

Final Project Report

Home Network Security Analyzer

C00250955 – Matthew Kavanagh

Table of Contents

Table of Contents.....	2
Introduction.....	3
Project Description.....	4
Conformance to Specification.....	9
Learning.....	10
Technical Learning:.....	10
Personal Learning:.....	12
Review.....	13
Acknowledgments.....	14
References.....	14

Introduction

The purpose of this document is to provide a final and conclusive report on my chosen Final Year Project, the Home Network Security Analyzer. This document contains the following sections:

- **Project Description:** A detailed overview of the projects purpose and its final design.
- **Conformance to Specification:** How closely the project follows its original specification and a defense of any deviations to this specification that were made.
- **Learning:** The technical and personal learning and growth I achieved over the course of the project.
- **Review:** An overview of the strengths and weaknesses of the finished project, any unfinished work, changes I would make if I had the time, advice to others, etc.

Project Description

The Home Network Security Analyzer is Python application that runs on a host in a network and provides a local web application to scan the network to find the connected devices, determine the running software and check for vulnerabilities that the software may have. The program uses Nmap(provided by the python3-nmap library) for the scanning functionality, the National Institute of Standards and Technology(NIST) National Vulnerability Database(NVD) API(accessed using the NVDLib python library) to check for vulnerabilities and the Flask framework to provide the web interface. The web interface is designed to be simple and uncluttered.

The first page the user is presented with allows for inputting a target for a Nmap ping scan, by default it gets the local network address to find all devices connected.(Figure 1)

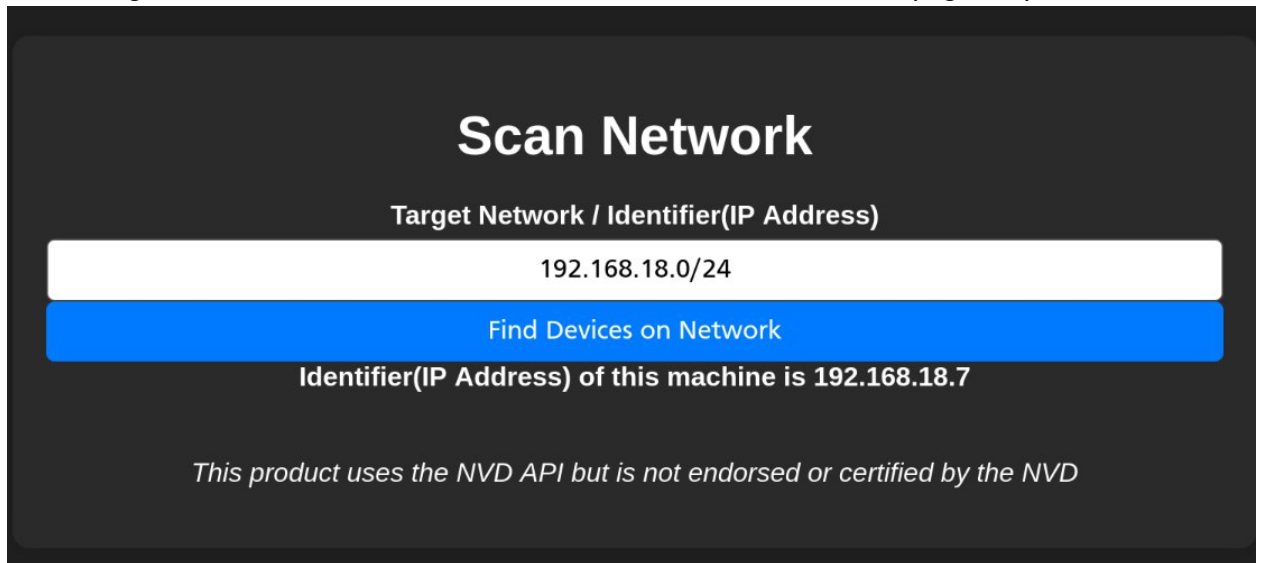


Figure 1 - Web App Main Page

After the scan is performed a page showing all addresses determined to be up is presented, the user can choose which of these to perform an OS scan on to find further details about their

systems and configurations(Figure 2).

