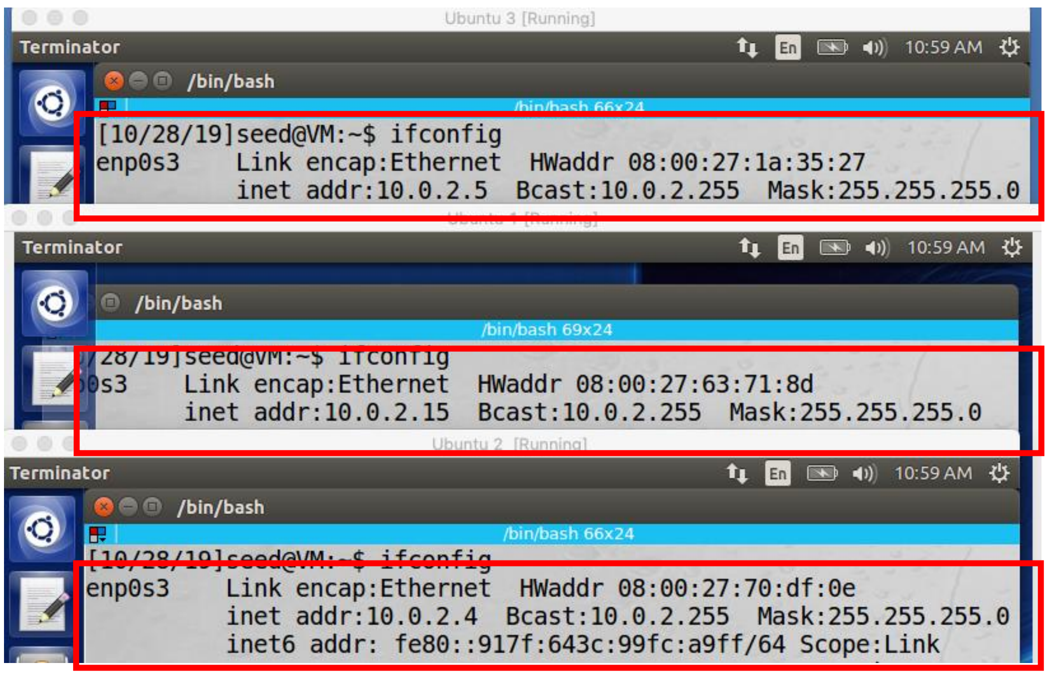
Cody Williams
CSCE 465
October 24 2019

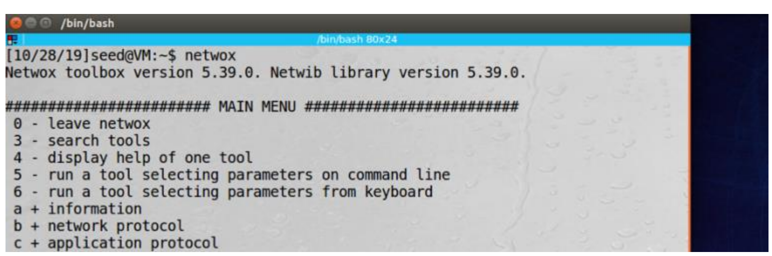
Homework 4 report

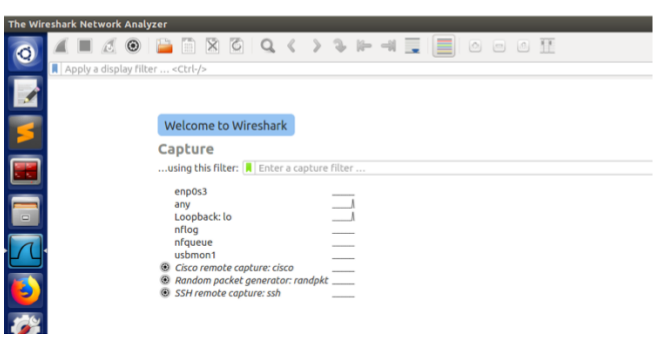
2.1 Environment Setup

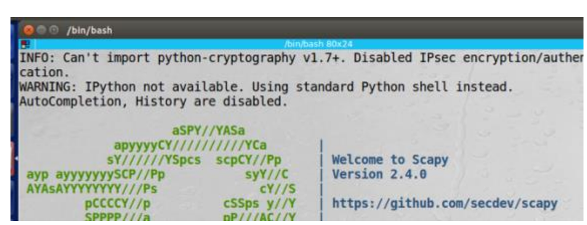
Setting up 3 VM's on the same local network. My VM's happen to exist in the 10.0.2.x address space




