TCP ATTACK

Make a Network

Name of VM

- seedubuntu
- seedubuntu1
- seedubuntu2

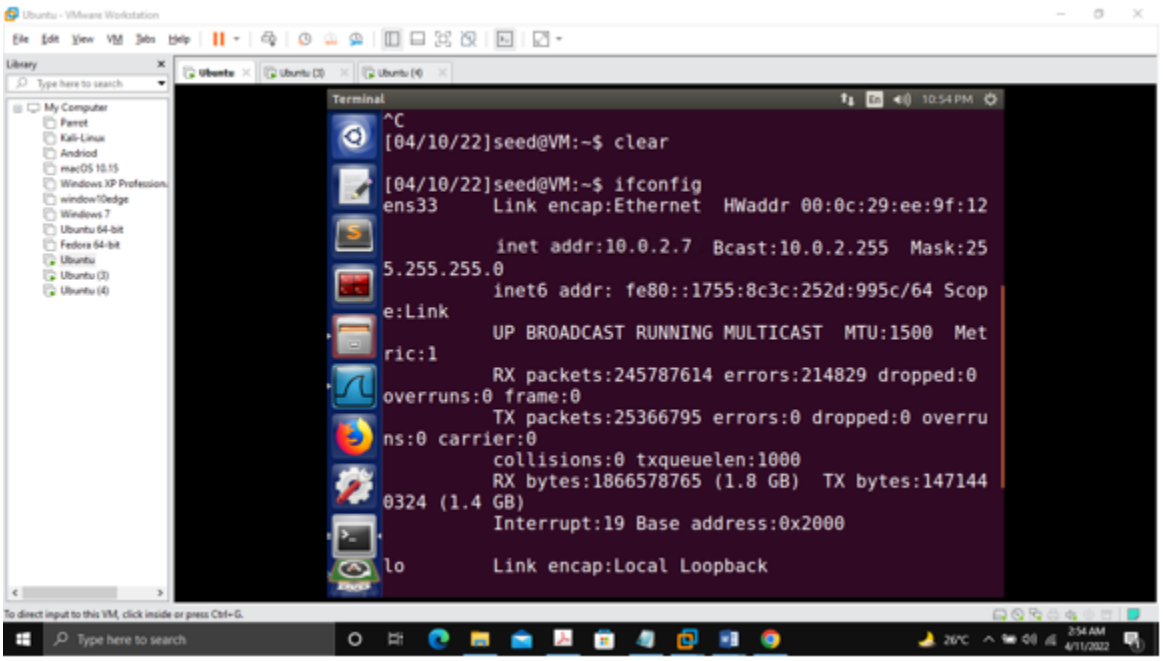
Role

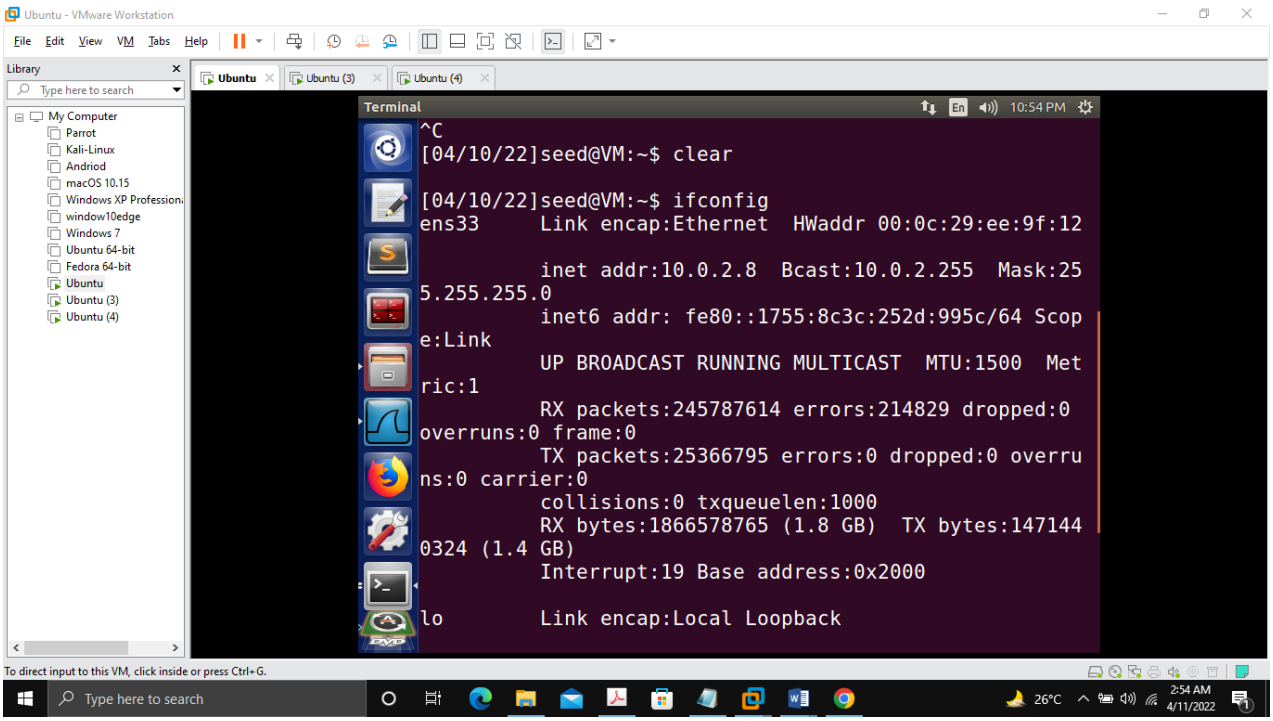
- Attacker
- Victim • client

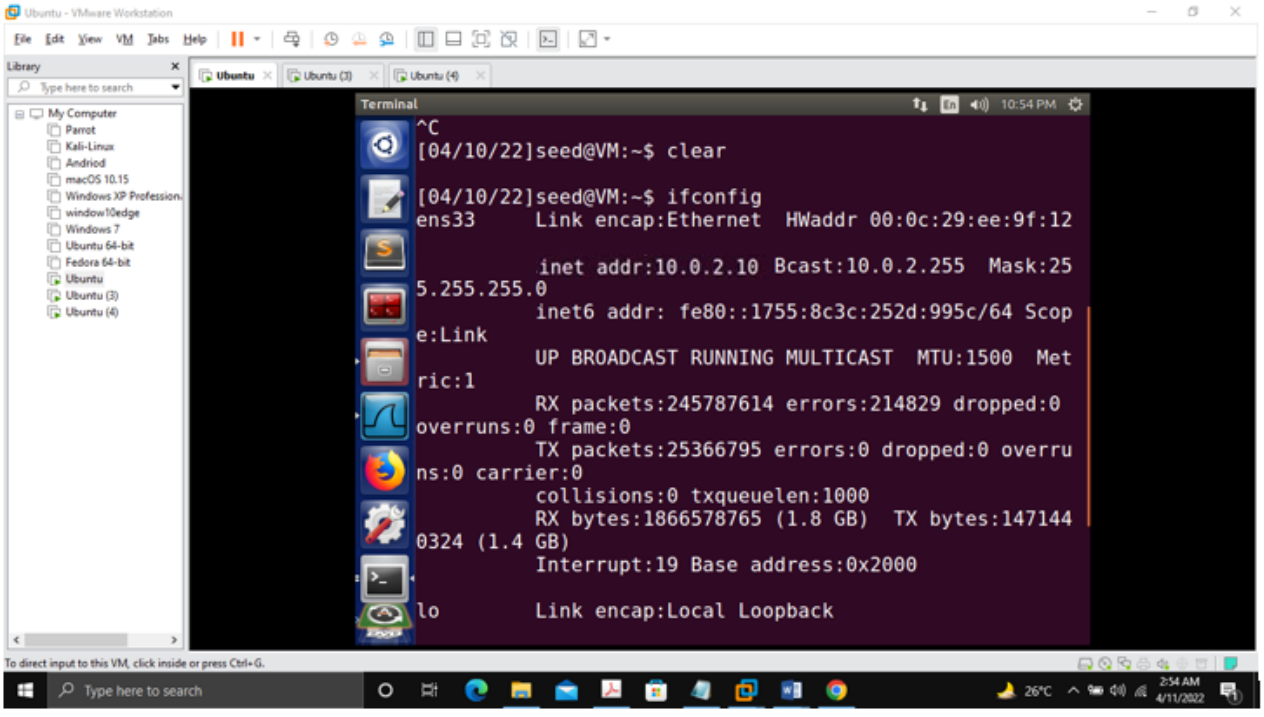
ip address

- 10.0.2.7
- 10.0.2.8
- 10.0.2.10

Ip configuration

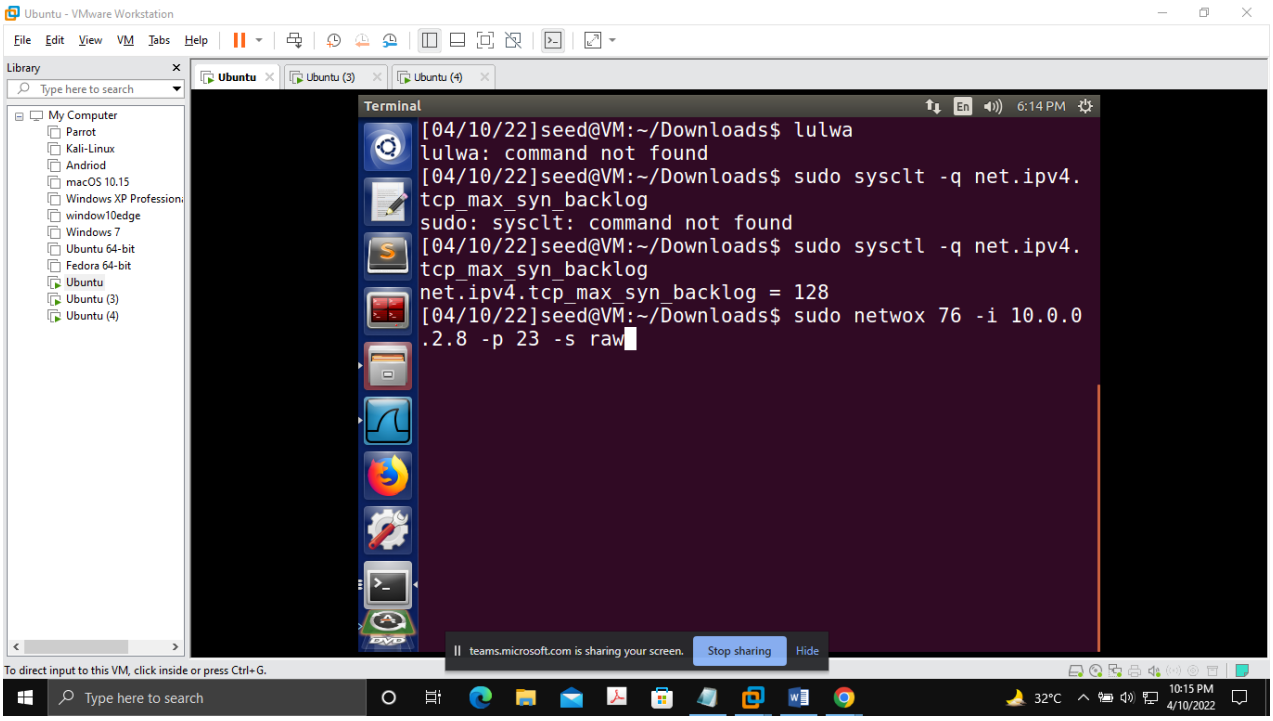



Ubuntu - VMware Workstation
File Edit View VM Tabs Help
Library
Type here to search
My Computer
Parrot
Kali-Linux
Andriod
macOS 10.15
Windows XP Profession
window10edge
Windows 7
Ubuntu 64-bit
Fedora 64-bit
Ubuntu
Ubuntu (3)
Ubuntu (4)
Ubuntu
Ubuntu (3)
Ubuntu (4)
Terminal
10:54 PM
^C
[04/10/22]seed@VM:~$ clear
[04/10/22]seed@VM:~$ ifconfig
ens33 Link encap:Ethernet HWaddr 00:0c:29:ee:9f:12
inet addr:10.0.2.8 Bcast:10.0.2.255 Mask:25
5.255.255.0
inet6 addr: fe80::1755:8c3c:252d:995c/64 Scop
e:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Met
ric:1
RX packets:245787614 errors:214829 dropped:0
overruns:0 frame:0
TX packets:25366795 errors:0 dropped:0 overru
ns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:1866578765 (1.8 GB) TX bytes:147144
0324 (1.4 GB)
Interrupt:19 Base address:0x2000
lo Link encap:Local Loopback
To direct input to this VM, click inside or press Ctrl+G.
Type here to search
26°C
2:54 AM
4/11/2022

