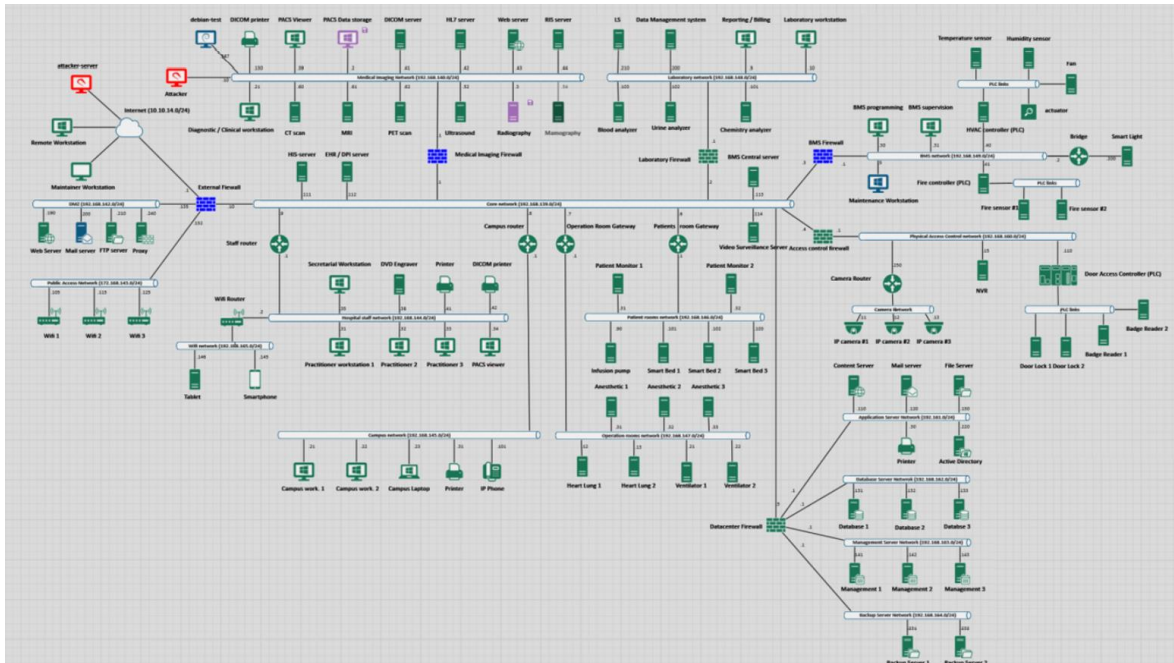


Figure 65: Virtual Hospital [58]

The network made comprised of the same components of the phase two plus an additional two vulnerable machines in each subnet. The Figure 66 shows three routers each communicating between each other through the corresponding 192.168.10X.0/24 network (being X between 0 and 2). The same routers are also assigned with the IP Address of 192.168.X.254, so that each machine can know how to communicate with the corresponding router.

On the subnetwork part, there are three of them with each one having a total of three virtual machines, where two are vulnerable, darker red the Metasploit machine and brighter red the OWASP machine, and another is a normal Ubuntu Server 16.04. The scanning machine is connected to the first router through the assigned IP Address 192.168.1.10 (blue laptop). Because of the connectivity between all the subnetworks, the scanning machine can detect every host and router present on the whole network.

