Harokopio University of Athens

Department of Information and Telematics

Msc in Web Engineering

Thesis

Techniques for Identifying
unique
characteristics of Servers

Nicholaos Koroniotis

Acknowledgements

This thesis, was composed as a requirement for completing the Master's in Web Engineering program of the Department of Information and Telematics at Harokopio University of Athens.

During the implementation of this project, I came in contact with a number of technologies and at the same time, I had to apply most of the skills and knowledge that I acquired during my studies.

Also, I had a close cooperation with the personnel of the Department of Information and would like to thank all the people that provided me with the much needed support, that made the completion of this thesis possible.

More specifically, I would like to thank Dr. Panagiotis Rizomiliotis who trusted me with the assignment of this thesis and provided me with the necessary support.

Finally, this work was completed with the full moral support of my family, which supported me through all the difficulties that I encountered in the course of my studies .

Nicholas Koroniotis

Contents

Acknowledgements....3
Contents....5
Abstract....7
Περίληψη....9
1. Introduction....11
  1.1 Defining the problem....11
  1.2 Purpose of this thesis....13
2. So what is a vulnerability scanner?....15
  2.1 Defining vulnerabilities....15
  2.2 Defining vulnerability scanners....17
  2.3 Our approach....18
    2.3.1 Nmap, the port scanner....19
    2.3.1 The National Vulnerability Database....21
3. Implementing our Scanner....23
  3.1 Development related information....23
    3.1.1 Setting up the development environment....23
    3.1.2 Describing some key parts of the project....25
  3.2 Running our program....33
4. Conclusion and future work....35
Bibliography....36
A. Appendices....38
  Appendix A: The scanning module code....38
  Appendix B: The testingXMLParsing module code....40
  Appendix C: The vulnerabilityDetector module code....45

Nicholas Koroniotis

Nicholas Koroniotis

Abstract

The Internet has had a massive impact in our lives. Allowing instant communication between people located anywhere on the globe, granting the ability to make purchases by remotely accessing one's bank accounts and even providing a reliable access to global news as they unfold, it should surprise no one, that the general public would come to rely so much on this medium to make their lives simpler. Naturally though, where some see a way to improve their everyday lives, others see a chance to exploit the symbiotic connection that exists between humanity and the Internet. This exploitation comes in various forms of network/computer security attacks, for example: targeting servers and attempting to disrupt their natural flow of execution (Denial Of Service attacks), stealthily stealing sensitive information (Phishing attacks), or even infecting a multitude of Personal Computers (PCs) in a process of planning future attacks (Bot nets). A lot of the aforementioned scenarios are made possible by exploiting various security bugs in programs. Although most, if not all bugs are eventually fixed in future versions or patches, a user must be willing, or at least remember to perform the necessary updates, leaving such sensitive security matters open to the wild card that is human nature. As a result, the ability to either directly identify the existence of a vulnerability, or indirectly, locate the version of a program and track down any known vulnerabilities, the knowledge of which should be sufficient motivation for any security-conscious administrator to make the appropriate updates, would be a step forward into making the Internet a safer place.

The purpose of this thesis is the development of a Python program that, given an IP address will produce a report listing all identified programs running on that machine as well as any existing vulnerabilities associated with these programs. Our software utilizes nmap, a well known and open source program for the identification procedure and accesses the National Vulnerability Database (NVD) in order to identify the existence of any vulnerabilities.

Περίληψη

Το διαδίκτυο έχει επηρεάσει αισθητά τις ζωές μας. Επιτρέποντας αστραπιαία επικοινωνία ανεξαρτήτως αποστάσεων, δίνοντας την δυνατότητα για αγωρές καιδιαχήριση των τραπεζικών λογαριασμών εξ αποστάσεως και παρέχοντας μια αξιόπιστη πυγή παγκόσμιας πληροφόρησης με ταχύτατες ενημερώσεις, δεν θα έπρεπε να μας ξαφνιάζει το γεγονός ότι κατά πλειοψηφία εξαρτόμαστε ολοένα και περισσότερο σε αυτό το σχετικά νέο μέσο. Φυσικά όμως, όπου μερικοί βλέπουν έναν νέο δρόμο για να βελτιώσουν την καθημερινότητα τους, άλλοι βλέπουν μία ευκαιρία για να εκμεταλλευτούν κακόβουλα την "εξάρτιση" που έχει σχηματιστεί μεταξύ κοινωνίας και διαδικτύου. Αυτή η εκμετάλλευση έρχεται με την μορφή διαφόρων δικτυακών επιθέσεων ασφαλείας όπως: στόχευση εξυπηρετητών με σκοπώ την διακοπή της λειτουργίας τους (Denial Of Service attacks), κρυφή κλοπή ευαίσθητων πληροφοριών (Phishing attacks), ακόμα και "μόλυνση" πληθώρας προσωπικών υπολογιστών με σκοπό την μεταγενέστερη χρήση τους σε περετέρω επιθέσεις (Bot nets). Αρκετά από τα προαναφερθέντα σενάρια γίνονται εφικτά μέσω της εκμετάλλευσης λαθών ασφαλείας που εντοπίζονται μέσα σε προγράμματα. Αν και τα περισσότερα από αυτά τα λάθει, διορθώνονται είτε σε μεταγενέστερες εκδόσεις είτε με την έκδοση τοπικών διορθώσεων (patches), ο χρήστης του εκάστοτε υπολογιστικού συστήματος θα πρέπει να είναι πρόθυμος ή τουλάχιστον να θυμηθεί να πράξει τις απαραίτητες ανανεώσεις, αφήνοντας έτσι το λεπτό ζήτημα της ασφάλειας του υπολογιστικού του συστήματος στην απρόβλεπτη φύση του ανθρώπου. Σαν αποτέλεσμα, η ικανότητα είτε να εντοπίζουμε άμεσα την ύπαρξη μιας αδυναμίας, είτε έμμεσα να αναγνωρίζουμε την έκδοση ενός προγράμματος και έπειτα να αναζητούμε την πιθανών αδυναμιών, θα αποτελούσε ένα βήμα προς την δημιουργία ενός ασφαλέστερου διαδικτύου.

Ο σκοπός της παρούσης διπλωματικής εργασίας, είναι η ανάπτυξη ενός λογισμικού με χρήση της προγραμματιστικής γλώσσας Python, το οποίο, δεδομένης μίας διεύθυνσης IP, θα παράγει μία αναφορά στην οποία θα καταγράφονται όλα τα αναγνωρισμένα από το λογισμικό μας προγράμματα που τρέχουνε στο απομακρυσμένο μηχάνημα, σε συνδυασμό με τυχόν υπαρκτές αδυναμίες που συσχετίζονται με τα προγράμματα αυτά. Το πρόγραμμα μας αξιοποιεί το nmap, ένα πασίγνωστο λογισμικό ανοικτού κώδικα για την διαδικασία της ταυτοποίησης και χρησιμοποιεί την Διεθνή Βάση Δεδομένων Αδυναμιών (National Vulnerability Database) για τον εντοπισμό των αδυναμιών.

Nicholas Koroniotis

1. Introduction

1.1 Defining the problem

Consisting of a multitude of interconnected networks, a fact which lead to its name in 1974 [1], the Internet has undoubtedly had a major impact on our life. Ever expanding and evolving, it has existed for many years in many forms, some of which are quite different than what we see today. With the introduction of social media: e-commerce, on-line gaming, Internet Telephony, the World Wide Web and even streaming media, it is no wonder that this new medium of information grows in popularity each day.

Over the years, as the general public has come to rely upon the Internet, so too have many companies recognized its usefulness and have ended up incorporating it into various aspects of their business procedures, such as on-line advertising, newsletters, on-line ordering and so forth. An example of this evolution that has been brought by the Internet, is Amazon.com, Inc. [2]. Originally an on-line bookstore, now a multi-billion dollar company which serves consumers around the globe, Amazon has evolved over the years into selling a large variety of items ranging from DVDs, audiobooks, video games, furniture and even food while at the same time provides users, with access to cloud computing services and other electronic goods and services.

Everyones attention is, in one way or another focused on the Internet. Considering the fact that, most conduct business over it, and since where there's a way to make an honest buck, there's also a way to make a dishonest one, it is pretty understandable that illegal activities would arise and, in some cases thrive in the Internet.

As stated in [3] and [4], cyber attacks that target every node of the Internet infrastructure, have become more stable in their frequency over the years, increasing both in complexity and in severity. Attacks which target both software as well as hardware have been recorded, costing billions to companies who are most frequently the targets. In most scenarios, a vulnerability is required in order to function as the fuse that detonates the attack, and so the ability to detect the presence of such vulnerabilities is paramount to ensure that ones infrastructure is approaching a secure state. In order to reach this goal, a certain type of software was developed, and this family of softwares is called vulnerability scanners.

