NMAM Python Script

Network Mapping and Monitoring (NMAM) Python Script

Paul Michael Oates

CMP320: Advanced Ethical Hacking

2022/23

Note that Information contained in this document is for educational purposes.

.

Abstract

In an age of increasing network complexity, network mapping allows administrators to understand what assets are on their networks. Data usage monitoring is also equally important to understand the traffic flows and can help network administrators to identify attacks. The Network Mapping and Monitoring (NMAM) is a Python script that aims to incorporate network mapping with data usage monitoring. The aims of this project are to research current network mapping tools and utilise these from the python script. The required development libraries were installed to develop the Network Mapping and Monitoring script. This was tested in a simulated network environment and evaluated and compared to other popular network tools.

The waterfall methodology was selected as it follows a clear structure and is suited to small projects to develop a Network scanning and mapping script. Through Nmap, Networkx, and Matplotlib the network map was developed to give the end user a clear visual diagram of the network including operating systems and open ports. A data usage monitor and alert system were developed by implementing Pyshark and Twilio. This allows the user to set a data threshold and alerts the user when this data usage is exceeded. To allow the script to be called from the command line interface the Argparse library was implemented. When comparing to paid alternatives the tools advantages include increased customisation, open source code, and better integration with other network tools.

The script was tested on an emulated Linux network and a home network and successfully identified all devices present. The data usage monitor was also tested using the ping command and the user was alerted once the data threshold was exceeded. The script enables effective network security by using Nmap's OS scan switch which is then displayed via the Networkx and Matplotlib libraries. This allows the user to quickly identify any outdated operating system versions on the network and begin to mitigate any risk posed by them. The script could be further improved by faster network scanning and improved OS detection. By uploading this to GitHub, this enables network administrators to use and improve this tool for free.

.

Contents

.

1 INTRODUCTION

1.1 BACKGROUND

Network mapping is vitally important for business of all sizes. It allows network administrators to understand what assets are on their networks, bottlenecks, and security flaws that may occur. Equally important is to understand the traffic that flows across a network to allow for maximum efficiency. Network usage monitoring can enable network administrators to identify attacks such as Denial of Service and resolve these before they become a serious issue. However, as networks become increasingly complex and support larger throughput the networking tools in place are quickly becoming outdated and not fit for purpose.

“Over the next decade, the complexity and load placed upon enterprise networks will rapidly outpace our ability to manage them effectively”. (Cisco.com,2021)

In the future new tools will be required to help assist with our understanding of networks.

“A survey of over 600 enterprise networks found an average of 4,400 monthly events impacting wireless networks.” (Cisco.com, 2021)

This suggests there is many issues which are having a detrimental effect on networks which in turn negatively impact businesses.

Administrators use tools such as Nmap to better understand a network topology. Nmap is a port scanning tool which allows administrators to identify assets on a network including operating systems and open ports. A visual layout of a network topology would also be helpful to an administrator to understand what assets are present. However, these tools do not go far enough to protect the network as no monitoring occurs in real time leaving it open to attacks and exploits.

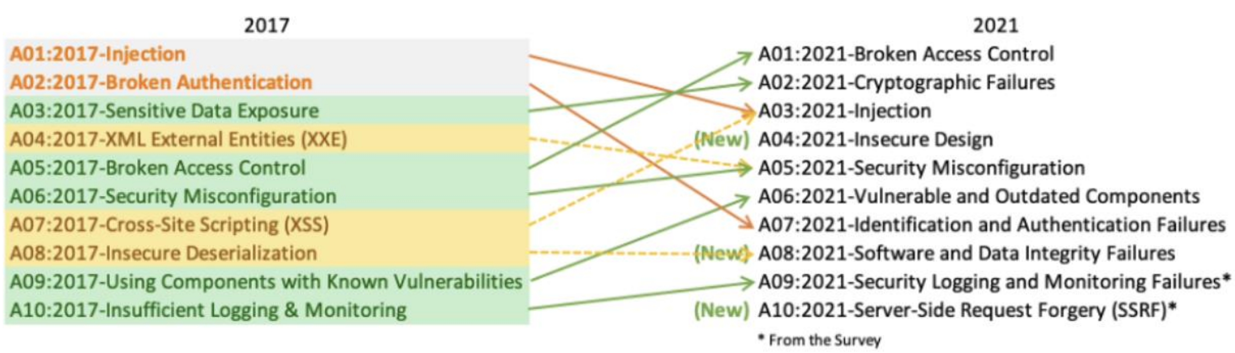


Figure 1-1 OWASP Top 10

Figure 1-1 shows the top ten vulnerabilities in 2017 and 2021.