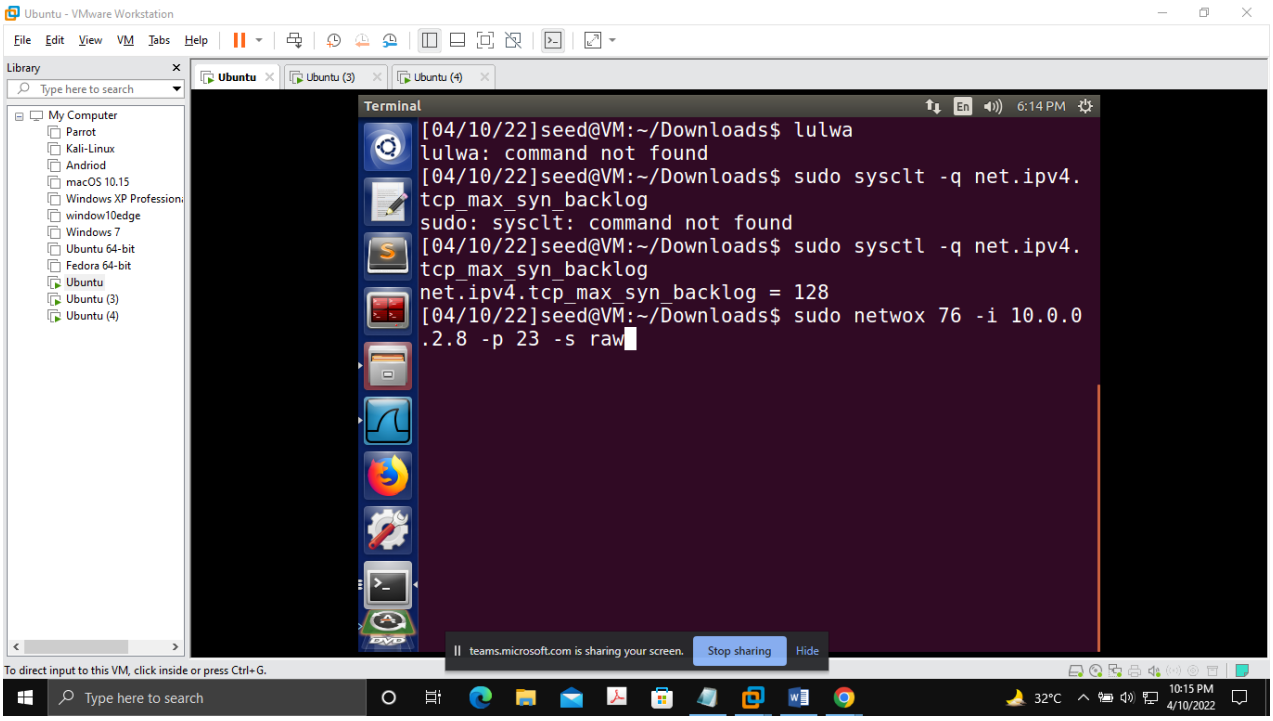

Ubuntu - VMware Workstation
File Edit View VM Tabs Help
Library
Type here to search
My Computer
Parrot
Kali-Linux
Andriod
macOS 10.15
Windows XP Profession
window10edge
Windows 7
Ubuntu 64-bit
Fedora 64-bit
Ubuntu
Ubuntu (3)
Ubuntu (4)
Ubuntu
Ubuntu (3)
Ubuntu (4)
Terminal
10:54 PM
^C
[04/10/22]seed@VM:~$ clear
[04/10/22]seed@VM:~$ ifconfig
ens33 Link encap:Ethernet HWaddr 00:0c:29:ee:9f:12
inet addr:10.0.2.10 Bcast:10.0.2.255 Mask:25
5.255.255.0
inet6 addr: fe80::1755:8c3c:252d:995c/64 Scop
e:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Met
ric:1
RX packets:245787614 errors:214829 dropped:0
overruns:0 frame:0
TX packets:25366795 errors:0 dropped:0 overru
ns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:1866578765 (1.8 GB) TX bytes:147144
0324 (1.4 GB)
Interrupt:19 Base address:0x2000
lo Link encap:Local Loopback
To direct input to this VM, click inside or press Ctrl+G.
Type here to search
26°C
2:54 AM
4/11/2022