Devices Found on 192.168.18.0/24

Select devices to gather more information on

- 192.168.18.1
- 192.168.18.2
- 192.168.18.4
- 192.168.18.6
- 192.168.18.8
- 192.168.18.9
- 192.168.18.11
- 192.168.18.7

[Start OS Scan](#)

[Return to Start](#)

Figure 2 - Devices Found after Ping Scan

When the OS Scan is completed the user is brought to a page detailing the detected configurations of the targets. The user can then choose from the results to check the NVD for

any associated vulnerabilities(Figure 3).

Device Info

Select Devices to Check for Vulnerabilities

- Possible Operating Systems for 192.168.18.4:**
 1. VxWorks - Accuracy:99 - CPE:cpe:/o:windriver:vxworks
 2. Keyence CV-X150F Image Sensor/Controller (VxWorks) - Accuracy:97 - CPE:cpe:/o:windriver:vxworks
 3. VxWorks: HP printer or Vocality BASICS Four Wire VoIP gateway - Accuracy:97 - CPE:cpe:/o:windriver:vxworks
 4. HP LaserJet M2727nf or P1505n printer - Accuracy:96 - CPE:No CPE Found
 5. HP printer (M1120n, M1522n, CP1515n, CP2025dn, or CP2525dn) - Accuracy:96 - CPE:No CPE Found
 6. NEC Univerge SV8300 PBX - Accuracy:96 - CPE:cpe:/h:nec:univerge_sv8300
 7. Blackboard transaction system serial-to-IP converter - Accuracy:96 - CPE:No CPE Found
 8. HP LaserJet CP2025dn printer - Accuracy:96 - CPE:cpe:/h:hp:laserjet_cp2025dn
 9. HP LaserJet M451dn, CM1415fnw, or CP4525 - Accuracy:96 - CPE:cpe:/h:hp:laserjet_m451dn
 10. HP iLO 4 remote management interface - Accuracy:96 - CPE:cpe:/o:hp:ilo:4

Check for Vulnerabilities

Return to Start

Figure 3 - Potential Operating Systems found after OS Scan

Finally the users is given a list of Common Vulnerabilities and Exposures(CVE) associated with the software's CPE along with details and links to recommended mitigations(Figure 4).

Vulnerabilities for the following devices:

If you are concerned about any of the findings show this page to a security expert

192.168.18.4

cpe:2.3:o:windriver:vxworks

ID :

CVE-2008-2476

Severity :

HIGH

Description :

The IPv6 Neighbor Discovery Protocol (NDP) implementation in (1) FreeBSD 6.3 through 7.1, (2) OpenBSD 4.2 and 4.3, (3) NetBSD, (4) Force10 FTOS before E7.7.1.1, (5) Juniper JUNOS, and (6) Wind River VxWorks 5.x through 6.4 does not validate the origin of Neighbor Discovery messages, which allows remote attackers to cause a denial of service (loss of connectivity) or read private network traffic via a spoofed message that modifies the Forward Information Base (FIB).

Stauts :

Modified

NVD Url :

<https://nvd.nist.gov/vuln/detail/CVE-2008-2476>

References to Advisories, Solutions, and Tools :

