UNIVERSITY OF PIRAEUS - DEPARTMENT OF INFORMATICS

MSc "Distributed Systems, Security and Emerging Information Technologies"

<u>MSc Thesis</u>

Thesis Title: Τίτλος Διατριβής	***“Design & Development of a Cybersecurity Tool for Attack Surface Discovery with Automated Target-Network Reconnaissance”*** *“Σχεδιασμός & Υλοποίηση Ενός Εργαλείου Κυβερνοασφάλειας για την Ανίχνευση Επιφάνειας Επίθεσης με Αυτοματοποιημένη Χαρτογράφηση Δικτύου-Στόχου”*
Student’s name-surname:	**Georgios Tasios**
Father’s name:	**Christos**
Student’s ID No:	**ΜΠΚΣΑ/20016**
Supervisor:	**Panagiotis Kotzanikolaou, Professor**

November 2024

3-Member Examination Committee

Panagiotis Kotzanikolaou	**Christos Douligeris**	**Michael Psarakis**
Professor	**Professor**	**Associate Professor**

Acknowledgements

I would like to express my deepest gratitude to Professor Mr. Panayiotis Kotzanikolaou for his invaluable assistance in the implementation of this thesis. I am also grateful to Mr. Christos Grigoriadis, Ph.D. (c), for his crucial support throughout this process. The insightful conversations and brainstorming sessions we shared in the field of cybersecurity, were instrumental in shaping my approach, helping me navigate and overcome various challenges.

My heartfelt thanks go to all my professors, from the beginning of my academic journey to the completion of my Master's degree. Their teachings have instilled in me the knowledge, confidence, and skills necessary to reach this significant milestone. The pursuit of academic and scientific knowledge is a never-ending journey, and I look forward to continuing on this path.

Above all, I am eternally grateful to my wife, Dr. Maria-Eirini Tselegkidi, for her endless patience, support, and understanding throughout this journey. However, my deepest and most profound gratitude goes to my young son, Christos, whose energy, laughter, and love have been a source of inspiration. As a toddler, his joy and innocence have provided me with the strength to keep moving forward, serving as a guiding light throughout the entire process.

Στη Μαρήνη και στον Χρηστάκη μου…

Table of Contents

Περίληψη

Οι σύγχρονες εξελίξεις των τεχνολογικών υποδομών, έχουν οδηγήσει σε μία μετατόπιση προς νέες τεχνολογίες όπως οι υπηρεσίες cloud, το Διαδίκτυο των Πραγμάτων (IoT) και οι ακροδικτυακές συσκευές. Αυτή η συνεχής επέκταση προς νέες τεχνολογίες προκαλεί αύξηση των εκτεθειμένων επιφανειών επίθεσης, δημιουργώντας σημαντικές προκλήσεις για την κυβερνοασφάλεια. Η παρούσα διατριβή εξετάζει πώς η αυξανόμενη έκθεση των επιχειρηματικών λειτουργιών δημιουργεί ευπάθειες, απαιτώντας τόσο αυξημένα μέτρα κυβερνοασφάλειας όσο και ευαισθητοποίηση των εργαζομένων. Η έννοια της επιφάνειας επίθεσης αποδομείται, ξεκινώντας από θεμελιώδη στοιχεία, όπως τα ονόματα domains, οι διευθύνσεις IP, οι δικτυακές θύρες και οι υπηρεσίες, που αποτελούν συνήθεις αρχικούς τρόπους πρόσβασης γνωστών εκστρατειών επίθεσης. Η παρούσα μελέτη εξετάζει επίσης πώς λανθασμένες διαμορφώσεις ρυθμίσεων, όπως οι ανοιχτές θύρες και οι εκτεθειμένες υπηρεσίες, μπορούν να καταγραφούν και να ευθυγραμμιστούν με αντίστοιχα δεδομένα Ανοιχτών Πηγών Δεδομένων Κυβερνοαπειλών (OSCTI). Λαμβάνοντας υπόψη τα θεμελιώδη στοιχεία της επιφάνειας επίθεσης, καθώς και τις διασυνδέσεις τους με δεδομένα OSCTI, η διατριβή αυτή παρουσιάζει μια μεθοδολογία, καθώς και μια υλοποίηση που μπορεί να αυτοματοποιήσει τη διαδικασία καταγραφής και οπτικοποίησης της επιφάνειας επίθεσης των οργανισμών, στο πλαίσιο των στοιχείων (υποδομής) και των ευπαθειών που μπορούν να ανιχνευθούν πίσω από συγκεκριμένα domains. Αυτή η προσέγγιση μπορεί να υποστηρίξει τον τρέχοντα ρυθμό επέκτασης, προσφέροντας μια λύση που μπορεί να χαρτογραφεί συνεχώς τα νέο-ενσωματωμένα (δομικά) στοιχεία σε ήδη εκτεθειμένες υποδομές, μέσω της επαναληπτικής χρήσης της. Η λύση αυτή προσφέρει μια βαθύτερη εικόνα για την ασφαλέστερη θωράκιση της υποδομής, ελαχιστοποιώντας την έκθεση και μετριάζοντας τους κινδύνους διαρκώς, συμβάλλοντας τελικά σε μια πιο προσαρμοσμένη στρατηγική κυβερνοασφάλειας.

Abstract

As modern infrastructures grow, they increasingly rely on modern technologies such cloud services, IoT and edge devices. This technological expansion, causes current attack surfaces to grow, posing significant challenges for cybersecurity. This thesis examines how the growing exposure of business functions creates vulnerabilities, requiring both increased cybersecurity measures and employee awareness. The concept of the attack surface is broken down, starting with foundational elements such as domain names, IP addresses, ports and services which are common initial access vectors of known attack campaigns. The current study also explores how misconfigurations such as open ports and exposed services can be enumerated and aligned to correlated Open-Source Cyber Threat Intelligence (OSCTI) data. Considering the foundational elements of the attack surface, as well as their interconnections with OSCTI, this thesis presents a methodology as well as an implementation that can automate the procedure of enumerating and visualizing the attack surface of organizations in the context of assets and vulnerabilities that can be traced under specific domains. This approach can support the current rate of expansion by offering a solution able to continuously map newly integrated assets in pre-existing exposed infrastructures, through iterative usage. This solution provides insights into securing infrastructure more effectively by minimizing exposure and mitigating risks in a perpetual manner, ultimately contributing to a more resilient cybersecurity posture.

1. Introduction

As modern infrastructure expands and integrates cutting-edge technology, more business functions are migrating to the cloud, significantly increasing their exposure to the open internet. While this shift offers greater functionality and accessibility to a broader customer base, it simultaneously expands the **network attack surface**, posing a significant challenge to the cybersecurity community.

On one hand, employees must be well-trained to recognize and avoid potential threats. On the other hand, cybersecurity specialists face the daunting task of securing the **network "paths"** that could be exploited by attackers to infiltrate an organization's infrastructure. This task is compounded by the ever-growing number of exposed network assets, making it increasingly challenging to protect against cyber threats.

The core issue for cybersecurity professionals is to analyze and fortify these assets since they often represent the first "links" in the kill chain of most cyber incidents. In this context, it's crucial to deconstruct the components that make up the network attack surface of an organization, beginning with the term "Domain."

A "**Domain**," in general terms, refers to a distinct area or field of knowledge or activity. In the context of computer networks, a domain represents a unique subset of the internet, typically associated with websites and online services. This subset consists the foundation for websites and online services. Domains registered to individuals or companies can rarely be found without being mapped to an online service or website. Domains are often subdivided into subdomains to manage different sections or services within an organization.

Another fundamental concept in networking is the Internet Protocol (**IP**) address, often abbreviated as IP address. An IP address is a numerical identifier assigned to every device connected to the internet (global IPs) or any local network (local IPs), allowing these devices to communicate. There are two main versions: IPv4, a 32-bit address, and IPv6, a 128-bit address. These IPs are essential for the transmission of data packets across networks and form the foundation of device communication.

Each IP address typically hosts a variety of services, such as websites or network applications, which operate behind virtual endpoints known as "**ports**." Ports enable multiple **services** to run on a single IP address, with up to 65,535 possible ports per IP. Certain ports are commonly associated with specific services (e.g., port 80 for HTTP, port 443 for HTTPS, and port 25 for email traffic). Cybersecurity measures like firewalls, Intrusion Prevention Systems (IPS), and Intrusion Detection Systems (IDS) help protect these ports from unauthorized access.

Despite these security measures, vulnerabilities often arise due to system misconfigurations, lack of awareness, or inadequate protection, leaving ports open and exposed to malicious exploitation. Today, the attack surface of many businesses includes various services connected to the web, linking to internal network assets. These assets can be compromised in a systematic manner, typically following well-known attack patterns known as attack lifecycles or Cyber Kill Chains, which consist of distinct stages [1] [2].

1.1. The current status quo

1.1.1. A view of the Attack lifecycle that a lot of adversaries implement

Out there we meet a lot of standards that provide guidance to industry or other governmental and private organizations to mitigate and manage cybersecurity risks. Some examples are the NIST Cybersecurity Framework (CSF)[1] whose audience is those who are responsible to develop cybersecurity programs as well as everybody else who is involved in Cybersecurity Risk management, the MITRE ATT&CK Framework[2], which actually is a knowledge base of adversary techniques based on real paradigms' observation and analysis, that consists the foundation on which specific threat models can be built and many others.

In this thesis we are based on and were motivated mostly by another well-known and worldwide spread framework, the "Cyber Kill Chain Framework"[3], (also referred to as Attack Lifecycle) developed by Lockheed Martin company, leading in defense tech and cybersecurity landscape.

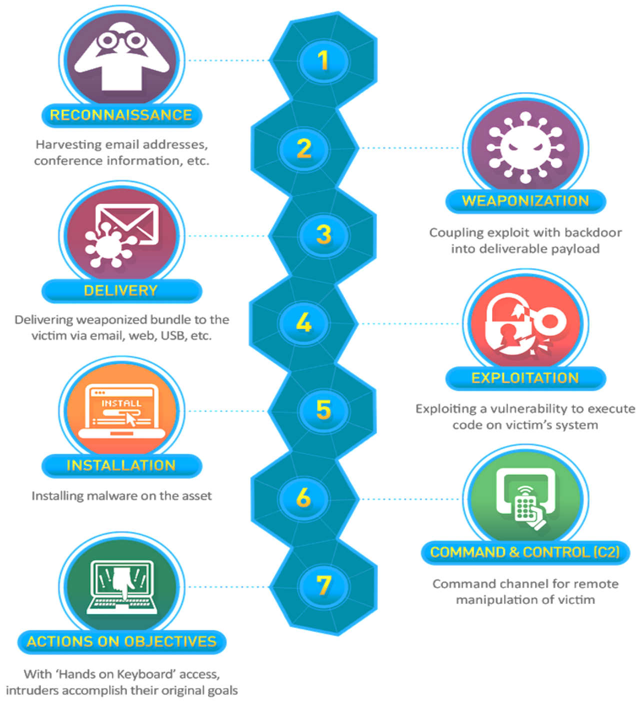

1
RECONNAISSANCE
Harvesting email addresses, conference information, etc.
2
WEAPONIZATION
Coupling exploit with backdoor into deliverable payload
3
DELIVERY
Delivering weaponized bundle to the victim via email, web, USB, etc.
4
EXPLOITATION
Exploiting a vulnerability to execute code on victim's system
5
INSTALLATION
Installing malware on the asset
6
COMMAND & CONTROL (C2)
Command channel for remote manipulation of victim
7
ACTIONS ON OBJECTIVES
With 'Hands on Keyboard' access, intruders accomplish their original goals

Figure 1 The Cyber Kill Chain steps (By Lockheed Martin)

In figure 1 we notice the visualization of the steps of the Cyber Kill Chain, a view of the Attack lifecycle that a lot of adversaries implement but also many huge organizations which supply cybersecurity services (like XDR, MDR etc.) follow.

[1] https://www.nist.gov/cyberframework
[2] https://attack.mitre.org/
[3] https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html

1.1.2. Attack lifecycle Stages

Despite we are based on the “Reconnaissance” step of the above for the implementation of this thesis, following we analyze in a few sentences all step of it, in order the reader to get a more spherical understanding of the project.

1.1.2.1. Reconnaissance

It is the first step of a cyber-attack. In lifecycle is the first stage, an initial phase where attackers gather all the information about the target, using open sources and popular basic or advanced techniques and aim to understand and map the target’s infrastructure, vulnerabilities and “open” doors from where might is achievable to enter.

The reconnaissance stage can include passive information gathering from publicly available sources, like social media, public websites, online forums and generally open public information, as well as active information gathering using targets' and networks' scanning in order to identify weaknesses and “opened” channels to enter.

An attacker, in order to achieve the above, employ techniques like open-source intelligence gathering (known as OSINT), networks scanning and footprinting, and collect valuable information regarding the target’s infrastructure, technology, employees’ roles and responsibilities and possible vulnerabilities that may be exploitable. Taking all this information into consideration, the attacker builds his strategy in order to find the most suitable attack vector from which will be able to achieve a successful attack.

With all the above, an attacker can also identify the most valuable assets of the target, in order to adopt a specific way of attack and manage to get the most valuable returns, if the aim is the profit.

There are various ways to defend against reconnaissance techniques, by adopting proactive approach, to threat intelligence and situational awareness. Techniques like monitoring network scanning, or reconnaissance tools usage, can be implemented in order to face and mitigate potential risks before escalate in completed attacks. Usual measures of security like access control, employees’ training for the achievement of a higher level of awareness, as well as the implementation of Security Information and Event Management systems (SIEM) can detect and respond to reconnaissance techniques in real-time.

1.1.2.2. Weaponization

It is the second step of the cyber-attack lifecycle. It is the phase that the attacker will adapt his weapons into the form that they will be suitable to be used against the target’s infrastructure that was mapped during the previous stage. Tools and payloads that will be used in order to exploit the identified vulnerabilities are being created in this stage. All the intelligence that gathered during the previous stage, is converted into the means that the attack will launch.

The attacker will create a custom-made malicious malware, trojan, virus, worm etc. and will use it in order to exploit a specific vulnerability, misconfiguration or security gap. Also, social engineering methods will may allow the threat actor to inject the above custom-made files to the target’s environment and execute harmful actions. In order to initiate these attacks, fake websites or phishing emails can be created, custom made, in order to lead the target or the victim to click a link, give its personal data or download malicious executables.

In general, in this stage the attacker must have a deep, detailed understanding of the target’s environment and infrastructure. Depending that environment, the attacker can use exploit kits that may are applicable to the target’s infrastructure and is very possible to achieve vulnerabilities’ exploitations and complete a successful system violation.

In this stage also a “Command and Control” center is created. It is a common tactic for threat actors, creation of these centers in order to communicate with compromised systems, after the successfully delivery of the malicious payloads.

Against this stage's techniques, there are many measures that can protect a system, like having updated the systems and software of the infrastructure and having also patched them with the latest security updates, use threat detection mechanisms and also have the users trained and informed to keep them in a high level of awareness, to avoid phishing attacks and others social engineering tactics.

1.1.2.3. Delivery

The abovementioned "Delivery" that we mentioned in the previous stage consists the 3rd step of the cyberattack lifecycle. The attacker transmits the custom-made weapon to the target's system. The weapon includes the malicious payload that will be executed in the target's system and do all the job. The social engineering methods, in this stage will transfer the malicious mean. A phishing attack, a fake email with a malware attacked, a malicious website or even an infected usb that will be let intentionally in the working place of the target, is enough to transit us from preparation stage to execution. The question, if actions that will take place in this stage are successful, can be answered depending the level of preparation of the 2nd step. The better physical and normal the attack moves seem, the more possible to have a successful attack. The more countermeasures to bypass the security systems have taken, the more possible for the attacker to complete his method successfully. And finally, the biggest the experience of the attacker is, the most possible to bypass the security measures, using advanced techniques.

1.1.2.4. Exploitation

Now the real action begins. The previously transferred and delivered malwares, now is the time to be executed. In this stage, the attacker activates a connection and interaction with the target's system. He is actually "in" the system and implements techniques that will give him full access to the system with elevated rights. In this stage, the attacker, remotely tries to exploit vulnerabilities, software bugs, other misconfigurations and even zero-day vulnerabilities that in most cases the target's system administrator doesn't know about.

In this stage, if the attacker gains higher-level privileges, usually use them in order to go deepest to the system, creating an attack path to gain access to adjacent machines and finally reach to the internal most important assets of the system. One of the aims of an attacker, is to achieve persistent access to the compromised system. In order to do that, the attacker can install additional malicious software, handle system's configuration and install more than one back door to the system that will be in "listening" state, ready to establish a connection with the attacker any time he triggers it from outside. Of course, this stage's techniques can be faced with the implementation of IDS/IPS systems, firewalls and others, as well as with the usage of whitelisted software, suitable access control policies etc.

1.1.2.5. Installation

The fifth step of the cyber-attack cycle, the "Installation", known also as "Persistence", is the stage where the attacker establishes his presence in the compromised system. After the security gap or vulnerability exploitation by the attacker, and while the attacker intrudes into the system, the installation of malicious software like Remote Access Trojans or backdoors follows, in order to maintain control over the system. For an attacker this stage is very important as it is the point where he will become a "part" of the system that will keep having presence in it and control over it after the vulnerability correction, the security gap removal or even after a reboot of the system. Using techniques to avoid detection, an attacker can use rootkits to hide the malicious software or change system configuration to launch the connection, quietly and successfully, any time the system starts.

1.1.2.6. Command and Control

After the attacker takes control over the compromised system, he establishes a “Command and Control” system in order to remotely handle and control it. The command-and-control center (known as C2 infrastructure) consists a centralized control point from where the attacker can handle the compromised systems, and is able remotely to send malicious files, upload payloads, change the system’s configuration and execute malicious code remotely. From a compromised system, the attacker can also hit and enter to adjacent (or not) systems, and afterwards to another, and with this technique build an attack path, difficult to be detected but also investigated by the appropriate authorities. The attacker usually changes the C2 servers’ location as often as it is possible and also use stealth techniques in order to stay invisible, like encrypted communications etc. For the compromised systems, therefore, is difficult to detect the malicious persistence and advanced techniques or countermeasures are required. Behavior analysis or anomaly detection software or systems should scan and watch the system in order to reveal the activity of a malicious “resident” of the system.

1.1.2.7. Impact

In this final step of the attack lifecycle the threat actors have successfully breached the network and have ensured their presence. Now is the time for the attackers to achieve their primary goals of the attack, from financial theft to disruption of services, while during this phase they leverage the access and control they have previously gained

1.1.3. Examples of the attack lifecycle being implemented on infrastructures

Considering the above categorization of the attack lifecycle steps, we indicatively mention the following incidents as examples of attacks that began from the internet (Network) side and in each case exposed services found by the adversaries.

-Equifax Data Breach [3]

Back in 2017, the American multinational credit reporting agency “EQUIFAX” was breached due to multiple well-known vulnerabilities that were exploited. Specifically, the most critical security gap that was exploited was one of the “Apache Struts” service which is an open-source web application that run in the systems of EQUIFAX. The attackers exploited the CVE 2017-5638 vulnerability of the above service and that allowed them to execute code remotely. This successful network side attack caused significant financial and reputational damage to the above company which agreed to pay several hundreds of millions in settlements with the U.S. authorities and affected consumers.

-Russian retail chain 'DNS' confirmed hack after data leaked online [4]

Russian retail chain DNS (Digital Network System) revealed that a data breach exposed the personal information of both customers and employees. DNS is the second-largest computer and home appliance store chain in Russia, as it has over 2,000 branches. The hackers who allegedly were from Ukraine, exploited a security gap in DNS's IT systems where one or more network assets with critical vulnerabilities were compromised. The breach was disclosed after the hacking group called 'NLB Team' began leaking DNS's data on a hacking forum. This data included full names, usernames, email addresses, and phone numbers of 16 million customers and employees.

-Hellenic National Post Services (ELTA) get struck by ransomware [5] [6]

The Hellenic Post (ELTA) experienced a significant cyberattack in 2022, from hackers who kept ELTA's services in hostage. Its outdated network assets in the IT systems, exposed critical security gaps whose exploitation opened the entrance for the adversaries to enter into the internal network. The attack began from the internet side from where a malicious code infected an ELTA workstation using an http reverse shell technique. Then the attackers built the attack path and managed to reach a critical server. Then, used this access to infiltrate ELTA's internal network, infecting additional systems. The malicious code executed on the compromised parts of the systems led to the massive encryption of main servers and ELTA terminals.

-Marriott International Data Breach [7]

Back in 2018 over 500 million guests' information, such as names, addresses, passport numbers and other personal details, were exposed after a breach in the reservation system of Marriott's multinational law firm. More specifically, back in 2014 an attacker gained access from the internet side to the "Starwood" hotel network and affected a machine long ago since the attack was committed. The compromised machine had access to the internet and also had administrative rights, in order to let the employees commit changes to the website of the hotel. After a couple of years, the Starwood hotels were acquired by the "Marriot International" and they carried along their vulnerable systems which the Marriott didn't examine deeply before they proceed with the acquisition. The vulnerable assets were running outdated windows server operating systems and also the ports for remote connections using the unsecure Telnet protocol and the Remote Desktop protocol, were left open and exposed to the open web.

After studying the phases of the attack lifecycle as well as their implementation in real world examples like the ones we briefly saw above, the need for proactive solutions that will help security professionals implement those steps before the attackers do, is crystal clear.

1.1.4. Solutions and Initiatives led by the cybersecurity community

Based on this fact we identify the enumeration of information derived from the internet layer of current infrastructures as the core challenge of this thesis. The above incidents highlight the obvious need and importance to implement new security measures and countermeasures, to enrich the security policies with new protective methods, procedures and Frameworks by evaluating their current state and therefore to ensure that the exposed digital surface of an infrastructure would not become an object of criminals' play game.

In today's digital era, the frequency and sophistication of cyber-attacks are escalating at an unprecedented pace, posing severe threats to individuals, businesses, and even national infrastructures. Cyber attackers constantly devise new strategies to breach security defenses and exploit vulnerabilities, not only in software and hardware but also in human behavior. From phishing schemes and ransomware to Distributed Denial of Service (DDoS) attacks and Advanced Persistent Threats (APTs), the spectrum of offensive tactics is both vast and ever-evolving.

In response to this growing threat landscape, structured frameworks and methodologies have become crucial. Red and Blue teaming frameworks, such as the MITRE "ATT&CK" that we already mentioned above as well as the MITRE "D3FEND" Framework offer systematic approaches to emulate adversarial tactics and develop corresponding defense mechanisms. Additionally, methodologies like threat hunting, anomaly detection, and machine learning analytics are increasingly essential in identifying and countering sophisticated threats, helping to protect against the ever-changing nature of cyberattacks [8] [9].

Below in a few sentences we mention some cyber security initiatives such as developed **Taxonomies** of the cybersecurity community, we enumerate some categories of **attack methods** and, we note the importance of having structured cybersecurity strategies by explaining some popular **Frameworks,** that play a key role in cybersecurity landscape

1.1.4.1. Taxonomies & classifications

In addition to the above, taxonomies and classifications for the cyber security community, like the Common Platform Enumeration (**CPE**)[4] and others, play a crucial role by providing standardized identifiers for hardware, software and operating systems because reduces errors in communication between cyber security professionals. They enable different security tools or platforms to work together seamlessly, allow the automation of vulnerability management along with classifications like the Common Vulnerability and Exposure (**CVE**)[5], helps cyber security specialists in sharing cyber threat intelligence more efficiently, and helps in creating accurate and detailed reports for compliance with industry regulations and standards.

Other similar taxonomies are the Common Weakness Enumeration (**CWE**)[6] which classifies and categorizes software weaknesses and vulnerabilities, the Common Attack Pattern Enumeration and Classification (**CAPEC**)[7] which provides a catalog of common attack patterns, Common Vulnerability Scoring System (**CVSS**)[8] which is a standardized method in rating the severity of security vulnerabilities that helps in the prioritization of the vulnerabilities significance, by taking into account the impact of a possible attack by exploiting a vulnerability in combination with the possibility for this attack to happen, and lastly the National Vulnerability Database (**NVD**) which with a combination of all the above, maintains a repository of vulnerability management data [10].

1.1.4.2. Methodologies of attacks

As far as the existing malicious **methodologies** of attacks, cyber-attackers employ a variety of methods to intrude into systems, exfiltrate data and cause damages. More often we meet methodologies like phishing, where the attacker deceive the victims into revealing sensitive private information or installing a malware, ransomware which are malicious software that encrypts the victim's data and demanding usually a (cryptocurrency) payment as ransom in order to release the data captivity and Distributed Denial of Service attacks (DDoS) which actually is the creation of enormous amounts of traffic, usually bigger than a system can handle, in order to render it unusable. APTs (Advanced Persistent Threats) represent another method where the adversaries establish their presence in the target systems in order to steal data or disrupt operations. It is a multi-stage attack type including an initial access, the building of an attack path in the network, maintaining persistence in order to keep 'working' in the network even after a reboot or a system update and various techniques implementation in order to hide the digital traces of the attack and avoid detection. Attackers can use a combination of tools, in more cases custom-made ones like custom malwares, or even legitimate software behind from which they hide their actions. In these cases, the techniques like threat hunting, anomaly detection and others that implement AI and Machine learning algorithms are used to identify and mitigate potential threats before they finished the job that they were created for.

1.1.4.3. Frameworks

In response to the above, various **frameworks** have been developed in order to win the battle against the cyber threats using structured, efficient and comprehensive defensive strategies. Red teaming

[4] *https://nvd.nist.gov/products/cpe*
[5] *https://www.cve.org/*
[6] *https://cwe.mitre.org/*
[7] *https://capec.mitre.org/*
[8] *https://www.first.org/cvss/*

frameworks are in the frontline of these strategies as they simulate adversarial attacks to identify vulnerabilities.

Red teamers benefit from these frameworks such as the **MITRE ATT&CK** framework which provides in details a structured matrix of tactics and techniques used by adversaries and therefore gives the opportunity for real-world attacks emulation. Similar to this is the “Cyber Kill Chain” framework, developed by Lockheed Martin which offers a structured approach for defenders to get a deep understanding of the attack lifecycle (similar steps referred above) [1].

On the other hand, blue teamers benefit from the **MITRE D3FEND** framework which is a detailed knowledge base that provides structured approach to implementing defensive techniques against the techniques that described by the ATT&CK framework. Focuses on defensive countermeasures against cyberthreats and in fact, works as complementary to the ATT&CK framework [8].

About how important are these defensive strategies to be deployed nowadays, known high-profile cyber incidents can persuade us about it, such as the WannaCry ransomware attack and the SolarWinds breach, which have exposed various critical security gaps and vulnerabilities that underscored the need of having advanced network reconnaissance tools. All these incidents and others with similar significance, **some of which were mentioned above,** and frameworks such as the MITRE framework that appear in the form of campaigns consisting of specific tactics (whose steps included in the attack lifecycle that referred above), gave a boost to the global cybersecurity community and led to the development of new technologies and strategies against the cyber threats that show no mercy any more.

1.2. Motivation

Taking into consideration all the above, we realize that we have to choose among plenty of solutions, frameworks, taxonomies etc. and actually, although we have a plethora of tools that exist by far, usually we are not sure which of them can be used in combination with another. And even better, it is not clear if a tool can be combined in automated tasks with others.

Moreover, in our point of interest, the Reconnaissance stage, which actually is the phase where these tools work, there are many steps that an attacker passes through, and of course the same goes for a cyber security specialist who tries to reveal the security gaps and map the Attack Surface of a potential target. The problem is that with the existing tools, in each step of this stage, the cyber security specialist has to use various tools, implement various techniques, and manage to combine them in order to get the needed results.

In addition, when we are talking about the reconnaissance stage from the network “side”, we mean a vast number of different techniques that a cyber security specialist can implement. The choices that the specialist has to make in each step, depends on the results that occur from each step and after their evaluation. The existing tools usually do not fill in the implementation gaps that are created due to the need of automation of the procedure.

What motivated us to develop this thesis’ tool is the need to have a tool that starts from a simple **domain name** and after the stages, the checking and of course in each step the results’ evaluation, to lead to final results which include **open ports** and of course **exposed running services.**

Additionally, the lack of wide **orchestration of existing tools** in order to pass through all these steps committing all the necessary checks automatically was a lack in the collection of existing tools.

Also, we saw that in the existing tools there was an implementation gap in the **results’ visualization** procedures. That created an undeniable need to generate robust results, easy to interpret into more digestible results.

Finally, what we considered as a lack in the field of the existing tools, is that only few of them achieved a **connection** of the final results to any kind of **OSCTI Databases**.

In the implementation of the tool of this thesis, on the contrary to what existing until now, we **orchestrate well known, robust and always updated tools** into a simple running tool. We tried to embed all the necessary checks in order the tool to interact with the user as less as possible. And if any result found, we give the user the choice to decide the depth of the port scanning, according to the machine resources, with a simple keystroke. In a few words we actually tried to achieve domain analysis, subdomain analysis, IPs, ports and services scans, and finally reach **from a simple domain name to the running services, with a single command.**

An addition to the above was the **automated visualization of the final results**, into a much more digestible graph, much more understandable, which would fill in the implementation gap of many of the existing tools.

Finally, the last part of our implementation was to achieve a correlation between the final results to the existing OSCTI databases. For that, **we connected the results to the CPE taxonomy**, which is a widely used standard for hardware and software identification and categorization. After that, the user will be able to identify the machine, the OS or the software that exists behind the found services

1.3. Contribution

The tool developed in this thesis primarily addresses the first step of the cyberattack cycle, known as "Reconnaissance." This tool leverages open-source techniques and tools to gather information, starting with a given domain name. It then uncovers related subdomains, identifies their corresponding IP addresses, and finally scans for open ports to reveal the services running behind them. While the main focus of this tool is on the reconnaissance phase, it can also touch on other stages of the cyberattack cycle, such as "Weaponization" and "Exploitation." The gathered information can be used to craft custom exploits in the Weaponization stage. Additionally, if the tool identifies any known vulnerability, an attacker could potentially use established frameworks with pre-built payloads to exploit these weaknesses, gaining unauthorized access and compromising the target during the Exploitation stage.

In the context of this thesis, given that the potential attack surface is exceedingly vast and difficult to fully cover through comprehensive reconnaissance and enumeration, our focus is on evaluating the exposed web surface of infrastructures. This area is particularly crucial as it represents the first step in launching attacks against an infrastructure. To facilitate this analysis, we divide our examination into the following layers:

1.3.1. From Domains to Running Services

In this sub-chapter we will enumerate the main steps of the developed tool. that starts from the target-domain and reaches to any revealed running service, delivering pure results in simple text.

1.3.1.1. Analysis of domains-subdomains

The tool that is described in this thesis in its first step analyzes the investigated domain, in order to reveal known public or unknown sub-domains that correspond to that. It applies known basic and advanced techniques, and uses professional open source developed tools in order to reach from a simple domain to the subdomains and the IP addresses they correspond

1.3.1.2. Analysis of IP Addresses

This tool can handle the IPv4 and also the IPv6 addresses that occur from the first step, identify and remove duplicates and also identify the IPs that consist local IPs and remove them, as they are accessible only internally in a system (locally) and do not communicate with others outside the local networks they are assigned. Finally in this step we get a clean “list” of unique, global IPv4 and IPv6 addresses that refer to websites or services of the same organization, ready to be used for further examination in the 3rd step of this tool, regarding the running services that run behind them.

1.3.1.3. Analysis of ports-services

This thesis' tool, in the 3rd step, tries to achieve a mapping of the attack surface relating to a body. It detects the open ports with as much accuracy it can, according to user's choice of how many ports wants to be examined and then makes an effort to detect the type, version and other information regarding the running service found to be running behind the open port.

1.3.1.4. Set of connected results

In this layer, the outcomes that were exported during the previous steps which are, the revealed open ports, along with the IP address that each one belongs, the type (IPv4 or IPv6) of each address, the service name and some more additional information, compose the final results form in a text file format with comma-separated values, which is nothing more than a universally accepted and easy to be processed CSV file.

1.3.2. Graph generation & visualization

As we realize, the csv results that we finally gain from the previous steps, are not easily interpretable, especially if they consist a large dataset. The information that we gain, although is all there, it is not clearly understandable from a simple user at first glance. Therefore, the csv results file is inserted into a graph database system (Neo4j graph dbms), formatted into an easy-to-understand graph and visualized in a form that makes obvious the critical parts of the results that the user needs [11].

1.3.3. Connection of services to OSCTI

In this tool we make an effort to connect the outcomes to Open-Source Cyber Threat Intelligence, in order to allow the user to identify potential threats by correlating the results with known attack patterns or vulnerabilities. The open ports (which are visualized as specific nodes in the final graph) and their corresponding network services along with their versions (which are both included in the graph as node properties), are crucial indicators of possible existing common vulnerabilities in outdated software versions or misconfigurations that can be identified by OSCTI. In addition, certain protocols of services (also appear as node properties) are frequently targeted by attackers. Lastly, the CPEs (Common Platform Enumerations) from which specific software, hardware or operating systems can be identified, are also included in the graph results of this tool and consist important elements to connect the results with already known vulnerabilities and security threats.

1.4. Structure of the document

The contents of this thesis are organized into 6 chapters which include various subsections, each of which addresses key aspects of the findings.

In **chapter 1** we give some general definitions of known terms, like “Domain name”, “IP address” and network “ports”, while describing in few words some of the current taxonomies and frameworks on which various modern Cyber Security models are based. We explain the Cyber Kill chain framework and the steps of its Attack lifecycle that was taken into consideration during the implementation of this thesis.

In **chapter 2** we enumerate Related Work that we found during our research to have been previously done, including Taxonomies, IT components and threats mapping models, as well as categorizations of devices, networks and threat actors. We also give a sample of the work that has been done so far, from developed cybersecurity tools, scientific research papers as well as OSCTI research.

In **chapter 3** we start from a short overview of our methodology in developing the tool and then we analyze in details each step of it, with a few explanations of enumerations and reconnaissance techniques that globally followed in each step.

In **chapter** 4 we analyze one by one, files and commands detailed, the behaviors of our tool, the steps from which it passes and all the operations and the checks that does until we reach to the final “pure” results. We call them “pure” because they are in a simple csv file, which means in simple text. Additionally, we describe the results visualization procedure and the conversion of text results, to a digestible well-formed graph.

In **chapter 5**, we simply test and validate all the operations of the tool, as mentioned in chapter 4, showing for each step the results, whether they were simple text-terminal output or they were graphs with visualized results.

Finally, **chapter 6** is an ending summary, in which we mention the conclusions we come to, including the tool’s earnings from its implementation and the benefits in comparison with the usage of one of the pre-existing tools. In addition, we enumerate some limitations we faced during the development or the validation of the tool, while we indicatively suggest what future work can be done, continuing the research on the same matter.

2. Related Work

In our exploration of notable efforts in risk assessment, we will highlight a select number of them, with a focus on well-established taxonomies. These taxonomies are used to evaluate different device, network, and user models that are prevalent in the cybersecurity domain, particularly in the areas of risk assessment, infrastructure management, and administration.

2.1. Taxonomies, Categorization and mapping Models of IT assets, Components and Threats

In the following three sections of this chapter, we enumerate various efforts to orchestrate **IT assets** and threats categorization and management. We meet the above in the forms of guidelines which helps in the assets' management, and also in guides for assets, devices and threats classification, depending on their significance for an organization and their sensitivity. In addition, we approach indicatively from the **"Networks"** side the efforts of organizations to model and categorize the network threats, such as the annually published "ENISA Cybersecurity Threat Landscape". Last, we enumerate some of the efforts for categorization of **Threat Actors** and model their behavior into databases, as well as model human tasks in order to enforce them in cyber threats detection.

2.1.1. Assets/Devices

CPE (MITRE)

As we have already mentioned above, CPE is a structured naming scheme for IT systems, software and Operating systems. Includes formal name formats and an official version of a CPE dictionary is occasionally updated and provided in XML format to the general public. The update is implemented to the dictionary mainly when new names added or a modification is necessary.

NIST SP 1800-5 [12]

NIST Special Publication 1800-5, also titled as "IT Asset Management -Special Publication 1800-5" focuses on improving the IT assets management within an organization's cybersecurity policy or framework. It consists a guide for actionable and practical IT asset management ensuring that they can control and secure their devices and systems. The publication includes best practices, tools and methodologies to establish a robust IT management system, crucial for maintaining cybersecurity and mitigating risks from potential threats that can find fields to deploy their action, because of unmanaged or unknown IT assets. The document is not truly a formal taxonomy for assets or devices in cybersecurity but covers aspects to the identification, categorization and management of IT assets. Its primary purpose is to enforce the sense of visibility and control over IT resources which in turn supports better risk management. With the implementation of the guides of this publication the suggested IT management process can create the field for the better and broader cybersecurity objectives to be accomplished.

Guidelines in threats and Assets (ENISA)

The "Guidelines[9] in Threats and Assets" by European Union Agency for Cybersecurity is not strictly a taxonomy for cybersecurity but it consists a "Technical Guideline on Threats and Assets" which provides the National Authorities with the necessary glossary terms to communicate about significant threats and network assets. The threats and assets that are included in this report are based on past incidents, as they reported by the National Regulatory Authorities to ENISA and the European Commission. The document helps organizations to understand deeply various types of threats that might face and categorize assets in order to detect the critical ones that need protection. ENISA's

[9] https://www.enisa.europa.eu/publications/technical-guideline-on-threats-and-assets

guidelines assist organizations in developing robust security polices and strategies and improve their cybersecurity posture. The guidelines include components of taxonomy such as classification or categorization guides, but its primary purpose is to offer practical advice on threat facing and asset management in order to provide best practices to enhance cybersecurity measures.

IBM Asset Classification framework [13]

IBM's approach[10] on asset classification, involves guidelines for categorizing assets based on their criticality and sensitivity for an organization which in turn helps in prioritizing security measures. These guidelines begin with the identification of all assets within an organization, including hardware, software and data. Then, classifies the found assets based on factors like 'Criticality' which is the importance of this asset in organization's operations, 'Sensitivity' which evaluates the level of sensitivity of the data or information this asset touches in the organization's infrastructure and 'Regulatory Requirements' which considers if any requirement exists in handling and managing this type of assets. Therefore, IBM's asset classification framework gives the ability to the organization to implement tailored security polices and measures according to the significance of its assets.

2.1.2. Networks

CISCO [14] [15] [16]

Cisco doesn't have a 'networks model' dedicated to cybersecurity but provides comprehensive frameworks and methodologies that integrate cybersecurity principles into its models and architectures. Cisco emphasizes a holistic security integration into every level of networking and IT infrastructure.