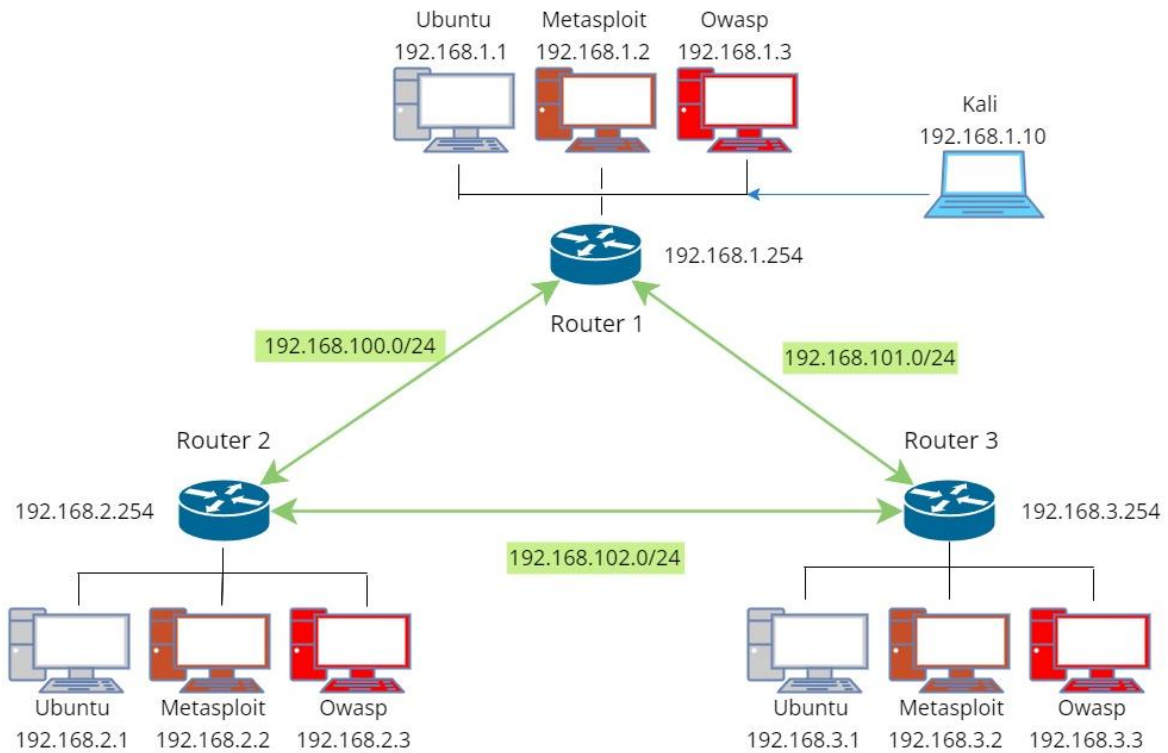


Figure 66: Phase two network with additional hosts

The results show that a scan of 13 hosts takes around 15min to complete with only the top 1000 ports to be scanned. It found 1035 vulnerabilities since there are 2 vulnerable machines in each subnetwork (Figure 67). From those, 54 were Critical, 429 High, 474 Medium, 83 Low and 4 Informative (Figure 69).

SCAN	START	FINISH	VULNERABILITIES FOUND	Details	Report	Delete
Final test	2022-10-23 13:09:10	2022-10-23 13:23:53	1035	Details	Report	Delete
Final test	2022-10-23 12:50:48	2022-10-23 13:23:53	1035	Details	Report	Delete
testing7	2022-10-22 21:29:32	2022-10-22 21:29:38	0	Details	Report	Delete
testing7	2022-10-22 21:25:49	2022-10-22 21:26:13	0	Details	Report	Delete
testing6	2022-10-22 21:24:36	2022-10-22 21:29:38	0	Details	Report	Delete
testing5	2022-10-22 17:07:12	2022-10-22 17:08:09	40	Details	Report	Delete
testing5	2022-10-22 17:04:09	2022-10-22 17:04:56	20	Details	Report	Delete
testing5	2022-10-22 17:03:21	2022-10-22 17:04:56	20	Details	Report	Delete

Figure 67: Scan results history – first network

IP	STATE	VULNERABILITIES	
192.168.1.1	up	20	<a href="#">Details</a>
192.168.1.2	up	203	<a href="#">Details</a>
192.168.1.3	up	102	<a href="#">Details</a>
192.168.1.254	up	20	<a href="#">Details</a>
192.168.1.10	up	0	<a href="#">Details</a>
192.168.2.1	up	20	<a href="#">Details</a>
192.168.2.2	up	102	<a href="#">Details</a>
192.168.2.4	up	203	<a href="#">Details</a>
192.168.2.254	up	20	<a href="#">Details</a>
192.168.3.1	up	20	<a href="#">Details</a>

Figure 68: Scan results hosts – first network

24/10/2022 20:13:06

### Scan Report: full\_scan\_twelve

Start Date: 2022-10-24 17:42:50

Finish Date: 2022-10-24 18:02:36

#### Scan Results

IP range	Hosts	Total Vulnerabilities
192.168.1-3.0/24	13	1035

#### Level of Vulnerabilities

Severity	Number
Critical	54
High	429
Medium	474
Low	93
Informative	4

Figure 69: First network scan report

The second network took the first one and expanded to accommodate more virtual machines. For the first and second router, 5 vulnerable machines were added, 3 Metasploit and 2 Owasp. On the third router, 4 machines were added, 3 Metasploit and 1 Owasp. However, during launch of all the network, the computer could not handle the execution of so many virtual machines at once, so 5 of them (from the second router) displayed an error and stopped executing. Even so, this network simulation counted with 22 hosts to perform the scans (Figure 70). Running a custom scan from the port 1 to 8080, a total of 2613 vulnerabilities were found, of which, 138 Critical, 1077 High, 1287 Medium, 153 Low and 4 Informative (Figure 71 and Figure 72).