<ftp://ftp.netbsd.org/pub/NetBSD/security/advisories/NetBSD-SA2008-013.txt.asc>

<http://secunia.com/advisories/32112>

<http://secunia.com/advisories/32116>

<http://secunia.com/advisories/32117>

<http://secunia.com/advisories/32133>

<http://secunia.com/advisories/32406>

<http://security.freebsd.org/advisories/FreeBSD-SA-08:10.nd6.asc>

<http://securitytracker.com/id?1020968>

<http://support.apple.com/kb/HT3467>

<http://www.kb.cert.org/vuls/id/472363>

<http://www.kb.cert.org/vuls/id/MAPG-7H2RY7>

<http://www.kb.cert.org/vuls/id/MAPG-7H2S68>

http://www.openbsd.org/errata42.html#015_ndp

http://www.openbsd.org/errata43.html#006_ndp

<http://www.securityfocus.com/bid/31529>

<http://www.securitytracker.com/id?1021109>

<http://www.securitytracker.com/id?1021132>

<http://www.vupen.com/english/advisories/2008/2750>

<http://www.vupen.com/english/advisories/2008/2751>

<http://www.vupen.com/english/advisories/2008/2752>

<http://www.vupen.com/english/advisories/2009/0633>

<https://exchange.xforce.ibmcloud.com/vulnerabilities/45601>

<https://oval.cisecurity.org/repository/search/definition/oval%3Aorg.mitre.oval%3Adef%3A5670>

<https://www.juniper.net/alerts/viewalert.jsp?>

[actionBtn=Search&txtAlertNumber=PSN-2008-09-036&viewMode=view](https://www.juniper.net/alerts/viewalert.jsp?actionBtn=Search&txtAlertNumber=PSN-2008-09-036&viewMode=view)

ID :

CVE-2010-2965

Severity :

HIGH

Description :

The WDB target agent debug service in Wind River VxWorks 6.x, 5.x, and earlier, as used on the Rockwell Automation 1756-ENBT series A with firmware 3.2.6 and 3.6.1 and other products, allows remote attackers to read or modify arbitrary memory locations, perform function calls, or manage tasks via requests to UDP port 17185, a related issue to CVE-2005-3804.

Stauts :

Analyzed

NVD Url :

<https://nvd.nist.gov/vuln/detail/CVE-2010-2965>

References to Advisories, Solutions, and Tools :

<http://blog.metasploit.com/2010/08/vxworks-vulnerabilities.html>

Figure 4 - List of CVEs returned by NVD

The program provides logging of Nmap and NVD results along with the final report in simple text files to be viewed at a later date(Figure 5).

```
2024-04-16 15:23:43.491169 - Results of Home Network Security Analyzer -  
---192.168.18.1---  
CPE: cpe:2.3:o:linux:linux_kernel:4.1  
CVE: {'ID': 'CVE-1999-0656', 'Severity': 'MEDIUM', 'Description': 'The ugid  
CVE-1999-0656', 'References to Advisories, Solutions, and Tools': ['http://c  
  
CVE: {'ID': 'CVE-2003-1327', 'Severity': 'HIGH', 'Description': 'Buffer over  
file with a long pathname, which triggers the overflow when wu-ftpd construc  
bugtraq/2003-09/0348.html', 'http://secunia.com/advisories/9835', 'http://se  
exchange.xforce.ibmcloud.com/vulnerabilities/13269'}}  
  
CVE: {'ID': 'CVE-2003-1332', 'Severity': 'HIGH', 'Description': 'Stack-based  
'https://nvd.nist.gov/vuln/detail/CVE-2003-1332', 'References to Advisories,  
  
CVE: {'ID': 'CVE-2003-1372', 'Severity': 'MEDIUM', 'Description': 'Cross-sit  
'Modified', 'NVD Url': 'https://nvd.nist.gov/vuln/detail/CVE-2003-1372', 'Re  
'https://exchange.xforce.ibmcloud.com/vulnerabilities/11376'}}
```

Figure 5 - Excerpt from Logs

Conformance to Specification

The initial specification of my project is for a program with the following core features:

- Scan a network to discover the connected devices.
- Determine the security on the devices.
- Generate a report on the programs findings.