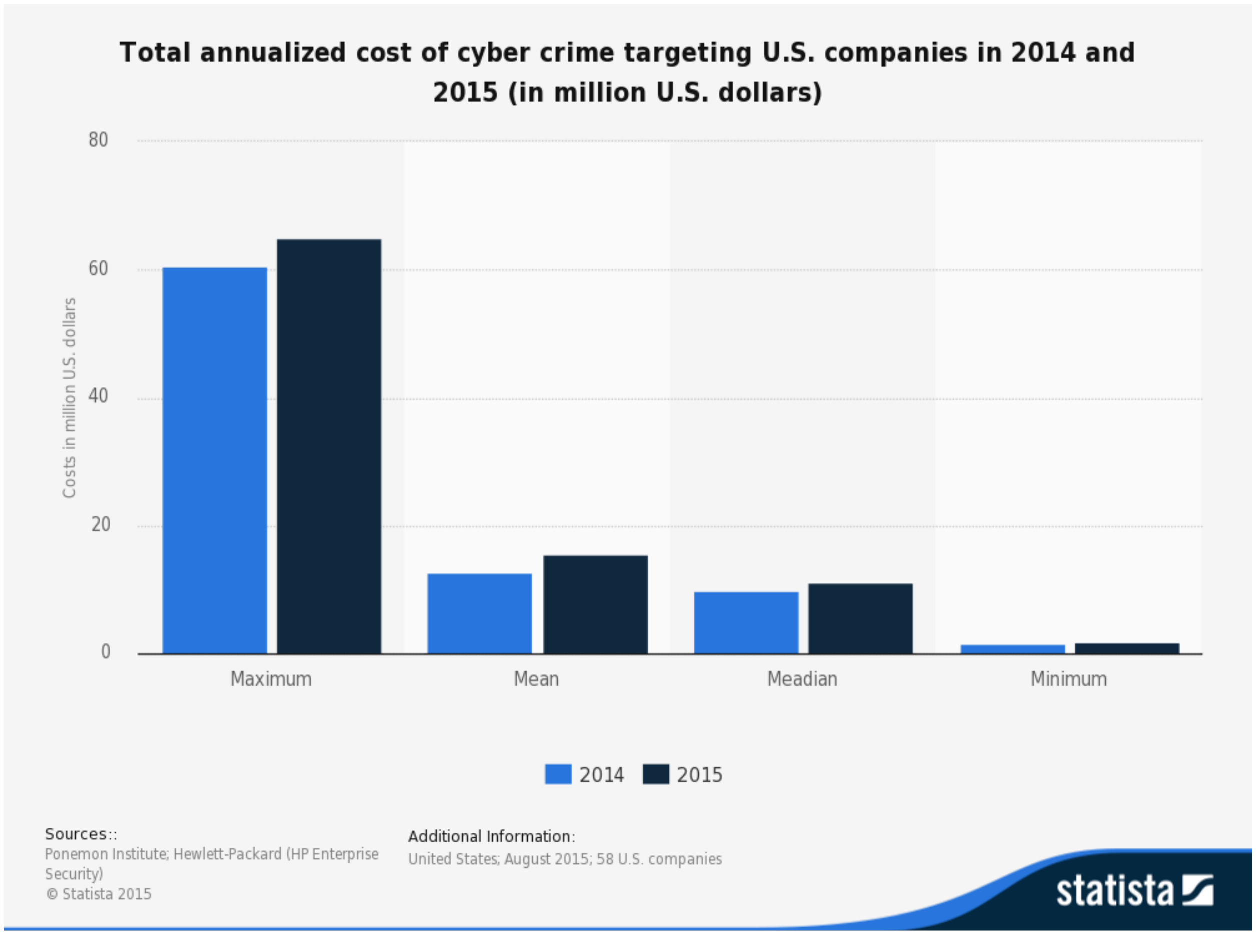


Image 1.1: Cost of cyber attacks targeting U.S. Companies 2014-2015

Source: http://www.statista.com/statistics/193444/financial-damage-caused-by-cyber-attacks-in-the-us/

The above chart (which was found on the statista site), depicts the estimated damage a successful cyber attack will cost a U.S. business between the years 2014 to 2015, with the maximum total annualized cost occurring in 2015 and reaching up to 65.05 million U.S. dollars.

1.2 Purpose of this thesis

At its core, the purpose for implementing this dissertation, is the creation of a software that scans a remote host by receiving it's IP address, identifies any applications that have network access and are running on the machine, and finally attempts to identify any vulnerabilities that might arise, due to either the existence of a vulnerable version of a program, or a vulnerable configuration of multiple programs.

The people that should be interested in this software, are server administrators who wish for an easy way to see if their system is at risk at any given time, security annalists who require a portable, platform-independent tool that will assist them in their work and finally any security aware individuals who intend to keep their Personal Computers (PCs) updated with the latest security patches.

2. So what is a vulnerability scanner?

In this chapter, we will define what a vulnerability scanner is and does. But to do so, we first need to understand what a vulnerability is.

2.1 Defining vulnerabilities

According to [5] and [6], there are many ways one can define a vulnerability, considering the fact that they exist in many forms and affect more than one type of asset (software, hardware, etc). For the purposes of this thesis, we are more interested in defining software vulnerabilities. A software vulnerability is a weakness in a software, which allows an attacker to violate the security of a system, or more generally access the system and perform actions, in a way not originally planned by the architect of that system, such as perform phishing attacks with Cross Site Scripting, attempt to compromise or illegally gain access to the system's database with SQLInjections and so forth. It is stated in [5], that a vulnerability can be thought as the intersection of three elements: 1) a system flaw, 2) an attacker's access to that flaw and 3) an attacker's capability to exploit the flaw, characterizing the vulnerability as the attack surface that is, the way that an adversary performs an attack. A vulnerability can surface due to various reasons, such as([5]):

1. Complexity:The complexity of the system, making a system complex, increases the risk of creating unintentional flaws and access points
2. Unchecked user input: Assuming that any input inserted by the user is safe, is a prelude to disaster, as neglecting to make the proper checks and sanitization may lead to direct execution of commands (SQL injection)
3. Software bugs: Can lead to the misuse of the application by a malicious user.

A famous, or rather infamous example of a software vulnerability, is the quite recent. Heartbleed vulnerability. The Heartbleed vulnerability, was discovered in the beginning of April 2014 and affected some versions of the OpenSSL cryptography library, more specifically OpenSSL 1.0.1 through 1.0.1f and OpenSSL 1.0.2-beta. The security bug functioned, by taking advantage of a lack of checking the integer field that depicted the sent text's length that was present in the heartbeat request of the heartbeat extension of the vulnerable OpenSSL versions, against the actual length of the text that was sent, allowing an attacker to send a relatively small string accompanied by a large integer value, which in turn would allow the attacker to receive, apart from the proper response, parts of the victim's memory, that could contain sensitive information, such as primary keys, username passwords (identifiers) etc [7], [8].

2.2 Defining vulnerability scanners

Now that we have a general idea of what a vulnerability is, we can define the purpose for vulnerability scanners and how they function. A vulnerability scanner at its core, is a piece of software that scanns computing systems, networks or applications in order to identify any existing flaws/vulnerabilities that may be exploited by an attacker([9]). There are many types of scanners, such as port scanners and web application security scanners, with each type having significant differences in their purpose and way of functioning, for instance, port scanners are programs that probe a host (machine) in order to determine the existence of open ports.

Nmap, is a good example of port scanning software, supporting both port and service scanning, the latter being the process of identifying the services running behind open ports, which **Nmap** achieves by probing the remote probes and comparing the results it receives to those stored in a local database. We shall describe Nmap in more detail in a further chapter, since we relied upon both its port and service scanning for the service identification of our project.

Other vulnerability scanners, include **Nessus** and **OpenVAS**. Initially named GNessUs, OpenVas is an open source vulnerability scanner which contains over 35.000 network vulnerability tests and was developed as a fork project since Tenable Network Security made Nessus a closed source software in October 2005 [15]. Nessus, is a vulnerability scanner the development of which initially started in 1998 and became the most famous vulnerability scanner of 2002, 2003 and 2006 [16].It providing a multitude of scanning operations, such as Denial of Service, remote access attacks and even various password-related attacks on system accounts [16].

2.3 Our approach

In our project, we chose to rely upon many well established technologies. We chose Python (3.4.3) as the programming language to use, for it is considered quite easy to use, and makes our software portable, since Python is platform independent. As we progressed with the development of our scanner, we chose to split its architecture into two main parts:

1. the service identification segment, and
2. the vulnerability searching segment.