Netwox, Scapy and wireshark tools



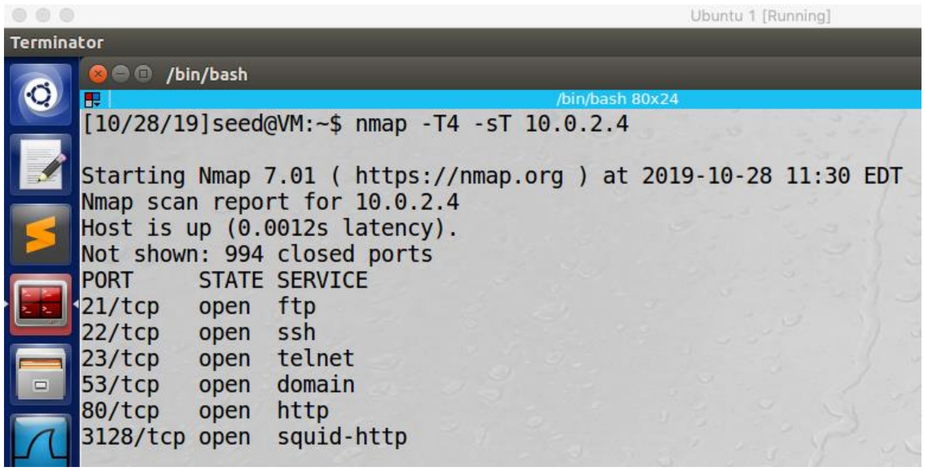



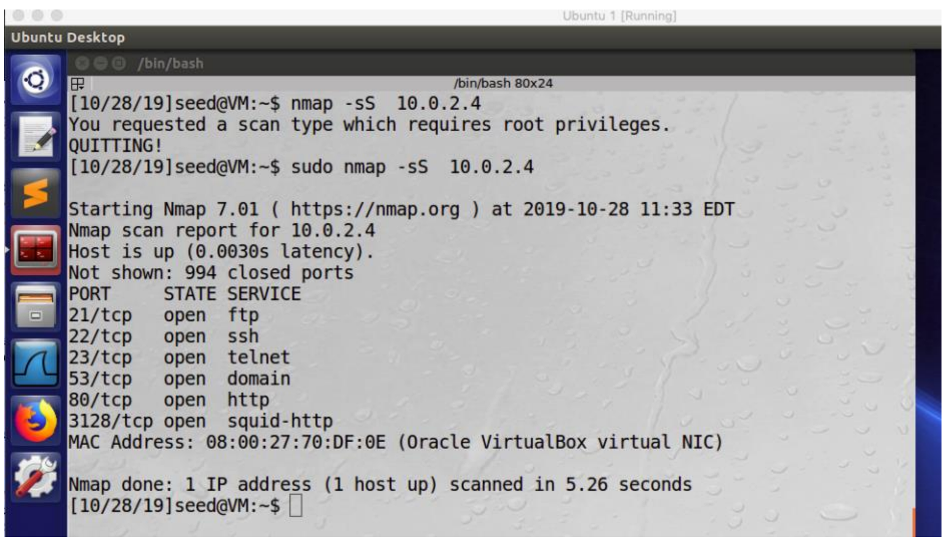
3.1 Task (1) : Surveillance Techniques

Port Scanning: Host 10.0.2.15 attacking host 10.0.2.4

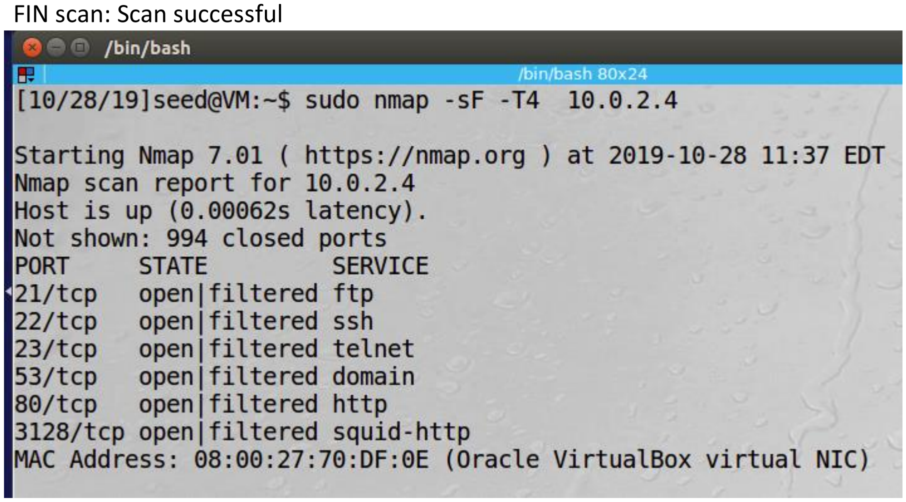
TCP Connect scan : Scan successful



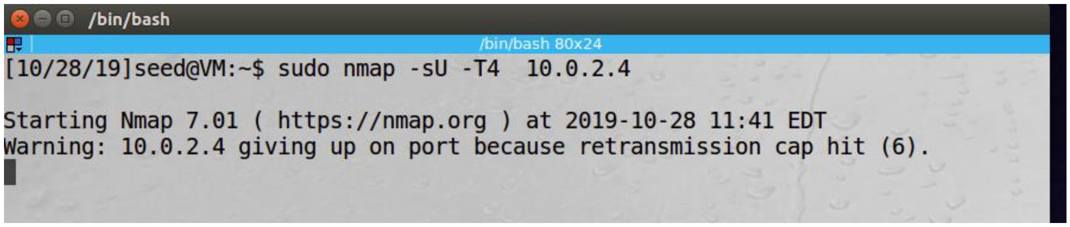

SYN stealth scan: Scan successful



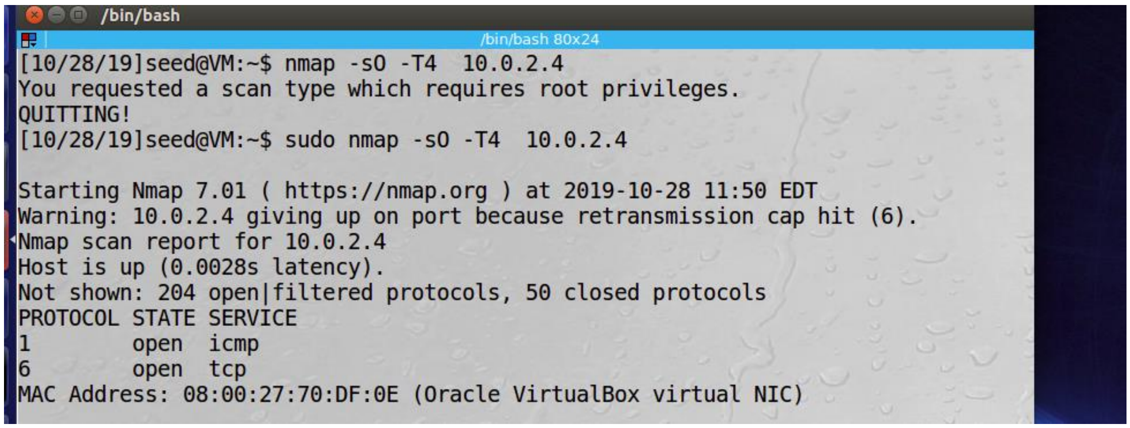

FIN scan: Scan successful



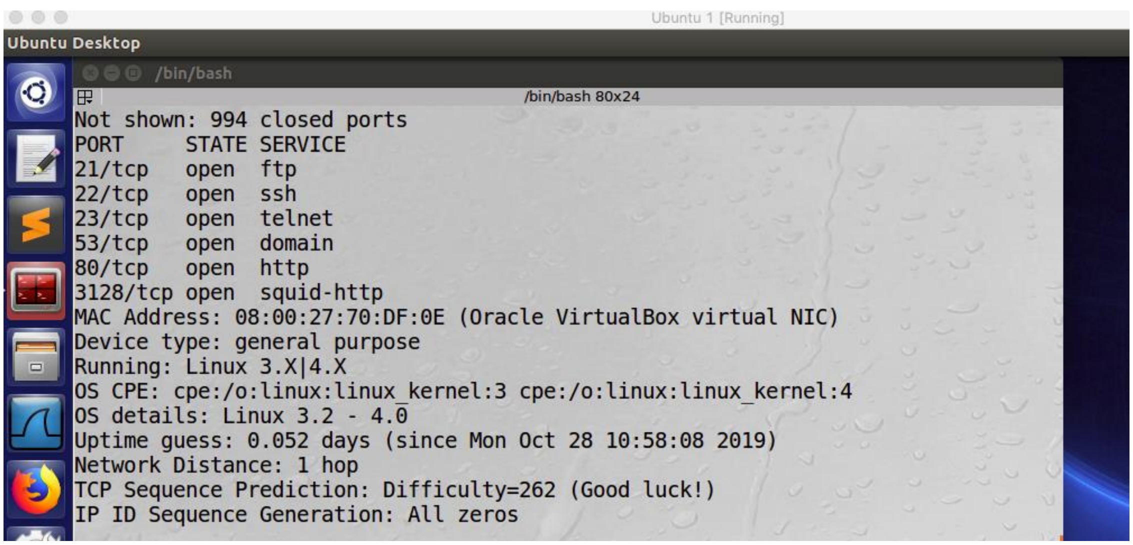
UDP scan: Scan appeared to fail




IP Protocol Scan: Scan successful




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




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](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 TPC 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

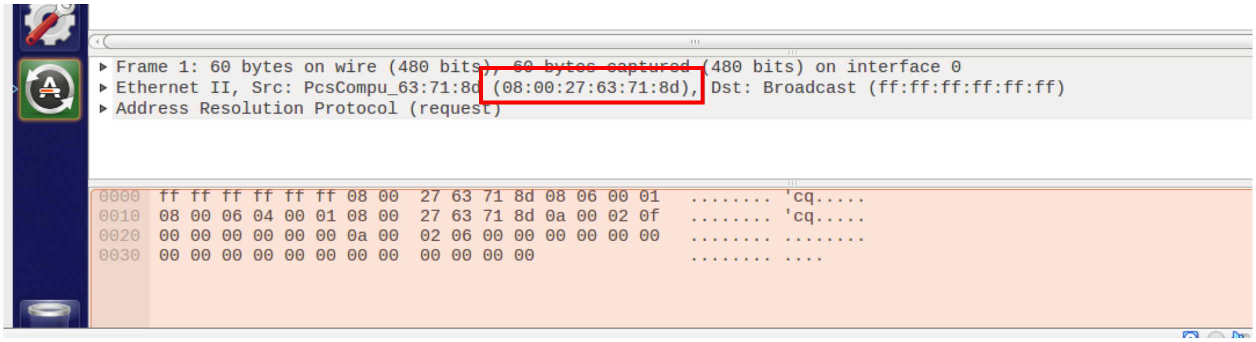
1. Use ARP -n to view the MAC addresses of the machines currently running on the network. In the below example, all machines are residing in on the ethernet network

```
/bin/bash
/bin/bash 97x24
[10/28/19]seed@VM:~$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
10.0.2.3                 ether   08:00:27:9d: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

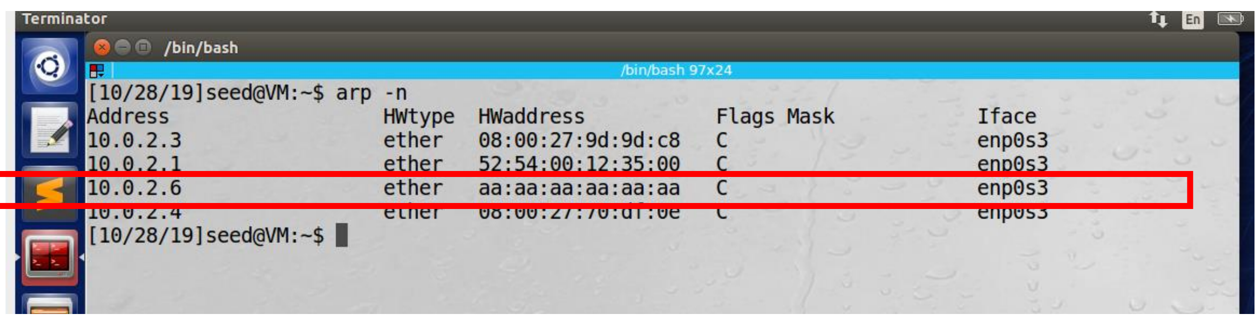
```
[10/28/19]seed@VM:~$ ping 10.0.2.6
PING 10.0.2.6 (10.0.2.6) 56(84) bytes of data.
From 10.0.2.15 icmp_seq=1 Destination Host Unreachable
From 10.0.2.15 icmp_seq=2 Destination Host Unreachable
From 10.0.2.15 icmp_seq=3 Destination Host Unreachable
From 10.0.2.15 icmp_seq=4 Destination Host Unreachable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

3. Use Wireshark to view the MAC address needed for assignment in the spoofing step using netwox




4. 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


5. 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:

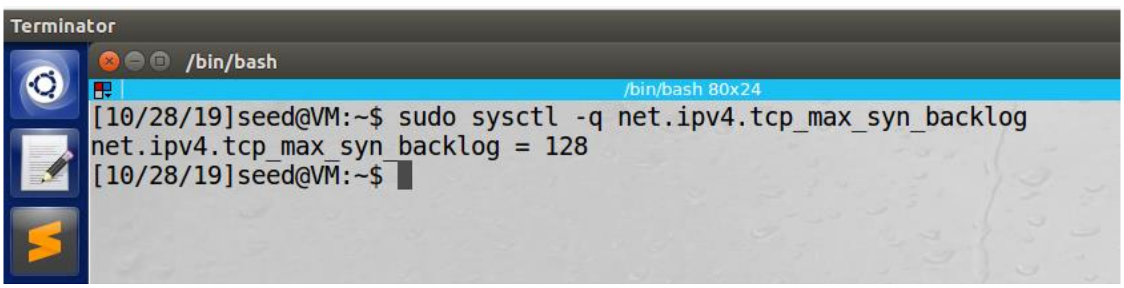
My attack was successful and could be completed remotely from the attacker's device using netwox and Wireshark

Defensive thought:

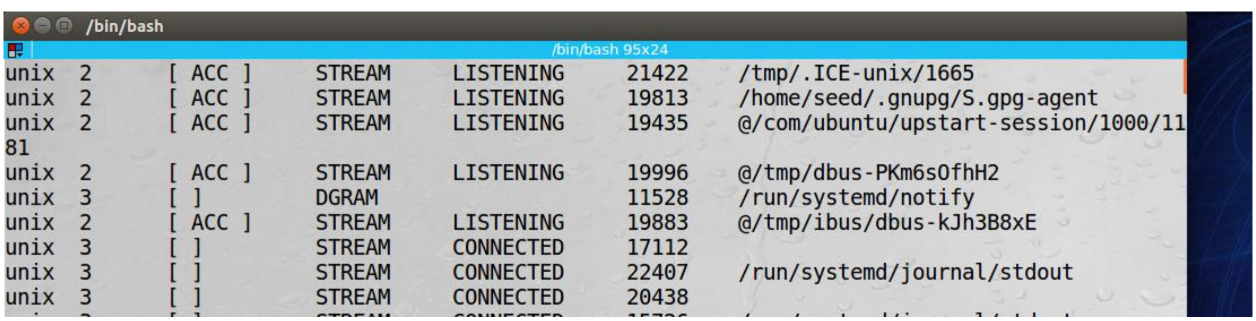
In order to defend against ARP cache poisoning, I think it would be very beneficial for network administrators to completely prevent the use of netwox tools on their machines. Disabling the ability to create a fake MAC address mapping in victim machines can help to prevent the kind of Dos attacks mentioned 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