Its methodology, called "Cisco Secure Development Lifecycle[11]" for developing secure software and hardware is applied to its products' production from the beginning to the end of this cycle.

In addition, "Cisco Security Reference Architecture[12]" provides an overview of cisco's security portfolio while offers a range of security solutions that can be integrated into various network architects in order to protect against threats and vulnerabilities.

The "Cisco Cyber-Security Framework Guidance[13]" provides frameworks and best practices for implementing cybersecurity measures on networks, helps organizations in assessing their cybersecurity readiness and in detecting and response to potential threats.

ENISA

Every year, the European Union Agency for Cybersecurity publishes its annual report regarding the state of cybersecurity threat landscape across various domains, including networks. In its 11th edition (October 19th 2023), ENISA's report contains categorizations of threats and attack techniques as well. About the threats, ENISA sorts them in a taxonomy, which includes 8 groups. "Ransomware", "Malware", "Social Engineering", "Data Threats", "Availability Threats - DoS", "Availability Threats – Internet Threats", "Information Manipulation" and "Supply Chain Targeting" threats are the categories. About the techniques, Zero-day exploits, Hacktivism, DDoS Attacks towards IoT, AI-enabled disinformation and deepfakes are some of the many trends used by threat actors. Finally, talking about Threat Actors, ENISA in its ETL 2023 report, uses a taxonomy which generally categorizes them in "State-nexus Threat Groups", "Cybercriminals", "Hackers-for-hire" and "Hacktivists" [17] [18].

[10] *https://www.ibm.com/docs/hu/iis/11.7?topic=data-classifying-asset-according-its*
[11] *https://www.cisco.com/c/en/us/about/trust-center/technology-built-in-security.html*
[12] *https://www.cisco.com/c/en/us/products/security/cisco-security-reference-architecture.html*
[13] *https://www.cisco.com/c/en/us/products/security/cybersecurity-framework-guidance.html*

NIST Cybersecurity Framework [19]

The Cybersecurity Framework of NIST (current version CSF 2.0) includes 5 major pillars that support the creation of a holistic and successful cybersecurity plan. The framework includes the ‘identify’, ‘protect’, ‘detect’, ‘respond’ and ‘recover’ procedures, which work concurrently and continuously and let other cybersecurity risk management methods to be built on them. NIST has its “Risks and Threats” section that includes resources of threats and risks like ransomwares, spywares, phishing and website security. Explaining each one of the above categories, NIST has also a “Risk Management” section that describes and explains the significance of managing security risk. It describes common misunderstandings about security risks, such as those caused because of lack of information etc., and also composes infographics that show the importance of managing cyber risks for manufacturers. It introduces common security techniques and activities that manufacturers should consider before they sell their products to their customers.

2.1.3. Human Entities – Users (Threat actors)

Intel-TAL [20]

A nice effort has been done by intel, that led to the formation of a Threat Agent Library (TAL). The lack of industry standards or reference definitions of threat agents and the different concepts that people have, caused a major problem of very different interpretation even of the most common agents. In fact, there is a wide spectrum of threats of all potential agents that Risk Managers must assess and characterize, in order to assess the overall risk to information assets.

Therefore, Intel since 2005 has been developing the “Threat Agent Library”, which contains archetypes that represent main categories of threat agents. This effort aims in creation of a set of standardized definitions and descriptions of threat agents that will help in information exchange, quick agent identification and more efficient building of defending systems appropriate for specific threats. The Threat Agent Library contains a simple taxonomy of eight attributes that define uniquely each agent and in addition contains for each one of them, its common tactics and actions. Starting with a general characterization of agents, TAL separates them in Hostile and Non-Hostile, depending on their intention to harm or not Intel’s information assets. We could indicatively mention some agents that are characterized as non-Hostile, such as “Untrained Employees” or “Info Partners”, and also some Hostile agents such as “Anarchists”, Government Spies”, “Terrorists”, “Thieves”, “Vandals” etc. For each one of the above two categories of agents, TAL lists their attributes in details but also groups them into more general categories. We could mention indicatively the agents’ attributes category “Outcome” (Acquisition/Theft, Damage, Tech Advantage etc.), the “Skills” (none, minimal, Operational etc.), the attribute “Objective” (copy, deny, destroy, damage etc.) etc. While the attributes of the agents remain stable over time, the strength of their threat may change, and also an agent may use different tactics or select different types of targets. Therefore, Intel rates the threat of each agent, depending on its recent activity and reviews/updates this rating, in order to keep TAL up-to-date.

MITRE’s Threat Group ATT&CK Framework [9]

MITRE’s ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) framework is a modelized database which includes a categorization of adversary behaviors. In its critical knowledge base, we find a depiction of various phases of an adversary attack lifecycle and the points that consist targets for it. In the ATT&CK model there are tactics and techniques that we meet in adversary actions, as they have been understood by offensive and by defensive specialists as well, of the cybersecurity domain. Of course, the categorization of the above actions as well as defending measures for them are not missing from the framework. MITRE’s ATT&CK framework database is continuously enriched by open sources and publicly available material, which can be analysts’ and threat hunters’ contribution, public threat intelligence and incident public reports.

The popular ATT&CK Matrix includes techniques of all phases of an attack cycle, starting with “Reconnaissance”, “Resource Development”, “initial access” and proceeding to the last phases, “command and control” and the “impact” which consists the techniques that an adversary uses to disrupt availability or compromise integrity. In addition to the above, the ATT&CK taxonomy consists a common language for red teamers, pen testers or cybersecurity analysts and defenders when referring to adversarial behaviors. Analytic methodologies of analysts track activity clusters by a common name in the security world. MITRE's ATT&CK uses the term "GROUP" when referring to an adversary activity cluster and maps some of the techniques that the GROUP used, according to publicly available reports.

Hamsters [21]

Following, we explore a method for modeling human tasks to detect cyber threats, specifically utilizing the HAMSTERS (Human-centered Assessment and Modelling to Support Task Engineering for Resilient Systems) notation. This approach is supported by tools and represents tasks performed by users interacting with operating systems.

In diverse maritime supply chains, the presence of various infrastructures and human actors with different business and operational goals can lead to potential errors by information system operators, thereby generating significant threats. The HAMSTERS method focuses on modeling human tasks to identify cyber threats among operators in supply chains.

The paper discusses applying this technique within the MITIGATE risk assessment methodology. It provides an example scenario in maritime transport supply chain services involving multiple organizations where users perform diverse tasks. Attacks may originate externally, targeting assets within the supply chain, or internally, with insiders intentionally deviating from planned tasks to execute malicious actions. Human errors also pose substantial risks within supply chains.

HAMSTERS modeling produces hierarchical descriptions of user tasks, represented as tree diagrams where nodes are tasks or ordering operators. Each task may involve manipulating data or objects for execution, enhancing HAMSTERS' capability to describe information, knowledge, objects, and devices comprehensively.

Assessing IOT Enabled Attack Paths [22]

Nobody would care if our smart bulb or smart temperature sensor were hacked. But what if through them someone could gain access in our deeper home network where our personal or business laptops were connected? And what if the attacker could get access to them through a vulnerable, old application which might we had even forgotten that we had it installed on our system? The above paper describes the potential of a critical breach in case of an IOT device is hacked and through that, the attack path would lead to a critical industry, transportation or even medical back-end system. The survey that has been done, examines various IOT-enabled cyber-attacks found since 2010, including an attempt to provide a taxonomy based on an assessment of real and verified world IOT-enabled attacks. The methodology that is followed includes analysis of representative attacks which demonstrate attack paths against critical targets and models the adversaries using three main characteristics: Their **access to IoT** device, their **capabilities** and their **motivation** for the attacks. The access to IoT device is distinguished in a) physical access (insider or outsider) and b) logical access (privileged or unprivileged access). The required capabilities model the skills and resources required by an adversary to successfully attack are distinguished in a) Technical skills and b) resources. The motivation actually is the reason or the potential gain that an adversary gains from the attack and could be a financial profit, cyber-terrorism, hacktivism etc. The stronger the motivation it is the more possible it is for the adversary to select a specific target.

2.2. Building Blocks for the development of Reconnaissance and Enumeration tools

The internet hosts a plethora of cybersecurity tools (both free and paid) tailored for searches, investigations, and reconnaissance. These tools are essential for cybersecurity professionals and researchers seeking to analyze and understand potential threats. Free offerings include open-source utilities like Nmap for network scanning and OSINT frameworks such as Maltego for gathering information from public sources. Meanwhile, commercial tools like Shodan provide advanced capabilities for scanning internet-connected devices. Whether searching for vulnerabilities, gathering threat intelligence, or conducting forensic analysis, these tools play a critical role in applying cyber defenses and ensuring proactive security measures. We enumerate some of them as follows:

DIG tool (Linux)

Dig (Domain Information Groper) is a powerful command-line utility used to query DNS name servers. The dig is actually a Linux OS embedded command that lets you retrieve information about various DNS records, such as host addresses, mail exchanges, and name servers. System administrators frequently use it for troubleshooting DNS issues due to its flexibility and ease of use. Additionally, it supports AXFR queries to check if a remote DNS server permits zone transfers.

[*AXFR is a protocol for zone transfers, which are segments of DNS servers' databases, allowing domain data replication across multiple DNS servers. This can reveal subdomains of the root domain.*]

HOST

On Unix-like operating systems, the host command is a DNS lookup utility used to find the IP address associated with a domain name. It can also perform reverse lookups to find the domain name associated with an IP address. The host command facilitates DNS lookups, enabling the conversion of domain names to IP addresses and vice versa.

Nslookup

Nslookup is a network administration cross-platform command and is used for querying the Domain Name System (DNS) to obtain the mapping between domain names and IP addresses, or other DNS records. It queries the specified DNS server and retrieves the requested records associated with the provided domain name. These records contain information such as the domain name's IP addresses.

DNSENUM2[14]

It is a multithreaded Perl script designed to enumerate DNS information for a domain and discover non-contiguous IP blocks. The script can retrieve the host's address, nameservers, and MX records (A, NS, MX records). Additionally, it can perform AXFR queries on nameservers and obtain extra names and subdomains through Google scraping (-www site:****). Furthermore, it calculates C class domain network ranges, performs WHOIS queries on them, brute forces subdomains from a file, conducts reverse lookups on net ranges, and writes the IP blocks to a file.

DNSRecon

DNSRecon[15] is a Python port of a Ruby script tool with capabilities for checking all NS Records for Zone Transfers. It can enumerate general DNS records for a given domain, including MX, SOA, NS, A, AAAA, SPF, and TXT records. The tool performs common SRV Records Enumeration (server services for specific domains and ports, as per RFC 2782), checks Top Level Domains (TLD) Expansion, and brute forces subdomains and host A and AAAA records using a domain and a wordlist. Additionally, DNSRecon can perform PTR Record lookups for a given IP range or CIDR,

[14] *https://github.com/SparrowOchon/dnsenum2*
[15] *https://www.kali.org/tools/dnsrecon/*

which are pointer records stored under a reversed IP address for security purposes. It also checks a DNS server's cached records for A, AAAA, and CNAME records, using a provided list of host records in a text file.

NMAP [23]

Nmap ("Network Mapper") is a free, open-source utility designed for network discovery and security auditing. It can be used for tasks such as network inventory, managing service upgrade schedules, and monitoring host or service uptime. Nmap employs raw IP packets in innovative ways to identify available hosts on a network, the services (including application names and versions) they offer, the operating systems (and their versions) they run, and the types of packet filters or firewalls in use. While it is capable of rapidly scanning large networks, it is also effective for scanning single hosts. Nmap is compatible with all major operating systems, with official binary packages available for Linux, Windows, and Mac OS X. The Nmap suite includes several tools in addition to the command-line Nmap executable such as Zenmap, an advanced graphical user interface and results viewer, Ncat, a versatile tool for data transfer, redirection, and debugging, Ndiff, a utility for comparing scan results and Nping, a tool for packet generation and response analysis.

FIERCE

Fierce[16] is a DNS reconnaissance tool designed to identify non-contiguous IP space. Written in Perl, it provides extensive options for performing DNS enumeration by scanning domains. Fierce can discover available domains and their NS records, and it attempts DNS transfers (AXFR queries). With the `-wide` parameter, it can also scan an entire network C class after identifying any matching hostnames within that class C. However, this comprehensive scanning process can be time-consuming to complete.

SUBLIST3R

Sublist3r[17] is a Python tool designed for enumerating subdomains using open-source intelligence (OSINT). It assists penetration testers and bug hunters in collecting and gathering subdomains for their target domains. Sublist3r uses various search engines such as Google, Yahoo, Bing, Baidu, and Ask to enumerate subdomains. Additionally, it employs services like Netcraft, Virustotal, ThreatCrowd, DNSdumpster, and ReverseDNS. To enhance its capability of finding more subdomains, Sublist3r integrates the Subbrute tool, which uses brute force with an improved wordlist.

SubBrute

SubBrute[18] (subdomain-bruteforcer) is a project aimed at developing the fastest and most accurate subdomain enumeration tool. A key feature of SubBrute is its use of open resolvers as proxies to bypass DNS rate-limiting[19] . This approach not only helps in circumventing rate limits but also adds a layer of anonymity, as SubBrute avoids sending traffic directly to the target's name servers.

Subfinder

Subfinder[20] is a specialized subdomain discovery tool that identifies valid subdomains for websites through passive online sources. It features a straightforward modular architecture and emphasizes speed. Subfinder focuses exclusively on passive subdomain enumeration, excelling in this area. The

[16] *https://github.com/mschwager/fierce*
[17] *https://github.com/aboul3la/Sublist3r.git*
[18] *https://github.com/TheRook/subbrute.git*
[19] *https://www.us-cert.gov/ncas/alerts/TA13-088A*
[20] *https://github.com/projectdiscovery/subfinder.git*

tool is designed to adhere to the licenses and usage restrictions of all passive sources, maintaining a consistently passive model. This approach ensures its utility for both penetration testers and bug bounty hunters. Developed in Go, Subfinder requires the installation of multiple service API keys to operate effectively.

The Harvester

The Harvester[21] is a user-friendly tool crafted for the reconnaissance phase of red team assessments or penetration tests. Its primary function is to conduct open-source intelligence (OSINT) gathering to assess the external threat landscape of a domain. The tool collects a range of information including names, email addresses, virtual hosts, IPs, subdomains, open ports, banners, employee names, and URLs. It achieves this by querying multiple public resources such as search engines and PGP key servers. The Harvester is essential for gathering comprehensive data during security assessments.

AttackSurfaceMapper

AttackSurfaceMapper [24] (ASM) is a reconnaissance tool designed to enhance the attack surface of a target using a blend of open-source intelligence and active techniques. By inputting one or more domains, subdomains, and IP addresses, ASM employs various methods to identify additional targets. It enumerates subdomains through brute-force and passive lookups, identifies other IPs owned by the same network block, and discovers IPs with multiple domain names pointing to them.

Once the target list is expanded, ASM conducts passive reconnaissance activities. This includes capturing screenshots of websites, generating visual maps of network architecture, checking for credentials in public breaches, performing passive port scanning using services like Shodan or Censys, and scraping employee information from LinkedIn. ASM thus provides a comprehensive view of potential vulnerabilities and attack vectors for security assessments.

KnockPy

Knockpy[22] is a Python3 tool created to efficiently enumerate subdomains of a target domain using dictionary attacks. It utilizes internal wordlists and allows users to specify a custom path for a text file containing additional wordlists. Knockpy also leverages search engines and online services such as Google and Virustotal to gather subdomain information. This multifaceted approach enhances its ability to comprehensively identify subdomains associated with the target domain.

DNSmap

Dnsmap[23] is a tool that scans a domain to discover common subdomains using either its built-in wordlist or an external one. The internal wordlist includes approximately 1000 words in English and Spanish, covering common subdomains such as ns1, firewall services, and smtp. This allows it to automatically search for subdomains like smtp.example.com within example.com. Dnsmap can save its results in CSV and human-readable formats for later analysis and processing.

Altdns

Altdns is a DNS reconnaissance tool designed to discover subdomains based on specified patterns. It operates by taking input from two lists: one containing words commonly found in subdomains (such as test, dev, staging), and another list comprising known subdomains associated with the domain in question. Using these inputs, Altdns generates a large output of "altered" or "mutated" potential subdomains that may exist. This output is saved and can be used with DNS brute-forcing tools. After generating these altered subdomains, Altdns offers the option to resolve them in a multi-threaded

[21] *https://github.com/laramies/theharvester*
[22] *https://github.com/guelfoweb/knock/releases/tag/7.0.1*
[23] *https://github.com/pagvac/dnsmap*

manner. The results of this resolution can then be saved to another file for further analysis or use with other tools. This process enables comprehensive exploration and enumeration of potential subdomains, enhancing the effectiveness of subsequent security assessments or penetration testing activities.

Assetfinder

Assetfinder[24] is a tool written in Golang that retrieves subdomains associated with a target domain. It leverages numerous publicly available data sources to discover these subdomains. Additionally, Assetfinder offers an option to uncover not only subdomains directly related to the target domain but also related domains.

Hacktrails

Hacktrails[25] is a reconnaissance tool written in Golang that interacts with the SecurityTrails API to retrieve domain information. Hacktrails specializes in subdomain enumeration as its primary function. It is designed with modularity in mind, allowing it to be easily integrated into workflows with other tools for comprehensive reconnaissance tasks.

Gau (GetAllUrls)

Gau is an open-source tool written in Golang designed for discovering URLs associated with a target domain during reconnaissance. It specializes in extracting URLs from a specified domain, aiding in the process of subdomain enumeration. Gau is freely available and serves as a valuable asset for security professionals and penetration testers aiming to gather comprehensive information about a domain's web presence and potential attack surface.

Garud

Garud[26] is a script tool which combines a plethora of other known and widely used tools in order to enumerate all the subdomains of a given target domain and creates an appropriate report after filtering all the results.

The Recon-ng Framework

Recon-ng[27] is a comprehensive reconnaissance framework specifically designed for conducting open-source web-based reconnaissance. Unlike other frameworks, Recon-ng focuses exclusively on web-based open-source intelligence gathering and is not meant to compete with existing tools in other areas of cybersecurity. This framework is built on a completely modular architecture, which allows for easy integration and development of new modules. This design makes it accessible for Python developers of all levels to contribute and extend its functionality. Recon-ng provides a robust environment for performing reconnaissance activities, enabling security professionals and penetration testers to gather valuable information about web-based targets efficiently.

Amass [25]

Amass is an open-source tool designed for network mapping and discovering attack surfaces by employing a variety of information gathering techniques, including active reconnaissance and external asset discovery. It utilizes its own internal mechanisms and integrates seamlessly with external services to maximize its effectiveness and efficiency in data collection. The tool places a

[24] https://github.com/tomnomnom/assetfinder
[25] https://github.com/hakluke/haktrails
[26] https://github.com/lc/gau
[27] https://github.com/lanmaster53/recon-ng

significant emphasis on gathering DNS, HTTP, and SSL/TLS data through specialized techniques and multiple API integrations. Additionally, Amass leverages web archiving engines to unearth valuable data from the depths of the internet, including forgotten or obscure information repositories. This comprehensive approach allows security professionals and penetration testers to conduct thorough reconnaissance and expand their understanding of potential attack surfaces.

Various GUI Tools

dnsdumpster

Dnsdumpster[28] is a domain research tool that can discover hosts related to a domain. Finding visible hosts from the attacker's perspective is an important part of the security assessment process.

GOOGLE toolbox – dig tool[29]

Is a web tool in a visual environment which practically can perform DNS lookups but without the ability to perform AXFR queries.

SecurityTrails.com - Attack Surface Intelligence

Attack Surface Intelligence[30] (ASI) offers an external perspective of an organization's digital presence and infrastructure. It enables teams to take a proactive approach in identifying, prioritizing, and mitigating risks across the entirety of their attack surface.

Maltego

Maltego[31] is a powerful open-source intelligence and graphical link analysis tool designed for gathering and connecting information during investigative tasks. It operates as a Java application compatible with Windows, Mac, and Linux operating systems. This versatile tool is utilized by a diverse array of users, including security professionals, forensic investigators, investigative journalists, and researchers. Its capabilities allow users to visualize and analyze relationships between various data points, enabling deeper insights into complex investigation

Firecompass[32]

The tool operates continuously, employing sophisticated reconnaissance techniques akin to those used by threat actors, to index information from the deep, dark, and surface web. Using these techniques, it automatically identifies an organization's dynamic digital attack surface. This includes uncovering previously unknown exposed databases, cloud storage buckets, leaked code, exposed credentials, vulnerable cloud assets, open ports, and other potentially risky digital assets.

Shodan

Shodan (www.shodan.io/) is famous as the world's first search engine dedicated to Internet-connected devices. It actively tracks devices accessible directly over the Internet, offering a comprehensive view of all exposed services. This capability enables users to search and analyze a wide range of Internet-connected devices, providing valuable insights into the security posture and potential vulnerabilities of these devices.

[28] *https://dnsdumpster.com/*
[29] *https://toolbox.googleapps.com/apps/dig/*
[30] *https://go.recordedfuture.com/attack-surface-intelligence-demo-request*
[31] *https://www.maltego.com/*
[32] *https://www.firecompass.com/*

Censys

Censys[33] is recognized as a leader in Attack Surface Management, leveraging extensive and proactive monitoring to comprehensively search and monitor your digital footprint. Its capabilities extend far beyond conventional methods, providing deep and broad coverage to identify and manage your organization's digital assets and potential attack surfaces.

Spiderfoot

Spiderfoot[34] is a persistent cyber reconnaissance tool designed to automatically query more than 100 public data sources, also known as OSINT (Open-Source Intelligence). This tool specializes in gathering intelligence on various entities such as IP addresses, domain names, and email addresses, among others. During reconnaissance operations, Spiderfoot offers flexibility by allowing users to activate specific modules based on the type of information required. This modular approach enhances efficiency and customization, ensuring targeted and comprehensive data collection from diverse online sources.

Datasploit

DataSploit[35] is an open-source intelligence collection (OSINT) tool. It is a simple way to dump data for a domain or other piece of metadata. It consists an OSINT Framework to perform various reconnaissance techniques on Companies, People, Phone Number, Bitcoin Addresses, etc., aggregate all the raw data, and give data in multiple formats. It is useful to collect relevant information about a target in order to expand the attack and defense surface very quickly.

[33] *https://search.censys.io/*
[34] *https://github.com/smicallef/spiderfoot*
[35] *https://github.com/datasploit*

2.3. OSCTI Research

The following papers are selected among various papers in which we find significant efforts for cybersecurity and cyber defense mechanisms enhancement, using innovative methodologies or frameworks. The result of the following is the necessity and reassurance we get about the critical role of cyber threat intelligence (CTI) and the implementation of comprehensive cybersecurity frameworks [26].

Security KG

It is a system for automated OSCTI (Open-Source Cyber Threat Intelligence) gathering and management, which collects OSCTI reports from various sources. It uses AI combined with NLP technologies and extracts high-fidelity knowledge about behaviors of threats, ending in a well-designed information-knowledge graph. The remarkable here is that it has the ability to handle the diversity of different OSCTI reports. Provides a wide coverage of entity and relation types to MODEL THE THREATS. It is consisted by few components, such as crawlers for OSCTI reports collecting, an extensive backend system for OSCTI reports management, a UI with the output as a graph which dynamically adopts to the gathered results and interacting with users etc. It is important to pay attention to the **ONTOLOGY** that is constructed by the Security KG, in which there is a categorization of the OSCTI reports, into three types: malware reports, vulnerabilities reports and attack reports. The above reports, as they come from specific but different vendors, contain information for various information concepts like threat actors, techniques, tools, etc. and therefore, SecurityKG creates **entities** (models) that contain relationships between them. The lists of Threat Actors, techniques and tools is the one constructed by the MITRE ATT&CK.

Finally, the system exports a knowledge graph which consists the visualized model of an ontology (that includes nodes as entities and connections between them), which represent various types of security knowledge [27] [9].

Towards Collaborative Cyber Threat Intelligence for Security Management

The next paper touches the critical matter of diversity in security mechanisms that implement various security policies, regarding the access control management. It refers to the importance of a direct interaction of security specialists with each security policy update, using collaborative knowledge in the latest cyber activities.

In this model, a new ontology of correlation between access control policies and security reports is described. In this project, although common formatted enumerations of types of malwares, vulnerabilities or exploitations are used (NIST, MITRE's etc.), a more mathematical model is also approached, in a stricter way, and refers to the STIX (Structured Threat Information Expression) standard which contains the above formatted concepts, and is nowadays the most widely used language for CTI description. We notice that **"Threat-Agents"** are considered as **objects,** that describe the profile of an adversary that may belong to multiple entities. This object may exploit Vulnerabilities, execute attack-patterns, use tools and deliver malwares. According to the above, a CTI Report contains multiple STIX objects (STIX Domain Objects and STIX Relationship Objects) known as "properties". Each one of them contains information about a certain cyberattack and as a result, a collection of SDOs or SROs consists a final report-description of a cyberattack [28].

BRON Data Graph

In this work we have an attempt for connections' creation, among MITRE's ATT&CK MATRIX of Tactics and Techniques, NIST's CWEs, CVEs, and CAPECs. The above paper introduces the BRON graph, which is an aggregate data graph with bi-directional relational paths, that represents sources as entries and relations with the ability to combine these sources. This model can identify attack patterns, tactics and techniques that exploit known CVEs and can provide information that is linked with these CVEs but has been retrieved by public sources (NIST's National Vulnerability Database, etc.). We meet relational bidirectional links between independent sources, such as Attack Techniques <-> Attack Patterns, Attack Patterns <-> CWE Weaknesses and CWE <-> CVE Vulnerabilities etc.

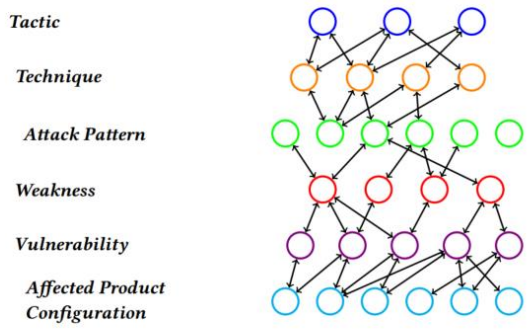


Figure 2 The BRON Graph [29]

In the above schematic figure of BRON Graph, Nodes represent the entries from the sources while Edges are the relations between them. In conclusion, the model of the above tool aims in representing and depicting a more schematically approach of the "Threat Landscape" and therefore it can achieve an evaluation of how complete and accurate the above sources provide us with the appropriate information [29].

<u>A Practical Approach to Constructing a Knowledge Graph for Cybersecurity</u> [30]

In the next paper, the authors aim in creating a new framework which will be a cybersecurity knowledge base and will be created by a specific procedure. Information obtaining, ontology creation and knowledge database construction will be the three steps of the creation procedure. Cyberattacks are complex and varied, making them hard to detect and predict. This paper explores combining knowledge graphs with cybersecurity to create a cybersecurity knowledge base using a quintuple model. Machine learning extracts entities and builds an ontology, while new rules are deduced with formulas and the path-ranking algorithm. The Stanford Named Entity Recognizer (NER) is used to train an extractor for useful information, showing promising results for future cybersecurity applications.

<u>Ontologies for Network Security and Future Challenges</u> [31]

Recent efforts have focused on creating ontologies for specific aspects of network security. This review identifies these aspects, including threats, intrusion detection systems (IDS), alerts, attacks, countermeasures, security policies, and network management tools. The study classifies existing ontologies, analyzes key issues and challenges, and proposes a three-stage process: inputs, processing, and outputs. It highlights the need for new ontologies to cover various aspects of network security, aiding in management tasks and guiding future research.

<u>Cybersecurity Ontology to Support Risk Information Gathering in Cyber-Physical Systems</u> [32]

This paper aims to define an extended cybersecurity ontology to aid in information gathering and risk assessment for complex cyber-physical systems. The ontology integrates data from various cybersecurity datasets, public security reports, network security tools, and information about threat actors and users. To demonstrate its effectiveness, a portion of the ontology was implemented as a knowledge graph using Python and Neo4J. The ontology was validated through two scenarios: filling gaps in the National Vulnerability Database using a logistic classifier, and identifying connections between security databases like NVD, CWE, CAPEC, and Intel-TAL.

3. Our Methodology

In this chapter we will describe in a concise explanation, our methodology and the stages we followed in order to develop our tool, by depicting schematically the tool's phases, followed then with a description for each step.

3.1. High-Level Overview

We illustrate schematically in a (flow) chart (figure 3), the stages that our tool gets through while it runs. As we already mentioned, the tool interacts with the user only where it is necessary and more specific in the point when the user has the choice to decide the depth of ports enumeration. For this, we have inserted various checkpoints where the tool decides its next operation according to the results. We also matched the stages in the diagram with the explanations that follows to the next subchapter.

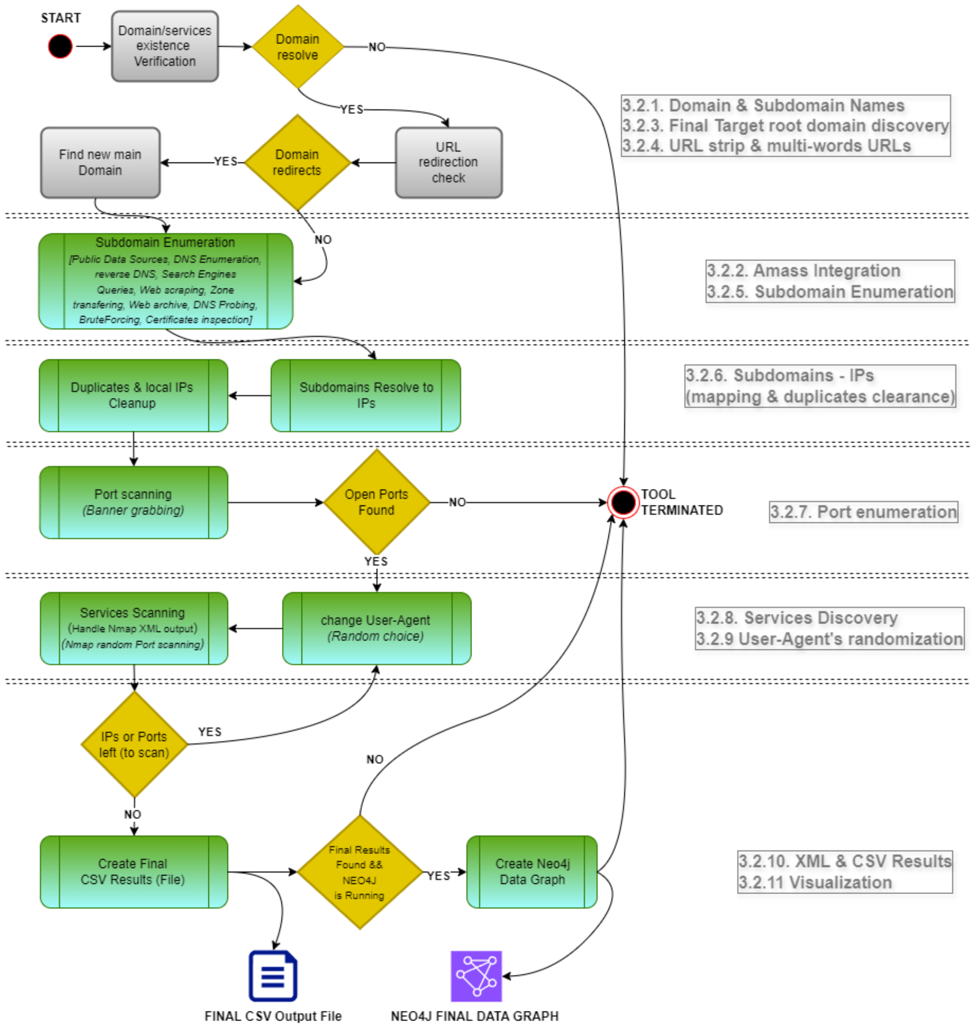

START
Domain/services existence Verification
Domain resolve
NO
YES
URL redirection check
Domain redirects
YES
Find new main Domain
NO
Subdomain Enumeration
[Public Data Sources, DNS Enumeration, reverse DNS, Search Engines Queries, Web scraping, Zone transfering, Web archive, DNS Probing, BruteForcing, Certificates inspection]
Subdomains Resolve to IPs
Duplicates & local IPs Cleanup
Port scanning (Banner grabbing)
Open Ports Found
NO
TOOL TERMINATED
YES
change User-Agent (Random choice)
Services Scanning (Handle Nmap XML output) (Nmap random Port scanning)
IPs or Ports left (to scan)
YES
NO
Create Final CSV Results (File)
Final Results Found && NEO4J is Running
NO
YES
Create Neo4j Data Graph
FINAL CSV Output File
NEO4J FINAL DATA GRAPH
3.2.1. Domain & Subdomain Names
3.2.3. Final Target root domain discovery
3.2.4. URL strip & multi-words URLs
3.2.2. Amass Integration
3.2.5. Subdomain Enumeration
3.2.6. Subdomains - IPs (mapping & duplicates clearance)
3.2.7. Port enumeration
3.2.8. Services Discovery
3.2.9 User-Agent's randomization
3.2.10. XML & CSV Results
3.2.11 Visualization

Figure 3. Flow chart of the thesis' tool procedure

3.2. Detailed methodology analysis

The Reconnaissance tool has been developed by taking into consideration some basic stages that a penetration tester (or even an attacker) passes through, during his efforts to initiate his malicious actions by mapping the structure and infrastructure of the target that has decided to attack.

3.2.1. Domain & Subdomain Names

As already mentioned, the first step to reach an organization from the Internet attack vector, is to reveal and find out the most or, if it is possible, all "**Domain names**" as well as subdomains, behind of which the infrastructure unfolds. Main domain names, called "root" domains, may include from none to several subdomains that are their 'subcategories', under which unfold particular websites, web-components or web services, that all together compose the online presence of the organization.

The first step that will give us a first sight about how wide is that online exposed surface, is to enumerate those revealed domains and subdomains and try to export from them others that have not been public announced (or somewhere posted) or published, and this can be achieved by various techniques, each one implemented in a different stage and level of the procedure.

The **DNS enumeration** technique, is the process of gathering information about DNS records associated with a domain. DNS records consist the 'phone book' of internet and contains obviously useful and important information. This technique can be **passive**, where the attacker has no interaction with its target, as well as **active** where the attacker should obtain information about DNS records directly by interacting with the target. In the passive one, techniques like search engines' queries, certificate grabbing and analysis, social media postings etc. are implemented in order to obtain and reveal useful information about the existed domains and subdomains of an organization. On the contrary, in the active method, the attacker may query specific DNS records, uncover subdomains by conducting reverse DNS lookups and may exploit zone transfers, using advanced hacking or pen testing tools and strategies, in order to commit a successful reconnaissance of domain and subdomain information.

In order to build the tool that is described in this thesis, we had to include various techniques for domain information gathering. In order to achieve that, many **issues** had to be faced, e.g. many of the tools that are being used for years and are supposed to serve on specific techniques and methods, have various conflicts or incompatibilities among them because are developed based on different platforms, some cannot be applied simultaneously from the same attack vector or even many of them were deprecated and not updated or upgraded to include modern infrastructures. Therefore, we had to find tools that are continuously being maintained by official bodies, will not show incompatibilities or inabilities due to future abandonment and of course will fulfill our goals criteria in order to be integrated to our tool.

3.2.2. Amass Integration [25]

The seeking of such tools led us to an opensource network mapping and asset discovery tool, "**Amass**", that is a free software by the **OWASP** (Open Worldwide Application Security Project) developed with a license for redistribution and modification under the terms of the Apache License. The fact that Amass integrates various different techniques and in other cases we would have to use a dozen of the abovementioned tools, was the major reason to choose it instead them.

3.2.3. Final target root domain discovery

But before using it, we had to face the issue where the investigated domain name is not the one it seems and actually redirects to another domain or subdomain as soon as it gets visited. And of course it becomes a little more complicated, if a URL inserted and not a 'clear' domain name. In order to automatically our tool get over this issue, firstly we send an initial http request to the target domain.

Then, according to the reply headers that we receive back, we analyze it to find out if any service is running behind the domain name (alive domain) and if it is the final domain that concern us or redirects to another one (30x redirection http status code).

3.2.4. URL strip & multi-words URLs

Afterwards, our tool, strips the inserted URL in order to leave only the root level domain name along with its TLD. In this point we had a new issue, concerning domain names with TLDs which include more than 2 words. This issue could not be overcome with the usual techniques of coding or scripting languages, like Regular Expressions usage, but we had to find another way to check and validate the existence of current TLD or part of it, that match our domain. The solution came from a library we used (python library 'tldextract') that was designed to do exactly the above and validate the TLD, according to the latest **Public Suffix List**[36] (maintained by Mozilla, as a community resource).

3.2.5. Subdomain Enumeration - Amass techniques in action [25]

Then comes the abovementioned OWASP AMASS tool and subdomain enumeration starts. In fact, the Amass tool **uses a vast number of techniques** in order to find out which subdomains exist and are related to the domain-target. In this Thesis we used the "amass" tool version 3.19.2.

The Amass as an information gathering tool that cooperates with public and non-public third-parties, retrieves data and information using them, and uses various other techniques as well. Some of them are, **Passive DNS sources** queries from where gathers dns data (like VirusTotal, PassiveTotal etc.), **Web Archives** queries from which historical references to subdomains are gathered from web archived data (e.g. Wayback machine), **Certificates grabbing** using certificates transparency logs' checks, where subdomains with SSL/TLS certificates can be revealed, **Websites and Search Engines** scraping, **Reverse DNS** lookups in IPs associated with an organization, **Brute Force** subdomain enumeration using a subdomain names dictionary, **APIs' calls** with which amass integrates, **Whois** queries and of course **Zone Transferring** (AXFR protocol) which although is a difficult technique to be succeeded nowadays due to modern security standards, it can reveal the whole existing list of subdomains related to a domain.

The main features of Amass are the "**intel**" and the "**enum**" subcommands. The amass intel subcommand gathers basic data and information regarding a target domain using various tools and gathers basic whois information. Actually, the tool of this thesis will not use intel because it is proved that many of the results gathered by amass intel, despite they lay behind the same ip address or maybe exist in the same autonomous system (ASN), actually have only in common with the target some shared resources of the same hosting machine and actually it concerns totally different and independent entities, hard to sort out in an automated way.

On the contrary, the amass enum subcommand is mainly used by the tool of this thesis because it does quite successfully the job that is mainly needed for our tool, by performing a plethora of requests and mapping around the wide internet (-passive flag operation) in order to reveal potential related information, but also commits data retrievals with active techniques (-active flag), including DNS enumeration, brute force the subdomains and usage of subdomain list files (wordlists).