The first part (service identification), utilizes nmap in order to identify any open ports and services running on those ports. When the nmap scan ends, the produced output is forwarded to the second segment (vulnerability search), which accesses the National Vulnerability Database(NVD), in order to identify any existing vulnerabilities based on the services that nmap identified.

In the following sub-chapters, we will describe all the aforementioned aspects of our project, starting with Nmap, then saying a few words about NVD and finally, in the next chapter we will examining the steps we took to implement our program.

2.3.1 Nmap, the port scanner

Nmap, is an open source multipurpose scanning tool, originally released in September 1997 by Gordon Lyon (also known as Fyodor Vaskovich)([10], [11]). Originally developed for Linux systems, it has since been ported into a multitude of other platforms, such as Windows and Solaris. It has various features, such as: 1) host discovery (Identifies hosts that respond to TCP or ICMP requests), 2) port scanning (Lists the open ports of a scanned host), 3)version detection (Determines the version of network services based on their response to specially crafted requests, found in Nmap's Database), OS detection (Detects the Operating System as well as general hardware characteristics of a host), and can be used by either an administrator or security analyst for performing routine scans on their machines and networks for maintenance or auditing reasons, or by an attacker (black hat) for identifying hosts with open ports their operating system as well as the services running on any open ports, which in turn leads to a better targeted attack on the targeted system.

Our program utilizes Nmap's service scanning feature (via a third party module), which performs a service scan (Nmap argument: -sV) and thus identifies any open ports and the services running under them. Nmap version scanning, is designed to be a fast and simple process, and can be described as follows:

Initially, a port scan is performed, in order to identify any open **or** possible open ports (characterized by Nmap as open|filtered). After the scan is complete, the open and open|filtered ports of either TCP or UDP protocol, are passed as input to a service scanning module, which probes each port in parallel, and attempts to identify the running services under each port. From here, Nmap's behavior depends upon the type of port it is currently scanning. If the port is a TCP port, then Nmap attempts to connect to that port. If the connection is established successfully, then Nmap waits for approximately 5 seconds, since many services produce a banner describing their identity, this is called the “Null probe” since Nmap does not send any data in this stage of the scan. If Nmap succeeds in fully identifying the service by using the data received in these first five minutes, then the scan for that port is over. Otherwise, if the identification is partial, Nmap proceeds by sending the next probe, making sure to pick probes that are likely to lead to a full identification based on the information that it has already extracted from the previous probe.

Now if the “Null probe” fails, Nmap uses the **probable ports** field of the probes, which indicates that a probe is considered to be most effective for the specified probes, to speed up the scanning process. Now if this method fails to identify the service as well, then the entirety of the probe database that Nmap keeps is used, which as mentioned in the official Nmap site, is quite time consuming [12].

2.3.1 The National Vulnerability Database.

The National Vulnerability Database or **NVD** for short, is a repository of vulnerability related data, has been publicly available since 2005, managed by the National Institute of Standards and Technologies, and belonging to the U.S Government. Information that is stored in the NVD, is freely accessible by anyone and it is kept in XML form. Updates for the various entries are often provided and when a new vulnerability is discovered, a new entry is added to the Database.

All XML files belonging to NVD have a similar built, starting with the *`entry`* tag which encapsulates each entry in the file, has a unique **ID** for every entry and all the information about a vulnerability. A *`vuln:vulnerable-configuration`* tag which contains lists of software, that if exist together in a machine, may cause the vulnerability that is described by that entry tag. A *`vuln:cvss`* tag which contains various information about the vulnerability's nature itself, such as: Access-vector, access complexity, the need for authentication (to use the vulnerability)and **Confidentiality Integrity** and **Availability** impact. There is also a *`Summary`* field which gives a short description about the vulnerability and what an adversary can do by using it.
Here is an example of an entry:

```
<entry id="CVE-2016-0003">
    <vuln:vulnerable-configuration id="http://nvd.nist.gov/">
      <cpe-lang:logical-test operator="OR" negate="false">
        <cpe-lang:fact-ref name="cpe:/a:microsoft:edge:-"/>
      </cpe-lang:logical-test>
    </vuln:vulnerable-configuration>
    <vuln:vulnerable-software-list>
      <vuln:product>cpe:/a:microsoft:edge:-</vuln:product>
    </vuln:vulnerable-software-list>
    <vuln:cve-id>CVE-2016-0003</vuln:cve-id>
    <vuln:published-datetime>2016-01-13T00:59:02.683-05:00</vuln:published-
datetime>
    <vuln:last-modified-datetime>2016-01-14T10:45:21.937-05:00</vuln:last-
modified-datetime>
    <vuln:cvss>
      <cvss:base_metrics>
```

```
        <cvss:score>9.3</cvss:score>
        <cvss:access-vector>NETWORK</cvss:access-vector>
        <cvss:access-complexity>MEDIUM</cvss:access-complexity>
        <cvss:authentication>NONE</cvss:authentication>
        <cvss:confidentiality-impact>COMPLETE</cvss:confidentiality-impact>
        <cvss:integrity-impact>COMPLETE</cvss:integrity-impact>
        <cvss:availability-impact>COMPLETE</cvss:availability-impact>
        <cvss:source>http://nvd.nist.gov</cvss:source>
        <cvss:generated-on-datetime>2016-01-13T20:49:08.090-
05:00</cvss:generated-on-datetime>
      </cvss:base_metrics>
    </vuln:cvss>
    <vuln:cwe id="CWE-119"/>
    <vuln:references xml:lang="en" reference_type="VENDOR_ADVISORY">
      <vuln:source>MS</vuln:source>
      <vuln:reference
href="http://technet.microsoft.com/security/bulletin/MS16-002"
xml:lang="en">MS16-002</vuln:reference>
    </vuln:references>
    <vuln:summary>Microsoft Edge allows remote attackers to execute arbitrary
code via unspecified vectors, aka "Microsoft Edge Memory Corruption
Vulnerability."</vuln:summary>
  </entry>
```

3. Implementing our Scanner

In this chapter, we will describe the steps we took to setup the development environment by giving the commands we used to install the various tools that were needed for the development of our project. Afterward, we shall describe some key parts of the code and give examples of our software in action.

3.1 Development related information

Our implementation took place on an Ubuntu 14.04, 64-bit processor personal computer (PC). We chose to work on Ubuntu, since it's an easy to use, open source operating system (OS), and we had some personal experience working on such machines. As such, all following commands regarding the installation of the various segments that make up our scanner, can be applied on a PC with the same, or possibly similar characteristics (OS family, CPU architecture,etc.) as the one we used during our own implementation.

3.1.1 Setting up the development environment

Since, as we mentioned earlier we have some experience in using Linux-based computers, we installed the Python 3.4.3 interpreter, via terminal. So, after accessing the terminal, we typed the following commands. It should be noted here, that in order to run the following commands, one should have superuser privileges (sudo).

```
sudo apt-get install python3.4.3
```

After installing python, we need to install **pip** in order to proceed. PIP is a package management system, that simplifies the installation of python packages, and so it will assist us in adding the third party packages we need in our implementation([13]).

```
sudo apt-get install python3-pip
```

Now, by using pip we are able to install some modules that our scanner utilizes. Specifically, we will install the lxml, urllib3 and python-nmap.

```
pip install lxml
pip install urllib3
pip install python-nmap
```

The lxml module provides easy access to xml files.Our software uses the lxml module, in order to access the National Vulnerability Database (NVD) which is a series of XML files containing vulnerabilities, the software version responsible for it and various other information describing the security bug's characteristics. In order to download locally the entirety of the NVD Database, as well as perform updates when needed, our project uses the urllib3 module to access the individual URLs and download the (XML) files necessary. Finally, we use nmap in order to perform scans on remote hosts, and so we need python-nmap which provides access between the nmap tool and Python code. Our last statement, implies that we require nmap for our scanner to function, so we installed nmap in our machine by using the following command.

```
sudo apt-get install nmap
```

To develop our Python code, we used the Python IDE (Intergrated Development Environment) PyCharm Community Edition 5.0.3 which can be found at their official site (*https://www.jetbrains.com/pycharm/*). In order to run pycharm, one needs to first install a java runtime environment (jre). For example :

```
sudo apt-get install openjdk-7-jre
```

3.1.2 Describing some key parts of the project

In this sub-chapter, we will attempt to give a thorough description of our source code. Lets start off by saying that our project consists of three Python modules and two text files, each containing 15 URLs. The modules are named: **scanning**, **testingXMLParsing**, and **vulnerabilityDetector** and the text files: **NVD_DBLinks**, and **NVD_DB_METADATA**.