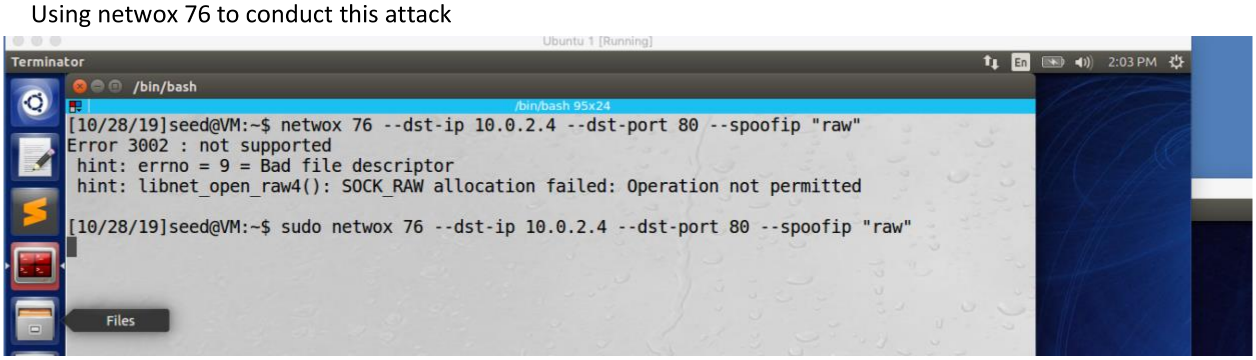
Checking the system queue size setting



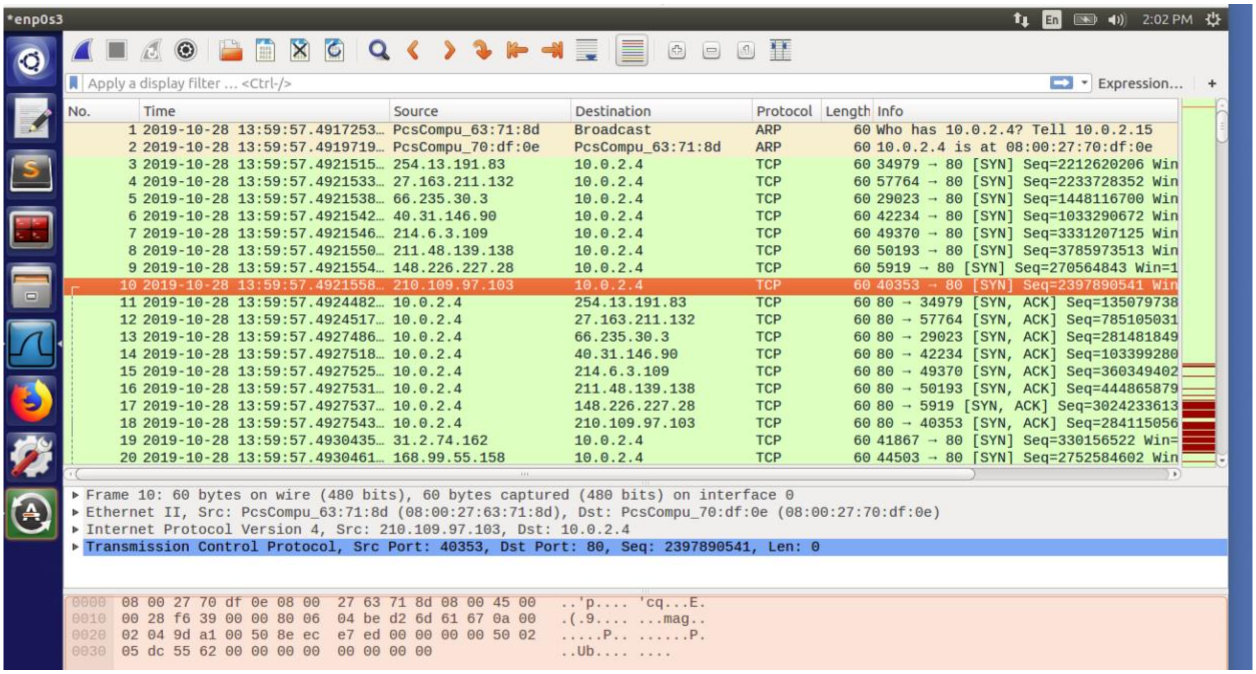

Checking usage of the queue



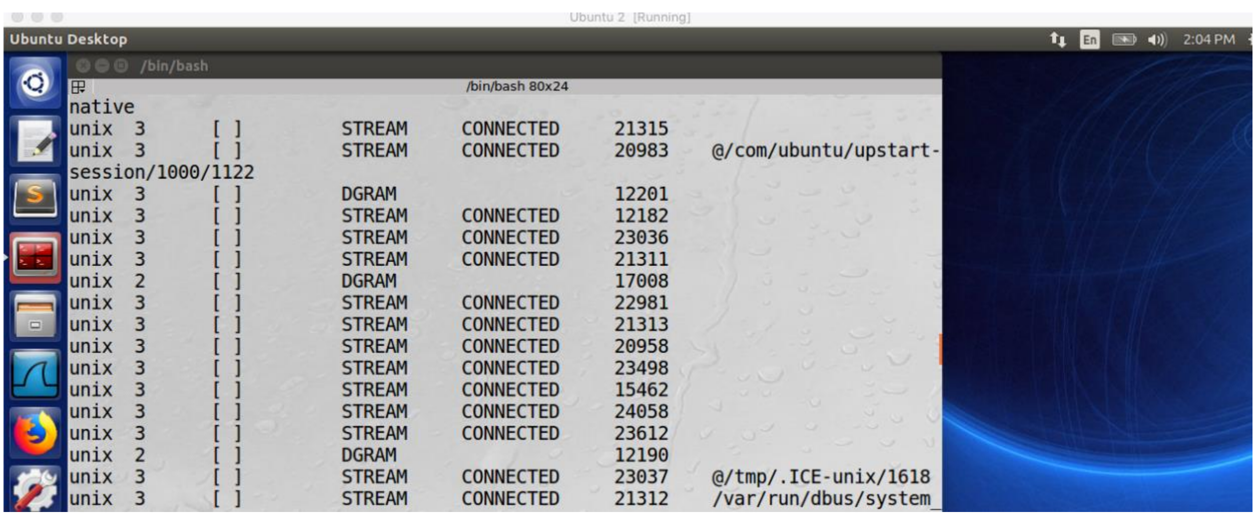

Using netwox 76 to conduct this attack



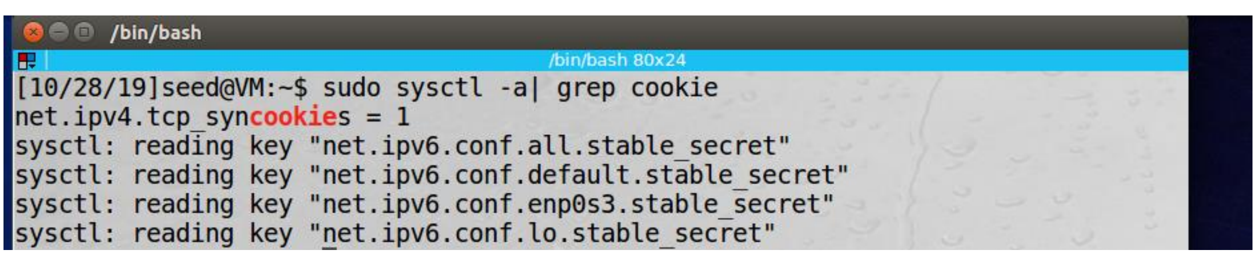
Sniffer tool observing the attack taking place: This was the traffic that was captured immediately after running the netwox 76 command



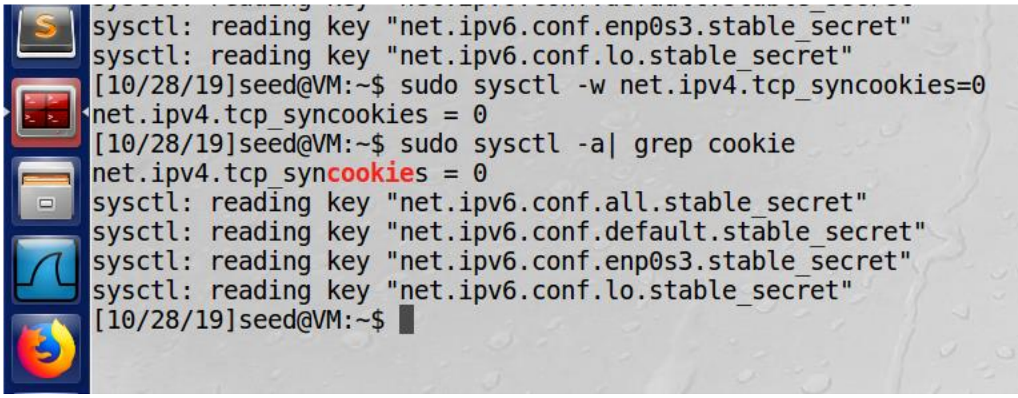

Running netstat -na on the victim machine during the attack: The output appears to be very similar to the output from the netstat -na command ran prior to the netwox 76 attack.



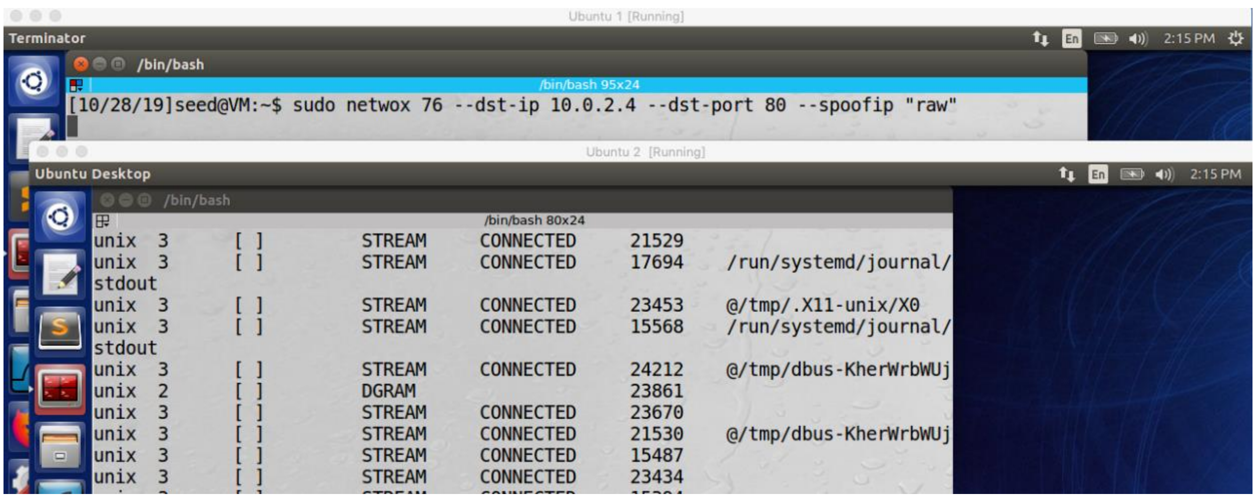

Checking the state of the SYN cookie. It appears to be on



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




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




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

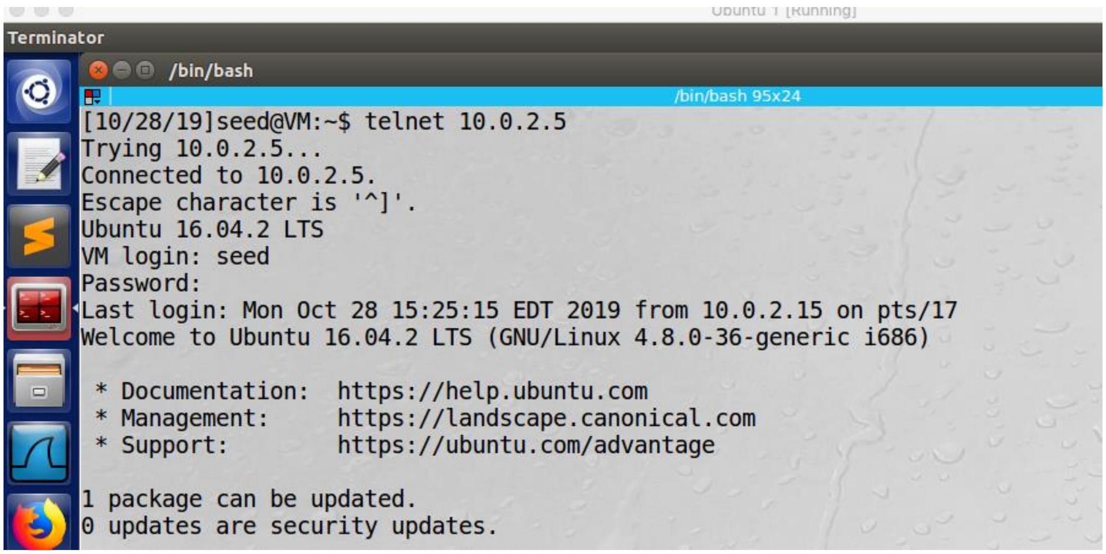
The attack appeared to be successful as observed through the packets captured through Wireshark.

Defensive thought

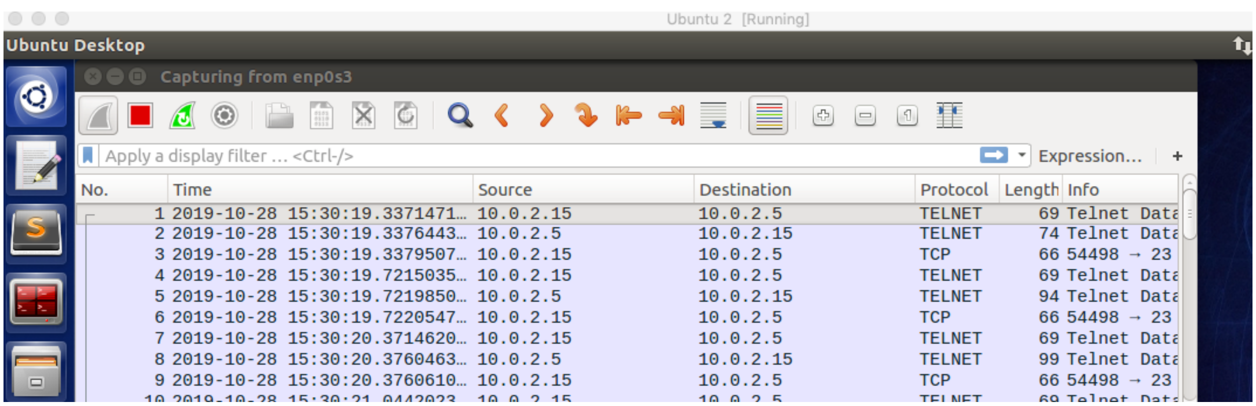
As mentioned in the assignment description, an effective way to defend against a SYN flooding attack is to have the SYN cookie turned on. By turning on the SYN cookie, an attacker is unable overload the Victim's SYN queue in order to create a form of denial of service attack.