Vulnerabilities such as insecure design, security misconfiguration, vulnerable and outdated components, and security logging and monitoring failures identified in Figure 1-1 could be mitigated by applying a system which maps and continually monitors a network in real time.

A scripting language such as Python could be used to incorporate the Nmap tool with mapping tools and messaging tools such as Twilio to alert the user of a network usage data limit breach in real time. Python scripting can also be easily developed and adapted to meet the unique needs of a business.

1.2 Aim

Therefore, the need for a tool to map and continually monitor the network clearly exists. As network complexity becomes increasingly sophisticated the effective management of these will become increasingly difficult unless new tools are developed.

This report aims to:

- Research current network mapping tools
- Install the required development libraries and software
- Develop a network mapping and real time monitoring tool
- Test this tool in a simulated network environment using GNS3 a network emulation tool
- Evaluate and compare this tool to other network tools

2 PROCEDURE

2.1 OVERVIEW OF PROCEDURE

The waterfall development method (codecademy.com, 2023) was identified as the chosen methodology as it follows a clear structure, is suited to small projects, and meets the requirements of the project. The waterfall methodology has the following phases: Requirements, Design, Implementation, Testing, and Maintenance.

For the purposes of this project the phases of the waterfall methodology are

- Requirements - Research and identify tools and libraries to use for the development of the script
- Design - Design a flow chart to identify the system architecture
- Implementation- Implement the design
- Testing - Test the final product
- Maintenance - Implement feedback

The app was called NMAM short for Network Mapping and Monitoring and will be known as this from this point onwards. To develop NMAM a Windows 11 operating system, as well as an Ubuntu 64-bit virtual machine was setup. This was to ensure that the program was cross compatible on Linux and Windows operating systems. The following development tools were utilised:

- GNS3
- VSCode
- Nmap
- Python 3.8

Further python libraries made use of include

- Nmap
- Scapy
- Matplotlib
- Networkx
- Twilio
- Pyshark

A full list of tools used and where they can be downloaded can be found in Appendix A – Tools Utilised.

2.2 REQUIREMENTS

Initially the requirements of the program were identified. The end user and functional requirements were as follows:

End User Requirements

- Command line User Interface
- The ability to scan different IP Subnets and classes
- The ability to run a Nmap scan without the network usage data limit monitor
- The ability to map the network and save an ".png" image
- The ability to set the data limit threshold per device for the subnet

Functional Requirements

- Ability to pass command line arguments to the program
- A Nmap scan of the network using the Nmap python library
- A Pyshark scan identifying data usage of devices on the network in real time
- A Twilio function is used to send a text message to the user if the data usage threshold on a single device is exceeded
- Scapy to identify devices to be scanned on the network
- Matplotlib and Networkx libraries to plot and generate the network map image

Research

Research into tools and techniques was undertaken, this identified Pyshark, a python wrapper for Tshark, allowing Python packet parsing using Wireshark dissectors, to enable live data usage monitoring. Research into the Scapy library identified it as a packet manipulation tool for network discovery. The Nmap tool is a network scanner used to discover hosts and services on a network. Using its Python library, it will be used within this project to retrieve asset information from the network. Matplotlib and Networkx libraries were used for the network map display as this makes it possible to plot and display devices on the network. Finally, from my research the Twilio library was identified as the most effective a way to send messages to the end user.

2.3 Design

- By making use of the design requirements a flow chart was drawn up to identify the software flow that NMAM will follow.