Task 1: SYN Flooding Attack

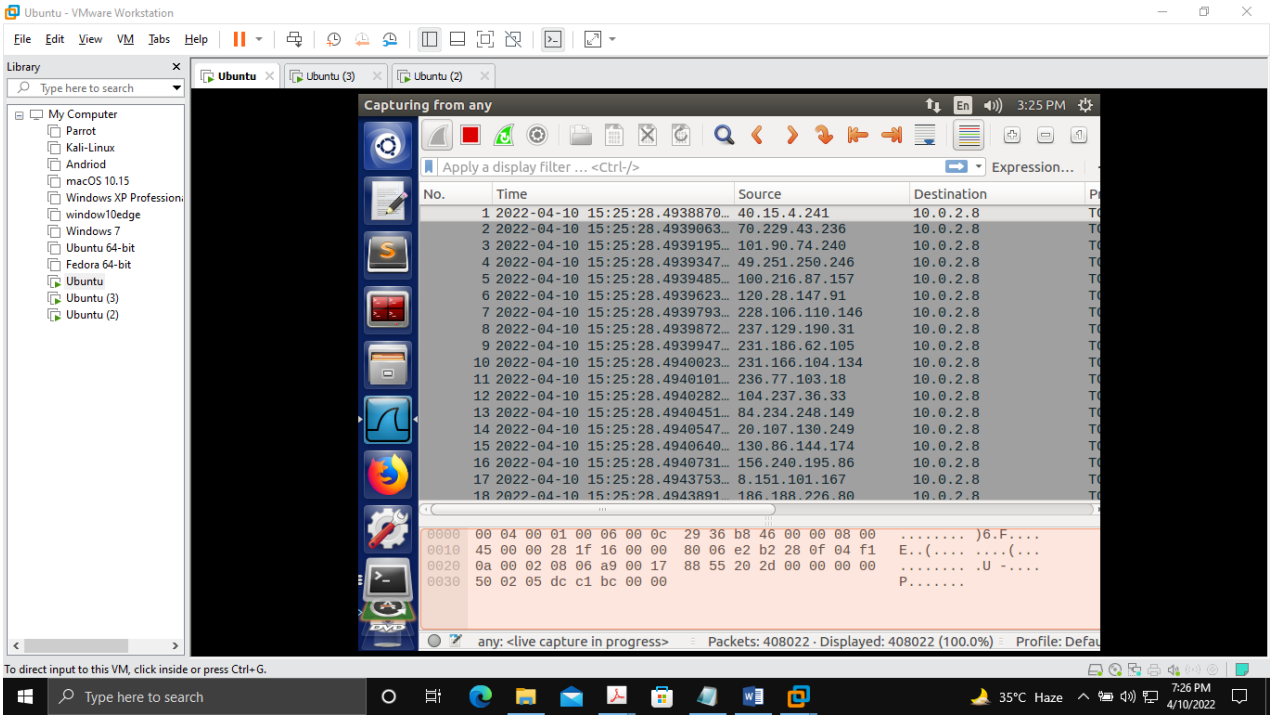
As seen in the screenshot, the victim's queue size is 128. We also see the current open ports that are awaiting connections (LISTEN stage.) If a port had a half-open connection (only SYN received and no ACK from the client), then the state would've been SYN_RECV. If the 3-way handshake completes, the state changes to ESTABLISHED.