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

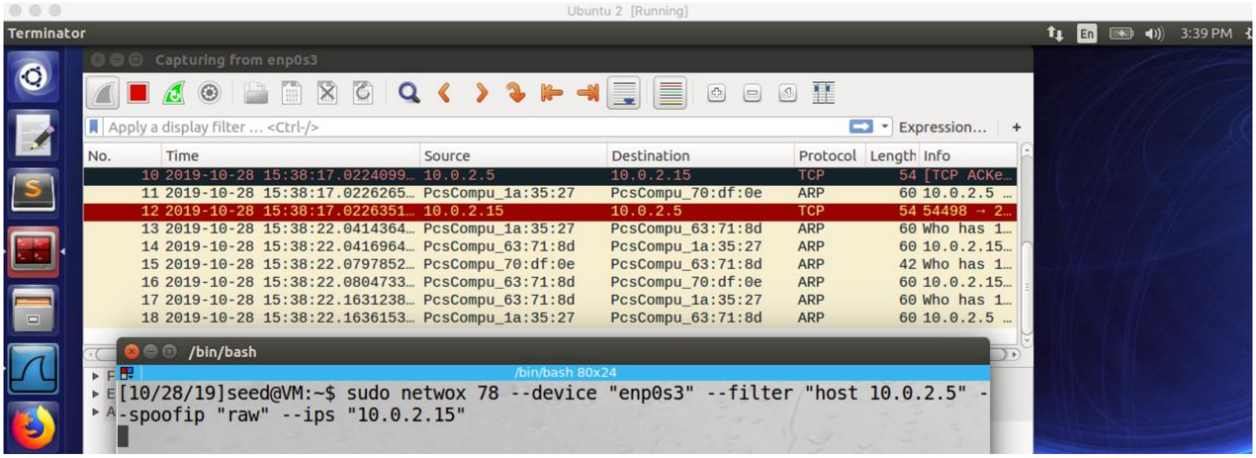
Create a telnet connection between two machines. Connection is between 10.0.2.5 and 10.0.2.15



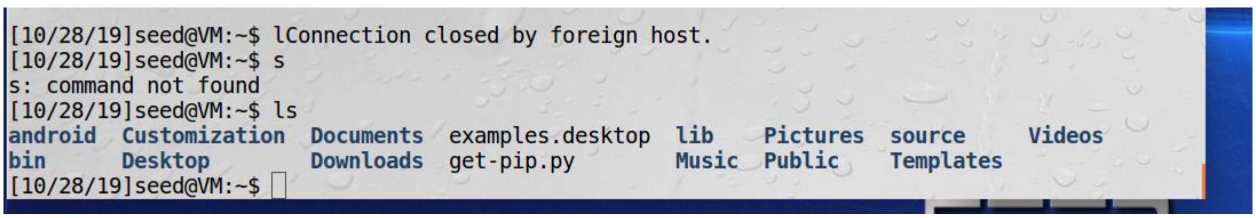

Capturing telnet traffic between 10.0.2.5 and 10.0.2.15 with Wireshark



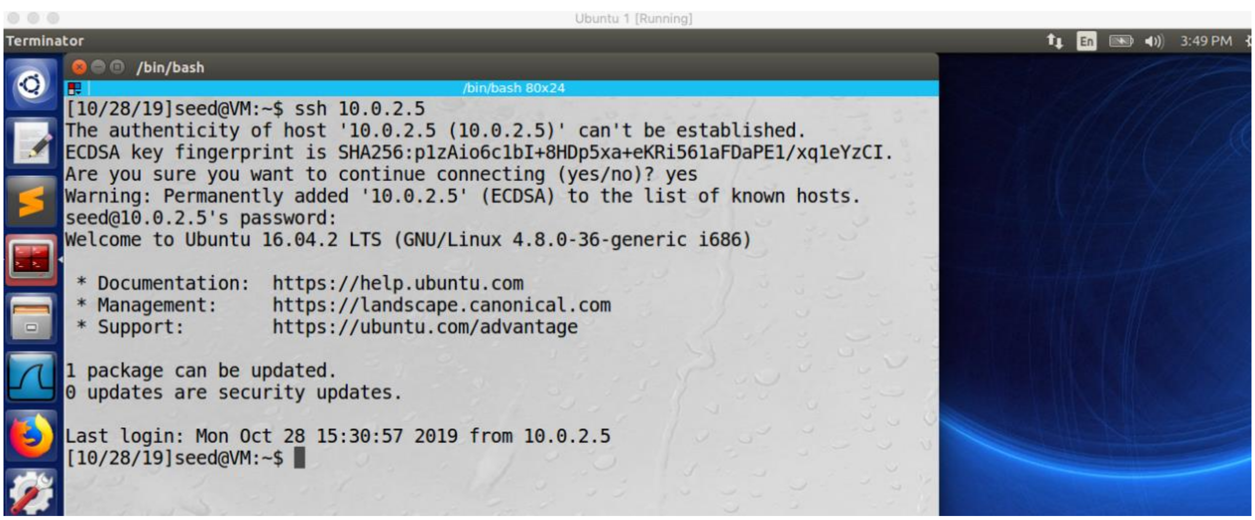
Launching the attack using netwox 78 and observing the behavior through Wireshark