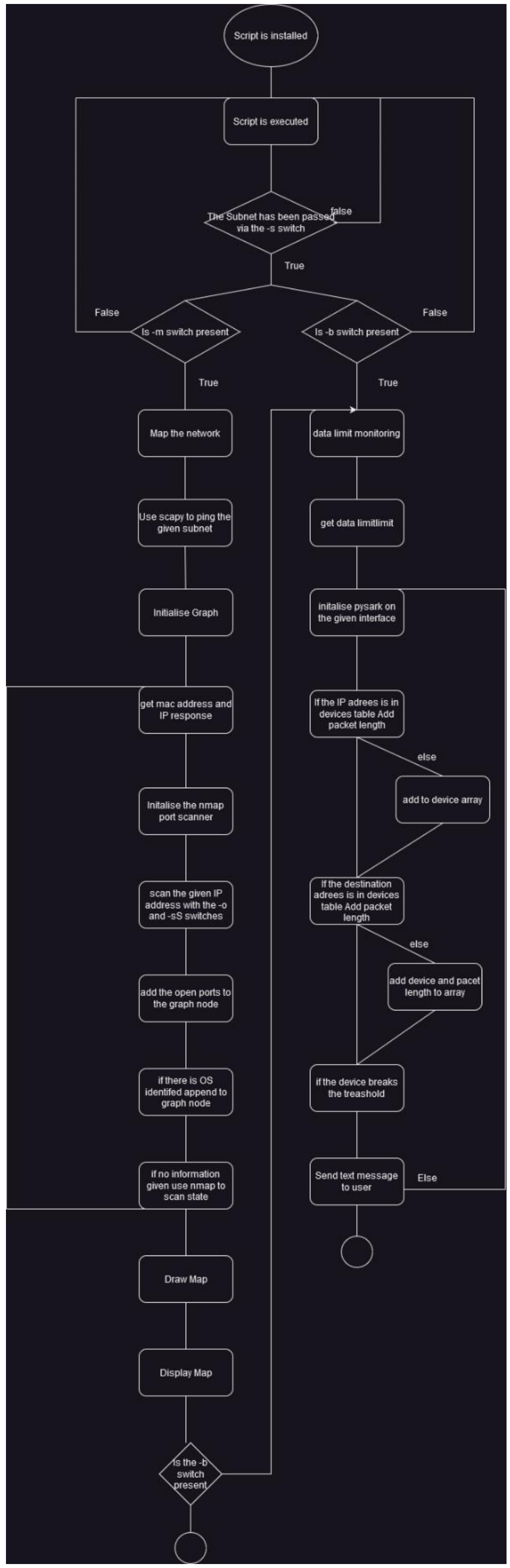


Figure 2-1 Flowchart

Figure 2-1 shows the tool includes the following four switches.

- -m maps the network
    - This fetches the IP and MAC addresses in the given IP range. It then proceeds to utilise Nmap to run an OS scan to identify the operating system that the device is running. It also identifies the ports that are open. If there is no operating system present it proceeds to check if the device is up. The program will proceed to display an image of the device's ports and the operating system.
- -b data usage monitoring
    - This will initially start a Wireshark capture using the Pyshark library, it then proceeds to loop round and create a device array. For each of the devices that the capture identifies the packets present are tallied and if the data usage threshold is exceeded a text message is sent to the given number.
- -s subnet address required
    - The program requires a subnet address with its CIDR notation such as "192.168.1.0/24"
- -h help page
    - Displays a help dialog which is generated by the program

The program when run without any switches will display the help dialog and introductory graphic.

2.4 IMPLEMENTATION

The design and requirements above were now implemented, this began by implementing the Scapy component of the NMAM script.

- Scapy

```
# use an arp scan to identify devices
arp = ARP(pdst=args.subnet)
ether = Ether(dst="ff:ff:ff:ff:ff:ff")
packet = ether/arp
result = srp(packet, timeout=3, verbose=0)[0]

Devices = []
host = []
for sent, received in result:
    Devices.append({'IP': received.psrc, 'MAC': received.hwsrc})
```

Figure 2-2 Scapy

Figure 2-2 identifies the initial component where the given subnet is pinged. This enables the program to identify live hosts on the network before passing to the Nmap scan to identify the operating system and open ports.

- Nmap

```python
#device scan
def scan_Device(addr):
    # add to graph
    Graph.add_node(addr)
    nm = nmap.PortScanner()
    # check open ports and log  #nmap scon with the os and servoce version
    nm.scan(hosts=addr, arguments='-O -sV')
    #tcp connections
    if 'tcp' in nm[addr]:
        for port in nm[addr]['tcp']:
            if nm[addr]['tcp'][port]['state'] == 'open':
                #add the open ports to the device
                Graph.nodes[addr]['open_ports'] = Graph.nodes[addr].get('open_ports', []) + [port]
                #if os matches add to graph
        if 'osmatch' in nm[addr]:
            os_matches = nm[addr]['osmatch']
            if os_matches:
                Graph.nodes[addr]['os'] = os_matches[0]['name']
                #quick list scan to check if device is up and plot on graph
        nm.scan(hosts=addr, arguments='-sL')
        for node in Graph.nodes():
            for host in nm.all_hosts():
                if host != node and nm[addr]['status']['state'] == 'up':
                    Graph.add_edge(node, host)
    else:
        print("device added but no information given")  # if device is unresponsive
```

Figure 2-3 Nmap and Graph plotting

Figure 2-3 identifies the Nmap scan switch options and graph data array. The program initially adds a node to the graph at the IP address and then proceeds to run an Nmap scan with the -O and -sV switches. After the scan of the address is complete the program proceeds to identify any open ports and assigns them to the graph data array. Next if an operating system match has been identified the program adds this to the graph data array.

Finally, a Nmap scan with the "-sL" switch is executed this allows Nmap to check if the device state is live. If the device is live and different from the host a connection is, then added to the graph data array.