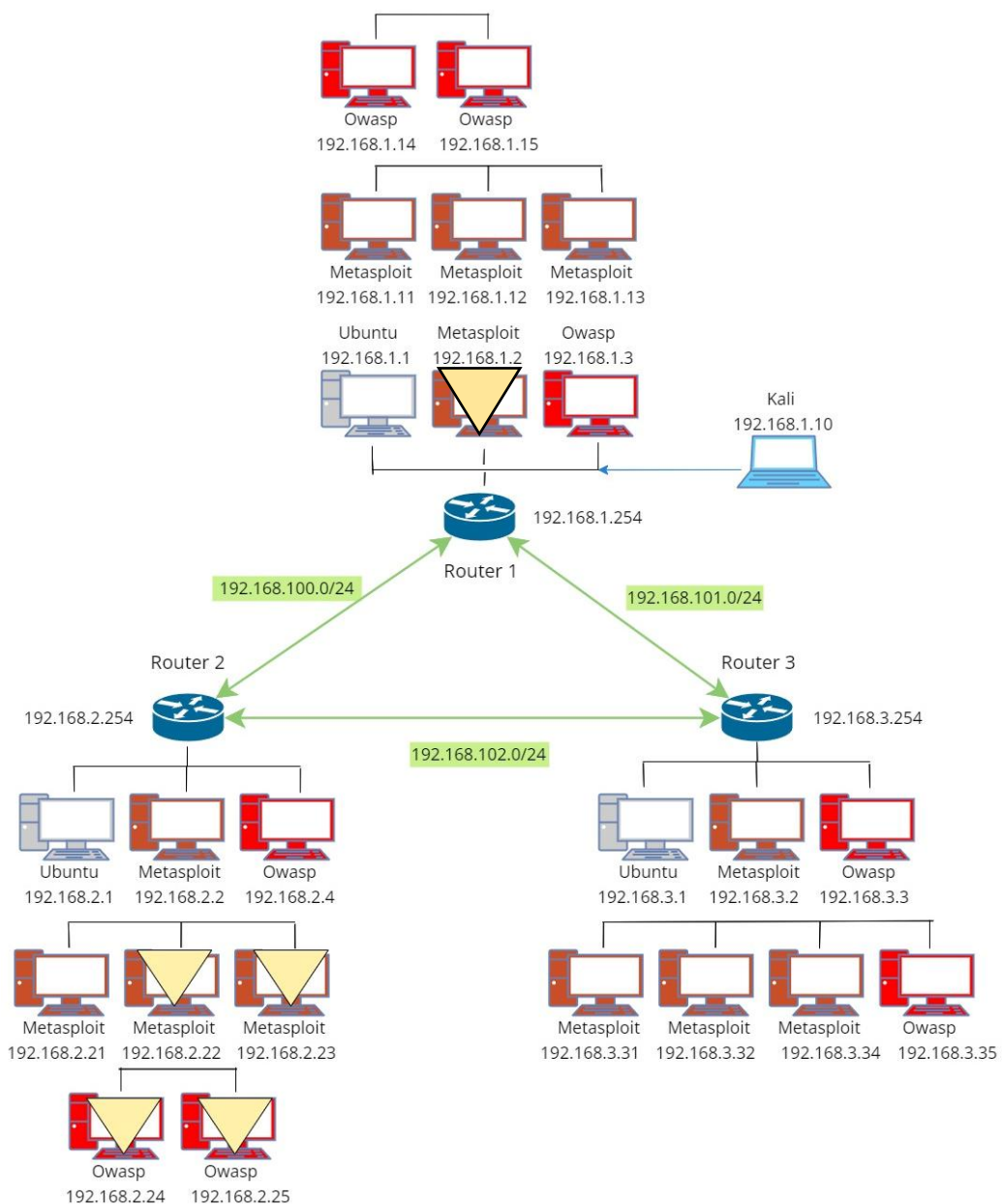


Figure 70: Phase three network



Figure 71: Scan results history – second network

24/10/2022 13:14:32

### Scan Report: network 2

Start Date: 2022-10-24 11:08:06  
Finish Date: 2022-10-24 11:58:51

Scan Results

IP range	Hosts	Total Vulnerabilities
192.168.1-3.0/24	22	2613

Level of Vulnerabilities

Severity	Number
Critical	138
High	1077
Medium	1287
Low	153
Informative	4

Host Discovered

Hosts	Vulnerabilities
192.168.1.1	20
192.168.1.3	102
192.168.1.11	209
192.168.1.12	209
192.168.1.13	209
192.168.1.14	102
192.168.1.15	102
192.168.1.254	20
192.168.1.10	0
192.168.2.1	20

192.168.2.2	102
192.168.2.4	209
192.168.2.21	209
192.168.2.254	20
192.168.3.1	20
192.168.3.2	209
192.168.3.3	102
192.168.3.31	209
192.168.3.32	102
192.168.3.34	209
192.168.3.35	209
192.168.3.254	20

Figure 72: Scan report – second network

A comparison of the time taken for each scan is illustrated on the Table 14. The tests of each scan were conducted 3 times to obtain a reasonable number of samples and an average time was calculated. The first three rows of the table correspond to the first simulated network with only 13 hosts and the time taken in each scan was roughly the same, only the basic scan had a slight advantage taken 2 minutes less to complete. On the last three