Although Amass tool has the ability to integrate external services through API keys, the need of having online accounts to all of these third-parties led us to omit them and actually allow amass to use only the totally free ones through the enum subcommand and therefore the tool of this thesis will be enough simplified to be used by all level users. Of course, the usage of the aforementioned API keys in amass, could be integrated to this thesis' tool in future improvements.

[36] https://publicsuffix.org

3.2.6. Subdomains – IPs (mapping & duplicates clearance)

After the reveal of the subdomains related to the target domain, the amass tool continues with DNS resolution, where the identified domains and subdomains are resolved and mapped to their corresponding ip addresses. All the results, stored in a txt file in a specific format in order to be easy to integrate to the next steps of our tool.

Moreover, the tool uses the generated report and commits some processes on it in order to clear the field where the next steps will intervene. Thus, in the file-list of discovered domains, subdomains and ip addresses, performs private and duplicates' cleanup and separates them into a file-list which contains only the domain-subdomain names and a file-list which contains only unique ip addresses, either IPv4 or ipv6.

3.2.7. Ports enumeration overview

Next, comes the step of ports scanning, where the above unique ip addresses one-by-one are scanned by applying a low-level technique called "banner grabbing". In this method, the grabber (network Socket) connects to a port and reads the initial data (banner) sent by the running service. In comparison with the most common and usual http requests method, this one works with a wider range of services and is not limited to (http) web services that the http request does. Although some services may not send a banner or may require specific commands to initiate a banner reply, the most important reason to use this technique is not only it is quite quicker but also it does not reveal information about the initiator of the connection attempt. On the contrary, when an http request is initiated, the “User-Agent” header must be sent and inform the target port about the client software making the request (e.g. web client, web scrapper, bot etc.) and in some cases, defensive systems like Web Application Firewalls (WAF), Intrusion Detection/Prevention Systems (IDS/IPS), Rate Limiting Applications, DDoS Protection Services or Access Control Mechanisms etc. can prevent a client to complete an http request and connect with a port, especially after repeated attempts to connect with multiple ports.

One more issue that we faced during the implementation of this thesis' tool is that in our testing attempts in port scanning using http requests method, some targets, used Intrusion Prevention or other defensive systems in order to pretend that ALL ports were opened, and therefore we could not export useful knowledge about the exposed surface of the target. That possibly happened because either the requests to various ports initiated from the same User-Agent was a “red flag” for the target's defensive systems, or the lack of port randomization in http requests was a “red flag” too. For all the above and without being assured about the totally effectiveness of the banner grabbing technique, using it was a one-way for our tool.

3.2.8. Services discovery

Once the previous steps completed, we get a list where there are stored the IPv4 and IPv6 addresses for which open ports found. Then, the tool goes through the ips of the list and for each one address the respectively discovered open ports are examined using the nmap tool (version 7.80). The nmap (Network Mapper) tool is an open-source network discovery tool which can also be used for security auditing, although it is often used by malicious users. Using nmap we can scan networks and identify running services behind ports as well as recognize respectively the devices along with their configurations. Nmap is also able to detect open ports, operating systems' details and is powerful enough to reveal and identify known vulnerabilities about the discovered assets.

As for the vulnerabilities, nmap uses the Common Platform Enumeration (CPE) standard to enumerate and identify discovered devices, OS or running software. CPE is a structured naming scheme for informative systems and software, developed by MITRE Corporation and since 2015 belongs to National Institute of Standards and Technology (NIST). The found CPEs are related with known vulnerabilities through processes involving service detection and version detection. This thesis' tool, includes the CPEs that occur from the nmap scanning, in the final report.

3.2.9. User-Agent’s randomization

In the tool, when nmap is used, we automatically and randomly choose a different User-Agent and use it for each open port scan, even concerning ports that belong to the same ip address. In fact, the tool selects randomly a user-agent from a pool of user-agents that we created and is included to out tool’s files as a text file. The above user-agents correspond to modern and updated network clients or web browsers and they pretend to be in mobile devices, computers, mobile phones etc. In addition, the open ports are not scanned in order, but randomly in order to avoid detection from defending systems.

3.2.10. XML & CSV results

After all these measures that help this tool not to be blocked from security countermeasures, the tool proceeds by exporting the partial results of each nmap execution, in a universal XML format. XMLs are plain text files that use custom tags to structure data in a hierarchical form. They can be read by humans as well as by software and generally are preferred in cases where information and data exchange between different systems is required.

Document Type Definition (DTD) files are documents which include sets of rules that define and verify the structure and format of an XML document. Each one of the above exported XML files is automatically processed and the data retrieved from them used in order to build the Final Results file. In order to do this automatically, successfully and independently of what the exported XML file includes or omits, this process is done according to the nmap DTD[37].

Final results are formatted into a unique Comma Separated Values file (CSV) which includes as value types, several tags from the abovementioned XMLs, e.g. ‘hoststatus’, ‘address’, ‘addresstype’, ‘portnum’, ‘CPE’, ‘OStype’ and more.

All of the above procedures, we decided to implement using Linux BASH code along with Python language code. **Bash** is a UNIX shell and command language, easy for commands execution, scripts’ running or automation of tasks in order to interact with the Operating System. **Python** is a high-level interpreted programming language, very expressive, very supportive in multiple programs and also very powerful [33] [34] [35].

3.2.11. Visualization

In the final stage of the procedure, all the discovered results are depicted in a **Neo4j Graph** in order to get visualized and become easier to understand and read by the user. The Neo4j is a modern Graph Database Management System (GDBMS) written in Java, that has been continuously developing since 2010. It stores the data into labeled (or not) “nodes” and creates the relationships among them, as “edges”. In addition, data is stored as node or edge attributes, grouped according to which entity they concern. In order to be accessible from software written in other programming languages, the “Cypher query language” has been developed and consist a declarative graph query language that is used through a transactional http endpoint [11].

[37] https://nmap.org/book/nmap-dtd.html

4. Implementation

In this chapter, we describe the development and implementation of the abovementioned tool, designed to enumerate running services behind a target domain, and generate an appropriate report. The tool has been developed in Linux OS (Ubuntu 22.04.4 LTS) using BASH shell & command language (GNU bash, version 5.1.16) and Python (Python 3.10.12) programming language, takes a domain name as input and after a comprehensive analysis identifies running services in the depth of the user's choice and outputs the findings in a CSV format, providing structured and easy interpretable summary. Additionally, in this tool we use the "Cypher query Language" of Neo4j database management system in order to achieve a visualization of the final results. Actually, using this GDBMS as mentioned above, the results appear in a graphical form, easier to be read and understand. We use the "Neo4j Browser" version 5.21.0, a tool for writing and running graph queries using Cypher query Language", while the "Neo4j Server's" on which runs the DBMS is version 5.21.0.

In fact, this tool is consisted by several Bash shell and Python scripts that cooperate in order to reach its goal. In this implementation, a variety of methods, techniques as well as libraries were leveraged in order to achieve an effective domain analysis. Bash shell handles system interactions and automated tasks (scripts) whereas Python scripts perform more complex data processing and output formatting. Our thinking was in using a combination of them, to ensure flexibility but also efficiency, allowing the tool to do what has been created for, as much automated as it gets [33].

4.1. Initial tool execution – Flags

Starting with the first main file of the tool, the **'recontool2.sh',** is the first executable file-script that initiates the program. The above name, was assigned in order to be easier to explain but also more understandable afterwards, when other files that serve in particular procedures come.

In this script we notice the flags that the execution of this file needs. The '**d**' flag defines the domain name that is inputted to the tool. Is the most important flag and of course it cannot exist sole in the command but needs a domain name as argument, which will be the object of the domain reconnaissance. Moreover, the next flag is the '**a**' flag, which is passed to the 'Amass' tool in order to force it commit an active (while 'passive' is the default method) domain investigation. In the 'active' mode the Amass tool, as already mentioned, performs data retrievals with active techniques, including DNS enumeration, brute force the subdomains and usage of subdomain list files (wordlists), and is not limited to open sources' information queries. The '**v**' flag does nothing more than enabling the verbose mode for the amass tool, in order to print (to the console) for the user more detailed information for each of its actions as it runs. The next option we have is the '**h**' flag which can be used alone without any argument or other flags. This flag when is used on the **./recontool2.sh** prints some information about the usage (flags) of this tool. Actually, when the 'h' flag is used, the '**usage**' function is called and executed and then the tool ends and does nothing more. The 'usage' function exists in the file '**func.sh**' which contains various functions, executed from more than one files of this tool. In order to avoid repetitions, we have stored some commonly used functions in the 'func.sh' and we simply import it as source (command "source ./func.sh") in the beginning of each bash script (.sh file) needs it. Our source code will be clearer and neater, as this method usually decreases significantly the number of rows of programming language code.

The syntax of the 'usage' function is in the figure 4:

```
# Function to display script usage options
usage() {
 echo
 echo "Usage: $0 [OPTIONS]"
 echo "Options:"
 echo " -h,      Display this help message"
 echo " -d,      insert the domain for investigation (required)"
 echo " -a,      Active mode enabled. Performs deep DNS enumeration techniques
like zone transfers, port scanning of SSL/TLS, certificates grabbing etc.
(*CAUTION: More than usually time needed)"
 echo " -v,      Enable verbose mode"
 echo " -b,      Brute force SubDomain Enumeration with default dictionary. Use
along with -w flag to use your own wordlist"
 echo " -w,      Path to use wordlist for sub-domains brute forcing."
 echo
 echo

 echo -e
"${BLUE}======================================================${ENDCOLOR}"
 echo -e "${BLUE}=================${ENDCOLOR}${RED} D O N E . . . !
${BLUE}===================${ENDCOLOR}"
 echo -e
"${BLUE}======================================================${ENDCOLOR}"

}
```

Figure 4. The “usage” Bash function

The entire code in the ‘func.sh’ file, as well as the programming code of all files that compose this tool, exist on **appendix A**.

But before the ‘usage’ function is executed, as soon as the first file run, another function which exists also in the ‘func.sh’, called ‘printTitle’ is executed. The code of this function is following in figure 5:

```
# Function to display script title and information
printTitle() {
 echo -e
“\n${GREEN}===============================================================${ENDCO
LOR}”
 echo -e “${RED}====== ${LIGHTYELLOWBACK}${BLUE}~Exposed Surface Reconnaissance~
Tool Version 1.0${ENDCOLOR}${RED} ======${ENDCOLOR}”
 echo -e
“${GREEN}===============================================================${ENDCOLO
R}”
 echo
 echo -e “${GREEN}-${RED}Author: ${BLUE}G.Tasios${ENDCOLOR}${GREEN}${ENDCOLOR}”
 echo
 echo
 echo -e “${GREEN}(${RED}WARNING! ${GREEN}Don’t forget to use a VPN
connection…)${ENDCOLOR}”

}
```

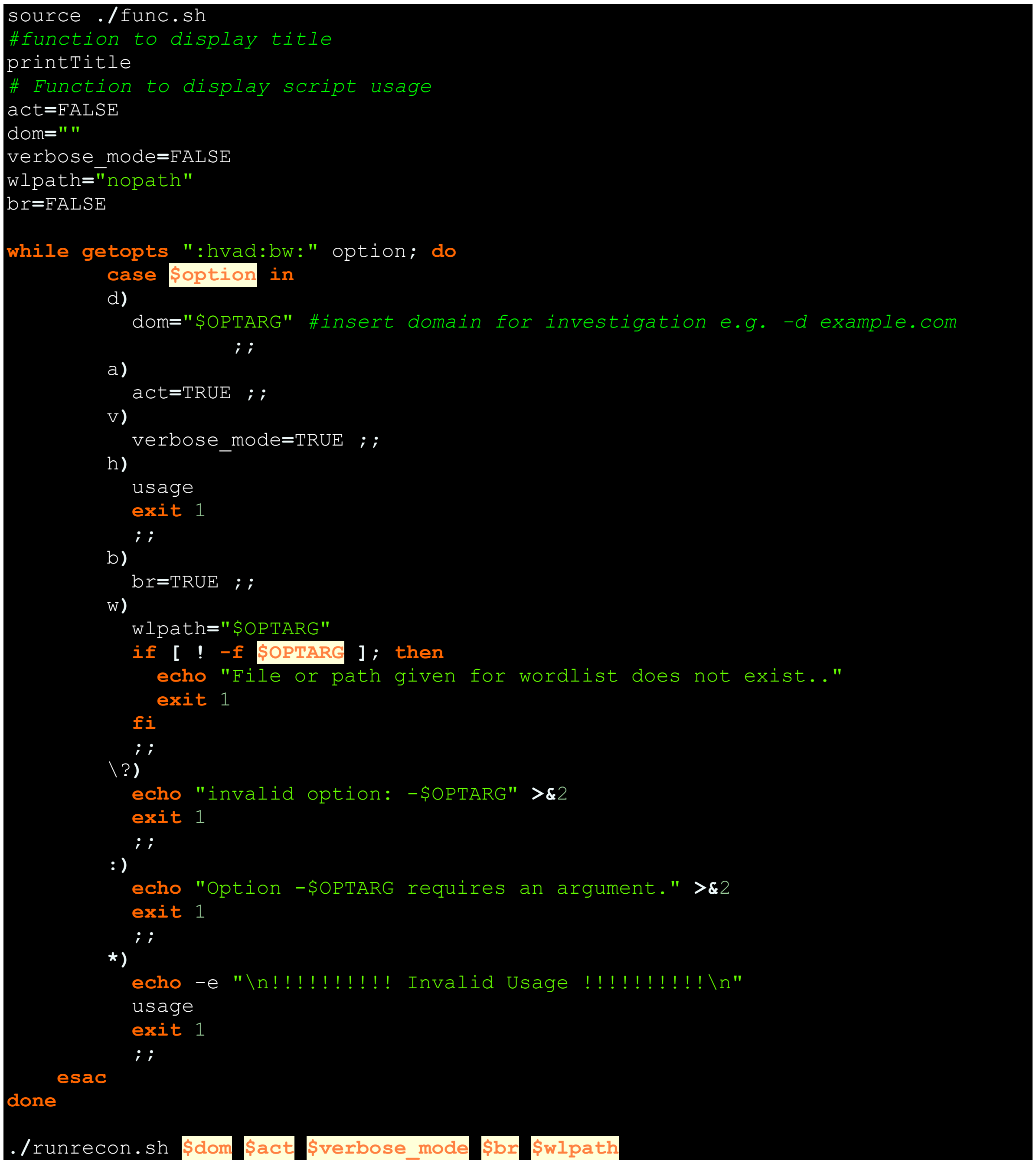
Figure 5. The code of “printTitle” function

The above function outputs the title of the tool as a banner, and is executed any time our tool starts.

Continuing with the last two flags, we have the ‘**b**’ flag which forces amass tool to perform subdomain brute force enumeration using predefined wordlist. In addition, with the ‘**w**’ flag we can specify a local wordlist file which can be used to the brute-force enumeration much more efficiently.

The entire Bash code of the file ‘recontool2.sh’ is as follows:

```
#!/bin/bash

# Author: George Tasios
# FileName: recontool2.sh
# The very first script that handles the flags (arguments)
```



```
source ./func.sh
#function to display title
printTitle
# Function to display script usage
act=FALSE
dom=""
verbose_mode=FALSE
wlpath="nopath"
br=FALSE

while getopts ":hvad:bw:" option; do
        case $option in
        d)
          dom="$OPTARG" #insert domain for investigation e.g. -d example.com
                  ;;
        a)
          act=TRUE ;;
        v)
          verbose_mode=TRUE ;;
        h)
          usage
          exit 1
          ;;
        b)
          br=TRUE ;;
        w)
          wlpath="$OPTARG"
          if [ ! -f $OPTARG ]; then
            echo "File or path given for wordlist does not exist.."
            exit 1
          fi
          ;;
        \?)
          echo "invalid option: -$OPTARG" >&2
          exit 1
          ;;
        :)
          echo "Option -$OPTARG requires an argument." >&2
          exit 1
          ;;
        *)
          echo -e "\n!!!!!!!!!! Invalid Usage !!!!!!!!!!\n"
          usage
          exit 1
          ;;
    esac
done

./runrecon.sh $dom $act $verbose_mode $br $wlpath
```

Figure 6. The Bash code of 'recontool2.sh' file

In the end of the file, we notice the command "**./runrecon.sh $dom $act $verbose_mode $br $wlpath**". This line actually runs the file 'runrecon.sh' while passes the flags that are user predefined and selected. This file in fact is the backbone of the entire procedure and of what our tool achieves, as it coordinates all the actions that will be taken then by almost all other files of this tool.

4.2. “Runrecon.sh” – The backbone of the tool

Proceeding with the **“runrecon.sh”** file, that “backbone” one as it was called, we meet again the importing of the “func.sh” file which contains functions that are going to be called and executed later. Flags from the file “recontool2.sh” passed as positional arguments to the “runrecon.sh” while it is executed. We meet the ‘dom’ variable created to insert the domain name, the ‘act’ variable that contains user’s choice whether to use active mode or passive mode for subdomain reconnaissance actions, the variable ‘verbose_mode’ which increases the verbosity of the output, the ‘br’ variable for the choice to brute force or not the found subdomains to reveal more of them and last the ‘wlpath’ variable which is used to define the path in the filesystem where a wordlist of domains is located and use it to the brute forcing actions.

In the beginning, in this file the function “checkInternetConnection” is called in order to let the tool verify if an internet connection exists. This function actually tries to connect to a popular website (which is almost impossible to find it down!!!) and if it fails, it terminates the tool execution and prints an appropriate message. This function exists in the ‘func.sh’ file, and its syntax is as follows:

```
#Checks internet connection. If it is down, stops the program
checkInternetConnection() {
  echo -e "\n\n${BLUE}Checking internet connection...${ENDCOLOR}\n"
  sleep 3
  wget -q --spider http://google.com
  if [ $? -eq 0 ]; then
    echo -e "${GREEN}Internet connection seems OK...${ENDCOLOR}"
  else
    echo -e "${RED}Internet connection seems down...\nCheck your connection and
try again later...${ENDCOLOR}\n"
    exit 1
  fi
}
```

Figure 7. The Bash code of ‘checkInternetConnection’ function

After that and if the above internet connection check succeeds, another function is called, the “**timer_start**”, which in a few words creates a timestamp and inputs it in a global variable. Afterwards the variable will be available to be used for time counting reasons. The above’s function syntax is in the figure 8:

```
timer_start() {
  declare -g start_time=$(date +%s)
  start_time_formated=$(date +"%H:%M.%S\"(%d-%m-%Y)")
  echo -e "\n---->${BLUE}Running started at: ${GREEN}$start_time_formated${ENDCOLOR}\n"
}
```

Figure 8. The code of ‘timer_Start’ function

Now, it is the point where more complexed and advanced python programming language scripts and files started to be executed. Starting with the “**check_url_redirect.py**”, is called from the previous file with the command ‘*python3 check_url_redirect.py $dom*’ which inserts as argument the domain name. Now let’s explain what python does and then will return to this point to continue with the rest of ‘runrecon.sh’ file

4.3. URL/domain validation-resolution-redirection checks

The above python script “check_url_redirect.py” runs its first function, the ‘**check_hostname**’ which checks and verify if the inserted domain name is in a valid format. It used the ‘validators’ python library which is designed exactly to validate forms of entities. The validators library is not included in the Python standard libraries but it is a third-party one that needs to be installed separately using pip. So, we installed this library, using the “*pip install validators*” command [36].

The syntax of this function appears in figure 9.

```
def check_hostname(my_domain):
    try:
        my_domain = my_domain.lower()
        valid = validators.domain(my_domain)
        if valid:
            print('\nDomain \"' + my_domain + '\" is a valid format.....')
        else:
            print('\nDomain didn\'t inserted or  is NOT a valid domain format')
            exit(1)
    except Exception as error:
        print('\nAn error occured while checking HOST (domain). Please try
again')
        exit(1)
```

Figure 9. The Python code of ‘check_hostname’ function

After a successful validation, the script proceeds with the next step, in which it verifies if the domain resolves to an IP address and as a result if a service runs behind it. Actually, in this point, the ‘Socket’ library of python’s is used and more specific the **‘socket.gethostbyname’** function is used. This function will succeed in resolving the domain to its corresponding IP address regardless of the specific services (web server, email server etc.) running behind that domain. In fact, this function performs a quick DNS lookup to map the domain name to its IP address. Therefore, the success of this function is independent of whether a web or other type of server are running on the domain, but it will still return the IP address if there is a valid DNS record for the domain. The code of the above is structured as follows in figure 10:

```
#FUNCTION to check if URL resolve to an IP
def domain_resolv(domain_to_resolv):
    try:
        r = socket.gethostbyname(domain_to_resolv)
        print("\nDomain \'" + domain_to_resolv + "\' resolved successfully to IP:
\'" + r + "\'\n")
    except socket.error:
        print("\nDomain didn't resolve... Script is stopped!!!\n")
        exit(1)
```

Figure 10. The Python code of the ‘domain_resolv’ function

After the last check to confirm if the domain resolves to an IP address, what follows is some checks if the domain redirects to another one, something that we often meet in real cases.

Using the ‘requests’ python library, the tool checks the actually response of the domain name to a simple URL request and if in the returned header the corresponding http redirection codes exist (301, 302, 307, etc.) the tool finds the real domain name where it ends. The above, is performed for http and for https protocol as well. Finally, whether the real domain is found after a redirection or not, it is passed as an argument to the Bash script “**pass_url.sh**” which is executed from within the python script, as a subprocess.

In the above bash script, the truly existing domain name stored in an environmental variable and after that stage, it is stored in a simple temporary text file for further processing. That file will be freely accessed by the rest bash and python parts of this tool that follow next, for practical reasons, and in the end will be automatically deleted.

The python code of the above as well as the Bash code of the last described bash script is as follows (figures 11 and 12):

```
try:
    resolved=False
    #Check HTTP requests
    try:
        r = requests.get(my_url, allow_redirects=True)
        print("\nHTTP Status Code:[" + str(r.status_code) +"]...")
        print("\nHTTP Redirections Codes History: " + str(r.history))
        resolved=True
    except:
        pass
    if(resolved == False):
        try:
            rs = requests.get(my_urls, allow_redirects=True)
            #The same for the https requests
            print("\nHTTPS Status Code:[" + str(rs.status_code) +"]...")
            print("\nHTTPS Redirections Codes History: " + str(rs.history))
        except:
            print("\nAn ERROR occured in 'http' handling or URL finaly could not resolve.
Please try again\n")
            exit(1)
    # checks if final url is http or https and prints it. Then it is passed to the bash
script 'pass_url.sh'
    if(my_url != r.url):
      print("\nFinal URL is: \'"+ r.url +"\'")
    elif(my_urls != rs.url):
      print("\nFinal URL is: \'"+ rs.url +"\'")
    #########
    if r.url:
      subprocess.call(['./pass_url.sh', r.url])
    elif rs.url:
      subprocess.call(['./pass_url.sh', rs.url])
    #print(r.headers)
except Exception as error:
    print("\nAn ERROR occured in 'http' handling or URL finaly could not resolve. Please try
again\n")
    exit(1)
```

Figure 11. The Python code of 'check_url_redirect.py' file which checks the target domain for redirections

(*the entire code of the above 'check_url_redirect.py' file exists on appendix A)

```
#!/bin/bash

# Author: George Tasios
# FileName: pass_url.sh

# Creation of an environmental variable which contains the passed domain (whether
http or https), from python file 'check_url_redirect.py'
echo -e "\nCreating necessary ENVIRONMENT Variables...."
export ENV_MY_URL=$1
echo
echo
echo -e  "\nCreating TEMP files..."
echo
```

```
echo
echo -e "('URLTEMP.txt' file.... )"

# Writes the 'url' which is stored in an environmental variable, into a txt file
printenv ENV_MY_URL > URLTEMP.txt

# The above txt file after it is used it will be deleted automatically
```

Figure 12. The Bash code of 'pass_url.sh' file that is called after the final target domain found

Now, returning where we left the '**runrecon.sh**' file, after the run of "check_url_redirect.py" from it, follows the "**checkPythonExit**" function. This function exists in the 'func.sh' file and runs right after the previous python script ends. It catches the exit status of the last executed command and checks if it's equal to '1'. If it is true, prints an appropriate message to the console and exits the script. In other cases, just let the tool proceed to the next commands. In the meanwhile, the 'checkPythonExit' function right before exit and terminate the tool (if it has to do it), calls the "printFooter" function, which prints a footer, a last "banner" down to the bash console. Both previous Bash functions are in the 'func.sh' file and their code is following in figure 13 and figure 14:

```
#catches the exit status of the previously executed commands. actually, gets the
#exit status code of 'check_url_redirect.py' file
checkPythonExit() {
if [ $? -eq 1 ]; then
    echo "Exiting Reconnaissance Tool....."
    printFooter
    exit 1
fi
}
```

Figure 13. The 'checkPythonExit' function that checks the output of the previous terminal results

```
# Function to display script footer when it is stopped or finishes its actions
printFooter() {
 echo -e "\n\n${RED}Deleting temporary files...  (e.g. url file,
etc...)${ENDCOLOR}"
 echo
 echo
 echo -e
"${BLUE}======================================================${ENDCOLOR}"
 echo -e "${BLUE}==================${ENDCOLOR}${RED} D O N E . . . !
${BLUE}===================${ENDCOLOR}"
 echo -e
"${BLUE}======================================================${ENDCOLOR}"

}
```

Figure 14. The Bash function that prints a footer of our tool each time the tool is terminated

Next, the function '**urltempcheck**' which is contained also in 'func.sh', is executed in order to perform a check if the URL that occurred from all the previous actions has been stored in the temp text file (URLTEMP.txt) in the previous step of execution of 'pass_url.sh' file. Clearly, the url contained in the aforementioned file, is firstly stored in the local variable FINALURL and then it is checked by the 'urltempcheck' function.

Then, if the url exists, the tool just proceeds to the next steps, whereas if it does not exist or if it exists in the text file but it couldn't be read successfully, prints an appropriate warning message and stops the program.

The bash commands for the placement of the url from the text file to the local variable (file 'runrecon.sh'), and also the above function that verifies it (file 'func.sh'), are shown in figure 15 and figure 16:

```
#Stores the first line of the URLTEMP.txt file in a variable. It is the final URL
FINALURL=$(head -n 1 URLTEMP.txt)
```

Figure 15. Command that stores in 'FINALURL' variable the final domain which is stored in 'URLTEMP.txt' file

```
urltempcheck() {
if [[ ! $FINALURL == http* ]]; then
    echo -e "\nFINAL URL seems that does not appear in URLTEMP.txt file.\nPlease
check again. (Variable does not start with http)\nExiting script....."
    exit 1
fi
}
```

Figure 16. The tool checks if a final URL was found and verified (If not, the tool stops)

4.4. Discover the "real" domain name

After all these steps and checks, we remain to the backbone 'runrecon.sh' file and proceed to the next stage which calls python file 'extract_tld.py' by executing the command "*python3 extract_tld.py $FINALURL*". As we notice, the target url that contained in the FINALURL variable is passed as argument to the command.

In this python script we use the powerful library '**tldextract'** which strips a url and in fact extracts the right domain name along with the right TLD, from the final url schema. The 'tldextract' is not included in the standard libraries of Python3 programming language, but had to install it manually to the OS by using the Linux 'pip' package installer with the simple command '*pip install tldextract*'

We often have situations with URLs that contain subpaths or URLs with the corresponding domain names having TLDs consisted by two or more words. In such cases, parts of TLDs are considered as subdomains and as a result, users get a wrong and maybe a nonexistent domain name.

The above library uses the "Public Suffix List (PSL)" of Mozilla Foundation, which is a continuously growing and community-maintained database of Top-Level Domains. The PSL was mainly created for the security and privacy policies of the Firefox web browser, but it is also used in a vast variety of internet technologies, always under the Mozilla Public License (MPL).

Then, the above python script appends to the 'URLTEMP.txt' file the right and clear target domain name. The code syntax of the above file is shown to the following figure:

```
# Author: George Tasios
# FileName: extract_tld.py

##called from 'runrecon.sh' file##

import tldextract #use this library to extract tld and main domain
import sys

global finaldomain

dom = sys.argv[1]

#The following line extracts the domain and the TLD, even if TLD
```

```
#is more than 2 words and keep the url clear from subdomains or paths
t = tldextract.extract(dom)

#Sets the variable 'finaldomain' with the domain+its tld, even if it is more than
one word.
#TLDextract library uses the Mozilla public TLDs list.
finaldomain = t.registered_domain

file = 'URLTEMP.txt'

#Appending the argument in the URLTEMP.txt file
with open(file, 'a') as f:
    f.write(finaldomain)
```

Figure 17. The Python file 'extract_tld.py' exports the right target 'root domain' with its valid tld, from a URL

In the next step, the tool stores in the FINALDOMAIN variable the final domain name along with its right TLD, as it was previously extracted and appended in 'URLTEMP.txt' file. The corresponding command is as follows:

```
 FINALDOMAIN=$(head -n 2 URLTEMP.txt | tail -n 1)
```

4.5. Files and Folders creation

During the next steps we have the very important creation stage of the files and folders' structure where the results of the whole procedure will be stored, separately for each target domain that this tool of this thesis run for.

Firstly, we create a "**report**" folder (if it doesn't already exist) in the same location where the files of this tool are located. In the 'report' folder other subfolders are going to be created each time the tool runs for a new target domain, while if the subfolder for a target domain already exists, the tool will store the new files into the already existed folder (but with different 'timestamps' in the name) and will not create a new one. The distinction between old and new files concerning the same target domain, that created on different times, can easily be accomplished by sorting them using the timestamps the files would include into their own names. The bash code commands that perform the above, are shown in the following figures:

```
mkdir -p -m 777 report
```

Figure 18. Creation (if does not exist) of 'report' folder

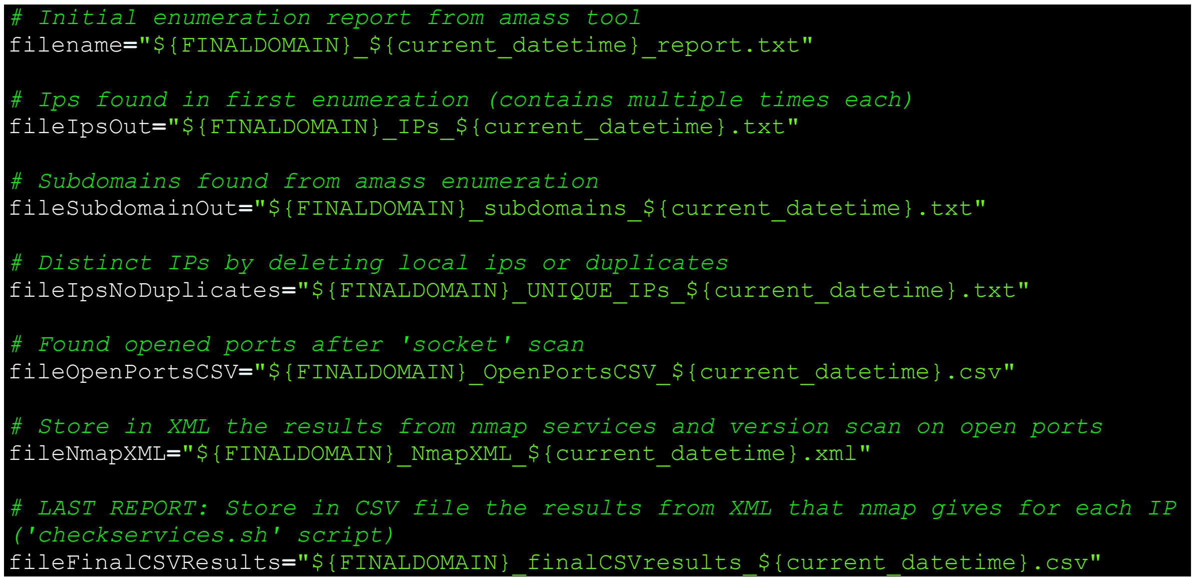
```
mkdir -p -m 777 report/${FINALDOMAIN}
```

Figure 19. Creation (if does not exist) of subfolder that will be used for the current target domain

As we notice to the last command, the current value of the FINALDOMAIN variable, which is actually the name of the current target domain name, is used in the naming of the subfolder.

Then, the tool composes the names of the files that are going to be created. Each of them will be used in a different stage of the procedure. The naming pattern that is followed, include three parts. The first one is actually the domain name itself; the second one declares what this file will contain and the third one is a current timestamp.

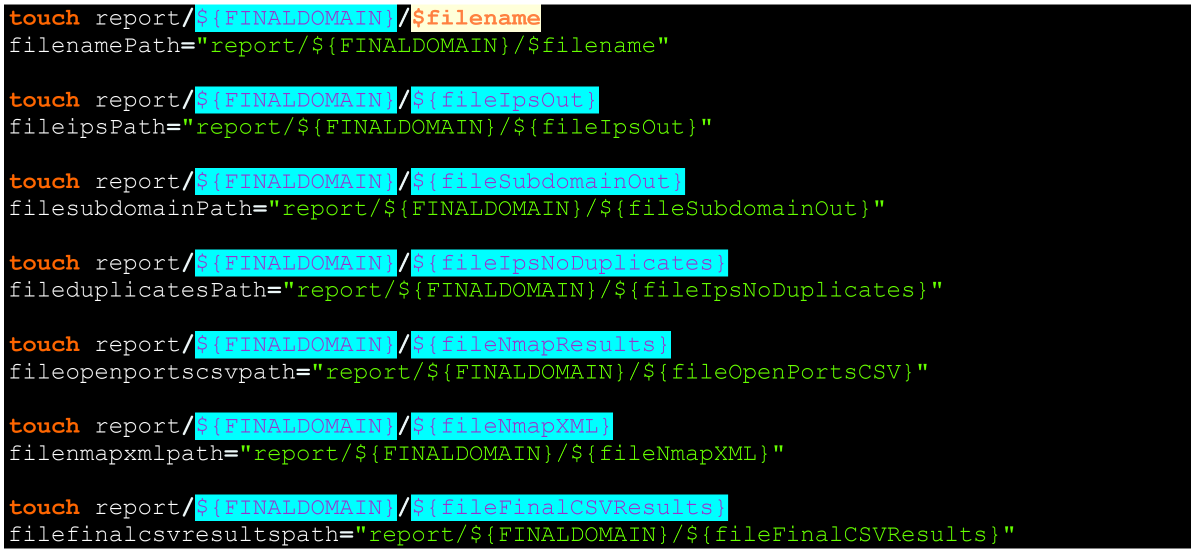
The bash code for their creation along with the comments that inform us on what each file will contain, is shown in the next figure:



```
# Initial enumeration report from amass tool
filename="${FINALDOMAIN}_${current_datetime}_report.txt"

# Ips found in first enumeration (contains multiple times each)
fileIpsOut="${FINALDOMAIN}_IPs_${current_datetime}.txt"

# Subdomains found from amass enumeration
fileSubdomainOut="${FINALDOMAIN}_subdomains_${current_datetime}.txt"

# Distinct IPs by deleting local ips or duplicates
fileIpsNoDuplicates="${FINALDOMAIN}_UNIQUE_IPs_${current_datetime}.txt"

# Found opened ports after 'socket' scan
fileOpenPortsCSV="${FINALDOMAIN}_OpenPortsCSV_${current_datetime}.csv"

# Store in XML the results from nmap services and version scan on open ports
fileNmapXML="${FINALDOMAIN}_NmapXML_${current_datetime}.xml"

# LAST REPORT: Store in CSV file the results from XML that nmap gives for each IP
('checkservices.sh' script)
fileFinalCSVResults="${FINALDOMAIN}_finalCSVresults_${current_datetime}.csv"
```

Figure 20. Bash code that creates names for the needed files that each step of tool's execution will use

After the creation of each file name, according to the naming pattern (that mentioned above), taking into consideration the target's domain name for each one of the above filenames, the corresponding file is created and an appropriate file path that points to each file is also created (as variable), as shown to the below figure:



```
touch report/${FINALDOMAIN}/$filename
filenamePath="report/${FINALDOMAIN}/$filename"

touch report/${FINALDOMAIN}/${fileIpsOut}
fileipsPath="report/${FINALDOMAIN}/${fileIpsOut}"

touch report/${FINALDOMAIN}/${fileSubdomainOut}
filesubdomainPath="report/${FINALDOMAIN}/${fileSubdomainOut}"

touch report/${FINALDOMAIN}/${fileIpsNoDuplicates}
fileduplicatesPath="report/${FINALDOMAIN}/${fileIpsNoDuplicates}"

touch report/${FINALDOMAIN}/${fileNmapResults}
fileopenportscsvpath="report/${FINALDOMAIN}/${fileOpenPortsCSV}"

touch report/${FINALDOMAIN}/${fileNmapXML}
filenmapxmlpath="report/${FINALDOMAIN}/${fileNmapXML}"

touch report/${FINALDOMAIN}/${fileFinalCSVResults}
filefinalcsvresultspath="report/${FINALDOMAIN}/${fileFinalCSVResults}"
```

Figure 21. Necessary files are created and variables that show to their locations respectively as well

The above created variables that point to the files, will be used in the next steps in order to store the results that will occur in each stage as the tool runs.

Following, comes the most important stage of our tool, the run of Amass enumeration tool which initiates with a command shown in the next figure:

```
amass enum$act$verbose_mode -ip$br$wlpath -d $FINALDOMAIN >
"report/${FINALDOMAIN}/$filename" &
PID2=$!
```

Figure 22. The command that initiates the enumeration by executing the Amass tool

Analyzing the above, the amass tool starts running by taking as arguments what has been previously explained. Flags that user chooses to perform active enumeration techniques, to show increased verbosity of the results and to brute force subdomains using a wordlist are some of them shown in the above command.

4.6. The running "spinner"

Moreover, in the above command we notice that the results are stored in the custom-named report file, as its creation explained above. In the end of this command, the "**& PID2=$!**" part is used to move the executing tool as a running process in the background, and store that process' pid (Process IDentifier) to the "PID2" variable. Then, after some messages, the function "spinner" comes to initiate a spinner, which truly is a few animated characters showing to the user that the program is still running. The ones that have been chosen for this thesis, give the user the illusion that an ever-moving character runs on bash terminal. Because of the extended times that it takes for this tool to complete its actions, this spinner was necessary to be added, in order to inform the user that the system is not "halted" and to know whether the tool is runs or stopped (by any external factor). The spinner function is in the 'func.sh' file and its code is depicted to the next figure:

```
spinner() {
    i=1
    sp="□□□□"
    echo -n ' '
    echo "Running......"
    while [ -d /proc/$1 ]
    do
        printf "\b${sp:i++%${#sp}:1}\b"
    done &
}
```

Figure 23. The bash function 'spinner' informs the user that the enumeration is still running

The above spinner runs with the simple command '**spinner**' which is followed by the command "**PID3=$!**". The last command stores the running process' pid to the PID3 variable in order to make it run in the background.