- Graph component

```python
node_labels = {
node: f'{node}\n{Graph.nodes[node].get("os", "")}\n{Graph.nodes[node].get("device_type", "")}\nOpen Ports: {", ".join(str(port) for port in Graph.nodes[node].get("open_ports", []))}'
for node in Graph.nodes()


nx.draw_networkx_nodes(Graph, pos, node_size=500, node_color='red')
nx.draw_networkx_edges(Graph, pos, width=1, alpha=0.5)
nx.draw_networkx_labels(Graph, pos, labels=node_labels, font_size=8)
plt.axis('off')
plt.savefig('img.png')
# Draw the graph
print(Graph.nodes)
print(Graph.edges(data=True))
```

Figure 2-4 Graph

The final part of the mapping component is the graph this begins by writing the graph labels such as open ports and device types on the nodes. This then plots this on the graph and saves the image as 'map.png'

- Pyshark component

```python
#data usage
def dataUsage(threshold, interface):

    #start the capture and initalse the devoce
    devices = {}
    capture = pyshark.LiveCapture(interface)
    #begin packet sniffing
    for packet in capture.sniff_continuously():
        #search ip
        if 'ip' in packet:
            # get src and dst ip's
            src_ip = packet.ip.src
            dst_ip = packet.ip.ds
            #add to or count up packet length from src
            if src_ip in devices:
                devices[src_ip] += int(packet.length)
            else:
                devices[src_ip] = int(packet.length)
            #add to or count up packet length from dst
            if dst_ip in devices:
                devices[dst_ip] += int(packet.length)
            else:
                devices[dst_ip] = int(packet.length)
                #check if any devoce has breached threshold
            for device in devices:

                if devices[device] > threshold:
                    #print and call message system
                    print(f"Device {device} has exceeded the threshold ({threshold} bytes).")
                    sms_Mesage(device)
                    #end the monitor
                    break
```

Figure 2-5 Pyshark

Figure 2-5 identifies the Pyshark component of the NMAM program. The packet sniffer identifies the data usage by the devices on the network in real time. This information is then added to an array. If the program identifies an IP address breaching its assigned data usage it displays a message and calls the SMS message function. If the data usage threshold is not exceeded the script continuously loops.

- Twilio component

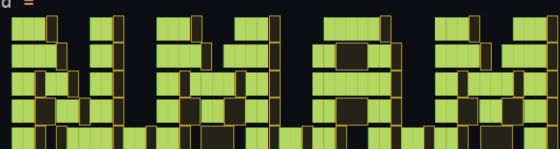
```python
def sms_Mesage():
  client = Client(account_sid, auth_token)
  message = client.messages.create(
    body="A device has exceeded the threshold",
    from_= TW_Number,
    to= DEST_Num
  )
  print(message.sid)
```

Figure 2-6 Twilio component

Figure 2-6 identifies the message function of the Pyshark component. If a breach is detected a SMS message is generated and sent to the desired number using the Twilio credentials.

- Argparse component

```python
# art headder
art_head = """
```



```python
        Network      Map          And       Monitor
        By Paul Oates
        """

print(art_head)
# Initialize the argument parser
parser = argparse.ArgumentParser(description='Network Monitoring Tool')


parser.add_argument('-m', '--map', action='store_true', help='Map the network')
parser.add_argument('-b', '--bytes', type=str, help='Monitor data usage. limit in bytes')
parser.add_argument('-s', '--subnet', type=str, help='Subnet to scan. Required', required="True")
```

Figure 2-7 Argparse and art

Figure 2-7 identifies the arguments that are required from the command line. This includes the -m, -s, -b switches. The -s switch is always required whilst the -m and -b switches can be used together or separately.

Arguments such as the Twilio auth codes and phone numbers are declared at the start of the program and would need to be included by the end user.

2.5 TESTING

Next the program is tested on both Linux and Windows operating systems. Network configurations were built using GNS3, a network emulation tool setup to test the script in a Linux environment. The Windows environment was tested using my home network. The Key Performance Indicators (KPI) of the program were identified and these include:

- Ability to execute the program on both Linux and Windows operating systems
- Ability to scan subnets with the script
- Ability to conduct real time data usage monitoring
- Ability to set a data usage threshold and be alerted when exceeded

GNS3 was initially ran with a network consisting of three Ubuntu Cloud guests on the 192.168.122.0/24 subnet.