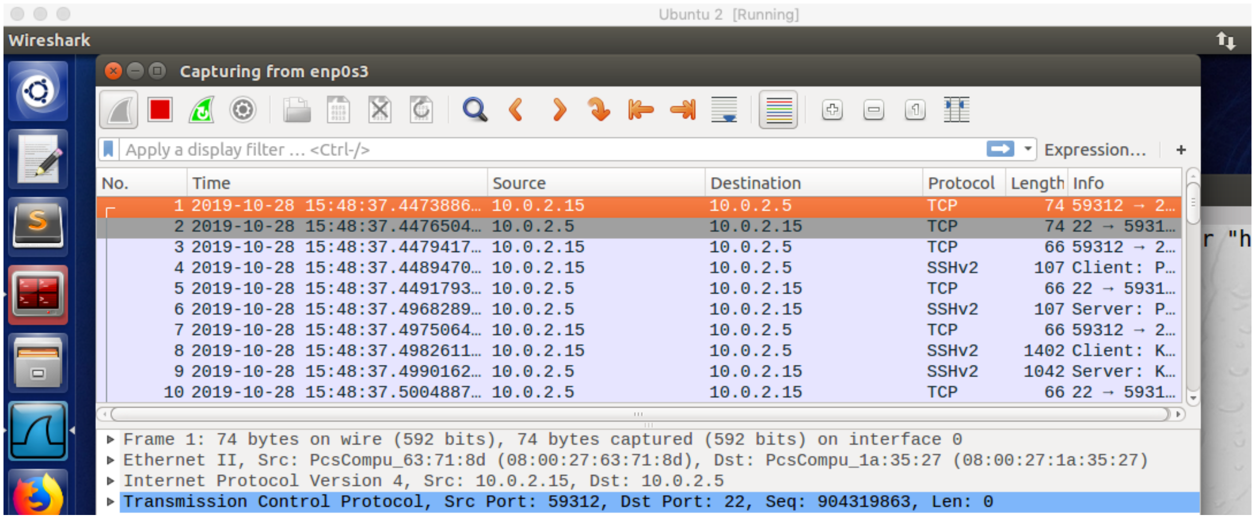

Victim can now see the files of the observer



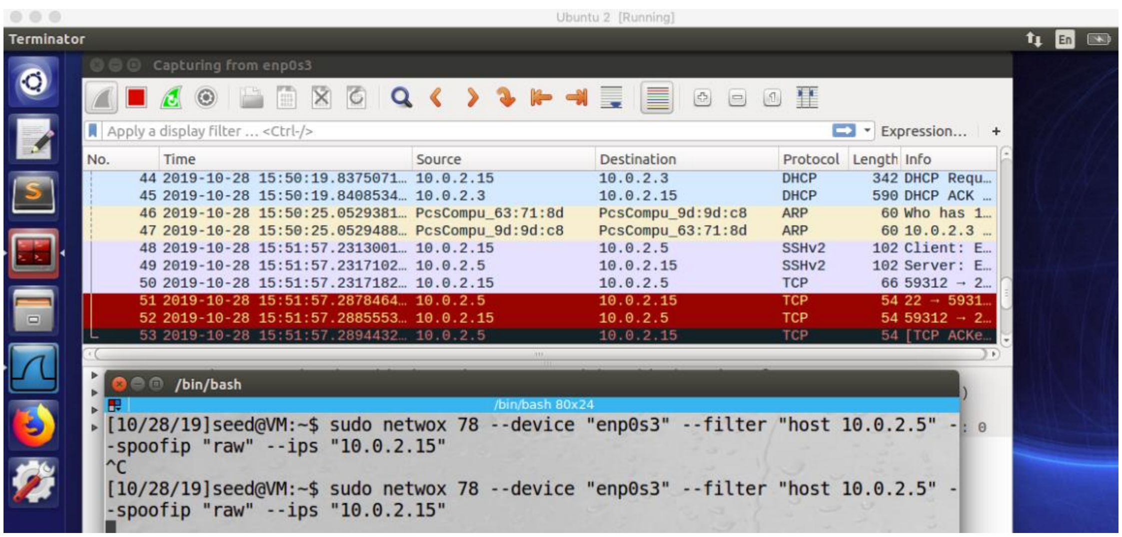

Creating an ssh connection between two machines 10.0.2.15 and 10.0.2.5



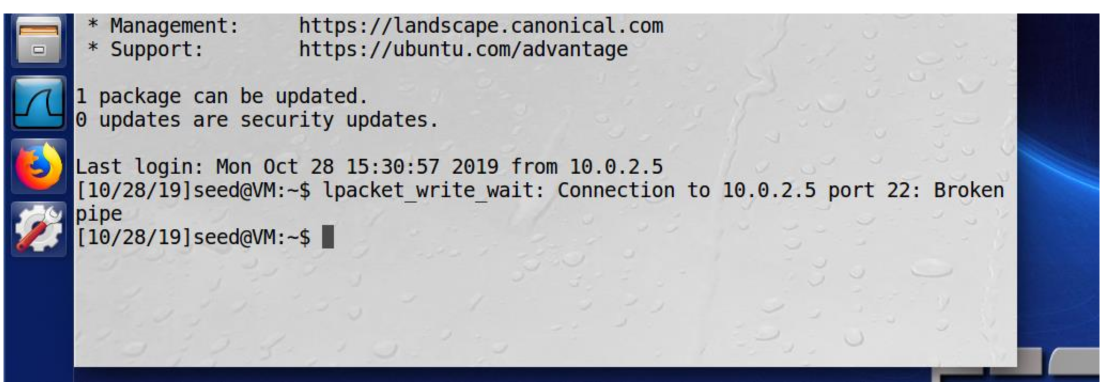
Observing ssh traffic between 10.02.5 and 10.0.2.15




Launching the attack using netwox 78



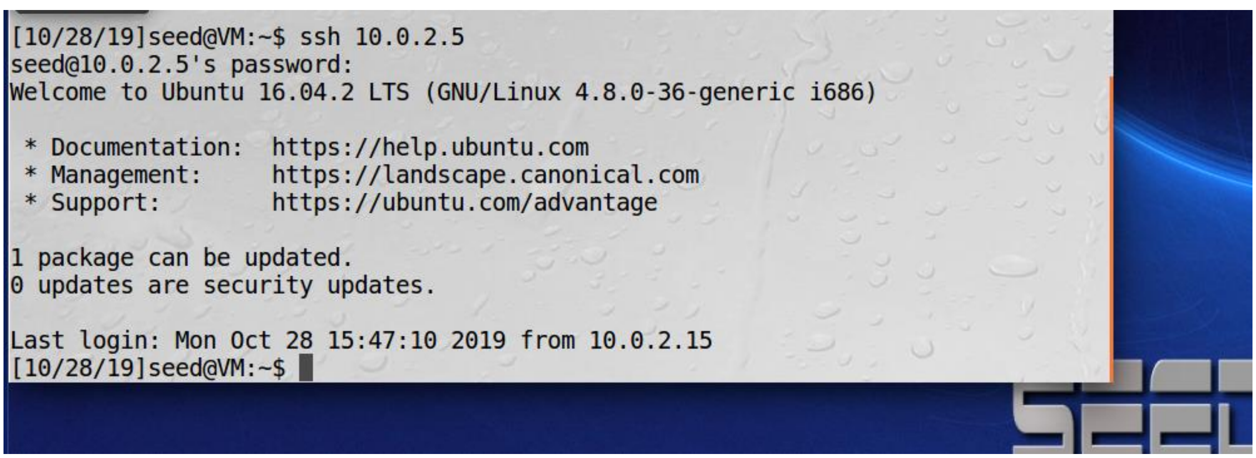

Observing broken pipe error on 10.0.2.15 connected to 10.0.2.5 via ssh



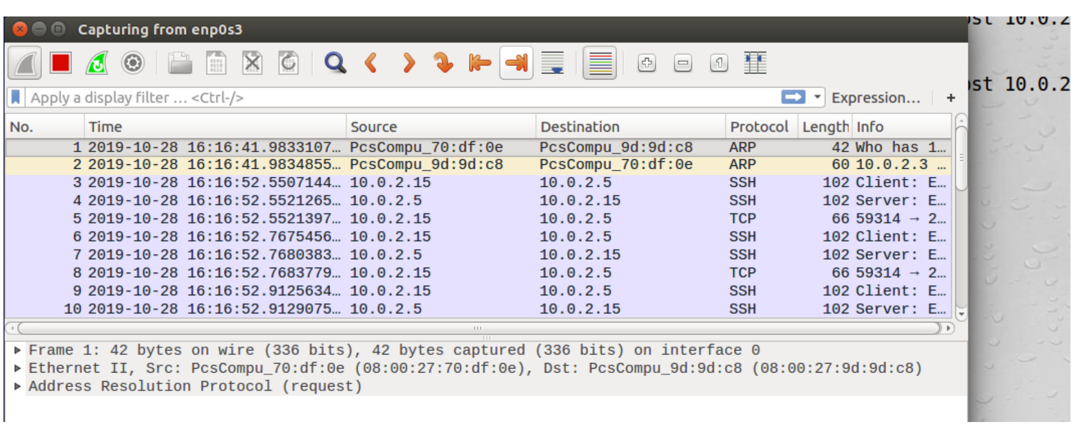
Using Scapy to perform the TCP RST Attack