The 'q' button is suggested before the spinner starts, to be pressed if the user wishes to terminate the enumeration procedure, with the prompt "*Enumeration will may run more than 5 min, depending on your machine resources and the revealed data. You can simply end it by pressing 'q'*". The above appears in the terminal.

While the previous processes are running in the background, an addition of a "while-loop" in the code, enables the tool to check periodically (every 3 seconds) if the 'Amass' tool is still running (PID2 process active), or if the 'q' button pressed by the user. The bash code is shown in figure 24:

```bash
while true; do
  # Asks the user if wants to stop
  # Read user input without a timeout
  read -t 3 -n 1 input
  # Check if the user pressed 'q'
  PIDCHECK7=$(pgrep "amass")
  if [[ $PIDCHECK7 == "" ]]; then
    echo -e "\nEnumeration process almost completed..."
    sudo kill $PID2 2>/dev/null
    sudo kill $PID3 2>/dev/null
    echo -e "Subdomain enumeration completed...."
    time_elapsed "Time elapsed."
    break
  fi
  if [ "$input" == "q" ]; then
    echo -e "\nEnumeration process stopped manually..."
    sudo kill $PID2 2>/dev/null
    sudo kill $PID3 2>/dev/null
    echo -e "Subdomain enumeration completed...."
    time_elapsed "Time elapsed.."
    break
  fi
done
```

Figure 24. The bash code checks if the 'q' button pressed and terminates the enumeration

Explaining the above, the while-loop searches every 3 seconds if any process with the name 'amass' exists and stores it to the 'PIDCHECK7' variable. Then checks if the above variable is empty or if the 'q' button is pressed and if any of these two conditions is true, it stops (kills) the aforementioned PID2 and PID3 processes, shows appropriate messages to the user (console), then prints the elapsed time and continues. The tool uses the timestamp variable that was previously created by the 'timer_start" function, and along with an appropriate message, prints the elapsed time since the start of execution of this tool.

In order to achieve this, as we see in the above code, a function called "**time_elapsed**" which is included in the 'func.sh' file will be executed using the command syntax '**time_elapsed "Time elapsed.."**'. This function can take as argument whatever text the user wants and prints it when it is executed. In our case, the tool inserts the argument "Time Elapsed.." and it is printed. The bash code of the 'time_elapsed' function is as follows:

```bash
time_elapsed() {
  current_time=$(date +%s)
  elapsed_time=$((current_time - start_time))
  days=$((elapsed_time / 86400))
  hours=$(( (elapsed_time % 86400) / 3600 ))
  minutes=$(( (elapsed_time % 3600) / 60 ))
  remaining_seconds=$((elapsed_time % 60))
  DAYS=""
  HOURS=""
  MINUTES=""
  if [ "$days" -gt 0 ]; then
    DAYS="${GREEN}$days ${YELLOW}Days,"
  fi
  if [ "$hours" -gt 0 ]; then
    HOURS="${GREEN}$hours ${YELLOW}Hours,"
  fi
  if [ "$minutes" -gt 0 ]; then
    MINUTES="${GREEN}$minutes ${YELLOW}Minutes,"
  fi

  echo -e "\n---->${BLUE}$1: $DAYS $HOURS $MINUTES ${GREEN}$remaining_seconds
${YELLOW}Seconds\"....${ENDCOLOR}\n"

}
```

Figure 25. 'time_elapsed' function that estimates the time since the start of tool's execution

In previous code, a comparison is performed between the first timestamp that was created and the current time, and as a result the actually elapsed time is printed using seconds, minutes, hours and even days if it is necessary.

4.7. Handle and process the discovered IPs

Now, what follows is a procedure to make the results of the previous stage more 'understandable' by the tool and in order to achieve this, a python file the '**convIP.py**' is called and executed while the results file and the other text files that were created above are passed as arguments, through their corresponding variables that also created together. The command "*python3 convIP.py $filenamePath $fileipsPath $filesubdomainPath $fileduplicatesPath*" is executed and takes as 1st argument the variable where is stored the path to the results file, as 2nd the variable of the file in which will be stored all the IP addresses when will be extracted from the results file, as 3rd the variable of the file that will contain the extracted subdomains from the results and lastly, as 4th the file that will contain only IPv4 and IPv6 addresses, while duplicates will have removed and only global IPs will have been kept.

Let's start step by step to explain the functions that this file contains, beginning with the '*check_if_any_ip_found(sys.argv[1])*'. This function uses the results file (1st argument) and checks if it contains any result or if it contains the default message of amass tool which informs the user that no ips found. The code of the above function is as bellow:

```
def check_if_any_ip_found(infile):
    try:
        with open(infile, 'r') as file:
            first_line = file.readline().strip()

        if not first_line or first_line == "No names were discovered":
            print("\nNo names were discovered. No results found\n")
            exit(1)
    except Exception as e:
        print("\nError opening results file..... please try again\n")
        exit(1)
```

Figure 26.'check_if_any_ip_found' function that checks if any result occurred from the enumeration

Of course, if no ip address is found the tool is terminated as we see the exit(1) command in the above figure 26. The same happens if because of corruption or any other reason the file is not readable, fact from which an exception occurs during the code execution and can cause disruption of the normal flow of the tool's execution.

The results file (the first non-processed output file from amass tool) in fact is a text file which is similarly formed like the comma separated files, where the first value of each line is one of the found subdomains and the rest values of each line are the ips found that correspond to the subdomain of the line.

Then, with some simple processing the results file takes the form that we wish in order to pass it to the next functions and handle its contents easier. For example, we remove the commas from the file and replace them with simple blanks. In order to do that we execute the function '*remove_commas(sys.argv[1])*' whose code is in figure 27:

```
def remove_commas(file):
    with open(file, 'r') as f:
        content = f.read()
    content_with_no_commas = content.replace(',', ' ')
    with open(file, 'w') as f:
        f.write(content_with_no_commas)
```

Figure 27. Remove commas from the first results with 'remove_commas' function

Now while the suitable format has been applied to the results file, we are ready to extract all the IP addresses and for that, with the "*extractWords(sys.argv[1])*" command the 'extractWords' function extracts all the space separated ips to the new file. The code for the above is in figure 28:

```
def extractWords(filein, fileout):
    with open(filein, 'r') as infile, open(fileout, 'w') as outfile:
        for line in infile:
            words = line.split()
            # Extract all words except the first word
            filtered_words = words[1:]
            # Write the extracted words to the output file
            for word in filtered_words:
                outfile.write(f"{word}\n")
```

Figure 28. The 'extractWords' function extracts the IPs and store them into a new file

Now we have a file that contains only ip addresses, one per line and we can then handle them much easier. Furthermore, the "*extract_urls(sys.argv[1], sys.argv[3])*" command executes the corresponding function in order to extract all subdomains found and are stored in the aforementioned file. The code for that function is as below:

```
def extract_urls(in_file, out_file):
    with open(in_file, 'r') as infile, open(out_file, 'w') as outfile:
        for line in infile:
            # Extract the url (subdomain) each line
            url = line.split(' ', 1)[0]
            # Write the URL to the output file
            outfile.write(url + '\n')
```

Figure 29.'extract_urls' function that extracts and stores only the subdomains into a separate file

Some processes such as the last one were done and stored in separated files, only for reasons of scalability in any future upgrading and has no useful value at this stage of development of the tool of this thesis.

Next, the tool removes the duplicates ips and also removes local ips (IPv4 or ipv6) that may occur during the previous DNS enumeration. This procedure is helpful in order to decrease the number of requests to the next steps but also to avoid errors for the tool while trying to further examine non-existing local ip addresses (or to be more correct, to examine only locally accessible ips). It is achieved by using the data structure 'set' which is actually a 'list' of unique elements. If an already existing element is added, in fact it will not be added for a 2nd time but will be rejected.

The above are achieved with the function “*remove_duplicate_and_private_ips*” which is executed using as arguments, the extracted ips file (sys.argv[2]) as well as the sys.argv[4] file which is the file with the unique global ips that will occur after the process. The python code for the above function is as seen in the below figure:

```
def remove_duplicate_and_private_ips(in_file, out_file):
    unique_ips = set()
    with open(in_file, 'r') as file:
        for line in file:
          ip = line.strip()
          if ip:
              unique_ips.add(ip)
    with open(out_file, 'w') as file:
        for ip in unique_ips:
            myip = ipaddress.ip_address(ip)
            if myip.is_global:
              file.write(ip + '\n')
```

Figure 30. The Python code of the function which removes duplicate and private Ips

As we see in details, a python set() named ‘unique_ips’ is created and all found ips that exist in the appropriate file (1st argument) added in the set(). The above function then, uses the ‘ipaddress’ python library, iterates the contents of the ‘unique_ips’ set, and examines each one if it’s a global or local ip address. In local ips it does nothing whereas in every global one adds it to the unique ips file (2nd argument). After the above we have the output inserted to the unique ips file.

The next and last stage of the currently described file is a function that counts the unique IPv4 and IPv6 addresses finally occur by all the previous steps. It prints the number of each ip type while it prints a message if no ips or results found. The Python code of the above appears in the next figure:

```
def count_ips(infile):
    ipv4_count = 0
    ipv6_count = 0
    with open(infile, 'r') as file:
        for line in file:
            line = line.strip()
            if line:
                try:
                    if ipaddress.IPv4Address(line):
                       ipv4_count += 1
                except ValueError:
                    try:
                        if ipaddress.IPv6Address(line):
                            ipv6_count += 1
                    except ValueError:
                        print("some lines contain nor an IPv4 neither an IPv6 address.
Please check again")

    print("Found " + str(ipv4_count) + " ipv4 IPs and " + str(ipv6_count) + " ipv6 IPs")
```

Figure 31.The ‘count_ips’ Python function counts the IPv4 and IPv6 found addresses and informs the user

Any of the above functions and commands that are included in the ‘convIP.py’ file may fail or for various reasons end in the command “exit(1)”. For this reason, after the execution of the ‘convIP.py’ file, the function “checkPythonExit” which exists in the ‘func.sh’ file, runs right after and catches the exit status of the last executed command. As it was explained before, checks the exit status if it's equal to '1'. If it is true, prints an appropriate message to the console and exits the script. In other cases, just let the tool proceed to the next commands. The code of checkPythonExit function has been previously described.

4.8. Explain possible (IP) security techniques

In the next step, the tool will check the found ips for open ports but before starting examining them will print an appropriate message which gives some warnings to the user. The user must be sure that some techniques (often popular ones) are not used behind the currently examined domain name, in order to be sure that the procedure of this tool will end into giving real results that correspond to reality and not results that occur from an implementation of security techniques, services or similar systems. The following message is printed:

"*Now found IPs, will be checked for open ports*
Just be sure that all IPs are not behind CDN, Proxies or similar services,
otherwise results will be inaccurate and it may take excessively long time to complete"

Since such techniques or services provided by a plethora of companies and organizations in the world and is quite difficult to be detected in an automated way, let's explain some of them and how can affect the reliability or accuracy of the results of this tool.

The **CDN** (Content Delivery Network) is a system of distributed servers located around the world. Is a network of servers, whose number may reach to many hundreds of thousands. These servers are in a balanced cooperation in order to work and provide web content to users usually based on their geographic location. Although this way a user can receive content from a server which might be much closer to his real location, with improved performance, this technique masks the real ip address of the real web service of an organization and replaces it maybe with more than one different ips which belong to the CDN. In a similar case, it is obviously that even advanced techniques for reconnaissance like those used by this tool cannot be applied.

Other similar techniques that hide the real ip of an organization, except CDNs are **reverse proxies**. These are servers that run in the middle of the organization's infrastructure and the open internet and in fact hides the real ip of the server in which the web services run, but forwards clients' requests to web servers while showing only proxy's ip address.

Additionally, Virtual Private Network (**VPN)** techniques mask the real ip address and actually what is exposed is the ip of a VPN server which may is deployed and run in a totally different geographical location. Unlike the proxies, VPN servers can hide the real ip without having the need to exist in the same location with the protected server which hosts the web services of the organization.

Similar to the Content Delivery Network but behind only from a single server independently what the user's geographical location is, exist services which are used with a purpose of **protecting from DDoS** attacks, which is also a technique of masking the real ip of the infrastructure where web services run and changes only if is needed. It can be achieved even after an online subscription, following by a simple system configuration, maybe with a very low or none cost. Simple to be implemented but quite protective by hiding the real ips.

All of the above-mentioned techniques cannot be detected without doing more than one online action, or using various techniques, sometimes advanced ones, to achieve it and **this is why cannot be detected for sure in an automated way**, because may exist hundreds of different ways to deploy them. Therefore, tools like the one of this thesis', cannot be used in such cases.

4.9. Open Ports Enumeration

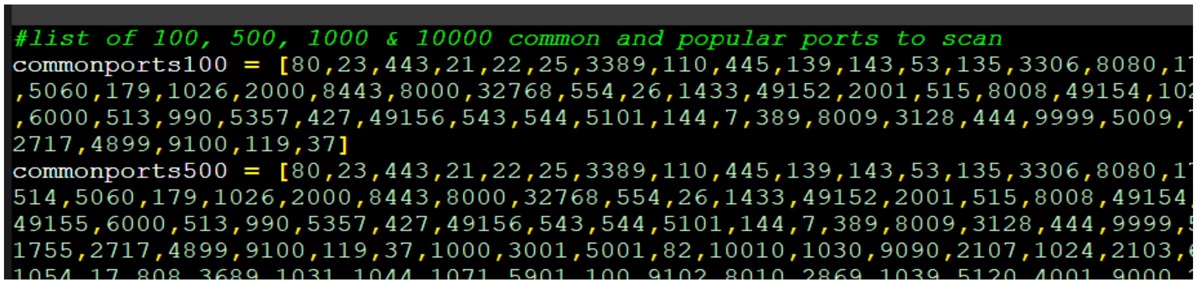
We are still in the “runrecon.sh” file where we continue with the open ports enumeration stage. In this stage, the tool calls the “**portsenum.py**” python file which takes as 1st argument the path to the unique IPs file ('fileduplicatespath' variable) and writes in the pre-created csv file (2nd argument - ‘fileopenportscsvpath’ variable), one ip with its open ports in each line. The command that initiates the above is “*python3 portsenum.py $fileduplicatesPath $fileopenportscsvpath*”.

Analyzing the ‘portsenum.py’ file and as we are in a critical stage of the whole procedure, the tool checks the internet connection and if it is ok proceeds to the next stage, while if it is down, it terminates the program and gives an appropriate message to the user. In the figure 32 we see the Python code of the above that exists in the body of the file and is executed directly without calling any functions:

```
# check internet connection
try:
    socket.setdefaulttimeout(2)
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(("8.8.8.8", 53))  # Google DNS IPv4
    print("\nInternet connection seems to be OK...\n")
except:
    print("\n" + 60 * "*")
    print("*Possibly Internet connection iS Down! Get online and retry*")
    print(60 * "*" + "\n")
    exit(1)
```

Figure 32. The Python code that checks the Internet connection state

Then in the above file, 4 lists of 100, 500, 1000 and 10000 port numbers are created. In these lists the port numbers appear randomly and not sorted. In the effort to acquire the port numbers and create the lists, we used the most popular common ports recommended by various online GitHub repositories, security researchers, cybersecurity online articles etc. Then we used some simple tools to come to a decision and use the best subset of them. In the figure 33 a small part of the above port numbers is depicted:



```
#list of 100, 500, 1000 & 10000 common and popular ports to scan
commonports100 = [80,23,443,21,22,25,3389,110,445,139,143,53,135,3306,8080,1
,5060,179,1026,2000,8443,8000,32768,554,26,1433,49152,2001,515,8008,49154,10
,6000,513,990,5357,427,49156,543,544,5101,144,7,389,8009,3128,444,9999,5009,
2717,4899,9100,119,37]
commonports500 = [80,23,443,21,22,25,3389,110,445,139,143,53,135,3306,8080,1
514,5060,179,1026,2000,8443,8000,32768,554,26,1433,49152,2001,515,8008,49154
49155,6000,513,990,5357,427,49156,543,544,5101,144,7,389,8009,3128,444,9999,
1755,2717,4899,9100,119,37,1000,3001,5001,82,10010,1030,9090,2107,1024,2103,
1054,17,808,3689,1031,1044,1071,5901,100,9102,8010,2869,1039,5120,4001,9000,
```

Figure 33. A small part of the port lists used for the enumeration procedure

Then in this step, various lists and a variable are created. More specifically, the ‘portsToScan’ list which will be set equal to the list that contains the ports the user chose to be scanned, the ‘open_ports’ list where all ports found open for the currently examined ip address will be appended on this list, the ‘closed_ports’ list where all the closed ports of the currently examined ip address will be appended, and finally the ‘ipsno’ variable where the number of the unique ips that exist in the unique ips file, will be stored after a count of it.

In the beginning of the above, a check if any argument was inserted while run the above file is done. The code for the above check that is in the body of the file and is executed in each case, is shown in the next figure:

```
#Checking if argument inserted, otherwise it stops running script
if len(sys.argv) < 3:
    print("Missing or no arguments inserted")
    exit(1)
else:
    ipfile = sys.argv[1]    # Set the inserted file equal to variable 'ipfile'
    portsfile = sys.argv[2]
    print(f"\nYou have inserted {ipfile}\n")
```

Figure 34. Python code that checks if an argument inserted in 'portsenum.py' file execution

After that and speaking about IP addresses number counting, the following Python command (figure 35) estimates it and stores the results in the abovementioned 'ipsno' variable:

```
with open(ipfile, 'r') as ipf:
     ipsno = len(ipf.readlines())
```

Figure 35. Calculates the number of IPs contained in the UNIQUE IPS file

The next part of code of the above file that is executed prints a message that prompts the user to choose the port enumeration depth. In this point the user can make a choice by pressing a button (from 1 to 5) depending on what depth wishes to run the port enumeration, starting with the 100 most common ports and ending with the choice of enumerating all 65.535 ports of each found ip address.

Of course, scanning all possible ports of an ip can be extremely time consuming and may need the tool to run for excessively much time, even for days.

In the next part of the code which is also in the file body and is executed after the above message, a "while-loop" starts and runs for about 60 seconds while checking each time if any key is pressed by the user. Decomposing this part of code, we start with the 'remainingtime' variable which is estimated every second and actually contains the remaining time for the while-loop to stop and proceed with the default choice '1' (100 ports to enumerate).

The estimation of the remaining time is done with the command "*remainingtime = int(endingtime - time.time())*". The predefined variable 'endingtime' is initialized with the timestamp that the while-loop is going to stop (61 seconds added to the time when the above while-loop started). The current remaining time is estimated by removing the current time (in seconds) from the endingtime. This way, in any moment until the while-loop ended, the tool can print the remaining seconds until a choice is made by the user or until the 60 seconds passed and the tool proceeds with the default option.

In the while-loop, the part of code of the following figure 36, checks if the time of 60 seconds has passed with no choice selected by the user. In this case it exits automatically the loop after having selected the default choice '1':

```
remainingtime = int(endingtime - time.time())
    print("***If nothing pressed in ", remainingtime ," secs, will proceed with
default choice (100 most common ports) :")
    try:
        rlist, _, _ = select.select([sys.stdin], [], [], remainingtime)
    except select.error:
        print("\nTimed out. Proceeding automatically to choice -1- (100 ports)")
        choice = "1"
        sys.exit(0)
```

Figure 36. The loop of 60 seconds where the tool waits for the users choice to be pressed

In the while-loop, the following part of code is executed if any choice is made by the user:

```
if rlist:
        number_str = sys.stdin.readline().strip()
        try:
            number = int(number_str)
            if 1 <= number <= 4:
                print("You entered", number)
                choice = str(number)
                break
            elif number == 5:
                if choice == "5":
                    break
                else:
                    print("You have chosen to scan ALL ports of the target!!!")
                    print("Are you sure? It will take excessively much more time
than usual")
                    print("Enter again and press ENTER to commit your choice or
review it in the next", remainingtime, "remaining seconds...")
                    choice = str(number)
            else:
                print("Invalid input. Please enter a valid choice from 1 to 5")
        except ValueError:
            print("Invalid input. Please enter a valid choice from 1 to 5 ")
    else:
        print("\nNo input received. Proceeding automatically\n")
        choice = "1"
        break
```

Figure 37. Python code with actions that take place if the user made a choice

As we notice in the above code, if any of the options from ‘1’ to ‘4’ is chosen by the user (100, 500, 1000 or 10.000 ports selected to enumerate respectively) the tool prints the choice to the user, breaks the while-loop and proceeds with the choice of the user. But in the case where the option ‘5’ is chosen (all 65.535 ports selected to enumerate), the tool prompts the user to reconsider the selected option and informs about the excessively much time that is needed for that, by printing a warning message, as shown in the Python code in the previous figure 37. Of course, appropriate messages are printed if a wrong number is inserted as an option by the user, as also is predicted by the code of the above figure 37.

Afterwards, according to the option selected by the user or by the tool (default), the ‘choice’ variable takes an appropriate value and a message about that choice is printed. In the following figure, in the part of code is obvious what is the message that is printed in each case:

```
if choice == "1":
    print("Continuiiiiing with choice: -", choice, "- scanning 100 ports for each
IP")
    portsToScan = commonports100
elif choice == "2":
    print("Continuiiiiing with choice: -", choice, "- scanning 500 ports for each
IP")
    portsToScan = commonports500
elif choice == "3":
    print("Continuiiiiing with choice: -", choice, "- scanning 1000 ports for
each IP")
    portsToScan = commonports1000
elif choice == "4":
    print("Continuiiiiing with choice: -", choice, "- scanning 10000 ports for
each IP")
    portsToScan = commonports10000
elif choice == "5":
    print("Continuiiiiing with choice: -", choice, "- scanning all 65.535. It
will take much time....")
    portsToScan = list(range(1, 65535))
else:
    print("Continuiiiiiing with default option -1-, scanning 100 ports")
    portsToScan = commonports100
```

Figure 38. After the user's choice an appropriate message is printed in each case

Moreover, with a simple command, independently of the chosen option, the elements of the selected 'portsToScan' list get an order randomization. The command "*random.shuffle(portsToScan)*" uses the 'random' python library in order to change randomly the order of the ports that are selected to be enumerated and exist in the list.

4.10. Scanning ports for services

Now it is the time to check the selected ports for each found ip address if any service runs behind. The following part of code runs and opens the 2 files that have been inserted to this python file as arguments. While iterating each line of the unique ips file, the "**scanport**" function is executed for the ip of each line. The part of the code is shown in figure 39:

```
with open(ipfile, 'r') as ipf, open(portsfile, 'w', newline='') as portscsv:
    csv_writer = csv.writer(portscsv)
    for line in ipf:
        ipsno=ipsno-1
        tempip = line.strip() #Strip the ip for spaces or linebrake characters
        scanport(tempip) #run the above script. It gives a list with open ports
        print(f"For ip {tempip} found {len(open_ports)} open ports")
        if len(open_ports) > 0:
            csv_writer.writerow([tempip] + open_ports) # Write the above list in
the CSV file
        open_ports=[]
        closed_ports=[]
    print(f"\nResults saved in CSV file in the path: {portsfile}")
```

Figure 39.Part of Python code that calls the 'scan_port' function

In the above code, we notice that if any open port is found for the currently investigated ip, which means that if the length of the 'open_ports' list is bigger than '0', the ip address along with the found open ports, are appended, separated with commas, as a new line, in the csv file (portsfile).

Now let's analyze the "scanport" function that is executed by the above code. The function gets one by one, the found ips and recognizes if the current ip is an IPv4 or an IPv6 type. Then tries to initiate a connection with the chosen ports using python sockets.

The python programming language, is selected to be used in this thesis because of the need for concepts like the port scanning, as it is easy to implement techniques like Socket connections, by using it.

A network Socket is an endpoint for communication between two devices over the network. Data exchanges can be achieved between two network devices using sockets because they consist fundamental concepts in network programming. Unlike http requests, **sockets reveal much less information about the client** that attempts to do the socket connection. The 'Socket' module of python is well-suited for port scanning because it is very simple to use, but using more technical terms, it is a module that allows low-level access to network interfaces. Since, it does not complete the tcp handshake while performs ports scanning, the randomization of the order of the ports that checked for each ip and the slow scanning speed helps this thesis' tool to increase significantly the possibilities to avoid detections from IDS systems or other similar with simple configurations.

As we see to the following figure of the 'scanport' function, the timeout for a socket connection is set to '1' second:

```
def scanport(ipaddress):
 print(f"\nScanning IP: {ipaddress} ({ipsno} IPs left to scan)")
 for porta in portsToScan:
   if '.' in ipaddress:
     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
   elif ':' in ipaddress:
     sock = socket.socket(socket.AF_INET6, socket.SOCK_STREAM)
   socket.setdefaulttimeout(1) #It should be '2' if internet connection is not fast
enough but usually it is not necessary nowadays
   result = sock.connect_ex((ipaddress,porta))
   #print(f"Printing result: {result}")
   if sock.connect_ex((ipaddress, porta)):
    print(f"---> Port {porta}: Closed <----", end='\r')
    closed_ports.append(porta)
   else:
    grabbanner(ipaddress,porta)
    #grabban(ipaddress, porta)
 print(f"\nPort scanning finished", 10*">")
 print(f"\nFound: Opened ports: {len(open_ports)} | Closed ports: {len(closed_ports)}")
 sock.close()
```

Figure 40. 'scanport' function initiates a socket connection with the currently IP, for previously chosen ports

In the above function we notice that if one port is found to be responsive to the socket connection, the function itself calls the "**grabbanner**" function and passes as arguments the current ip address and the port that responded to the socket connection attempt. This function uses the "**Banner Grabbing**" technique, which is a technique to determine if a port is open by grabbing the initial data sent by any online service (banner). Banner Grabbing technique works with a wide range of online services and not just web services (http protocol). Is much more versatile than other methods for port scanning and is suitable for identifying a wider range of services, while does not reveal information like client's User-Agent etc. The code of the 'grabbanner' function is shown as follows:

```
def grabbanner(ip,porta):
  socket.setdefaulttimeout(1)
  # A little bit of unorthodox way to recognize IPv4 and IPv6 addresses
  if '.' in ip:
      bangrab = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  elif ':' in ip:
      bangrab = socket.socket(socket.AF_INET6, socket.SOCK_STREAM)
  try:
    bangrab.connect((ip, porta))
    banner = bangrab.recv(128)
    print(f"*Something seems to be running behind port {porta} :")
    open_ports.append(porta)
    bangrab.close()
    print(f"====>Printing received banner : {banner}\n", end='\r')
  except:
    print(f"---> Port {porta}: Possibly closed <--", end='\r')
    closed_ports.append(porta)
    bangrab.close()
```

Figure 41. 'grabbanner' function called after a socket connection initiation succeeds

As we can see in the above code, the banner grabber uses Sockets to connect to the open port and grabs the data (banner) that sends with its reply. Then the port number is appended to the 'open_ports' list. If no banner received, the port is considered as closed and is appended to the 'closed_ports' list. With the banner grabber technique in combination with socket connections, we ensure that **if an open port found, it is a real open one** which has a running service behind. One-by-one the ports are examined with the above techniques, and while the tool is running, we get messages concerning the current port state, whether it is open or closed.

Even if a banner that was grabbed seems to be in a readable form, and gives us information about the running service, software version etc., **in most cases received banners are some characters that usually don't make any sense**. In order to be translated into a known service (the real service that runs behind the specific port) a most comprehensive examination of the port reply and the grabbed banner, must be done. This concept is implemented in the last steps of this tool's scanning procedure.

In the end of the current step, the tool informs the user about the number of open or closed ports, as well as about the numbers of ports that found opened, and stored in specific location and file.

The ips for which open ports found, are stored in one of the pre-created files, along with its open ports, in a single line for each ip address, comma separated with its ports.

4.11. Reveal running services

Now, we have reached in the last stage of the whole procedure of this tool. In this step the ip addresses that were previously found having open ports, will be further scanned in order to reveal with as much accuracy as it is possible, the running services and the device or software information that are deployed behind them.

The next file that is called and executed, is the bash script file "**checkservices.sh**" which runs with the command "*./checkservices.sh $fileopenportscsvpath $filenmapxmlpath $filefinalcsvresultspath*". As we notice, it takes as 1st argument the file that includes the open ports that found in the previous stage, as 2nd the filename and path where the final results will be stored as an XML file and as 3rd the filename and path, where final results will be stored in a CSV file.

Explaining the contents of the 'checkservices.sh' file, in its beginning the file imports the 'func.sh' file, which as mentioned earlier, includes several functions that can be called from here.

Firstly, while this file is executed, the "nmapCheck" function is called, in order to check if the 'nmap' software is installed in the system and if nmap is found, the function checks if the last ips file with the open ports, exists, otherwise the program stops its operation. The above function is defined as follows:

```
nmapCheck() {
  #firstly we will check if the nmap tool is installed
  command -v nmap >/dev/null 2>&1 || { echo >&2 "Nmap is required and it seems not to be
installed. Please install it."; exit 1; }
```

```
  #check if the IPs file exists
  if [ ! -f "$fileopenportscsvpath" ]; then
    echo "Error: Input file not found: $fileopenportscsvpath"
    exit 1
  fi
```

Figure 42. The 'nmapCheck' Bash function checks if nmap is installed to the system

After this check is finished with success, the tool proceeds with the code of the figure 43:

```
while IFS=',' read -r ip ports
do
    isipv6=False
    #echo "$ip"
    #echo "$ports"
    test_ip #Checks the current selected IP and checks if it is ipv6. The
'test_ip' function an ipv6 found sets the variable/flag==True
    print_ports
    if [[ $isipv6 == True ]]; then
        echo -e "\nfound IPv6\n"
        nmapipv6
    elif [[ $isipv6 == False ]]; then
        echo -e "\nfound IPv4\n"
        nmapipv4
    fi
done < $input_file
```

Figure 43. Bash code that initiates the service scanning functions for Ipv4 and Ipv6 addresses that found

This piece of code, demonstrates the most important action of this tool, reads IP addresses and associated ports from a file, determines whether each IP address is IPv4 or IPv6, and then runs a network scan using nmap tool based on the type of IP address. This algorithm will lead us in any existing information regarding the exposed surface of the domain-target.

Decomposing the above, we notice that it starts by setting an Internal Field Separator (**IFS**) to a comma and therefore it splits each line on commas. Then, reads each line from the input file which was previously set equal to the 1st argument inserted while this script run ("*input_file=$1*"). Each line is splitting into two variables. The 'ip' which contains the ip of the current line and the 'ports' which contains the ports of the current line of the input file. Then in the above while-loop the "isipv6" boolean variable is set as "*False*" which means that in this initialization, we consider the current ip of the line to be an IPv4 address.

Continuing, proceeds with the "**test_ip**" function that is defined by the code snippet in the next figure:

```
test_ip() {
    # Check if the ip variable contains an IPv4 address
    if [[ "$ip" =~ ^([0-9]{1,3}\.){3}[0-9]{1,3}$ ]]; then
        echo "$ip is an IPv4 address."
    # Check if the ip variable contains an IPv6 address
    elif [[ "$ip" =~ ^([0-9a-fA-F]{1,4}:){7,7}[0-9a-fA-F]{1,4}$|^([0-9a-fA-
F]{1,4}:){1,7}:|^([0-9a-fA-F]{1,4}:){1,6}:[0-9a-fA-F]{1,4}$|^([0-9a-fA-F]{1,4}:){1,5}(:[0-
9a-fA-F]{1,4}){1,2}$|^([0-9a-fA-F]{1,4}:){1,4}(:[0-9a-fA-F]{1,4}){1,3}$|^([0-9a-fA-
F]{1,4}:){1,3}(:[0-9a-fA-F]{1,4}){1,4}$|^([0-9a-fA-F]{1,4}:){1,2}(:[0-9a-fA-
F]{1,4}){1,5}$|^[0-9a-fA-F]{1,4}:((:[0-9a-fA-F]{1,4}){1,6})$|^:((:[0-9a-fA-
F]{1,4}){1,7}|:)$|^fe80:(:[0-9a-fA-F]{0,4}){0,4}%[0-9a-zA-
Z]{1,}$|^::(ffff(:0{1,4}){0,1}:){0,1}((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\.){3,3}(25[0-
5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])$|^([0-9a-fA-F]{1,4}:){1,4}:((25[0-5]|(2[0-4]|1{0,1}[0-
9]){0,1}[0-9])\.){3,3}(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])$ ]]; then
        echo "$ip is an IPv6 address."
        isipv6=True #Variable used as a flag, noting that the current ip is IPv6 version
    else
        echo "$ip is not a valid IP address."
    fi
}
```

Figure 44. 'test_ip' function that defines if an IP is a valid Ipv4 or Ipv6

The above function 'test_ip' checks whether a given IP address is IPv4 address, an IPv6 address or an invalid ip address, and sets a flag accordingly. With a more comprehensive explanation, the function uses a regular expression (regex) to check if the variable ip matches the pattern of an IPv4 address and if it is true, prints a confirmation message. If ip is not matched, proceeds and uses a complex regular expression to check if the ip matches the pattern of an IPv6 address. The regex covers various valid IPv6 formats, including full IPv6 addresses, compressed IPv6 addresses and IPv6 addresses with embedded IPv4. If the ip variable matches the IPv6 pattern a confirmation message is printed while the 'isipv6' variable is set as "True". But before that part of code ended, a last check is done by the function. If none of the above regular expressions do not match with the ip variable, the function prints a message that the ip is not a valid ip address and proceed with the next steps.

Going back to the previous while-loop, the tool continues by calling the "**print_ports**" function which is implemented as follows in the figure 45:

```
print_ports() {
    local num=0
    for i in ${ports//,/ }
    do
      ((num++))
      echo -e "No $num opened port is: $i"
    done
}
```

Figure 45. 'print_ports' function edits the results of open ports

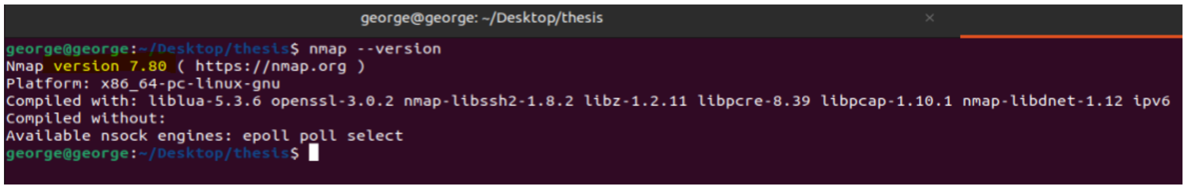
This function iterates over each port contained in the '*ports'* variable, while replaces each comma in the '*ports*' variable with a space, converting it from a comma-separated list into a space-separated list. Then it enters into a loop which keeps track of the current port's sequence number and prints the current open port number ('*num*' variable) and the port '**i**'. This procedure gives to the user information about the ip address type, the number and the order of ports regarding the current ip, that are going to be scanned.

In the end of the while-loop, the tool checks if the 'isipv6' variable is "True", prints an informative message about the found ip address type and proceeds accordingly using one of the functions, '**nmapipv6**' to perform an nmap scan if the ip address is an IPv6 or '**nmapipv4**' to perform an nmap scan if it is an IPv4. But before explaining the above two functions, let's see what is the nmap tool which is used in this thesis.

The **Nmap** (Network Mapper) is one of the most powerful and maybe the best, widely-used open-source tool for network discovery and security auditing, used by network administrators, security professionals and anyone involved in network management and security. Nmap is designed to scan large networks efficiently, but it was chosen for the tool developed in this thesis as it works perfectly for single hosts or smaller networks as well.

Among its features, running **services detection**, **service version** and **OS Version** detection after open ports probing and response analysis as well as **Masquerading as Different Clients** (mimicking different clients e.g., browsers, mobile devices etc.) using its Nmap Scripting Engine (NSE) in order to specify custom user-agent strings in its http or other requests, were instrumental for the author of this thesis in choosing nmap to be included in this tool's implementation.