The text files contain URLs that allow our program to access and download specific files, more precisely in the NVD_DBLinks text file, reside the URLs that point to the compressed forms of the NVD Database (which is segmented based on the years since 2002, and are in XML form) and in the NVD_DB_METADATA, the URLS that point to the metadata describing the individual XML file, which allows our application to update any part of its database if it notices any changes based on the metadata (our database keeps a local copy of the metadata files for each segment from the latest download, so that it can tell when our database is outdated).

As we mentioned in the above text, our Python program consists of three modules. Here we are going to describe some key parts of the code with some detail and afterwards, we will display our software in action. Our fully commented and complete code can be found in the Appendix section.

Describing the scanning module code

To begin with, we have the **scanning** module. This module handles the user input (an ip address), and attempts to scan the host by accessing nmap via the python-nmap module.

```
def mkProfilesDir():
    if os.path.isdir(profileDir)==False:
        os.mkdir(profileDir)
def mkProfDir(ip):
    global currentPath
    if os.path.isdir(profileDir+"/"+ip)==False:
        os.mkdir(profileDir+"/"+ip)
    currentPath=profileDir+"/"+ip
```

These two functions, are tasked with creating the appropriate directories for storing a report produced by scanning a host. The **`mkProfilesDir`**, produces the single **profiles** directory (if it isn't present), under which all the individual profile directories for each distinct host scanned are crafted by the **`mkProfDir`**.

```
def fullNmapScan(ip):#Nmap scan
    nmS=nmap.PortScanner()
    nmS.scan(hosts=ip,arguments='-sV')
    return nmS.csv()
```

The **`fullNmapScan`**, as its name sugests, calls the nmap tool, in order to start scanning the host to which the IP address we provide belongs to, and uses the arguments we specified in the code. Specifically the `'-sV'` arguments inform nmap that we wish to perform a probing scan of all open ports that can be found by nmap and produce the service name and version of any services running under those ports.

```
def createReport(writeContent):#Method that creates a report
    global currentPath
    if os.path.isdir(currentPath):
        currentPath=currentPath+"/"+str(datetime.date.today().year)
+"_"+str(datetime.date.today().month)+"_"+str(datetime.date.today().day)
        if os.path.isdir(currentPath)==False:
            os.mkdir(currentPath)

currentPath=currentPath+"/report_"+str(datetime.datetime.now().time())
        if os.path.isdir(currentPath)==False:
            os.mkdir(currentPath)
        currentPath=currentPath+"/report_Simple"
        profile=open(currentPath,'w')
        profile.write(writeContent)
        profile.close()
        return currentPath
```

As the name suggests, **`createReport`**,is the function that produces one of the two kinds of reports that our scanner creates. This function creates a simple report, which informs the user about the network services that were located by nmap, and the number of any possible existing vulnerabilities that our program located by accessing NVD. As can be seen in our code, in this function, more directories are produced, separating the scans of an individual machine, based on the date and afterwards on the time of the scan.

```
def main():
    global currentPath
    while(True):
        ip=input("Enter ip address: ")
        address=None
        try:
            address=socket.gethostbyname(ip)#Try Catch way of checking if ip
is valid. IP returned
            break
        except:
            print("\tWrong input please enter IPv4 address of a form like
192.168.1.1")
    print("Commencing scan of "+ip+" host")
    buffer=fullNmapScan(address)
    print("Scan complete.")
    results=buffer.split("\r\n")
    mkProfilesDir()
    mkProfDir(ip)
    reportPath=createReport(buffer)
    findVulnerability(reportPath)
```

Finally, we have the main function, which is called when we run the scanning module. Here we can see the execution flow of the program, starting with handling the user input. The user is prompted to type in a valid IP address. If the IP address is valid, the program exits the while loop , forwarding the given address to nmap and awaiting the data returned by it. Afterwards the appropriate directories are created (if they do not exist) and finally, in order to initiate the vulnerability search procedure, a function named **`findVulnerability`**, which belongs to the the **vulnerabilityDetector** module is called.

Describing the vulnerabilityDetector module code

Next up, we have the **vulnerabilityDetector** module. This module's main responsibility, is to make the initial download of the database, create the necessary directories within which the database is stored and finally keep the database up to date (when the need arises).

```
def downloadZip():
    global url,path
    checkMetaData()
    nvdLinks=open('NVD_DBLinks','r')
    http=urllib3.PoolManager()
    for link in nvdLinks:#Maybe thread this and add the META file in order to
update propperly
        url=link.rstrip("\n")
        buf=url
        buf=buf.replace(".xml.zip","")
        bufList=buf.split("/")
        dirName=bufList[len(bufList)-1]
        pathL=path
        if os.path.isdir(pathL)==False:
            os.mkdir(pathL)
        if os.path.isdir(pathL+"/"+dirName)==False:
            os.mkdir(pathL+"/"+dirName)
        pathL=pathL+"/"+dirName
        if os.path.isfile(pathL+"/"+dirName+".xml")==True:#If xml file exists
            if metaDictionary[dirName]==False:#And there is no need to re-
download it
                continue
        print("\t entering "+url)
        response=http.request('GET',url)
        if os.path.isfile(pathL+"/"+dirName+".xml"):
            os.remove(pathL+"/"+dirName+".xml")
        if os.path.isfile(pathL+"/"+dirName+"-sanitized.xml"):
            os.remove(pathL+"/"+dirName+"-sanitized.xml")
        file=open(pathL+"/"+dirName+".xml.zip","wb")
        file.write(response.data)
        file.flush()
        file.close()
        response.release_conn()
```

As the name implies, the **`downloadZip`** function downloads the NVD database segments (in .zip form) based on the values in **metaDictionary.** If there was a difference in the metadata files that are locally stored with the ones fetched from the site of NVD, we download that part of the XML database which has changed. When a download is completed, any pre-existing .xml files that are going to be updated, are deleted.

```
def checkMetaData():###allways downloads check why
    global path,metaDictionary
    nvdMeta=open('NVD_DB_METADATA','r')
    http=urllib3.PoolManager()
    for link in nvdMeta:#Maybe thread this and add the META file in order to update proppperly
        url=link.rstrip("\n")
        buf=url
        buf=buf.replace(".meta","")
        bufList=buf.split("/")
        dirName=bufList[len(bufList)-1]
        pathM=path
        if os.path.isdir(pathM)==False:
            os.mkdir(pathM)
        if os.path.isdir(pathM+"/"+dirName)==False:
            os.mkdir(pathM+"/"+dirName)
        pathM=pathM+"/"+dirName+"/"+dirName+".meta"
        if os.path.isfile(pathM)==False:#If path to metadata doesn't exist
            http=urllib3.PoolManager()
            response=http.request('GET',url)
            file=open(pathM,"wb")
            file.write(response.data)
            file.flush()
            file.close()
            metaDictionary[dirName]=True#will let the downloadZip() method download this zip File
        else:
            response=http.request('GET',url)
            file=open("TempMeta","wb")#Temporary fle containing the fetched metadata for the
current nvd xml

            file.write(response.data)

            file.flush()

            file.close()

            file=open("TempMeta","r")

            reading=file.read()

            splitting=reading.split("\n")

            fieldsMax=len(splitting)

            countingHits=0

            checkingData={}

            for item in splitting:

                buf=item.split(":")

                checkingData[buf[0].rstrip(":")]=item.replace(buf[0]+":","")

            file.close()

            fileM=open(pathM)

            Mdata=fileM.read().split("\n")

            exitFlag=True

            for item in Mdata:#Check if the fields of the fetched metadata are identical to the
one fetched from NVD servers (if they aren't then download that xml by setting metaDictionary of
that xml to true)

                exitFlag=True

                for key in checkingData:

                    if item.find(key)!=-1:

                        if item.replace(key+":","")==checkingData[key]:

                            countingHits+=1

                            exitFlag=False

                        else:

                            exitFlag=True

                            break

                if exitFlag==True:

                    break

            fileM.close()

            os.remove("TempMeta")

            if fieldsMax==countingHits:#If all local metadata fields match the remote ones

                metaDictionary[dirName]=False#Do not download zip

            else:

                metaDictionary[dirName]=True#Download zip

                metaLocal=open(pathM,"w")##Update the local Metadata file

                metaLocal.write(reading)

                metaLocal.close()
```