rows, the time taken was largely increased as the number of hosts went from 13 to 22. Despite the basic scan having a smaller time, the time gap to the full and custom scan is not surprisingly much bigger. The possible reason has to do with the basic scan performing the detection in the top 1000 most common ports, and the vulnerable machines having all or almost all of the services associated with those 1000 ports. Its also worth noticing that only one API (NVD) was active during these tests, since the VulDB API did not allow the number of requests necessary to perform all the existing scans.

Table 14: Comparison of scan time

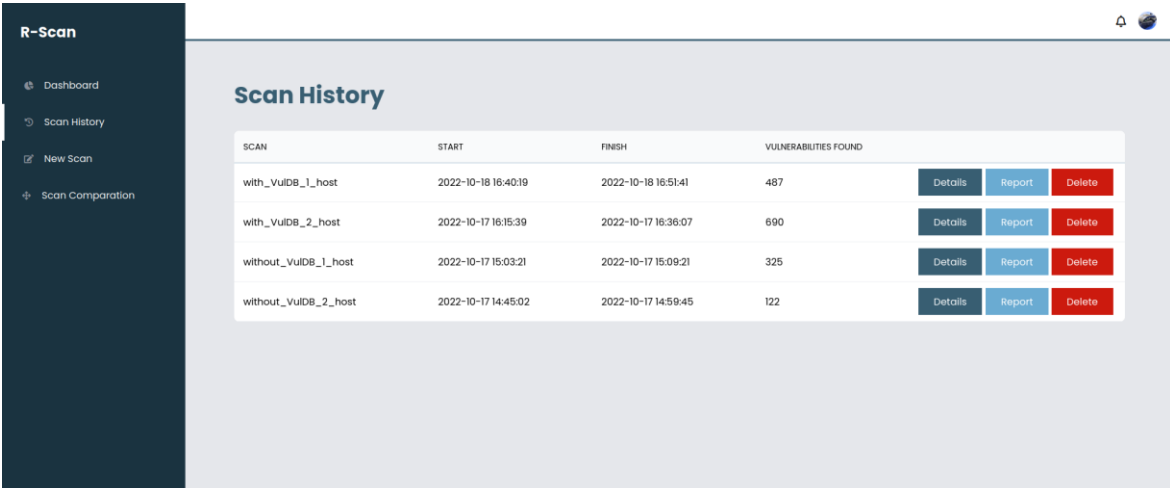
Type of scan	Number of hosts	Time
Basic	13	~ 18 minutes
Full	13	~ 20 minutes
Custom (1-8080 ports)	13	~ 20 minutes
Basic	22	~ 46 minutes
Full	22	~ 51 minutes
Custom (1-8080 ports)	22	~ 50 minutes

## 6.4. LIMITATION

During the first tests with only two vulnerable machines, it was found that only the NVD API could be used for the remaining experiments. As it can be seen on the Figure 73, when executing a scan without the VulDB API and only 1 host, 102 vulnerabilities are found. With 2 hosts the number increases to 325. In contrast, when the VulDB API is active, the number of vulnerabilities increases to 487 and 690, respectively.