The version of Nmap software used in this thesis is "Nmap version 7.80" and while it was not by default installed in the Linux OS (Ubuntu 22.04.4 LTS) in which this tool was developed, we proceeded with its installation using the default Linux Ubuntu primary software repositories in which nmap was included. The installation command was "*sudo apt install nmap*" and the verification of its successful installation as well as its version check committed using the "*nmap --version*" command, as is shown to the next figure:



```
george@george: ~/Desktop/thesis
george@george:~/Desktop/thesis$ nmap --version
Nmap version 7.80 ( https://nmap.org )
Platform: x86_64-pc-linux-gnu
Compiled with: liblua-5.3.6 openssl-3.0.2 nmap-libssh2-1.8.2 libz-1.2.11 libpcre-8.39 libpcap-1.10.1 nmap-libdnet-1.12 ipv6
Compiled without:
Available nsock engines: epoll poll select
george@george:~/Desktop/thesis$
```

Figure 46. "Nmap" version used and embedded in this thesis' tool

Returning back to this tool's implementation in the analysis of the last script file, we have the two most important functions coordinated by this tool. The "**nmapipv4**" and "**nmapipv6**" are similar functions that run if the current scanned ip address is an IPv4 or an IPv6 respectively. These functions' implementation is demonstrated in the following two figures:

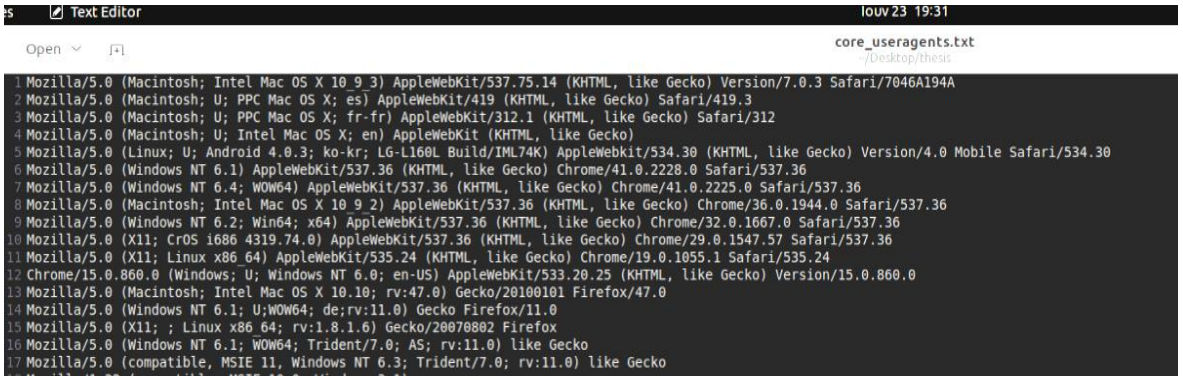
```
nmapipv4() {
    # Creating local array variable where the ports will be stored as an array and not as a
comma separated list
    local ports_array

    # convert comma separated ports from $ports into array which contains the ports
    IFS=',' read -r -a ports_array <<< "$ports"

    for port in "${ports_array[@]}"; do
        # function that selects Random user agent from file-list
        select_random_useragent
        # nmap command using custom selected user agent
        nmap -Pn -sV -T4 -n --script http-methods --script-args
http.useragent="$random_user_agent" $ip -p "$port" -oX $output_XML
        # Here will run the XML parsing procedure ('XMLprocess.py' script) and elements will
be appended to final csv output file (3rd argument of 'checkservices.sh')
        python3 XMLprocess.py $output_XML $final_output_CSV
    done


}
```

Figure 47. The 'nmapipv4' function scans the found IPv4 addresses for services behind their open ports

```
nmapipv6() {
    # Creating local array variable where the ports will be stored as an array and not as a
comma separated list
    local ports_array

    # convert comma separated ports from $ports into array which contains the ports
    IFS=',' read -r -a ports_array <<< "$ports"

    for port in "${ports_array[@]}"; do
        # function that selects Random user agent from file-list
        select_random_useragent
        # nmap command using custom selected user agent
        nmap -Pn -sV -T4 -6 -n --script http-methods --script-args
http.useragent="$random_user_agent" $ip -p "$port" -oX $output_XML
        # Here will run the XML parsing procedure ('XMLprocess.py' script) and elements will
be appended to final csv output file (3rd argument of 'checkservices.sh')
        python3 XMLprocess.py $output_XML $final_output_CSV
    done

}
```

Figure 48. The ‘nmapipv6’ function scans the found IPv6 addresses for services behind their open ports

Explaining the above, each function firstly runs an iterator which iterates over the comma-separated ports that are contained (as string) in ‘ports’ variable and convert them into an array (‘*ports_array*’) whose elements are the ports. Then with the for-loop, for each port runs the command ‘**select_random_useragent**’ which is a function responsible to choose a random user-agent string from a custom-made list (‘*core_useragents.txt*’ file) of user agents, for popular web browsers of desktop and mobile platforms. We created this list of user agents, by taking into consideration various sources suggested by a plethora of websites, articles and online repositories that belong to security researchers or network analysts (both amateurs and professionals). This user agent list is actually a text file and can be periodically updated manually by replacing or adding newer user agents’ strings. A sample of the contents of this file is shown in the following figure:



```
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3) AppleWebKit/537.75.14 (KHTML, like Gecko) Version/7.0.3 Safari/7046A194A
Mozilla/5.0 (Macintosh; U; PPC Mac OS X; es) AppleWebKit/419 (KHTML, like Gecko) Safari/419.3
Mozilla/5.0 (Macintosh; U; PPC Mac OS X; fr-fr) AppleWebKit/312.1 (KHTML, like Gecko) Safari/312
Mozilla/5.0 (Macintosh; U; Intel Mac OS X; en) AppleWebKit (KHTML, like Gecko)
Mozilla/5.0 (Linux; U; Android 4.0.3; ko-kr; LG-L160L Build/IML74K) AppleWebkit/534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30
Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2228.0 Safari/537.36
Mozilla/5.0 (Windows NT 6.4; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2225.0 Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1944.0 Safari/537.36
Mozilla/5.0 (Windows NT 6.2; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1667.0 Safari/537.36
Mozilla/5.0 (X11; CrOS i686 4319.74.0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/29.0.1547.57 Safari/537.36
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/535.24 (KHTML, like Gecko) Chrome/19.0.1055.1 Safari/535.24
Chrome/15.0.860.0 (Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.20.25 (KHTML, like Gecko) Version/15.0.860.0
Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:47.0) Gecko/20100101 Firefox/47.0
Mozilla/5.0 (Windows NT 6.1; U;WOW64; de;rv:11.0) Gecko Firefox/11.0
Mozilla/5.0 (X11; ; Linux x86_64; rv:1.8.1.6) Gecko/20070802 Firefox
Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; AS; rv:11.0) like Gecko
Mozilla/5.0 (compatible, MSIE 11, Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko
```

Figure 49. Sample (part) of user-agents file. One selected randomly, each time a port scanned

The code of the ’select_random_useragent’ function is shown below:

```
select_random_useragent() {
    #Checks if file with user agents exists
    if [ ! -f "$user_agents_file" ]; then
        echo "File with user agents not found: $user_agents_file"
        random_user_agent="Mozilla/5.0"
        echo "setting default user agent: $useragent"
        exit 1
    fi
    # Select randomly a user agent from the file
    random_user_agent=$(shuf -n 1 "$user_agents_file")
    echo -e "Random user agent selected is:\n$random_user_agent\n"
}
```

Figure 50.Bash function ‘select_random_useragent’ that selects randomly a new user-agent

The function firstly checks if the user agents' text file exists. This file's name is inserted in the variable '*user_agents_file*' in the beginning of the 'checkservices.sh' script file, where some variables are defined and initialized, as shown:

```
user_agents_file="core_useragents.txt"
useragent="Mozilla/5.0"
random_user_agent=""
```

As we notice, a variable named '*random_user_agent*" is defined and will be used to store the new user agent that will be selected before each time the above functions scan a port. In the above check, if the user agent's file is not found, the function prints a message to the user and sets the 'random_user_Agent' variable equal to "Mozilla/5.0" while prints also an informative message.

If the user agents file is found, the function randomly chooses a user agent with the command "*random_user_agent=$(shuf -n 1 "$user_agents_file")*" (in fact it chooses randomly a line of it) and prints a message with the choice that will be used in the next port scan.

The 'select_random_useragent' runs in the same way in both 'nmapipv4' and 'nmapipv6' functions, but nevertheless the next command which starts the nmap scan, differs as is clearly shown above, in each case. When we have an IPv4 address to scan, the command is:

```
nmap -Pn -sV -T4 -n --script http-methods --script-args http.useragent="$random_user_agent"
$ip -p "$port" -oX $output_XML
```

while when the ip address is an IPv6 the command is:

```
nmap -Pn -sV -T4 -6 -n --script http-methods --script-args
http.useragent="$random_user_agent" $ip -p "$port" -oX $output_XML
```

The difference is the '**-6**' flag that is used in order to prompt the nmap tool to handle the ip that is going to scan as an IPv6 version.

In a piece-by-piece explanation the above command demonstrates how the Nmap command is used by this tool to scan open ports and detect services.

The -***Pn*** option disables host discovery and treats all hosts as online. The ***-sV*** enables the Version detection while the -***T4*** sets the timing level to 4 and makes scanning a little bit faster but generates more traffic, the -***n*** option disables DNS resolution, the --***script http-methods*** detects supported http methods and the --***script-args*** along with ***http.useragent="$random_user_agent"*** sets a random user agent string as it occurs from the results of 'select_random_useragent' function. The output of each scan is an XML file. We chose this type over the others because it is much easier to parse and process the output of it, in comparison with the unstructured plain text Nmap reports.

4.12. Formatting the final results

In both of the 'nmapipv4' and 'nmapipv6' functions, after the Nmap command finished its job, we get the scan results for the current port in an XML file. Then we have to append those real results in the final CSV file but firstly we had to export the results for that port scan, by examining the XML file successfully and with no errors. That was quite difficult because despite that it is an XML file which means fixed tags and fields, there are excessively many odds for each port to occur different forms of related results, and consequently despite it will always be an XML file, we cannot create a simple parser to handle the xml results.

Due to the above, we took into account the dtd explanation (**dtd file**) of the official Nmap documentation[38] files, with which we essentially had in our hands all the possible fields that could

[38] https://nmap.org/book/nmap-dtd.html

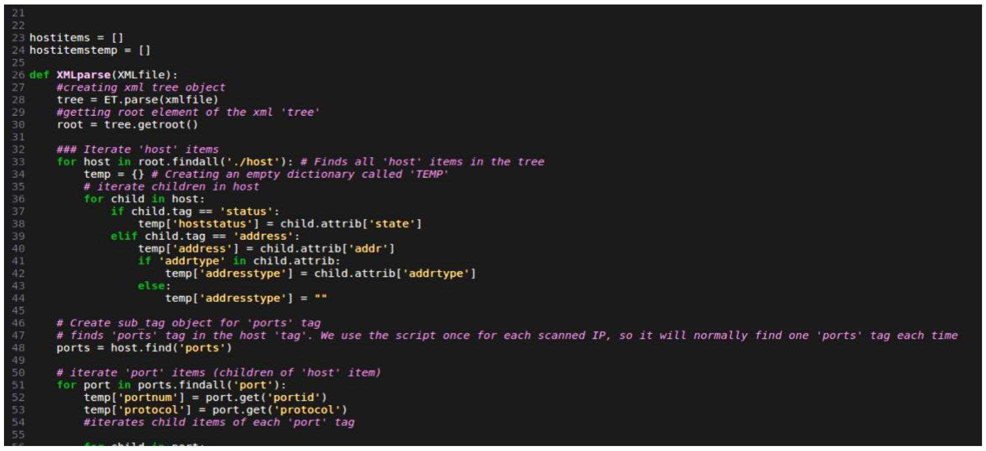
exist in a nmap XML results file. With this, we built the last script file (‘***XMLprocess.py***’) that handles the XML results of each run of the above nmap command and according to the dtd, appends parts of it that are useful to us, in the Final results CSV file.

As we have already seen above, the ‘XMLprocess.py’ python script is called from within nmapipv4’ and ‘nmapipv6’ functions with the command ‘***python3 XMLprocess.py $output_XML $final_output_CSV***“ ’ and takes as 1st argument the ‘***output_XML’*** file from nmapipv4 or nmapipv6 functions of 'checkservices.sh' and as 2nd argument the output file ‘**final_output_csv**’ where the final results will be stored.

The ‘**XMLprocess.py**’ file starts by creating an XML parser that handles the XML file as a tree and gets the items that we want to include in our report. Among all the items in the tags of the XML file, the ones that we are interested in including them in our final CSV report are the following:

Attribute ‘State’ as ‘**hoststatus**’ tag, ‘addr’ as ‘**address**’, ‘addrtype’ as ‘**addresstype**’, ‘portid’ as ‘**portnum**’, ‘protocol’ as ‘**protocol**’, ‘state’ as ‘**portstate**’, ‘service/name’ = as ‘**servNAme**’, ‘conf’ as ‘**servConfidence**’, ‘method’ as ‘**scanMethod**’, ‘product’ as ‘**Product**’, ‘version’ as ‘**serviceVer**’, ‘extrainfo’ as ‘**extraServInfo**’, ‘ostype’ as ‘**OStype**’, ‘devicetype’ as ‘**deviceType**’, and ‘cpe’ as ‘**CPE**’.

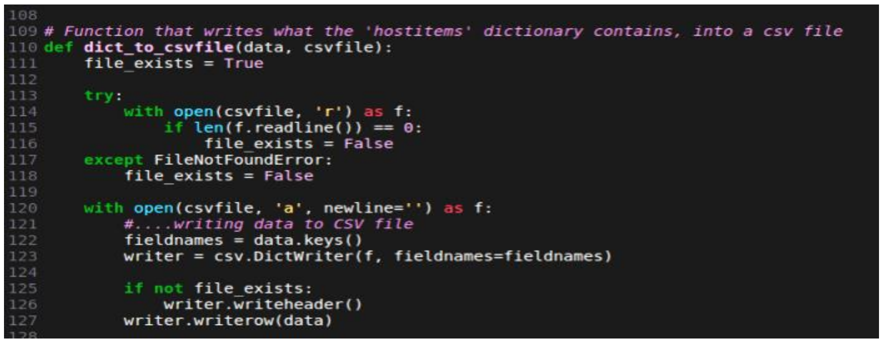
In the ‘XMLprocess.py’ file we can see two functions, the “*XMLparse*” and the “*dict_to_csvfile*”. The first one is called immediately when the file is executed and takes as argument the XML file, and searches if any of the above tags exist. Then composes a temporary dictionary (‘***temp{}***’) with the found elements. One sample part of the ‘*XMLparse*’ function is depicted in the figure 51:



```


hostitems = []
hostitemstemp = []

def XMLparse(XMLfile):
    #creating xml tree object
    tree = ET.parse(xmlfile)
    #getting root element of the xml 'tree'
    root = tree.getroot()

    ### Iterate 'host' items
    for host in root.findall('./host'): # Finds all 'host' items in the tree
        temp = {} # Creating an empty dictionary called 'TEMP'
        # iterate children in host
        for child in host:
            if child.tag == 'status':
                temp['hoststatus'] = child.attrib['state']
            elif child.tag == 'address':
                temp['address'] = child.attrib['addr']
                if 'addrtype' in child.attrib:
                    temp['addresstype'] = child.attrib['addrtype']
                else:
                    temp['addresstype'] = ""

    # Create sub_tag object for 'ports' tag
    # finds 'ports' tag in the host 'tag'. We use the script once for each scanned IP, so it will normally find one 'ports' tag each time
    ports = host.find('ports')

    # iterate 'port' items (children of 'host' item)
    for port in ports.findall('port'):
        temp['portnum'] = port.get('portid')
        temp['protocol'] = port.get('protocol')
        #iterates child items of each 'port' tag

```

Figure 51. XML Parser which handles the XML reports of nmap

Then, from within the ‘*XMLparse*’, the other function ‘*dict_to_csvfile’* is executed and appends all the results from the dictionary to the final CSV file while adds a headers row if doesn’t exist. The function is shown as follows in the next figure:



```

# Function that writes what the 'hostitems' dictionary contains, into a csv file
def dict_to_csvfile(data, csvfile):
    file_exists = True

    try:
        with open(csvfile, 'r') as f:
            if len(f.readline()) == 0:
                file_exists = False
    except FileNotFoundError:
        file_exists = False

    with open(csvfile, 'a', newline='') as f:
        #....writing data to CSV file
        fieldnames = data.keys()
        writer = csv.DictWriter(f, fieldnames=fieldnames)

        if not file_exists:
            writer.writeheader()
        writer.writerow(data)
```

Figure 52. Function 'dict_to_csvfile' appends current results from ports scanning, in the final csv report

The headers' names of the results in the Final results files, along with a short description of them are the following: "hoststatus" shows if the host (ip & port) is up or not, "address" the current examined ip address, "addresstype" the version (ipv4 or ipv6) of line's address, "portnum" the port number which this line of results are concerned, "protocol" the found protocol of the service that seems to run on the current ip-port, "portState" shows whether the port is open or closed, "CPE" the name of the service according to the NIST cpe naming scheme, "servName" the service name or type, "servConfidence" the level of confidence about the results of the current line (usually depends on the scan method), "scanMethod" the way the results of the current line occur, whether from a port probe or from a port default assignment table, "Product" includes additional information about the product that runs the found service, "serviceVer" running service version, "extraServInfo" extra service info if exists, "OStype" the type or protocol of running Operating System, and "deviceType" additional information about the device behind the port.

In the end of the 'runrecon.sh' execution the tool removes the unnecessary files (e.g. the URLTEMP.txt file) that created during the reconnaissance progress, prints the elapsed time and finally prints a footer which declares the end of execution, with the aforementioned 'printFooter' function.

But before the tool print the footer and after the csv file with the results is filled with the found data, it uses the **Neo4j database management system** (if found installed to the system and found to be running) to graphically depict the results by creating a graph from them. This procedure's description is following to the next chapter.

4.13. Graph of Visualized final Results – Neo4j

In this chapter we will describe the conversion of the CSV results into visualized, colorful and digestible results and the procedure that the tool follows in order to achieve it automatically without the need of user's interaction.

4.13.1. A few words overview

Taking already the results into a csv file is quite easy to use it with various applications and process the information we take by them. The user can parse the data in any software by simply use the final csv file. The matter is **when the user wishes to immediately examine the results'** data using a more friendly and easy to understand way to give the opportunity to the user to recognize and depict the "state of things". In what way we move from the initial target-domain to related subdomains, and after how they resolve to IP addresses and finally how these IPs contain open ports with services. All this structure is depicted by the tool of this thesis with the help of 2 script files that use the Neo4j graph database system.

Neo4j graph database management system is a popular, high-performance NoSQL graph database system that handles large scaled graph data and is able to successfully depict relationships between nodes using directed vectors and in the same time show details for each node. As mentioned above, it uses the Cypher language, which is very powerful and expressive, specifically designed for working with graph data. In this tool, we use the **Neo4j version 5.21.0.**

Starting with the 'runrecon.sh' file, before its end and after the 'checkservices.sh' script call and execution, contains the code as it appears in figure 53:

```
haveresults=0
```

```
if [[ -s "$filefinalcsvresultspath" ]]; then
    ./prepareNeo4jFiles.sh $FINALDOMAIN $filesubdomainPath $fileduplicatesPath
$filenamePath $filefinalcsvresultspath $fileopenportscsvpath
    haveresults=1
fi
```

Figure 53. The command which executes the 'prepareNeo4jFiles.sh' file/script

As we see in the figure above, the boolean variable "*haveresults*" is initialized with the value '0'. It will take the value '1' if we have results to modify and pass to Neo4j database system. In fact, with the "if-then" statement that follows, we check if any result found by checking the final results csv file. The -s flag returns 'True' if the file's size is greater than zero and, in that case, which means that we have results to visualize, the following command (script file) runs and creates new files from the results files, with the appropriate formation, in order to be passes into the Neo4j database.
"**./prepareNeo4jFiles.sh $FINALDOMAIN $filesubdomainPath $fileduplicatesPath $filenamePath $filefinalcsvresultspath $fileopenportscsvpath"**
Then the 'haveresults' flag is set to 1 in order to permit to run the Neo4j graph creation python script.

4.13.2. Files creation and formatting

Describing the '**prepareNeo4jFiles.sh**' file, it starts with the command:

```
mkdir -p -m 777 report/$1/neo4jVisuals
```

With this command the tool creates a subfolder in the 'report' folder of the current domain name that is being scanned, with the name 'neo4jVisuals'.

Then, it takes the subdomains txt file (2nd positional argument) and creates the 'subdomains.csv' file in which adds the header "subdomain" with the following commands:

```
echo "subdomain" > report/$1/neo4jVisuals/subdomains.csv
cat $2 >> report/$1/neo4jVisuals/subdomains.csv
csv_subdomains_path="report/$1/neo4jVisuals/subdomains.csv"
```

Then similarly, takes the unique ips txt file (3rd argument), adds the header "ip" and creates the 'uniqueips.csv' file, with the following commands:

```
echo "ip" > report/$1/neo4jVisuals/uniqueips.csv
cat $3 >> report/$1/neo4jVisuals/uniqueips.csv
csv_uniqueips_path="report/$1/neo4jVisuals/uniqueips.csv"
```

Afterwards, it takes the results txt file (4th argument) which contains the found subdomains along with their corresponding ips, separated with spaces (one pair per line) and then, the tool replaces the spaces with commas, adds the header "subdomain,ip" (for 2 columns) and creates the 'domain_ips.csv' file, with the following commands:

```
echo "subdomain,ip" > report/$1/neo4jVisuals/domain_ips.csv
cat $4 >> report/$1/neo4jVisuals/domain_ips.csv
csv_domain_ips_path="report/$1/neo4jVisuals/domain_ips.csv"
content=$(cat "report/$1/neo4jVisuals/domain_ips.csv")
modif=${content// /,}
echo "$modif" > "report/$1/neo4jVisuals/domain_ips.csv"
```

The next command creates a copy of the last csv results file (5th positional argument) and gives the name 'services.csv':

```
cat $5 > report/$1/neo4jVisuals/services.csv
```

In the last step, the tool runs the code as described in the following figure:

```
echo "IP,Ports" > "report/$1/neo4jVisuals/open_ports.csv"
# read the csv input file line by line
while IFS= read -r line; do
    # Extract IPs and the ports
    ip=$(echo $line | cut -d',' -f1)
    ports=$(echo $line | cut -d',' -f2- | tr -d '\n' | tr -d '\r')
    modified_ports="[$ports]"

    # write the formatted data to the open_ports.csv file
    echo "$ip,\"$modified_ports\"" >> "report/$1/neo4jVisuals/open_ports.csv"
done < $6
```

Figure 54. Modifying the form of final results in order to be parsed into the Neo4j database graph

Giving a short description of the code of figure 54, the tool creates a file named ‘open_ports.csv’ and adds the header “IP,Ports” (for the 2 columns). Then it takes the input file, which is the csv file with the open ports results (6th argument) and reads it line-by-line. Then modifies each line, removes the spaces, and takes all the found ports for each ip (2nd value of each line and then) and creates a list of ports from them. Then it stores the pairs “**ip,”[port1,port2,port3,…]**” one in each line, into the ‘open_ports.csv’ file.

Now that we have reached the point where we have all the necessary files to create a Neo4j graph, we go back to the ‘runrecon.sh’ file which proceeds with the next command which actually runs the ‘check_neo4j_running’ function, and is as follows:

```
returnvalue="$(check_neo4j_running)"
```

In the above command we create the “returnvalue” variable which stores the result of the “check_neo4j_running” function result. This function exists in the “func.sh” file and checks if Neo4j database management system is running. The corresponding code is in figure 55:

```
check_neo4j_running() {
    if neo4j status | grep "is running at" --quiet; then
        val=1
    else
        val=0
    fi
echo $val
}
```

Figure 55. ‘check_neo4j_running’ Bash function which checks if “Neo4j” GDBMS is running

The above function checks if Neo4j dbms is running and if it is, prints as output the '1'. Then this output is stored in the ‘returnvalue’ variable. In an opposite case it prints the ‘0’.

Next, are following some checks of the ‘returnvalue’ and ‘haveresults’ variables. In case that both ‘returnvalue’ and ‘haveresults’ are equal to ‘1’, the tool runs the “pyToNeo.py” python3 script that will be described in a while, which creates the graph from the current domain’s results and then shows to the user the URL where that graph is available to be opened (*http://localhost:7474*).

In case the ‘returnvalue’ is equal to ‘0’ but the ‘haveresults’ is equal to ‘1’, the tool prints an informative message that the Neo4j service is not running, prompts the user to start the Neo4j service and then shows the command that the user can write for the current domain name in order to achieve the final results visualization. The command is “python3 pyToNeo.py $FINALDOMAIN” where in the FINALDOMAIN variable it will print the currently investigated domain name.

Last, if the ‘haveresults’ is equal to ‘0’ it means that we have no data to visualize, and then, whether the Neo4j is running or not, the tool does nothing more but prints a related message to the user. All the above actions are the result of the following code execution (figure 56):

```
if [[ $returnvalue -eq 1 && $haveresults -eq 1 ]]; then
    python3 pyToNeo.py $FINALDOMAIN
    echo -e "${GREEN}\nNeo4j graph is available at
http://localhost:7474${ENDCOLOR}"
fi

if [[ $haveresults -eq 0 ]]; then
    echo "No results found. No graph for data visualization will be created"
fi

if  [[ $returnvalue -eq 0 && $haveresults -eq 1 ]];then
    echo "Neo4j is not running. Start neo4j service and then run the command
\"python3 pyToNeo.py $FINALDOMAIN \""
    echo "After, the Graph will be available at http://localhost:7474"
fi
```

Figure 56. Checks if the Neo4j GDBMS is running and if we have final results to visualize

4.13.3. From CSV to Graph creation [37]

Now, we are in the case that both ‘returnvalue’ and ‘haveresults’ variables are equal to ‘1’ and the ‘pyToNeo.py’ script is executed. This script uses the ‘**pandas’** open-source library which is ideal for data manipulation and analysis in python language. With this library we can handle the csv files that we created before, quite efficiently and integrate with the Neo4j graph database management system.

Firstly, with the commands of figure 57, the script creates variables that contain the paths of all csv files that the tool will need and then creates ‘pandas’ library dataframes from them:

```
subdomains_path = 'report/'+target+'/neo4jVisuals/subdomains.csv'
unique_ips_path = 'report/'+target+'/neo4jVisuals/uniqueips.csv'
domain_ips_path = 'report/'+target+'/neo4jVisuals/domain_ips.csv'
services_path = 'report/'+target+'/neo4jVisuals/services.csv'
open_ports_path = 'report/'+target+'/neo4jVisuals/open_ports.csv'
subdomains_df = pd.read_csv(subdomains_path)
unique_ips_df = pd.read_csv(unique_ips_path)
domain_ips_df = pd.read_csv(domain_ips_path)
open_ports_df = pd.read_csv(open_ports_path)

if not is_file_empty(services_path):
    services_df = pd.read_csv(services_path)
```

Figure 57. Neo4j graph’s necessary variables initialization and ‘pandas’ dataframes creation

In the script, the function “is_file_empty()” is defined and the tool executes it in order to check if a file is empty. We created this function in order to avoid errors that occur while ‘pandas’ library tried to handle zero size files. The above function ‘is_file_empty()’ and a usage of it, appear in the script as shown in the next figure:

```
def is_file_empty(file_path):
    return os.path.getsize(file_path) == 0

# Checks if exist results of services in 'services.csv' file
if not is_file_empty(services_path):
    services_df = pd.read_csv(services_path)
```

Figure 58. 'is_file_empty' function checks the files that will be imported to 'pandas' commands

Then the script connects to the graph database system Neo4j using the username "neo4j" and the password "neo4j12345" and cleans the database from any previous graphs:

```
# Connect to Neo4j
graph = Graph("bolt://localhost:7687", auth=("neo4j", "neo4j12345"))

# Clear existing graph data
graph.delete_all()
```

Figure 59. Neo4j database connection

Next, the creation of a main node in the graph that represents the main target-domain is created with the following command, with the domain name to be passed as an argument from the beginning of this script execution:

```
# Creation of main node of the target-domain
domain_node = Node("Domain", name=domain)
graph.create(domain_node)
```

Figure 60. First main node of Neo4j graph creation

The creation of subdomain nodes and the relationships ("ContainSubdomain" connection vectors) with the main domain node are created with the following code:

```
# Create subdomain nodes and relationships
for _, row in subdomains_df.iterrows():
    subdomain_node = Node("Subdomain", name=row['subdomain'])
    graph.create(subdomain_node)
    if domain_node:
        contains_subdomain_rel = Relationship(domain_node, "ContainsSubdomain",
subdomain_node)
        graph.create(contains_subdomain_rel)
```

Figure 61.Subdomains creation as nodes in the Neo4j graph

The creation of unique ips nodes is achieved by the following:

```
# Create ips unique nodes
for _, row in unique_ips_df.iterrows():
    ipname = str(row['ip'])
    ip_node = Node("IP", address=row['ip'])
    graph.create(ip_node)
```

Figure 62. IPs nodes creation in the Neo4j graph.

Now it is the time to build the (ResolveTo) relationships between subdomains and ips. As it is expected, each ip can be connected from one-to-many subdomains, because very often subdomains of the same domain name resolve to the same ip. For this case, we iterate the "domain_ips" dataframe and for each subdomain we create a relationship with the previously created unique IP nodes. The code that achieves this is in figure 63:

```
 # Create IP-Subdomains relationships
for _, row in domain_ips_df.iterrows():
    current_subdomain_node = graph.nodes.match("Subdomain", name=row['subdomain']).first()
    # List with IP nodes
    ip_nodes = list(graph.nodes.match("IP", address=row['ip']))
    for node_a in ip_nodes:
        common_ip = str(node_a['address'])
        if row['ip'] == common_ip:
                resolve_to_rel = Relationship(current_subdomain_node, "ResolveTo", node_a)
                graph.create(resolve_to_rel)
```

Figure 63. Relationships creation as vectors between IPs, Domain and Subdomains

Finally, in the end we check if the final results csv file is empty and if it is empty we end the tool execution and the graph will not contain any service node but only the domain, subdomains and ips nodes.

If the csv file size is not equal to zero, the tool continues with the creation of a node for each line of the results file (for each port/service found to be running). Each node will have as name, the name of the service type that represents, will be connected with the ip node that is related according to the csv file values, and will include several information fields, like “Service” type, “portnum”, “protocol”, “CPE”, etc. Then the appropriate (ContainsService) relationships will be added to the graph. The above can be achieved with the code in the following snippet (figure 64):

```
if is_file_empty(services_path):
    exit(1)
else:
    # Create service nodes and relationships
    for _, row in services_df.iterrows():
        ip_node = graph.nodes.match("IP", address=row['address']).first()
        if ip_node:
            service_node = Node("Service", port=row['portnum'], protocol=row['protocol'],
state=row['portState'], cpe=row['CPE'],
                                service_name=row['servName'],
confidence=row['servConfidence'], method=row['scanMethod'], product=row['Product'])
            graph.create(service_node)
            contains_service_rel = Relationship(ip_node, "ContainsService", service_node)
            graph.create(contains_service_rel)
```

Figure 64. Creation of running services nodes

The user can now display on the local host page ‘http://localhost:7474” the results in a more appealing and figurative way.

All code files, along with descriptions for the installations, are available in the project’s repository.

You can access the repository through the following link:

https://github.com/gtascyber/recontool2.git

4.13.4. States of tool - "Steps of Actions" diagram

Summarizing, all the above procedure which could be characterized as an “algorithm” for an automated online target-domain reconnaissance for live services, is depicted briefly to the following figure 65, where we can see clearly each file and almost each function or command of it, as well as which one of them makes calls to the other files or functions:

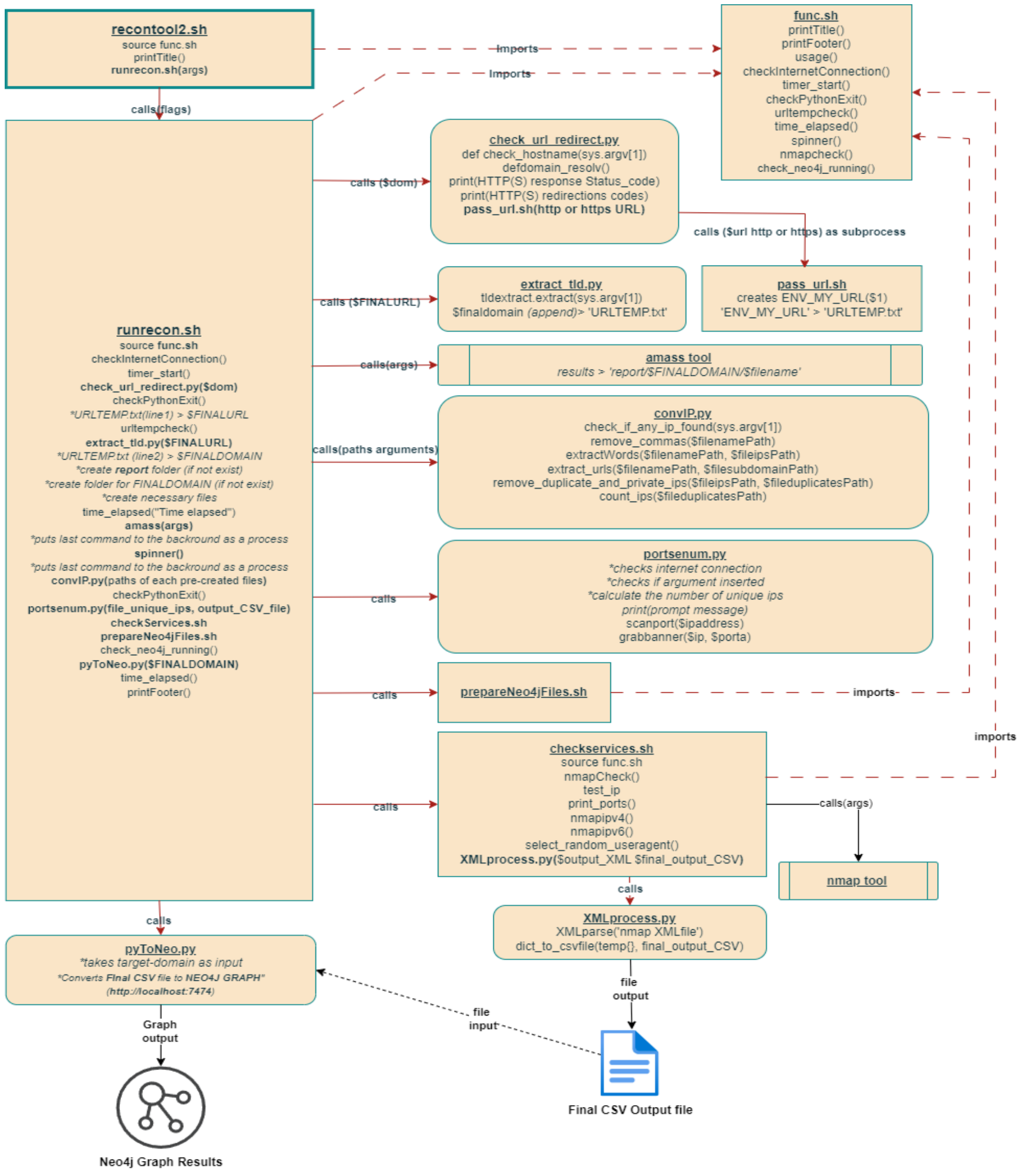
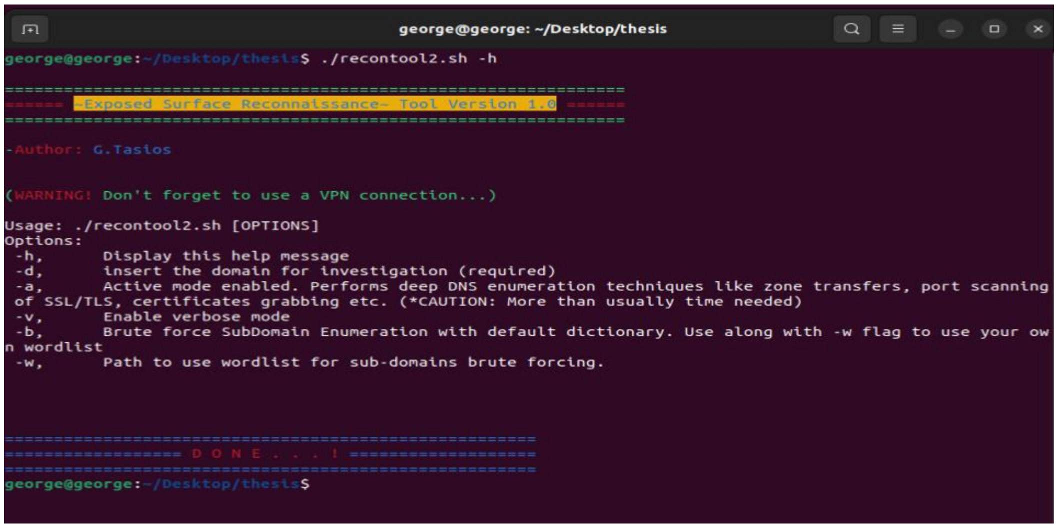

recontool2.sh
source func.sh
printTitle()
runrecon.sh(args)
Imports
Imports
func.sh
printTitle()
printFooter()
usage()
checkInternetConnection()
timer_start()
checkPythonExit()
urltempcheck()
time_elapsed()
spinner()
nmapcheck()
check_neo4j_running()
calls(flags)
check_url_redirect.py
def check_hostname(sys.argv[1])
defdomain_resolv()
print(HTTP(S) response Status_code)
print(HTTP(S) redirections codes)
pass_url.sh(http or https URL)
calls ($dom)
calls ($url http or https) as subprocess
extract_tld.py
tldextract.extract(sys.argv[1])
$finaldomain (append)> 'URLTEMP.txt'
calls ($FINALURL)
pass_url.sh
creates ENV_MY_URL($1)
'ENV_MY_URL' > 'URLTEMP.txt'
amass tool
results > 'report/$FINALDOMAIN/$filename'
calls(args)
runrecon.sh
source func.sh
checkInternetConnection()
timer_start()
check_url_redirect.py($dom)
checkPythonExit()
*URLTEMP.txt(line1) > $FINALURL
urltempcheck()
extract_tld.py($FINALURL)
*URLTEMP.txt (line2) > $FINALDOMAIN
*create report folder (if not exist)
*create folder for FINALDOMAIN (if not exist)
*create necessary files
time_elapsed("Time elapsed")
amass(args)
*puts last command to the backround as a process
spinner()
*puts last command to the backround as a process
convIP.py(paths of each pre-created files)
checkPythonExit()
portsenum.py(file_unique_ips, output_CSV_file)
checkServices.sh
prepareNeo4jFiles.sh
check_neo4j_running()
pyToNeo.py($FINALDOMAIN)
time_elapsed()
printFooter()
calls(paths arguments)
convIP.py
check_if_any_ip_found(sys.argv[1])
remove_commas($filenamePath)
extractWords($filenamePath, $fileipsPath)
extract_urls($filenamePath, $filesubdomainPath)
remove_duplicate_and_private_ips($fileipsPath, $fileduplicatesPath)
count_ips($fileduplicatesPath)
portsenum.py
*checks internet connection
*checks if argument inserted
*calculate the number of unique ips
print(prompt message)
scanport($ipaddress)
grabbanner($ip, $porta)
calls
prepareNeo4jFiles.sh
calls
imports
imports
checkservices.sh
source func.sh
nmapCheck()
test_ip
print_ports()
nmapipv4()
nmapipv6()
select_random_useragent()
XMLprocess.py($output_XML $final_output_CSV)
calls
calls(args)
nmap tool
calls
XMLprocess.py
XMLparse('nmap XMLfile')
dict_to_csvfile(temp{}, final_output_CSV)
calls
pyToNeo.py
*takes target-domain as input
Converts Final CSV file to NEO4J GRAPH
(http://localhost:7474)
file output
file input
Graph output
Final CSV Output file
Neo4j Graph Results

Figure 65. States of tool - "Steps of Actions" diagram

5. Validation

In this stage we test the tool of this thesis, showing below in the following figures the output we get while its commands (script files) are executed. The following tests were performed on domain names of corporate environments, with the permission of the network operators, but we do not have permission to report or publish to whom the results belong or to disclose specific information about them, so as not to show or point a specific network infrastructure in case a vulnerability or serious security gap occurred.