In the **checkMetaData** function, the local metadata files are compared with the ones received from the NVD site and if changes are recorded, the local database is updated(via the **`downloadZip`** function). The text files containing URLs pointing to the remote metadata files (in the NVD site) are accessed here.

```
def unzipAll():
    global path
    if os.path.isdir(path)!=False:
        dirs=os.listdir(path)
        for dir in dirs:
            if os.path.isdir(path+"/"+dir)!=False:
                localCont=os.listdir(path+"/"+dir)
                for file in localCont:
                    if os.path.isfile(path+"/"+dir+"/"+file)!=False and file.endswith(".zip"):
                        zFile=zipfile.ZipFile(path+"/"+dir+"/"+file)
                        zFile.extractall(path+"/"+dir+"/")
                        zFile.close()
                        os.remove(path+"/"+dir+"/"+file)
```

The **unzipAll** function, uncompresses any .zip files that were downloaded from the NVD site and removes the compressed files.

```
def searchDB(fileName):
    report=open(fileName,"r")
    list=report.read().splitlines()
    report.close()
    buf=list[0]
    bufList=buf.split(';')
    reportContent=list.copy()
    list.remove(buf)
    dictOfCpe=dict()
    for item in list:
        seg=item.split(';')
        for b in bufList:
            if b.find("cpe")!=-1:
                if seg[bufList.index(b)]!='':
                    dictOfCpe[seg[bufList.index("host")]
+"_"+seg[bufList.index("port")]]=seg[bufList.index(b)]
                    None
    dictOfVulnerabilities=vulnerabilitySearcher(dictOfCpe,fileName)
    writeReport=open(fileName,"w")#Write report including vulnerabilities
    for item in reportContent:
        if item.find(reportContent[0])!=-1:
            writeReport.write(item+";vulnerabilities\n")
        else:
            line=""
            bufLine=item.split(";")
            numberOfVulnerabilities=0##Variable that keeps number of vulnerabilities in order to
print a message to user at the end
            for key in dictOfVulnerabilities:
                tempBuf=key.split('_')
                ip=tempBuf[0]
                portNum=tempBuf[1]
                if bufLine[bufList.index("host")]==ip and
bufLine[bufList.index("port")]==portNum:#Locate the line with the correct ip address, portNumber
                    line=item+";Found_"+str(dictOfVulnerabilities[key])+"_Vulnerabilities"
                    numberOfVulnerabilities=dictOfVulnerabilities[key]
                    break
                else:
                    line=item+";Found_0_Vulnerabilities"
                    numberOfVulnerabilities=0
            print("Port: "+bufLine[bufList.index("port")]+" Product:
'"+bufLine[bufList.index("product")]+"' Version: '"+bufLine[bufList.index("version")]+"'
Vulnerabilities: "+str(numberOfVulnerabilities))
            writeReport.write(line+"\n")
    writeReport.close()
```

The **searchDB** function, uses our last module, in order to traverse the .xml files and determine if there are any vulnerabilities present (based on the Common Platform Enumeration or CPE for short [14]). It also alters the simple report, by adding a count of the vulnerabilities that have been found.

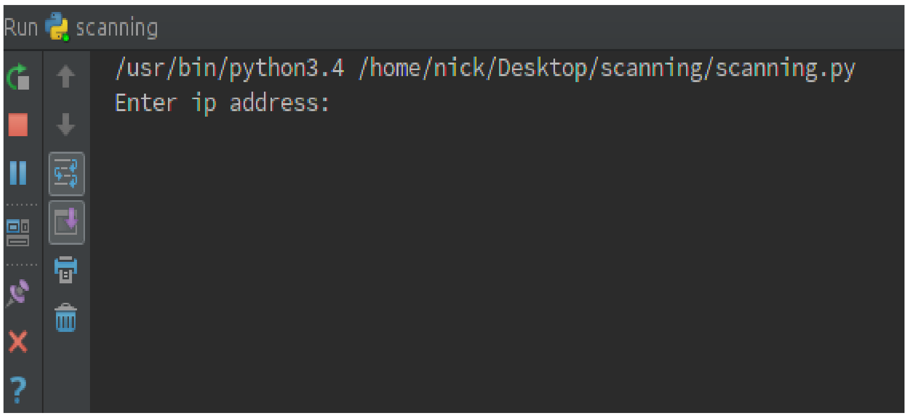
Describing the testingXMLParsing module code

Finally, we have the **testingXMLParsing** module, which accesses the NVD database and reports if the services that were identified by nmap are vulnerable.

The **vulnerabilitySearcherSafeModeWithlxml** handles the searching of all .xml files for the existence of the CPEs reported by nmap, and if there is a match, reports that a vulnerability was located. The function also produces a full report containing all the "hits" from the database, with more detailed information, as to the nature of the vulnerability, the date that it was discovered and if necessary **a list of software** that, combined with the the program that is being scanned, produce this reported vulnerability. In some cases there has to be a subcategory of programs (defined in a number of lists in the XML database) in a machine, in order for a vulnerability to manifest. Our program crosschecks these lists (from the DB) against the identified applications (returned from nmap), and if we have identified applications from all the lists in the database (under the vuln:vulnerable-configuration tag in XML) then, in the Full report we indicate that there's a high probability that the vulnerability is active (field in FULL report: **Likelihood of Vulnerability Existence:** , possible values **High** or **Medium**). If we have identified programs from a subset of the lists (DB) then we say that there is a medium chance that the vulnerability is active. The full code of **vulnerabilitySearcherSafeModeWithlxml** can be found in **Appendix B**.

3.2 Running our program

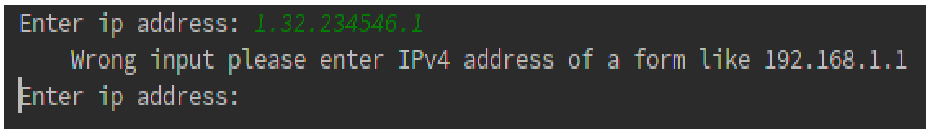
In the following sub-chapter, we will present our program's phases of execution. Initially, the user is prompted to type the IP address of a machine he or she wishes to scan. For the following examples, we used a **Metasploitable** virtual machine running locally in a virtual box as the target machine.



```
/usr/bin/python3.4 /home/nick/Desktop/scanning/scanning.py
Enter ip address:
```

Image 3.1: Initial prompt

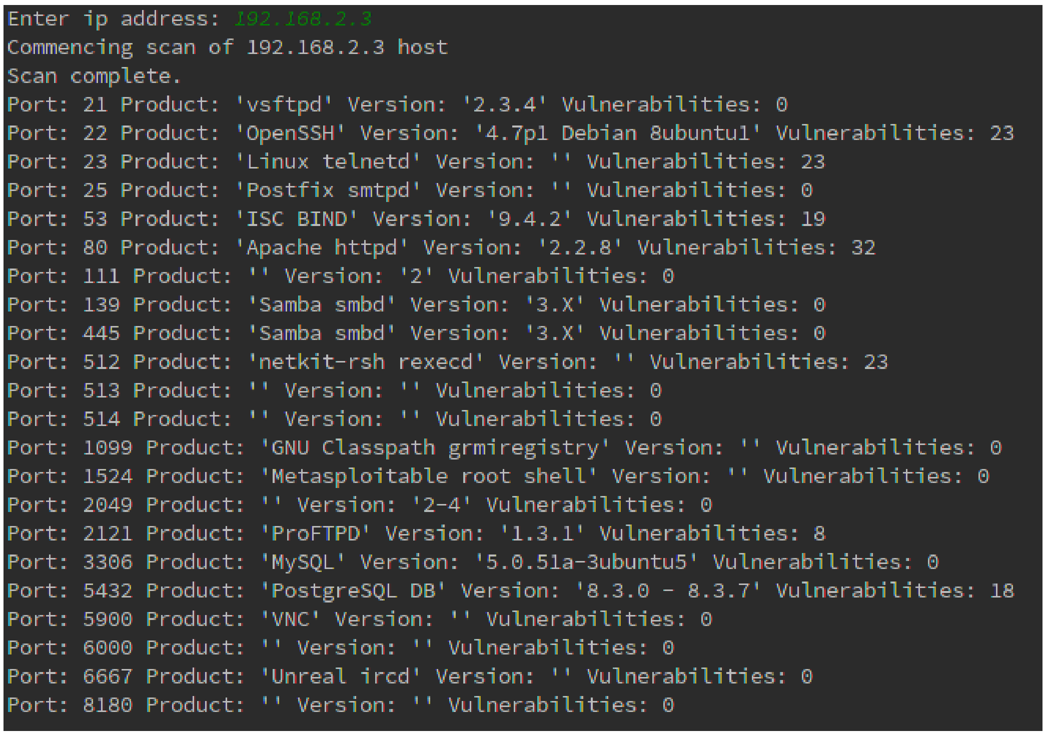
Then, based on the user's input, we have two possibilities. If the IP address is invalid, our program informs the user with this screen:



```
Enter ip address: 1.32.234546.1
    Wrong input please enter IPv4 address of a form like 192.168.1.1
Enter ip address:
```