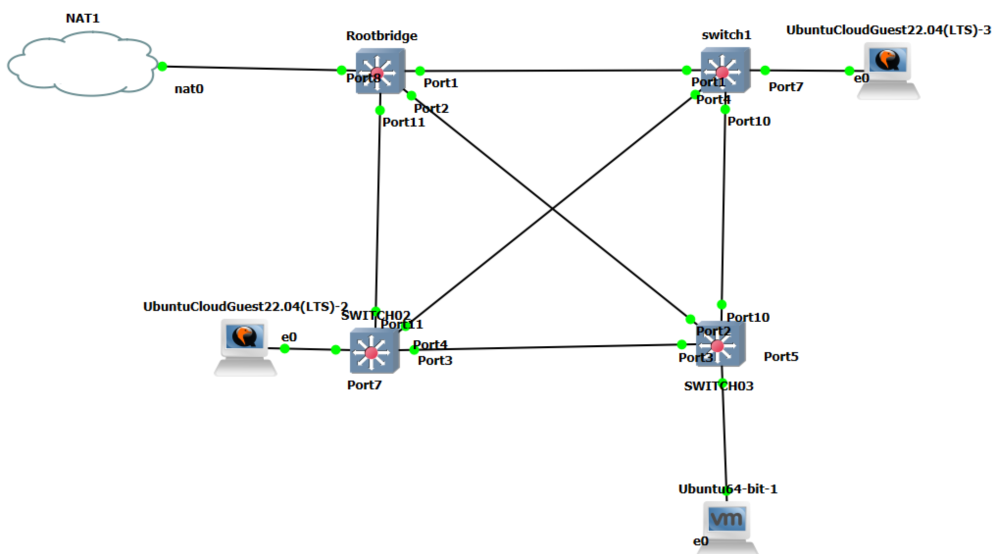


Figure 2-8 GNS3

Figure 2-8 identifies the network setup. The setup consists of four EXOS switches (GitHub, 2017) in a Rapid Spanning Tree Protocol (RSTP) network. On the network there are two Ubuntu Virtual Computers and a VMware Ubuntu machine. The VMware Ubuntu machine is where the NMAM script is ran from.

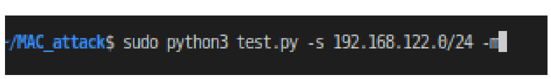


Figure 2-9 NMAM script

Figure 2-9 identifies the command ran to execute the NMAM mapping function. The script makes use of the “-s” switch for the subnet option and “-m” for the mapping option.

Figure 2-10 NMAM result

Figure 2-10 identifies the three hosts on the network as Linux devices. The open ports and Linux kernel version of each device was also displayed.

Next NMAM was tested against my home network, this was done using a Windows 11 machine.

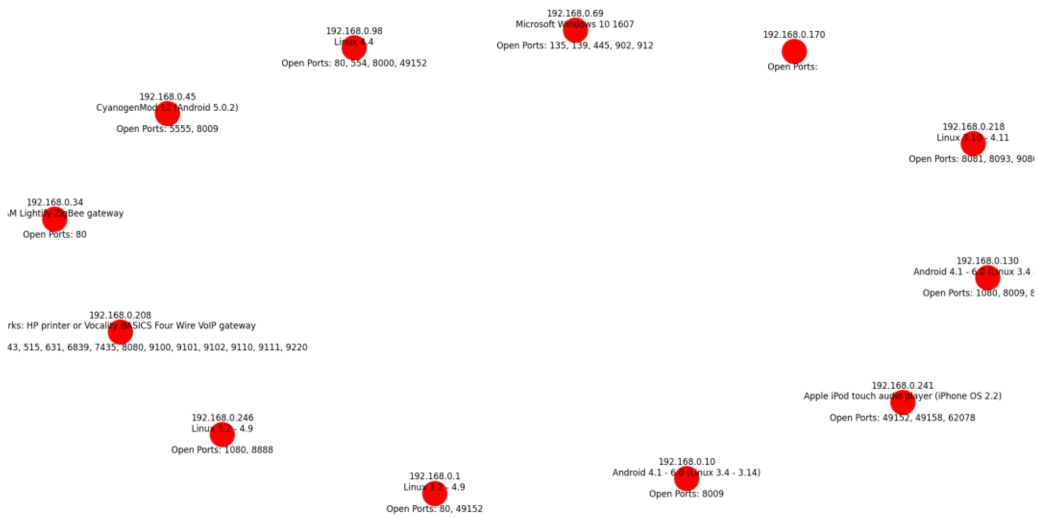


Figure 2-11 Scan result

Figure 2-11 identifies the following hosts live on my home network.