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

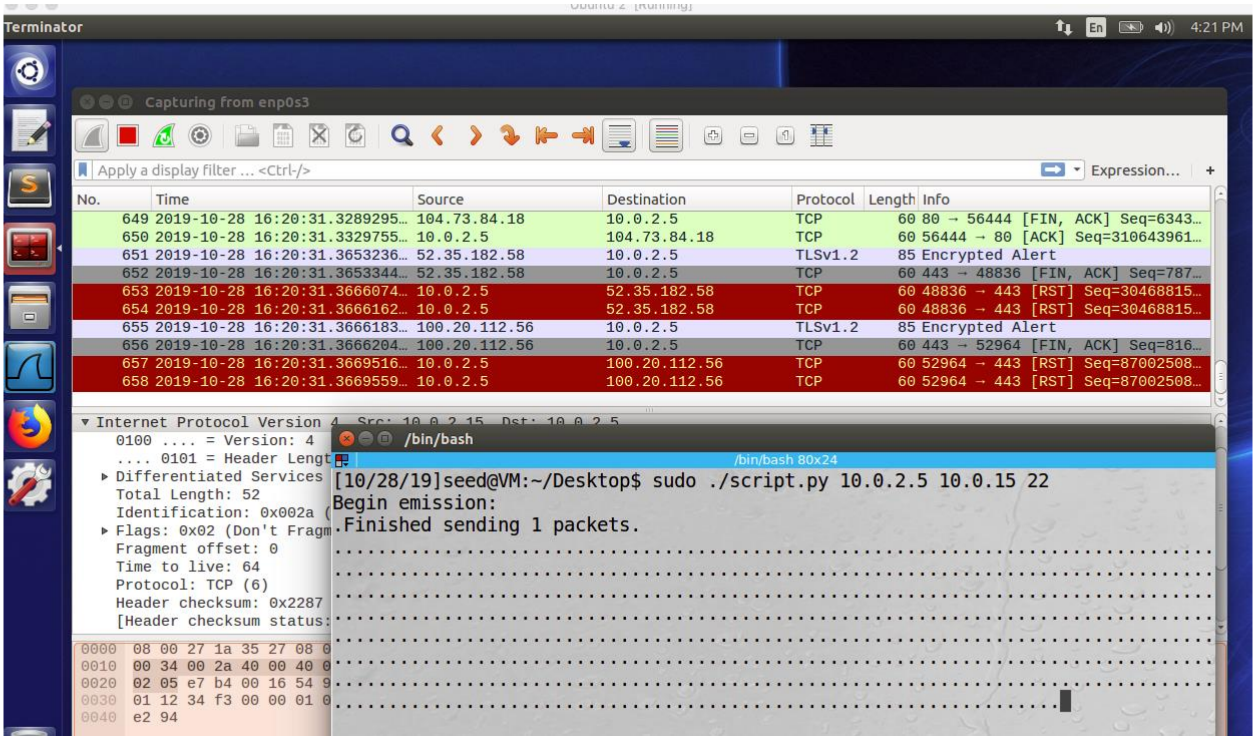

Establishing an ssh connection between 10.0.2.5 and 10.0.2.15




Observing the ssh traffic using wireshark



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

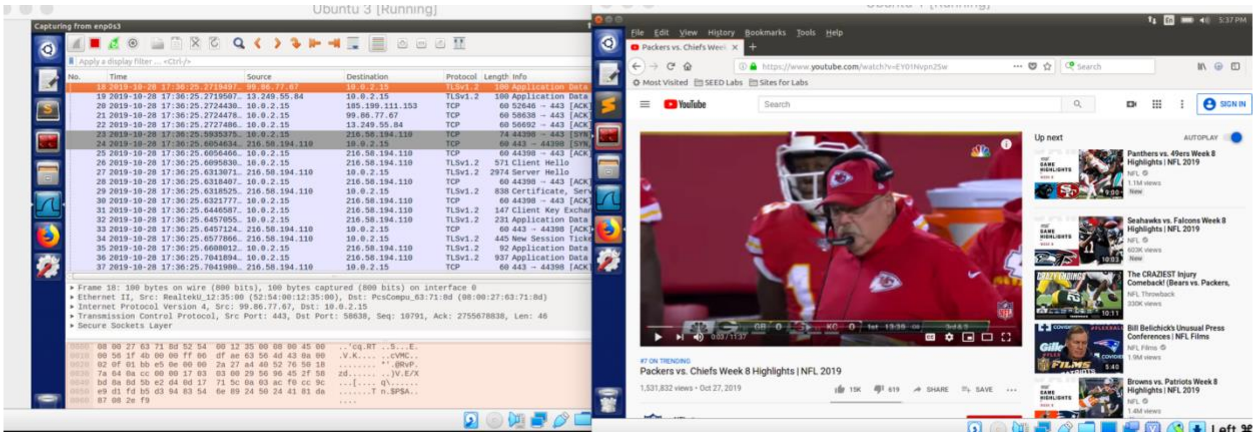
Defensive Thought

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 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.

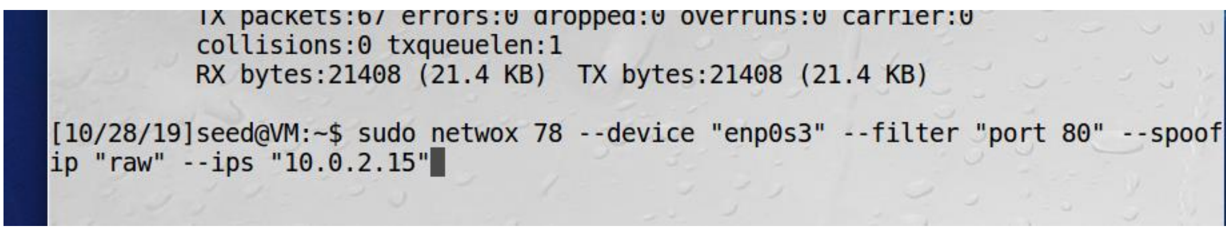
3.5 Task (5) : TCP RST Attacks on Video Streaming Applications

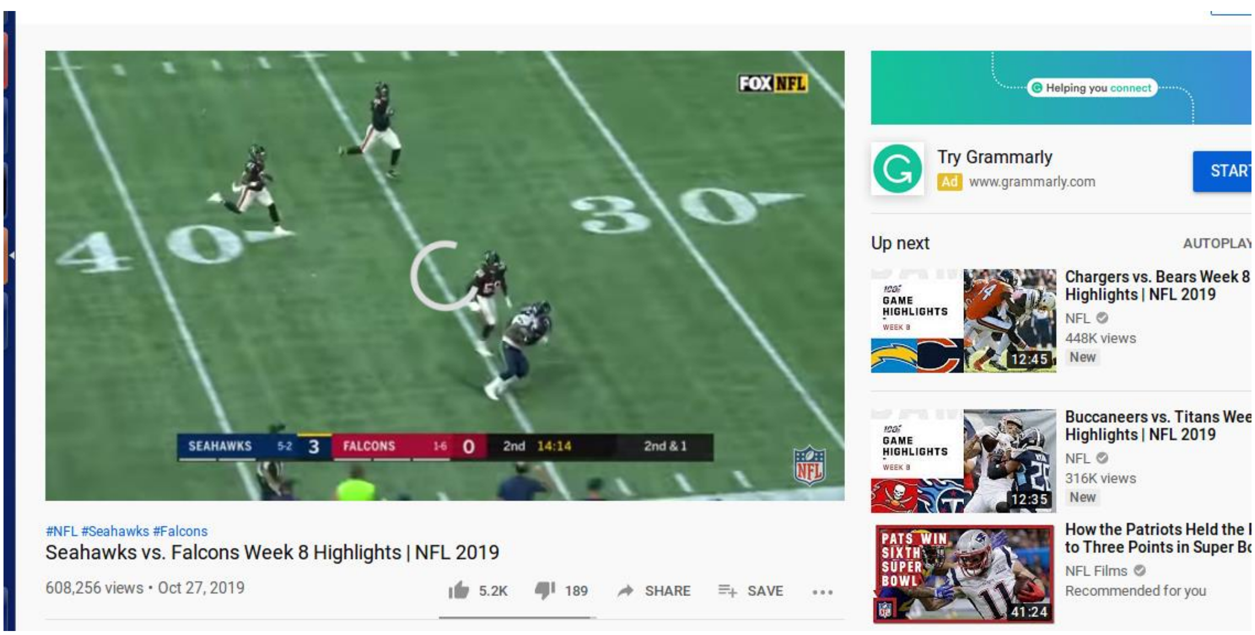
Establishing a TCP connection between the Victim and video streaming service. Observer observes this tcp connection through wireshark



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 vido 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

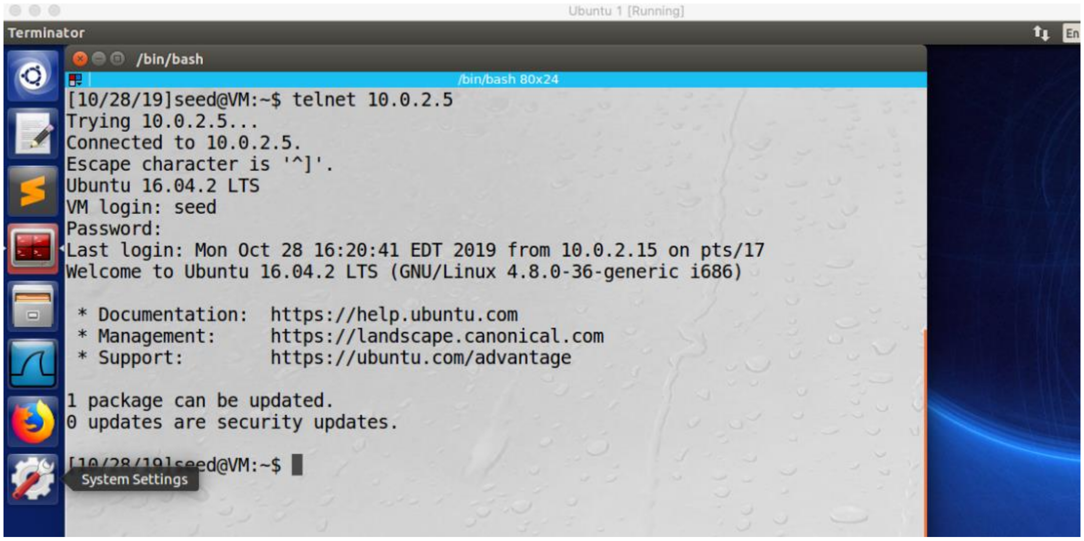
Defensive Thought

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.

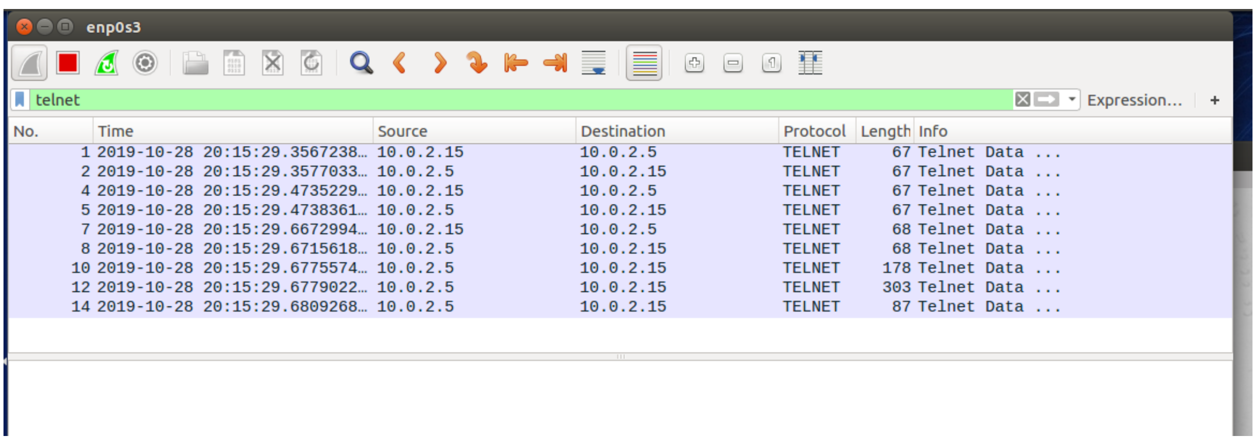
1. If it is recognized that the source sends the RST message, it can recognize the message as valid and shut down session on the device / router
2. If it is recognized that the source did not send the RST message, the message is flagged as invalid and the session will no be interrupted or shut down.