On one side, this is extremely good, as it finds even more vulnerabilities, having a complete overview of the real vulnerabilities these machines may have. So, if it even increases the number of vulnerabilities, why it is a limitation to the application. The answer lies on the maximum number of requests a user can perform to the VulDB API on a free license. The current number is 50 requests per 24h, but this number was reduced to 30 requests due to penalizations by exceeding a few times the maximum number and continuing to make requests (this happens during the scan process, as only at the end its possible to visualize the exciting occurrence).

Realizing this limitation, the following experiments carried out without the use of the VulDB API since with only 2 hosts it already exceeds the limit, the addition of extra hosts will only further aggravate the problem. In conclusion, the number of vulnerabilities displayed throughout the chapter 7 is smaller than it could possibly be.



SCAN	START	FINISH	VULNERABILITIES FOUND			
with_VulDB_1_host	2022-10-18 16:40:19	2022-10-18 16:51:41	487	Details	Report	Delete
with_VulDB_2_host	2022-10-17 16:35:39	2022-10-17 16:36:07	690	Details	Report	Delete
without_VulDB_1_host	2022-10-17 15:03:21	2022-10-17 15:09:21	325	Details	Report	Delete
without_VulDB_2_host	2022-10-17 14:45:02	2022-10-17 14:59:45	122	Details	Report	Delete

Figure 73: Comparison of API results





# 7. CONCLUSIONS

This final chapter details all the conclusions and achievements done throughout this project. The goal is to link the previously set objectives with the accomplishments that have been done. It will also consider the limitations and future work, which can be included as an enhancement of the Risk Management web application.

## 7.1. OBJECTIVES ACHIEVED

It is important to denote that this thesis started later, with the actual beginning at around April and required a lot of research from the author to understand a variety of topics that were covered throughout the thesis and, were necessary in order to execute them in a successful way. Despite the slight shift in the Master's degree subject, the introduction into the cybersecurity world was well accepted.

The goal of displaying a functional Risk Management web application that could scan a network and detect vulnerabilities in different assets, was successfully achieved. Furthermore, the results have shown that the implementation of the project into any network is simple and does not require any special configuration besides the ones stated on this document.

In continuation of the objectives, it was proposed and well completed the integration of a database with all the vulnerabilities detected, and a way to connect through an API to two

well-known vulnerability databases, in order to validate the existence of vulnerabilities in different assets. Even though both APIs were initially used, the VulDB API had to be excluded given the maximum number of requests permitted per day. This limited the actual output number of vulnerabilities found as it could be seen on chapter 7.

The addition of a scan comparison between the hosts to validate the correction of vulnerabilities was introduced later into the project but was successfully executed, giving an extra feature to the application.

Lastly the initial objective of implementing a simulated network based on an existing simulated hospital network, was greatly reduced to a network of three routers and more than a dozen hosts, given the complexity of a bigger network and the hardware limitations.

## 7.2. LIMITATIONS

Regarding the limitations encountered while developing the thesis's main components, it is viable to deduce that the following restraints played a role in the implementation:

- **Hardware limitations:** Despite the fact that the computer used to perform the simulations was a fairly recent one, it was not enough to expand further the number of virtual machines that can be executed at once and grow the developed simulated network.
- **Time limitations:** Even though this thesis took some months to complete, the time required given the initial knowledge of the author on this topic, to expand further the application, was not enough, as a few more months would be needed for this accomplishment.
- **API limitations:** Despite the initial use of two APIs to better complete the detection of vulnerabilities, the VulDB ended up not being as useful as initially planned because of the maximum number of requests allowed, having to be excluded from the rest of the experiments.

## 7.3. FUTURE WORK

Following the completion and documentation of this project, it was verified that a simple Risk Management web application can be built with the junction of different tools and methods, in order to achieve a full network scan.

A simulated network was used for the testing and as a proof-of-concept of the functionality of the application. By applying this project to a real-world network, some features could be better enhanced and tuned so that a better performance would be upgraded, and a broader range could be covered.

Given the limitation of one of the APIs used, an enhancement to this project would be the addition of another vulnerability database to validate and discover new vulnerabilities on the scanned hosts.

The feature of the comparison of scans, could and should be used to validate other technologies in the effectiveness of correcting vulnerabilities in different hosts of a network. The main idea would be to experiment and train neural network models that had a network environment to patch vulnerabilities and an application to validate if changes were indeed made and some vulnerabilities corrected.