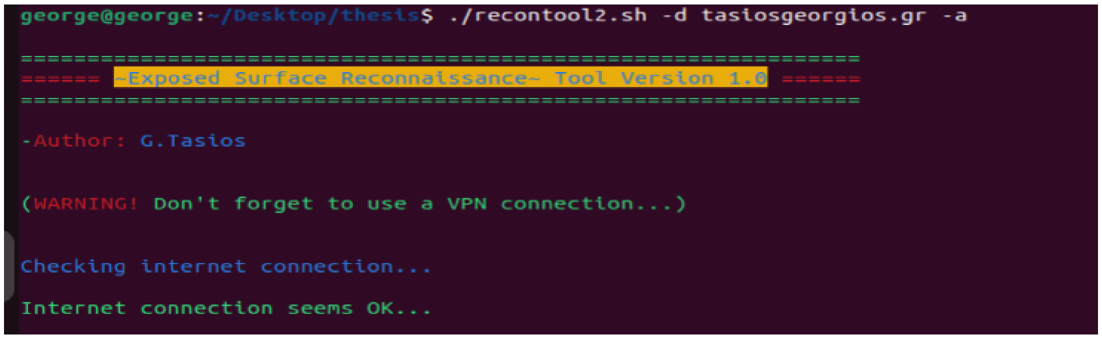
Starting with the tests of our tool, the function 'printTitle' is executed any time our tool starts and the output of the execution along with the 'usage' function is shown in figure 66:



```
george@george: ~/Desktop/thesis
george@george:~/Desktop/thesis$ ./recontool2.sh -h

==============================================================
====== ~Exposed Surface Reconnaissance~ Tool Version 1.0 ======
==============================================================

-Author: G.Tasios

(WARNING! Don't forget to use a VPN connection...)

Usage: ./recontool2.sh [OPTIONS]
Options:
 -h,      Display this help message
 -d,      insert the domain for investigation (required)
 -a,      Active mode enabled. Performs deep DNS enumeration techniques like zone transfers, port scanning
 of SSL/TLS, certificates grabbing etc. (*CAUTION: More than usually time needed)
 -v,      Enable verbose mode
 -b,      Brute force SubDomain Enumeration with default dictionary. Use along with -w flag to use your ow
n wordlist
 -w,      Path to use wordlist for sub-domains brute forcing.



=====================================================
================== D O N E . . . ! ==================
=====================================================
george@george:~/Desktop/thesis$
```

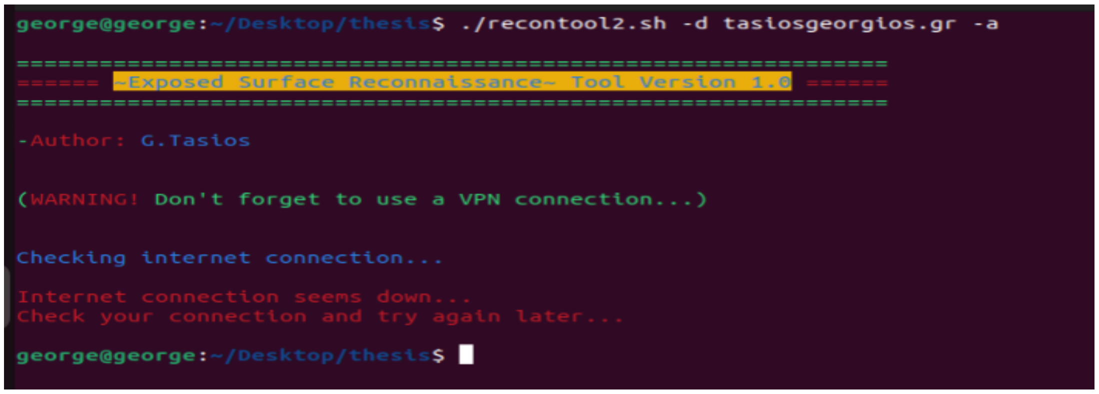
Figure 66. Banner (title) of the tool while it starts

Then the tool checks the internet connection by executing the function 'checkInternetConnection'. The output of this function with the internet connection up and then down, is depicted in the following figures 67-68:



```
george@george:~/Desktop/thesis$ ./recontool2.sh -d tasiosgeorgios.gr -a

==============================================================
====== ~Exposed Surface Reconnaissance~ Tool Version 1.0 ======
==============================================================

-Author: G.Tasios

(WARNING! Don't forget to use a VPN connection...)

Checking internet connection...

Internet connection seems OK...
```

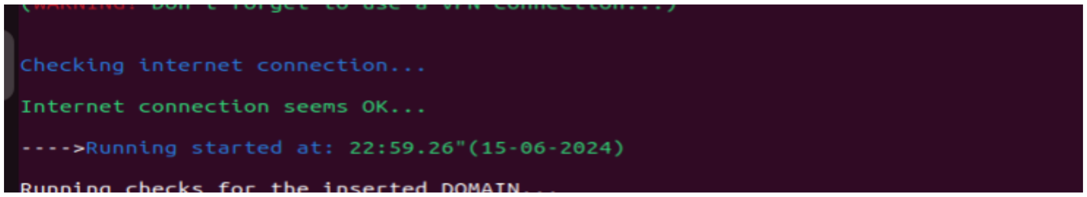
Figure 67. Internet connection (up) check function output



```
george@george:~/Desktop/thesis$ ./recontool2.sh -d tasiosgeorgios.gr -a

==============================================================
====== ~Exposed Surface Reconnaissance~ Tool Version 1.0 ======
==============================================================

-Author: G.Tasios

(WARNING! Don't forget to use a VPN connection...)

Checking internet connection...

Internet connection seems down...
Check your connection and try again later...

george@george:~/Desktop/thesis$
```

Figure 68. Internet connection (down) check function output

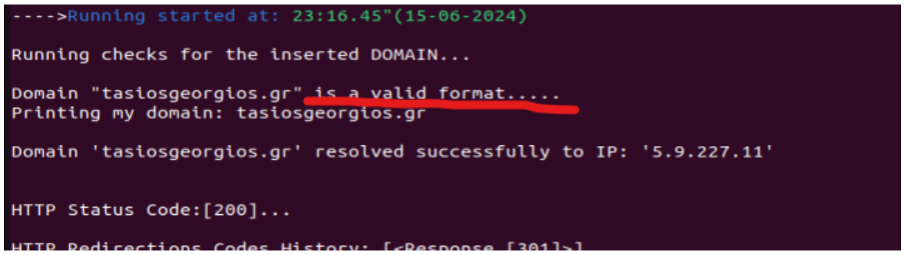
Then the function 'timer_start' is automatically executed in addition to the above. Its execution, prints the time that the tool run started, as shown in figure 69:



```
Checking internet connection...

Internet connection seems OK...

---->Running started at: 22:59.26"(15-06-2024)

Running checks for the inserted DOMAIN...
```

Figure 69. Tool's starting timestamp

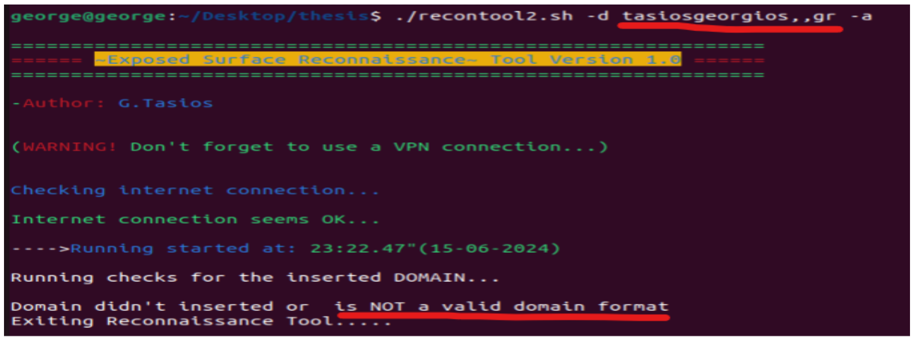
The next function that is executed is the '**check_hostname**' which checks and verifies if the inserted domain name is in a valid format. The results shown to the next figure:



```
---->Running started at: 23:16.45"(15-06-2024)

Running checks for the inserted DOMAIN...

Domain "tasiosgeorgios.gr" is a valid format.....
Printing my domain: tasiosgeorgios.gr

Domain 'tasiosgeorgios.gr' resolved successfully to IP: '5.9.227.11'


HTTP Status Code:[200]...

HTTP Redirections Codes History: [<Response [301]>]
```

Figure 70. Domain form validation check & result

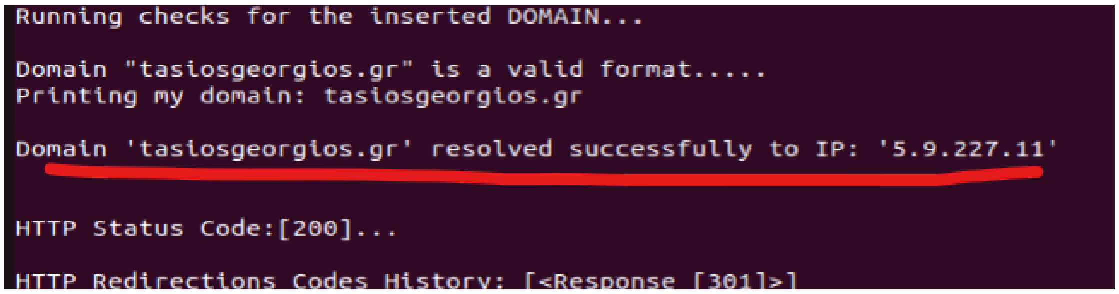
The same as above but with an invalid domain format inserted is depicted in figure 71:



```
george@george:~/Desktop/thesis$ ./recontool2.sh -d tasiosgeorgios,,gr -a

==============================================================
====== ~Exposed Surface Reconnaissance~ Tool Version 1.0 ======
==============================================================

-Author: G.Tasios

(WARNING! Don't forget to use a VPN connection...)

Checking internet connection...

Internet connection seems OK...

---->Running started at: 23:22.47"(15-06-2024)

Running checks for the inserted DOMAIN...

Domain didn't inserted or  is NOT a valid domain format
Exiting Reconnaissance Tool.....
```

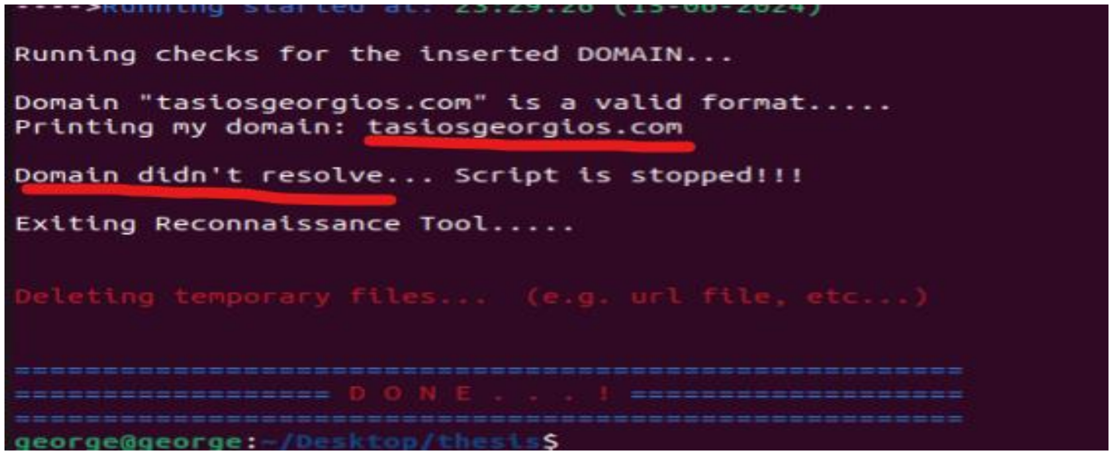
Figure 71. Domain name form validation fails

Then the tool checks if the inserted domain resolves to an IP address and accordingly prints a message and continues. The results of a successful domain resolution and of a failed one, are shown in figure 72 and figure 73:



```
Running checks for the inserted DOMAIN...

Domain "tasiosgeorgios.gr" is a valid format.....
Printing my domain: tasiosgeorgios.gr

Domain 'tasiosgeorgios.gr' resolved successfully to IP: '5.9.227.11'


HTTP Status Code:[200]...

HTTP Redirections Codes History: [<Response [301]>]
```

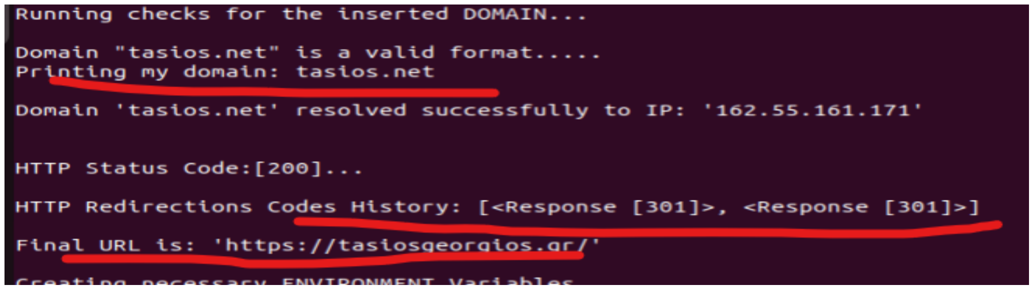
Figure 72. Results when a domain name successfully resolves to an IP address



```
Running checks for the inserted DOMAIN...

Domain "tasiosgeorgios.com" is a valid format.....
Printing my domain: tasiosgeorgios.com

Domain didn't resolve... Script is stopped!!!

Exiting Reconnaissance Tool.....


Deleting temporary files...  (e.g. url file, etc...)


===================================================================
=================== D O N E . . . ! ===============================
===================================================================
george@george:~/Desktop/thesis$
```

Figure 73. Results when a domain name does not resolve to an IP address

Afterwards, the tool checks the target domain name if it redirects to another one. The above redirection is depicted clearly to the following figure 74, where for example, despite that we inserted the 'tasios.net'[39] domain name as a target domain, the tool found that it redirects to the 'tasiosgeorgios.gr'[39] and turned in that direction to investigate:



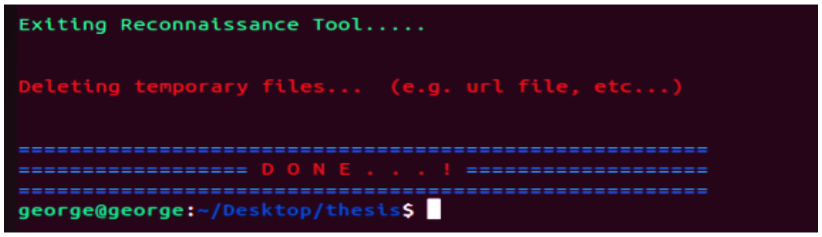
```
Running checks for the inserted DOMAIN...

Domain "tasios.net" is a valid format.....
Printing my domain: tasios.net

Domain 'tasios.net' resolved successfully to IP: '162.55.161.171'


HTTP Status Code:[200]...

HTTP Redirections Codes History: [<Response [301]>, <Response [301]>]

Final URL is: 'https://tasiosgeorgios.gr/'
```

Figure 74. Domain name redirection found

[39] '***tasios.net***' and '***tasiosgeorgios.gr***' are existed domain names which are administered by the author and are used in this Thesis for testing and demonstration reasons

In many cases the script terminates its execution after various checks that it does during it and prints an information footer using the aforementioned ‘printFooter’ bash function. The results of the above is shown in the next figure:

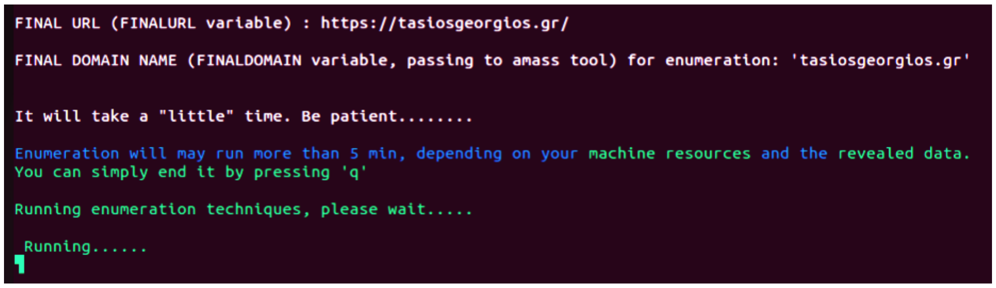
The abovementioned footer that prints the ‘printFooter’ function is shown next, in figure 75:



```
Exiting Reconnaissance Tool.....

Deleting temporary files...  (e.g. url file, etc...)


=====================================================
================== D O N E . . . ! ===================
=====================================================
george@george:~/Desktop/thesis$
```

Figure 75. Tool's footer printed by the ‘printFooter’ function

In the next steps of execution the tool starts with the abovementioned command the Amass tool and initiates the enumeration. While the enumeration is running the previously described spinner runs on the screen to inform the user that the enumeration is still going. What the tool prints while it runs, is shown to the next figure where we notice the spinner’s moving characters:

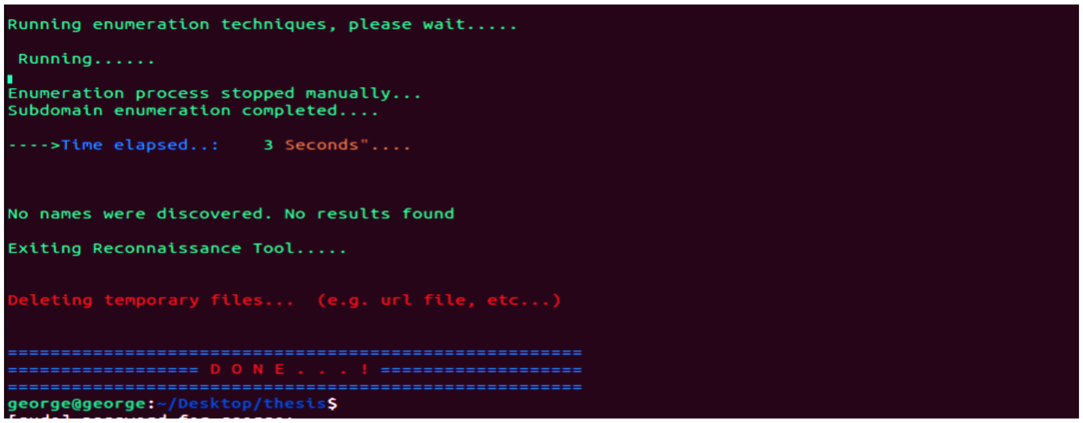


```
FINAL URL (FINALURL variable) : https://tasiosgeorgios.gr/

FINAL DOMAIN NAME (FINALDOMAIN variable, passing to amass tool) for enumeration: 'tasiosgeorgios.gr'


It will take a "little" time. Be patient........

Enumeration will may run more than 5 min, depending on your machine resources and the revealed data.
You can simply end it by pressing 'q'

Running enumeration techniques, please wait.....

 Running......
```

Figure 76. (sub)domain enumeration and resolving phase while “running spinner” chars appear

We notice the prompt to press the ‘q’ button, that appears before the spinner starts, and informs the user that have the choice to terminate the enumeration procedure.

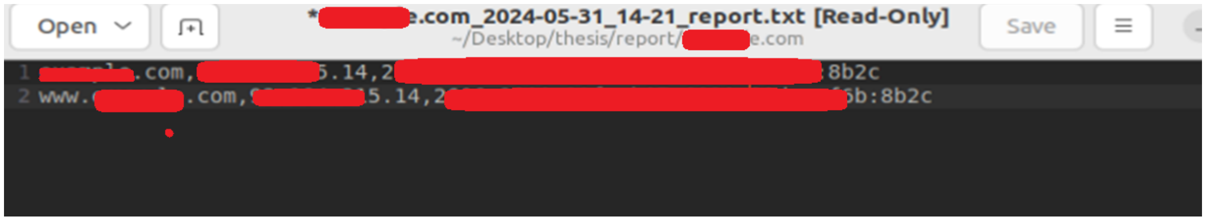
The forced termination of the tool by the user when presses the ‘q’ button and the printing of the elapsed time since the start of execution of this tool is shown in figure 77:



```
Running enumeration techniques, please wait.....

 Running......

Enumeration process stopped manually...
Subdomain enumeration completed....

---->Time elapsed..:    3 Seconds"....



No names were discovered. No results found

Exiting Reconnaissance Tool.....


Deleting temporary files...  (e.g. url file, etc...)


=====================================================
================== D O N E . . . ! ===================
=====================================================
george@george:~/Desktop/thesis$
```

Figure 77. Forced termination of the tool by the user

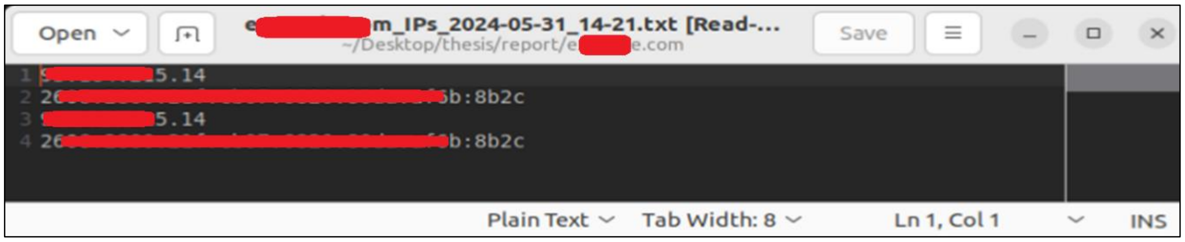
After a successful enumeration the tool stores the “pure” results. An example of such results is shown in the figure 78:



```
*[redacted]e.com_2024-05-31_14-21_report.txt [Read-Only]
~/Desktop/thesis/report/[redacted]e.com

[redacted].com,[redacted]5.14,2[redacted]8b2c
www.[redacted].com,[redacted]5.14,2[redacted]b:8b2c
```

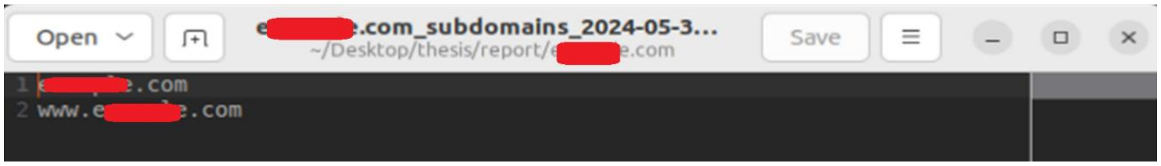
Figure 78. first results of subdomain enumeration, with IP resolving

After the results automated editing by the tool, we take the extracted IPs into a separate file as well as the extracted subdomains into another one, as they are depicted in the next figures 79-80:



```
e[redacted]m_IPs_2024-05-31_14-21.txt [Read-...
~/Desktop/thesis/report/e[redacted]e.com

[redacted]5.14
26[redacted]b:8b2c
[redacted]5.14
26[redacted]b:8b2c
```

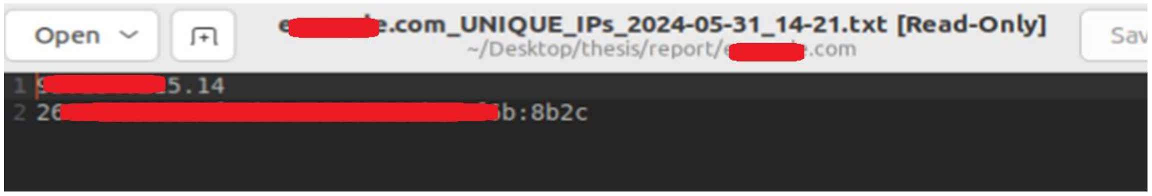
Figure 79. Just extracted IPs (with no further processing)



```
e[redacted].com_subdomains_2024-05-3...
~/Desktop/thesis/report/e[redacted]e.com

[redacted].com
www.e[redacted].com
```

Figure 80. Only subdomains extracted and stored in this file

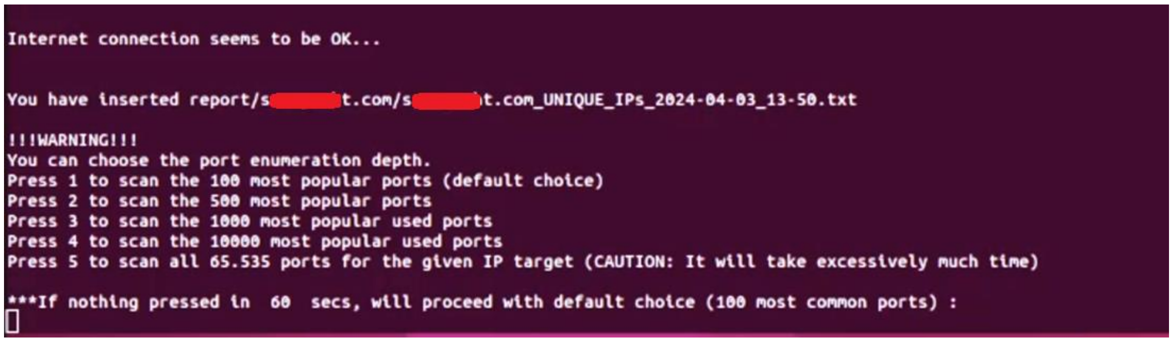
In the next step we examine the ‘xxxxxxx.xxx_UNIQUE_IPs_” file that is created and contains only the global IPs from the found ones, after the removal of duplicate IPs. The results are in figure 81:



```
e[redacted].com_UNIQUE_IPs_2024-05-31_14-21.txt [Read-Only]
~/Desktop/thesis/report/[redacted].com

[redacted]5.14
26[redacted]b:8b2c
```

Figure 81. Unique IPs file (no local IPs or duplicates)

In the next step the tool prints a message that prompts the user to choose the port enumeration depth, by pressing a button (from 1 to 5) depending on what depth wishes to run the port enumeration. The above message is shown in figure 82:

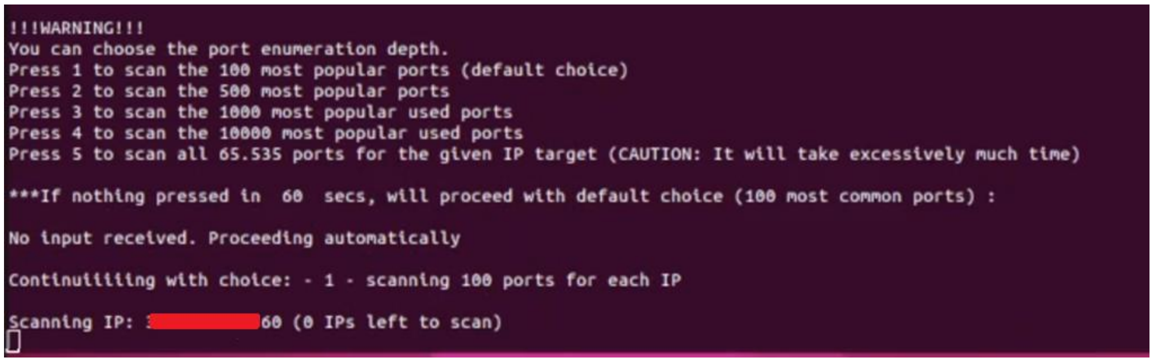


```
Internet connection seems to be OK...

You have inserted report/s[redacted]t.com/s[redacted]t.com_UNIQUE_IPs_2024-04-03_13-50.txt

!!!WARNING!!!
You can choose the port enumeration depth.
Press 1 to scan the 100 most popular ports (default choice)
Press 2 to scan the 500 most popular ports
Press 3 to scan the 1000 most popular used ports
Press 4 to scan the 10000 most popular used ports
Press 5 to scan all 65.535 ports for the given IP target (CAUTION: It will take excessively much time)

***If nothing pressed in  60  secs, will proceed with default choice (100 most common ports) :
```

Figure 82. Prompts the user to select a choice for ports scanning

Next, the tool prints a message and prompts the user to make a choice from ‘1’ to ‘4’ which means that 100, 500, 1000 or 10.000 ports selected respectively to enumerate.

A sample of the execution of the above part of code is depicted in the following figure:

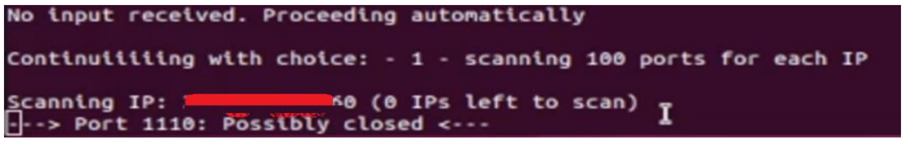


```
!!!WARNING!!!
You can choose the port enumeration depth.
Press 1 to scan the 100 most popular ports (default choice)
Press 2 to scan the 500 most popular ports
Press 3 to scan the 1000 most popular used ports
Press 4 to scan the 10000 most popular used ports
Press 5 to scan all 65.535 ports for the given IP target (CAUTION: It will take excessively much time)

***If nothing pressed in  60  secs, will proceed with default choice (100 most common ports) :

No input received. Proceeding automatically

Continuiiiing with choice: - 1 - scanning 100 ports for each IP

Scanning IP: [redacted]60 (0 IPs left to scan)
```

Figure 83. Proceeding to default choice for ports' scanning (no user selection within 60’’)

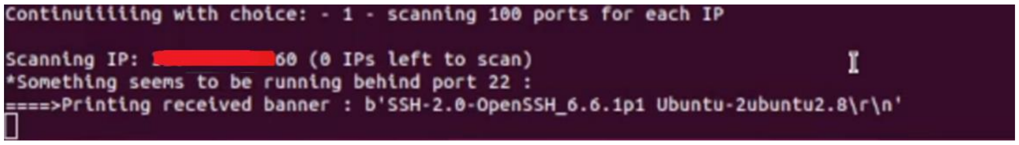
In the above shown case, we didn’t choose any option, didn’t insert any number and the tool proceeded with the default option ‘1’, to enumerate the 100 most popular and common ports.

Now we reached at the point where the selected ports, according to user’s choice, will be scanned for running services. After the ‘scanport’ and ‘bannergrabber’ functions were executed, as previously explained, we get the results as shown in the cases in the following two figures 84 and 85, one of a closed port and one of a port which found open and the corresponding banner grabbed:



```
No input received. Proceeding automatically

Continuiiiing with choice: - 1 - scanning 100 ports for each IP

Scanning IP: [redacted]60 (0 IPs left to scan)
--> Port 1110: Possibly closed <---
```

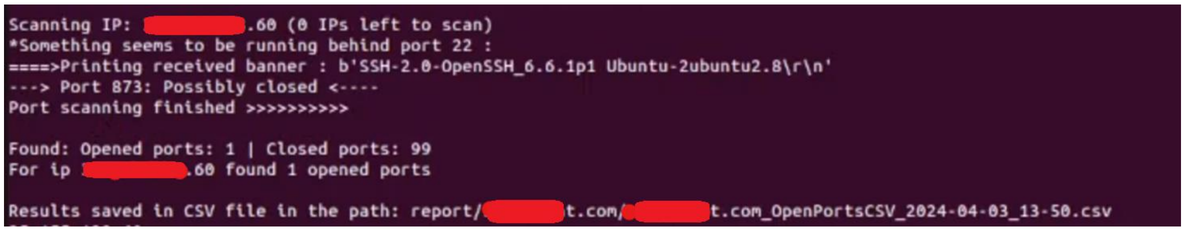
Figure 84. Example of port found closed with “BannerGrabbing” technique



```
Continuiiiing with choice: - 1 - scanning 100 ports for each IP

Scanning IP: [redacted]60 (0 IPs left to scan)
*Something seems to be running behind port 22 :
====>Printing received banner : b'SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8\r\n'
```

Figure 85. Example of an open port found. Function prints the “grabbed” banner

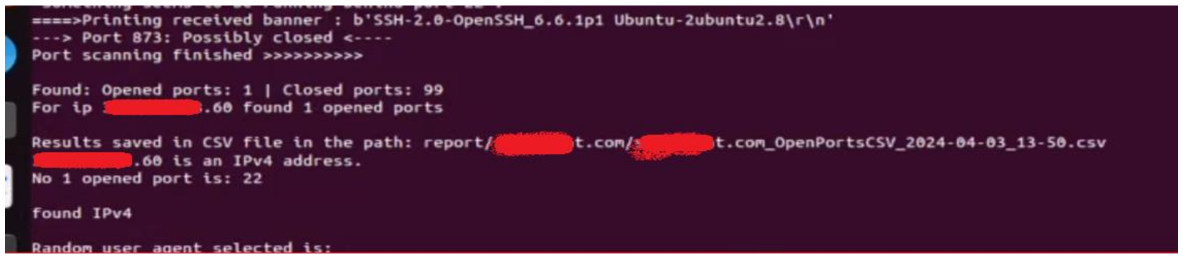
At the end of the current step, we get the information about the number of open and closed ports, about the numbers of the ports that found open and the file location where they were stored, as in the example shown in the next figure:



```
Scanning IP: [redacted].60 (0 IPs left to scan)
*Something seems to be running behind port 22 :
====>Printing received banner : b'SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8\r\n'
---> Port 873: Possibly closed <----
Port scanning finished >>>>>>>>>>

Found: Opened ports: 1 | Closed ports: 99
For ip [redacted].60 found 1 opened ports

Results saved in CSV file in the path: report/[redacted]t.com/[redacted]t.com_OpenPortsCSV_2024-04-03_13-50.csv
```

Figure 86. Results after port enumeration with banner grabbing technique

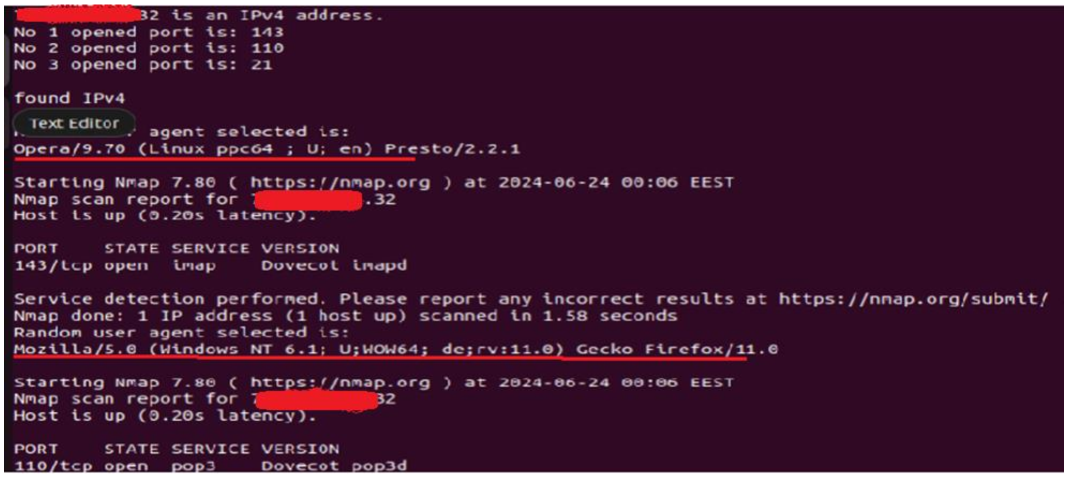
While the tool iterates over each IP address, prints the order of its open ports that will be scanned. In the example shown in figure 87, only one IP found and seems to have only one open port:



```
====>Printing received banner : b'SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.8\r\n'
---> Port 873: Possibly closed <----
Port scanning finished >>>>>>>>>>

Found: Opened ports: 1 | Closed ports: 99
For ip [redacted].60 found 1 opened ports

Results saved in CSV file in the path: report/[redacted]t.com/[redacted]t.com_OpenPortsCSV_2024-04-03_13-50.csv
[redacted].60 is an IPv4 address.
No 1 opened port is: 22

found IPv4

Random user agent selected is:
```

Figure 87. List of found open ports for the current IP addresses

Next, the tool has reached the point where the port scanning must start. As already described above, the tool chooses randomly a new user-agent for every port that is going to scan. The above is depicted in the following figure 88 where a specific user-agent is selected:

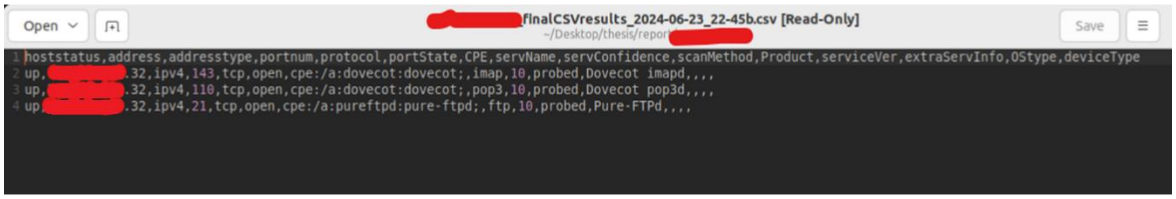


```
[redacted]32 is an IPv4 address.
No 1 opened port is: 143
No 2 opened port is: 110
No 3 opened port is: 21

found IPv4
            agent selected is:
Opera/9.70 (Linux ppc64 ; U; en) Presto/2.2.1

Starting Nmap 7.80 ( https://nmap.org ) at 2024-06-24 00:06 EEST
Nmap scan report for [redacted].32
Host is up (0.20s latency).

PORT     STATE SERVICE VERSION
143/tcp open  imap    Dovecot imapd

Service detection performed. Please report any incorrect results at https://nmap.org/submit/
Nmap done: 1 IP address (1 host up) scanned in 1.58 seconds
Random user agent selected is:
Mozilla/5.0 (Windows NT 6.1; U;WOW64; de;rv:11.0) Gecko Firefox/11.0

Starting Nmap 7.80 ( https://nmap.org ) at 2024-06-24 00:06 EEST
Nmap scan report for [redacted]32
Host is up (0.20s latency).

PORT     STATE SERVICE VERSION
110/tcp open  pop3    Dovecot pop3d
```

Figure 88. An example of randomly new User-agent selection for each port

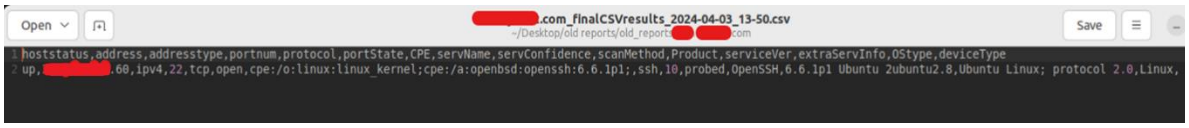
All the above scans of each open port, gives the partial results in an XML format. Then, this XML parsed and edited by the tool and checked if contains any useful information. In a positive case, those results are converted into CSV lines and appended into the Final CSV results file.