Image 3.2: Invalid IP Address

Else, a full nmap scann commences, and after that is finished, and the NVD database has been used in the vulnerability search, we have the following display:



```
Enter ip address: 192.168.2.3
Commencing scan of 192.168.2.3 host
Scan complete.
Port: 21 Product: 'vsftpd' Version: '2.3.4' Vulnerabilities: 0
Port: 22 Product: 'OpenSSH' Version: '4.7p1 Debian 8ubuntu1' Vulnerabilities: 23
Port: 23 Product: 'Linux telnetd' Version: '' Vulnerabilities: 23
Port: 25 Product: 'Postfix smtpd' Version: '' Vulnerabilities: 0
Port: 53 Product: 'ISC BIND' Version: '9.4.2' Vulnerabilities: 19
Port: 80 Product: 'Apache httpd' Version: '2.2.8' Vulnerabilities: 32
Port: 111 Product: '' Version: '2' Vulnerabilities: 0
Port: 139 Product: 'Samba smbd' Version: '3.X' Vulnerabilities: 0
Port: 445 Product: 'Samba smbd' Version: '3.X' Vulnerabilities: 0
Port: 512 Product: 'netkit-rsh rexecd' Version: '' Vulnerabilities: 23
Port: 513 Product: '' Version: '' Vulnerabilities: 0
Port: 514 Product: '' Version: '' Vulnerabilities: 0
Port: 1099 Product: 'GNU Classpath grmiregistry' Version: '' Vulnerabilities: 0
Port: 1524 Product: 'Metasploitable root shell' Version: '' Vulnerabilities: 0
Port: 2049 Product: '' Version: '2-4' Vulnerabilities: 0
Port: 2121 Product: 'ProFTPD' Version: '1.3.1' Vulnerabilities: 8
Port: 3306 Product: 'MySQL' Version: '5.0.51a-3ubuntu5' Vulnerabilities: 0
Port: 5432 Product: 'PostgreSQL DB' Version: '8.3.0 - 8.3.7' Vulnerabilities: 18
Port: 5900 Product: 'VNC' Version: '' Vulnerabilities: 0
Port: 6000 Product: '' Version: '' Vulnerabilities: 0
Port: 6667 Product: 'Unreal ircd' Version: '' Vulnerabilities: 0
Port: 8180 Product: '' Version: '' Vulnerabilities: 0
```

Image 3.3: Complete execution results

Finally, the user is able to review the scann results, by viewing the report files located under an easily deduced path of directories, based on the following pattern: [<Profiles Directory> / <IP Address OR Name of machine> / <date of scan> / <time of scan> /].

4. Conclusion and future work

In the context of this thesis, I came across many technologies and techniques regarding the identification of unique characteristics of remote Servers. Generally we noticed that there are a few open source tools a developer can use to implement a scanner similar to our own, although the attempt to build the entire scanner from scratch (port scanner, service recognition and database) is going to be met with some difficulties.

The software that we implemented, fully covers the requirements of this thesis, since it does identify services running on a machine and gives a report on the possible vulnerabilities, although, there is always room for improvement in the future.

For starters, we could add other port scanners, service recognition software and databases containing vulnerabilities, or develop our own scanners, in order to improve the accuracy of our results. Also, the software could be used in order to conduct Internet-wide searches to discover information related to the spread of vulnerabilities, or even the security-update habits of people around the globe, something that can be made possible by adding threading characteristics to our software. Finally, we could also add new functionality to our existing scanner, such as making it automatically perform test in order to verify the identified vulnerabilities, which would greatly improve the accuracy of our results (although such test should always only be conducted with the willing consent of the target machine's owner).

Bibliography

[1] Wikipedia. **Internet** *[website]* URL: https://en.wikipedia.org/wiki/Internet#History
(visited 11/02/2016)

[2] Wikipedia. **Amazon.com** *[website]* URL: https://en.wikipedia.org/wiki/Amazon.com
(visited 11/02/2016)

[3] Signal. **Cyber attacks frequency** *[website]* URL: http://www.afcea.org/content/?q=Article-destructive-cyber-attacks-increase-frequency-sophistication
(visited 11/02/2016)

[4] Hackmagedon. **2015 Cyber attack Statistics** *[website]* URL:
http://www.hackmageddon.com/2016/01/11/2015-cyber-attacks-statistics/
(visited 11/02/2016)

[5] Wikipedia. **Vulnerability (computers)** *[website]* URL:
https://en.wikipedia.org/wiki/Vulnerability_%28computing%29
(visited 13/02/2016)

[6] CVE. **Terminology** *[website]* URL: https://cve.mitre.org/about/terminology.html
(visited 13/02/2016)

[7] Wikipedia. **Heartbleed** *[website]* URL: https://en.wikipedia.org/wiki/Heartbleed
(visited 14/02/2016)

[8] US-CERT. **OpenSSL Heartbleed Vulnerability** *[website]* URL: https://www.us-cert.gov/ncas/alerts/TA14-098A
(visited 14/02/2016)

[9] Wikipedia. **Vulnerability Scanners** *[website]* URL:
https://en.wikipedia.org/wiki/Vulnerability_scanner
(visited 14/02/2016)

[10] Wikipedia. **Nmap** *[website]* URL: https://en.wikipedia.org/wiki/Nmap
(visited 16/02/2016)

[11] Official Nmap site. **Nmap Description** *[website]* URL: https://nmap.org/book/man.html
(visited 16/02/2016)

[12] Official Nmap site. **Service and Application Version Detection** *[website]* URL:
https://nmap.org/book/vscan-technique.html
(visited 16/02/2016)

[13] Wikipedia. **PIP (package manager)** *[website]* URL:
https://en.wikipedia.org/wiki/Pip_(package_manager)
(visited 17/02/2016)

[14] Official NIST site. **Common Platform Enumeration (CPE)** *[website]* URL:
http://scap.nist.gov/specifications/cpe/
(visited 19/02/2016)

[15] Wikipedia. **OpenVAS** *[website]* URL: https://en.wikipedia.org/wiki/OpenVAS
(visited 22/02/2016)

[16] Wikipedia. **Nessuss_(software)** *[website]* URL:
https://en.wikipedia.org/wiki/Nessus_(software)
(visited 22/02/2016)

A. Appendices

Appendix A: The scanning module code

Code of scanning module:

```
import nmap
import socket
import re
import os
import datetime
from vulnerabilityDetector import findVulnerability
profileDir="./profiles"
currentPath=None
def mkProfilesDir():
    if os.path.isdir(profileDir)==False:
        os.mkdir(profileDir)
def mkProfDir(ip):
    global currentPath
    if os.path.isdir(profileDir+"/"+ip)==False:
        os.mkdir(profileDir+"/"+ip)
    currentPath=profileDir+"/"+ip
def fullNmapScan(ip):#Nmap scan
    nmS=nmap.PortScanner()
    nmS.scan(hosts=ip,arguments='-sV')
    return nmS.csv()
def createReport(writeContent):#Method that creates a report
    global currentPath
    if os.path.isdir(currentPath):
        currentPath=currentPath+"/"+str(datetime.date.today().year)
+"_"+str(datetime.date.today().month)+"_"+str(datetime.date.today().day)
        if os.path.isdir(currentPath)==False:
            os.mkdir(currentPath)
        currentPath=currentPath+"/report_"+str(datetime.datetime.now().time())
        if os.path.isdir(currentPath)==False:
            os.mkdir(currentPath)
        currentPath=currentPath+"/report_Simple"
        profile=open(currentPath,'w')
        profile.write(writeContent)
        profile.close()
        return currentPath
def main():
    global currentPath
    while(True):
        ip=input("Enter ip address: ")
        address=None
        try:
            address=socket.gethostbyname(ip)#Try Catch way of checking if ip is valid. IP
returned
            break
        except:
            print("\tWrong input please enter IPv4 address of a form like 192.168.1.1")
    print("Commencing scan of "+ip+" host")
    buffer=fullNmapScan(address)
    print("Scan complete.")
    results=buffer.split("\r\n")
    mkProfilesDir()
    mkProfDir(ip)
    reportPath=createReport(buffer)
    findVulnerability(reportPath)
if __name__ == '__main__':
    main()
```