Since nowadays any security application involves the cooperation of different teams and also includes the management department by displaying concrete data of the security performance put into place, a good addition to the application would be a graphical interface that would show not only, a diagram of all the hosts found and how they are connected through the network, but also a way to measure the possibility of a vulnerability in a single host to spread to others. This method could easily identify hosts that have known vulnerabilities that have the consequence of affecting others hosts and would serve as a way to know which hosts should be isolated and taken care of as a top priority.

As stated on the introduction, this application was initially thought to be built into an existing React application and expand the capabilities by adding the vulnerability scanning and detection methods. For a future work, the integration could be possible to achieve. In this integration, the process could be moved to an automated one, without the need for manual input from the user and instead have scheduled periodic scans. This was initially thought to be implemented on this thesis but later discarded.

## 7.4. FINAL CONSIDERATIONS

As a final note, since most of the objectives have been achieved, the author considers this thesis to be a successful journey. It is important to highlight that the author had to learn a big portion of concepts and technologies used since the cybersecurity field was not the main area of the Master's degree. Given this and the results obtained, it can be said that the project conclusion was pretty satisfactory.

The author expects that this application can be expanded and integrated into more complex projects, to make a real risk management impact.

This was an appealing project that challenged the author to think of the best way to conduct research and how to successfully manage the thesis and the consequently implementation of the various components. Ultimately, the author views this project as a substantial step into the cybersecurity field and a highly satisfying opportunity for the author to reflect on his own growth.

## *Bibliography*

- [1] H. Hanif, M. H. N. Md Nasir, M. F. Ab Razak, A. Firdaus, and N. B. Anuar, "The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches," *J. Netw. Comput. Appl.*, vol. 179, no. November 2020, p. 103009, 2021, doi: 10.1016/j.jnca.2021.103009.
- [2] L. A. B. Sanguino and R. Uetz, "Software Vulnerability Analysis Using CPE and CVE," pp. 217–231, 2017, [Online]. Available: <http://arxiv.org/abs/1705.05347>
- [3] T. M. Corporation, "Mitre ATT&CK," 2015. <https://attack.mitre.org/> (accessed Oct. 13, 2022).
- [4] B. Ampel, S. Samtani, S. Ullman, and H. Chen, "Linking Common Vulnerabilities and Exposures to the MITRE ATT&CK Framework: A Self-Distillation Approach," *Work. AI-enabled Cybersecurity Anal. 2021 ACM Conf. Knowl. Discov. Data Min.*, pp. 1–5, 2021.
- [5] J. Sun, K. Pan, X. Chen, and J. Zhang, "Security Patterns from Intelligent Data: A Map of Software Vulnerability Analysis," *Proc. - 3rd IEEE Int. Conf. Big Data Secur. Cloud*,

- BigDataSecurity 2017, 3rd IEEE Int. Conf. High Perform. Smart Comput. HPSC 2017 2nd IEEE Int. Conf. Intell. Data Secur.*, pp. 18–25, 2017, doi: 10.1109/BigDataSecurity.2017.9.
- [6] MITRE, “Common Vulnerabilities and Exposures.” <http://cve.mitre.org> (accessed Sep. 20, 2022).
- [7] “Microsoft Security Advisories and Bulletins,” 2022. <https://technet.microsoft.com/en-us/library/security> (accessed Sep. 10, 2022).
- [8] “HP IT Resource Center.” <http://itrc.hp.com> (accessed Oct. 23, 2022).
- [9] IBM, “IBM Internet Security Systems Ahead of the threat.” <http://xforce.iss.net> (accessed Oct. 23, 2022).
- [10] “Bugtraq.” <http://www.securityfocus.com/> (accessed Jun. 13, 2022).
- [11] “US-CERT.” <https://www.us-cert.gov> (accessed Jun. 13, 2022).
- [12] I. FIRST.org, “Common Vulnerability Scoring System, V3 Development Update,” 2017. <https://www.first.org/cvss/> (accessed Oct. 23, 2022).
- [13] W. F. Chong, R. Feng, H. Hu, and L. Zhang, “Cyber Risk Assessment for Capital Management,” 2022, [Online]. Available: <http://arxiv.org/abs/2205.08435>
- [14] M. E. Paté-Cornell, M. Kuypers, M. Smith, and P. Keller, “Cyber Risk Management for Critical Infrastructure: A Risk Analysis Model and Three Case Studies,” *Risk Anal.*, vol. 38, no. 2, pp. 226–241, 2018, doi: 10.1111/risa.12844.
- [15] M. McShane, M. Eling, and T. Nguyen, “Cyber risk management: History and future research directions,” *Risk Manag. Insur. Rev.*, vol. 24, no. 1, pp. 93–125, 2021, doi: 10.1111/rmir.12169.
- [16] E. Humphreys, “Information security management standards: Compliance, governance and risk management,” *Inf. Secur. Tech. Rep.*, vol. 13, no. 4, pp. 247–255, 2008, doi: 10.1016/j.istr.2008.10.010.