The data usage monitoring component was also tested within the GNS3 virtual network.

```
ubuntu@ubuntu-virtual-machine:~/MAC_attack$ sudo python3 test.py -s 192.168.122.0/24 -b 512
```

Figure 2-12 data usage monitoring

Figure 2-12 identifies the command ran against the network for data usage monitoring, it includes the subnet and the -b switch this is followed by the number of bytes the data usage limit is set at.

```
PING 1.1.1.1 (1.1.1.1) 56(84) bytes of data.
64 bytes from 1.1.1.1: icmp_seq=1 ttl=127 time=52.0 ms
64 bytes from 1.1.1.1: icmp_seq=2 ttl=127 time=42.4 ms
64 bytes from 1.1.1.1: icmp_seq=3 ttl=127 time=46.5 ms
64 bytes from 1.1.1.1: icmp_seq=4 ttl=127 time=39.0 ms
64 bytes from 1.1.1.1: icmp_seq=5 ttl=127 time=52.9 ms
64 bytes from 1.1.1.1: icmp_seq=6 ttl=127 time=24.6 ms
64 bytes from 1.1.1.1: icmp_seq=7 ttl=127 time=32.2 ms
64 bytes from 1.1.1.1: icmp_seq=8 ttl=127 time=31.1 ms
64 bytes from 1.1.1.1: icmp_seq=9 ttl=127 time=27.1 ms
64 bytes from 1.1.1.1: icmp_seq=10 ttl=127 time=23.0 ms
```

Figure 2-13 192.168.122.83 ping

Figure 2-13 identifies the Ubuntu Linux machine pinging 1.1.1.1 this will exceed the data usage limit after a short number of ping attempts.

Figure 2-14 NMAM

Figure 2-14 identifies the IP address of the device which exceeded the data usage limit.

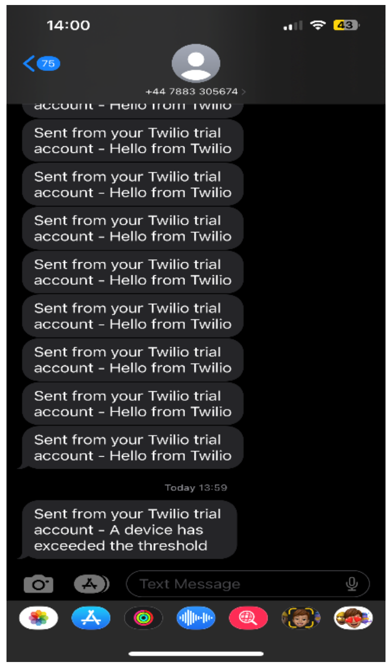


Figure 2-15 Text message

Figure 2-15 identifies the message sent to the device identifying that the data usage has been exceeded by the device.

2.6 Maintenance

I aim to make the script open source on GitHub so it can be improved upon and free for anyone to use. A Readme file will be included to identify the switches and libraries required for the program to run as well as a description of how to use the tool.

3 DISCUSSION

3.1 GENERAL DISCUSSION

The NMAM tool will enable network administrators to thoroughly map a network identifying both ports and operating systems on a given subnet. By monitoring the data usage this allows users to identify if devices on the network exceed the defined data limit. The tool is cross compatible on both Linux and Windows machines enabling a complete scanning and data usage monitoring tool regardless of the environment it is deployed in. The tool enables effective network security by using Nmap's OS scan switch which is then displayed via the Networkx and Matplotlib libraries. This allows the user to quickly identify any outdated operating system versions on the network and begin to mitigate any risk posed by them. By displaying the open ports on the network devices, this also helps as it enables the end user to review and assess the open port usage. The Pyshark library enables the program to have live data usage monitoring which would enable the user to see if a device has gone over the predefined data usage threshold limit. An attack like a Denial of Service (DOS) would be quickly identified and using the Twilio library a SMS message could be sent to the end user.

<u>User Friendliness</u>

The tool adheres to similar formats as other networking tools such as Nmap by passing arguments, utilising switches which are entered from the command line. The use of the command line is standard for tools such as this however this presents a significant challenge to a user with no experience in command line interfaces. This could be beneficial in preventing the tool from being misused.

NMAM furthers its user friendliness by saving the network scan in a ".png" image this enables the user to add their network scan to various other reports and documentation. The tool could be integrated into a testing suite of tools that a network administrator may make use of to evaluate the security of the network. The tool makes use of Nmap and Wireshark which are common tools widely understood by network administrators so the learning curve to integrate this tool would not be too difficult.

The Readme file informs the user on how to use the NMAM script and this can be found in Appendix B-Readme. It consists of the required libraries and instructions on how to install them. Command line interface examples and a description of what the tool does. This adds to user friendliness as the user is informed on how to set up the tool.

<u>Scalability</u>

As seen in testing 2.5 the script worked on a small GNS3 emulated network consisting of 3 Linux machines. It also successfully ran on a larger home network consisting of various IOT, mobile and Windows devices. The script successfully identified the operating systems and open ports on the connected devices. As future growth in networks is inevitable, the tool aims to accommodate this by enabling support for larger subnets such as /16 and /8. This would however take longer to execute, and multithreading should be implemented to allow multiple IP's to be scanned simultaneously.