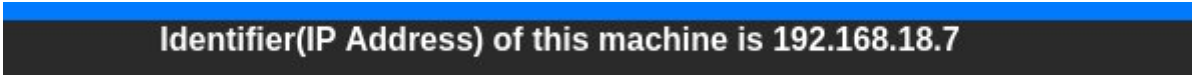
And the following optional features:

- Options to analyze all devices on a network or a specific device.
- Give recommendations on changes to make to improve network security.

The specification calls for these to be implemented if possible in a simple web based interface and be easy to use while avoiding jargon and confusing language to make it as accessible as reasonably possible.

Throughout the work of the project I've tried to following this specification as closely as possible. As seen from the screenshots(Figures 1-5 inclusive) the program is capable of all of the outlined features, it can scan networks and single devices, it uses the CPEs found in the scan to check for vulnerabilities on devices, the final page gives a condensed report along with links to possible mitigations and other resources.

I have also tried to avoid adding too much technical information that would confuse a user, e.g. the use of the term identifier instead of IP address(Figure 6).



Identifier(IP Address) of this machine is 192.168.18.7

Figure 6 - Message from Main Page

Learning

The final year project has been an excellent opportunity for hands on learning with the experience of designing a program from scratch over a period of time. I can split my learning experience between two categories: Technical and Personal.

Technical Learning:

While I was previously familiar to Python and Flask my work on the Home Network Security Analyzer has helped to hone my existing skills and introduce me to new technologies and techniques.

The first of which is the Jupyter Notebook format, a shareable document that, among other things, can contain code that is executed in a self contained environment called a cell. Due to this, the format excels at sandboxing and testing code which I made ample use of early in the project for testing concepts and prototyping the functions used with the Nmap and NVD before they were used in the project proper(Figure 7).

```

#remove empty results and empty fields
def removeEmptyResult(result):
    for key in list(result.keys()):
        if key.find('.') != -1:
            if "state" in result[key]:
                if result[key]["state"] == "down":
                    result.pop(key)
                    continue
            for subkey in list(result[key].keys()):
                if ((result[key][subkey] is None) or (len(result[key][subkey]) == 0)):
                    result[key].pop(subkey)

#parse ping results
def parsePing(result):
    parsedResult = {}
    for key in list(result.keys()):
        if key.find('.') != -1:
            host = {}
            host["address"] = key
            if "hostname" in result[key]:
                host["hostname"] = result[key]["hostname"][0]["name"]
            host["state"] = result[key]["state"]
            parsedResult[key] = host
    return parsedResult

def parseVersion(result):
    parsedResult = {}
    entries = []
    for key in result:
        if '.' in key:
            host = key
            for subkey in result[key]["ports"]:
                if "cpe" in subkey:
                    for entry in subkey["cpe"]:
                        if not entry["cpe"] == "cpe:/o:linux:linux_kernel":
                            entries.append(entry["cpe"].replace("/", "2.3:"))
            parsedResult[host] = entries
    return parsedResult

```

✓ 0.0s

```

#regular ping scan
import nmap3, json
nmap = nmap3.NmapScanTechniques()
target = "192.168.18.0/24"
result = nmap.nmap_ping_scan(target)
removeEmptyResult(result)
result = parsePing(result)
result = json.dumps(result, indent=4)
print(result)

```

[2] ✓ 6.8s

```

... {
    "192.168.18.1": {
        "address": "192.168.18.1",
        "state": "up"
    },
    "192.168.18.2": {
        "address": "192.168.18.2",
        "state": "up"
    }
}

```

Figure 7 - Excerpts from Jupyter Notebook used to test Nmap

Another format I have come into contact with is JSON, a data format. Due to the results of the Nmap library being returned in JSON format I've had to write parsing code so the results can be displayed to the user in a simple clean way. As JSON's data types are also present in Python—numbers, strings, booleans, arrays and objects (key-pair values like Python's dictionaries)—the functions to parse the results are relatively simple (Figure 8).

```
#parse ping results, returns dict of target : dict of address,hostname and state
def parsePing(result):
    parsedResult = {}
    for key in list(result.keys()):
        if '.' in key: #check if entry is a host ( e.g: ip address x.x.x.x or domain name x.com)
            host = {}
            host["address"] = key
            if "hostname" in result[key]: #check for hostnames and include them if present
                host["hostname"] = result[key]["hostname"][0]["name"]
            host["state"] = result[key]["state"]["state"] #state of host (up or down)
            parsedResult[key] = host
    return parsedResult
```