- [17] J. Moteff, "Risk Management and Critical Infrastructure Protection : Assessing , Integrating , and Managing Threats , Vulnerabilities and Consequences," *Sci. Technol.*, pp. 1–29, 2005.
- [18] S. Furnell, A. I. Awad, M. Paprzycki, and S. K. Sharma, "Security in Cyber-Physical Systems - Foundations and Applications," *Stud. Syst. Decis. Control*, vol. 339, 2021, doi: 10.1007/978-3-030-67361-1\_8.
- [19] R. Böhme, S. Laube, and M. Riek, "A Fundamental Approach to Cyber Risk Analysis," *Variance. Adv. Sci. Risk*, vol. 12, no. 2, pp. 161–185, 2019.
- [20] E. Saponi, M. Sciutto, and G. Sciutto, "A quantitative approach to risk management in critical infrastructures," *Transp. Res. Procedia*, vol. 3, no. July, pp. 740–749, 2014, doi: 10.1016/j.trpro.2014.10.053.
- [21] H. Holm, "Performance of automated network vulnerability scanning at remediating security issues," *Comput. Secur.*, vol. 31, no. 2, pp. 164–175, 2012, doi: 10.1016/j.cose.2011.12.014.
- [22] I. Lee, "Cybersecurity: Risk management framework and investment cost analysis," *Bus. Horiz.*, vol. 64, no. 5, pp. 659–671, 2021, doi: 10.1016/j.bushor.2021.02.022.
- [23] F. Kim, "How to make sense of cybersecurity frameworks," 2019. <https://www.frankkim.net/blog/how-to-make-sense-of-cybersecurity-frameworks>
- [24] NIST, "NIST SP 800-53," 2020. <https://www.nist.gov/privacy-framework/nist-sp-800-53> (accessed Oct. 23, 2022).
- [25] ISO, "ISO/IEC 27001 and related standards." <https://www.iso.org/isoiec-27001-information-security.html> (accessed Oct. 23, 2022).
- [26] S. Almuhammadi and M. Alsaleh, "Information Security Maturity Model for Nist Cyber Security Framework," pp. 51–62, 2017, doi: 10.5121/csit.2017.70305.
- [27] D. Maclean, "The NIST Risk Management Framework: Problems and recommendations," *Cyber Secur. A Peer-Reviewed J.*, vol. 1, pp. 207–217, 2017.

- [28] P. Radanliev, "Cyber Risk Management for the Internet of Things," *Univ. Oxford Comb. Work. Pap. Proj. reports Prep. PETRAS Natl. Cent. Excell. Cisco Res. Cent.*, no. April, pp. 1–27, 2019, doi: 10.20944/preprints201904.0133.v1.
- [29] V. McCoy, "FAIR On-A-Page: Same Great Model, Fresh New Look," 2017. <https://www.fairinstitute.org/blog/fair-model-on-a-page> (accessed Oct. 23, 2022).
- [30] A. nationale de la sécurité des Systèmes and D'information, "EBIOS — Expression des Besoins et Identification des Objectifs de Sécurité."
- [31] P. B. Nassar, Y. Badr, K. Barbar, and F. Biennier, "Risk Management and Security in Service-based Architectures," *2017 IEEE 7th Annu. Comput. Commun. Work. Conf.*, pp. 214–218, 2009, doi: 10.1109/CCWC.2017.7868444.
- [32] B. Fatima and T. Rabat, "Risk analysis in Internet of Things using EBIOS Berrehili Fatima zahra," *Zahra, Berrehili Fatima Belmekki Abdelhamid. "Risk Anal. Internet Things using EBIOS."* *2017 IEEE 7th Annu. Comput. Commun. Work. Conf. 1-7.*, 2017, doi: 10.1109/CCWC.2017.7868444.
- [33] J. Mcdonald, A. Hecker, and F. Planchon, "Application of EBIOS for the risk assessment of ICT use in electrical distribution sub-stations," 2010.
- [34] S. S. Alizadeh and P. Moshashaei, "The Bowtie method in safety management system - A literature review," *Sci. J. Rev.*, vol. 4, no. 9, pp. 133–138, 2015, doi: 10.14196/sjr.v4i9.1933.
- [35] "Bow tie analysis," 2019. <https://broadleaf.com.au/resource-material/bow-tie-analysis/> (accessed Oct. 23, 2022).
- [36] W. J. Chun, "History of Python - Core Python Programming [Book]," *December 2000*. [https://www.oreilly.com/library/view/core-python-programming/0130260363/0130260363\\_ch01lev1sec2.html](https://www.oreilly.com/library/view/core-python-programming/0130260363/0130260363_ch01lev1sec2.html)
- [37] P. S. Foundation, "Python." <https://www.python.org/> (accessed Sep. 10, 2022).
- [38] "Flask." <https://flask.palletsprojects.com/en/2.1.x/> (accessed Sep. 10, 2022).