Methodology Used

The waterfall methodology proved successful as it met the criteria it was selected for.

Comparison

When comparing the NMAM script with other market leading tools such as SolarWinds the "most highly recommended" (logicalread.com, 2021) network topology mapper several key benefits to using the NMAM script were identified. These include:

- Free to use – Solar winds costs over £1000 (solarwinds.com, nd) to use whilst NMAM is free and Open Source
- Customisability – due to it being Open Source the software can be customised to meet the specific need of a network engineer
- Scalability – The program can be utilised on small home networks and large enterprise networks. The script can be modified to accommodate the scalability needs of the user
- Integration – Due to the script being written in python it can be integrated within a larger Security Information and Event Management (SIEM) Suite

Limitations

One certain limitation of the tool was Nmap's OS scan. The scan shown in Figure 2-11 successfully identified an Apple IOS device however it failed to identify the correct operating system. This in a business environment could cause issues. The scan could identify false positive operating systems from a scanned subnet. A more rigorous OS detection would need to be implemented to mitigate against this.

Another limitation is the scanning speed of NMAM. The scanning speed is O(N) where N is the number of assets on the network meaning larger networks will take a considerable time to scan. In future iterations I would implement multithreading to reduce the scanning time closer to O(1). Further code optimisations would need to occur to enable this and increase efficiency.

A further issue is multiple subnet scanning, currently if a user wants to scan two or more separate subnets this requires the user to run these as separate scans. In future updates my aim would be to pass in a list of subnets for the tool to scan and appropriately map out all connections.

An additional issue is Router and L3 switch identification. In future releases I aim to include a clearer device map differentiating routers and switches from other devices.

Finally in future versions it should be possible for the end user information to be entered via the command line rather than editing the code.

3.2 COUNTERMEASURES

Data Usage

As the NMAM script detects high data usage an exploit such as the Slowloris would not be detected.

Countermeasure

In Intrusion Detection System (IDS)or Intrusion Prevention System (IPS) would detect this.

Firewalls

Firewalls are likely to block the network mapping component of the NMAM tool making it difficult to detect operating systems and open ports.

Countermeasure

This could however be disabled by the network administrator temporarily for the tool to run.

Network Fuzzing /cloning

Network fuzzing and network clones are likely to increase the number of false positives on the network which can lead to time being wasted mitigating issues on networks that do not exist.

Countermeasure

It is important for the network administrator utilising the tool to have a good knowledge of the network and this being carefully documented

Updates

Due to the Nmap tool being last updated in 2020 more modern operating systems are unlikely to be detected correctly by the tool.

Countermeasure

Develop a new OS scanning tool which is kept regularly updated

Finally, the NMAM tool should be used within a wide array of security tools such as Intrusion Detection Systems (IDS), Intrusion Protection Systems (IPS), Firewalls, Virtual Private Networks (VPN's), Clones, and Antivirus Software to ensure a complete security package that protects the network.

3.3 FUTURE WORK

I believe the NMAM script is a good starting point for targeted network scanning. It has certain limitations that could be improved upon in future revisions of the script.

Router and L3 Switch Identification

The ability to identify routers and layer three switches on the network would be beneficial as this would allow network administrators to identify bottlenecks on the network and be able to mitigate against them.

Multithreaded Scanning Techniques

By utilising multithreading this would significantly speed up the script adding to its effectiveness allowing it to scan larger and more complex networks.

Improved OS Detection

Revised OS detection would enable the script to effectively identify vulnerable devices on the network making it more effective against a wide array of operating systems.

Better Integration

The tool in future revisions should include a PDF output report to document devices, operating systems, and network maps. This increases the usability of the script allowing for the information to be captured in a network report.

Better Alert Dialog

When the data usage is exceeded an alert dialog, which identifies to the user what device has exceeded the limit and by how much could be included in the script.

3.4 CONCLUSION

In conclusion the NMAM script enables a user, such as a network administrator to map a network which is then saved as a ".png" image. Using Pyshark as a data usage monitor allows a data threshold for devices on the network to be set and an alert sent if this is exceeded in real time.

After researching current network mapping tools Nmap and Scapy were integrated into the NMAM script to ensure a reliable network scan. All the required development libraries and software were installed for the script to utilise. Utilising tools such as Matplotlib and Networkx enables the end user to visualise the network to better understand it. By using tools such as GNS3 enabled testing of the tool on complex network layouts. Comparing the NMAM script to paid tools such as SolarWinds topology mapper highlighted there were clear benefits of using the Python script.