Figure 8 - Code to parse JSON result of a Nmap ping scan

While I have used the Flask framework in the past, I only had a surface level understanding of it but I have been able to learn more about the framework during the course of creating the web interface for the program, thanks in large part to the “The Flask Mega-Tutorial” series by Miguel Grinberg on his blog (Miguel Grinberg, 2024)

In checking for vulnerabilities I have come to rely on the National Vulnerabilities Database (NVD) and learned how to use its API via the NVDLib Python library.

Personal Learning:

Overall the project has been a deeply personal learning experience, I am of the opinion that I have grown as a person.

An area of personal improvement I've noticed is in timekeeping and staying in schedule. Due to the large size of the project, the time frame to complete it and the work needed to be done for other subjects at the same time it was essential for me to improve my skills in this area to get work done on time and in an acceptable state.

I have also learned to how many problems can arise when carrying out the development of a piece of software like this and the best practices for examining and dealing with issues and bugs as they arise while avoiding overly stressing myself to the point of resulting in less work being done.

Review

In review I am satisfied with the result of my project and the work I have achieved with just a few small regrets. I am confident that the technologies chosen are ideal for my project with the libraries being fully functional and the Python language making it easy to integrate them together into one fully featured program.

As show in the screenshots in the Project Description section the program works in it's intended purpose as a simple interface that works fully from start to finish in discovering devices and checking for any known vulnerabilities.

One slight hitch in the beginning of the project was the changing of the interface for the application, while I had originally planned to implement it as a desktop application with a GUI using a library such as Tkinter or PyQt I made the decision to switch to a web interface using Flask as I had experience with it, whereas I would be learning Tkinter/PyQT from scratch. On top of this the features and capabilities of Flask where more than enough for my purposes.

A much larger issue present in the final product is the problem of speed. The NVD API has a rate limit in place, 5 requests in a 30 second span, this can be raised with the use of an API key provided on request by NIST.

A more difficult speed issue to tackle is the scanning of multiple targets, as even when given a list of targets Nmap scans targets sequentially causing problems in my program if a large group of targets(e.g. a entire network) are chosen to be scanned by a slower scan such as the OS scan. If I had more time or was starting over I would attempt to remedy this issue with the use of multithreading to scan multiple targets simultaneously, I did not attempt this as the issue was found late into development, would require rewriting large sections of the program and I have no previous experience with Python's multithreading library Threading.

If I were advising someone in the creation of a similar program I would recommend to look into available libraries, choose what suits their needs best, and then create and test the functions and code using that library that is to be used with the main program on it's own, similar to how I used Jupyter to test my code for Nmap and NVD before inserting it into the main program. This has the benefit of giving you a deeper understanding of the libraries and can help find any issues you may have with them early on in development.

Acknowledgments

While I have written this report from a very personal perspective about my work and achievements I must clarify that there are many people and groups whose involvement, both directly and indirectly, allowed to me achieve this undertaking and without which I would have achieved only a fraction of the work before you.

I would like to extend my deepest thanks to the South East Technological University who allowed me to carry out this project in the first place, my project supervisor Paul Barry whose advice and supervision was instrumental to the project, Miguel Grinberg for his Flask tutorials which I used for reference when designing the web interface, the developers and maintainers of Flask, Nmap, NVD, Python, Jupyter and all other technologies that were used during the course of my work, my friends and family who supported me the entire time and my dog Lenny who provided moral support.

References

Miguel Grinberg(2024) The Flask Mega-Tutorial. Available at <https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world> (Accessed 17th February 2024)