- [39] D. S. Foundation, "Django." <https://www.djangoproject.com/> (accessed Sep. 10, 2022).
- [40] "Nmap." <https://nmap.org/> (accessed Sep. 10, 2022).
- [41] Oracle, "VirtualBox." <https://www.virtualbox.org/> (accessed Sep. 10, 2022).
- [42] M. Bose, "VirtualBox Network Modes," *NAKIVO*, 2019. <https://www.nakivo.com/blog/virtualbox-network-setting-guide/> (accessed Sep. 13, 2022).
- [43] SolarWinds, "GNS3." <https://www.gns3.com/> (accessed Sep. 10, 2022).
- [44] SolarWinds, "GNS3 docs." <https://docs.gns3.com/docs/> (accessed Jun. 13, 2022).
- [45] Nmap, "Nmap Scan types." <https://nmap.org/book/host-discovery-techniques.html> (accessed Oct. 13, 2022).
- [46] Nmap, "Syn Scan Nmap." <https://nmap.org/book/synscan.html> (accessed Oct. 13, 2022).
- [47] Nmap, "ACK Scan Nmap." <https://nmap.org/book/scan-methods-ack-scan.html> (accessed Oct. 13, 2022).
- [48] R. Hat, "What is an API?," 2022. <https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces> (accessed Oct. 10, 2022).
- [49] N. V. Database, "CVE API," 2022. <https://nvd.nist.gov/developers/vulnerabilities> (accessed Oct. 10, 2022).
- [50] V. pyxyp Inc, "VulDB," 2022. <https://vuldb.com/?kb.api> (accessed Oct. 16, 2022).
- [51] I. Tenable®, "SSL expired date," 2022. <https://www.tenable.com/plugins/nessus/15901> (accessed Oct. 19, 2022).
- [52] I. Tenable®, "SSL Protocol Version," 2022. <https://www.tenable.com/plugins/was/112546> (accessed Oct. 19, 2022).

- [53] C. Willis, "OWASP Project," 2015. [https://owasp.org/www-project-broken-web-applications/migrated\\_content](https://owasp.org/www-project-broken-web-applications/migrated_content) (accessed Oct. 13, 2022).
- [54] Metasploit, "Metasploitable 2." <https://docs.rapid7.com/metasploit/metasploitable-2/> (accessed Oct. 13, 2022).
- [55] B. Linkletter, "How to emulate a network using VirtualBox," *July 4, 2016*. <https://www.brianlinkletter.com/2016/07/how-to-use-virtualbox-to-emulate-a-network/> (accessed Oct. 22, 2022).
- [56] O. V. VirtualBox, "Cloning VirtualBox." <https://docs.oracle.com/en/virtualization/virtualbox/6.0/user/clone.html> (accessed Oct. 23, 2022).
- [57] P. Jakma, "Quagga," 2018. <https://www.nongnu.org/quagga/> (accessed Oct. 22, 2022).
- [58] E. Maia, D. Lancelin, J. Carneiro, T. Oudin, Á. Dória, and I. Praça, "Intelligent Cyberattack Detection on SAFECARE Virtual Hospital," *Lect. Notes Networks Syst.*, vol. 470 LNNS, no. 787002, pp. 327–337, 2022, doi: 10.1007/978-3-031-04829-6\_29.