While we were testing this tool, we got appropriate results, a sample of which is depicted in the following figure 89 and figure 90:



```
[redacted]finalCSVresults_2024-06-23_22-45b.csv [Read-Only]
~/Desktop/thesis/report[redacted]

hoststatus,address,addresstype,portnum,protocol,portState,CPE,servName,servConfidence,scanMethod,Product,serviceVer,extraServInfo,OStype,deviceType
up,[redacted].32,ipv4,143,tcp,open,cpe:/a:dovecot:dovecot;,imap,10,probed,Dovecot imapd,,,,
up,[redacted].32,ipv4,110,tcp,open,cpe:/a:dovecot:dovecot;,pop3,10,probed,Dovecot pop3d,,,,
up[redacted].32,ipv4,21,tcp,open,cpe:/a:pureftpd:pure-ftpd;,ftp,10,probed,Pure-FTPd,,,,
```

Figure 89. Final CSV results sample 1

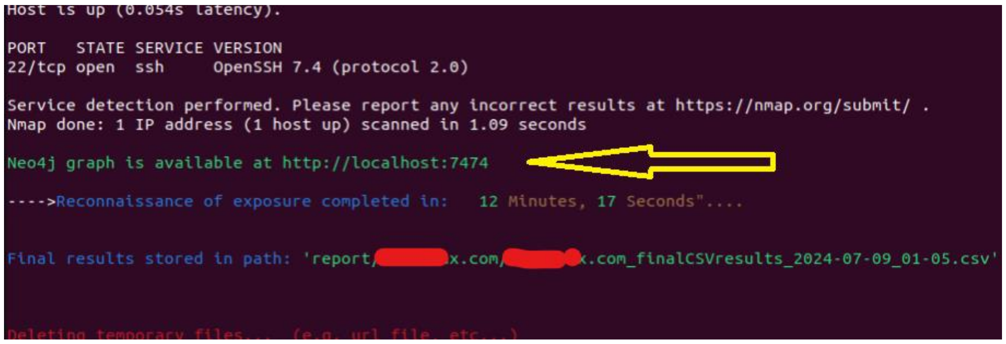


```
[redacted].com_finalCSVresults_2024-04-03_13-50.csv
~/Desktop/old reports/old_reports[redacted][redacted]com

hoststatus,address,addresstype,portnum,protocol,portState,CPE,servName,servConfidence,scanMethod,Product,serviceVer,extraServInfo,OStype,deviceType
up,[redacted].60,ipv4,22,tcp,open,cpe:/o:linux:linux_kernel;cpe:/a:openbsd:openssh:6.6.1p1;,ssh,10,probed,OpenSSH,6.6.1p1 Ubuntu 2ubuntu2.8,Ubuntu Linux; protocol 2.0,Linux,
```

Figure 90. Final CSV results sample 2

After the previous steps, the tool runs various checks concerning the final results. It checks if final results occurred from the previous steps and then modifies them by creating new suitable files to be parsed into the Neo4j GDBMS.

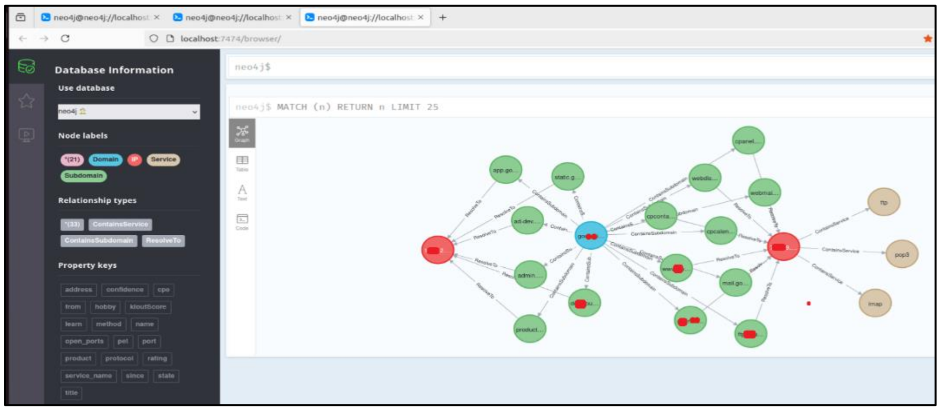
In figure 91 we have a sample of our tests where we see the ending of an accomplished execution of our tool, along with an information line (terminal message) regarding the successful graph generation, using the current final results.



```
Host is up (0.054s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.4 (protocol 2.0)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1.09 seconds

Neo4j graph is available at http://localhost:7474

---->Reconnaissance of exposure completed in:   12 Minutes, 17 Seconds"....


Final results stored in path: 'report/[redacted]x.com/[redacted]x.com_finalCSVresults_2024-07-09_01-05.csv'


Deleting temporary files... (e.g. url file, etc...)
```

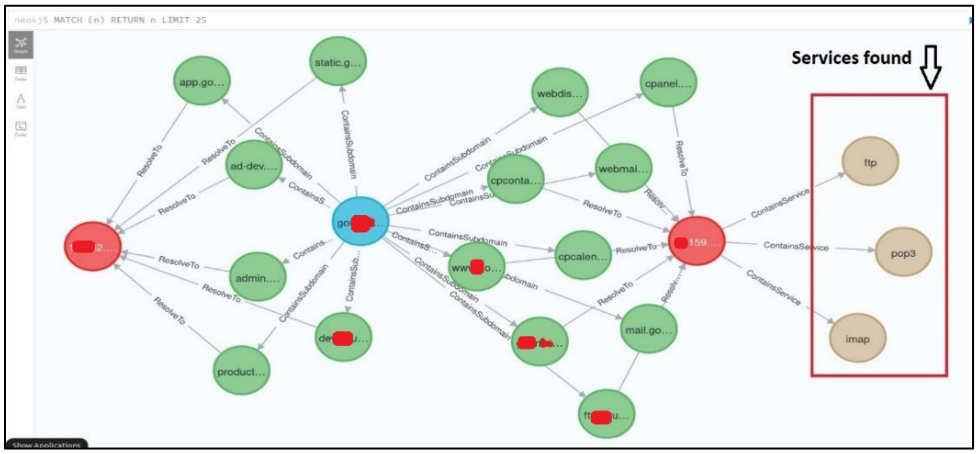
Figure 91. Successfully Neo4j graph generation and its URL

After the tests we run in order to validate our tool, the following pictures (figure 92 to figure 95) depict the Neo4j graphs that were automatically generated:



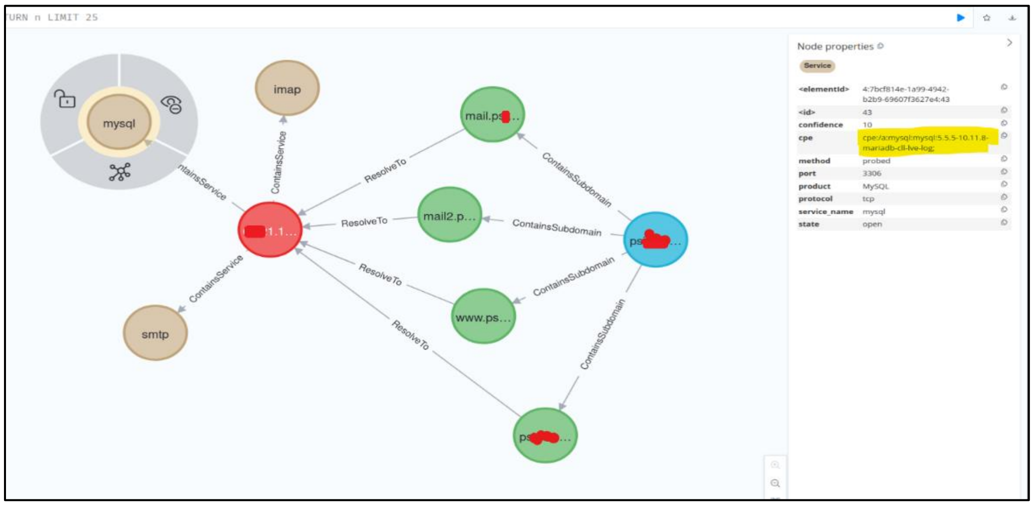
```
neo4j$ MATCH (n) RETURN n LIMIT 25
```

Figure 92. Neo4j results case 1. Showing the node labels on Neo4j DBMS information



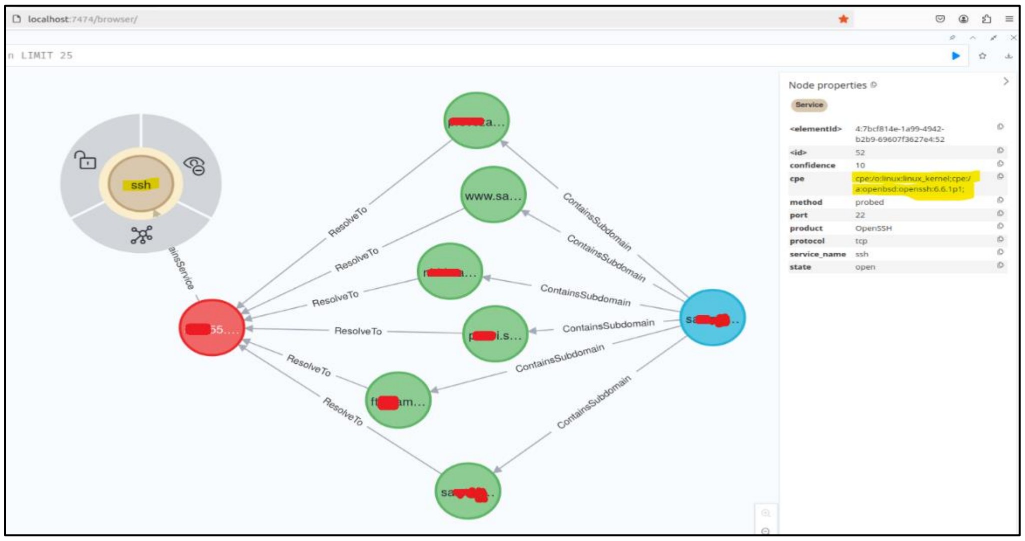
```
neo4j$ MATCH (n) RETURN n LIMIT 25
```

Figure 93. Neo4j results case 1 Graph



<elementId>	4:7bcf814e-1a99-4942-b2b9-69607f3627e4:43
<id>	43
confidence	10
cpe	cpe:/a:mysql:mysql:5.5.5-10.11.8-mariadb-cll-lve-log:
method	probed
port	3306
product	MySQL
protocol	tcp
service_name	mysql
state	open

Figure 94. Neo4j results case 2 Graph, with visible service version in CPE element



Node properties

Service

<elementId>	4:7bcf814e-1a99-4942-b2b9-69607f3627e4:52
<id>	52
confidence	10
cpe	cpe:/o:linux:linux_kernel;cpe:/a:openbsd:openssh:6.6.1p1;
method	probed
port	22
product	OpenSSH
protocol	tcp
service_name	ssh
state	open

Figure 95. Neo4j results case 3 Graph, with visible service version in CPE element

6. Ending Summary

This chapter provides a brief summary of the key conclusions drawn from the research conducted for this thesis. In order to procure a complete perspective, it highlights the insights gained through this work along with the challenges faced and limitations encountered along the way. Finally, it suggests future research subjects based on the limitations of the provided methodology and tool.

6.1. Conclusions

The rapid evolution and expansion of modern infrastructure have significantly increased the exposure of networks to potential cyber threats. As discussed in this thesis, this growth in network connectivity although enhances functionality and access, broadens the attack surface, making the task of cybersecurity complex. Identifying, understanding, and securing these vulnerabilities have become critical, while even a single unprotected network asset can serve as a gateway for attackers.

This thesis examined how cyber attackers exploit weaknesses in domains, IP addresses, and network services, using these elements as the initial stages in their attack kill chain. In today’s threat landscape, attackers employ a broad range of strategies. The relevance of frameworks such as MITRE "ATT&CK", MITRE "D3FEND," Lockheed Martin’s “Cyber kill Chain” and others, along with methodologies and technologies like threat hunting and machine learning, underscores the necessity for systematic and automated strategies to counteract these evolving threats.

The earnings of this thesis include the development of a tool that focuses on **evaluating the exposed network surface** of infrastructures, which often serves as the first point of vulnerability in cyber-attacks. On the contrary to many **pre-existing tools** whose functionality is limited in the analysis of one or two concepts, this one systematically and **automatically analyzes** the whole route’s assets, starting **from a single domain name** and reaching **to the final running services,** applying a structured method for identifying potential exposure.

Another point is that the **automation** of the network **reconnaissance** procedures in this tool, by **orchestrating the operation of various tools** and using effective techniques, in comparison to other similar tools, was the first priority during its implementation and led in creating a tool that starts from the identification of domains and subdomains, followed by the analysis of corresponding global IP addresses and concluding with the identification of open ports and running services, **without constantly need to interact with the user**, as the tool itself decides its next operations by evaluating the results of each stage. This thorough analysis ensures that the tool captures an almost complete view of an organization’s exposed network assets.

In addition, a critical feature of this tool is its ability to automatically visualize the results and **integrate them into a graph database** (Neo4j), providing a clear representation of the revealed surface, while most of other tools are limited only in text formatted results.

Finally, on the contrary to most of the tools that existed so far, this one, correlates digestibly the identified services and open ports with Open-Source Cyber Threat Intelligence (OSCTI), and more specifically connects **CPEs to each found running service**, enhancing the capability to detect potential threats and allowing organizations to address known vulnerabilities and misconfigurations proactively.

6.2. Limitations and Future Work

As shortly mentioned, some techniques that are often used by organizations in order to cover their real online presence and make them resistant in attacks, can cause the tool not to run efficiently and lead to insufficient results. Assets like proxy servers, technologies like IPs’ WRAP or even others that are not intended in securing, like Networks designed for content delivery (CDNs), are some of them that prevents similar tools to unfold their operations.

Moreover, some advanced techniques (used by very sophisticated attackers) cannot be applied automatically or auto-orchestrated in a tool like this, because their results cannot be auto-reviewed by a tool but always need the judgment of an individual.

Another limitation in the usage of our tool, is the requirement for long time to run, because these procedures are generally time consuming. The point is that although times depend on hardware resources, faster 'moves' in networks are not preferred by malicious users, because could be detected and considered malicious behavior, by Intrusion Detection Systems. That was a limitation also for us, that we faced during the validation process.

But, despite the last limitation, the ever-evolving development of technology and computational power, in the future could let us integrate more advanced techniques in such tools. Many limitations that now are set by the IT assets, could over-passed in the future by using stronger resources. As already mentioned above, detecting services like CDNs or IPs' WRAP or proxy servers, now is difficult or even impossible in an automated way. In the future, it could be a matter for future research and if any automated solution exists, it could integrate with the current tool.

Closing, regarding the OSCTI correlations in the results of the above tool (CPEs correlations), in future work they could be expanded by creating connections also with other types of OSCTI, such as the Common Vulnerabilities and Exposures system (CVEs) which would connect the results to publicly known information security vulnerabilities as well as with the Common Attack Pattern Enumeration and Classification system (CAPEC) which would correlate the findings to known security attack patterns and give the network or systems operators the chance to take faster more targeted countermeasures to protect the infrastructures.

Bibliography - References

[1] L. Martin, *Cyber Kill Chain,* 2024.

[2] E. Hemberg, M. J. Turner, N. Rutar and U.-M. O'Reilly, "Enhancements to Threat, Vulnerability, and Mitigation Knowledge For Cyber Analytics, Hunting, and Simulations," *Digital threats,* August 2023.

[3] EPIC, *EPIC - Equifax Data Breach,* Electronic Privacy Information Center, 2021.

[4] C. Neagu, *Russian Retailer DNS Confirms Data Breach,* 2022.

[5] I. Arghire, *Cybercrime Ransomware Gang Leaks Data Allegedly Stolen From Greek Gas Supplier,* 2022.

[6] E. L. T. A. , *Press Release 21.03.2022 Cyberattack against the Hellenic Post,* 2022.

[7] R. Denuwan, *Marriott international data breach,* 2023.

[8] M. I. T. R. E. , *D3FEND Matrix* | *MITRE D3FEND™.*

[9] MITRE, *MITRE ATT&CK™.*

[10] N. I. S. T. , *NVD - CPE.*

[11] Neo4j, *Neo4j Graph Platform – The Leader in Graph Databases,* 2017.

[12] M. Stone, C. Irrechukwu, H. Perper, D. Wynne and L. Kauffman, "IT asset management: financial services," *Nist Special Publications,* September 2018.

[13] I. B. M. , *Categorizing an asset according to its data in InfoSphere Information Governance Catalog,* 2024.

[14] C. I. S. C. O. , *Cisco Security Reference Architecture,* 2022.

[15] C. I. S. C. O. , *Trustworthy Solutions,* 2024.

[16] C. I. S. C. O. , *Cybersecurity Framework Guidance.*

[17] E. N. I. S. A. , *ENISA Threat Landscape 2023,* 2023.

[18] E. N. I. S. A. , *Technical Guideline on Threats and Assets,* 2015.

[19] N. I. of Standards and Technology, "The NIST Cybersecurity Framework (CSF) 2.0," *The NIST Cybersecurity Framework (CSF) 2.0,* vol. 2.0, February 2024.

[20] T. Casey, "Threat agent library helps identify information security risks," September 2007.

[21] C. Martinie, C. Grigoriadis, E.-M. Kalogeraki and P. Kotzanikolaou, "Modelling Human Tasks to Enhance Threat Identification in Critical Maritime Systems," *PCI '21: Proceedings of the 25th Pan-Hellenic Conference on Informatics,* November 2021.

[22] I. Stellios, P. Kotzanikolaou, M. Psarakis, C. Alcaraz and J. Lopez, "A Survey of IoT-Enabled Cyberattacks: Assessing Attack Paths to Critical Infrastructures and Services," *IEEE Communications Surveys & Tutorials,* vol. 20, pp. 3453-3495, 2018.

[23] N. M. A. P. , *Nmap,* 2024.

[24] superhedgy, "GitHub - superhedgy/AttackSurfaceMapper: AttackSurfaceMapper is a tool that aims to automate the reconnaissance process.," *GitHub,* 2019.

[25] *OWASP Amass,* 2023.

[26] O. Osliak, A. Saracino, F. Martinelli and T. Dimitrakos, "Towards Collaborative Cyber Threat Intelligence for Security Management," *Proceedings of the 7th International Conference on Information Systems Security and Privacy,* 2021.

[27] P. Gao, X. Liu, E. Choi, B. Soman, C. Mishra, K. Farris and D. Song, *A System for Automated Open-Source Threat Intelligence Gathering and Management,* 2021.

[28] F. Marchiori, M. Conti and N. V. Verde, "STIXnet: A Novel and Modular Solution for Extracting All STIX Objects in CTI Reports," *arXiv (Cornell University),* August 2023.

[29] E. Hemberg, J. Kelly, M. Shlapentokh-Rothman, B. Reinstadler, K. Xu, N. Rutar and U.-M. O'Reilly, *Linking Threat Tactics, Techniques, and Patterns with Defensive Weaknesses, Vulnerabilities and Affected Platform Configurations for Cyber Hunting,* 2021.

[30] Y. Jia, Q. Yulu, S. Huaijun, R. Jiang and A. Li, "A Practical Approach to Constructing a Knowledge Graph for Cybersecurity," *Engineering,* vol. 4, pp. 53-60, April 2018.

[31] D. Velasco and G. Rodriguez, "Ontologies for Network Security and Future Challenges," *arXiv (Cornell University),* January 2017.

[32] C. Grigoriadis, A. M. Berzovitis, I. Stellios and P. Kotzanikolaou, "A Cybersecurity Ontology to Support Risk Information Gathering in Cyber-Physical Systems," *Springer eBooks,* pp. 23-39, January 2022.

[33] *python.org.*

[34] *The Full DTD* | *Nmap Network Scanning — nmap.org.*

[35] *Bash - GNU Project - Free Software Foundation,* 2009.

[36] K. Vesterinen, *validators: Python Data Validation for Humans™.*

[37] P. , *pandas,* 2020.

APPENDIX A.

File “**recontool2.sh**”

```
File#!/bin/bash

# Author: George Tasios
# FileName: recontool2.sh
# The very first script that handles the flags (arguments)

source ./func.sh
#function to display title
printTitle
# Function to display script usage
act=FALSE
dom=""
verbose_mode=FALSE
wlpath="nopath"
br=FALSE

while getopts ":hvad:bw:" option; do
        case $option in
        d)
          dom="$OPTARG" #insert domain for investigation e.g. -d example.com
          ;;
        a)
          act=TRUE ;;
        v)
          verbose_mode=TRUE ;;
        h)
          usage
          exit 1
          ;;
        b)
          br=TRUE ;;
        w)
          wlpath="$OPTARG"
          if [ ! -f $OPTARG ]; then
            echo "File or path given for wordlist does not exist.."
            exit 1
          fi
          ;;
        \?)
          echo "invalid option: -$OPTARG" >&2
          exit 1
          ;;
        :)
          echo "Option -$OPTARG requires an argument." >&2
          exit 1
          ;;
        *)
          echo -e "\n!!!!!!!!!! Invalid Usage !!!!!!!!!!\n"
          usage
          exit 1
          ;;
    esac
done

./runrecon.sh $dom $act $verbose_mode $br $wlpath
```

File “**runrecon.sh**”

```
#!/bin/bash

# Author: George Tasios
# FileName: runrecon.sh

#insert file containing functions to call
source ./func.sh

# Deleting previous URL file if exists
rm -f URLTEMP.txt

# positional arguments come from recontool2.sh in the following order:
# ./runrecon.sh $dom $act $verbose_mode $br $wlpath

dom="$1"

if [ $2 == TRUE ]; then
  act=" -active"
fi

if [ $3 == TRUE ]; then
  verbose_mode=" -v"
fi

if [ $4 == TRUE ]; then
  br=" -brute"
fi

if [ ! $5 == "nopath" ]; then
  wlpath=" -w $5"
fi

#==============================================================================================
======================
#Checks internet connection. If it is down, stops the script
checkInternetConnection

#Function that creates a timestamp. To be used in running time calculation
timer_start

#==============================================================================================
======================

echo -e "Running checks for the inserted DOMAIN..."
python3 check_url_redirect.py $dom

#check if the python script exits with 1 which mean that terminated the programm execution
checkPythonExit

#Stores the first line of the URLTEMP.txt file in a variable. It is the final URL
FINALURL=$(head -n 1 URLTEMP.txt)

# Checks if the URL has been written in the URLTEMP.txt file and if not terminates the
programm. This function validates the successful insertion of URL.
# It was necessary as an additional step of validation
urltempcheck


# Opens the final URL and extracts the registered_domain. Then APPENDS it to the URLTEMP.txt
file
#removing any possible subdomains or paths in the URL or http, www etc, keeping only the TLD
and the 2nd level Domain
sudo python3 extract_tld.py $FINALURL


#Stores the second line of the URLTEMP.txt file in a variable. It is the final DOMAIN that
will be passed as argument to 'AMASS' tool
```

```
FINALDOMAIN=$(head -n 2 URLTEMP.txt | tail -n 1)

echo -e "\nFINAL URL (FINALURL variable) : $FINALURL"
echo -e "\nFINAL DOMAIN NAME (FINALDOMAIN variable, passing to amass tool) for enumeration:
'$FINALDOMAIN'\n"
echo -e "\nIt will take a \"little\" time. Be patient........\n"

#creating folder with certain permissions and only if it donsn't exist
mkdir -p -m 777 report


# Get the current date and time in the format YYYY-MM-DD_HH-MM
# Current_datetime=$(date +"%Y-%m-%d_%H-%M-%S")
current_datetime=$(date +"%Y-%m-%d_%H-%M")

# Creation of subfolder for the current domain
mkdir -p -m 777 report/${FINALDOMAIN}

#combining date-time-domainName
filename="${FINALDOMAIN}_${current_datetime}_report.txt" #Initial enumeration report from
amass tool
fileIpsOut="${FINALDOMAIN}_IPs_${current_datetime}.txt" #Ips found in first enumeration
(contains multiple times each)
fileSubdomainOut="${FINALDOMAIN}_subdomains_${current_datetime}.txt" #Subdomains found from
amass enumeration
fileIpsNoDuplicates="${FINALDOMAIN}_UNIQUE_IPs_${current_datetime}.txt" #Distinct IPs by
deleting local ips or duplicates
fileOpenPortsCSV="${FINALDOMAIN}_OpenPortsCSV_${current_datetime}.csv" # Found opened ports
after 'socket' scan
fileNmapXML="${FINALDOMAIN}_NmapXML_${current_datetime}.xml" # Store in XML form results
from nmap services and version scan on opened ports
fileFinalCSVResults="${FINALDOMAIN}_finalCSVresults_${current_datetime}.csv" # LAST REPORT:
Store in CSV file the results from XML that nmap gives for each IP ('checkservices.sh'
script)


#files where discovered sub-domains, IPs, nmap results  etc. will be stored
touch report/${FINALDOMAIN}/$filename
filenamePath="report/${FINALDOMAIN}/$filename"
touch report/${FINALDOMAIN}/${fileIpsOut}
fileipsPath="report/${FINALDOMAIN}/${fileIpsOut}"
touch report/${FINALDOMAIN}/${fileSubdomainOut}
filesubdomainPath="report/${FINALDOMAIN}/${fileSubdomainOut}"
touch report/${FINALDOMAIN}/${fileIpsNoDuplicates}
fileduplicatesPath="report/${FINALDOMAIN}/${fileIpsNoDuplicates}"
touch report/${FINALDOMAIN}/${fileNmapResults}
fileopenportscsvpath="report/${FINALDOMAIN}/${fileOpenPortsCSV}"
touch report/${FINALDOMAIN}/${fileNmapXML}
filenmapxmlpath="report/${FINALDOMAIN}/${fileNmapXML}"
touch report/${FINALDOMAIN}/${fileFinalCSVResults}
filefinalcsvresultspath="report/${FINALDOMAIN}/${fileFinalCSVResults}"


# Constructing the arguments of amass command and pass the results in a named txt file
# after the following command executed, it goes to the backround as a running process.
# Its process PID is stored in the 'PID2' variable
amass enum$act$verbose_mode -ip$br$wlpath -d $FINALDOMAIN >
"report/${FINALDOMAIN}/$filename" &
PID2=$!

echo -e "${BLUE}Enumeration will may run more than 5 min, depending on your ${GREEN}machine
resources ${BLUE}and the ${GREEN}revealed data.\n${GREEN}You can simply end it by pressing
'q'${NOCOLOR}\n"

echo -e "Running enumeration techniques, please wait.....\n"

# Runs the spinner animation and put in in the backround as a running process
# Then it stores the running process' PID to the 'PID3' variable
```

```
spinner
PID3=$!

# While the previous processes are running, the following while-loop checks periodically
(every 3 secs) if the above amass tool is still running (PID2 process active)
# or if the 'q' button pressed. If 'q' is pressed or the process is not running any more, it
kills the afforementioned PID2 and PID3 processes,
# prints the elapse time and continues
while true; do
  # Asks the user if wants to stop
  # Read user input without a timeout
  read -t 3 -n 1 input
  # Check if the user pressed 'q'
  PIDCHECK7=$(pgrep "amass")
  if [[ $PIDCHECK7 == "" ]]; then
    echo -e "\nEnumeration process almost completed..."
    sudo kill $PID2 2>/dev/null
    sudo kill $PID3 2>/dev/null
    echo -e "Subdomain enumeration completed...."
    time_elapsed "Time elapsed."
    break
  fi
  if [ "$input" == "q" ]; then
    echo -e "\nEnumeration process stopped manually..."
    sudo kill $PID2 2>/dev/null
    sudo kill $PID3 2>/dev/null
    echo -e "Subdomain enumeration completed...."
    time_elapsed "Time elapsed.."
    break
  fi
done

echo -e "${GREEN}" # this line colorizes the output of convIP.py file
# TEMP modification ---> change the file convIP2 to convIP
# Runs a python script that separates urls from ips and deletes duplicate and non-global ips
python3 convIP.py $filenamePath $fileipsPath $filesubdomainPath $fileduplicatesPath

#check if the python script exits with 1 which mean that terminated the program execution
checkPythonExit

echo -e "${ENDCOLOR}"


echo -e "${GREEN}Now found IPs, will be checked for open ports\nJust be sure that all IPs
are not behind ${RED}CDN, Proxies ${GREEN}or similar services,\notherwise results will be
inaccurate and it may take excessively long time to complete ${ENDCOLOR}\n"

# At this point the portsenum.py script will be called to find the open ports of each unique
IP
#Takes as 1st argument the unique IPs file ('fileduplicatespath') and writes in the pre-
created csv file (2nd argument), one ip with its open ports in each line
python3 portsenum.py $fileduplicatesPath $fileopenportscsvpath

# When the above python file-script ends, will continue with next script which is
'checkservices.sh' where nmap tool comes..... ;)


# At this point 'checkservices.sh' script will scan found open ports (1st argument csv file)
to identify running services types and their versions
# Will use nmap with '-oX' option to output the foundings in an XML file, which will be the
2nd argument (xml precreated file) to the following script
# -->
# As 3rd argument takes the 'fileFinalCSVResults' which is the final CSV file. It will be
used as (2nd) argument to the 'XMLprocess.py' script which will be run
# several times in the following (.sh) script. For each IP that will be found to have open
ports, it will append the XML results of nmap in the CSV final file
./checkservices.sh $fileopenportscsvpath $filenmapxmlpath $filefinalcsvresultspath
```

```
# Boolean variable takes the value '0' if we have results to modify and pass to
Neo4j database system
haveresults=0

# Checks if any result found. -s flag returns true if file size is greater than
zero. If we have results, the following command runs in order to create files to
be passed into Neo4j database
# Then the 'haveresults' flag is set to 1 in order to allow run the Neo4j graph
creation command
if [[ -s "$filefinalcsvresultspath" ]]; then
    ./prepareNeo4jFiles.sh $FINALDOMAIN $filesubdomainPath $fileduplicatesPath
$filenamePath $filefinalcsvresultspath $fileopenportscsvpath
    haveresults=1
fi

# Function 'check_neo4j_running' exists in func.sh and checks if Neo4j database
system is running
# adds the output of the previous command to a the returnvalue variable,
depending if Neo4j is running or not
returnvalue="$(check_neo4j_running)"

if [[ $returnvalue -eq 1 && $haveresults -eq 1 ]]; then
    python3 pyToNeo.py $FINALDOMAIN
    echo -e "${GREEN}\nNeo4j graph is available at
http://localhost:7474${ENDCOLOR}"
fi

if [[ $haveresults -eq 0 ]]; then
    echo "No results found. No graph for data visualization will be created"
fi

if  [[ $returnvalue -eq 0 && $haveresults -eq 1 ]];then
    echo "Neo4j is not running. Start neo4j service and then run the command
\"python3 pyToNeo.py $FINALDOMAIN \""
    echo "After, the Graph will be available at http://localhost:7474"
fi



# Deletes the URLTEMP.txt file if has been created
rm -f URLTEMP.txt
time_elapsed "Reconnaissance of exposure completed in"

#echo -e "\nFinal results stored in '/report/${FINALDOMAIN}'\n"
echo -e "\n${BLUE}Final results stored in path:
${GREEN}'$filefinalcsvresultspath'${ENDCOLOR}\n"

# Footer printing function. Imported function from func.sh file
printFooter
```

File "**func.sh**"

```
#!/bin/bash

# Author: George Tasios
# FileName: func.sh
RED='\e[31m'
REDI='\e[3;31m'
GREEN='\e[92m'
BLUE='\e[94m'
YELLOW='\e[33m'
```

```
LIGHTYELLOWBACK='\e[103m'
LIGHTRED='\e[91m'
ENDCOLOR='\e[0m'


# Function to display script title and information
printTitle() {
 echo -e
"\n${GREEN}===============================================================${ENDCOLOR}"
 echo -e "${RED}====== ${LIGHTYELLOWBACK}${BLUE}~Exposed Surface Reconnaissance~ Tool
Version 1.0${ENDCOLOR}${RED} ======${ENDCOLOR}"
 echo -e
"${GREEN}===============================================================${ENDCOLOR}"
 echo
 echo -e "${GREEN}-${RED}Author: ${BLUE}G.Tasios${ENDCOLOR}${GREEN}${ENDCOLOR}"
 echo
 echo
 echo -e "${GREEN}(${RED}WARNING! ${GREEN}Don't forget to use a VPN
connection...)${ENDCOLOR}"

}

# Function to display script footer when it is stopped or finishes its actions
printFooter() {
 echo -e "\n\n${RED}Deleting temporary files...  (e.g. url file, etc...)${ENDCOLOR}"
 echo
 echo
 echo -e "${BLUE}======================================================${ENDCOLOR}"
 echo -e "${BLUE}==================${ENDCOLOR}${RED} D O N E . . . !
${BLUE}===================${ENDCOLOR}"
 echo -e "${BLUE}======================================================${ENDCOLOR}"

}


# Function to display script usage options
usage() {
 echo
 echo "Usage: $0 [OPTIONS]"
 echo "Options:"
 echo " -h,      Display this help message"
 echo " -d,      insert the domain for investigation (required)"
 echo " -a,      Active mode enabled. Performs deep DNS enumeration techniques like zone
transfers, port scanning of SSL/TLS, certificates grabbing etc. (*CAUTION: More than usually
time needed)"
 echo " -v,      Enable verbose mode"
 echo " -b,      Brute force SubDomain Enumeration with default dictionary. Use along with -
w flag to use your own wordlist"
 echo " -w,      Path to use wordlist for sub-domains brute forcing."
 echo
 echo
 echo
 echo

 echo -e "${BLUE}======================================================${ENDCOLOR}"
 echo -e "${BLUE}==================${ENDCOLOR}${RED} D O N E . . . !
${BLUE}===================${ENDCOLOR}"
 echo -e "${BLUE}======================================================${ENDCOLOR}"

}

#Checks internet connection. If it is down, stops the program
checkInternetConnection() {
  echo -e "\n\n${BLUE}Checking internet connection...${ENDCOLOR}\n"
  sleep 3
  wget -q --spider http://google.com
  if [ $? -eq 0 ]; then
    echo -e "${GREEN}Internet connection seems OK...${ENDCOLOR}"
  else
    echo -e "${RED}Internet connection seems down...\nCheck your connection and try again
later...${ENDCOLOR}\n"
    exit 1
  fi
}



# Function that creates a timestamp and inputs it in a global variable
# will be used in next steps for time counting reasons
timer_start() {
  declare -g start_time=$(date +%s)
  start_time_formated=$(date +"%H:%M.%S\"(%d-%m-%Y)")
  echo -e "\n---->${BLUE}Running started at: ${GREEN}$start_time_formated${ENDCOLOR}\n"
}


#catches the exit status of the previously executed commands. actually gets the
#exit status code of 'check_url_redirect.py' file
checkPythonExit() {
if [ $? -eq 1 ]; then
    echo "Exiting Reconnaissance Tool....."
    printFooter
    exit 1
fi
}


# Check if the FINALURL variable starts "http" in order to stop the program if URLTEMP file
hasn't been read succesfully
urltempcheck() {
if [[ ! $FINALURL == http* ]]; then
    echo -e "\nFINAL URL seems that does not appear in URLTEMP.txt file.\nPlease check
again. (Variable does not start with http)\nExiting script....."
    exit 1
fi
}


#Used as time checkpoint. We can insert an argument $1 in order to appear when this function
prints the elapsed time
time_elapsed() {
  current_time=$(date +%s)
  elapsed_time=$((current_time - start_time))
  days=$((elapsed_time / 86400))
  hours=$(( (elapsed_time % 86400) / 3600 ))
  minutes=$(( (elapsed_time % 3600) / 60 ))
  remaining_seconds=$((elapsed_time % 60))
  DAYS=""
  HOURS=""
  MINUTES=""
  if [ "$days" -gt 0 ]; then
    DAYS="${GREEN}$days ${YELLOW}Days,"
  fi
  if [ "$hours" -gt 0 ]; then
    HOURS="${GREEN}$hours ${YELLOW}Hours,"
  fi
  if [ "$minutes" -gt 0 ]; then
    MINUTES="${GREEN}$minutes ${YELLOW}Minutes,"
  fi

  echo -e "\n---->${BLUE}$1: $DAYS $HOURS $MINUTES ${GREEN}$remaining_seconds
${YELLOW}Seconds\"....${ENDCOLOR}\n"

}

# Spinner. Animation that indicates to the user the programm is still running.
# Because of the extend times that it takes to run, this spinner is necessary to exist, in
order to inform the user that the system is not "halted"
spinner() {
    i=1
    sp="□□□□"
    echo -n ' '
    echo "Running......"
    while [ -d /proc/$1 ]
    do
        printf "\b${sp:i++%${#sp}:1}\b"
    done &
}



#checks if nmap is installed and if file with IPs exists. Called from 'checkservices.sh'
nmapCheck() {
  #firstly we will check if the nmap tool is installed
  command -v nmap >/dev/null 2>&1 || { echo >&2 "Nmap is required and it seems not to be
installed. Please install it."; exit 1; }

  #check if the IPs file exists
  if [ ! -f "$input_file" ]; then
    echo "Nmap Error: Input file not found: $input_file"
    exit 1
  fi

}

amassProcessCheck() {
  PIDCHECK2=$(pgrep "amass")
  if [[ ! $PIDCHECK2 == "" ]]; then
    echo -e "\nKilling enumeration process..."
    sudo kill $PID2
  fi
}

# This Function checks if a process is still running
isProcessRunning() {
    process_id=$1
    if [ ps -p $process_id ] > /dev/null; then
        return 0  # Process is running
    else
        return 1  # Process is not running
    fi
}

# Checks if Neo4j is running. If it is, prints as output the '1'
check_neo4j_running() {
    if neo4j status | grep "is running at" --quiet; then
        val=1
    else
        val=0
    fi
echo $val
}
```

File “**check_url_redirect.py**”

```
# Author: George Tasios
# FileName: check_url_redirect.py

import sys
import requests
import validators
import socket
import re
import subprocess
import os


#The following code checks if the URL is in the right Format and if it redirects to another
url
#then shows the final URL where it ends

#FUNCTION checks URL format
def check_hostname(my_domain):
    try:
        my_domain = my_domain.lower()
        valid = validators.domain(my_domain)
        if valid:
            print('\nDomain \"' + my_domain + '\" is a valid format.....')
        else:
            print('\nDomain didn\'t inserted or  is NOT a valid domain format')
            exit(1)
    except Exception as error:
        print('\nAn error occured while checking HOST (domain). Please try again')
        exit(1)

#FUNCTION to check if URL resolve to an IP
def domain_resolv(domain_to_resolv):
    try:
        r = socket.gethostbyname(domain_to_resolv)
        print("\nDomain \'" + domain_to_resolv + "\' resolved successfully to IP: \'" + r +
"\'\n")
    except socket.error:
        print("\nDomain didn't resolve... Script is stopped!!!\n")
        exit(1)

#============================================================================================
====

#checks if argument is in valid host format. otherwise it exits() the script
check_hostname(sys.argv[1])

#composes the url from domain name (host)
my_url=("http://"+sys.argv[1])
my_urls=("https://"+sys.argv[1])

my_domain=(sys.argv[1])

print("Printing my domain: " + my_domain)

domain_resolv(my_domain)

#the following runs only if a valid HOST (domain) is given
#returns successfully a status code if the domain resolves to a service or webpage
try:
    resolved=False
    #Check HTTP requests
    try:
        r = requests.get(my_url, allow_redirects=True)
        print("\nHTTP Status Code:[" + str(r.status_code) +"]...")
        print("\nHTTP Redirections Codes History: " + str(r.history))
        resolved=True
    except:
        pass
    if(resolved == False):
        try:
            rs = requests.get(my_urls, allow_redirects=True)
            #The same for the https requests
            print("\nHTTPS Status Code:[" + str(rs.status_code) +"]...")
            print("\nHTTPS Redirections Codes History: " + str(rs.history))
        except:
            print("\nAn ERROR occured in 'http' handling or URL finaly could not resolve.
Please try again\n")
            exit(1)
    # checks if final url is http or https and prints it. Then it is passed to the bash
script 'pass_url.sh'
    if(my_url != r.url):
      print("\nFinal URL is: \'"+ r.url +"\'")
    elif(my_urls != rs.url):
      print("\nFinal URL is: \'"+ rs.url +"\'")
    #########
    if r.url:
      subprocess.call(['./pass_url.sh', r.url])
    elif rs.url:
      subprocess.call(['./pass_url.sh', rs.url])
    #print(r.headers)
except Exception as error:
    print("\nAn ERROR occured in 'http' handling or URL finaly could not resolve. Please try
again\n")
    exit(1)
```