Appendix B: The testingXMLParsing module code

Code of the testingXMLParsing module:

```
import os

import xml.etree.ElementTree as ET

from lxml import etree

def fixNSProblem():#Initially there was an issue with the xmlns attribute in the xml files that
prevented the programm from executing (and searching these files). This method removes that
parameter

    path="./DB"

    subPath=None

    if os.path.isdir(path):

        subDirs=os.listdir(path)

        for dir in subDirs:

            if os.path.isdir(path+"/"+dir)!=False:

                localCont=os.listdir(path+"/"+dir)

                for file in localCont:

                    sanAlreadyExists=False

                    for buf in localCont:#check if sanitized file exists (the xml file with xmlns
removed that allows the program to execute(had some issues because xmlns's link lead to a 404
page))

                        if buf.endswith("sanitized.xml")!=False:

                            sanAlreadyExists=True

                            break

                    if sanAlreadyExists==True:

                        break

                    if os.path.isfile(path+"/"+dir+"/"+file)!=False and file.endswith(".xml") and
file.endswith("sanitized.xml")==False:#reads all xml files in subDirs of DB (could enter if
condition to make it read only xml files with same file name as parent dir)

                        subPath=path+"/"+dir+"/"+file

                        file=open(subPath,"r")

                        file2=open(subPath.replace(".xml","-sanitized.xml"),"w")

                        for buffer in file:

                            if
buffer.find("xmlns=\"http://scap.nist.gov/schema/feed/vulnerability/2.0\"")!=-1:


buffer=buffer.replace("xmlns=\"http://scap.nist.gov/schema/feed/vulnerability/2.0\"","")

                            file2.write(buffer)

                            file2.flush()

                        file.close()

                        file2.close()

def vulnerabilitySearcherSafeModeWithlxml(dictOfCpe,reportPathName):

    path="./DB"

    subPath=None
    dictOfVuln={}
    dictOfFullReport={}
    ipAddress=""
    for key in dictOfCpe:
        dictOfVuln[key]=0
        ipAddress=key
        dictOfFullReport[(key.split("_"))[1]]=""
    ipAddress=str(ipAddress).split("_")
    fixNSProblem()
    reportOutput=open(reportPathName.replace("_Simple","")+"_Full","w")
    reportOutput.write("Full report for `"+ipAddress[0]+"` host\n")
    if os.path.isdir(path):
        subDirs=os.listdir(path)
        for dir in subDirs:
            if os.path.isdir(path+"/"+dir)!=False:
                localCont=os.listdir(path+"/"+dir)
                for file in localCont:#Traverse the DB subdirectories and access any .xml files
found that contain the word sanitized in their names
                    if os.path.isfile(path+"/"+dir+"/"+file)!=False and file.endswith(".xml") and
file.find("sanitized")!=-1:
                        subPath=path+"/"+dir+"/"+file
                        tree = etree.parse(subPath)
                        root = tree.getroot()
                        for element in root.findall("entry/vuln:vulnerable-software-
list/",root.nsmap):
                            for key in dictOfCpe:
                                bufferText=""
                                if dictOfCpe[key]==element.text:
                                    bufferElement=element
                                    fullReportDict={}
                                    #bufferText+="\nPort: "+(key.split("_"))[1]
                                    while bufferElement.tag!="entry":#Locate entry entity that
encapsulates the found vulnerability
                                        bufferElement=bufferElement.getparent()
                                    fullReportDict["cve_Id"]=bufferElement.attrib["id"]
                                    bufferText+="\n\tCVE ID: "+fullReportDict["cve_Id"]

fullReportDict["published_Date_Time"]=bufferElement.find('./vuln:published-
datetime',root.nsmap).text
                                    bufferText+="\n\tPublished Date time:
"+fullReportDict["published_Date_Time"]

fullReportDict["lastModified_Date_Time"]=bufferElement.find('./vuln:published-
datetime',root.nsmap).text
                                    bufferText+="\n\tLast modified Date time:
"+fullReportDict["lastModified_Date_Time"]
```

```
fullReportDict["cvss_Score"]=bufferElement.find('./vuln:cvss/cvss:base_metrics/cvss:score',root.n
smap).text
                                    bufferText+="\n\tCvss score: "+fullReportDict["cvss_Score"]

fullReportDict["cvss_AccessVector"]=bufferElement.find('./vuln:cvss/cvss:base_metrics/cvss:access
-vector',root.nsmap).text
                                    bufferText+="\n\tCvss accessVector:
"+fullReportDict["cvss_AccessVector"]

fullReportDict["cvss_Access_Complexity"]=bufferElement.find('./vuln:cvss/cvss:base_metrics/cvss:a
ccess-complexity',root.nsmap).text
                                    bufferText+="\n\tCvss access Complexity:
"+fullReportDict["cvss_Access_Complexity"]

fullReportDict["cvss_Authentication_Needed"]=bufferElement.find('./vuln:cvss/cvss:base_metrics/cv
ss:authentication',root.nsmap).text
                                    bufferText+="\n\tCvss # Authentications Needed:
"+fullReportDict["cvss_Authentication_Needed"]

fullReportDict["cvss_Confidentiality_Impact"]=bufferElement.find('./vuln:cvss/cvss:base_metrics/c
vss:confidentiality-impact',root.nsmap).text
                                    bufferText+="\n\tCvss Confidentiality impact:
"+fullReportDict["cvss_Confidentiality_Impact"]

fullReportDict["cvss_Integrity_Impact"]=bufferElement.find('./vuln:cvss/cvss:base_metrics/cvss:in
tegrity-impact',root.nsmap).text
                                    bufferText+="\n\tCvss Integrity impact:
"+fullReportDict["cvss_Integrity_Impact"]

fullReportDict["cvss_Availability_Impact"]=bufferElement.find('./vuln:cvss/cvss:base_metrics/cvss
:availability-impact',root.nsmap).text
                                    bufferText+="\n\tCvss Availability Impact:
"+fullReportDict["cvss_Availability_Impact"]

fullReportDict["cve_Summary"]=bufferElement.find('./vuln:summary',root.nsmap).text
                                    bufferText+="\n\tSummary:
\n\t\t"+fullReportDict["cve_Summary"]
                                    bufferX=bufferElement.findall('./vuln:vulnerable-
configuration',root.nsmap)
                                    for vulnConfigItem in bufferX:#Series of for loops that
traverse the vulnerable configurations tags and produce a likelihood that a specific
vulnerability could be present on the machine we are scanning. Values of likelihood range from
High indicating that we identified programs on the machine, that appear on all the existing
vulnerable configuration lists and Medium if we located software that exist in a subcategory of
all the lists
                                        tempList=[]
                                        garboFlag=False#Variable to indicate the existence of a
cpe in thexml list
                                        likelihoodOfVuln=0#Integer for counting and producing the
likelihood of the vulnerability existence
                                        numberOfVulnSubgroups=0#Integer indicating the number of
subLists that exist in the xml DB under the <vuln:vulnerable-configuration> tag
                                        children=vulnConfigItem.getchildren()#Get children of
current vulnerable configuration
                                        for child in children:
```

```
                                                  if str(child.tag).find("logical-test")!=-1 and
("operator" in child.attrib):
                                                      if child.attrib["operator"].upper()=="AND":
                                                          for GrandChild in child:
                                                              garboFlag=False#Empties the list
                                                              numberOfVulnSubgroups+=1
                                                              flagB=True
                                                              for list in GrandChild:#for loop to fill
the vulnerable configuration list (keeps only the CPEs list that doesn't contain the cpe that is
beeing scanned)
                                                                  tempList.append(list.attrib["name"])
                                                                  if
list.attrib["name"]==dictOfCpe[key]:
                                                                      garboFlag=True
                                                                  if flagB==True:#Limits the program to
identify one CPE per list of vulnerable configuration in XML DB
                                                                      for ke in dictOfCpe:#Check if
current xml DB entry in vulnerable configuration list, has been identified by the scanner and is
not the current CPE for which we are identifying vulnerabilities(excludes the cpe that is being
scanned at each time)
                                                                          if
list.attrib["name"]==dictOfCpe[ke]:
                                                                              likelihoodOfVuln+=1
                                                                              flagB=False
                                                                              break
                                                              if garboFlag==True:
                                                                  tempList.clear()
                                                      elif child.attrib["operator"].upper()=="OR":
                                                          GrandChild=child
                                                          garboFlag=False#Empties the list
                                                          numberOfVulnSubgroups+=1
                                                          flagB=True
                                                          for list in GrandChild:#for loop to fill the
vulnerable configuration list (keeps only the CPEs list that doesn't contain the cpe that is
beeing scanned)
                                                              tempList.append(list.attrib["name"])
                                                              if list.attrib["name"]==dictOfCpe[key]:
                                                                  garboFlag=True
                                                              if flagB==True:#Limits the program to
identify one CPE per list of vulnerable configuration in XML DB
                                                                  for ke in dictOfCpe:#Check if current
xml DB entry in vulnerable configuration list, has been identified by the scanner and is not the
current CPE for which we are identifying vulnerabilities(excludes the cpe that is being scanned
at each time)
                                                                      if
list.attrib["name"]==dictOfCpe[ke]:
                                                                          likelihoodOfVuln+=1
                                                                          flagB=False
                                                                          break
```

```
                                                    if garboFlag==True:

                                                        tempList.clear()

                                            fullReportDict["Vulnerable_Configuration"]=tempList

                                            if likelihoodOfVuln!=0:#if condition, so that we can
display a list for which there are some "hits"

                                                bufferText+="\n\tVulnerable Configurations with this
program:"+str(fullReportDict["Vulnerable_Configuration"])


fullReportDict["Likelihood_Of_Vulnerability_Existence"]=("High" if
likelihoodOfVuln==numberOfVulnSubgroups and likelihoodOfVuln!=0 else ("Medium" if
(likelihoodOfVuln<numberOfVulnSubgroups and likelihoodOfVuln>0) else ("Low"
if(likelihoodOfVuln<numberOfVulnSubgroups and likelihoodOfVuln==0) else "Error")))

                                                bufferText+="\n\tLikelihood of Vulnerability
Existence: "+fullReportDict["Likelihood_Of_Vulnerability_Existence"]+"\n"

                                        bufferText+="\n"+"~"*len(fullReportDict["cve_Summary"])+"\n"

                                        dictOfFullReport[(key.split("_"))[1]]+=bufferText

                                        dictOfVuln[key]=dictOfVuln[key]+1

    for key in dictOfFullReport:

        if dictOfFullReport[key]!="":

            reportOutput.write("\nPort: "+(key)+dictOfFullReport[key])

    reportOutput.close()

    return dictOfVuln
```