3.6 Task (6) : TCP Session Hijacking

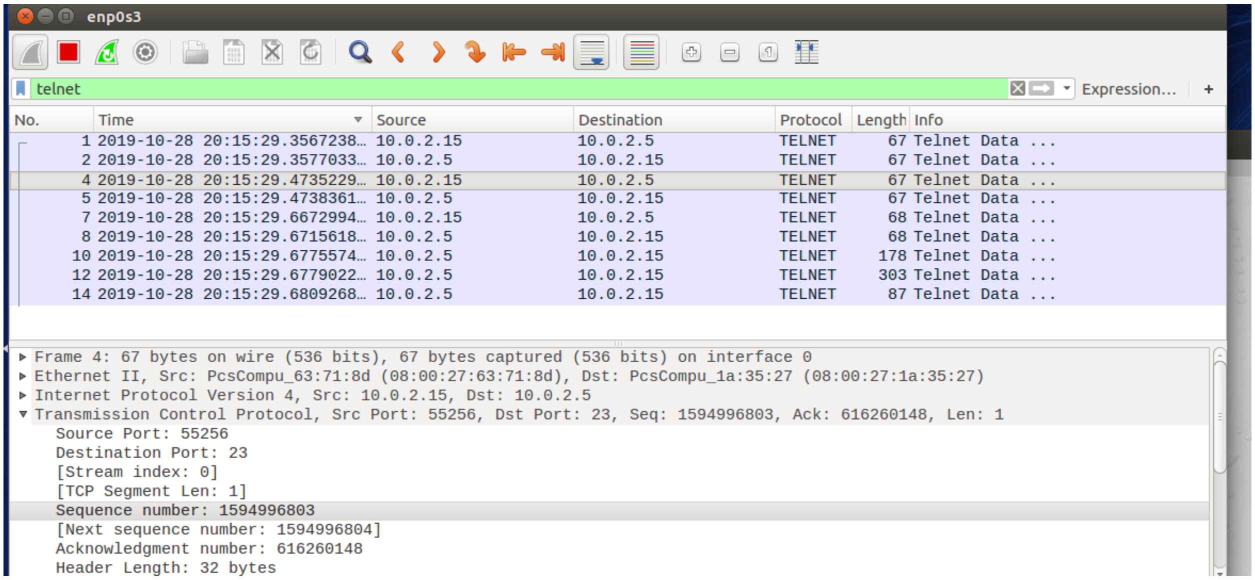
Establishing a telnet connection between hosts 10.0.2.5 and 10.0.2.15c




Observing the telnet traffic with Wireshark from the attacking machine



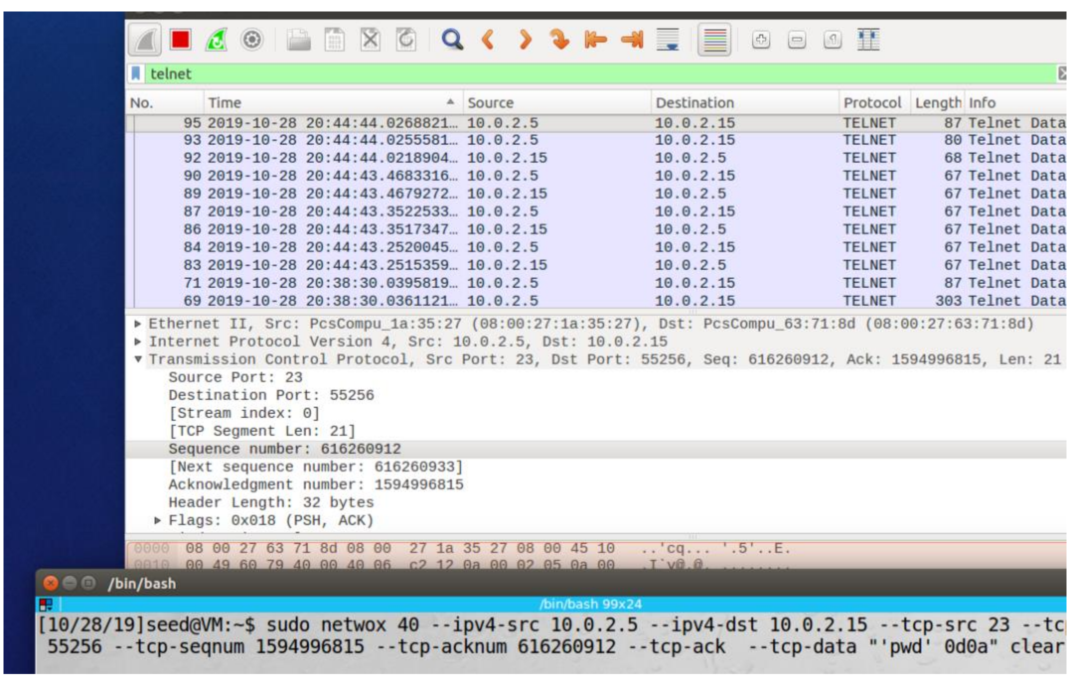
Breaking down the message sequences from the telnet traffic




FROM THE INITIAL PACKET CAPTURE
When source = 10.0.2.15
Source port = 55256
Initial sequence number = 1594996802
Next sequence number = 1594996803
Acknowledge 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



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

```
▸ Flags: 0x018 (PSH, ACK)
0000  08 00 27 63 71 8d 08 00  27 1a 35 27 08 00 45 10   ..'cq... '.5'..E.
0010  00 42 60 78 40 00 40 06  c2 1a 0a 00 02 05 0a 00   .B`x@.@. ........
0020  02 0f 00 17 d7 d8 24 bb  65 22 5f 11 b8 4f 80 18   ......$. e"_..O..
0030  00 e3 01 e3 00 00 01 01  08 0a 00 08 03 1e 00 08   ........ ........
0040  e8 8e 0d 0a 2f 68 6f 6d  65 2f 73 65 65 64 0d 0a   ..../hom e/seed..
```

Now, the telnet session will attempt to be hijacked using scapy

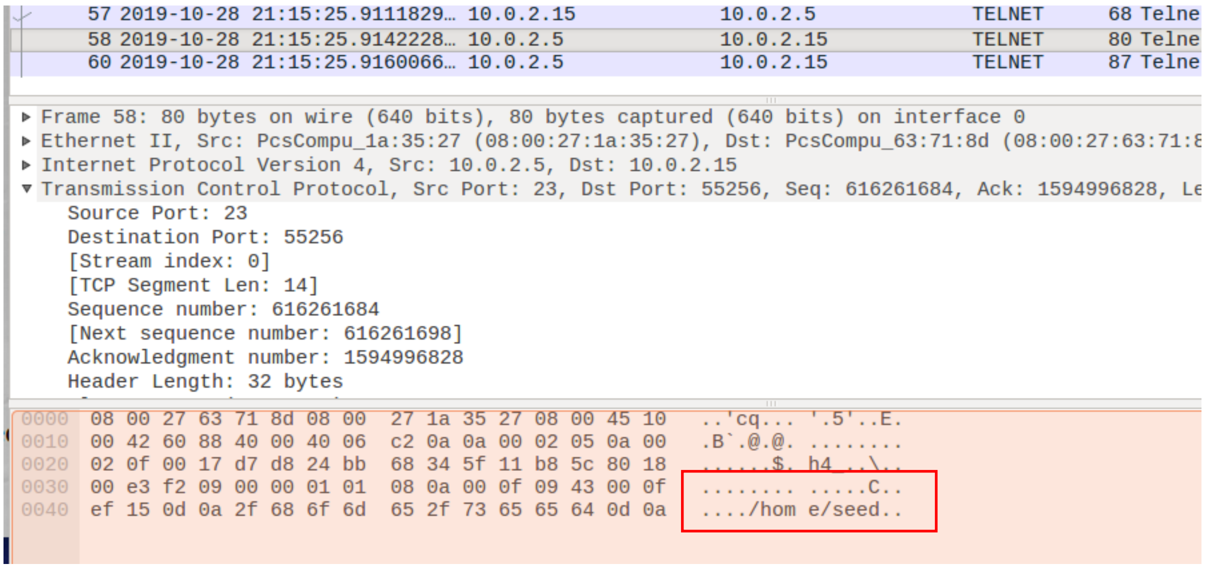
Constructing the scapy script in python

```python
#!/usr/bin/python
from scapy.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

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




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.