File “**pass_url.sh**”

```
#!/bin/bash

# Author: George Tasios
# FileName: pass_url.sh

# Creation of an environmental variable which contains the passed domain (whether
http or https), from python file 'check_url_redirect.py'
echo -e "\nCreating necessary ENVIRONMENT Variables...."
export ENV_MY_URL=$1
echo
echo
echo -e  "\nCreating TEMP files..."
echo
echo
echo -e "('URLTEMP.txt' file.... )"

# Writes the 'url' which is stored in an environmental variable, into a txt file
printenv ENV_MY_URL > URLTEMP.txt

# The above txt file after it is used it will be deleted automatically
```

File “**extract_tld.py**”

```
# Author: George Tasios
# FileName: extract_tld.py.sh

##called from 'runrecon.sh' file##

import tldextract #use this library to extract tld and main domain
import sys

global finaldomain

dom = sys.argv[1]

#The following line extracts the domain and the TLD, even if TLD
#is more than 2 words and keep the url clear from subdomains or paths
t = tldextract.extract(dom)
```

```
#Sets the variable 'finaldomain' with the domain+its tld, even if it is more than
one word.
#TLDextract library uses the Mozilla public TLDs list.
finaldomain = t.registered_domain

file = 'URLTEMP.txt'

#Appending the argument in the URLTEMP.txt file
with open(file, 'a') as f:
    f.write(finaldomain)
```

File “**convIP.py**”

```
# Author: George Tasios
# FileName: convIP.py

import re
import ipaddress
import sys


# This is how it is called from the runrecon.sh file:
# python3 convIP.py $filenamePath $fileipsPath $filesubdomainPath
$fileduplicatesPath
# 1: first report file 2: extracted ips 3: extracted subdomains file 4: remove
the duplicate IPs


# checks if any ip found or if in the results file contained the appropriate
message generated by the amass tool which starts with "No names were
discovered...."
def check_if_any_ip_found(infile):
    try:
        with open(infile, 'r') as file:
            first_line = file.readline().strip()

        if not first_line or first_line == "No names were discovered":
            print("\nNo names were discovered. No results found\n")
            exit(1)
    except Exception as e:
        print("\nError opening results file..... please try again\n")
        exit(1)


# Removes commas from the report file in order to handle more easy the content
# The report file contains a subdomain and its ips, in each line, all separated
with commas
def remove_commas(file):
    with open(file, 'r') as f:
        content = f.read()
    content_with_no_commas = content.replace(',', ' ')
    with open(file, 'w') as f:
        f.write(content_with_no_commas)


# This function handles the IPs (ipv4 or ipv6) as words separated by spaces.
# It takes from the 2nd to the last 'word' of each line and appends each one to
the 'fileout' file, as a new line
```

```
def extractWords(filein, fileout):
    with open(filein, 'r') as infile, open(fileout, 'w') as outfile:
        for line in infile:
            words = line.split()
            # Extract all words except the first word
            filtered_words = words[1:]
            # Write the extracted words to the output file
            for word in filtered_words:
                outfile.write(f"{word}\n")


# This function extracts all urls (subdomains) in a file. It handles each line as
words separated by spaces.
# It takes the 1st 'word' of each line and appends it to the 'out_file', as a new
line
def extract_urls(in_file, out_file):
    with open(in_file, 'r') as infile, open(out_file, 'w') as outfile:
        for line in infile:
            # Extract the url (subdomain) each line
            url = line.split(' ', 1)[0]
            # Write the URL to the output file
            outfile.write(url + '\n')


# Handles the duplicates ips and also removes local ips (ipv4 or ipv6).
# This is committed in order to decrease the amount of requests to the next
steps.
# It uses the data structure 'set' which is actually a 'list' with unique
elements.
# If an already existed element is added, it is not added a 2nd time but rejected
def remove_duplicate_and_private_ips(in_file, out_file):
    unique_ips = set()
    with open(in_file, 'r') as file:
        for line in file:
          ip = line.strip()
          if ip:
              unique_ips.add(ip)
    with open(out_file, 'w') as file:
        for ip in unique_ips:
            myip = ipaddress.ip_address(ip)
            if myip.is_global:
              file.write(ip + '\n')


# Counts how many IPS v4 and v6 found and prints the results. If something other
than an IP address found,
# it prints an appropriate message and then stops the programm
def count_ips(infile):
    ipv4_count = 0
    ipv6_count = 0

    with open(infile, 'r') as file:
        for line in file:
            line = line.strip()
            if line:
                try:
                    if ipaddress.IPv4Address(line):
                        ipv4_count += 1
                except ValueError:
                    try:
                        if ipaddress.IPv6Address(line):
                            ipv6_count += 1
                    except ValueError:
                        print("some lines contain nor an IPv4 neither an IPv6
address. Please check again")

    print("Found " + str(ipv4_count) + " ipv4 IPs and " + str(ipv6_count) + "
ipv6 IPs")


try:
    # checks if any result occured and then run the rest scripts. If not stops
the scripts
    check_if_any_ip_found(sys.argv[1])

    # Remove commas from the  first report file for better content handling
    remove_commas(sys.argv[1])

    # Extract IPv4 and IPv6 and write them to the output file
    extractWords(sys.argv[1], sys.argv[2])

    # Extract URLs from each line and write them to the output file
    extract_urls(sys.argv[1], sys.argv[3])

    # Remove duplicate ips from file iptemp.txt and writes them to
"ipsnoduplicates.txt"
    # Also removes non global IPs
    remove_duplicate_and_private_ips(sys.argv[2], sys.argv[4])

    #count final and unique ips found
    count_ips(sys.argv[4])
except Exception as e:
    print("An unexpected error occured: \n", e, "\nPlease refer to the creator of
this tool\n" )
    exit(1)


print("\nFound results stored in '/report' folder\n")
```

File “**portsenum.py**” (*ports lists for 5000 and 10000 ports are shorted intentionally for obvious reasons of space saving*)

```
# Author: George Tasios
# Filename: portsenum.py

# This script called from 'runrecon.sh' script line '178'
# Inserted in this file as first argument the file with clear and unique IPs (IPs file
('fileduplicatespath')
# Then it will check if any port of them is opened. According to the choice of the user
(will be asked and prompt to choose...)
# The results written in the pre-created csv file (2nd argument), one ip with its open ports
in each line


import sys
import os
import socket
import random
from datetime import datetime
import time
import threading
import select
import csv

# check internet connection
try:
    socket.setdefaulttimeout(2)
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(("8.8.8.8", 53))  # Google DNS IPv4
    print("\nInternet connection seems to be OK...\n")
except:
    print("\n" + 60 * "*")
    print("*Possibly Internet connection iS Down! Get online and retry*")
    print(60 * "*" + "\n")
    exit(1)




#list of 100, 500, 1000 & 10000 common and popular ports to scan
commonports100 =
[80,23,443,21,22,25,3389,110,445,139,143,53,135,3306,8080,1723,111,995,993,5900,1025,587,888
8,199,1720,465,548,113,81,6001,1993,514,5060,179,1026,2000,8443,8000,32768,554,26,1433,49152
,2001,515,8008,49154,1027,5666,646,5000,5631,631,49153,8081,2049,88,79,5800,106,2121,1110,49
155,6000,513,990,5357,427,49156,543,544,5101,144,7,389,8009,3128,444,9999,5009,7070,5190,300
0,5432,1900,3986,13,1029,9,5051,6646,49157,1028,873,1755,2717,4899,9100,119,37]
commonports500 =
[80,23,443,21,22,25,3389,110,445,139,143,53,135,3306,8080,1723,111,995,993,5900,1025,587,888
8,199,1720,465,548,113,81,6001,10000,514,5060,179,1026,2000,8443,8000,32768,554,26,1433,4915
2,2001,515,8008,49154,1027,5666,646,5000,5631,631,49153,8081,2049,88,79,5800,106,2121,1110,4
9155,6000,513,990,5357,427,49156,543,544,5101,144,7,389,8009,3128,444,9999,5009,7070,5190,30
00,5432,1900,3986,13,1029,9,5051,6646,49157,1028,873,1755,2717,4899,9100,119,37,1000,3001,50
01,82,10010,1030,9090,2107,1024,2103,6004,1801,5050,19,8031,1041,255,1049,1048,2967,1053,370
3,1056,1065,1064,1054,17,808,3689,1031,1044,1071,5901,100,9102,8010,2869,1039,5120,4001,9000
,2105,636,1038,2601,1,7000,1066,1069,625,311,280,254,4000,1993,1761,5003,2002,2005,1998,1032
,1050,6112,3690,1521,2161,6002,1080,2401,4045,902,7937,787,1058,2383,32771,1033,1040,1059,50
000,5555,10001,1494,593,2301,3,1,3268,7938,1234,1022,1074,8002,1036,1035,9001,1037,464,497,1
935,6666,2003,6543,1352,24,3269,1111,407,500,20,2006,3260,15000,1218,1034,4444,264,2004,33,1
042,42510,999,3052,1023,1068,222,7100,888,4827,1999,563,1717,2008,992,32770,32772,7001,8082,
2007,740,5550,2009,5801,1043,512,2701,7019,50001,1700,4662,2065,2010,42,9535,2602,3333,161,5
100,5002,2604,4002,6059,1047,8192,8193,2702,6789,9595,1051,9594,9593,16993,16992,5226,5225,3
2769,3283,1052,8194,1055,1062,9415,8701,8652,8651,8089,65389,65000,64680,64623,55600,55555,5
2869,35500,33354,23502,20828,1311,1060,4443,730,731,709,1067,13782,5902,366,9050,1002,85,550
0,5431,1864,1863,8085,51103,49999,45100,10243,49,3495,6667,90,475,27000,1503,6881,1500,8021,
340,78,5566,8088,2222,9071,8899,6005,9876,1501,5102,32774,32773,9101,5679,163,648,146,1666,9
01,83,9207,8001,8083,5004,3476,8084,5214,14238,12345,912,30,2605,2030,6,541,8007,3005,4,1248
,2500,880,306,4242,1097,9009,2525,1086,1088,8291,52822,6101,900,7200,2809,395,800,32775,1200
0,1083,211,987,705,20005,711,13783,6969,3071,5269,5222,1085,1046,5987,5989,5988,2190,11967,8
600,3766,7627,8087,30000,9010,7741,14000,3367,1099,1098,3031,2718,6580,15002,4129,6901,3827,
3580,2144,9900,8181,3801,1718,2811,9080,2135,1045,2399,3017,10002,1148,9002,8873,2875,9011,5
718,8086,3998,2607,11110,4126,5911,5910,9618,2381,1096,3300,3351,1073,8333,3784,5633,15660,6
123,3211,1078,3659,3551,2260,2160,2100,16001,3325,3323,1104,9968,9503,9502,9485,9290,9220,89
94,8649,8222,7911,7625,7106,65129,63331,6156,6129,60020,5962,5961,5960,5959,5925,5877,5825,5
810,58080,57294,50800]
commonports1000 = [80,23,443,21,22,{---- 5000 ports ------},7878,3304, 3307, 1259,1092]
commonports10000 = [1601,2878,5605,{---- 10000 ports ------},5602,5603,3284,1742]


portsToScan = []
open_ports = []
closed_ports = []
ipsno = 0



#Checking if argument inserted, otherwise it stops running script
if len(sys.argv) < 3:
    print("Missing or no arguments inserted")
    exit(1)
```

```
else:
    ipfile = sys.argv[1]    # Set the inserted file equal to variable 'ipfile'
    portsfile = sys.argv[2]
    print(f"\nYou have inserted {ipfile}\n")


# Calculates the length (number) of IPs contained in the UNIQUE IPS file (1st argument)
with open(ipfile, 'r') as ipf:
     ipsno = len(ipf.readlines())


#>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>
# Function with 'Banner grabbing' technique USING SOCKET to ensure the found ports are
opened
def grabbanner(ip,porta):
  socket.setdefaulttimeout(1)
  # A little bit of unorthodox way to recognize IPv4 and IPv6 addresses
  if '.' in ip:
      bangrab = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  elif ':' in ip:
      bangrab = socket.socket(socket.AF_INET6, socket.SOCK_STREAM)
  try:
    bangrab.connect((ip, porta))
    banner = bangrab.recv(128)
    print(f"*Something seems to be running behind port {porta} :")
    open_ports.append(porta)
    bangrab.close()
    print(f"====>Printing received banner : {banner}\n", end='\r')
  except:
    print(f"---> Port {porta}: Possibly closed <--", end='\r')
    closed_ports.append(porta)
    bangrab.close()

#>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>
# Following function scans the requested ports (according to the chosen 'portsToScan' file),
for the given IP.
# If a port found closed, it is added to a list. If the socket method finds it opened,
# the banner grabbing technique ('grabbanner' function) comes and checks each of our
choice's ports, randomly
def scanport(ipaddress):
  print(f"\nScanning IP: {ipaddress} ({ipsno} IPs left to scan)")
  for porta in portsToScan:
      if '.' in ipaddress:
          sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
      elif ':' in ipaddress:
          sock = socket.socket(socket.AF_INET6, socket.SOCK_STREAM)
      socket.setdefaulttimeout(1) #It should be '2' if internet connection is not fast
enough
      result = sock.connect_ex((ipaddress,porta))
      #print(f"Printing result: {result}")
      if sock.connect_ex((ipaddress, porta)):
        print(f"---> Port {porta}: Closed <----", end='\r')
        closed_ports.append(porta)
      else:
        grabbanner(ipaddress,porta)
  print(f"\nPort scanning finished", 10*">")
  print(f"\nFound: Opened ports: {len(open_ports)} | Closed ports: {len(closed_ports)}")
  sock.close()

#>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>
prompt_message = """!!!WARNING!!!\nYou can choose the port enumeration depth.
Press 1 to scan the 100 most popular ports (default choice)
Press 2 to scan the 500 most popular ports
Press 3 to scan the 1000 most popular used ports
Press 4 to scan the 10000 most popular used ports
Press 5 to scan all 65.535 ports for the given IP target (CAUTION: It will take excessively
much time)\n"""


endingtime = time.time() + 61

print(prompt_message)
choice = ""

# waiting to input something for xxx seconds. If not, will proceed with default port number
(100)
while True:
    remainingtime = int(endingtime - time.time())
    print("***If nothing pressed in ", remainingtime ," secs, will proceed with default
choice (100 most common ports) :")
    try:
        rlist, _, _ = select.select([sys.stdin], [], [], remainingtime)
    except select.error:
        print("\nTimed out. Proceeding automatically to choice -1- (100 ports)")
        choice = "1"
        sys.exit(0)

    if rlist:
        number_str = sys.stdin.readline().strip()
        try:
            number = int(number_str)
            if 1 <= number <= 4:
                print("You entered", number)
                choice = str(number)
                break
            elif number == 5:
                if choice == "5":
                    break
                else:
                    print("You have chosen to scan ALL ports of the target!!!")
                    print("Are you sure? It will take excessively much more time than
usual")
                    print("Enter again and press ENTER to commit your choice or review it in
the next", remainingtime, "remaining seconds...")
                    choice = str(number)
            else:
                print("Invalid input. Please enter a valid choice from 1 to 5")
        except ValueError:
            print("Invalid input. Please enter a valid choice from 1 to 5 ")
    else:
        print("\nNo input received. Proceeding automatically\n")
        choice = "1"
        break


if choice == "1":
    print("Continuiiiiing with choice: -", choice, "- scanning 100 ports for each IP")
    portsToScan = commonports100
elif choice == "2":
    print("Continuiiiiing with choice: -", choice, "- scanning 500 ports for each IP")
    portsToScan = commonports500
elif choice == "3":
    print("Continuiiiiing with choice: -", choice, "- scanning 1000 ports for each IP")
    portsToScan = commonports1000
elif choice == "4":
    print("Continuiiiiing with choice: -", choice, "- scanning 10000 ports for each IP")
    portsToScan = commonports10000
elif choice == "5":
    print("Continuiiiiing with choice: -", choice, "- scanning all 65.535. It will take much
time....")
    portsToScan = list(range(1, 65535))
else:
    print("Continuiiiiing with default option -1-, scanning 100 ports")
    portsToScan = commonports100

random.shuffle(portsToScan) #randomize the order of the selected choice

# Following, 'ipfile' is 1st argument inserted with distincted IPs.
# 'portsfile' is the 2nd argument inserted when run this script (run from 'runrecon.sh'). It
is where the open_ports results will be stored
with open(ipfile, 'r') as ipf, open(portsfile, 'w', newline='') as portscsv:
    csv_writer = csv.writer(portscsv)
    for line in ipf:
        ipsno=ipsno-1
        tempip = line.strip() #Strip the ip for spaces or linebrake characters
        scanport(tempip) #run the above script. It gives a list with open ports
        print(f"For ip {tempip} found {len(open_ports)} open ports")
        if len(open_ports) > 0:
            csv_writer.writerow([tempip] + open_ports) # Write the above list in the CSV
file
        open_ports=[]
        closed_ports=[]
    print(f"\nResults saved in CSV file in the path: {portsfile}")
```

File "**checkservices.sh**"

```
#!/bin/bash

# Author: George Tasios
# FileName: checkservices.sh

input_file=$1
output_XML=$2
final_output_CSV=$3
user_agents_file="core_useragents.txt"
useragent="Mozilla/5.0"
random_user_agent=""
isipv6=False


#insert file containing functions to call
source ./func.sh


#Initializing variables
ip=""
ports=""


test_ip() {
    # Check if the ip variable contains an IPv4 address
    if [[ "$ip" =~ ^([0-9]{1,3}\.){3}[0-9]{1,3}$ ]]; then
        echo "$ip is an IPv4 address."
    # Check if the ip variable contains an IPv6 address
    elif [[ "$ip" =~ ^([0-9a-fA-F]{1,4}:){7,7}[0-9a-fA-F]{1,4}$|^([0-9a-fA-
F]{1,4}:){1,7}:|^([0-9a-fA-F]{1,4}:){1,6}:[0-9a-fA-F]{1,4}$|^([0-9a-fA-
F]{1,4}:){1,5}(:[0-9a-fA-F]{1,4}){1,2}$|^([0-9a-fA-F]{1,4}:){1,4}(:[0-9a-fA-
F]{1,4}){1,3}$|^([0-9a-fA-F]{1,4}:){1,3}(:[0-9a-fA-F]{1,4}){1,4}$|^([0-9a-fA-
F]{1,4}:){1,2}(:[0-9a-fA-F]{1,4}){1,5}$|^[0-9a-fA-F]{1,4}:((:[0-9a-fA-
F]{1,4}){1,6})$|^:((:[0-9a-fA-F]{1,4}){1,7}|:)$|^fe80:(:[0-9a-fA-
F]{0,4}){0,4}%[0-9a-zA-Z]{1,}$|^::(ffff(:0{1,4}){0,1}:){0,1}((25[0-5]|(2[0-
4]|1{0,1}[0-9]){0,1}[0-9])\.){3,3}(25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])$|^([0-
9a-fA-F]{1,4}:){1,4}:((25[0-5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])\.){3,3}(25[0-
5]|(2[0-4]|1{0,1}[0-9]){0,1}[0-9])$ ]]; then
        echo "$ip is an IPv6 address."
        isipv6=True #Variable used as a flag, noting that the current ip is IPv6
version
    else
        echo "$ip is not a valid IP address."
    fi
}


print_ports() {
    local num=0
    for i in ${ports//,/ }
    do
      ((num++))
      echo -e "No $num opened port is: $i"
    done
}


select_random_useragent() {
    #Checks if file with user agents exists
    if [ ! -f "$user_agents_file" ]; then
        echo "File with user agents not found: $user_agents_file"
        random_user_agent="Mozilla/5.0"
        echo "setting default user agent: $useragent"
        exit 1
    fi
    # Select randomly a user agent from the file
    random_user_agent=$(shuf -n 1 "$user_agents_file")
    echo -e "Random user agent selected is:\n$random_user_agent\n"
}


nmapipv4() {
    # Creating local array variable where the ports will be stored as an array
and not as a comma separated list
    local ports_array

    # convert comma separated ports from $ports into array which contains the
ports
    IFS=',' read -r -a ports_array <<< "$ports"

    for port in "${ports_array[@]}"; do
        # function that selects Random user agent from file-list
        select_random_useragent
        # nmap command using custom selected user agent
        nmap -Pn -sV -T4 -n --script http-methods --script-args
http.useragent="$random_user_agent" $ip -p "$port" -oX $output_XML
        # Here will run the XML parsing procedure ('XMLprocess.py' script) and
elements will be appended to final csv output file (3rd argument of
'checkservices.sh')
        python3 XMLprocess.py $output_XML $final_output_CSV
    done
}


nmapipv6() {
    # Creating local array variable where the ports will be stored as an array
and not as a comma separated list
    local ports_array
    # convert comma separated ports from $ports into array which contains the
ports
    IFS=',' read -r -a ports_array <<< "$ports"

    for port in "${ports_array[@]}"; do
        # function that selects Random user agent from file-list
        select_random_useragent
        # nmap command using custom selected user agent
        nmap -Pn -sV -T4 -6 -n --script http-methods --script-args
http.useragent="$random_user_agent" $ip -p "$port" -oX $output_XML
        # Here will run the XML parsing procedure ('XMLprocess.py' script) and
elements will be appended to final csv output file (3rd argument of
'checkservices.sh')
        python3 XMLprocess.py $output_XML $final_output_CSV
    done
}

# In the 1st step of this file the following function is executed. Checks if nmap
tool is installed. If not, the script stops (this function exists in func.sh)
nmapCheck


# Reads the whole file with the IPs and the ports.
# Iterates EVERY LINE and stores the IP in the 'ip' variable and the ports in the
'ports' variable
while IFS=',' read -r ip ports
do
    isipv6=False
    #echo "$ip"
    #echo "$ports"
    test_ip #Checks the current selected IP and checks if it is ipv6. The
'test_ip' function an ipv6 found sets the variable/flag==True
    print_ports
    if [[ $isipv6 == True ]]; then
        echo -e "\nfound IPv6\n"
        nmapipv6
    elif [[ $isipv6 == False ]]; then
        echo -e "\nfound IPv4\n"
        nmapipv4
    fi
done < $input_file
```

File “**XMLprocess.py**”

```
# Author: George Tasios
# FileName: XMLprocess.py

# XML process: Takes as 1st argument the output XML file from nmapipv4 or
nmapipv6 functions of 'checkservices.sh' script
# exports and append the appropriate elements to the last output CSV file


import csv
import xml.etree.ElementTree as ET
import sys

# importing XML file as 1st argument. Used in XMLparse function
xmlfile = sys.argv[1]
# importing as 2nd argument
final_output_CSV = sys.argv[2]
finalCSVfilepath = ''
```

```
# CSV headers titles:
# hoststatus", "address,  addresstype,  portnum,  protocol,  portState,  CPE,
servName,  servConfidence,  scanMethod,  Product,  serviceVer,  extraServInfo,
OStype,  deviceType


hostitems = []
hostitemstemp = []

def XMLparse(XMLfile):
    #creating xml tree object
    tree = ET.parse(xmlfile)
    #getting root element of the xml 'tree'
    root = tree.getroot()

    ### Iterate 'host' items
    for host in root.findall('./host'): # Finds all 'host' items in the tree
        temp = {} # Creating an empty dictionary called 'TEMP'
        # iterate children in host
        for child in host:
            if child.tag == 'status':
                temp['hoststatus'] = child.attrib['state']
            elif child.tag == 'address':
                temp['address'] = child.attrib['addr']
                if 'addrtype' in child.attrib:
                    temp['addresstype'] = child.attrib['addrtype']
                else:
                    temp['addresstype'] = ""

    # Create sub_tag object for 'ports' tag
    # finds 'ports' tag in the host 'tag'. We use the script once for each
scanned IP, so it will normally find one 'ports' tag each time
    ports = host.find('ports')

    # iterate 'port' items (children of 'host' item)
    for port in ports.findall('port'):
        temp['portnum'] = port.get('portid')
        temp['protocol'] = port.get('protocol')
        #iterates child items of each 'port' tag

        for child in port:
            if child.tag == 'state': # state element is child of port and appears
always  ONCE
                temp['portState'] = child.get('state')
            temp['CPE'] = "" #initializing temp['CPE'] value

            if child.tag == 'service': # Cardinality of 'service' is -?- (none or
one)
                temp['servName'] = child.get('name') # Required
                temp['servConfidence'] = child.get('conf') # Required
                temp['scanMethod'] = child.get('method') # Required
                if 'product' in child.attrib:
                    temp['Product'] = child.get('product')
                else:
                    temp['Product'] = ""

                if 'version' in child.attrib:
                    temp['serviceVer'] = child.get('version')
                else:
                    temp['serviceVer'] = ""

                if 'extrainfo' in child.attrib:
                    temp['extraServInfo'] = child.get('extrainfo')
                else:
                    temp['extraServInfo'] = ""

                if 'ostype' in child.attrib:
                    temp['OStype'] = child.get('ostype')
                else:
                    temp['OStype'] = ""

                if 'devicetype' in child.attrib:
                    temp['deviceType'] = child.get('devicetype')
                else:
                    temp['deviceType'] = ""

                # Find 'Service' tag children and then check if any CPE element
exists
                service = port.find('service')
                tmp = ""
                for childr in service:
                    if childr.tag == 'cpe':
                        tmp = childr.text + ";" + tmp
                        temp['CPE'] = tmp
                    else:
                        temp['CPE'] = ""
            else: #what follows run only if 'service' child wasn't found
                temp['servName'] = temp['servConfidence'] = temp['scanMethod'] =
temp['Product'] = temp['serviceVer'] = ""
                temp['extraServInfo'] = temp['CPE'] = temp['OStype'] =
temp['deviceType'] = ""

        hostitems.append(temp)
        dict_to_csvfile(temp, final_output_CSV) # Append results in the csv file.
Creates headers row if do not exist
        hostitems.clear()



# Function that writes what the 'hostitems' dictionary contains, into a csv file
def dict_to_csvfile(data, csvfile):
    file_exists = True

    try:
        with open(csvfile, 'r') as f:
            if len(f.readline()) == 0:
                file_exists = False
    except FileNotFoundError:
        file_exists = False

    with open(csvfile, 'a', newline='') as f:
        #....writing data to CSV file
        fieldnames = data.keys()
        writer = csv.DictWriter(f, fieldnames=fieldnames)

        if not file_exists:
            writer.writeheader()
        writer.writerow(data)
```

```
XMLparse(xmlfile)
```

File “**prepareNeo4jFiles.sh**”

```
#!/bin/bash

# Author: George Tasios
# FileName: prepareNeo4jFiles.sh

# Creation of subfolder for the visualization graphs of the current domain
# visualsfoldername="${FINALDOMAIN}_neo4jVisuals"
mkdir -p -m 777 report/$1/neo4jVisuals

# convert results files of subdomains, ips, ports, services etc into csv files
with header suitable to import in neo4j
echo "subdomain" > report/$1/neo4jVisuals/subdomains.csv
cat $2 >> report/$1/neo4jVisuals/subdomains.csv
csv_subdomains_path="report/$1/neo4jVisuals/subdomains.csv"

echo "ip" > report/$1/neo4jVisuals/uniqueips.csv
cat $3 >> report/$1/neo4jVisuals/uniqueips.csv
csv_uniqueips_path="report/$1/neo4jVisuals/uniqueips.csv"

echo "subdomain,ip" > report/$1/neo4jVisuals/domain_ips.csv
cat $4 >> report/$1/neo4jVisuals/domain_ips.csv
csv_domain_ips_path="report/$1/neo4jVisuals/domain_ips.csv"
content=$(cat "report/$1/neo4jVisuals/domain_ips.csv")
modif=${content// /,}
echo "$modif" > "report/$1/neo4jVisuals/domain_ips.csv"


cat $5 > report/$1/neo4jVisuals/services.csv

# Convert open ports csv results file into a suitable to parse in neo4j

echo "IP,Ports" > "report/$1/neo4jVisuals/open_ports.csv"
# read the csv input file line by line
while IFS= read -r line; do
    # Extract IPs and the ports
    ip=$(echo $line | cut -d',' -f1)
    ports=$(echo $line | cut -d',' -f2- | tr -d '\n' | tr -d '\r')
    modified_ports="[$ports]"

    # write the formatted data to the open_ports.csv file
    echo "$ip,\"$modified_ports\"" >> "report/$1/neo4jVisuals/open_ports.csv"
done < $6
```

File “**pyToNeo.py**”

```
# Author: George Tasios
# FileName: pyToNeo.py

import pandas as pd
from py2neo import Graph, Node, Relationship
import os
import sys
import csv
```

```
# first argument is the domain name
target = sys.argv[1]

# File paths
subdomains_path = 'report/'+target+'/neo4jVisuals/subdomains.csv'
unique_ips_path = 'report/'+target+'/neo4jVisuals/uniqueips.csv'
domain_ips_path = 'report/'+target+'/neo4jVisuals/domain_ips.csv'
services_path = 'report/'+target+'/neo4jVisuals/services.csv'
open_ports_path = 'report/'+target+'/neo4jVisuals/open_ports.csv'

# Checks if exists anything in a file
def is_file_empty(file_path):
    return os.path.getsize(file_path) == 0


# Reads the domain-target ($FINALDOMAIN variable)
domain = sys.argv[1]

# Read the CSV files
subdomains_df = pd.read_csv(subdomains_path)
unique_ips_df = pd.read_csv(unique_ips_path)
domain_ips_df = pd.read_csv(domain_ips_path)
open_ports_df = pd.read_csv(open_ports_path)
# Checks if exist results of services in 'services.csv' file
if not is_file_empty(services_path):
    services_df = pd.read_csv(services_path)


# Connect to Neo4j (make sure Neo4j is running and accessible)
graph = Graph("bolt://localhost:7687", auth=("neo4j", "neo4j12345"))

# Clear existing graph data
graph.delete_all()

# Creation of main node of the target-domain
domain_node = Node("Domain", name=domain)
graph.create(domain_node)


# Create subdomain nodes and relationships
for _, row in subdomains_df.iterrows():
    subdomain_node = Node("Subdomain", name=row['subdomain'])
    graph.create(subdomain_node)
    if domain_node:
        contains_subdomain_rel = Relationship(domain_node, "ContainsSubdomain",
subdomain_node)
        graph.create(contains_subdomain_rel)


# Create ips unique nodes
for _, row in unique_ips_df.iterrows():
    ipname = str(row['ip'])
    ip_node = Node("IP", address=row['ip'])
    graph.create(ip_node)


# Create IP-Subdomains relationships
for _, row in domain_ips_df.iterrows():
    current_subdomain_node = graph.nodes.match("Subdomain",
name=row['subdomain']).first()
```

```
    # List with IP nodes
    ip_nodes = list(graph.nodes.match("IP", address=row['ip']))
    for node_a in ip_nodes:
        common_ip = str(node_a['address'])
        if row['ip'] == common_ip:
                resolve_to_rel = Relationship(current_subdomain_node,
"ResolveTo", node_a)
                graph.create(resolve_to_rel)


if is_file_empty(services_path):
    exit(1)
else:
    # Create service nodes and relationships
    for _, row in services_df.iterrows():
        ip_node = graph.nodes.match("IP", address=row['address']).first()
        if ip_node:
            service_node = Node("Service", port=row['portnum'],
protocol=row['protocol'], state=row['portState'], cpe=row['CPE'],
                                service_name=row['servName'],
confidence=row['servConfidence'], method=row['scanMethod'],
product=row['Product'])
            graph.create(service_node)
            contains_service_rel = Relationship(ip_node, "ContainsService",
service_node)
            graph.create(contains_service_rel)
```

File “**core_useragents.txt**”

```
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3) AppleWebKit/537.75.14 (KHTML, like Gecko)
Version/7.0.3 Safari/7046A194A
Mozilla/5.0 (Macintosh; U; PPC Mac OS X; es) AppleWebKit/419 (KHTML, like Gecko)
Safari/419.3
Mozilla/5.0 (Macintosh; U; PPC Mac OS X; fr-fr) AppleWebKit/312.1 (KHTML, like Gecko)
Safari/312
Mozilla/5.0 (Macintosh; U; Intel Mac OS X; en) AppleWebKit (KHTML, like Gecko)
Mozilla/5.0 (Linux; U; Android 4.0.3; ko-kr; LG-L160L Build/IML74K) AppleWebkit/534.30
(KHTML, like Gecko) Version/4.0 Mobile Safari/534.30
Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2228.0
Safari/537.36
Mozilla/5.0 (Windows NT 6.4; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/41.0.2225.0 Safari/537.36
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_2) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/36.0.1944.0 Safari/537.36
Mozilla/5.0 (Windows NT 6.2; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/32.0.1667.0 Safari/537.36
Mozilla/5.0 (X11; CrOS i686 4319.74.0) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/29.0.1547.57 Safari/537.36
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/535.24 (KHTML, like Gecko) Chrome/19.0.1055.1
Safari/535.24
Chrome/15.0.860.0 (Windows; U; Windows NT 6.0; en-US) AppleWebKit/533.20.25 (KHTML, like
Gecko) Version/15.0.860.0
Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:47.0) Gecko/20100101 Firefox/47.0
Mozilla/5.0 (Windows NT 6.1; U;WOW64; de;rv:11.0) Gecko Firefox/11.0
Mozilla/5.0 (X11; ; Linux x86_64; rv:1.8.1.6) Gecko/20070802 Firefox
Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; AS; rv:11.0) like Gecko
Mozilla/5.0 (compatible, MSIE 11, Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko
Mozilla/1.22 (compatible; MSIE 10.0; Windows 3.1)
Mozilla/5.0 (Windows; U; MSIE 9.0; Windows NT 9.0; en-US)
Mozilla/4.0 (MSIE 6.0; Windows NT 5.1)
Mozilla/4.0 (compatible; MSIE 5.5;)
Mozilla/5.0 (compatible; MSIE 9.0; Windows Phone OS 7.5; Trident/5.0; IEMobile/9.0)
```

```
Opera/9.70 (Linux ppc64 ; U; en) Presto/2.2.1 Version/2.2
Opera/9.70 (Linux ppc64 ; U; en) Presto/2.2.1
Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.0) Opera/34
Mozilla/5.0 (Windows NT 6.1; U; de; rv:1.9.1.6) Gecko/20091201 Firefox/3.5.6 Opera 11.01
HTC_HD2_T8585 Opera/9.70 (Windows NT 5.1; U; de)
Opera/9.63 (X11; Linux i686; U; ru)
Opera/9.80 (Linux i686; Opera Mobi/1038; U; en) Presto/2.5.24 Version/10.00
Mozilla/5.0 (compatible; MSIE 9.0; Windows Phone OS 7.5; Trident/5.0; IEMobile/9.0; SAMSUNG;
SGH-i917)
Mozilla/4.0 (compatible; MSIE 7.0; Windows Phone OS 7.0; Trident/3.1; IEMobile/7.0; LG;
GW910)
Mozilla/5.0 (compatible; MSIE 10.0; Windows Phone 8.0; Trident/6.0; IEMobile/10.0; ARM;
Touch; NOKIA; Lumia 920)
Mozilla/5.0 (iPad; CPU OS 7_0 like Mac OS X) AppleWebKit/537.51.1 (KHTML, like Gecko)
CriOS/30.0.1599.12 Mobile/11A465 Safari/8536.25 (3B92C18B-D9DE-4CB7A02A-22FD2AF17C8F)
Mozilla/5.0 (iPad; CPU OS 7_0 like Mac OS X) AppleWebKit/537.51.1 (KHTML, like Gecko)
Version/7.0 Mobile/11A465 Safari/9537.53
Mozilla/5.0 (iPhone; CPU iPhone OS 6_1_4 like Mac OS X) AppleWebKit/536.26 (KHTML, like
Gecko) Version/6.0 Mobile/10B350 Safari/8536.25
Mozilla/5.0 (iPhone; CPU iPhone OS 7_0 like Mac OS X) AppleWebKit/537.51.1 (KHTML, like
Gecko) Mobile/11A465 Twitter for iPhone
Mozilla/5.0 (iPhone; U; CPU iPhone OS 7_0_4 like Mac OS X; en-US) AppleWebKit/534.35 (KHTML,
like Gecko) Chrome/11.0.696.65 Safari/534.35 Puffin/3.11505IP Mobile
Mozilla/5.0 (iPod; CPU iPhone OS 5_1_1 like Mac OS X) AppleWebKit/534.46 (KHTML, like Gecko)
Version/5.1 Mobile/9B206 Safari/7534.48.3
Mozilla/5.0 (iPad; iOS 8.1.3 like OS X) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/40.0.2214.73 Safari/537.36
Mozilla/5.0 (Linux; U; Android 4.0.3; ko-kr; LG-L160L Build/IML74K) AppleWebkit/534.30
(KHTML, like Gecko) Version/4.0 Mobile Safari/534.30
Mozilla/5.0 (Linux; U; Android 2.3.4; fr-fr; HTC Desire Build/GRJ22) AppleWebKit/533.1
(KHTML, like Gecko) Version/4.0 Mobile Safari/533.1
Mozilla/5.0 (Linux; U; Android 2.3; en-us) AppleWebKit/999+ (KHTML, like Gecko) Safari/999.9
Mozilla/5.0 (Linux; U; Android 2.2.1; en-ca; LG-P505R Build/FRG83) AppleWebKit/533.1 (KHTML,
like Gecko) Version/4.0 Mobile Safari/533.1
```