Appendix C: The vulnerabilityDetector module code

Code of the vulnerabilityDetector module:

```
import zipfile
import urllib3
import os
from testingXMLParsing import vulnerabilitySearcherSafeModeWithlxml
url='https://nvd.nist.gov/feeds/xml/cve/nvdcve-2.0-2016.xml.zip'
path="./DB"
metaDictionary={}
def downloadZip():
    global url,path
    checkMetaData()
    nvdLinks=open('NVD_DBLinks','r')
    http=urllib3.PoolManager()
    for link in nvdLinks:#Maybe thread this and add the META file in order to update propperly
        url=link.rstrip("\n")
        buf=url
        buf=buf.replace(".xml.zip","")
        bufList=buf.split("/")
        dirName=bufList[len(bufList)-1]
        pathL=path
        if os.path.isdir(pathL)==False:
            os.mkdir(pathL)
        if os.path.isdir(pathL+"/"+dirName)==False:
            os.mkdir(pathL+"/"+dirName)
        pathL=pathL+"/"+dirName
        if os.path.isfile(pathL+"/"+dirName+".xml")==True:#If xml file exists
            if metaDictionary[dirName]==False:#And there is no need to re-download it
                continue
        print("\t entering "+url)
        response=http.request('GET',url)
        if os.path.isfile(pathL+"/"+dirName+".xml"):
            os.remove(pathL+"/"+dirName+".xml")
        if os.path.isfile(pathL+"/"+dirName+"-sanitized.xml"):
            os.remove(pathL+"/"+dirName+"-sanitized.xml")
        file=open(pathL+"/"+dirName+".xml.zip","wb")
        file.write(response.data)
        file.flush()
        file.close()
        response.release_conn()
def checkMetaData():###allways downloads check why
    global path,metaDictionary
    nvdMeta=open('NVD_DB_METADATA','r')
    http=urllib3.PoolManager()
    for link in nvdMeta:#Maybe thread this and add the META file in order to update propperly
        url=link.rstrip("\n")
        buf=url
        buf=buf.replace(".meta","")
        bufList=buf.split("/")
        dirName=bufList[len(bufList)-1]
        pathM=path
        if os.path.isdir(pathM)==False:
            os.mkdir(pathM)
        if os.path.isdir(pathM+"/"+dirName)==False:
            os.mkdir(pathM+"/"+dirName)
        pathM=pathM+"/"+dirName+"/"+dirName+".meta"
        if os.path.isfile(pathM)==False:#If path to metadata doesn't exist
            http=urllib3.PoolManager()
            response=http.request('GET',url)
            file=open(pathM,"wb")
            file.write(response.data)
            file.flush()
            file.close()
            metaDictionary[dirName]=True#will let the downloadZip() method download this zip File
        else:
            response=http.request('GET',url)
            file=open("TempMeta","wb")#Temporary fle containing the fetched metadata for the
current nvd xml
            file.write(response.data)
            file.flush()
            file.close()
            file=open("TempMeta","r")
            reading=file.read()
            splitting=reading.split("\n")
            fieldsMax=len(splitting)
            countingHits=0
            checkingData={}
            for item in splitting:
                buf=item.split(":")
                checkingData[buf[0].rstrip(":")]=item.replace(buf[0]+":","")
            file.close()
            fileM=open(pathM)
            Mdata=fileM.read().split("\n")
            exitFlag=True
            for item in Mdata:#Check if the fields of the fetched metadata are identical to the
one fetched from NVD servers (if they aren't then download that xml by setting metaDictionary of
that xml to true)
                exitFlag=True
                for key in checkingData:
                    if item.find(key)!=-1:
                        if item.replace(key+":","")==checkingData[key]:
                            countingHits+=1
                            exitFlag=False
                        else:
                            exitFlag=True
                            break
                if exitFlag==True:
                    break
            fileM.close()
            os.remove("TempMeta")
            if fieldsMax==countingHits:#If all local metadata fields match the remote ones
                metaDictionary[dirName]=False#Do not download zip
            else:
                metaDictionary[dirName]=True#Download zip
                metaLocal=open(pathM,"w")##Update the local Metadata file
                metaLocal.write(reading)
                metaLocal.close()
def unzipAll():
    global path
    if os.path.isdir(path)!=False:
        dirs=os.listdir(path)
        for dir in dirs:
            if os.path.isdir(path+"/"+dir)!=False:
                localCont=os.listdir(path+"/"+dir)
                for file in localCont:
                    if os.path.isfile(path+"/"+dir+"/"+file)!=False and file.endswith(".zip"):
                        zFile=zipfile.ZipFile(path+"/"+dir+"/"+file)
                        zFile.extractall(path+"/"+dir+"/")
                        zFile.close()
                        os.remove(path+"/"+dir+"/"+file)
def searchDB(fileName):
    report=open(fileName,"r")
    list=report.read().splitlines()
    report.close()
    buf=list[0]

    bufList=buf.split(';')

    reportContent=list.copy()

    list.remove(buf)

    dictOfCpe=dict()

    for item in list:

        seg=item.split(';')

        for b in bufList:

            if b.find("cpe")!=-1:

                if seg[bufList.index(b)]!='':

                    dictOfCpe[seg[bufList.index("host")]
+"_"+seg[bufList.index("port")]]=seg[bufList.index(b)]

                    None

    dictOfVulnerabilities=vulnerabilitySearcher(dictOfCpe,fileName)

    writeReport=open(fileName,"w")#Write report including vulnerabilities

    for item in reportContent:

        if item.find(reportContent[0])!=-1:

            writeReport.write(item+";vulnerabilities\n")

        else:

            line=""

            bufLine=item.split(";")

            numberOfVulnerabilities=0##Variable that keeps number of vulnerabilities in order to
print a message to user at the end

            for key in dictOfVulnerabilities:

                tempBuf=key.split('_')

                ip=tempBuf[0]

                portNum=tempBuf[1]

                if bufLine[bufList.index("host")]==ip and
bufLine[bufList.index("port")]==portNum:#Locate the line with the correct ip address, portNumber

                    line=item+";Found_"+str(dictOfVulnerabilities[key])+"_Vulnerabilities"

                    numberOfVulnerabilities=dictOfVulnerabilities[key]

                    break

                else:

                    line=item+";Found_0_Vulnerabilities"

                    numberOfVulnerabilities=0

            print("Port: "+bufLine[bufList.index("port")]+" Product:
'"+bufLine[bufList.index("product")]+"' Version: '"+bufLine[bufList.index("version")]+"'
Vulnerabilities: "+str(numberOfVulnerabilities))

            writeReport.write(line+"\n")

    writeReport.close()

    None

def vulnerabilitySearcher(dictOfCpe,reportPathName):#Access the XML Databases and find any
vulnerabilities

    return vulnerabilitySearcherSafeModeWithlxml(dictOfCpe,reportPathName)
    None
def updateDB():
    downloadZip()
    unzipAll()
def findVulnerability(reportName):
    updateDB()
    searchDB(reportName)
if __name__=="__main__":#Can be used to update the DB (NVD XMLs)
    downloadZip()
    unzipAll()
```