This tool will lead to better network visibility and enable network administrators to identify assets and resolve issues. Moving forward I aim to upload the scanning tool to GitHub where improvements can be implemented.

References

Grant Shirk, Complexity, Connectivity, and Capacity: Putting Network Data to Work, https://meraki.cisco.com/blog/2021/09/complexity-connectivity-and-capacity-putting-network-data-to-work/ [September 22, 2021] [Accessed On 22/05/2023]

Welcome to the OWASP Top 10 – 2021, https://owasp.org/Top10/ , [No Date] [Accessed On 22/05/2023]

Staff, 7 Best Network Mapping Tools, https://logicalread.com/network-mapping-tools/#.ZGvHO33MLtU [7th January 2021] [Accessed On 22/05/2023]

Solar winds Checkout , https://ecomm.solarwinds.com/onepagecheckout/SolarWinds?sku=3340&qty=1&CMPSource-SWINTMPDPHERO%20&CMP=ORG-BLG-LOGIC-X_WW_X_NP_X_X_EN_0_X-NTM-20210107_NetworkMappingT_X_X_X_X-X#/ [No Date] [Accessed On 22/05/2023]

steWilliams-extr, RSTP (802.1w) example GNS3 project, https://github.com/extremenetworks/Virtual_EXOS/tree/master/gns3_projects/RSTP_LAB [June 2017] [Accessed On 22/05/2023]

Waterfall, https://www.codecademy.com/resources/docs/general/software-development-life-cycle/waterfall-model [March 2023][Accessed On 22/05/2023]

Code

Networkx examples, https://networkx.org/documentation/stable/auto_examples/index.html [No Date][Accessed On 22/05/2023]

Pyshark documentation, https://github.com/KimiNewt/pyshark/ , [March 2023] [Accessed On 22/05/2023]

Python Nmap Module Fully Explained with 8 Programs, https://www.pythonpool.com/python-nmap/ [July 2021] [Accessed On 22/05/2023]

Send SMS and MMS Messages in Python, https://www.twilio.com/docs/sms/tutorials/how-to-send-sms-messages-python [No Date] [Accessed On 22/05/2023]

Tshepang Lekhonkhobe, Argparse Tutorial, https://docs.python.org/3/howto/argparse.html [No Date] [Accessed On 22/05/2023]

Examples, https://matplotlib.org/stable/gallery/index.html [No Date] [Accessed On 22/05/2023]

How to Use Scapy – Python Networking Tool Explained, https://www.freecodecamp.org/news/how-to-use-scapy-python-networking/ [21 December2022] [Accessed On 22/05/2023]

APPENDICES

APPENDIX A – TOOLS UTILISED

GNS3 - https://docs.gns3.com/docs/getting-started/installation/windows/

Visual Studio Code - https://code.visualstudio.com/

Python 3 - https://www.python.org/downloads/

Nmap - https://nmap.org/download.html

Twilio - https://www.twilio.com/login

Scapy - https://scapy.net/

Matplotlib - https://matplotlib.org/

Networkx - https://networkx.org/

Tshark - https://tshark.dev/setup/install/

Wireshark - https://www.wireshark.org/

Ubuntu - https://ubuntu.com/

Windows 11 - https://www.microsoft.com/en-gb/software-download/windows11/

EXOS switch - https://github.com/extremenetworks/Virtual_EXOS

APPENDIX B – README

Readme.txt

```
N.M.A.M
        Network      Map           And        Monitor
        By Paul Oates

Welcome to NMAM a tool for network mapping and badwidth monitoring. The following libaries are required:

pip install argparse
pip install nmap
pip install networkx
pip install matplotlib
pip install scapy
pip install pyshark
pip install twilio

The following programs are required

https://nmap.org/
https://www.wireshark.org/download.html

Twilio keys can be genarated here. These are not incuded

https://www.twilio.com/en-us

Usage
python .\NMAM.py -s [Subnet Required] -m -b [bytes]
Note -m and -b are optional

Examples

python .\NMAM.py -s 192.168.1.0/24 -m -b 4096
python .\NMAM.py -s 192.168.1.0/24  -b 4096
python .\NMAM.py -s 192.168.1.0/24 -m
python .\NMAM.py -s 192.168.1.0/24 -h


help dialog

usage: NMAM.py [-h] [-m] [-b BYTES] -s SUBNET

Network Monitoring Tool

optional arguments:
  -h, --help            show this help message and exit
  -m, --map             Map the network
  -b BYTES, --bytes BYTES
                        Monitor data usage. limit in bytes
  -s SUBNET, --subnet SUBNET
                        Subnet to scan. Required
```