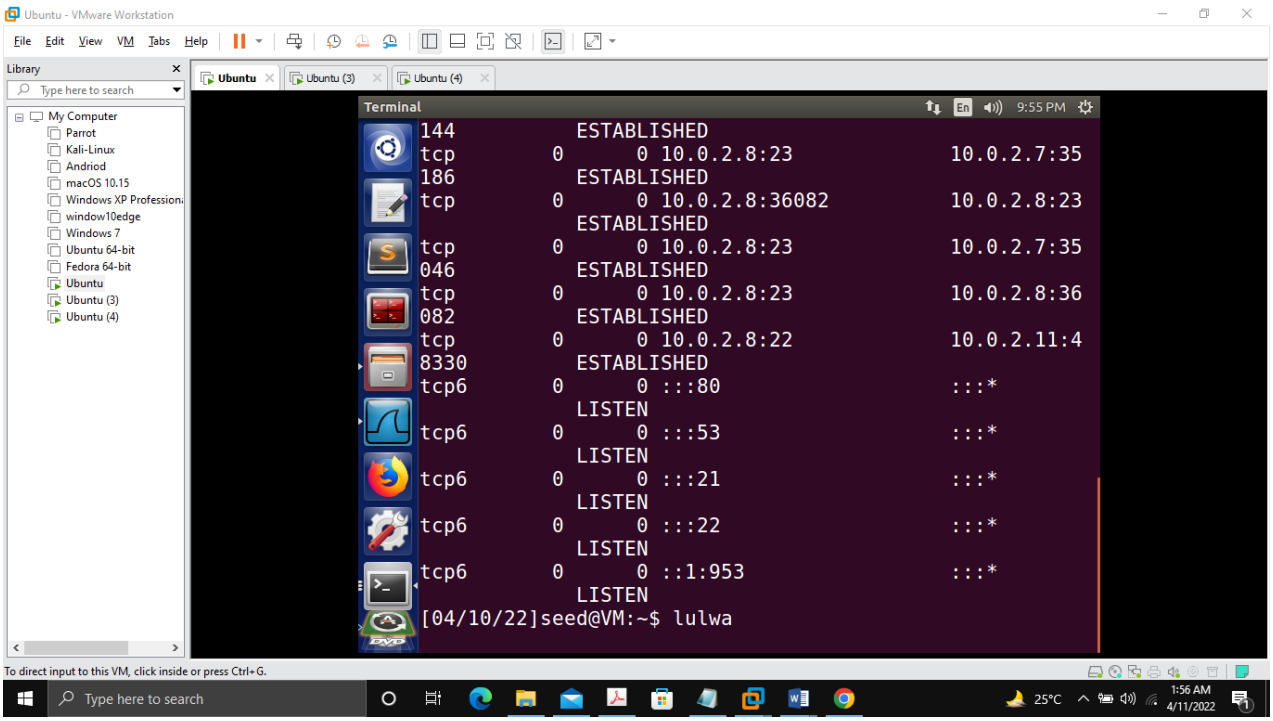
Now, in order to perform the SYN flooding attack, we run the netwox tool with task number 76:



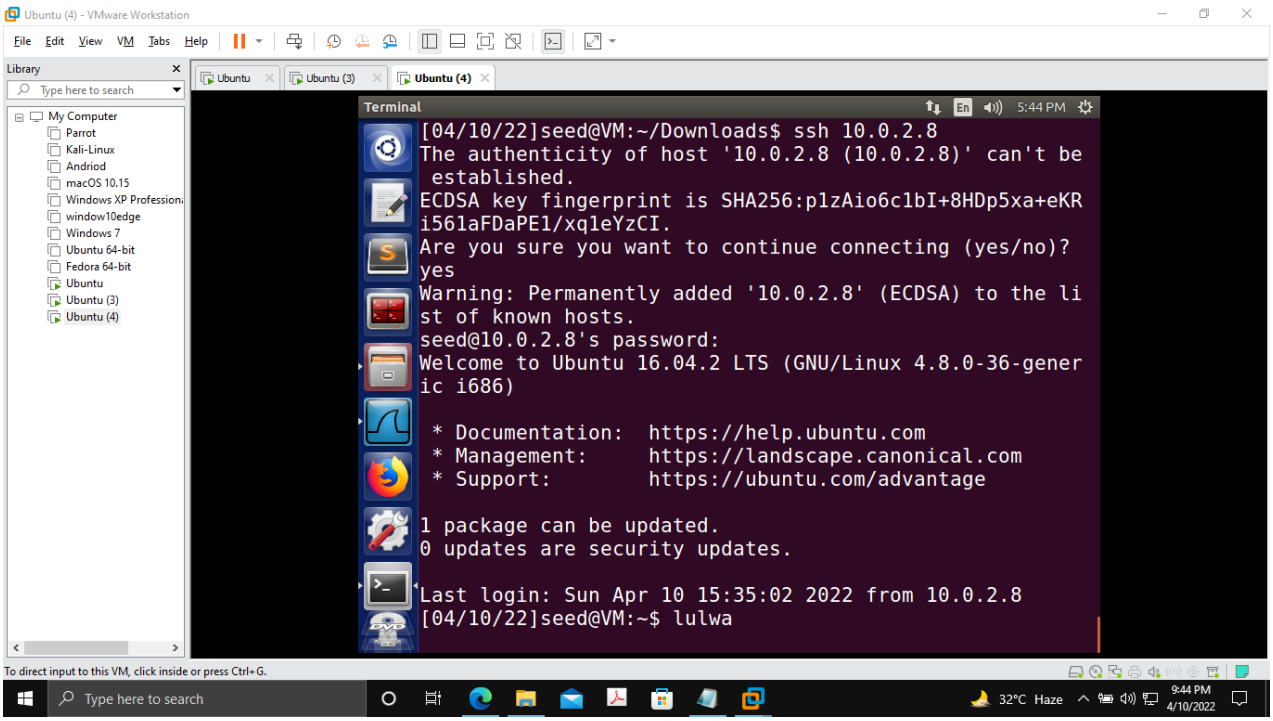
The Wireshark trace for the attack is given below. We see that the victim machine receives numerous numbers of connection on port 23 from random IP addresses (spoofed by the netwox tool.) We also see that the victim machine replies these IP addresses with a SYN ACK initially. Soon there are RST ACK packets visible on the network. This is because the host with the source IP is alive and realizes that it had never started a connection in the first place in order to receive a SYN ACK. If the victim machine receives these RST packets, the entry is removed from the queue because it is no more a half-open connection. Even though some spoofed connections are retrieved from the queue, many other half-open connections are established by the tool continuously, as seen:



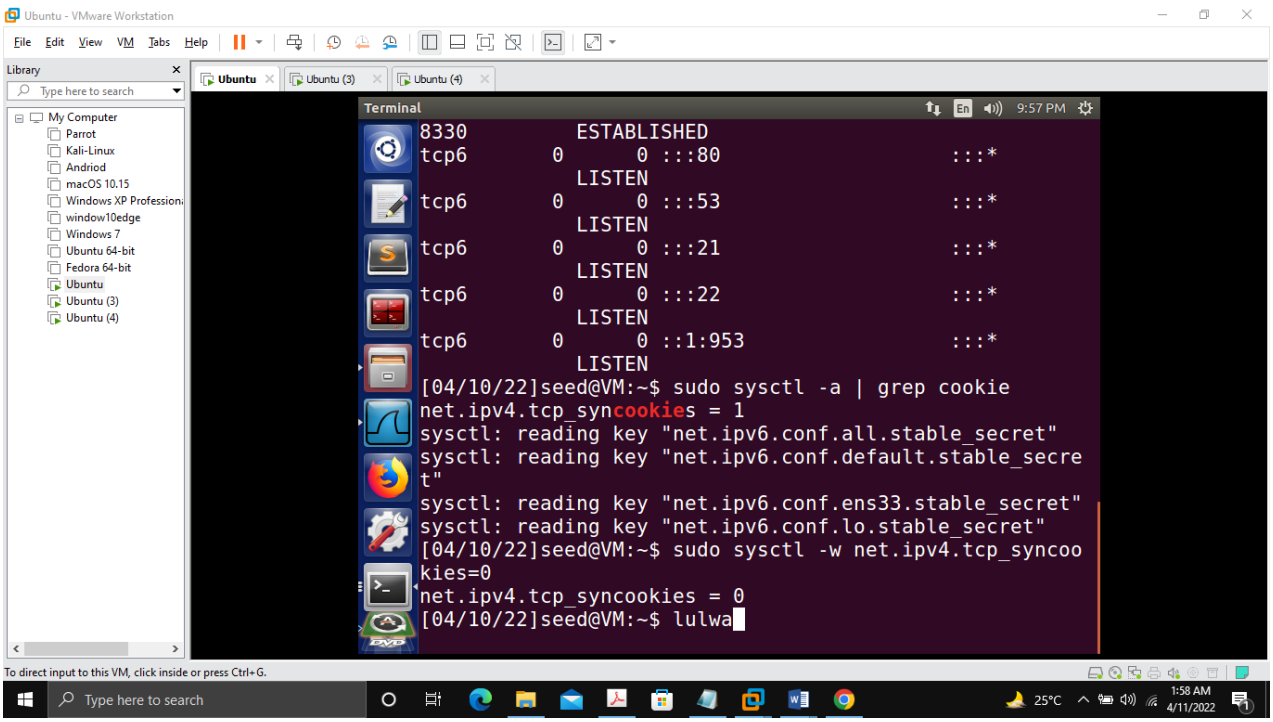
Now on seeing the network statistics on the victim machine, we see that multiple connections have the state as SYN_RECV, indicating half-open connections:



In order to see if our attack was successful, we try to initiate a legit telnet connection to the server i.e. the victim. If the attack is successful, then the telnet connection will not be established because the entire queue is filled with spoofed half-open connection, hence it will not accept any new connections. We see that, we were easily able to connect to the server:

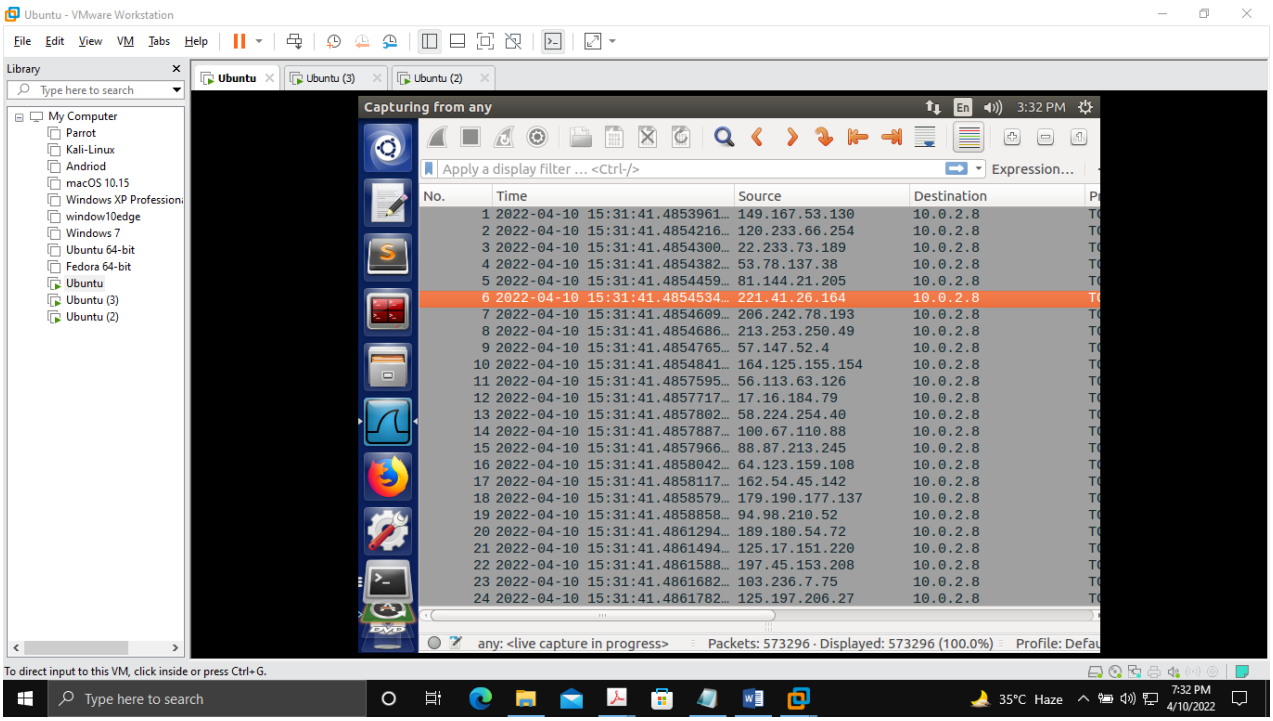


This indicates that the attack was not successful, and the Server was not a victim of SYN flooding. Now, we check if the SYN Cookie mechanism i.e. defense mechanism to counter SYN flooding, is turned on. We see that it is indeed on and hence our attack might have been unsuccessful. We turn off this mechanism and try the attack again.

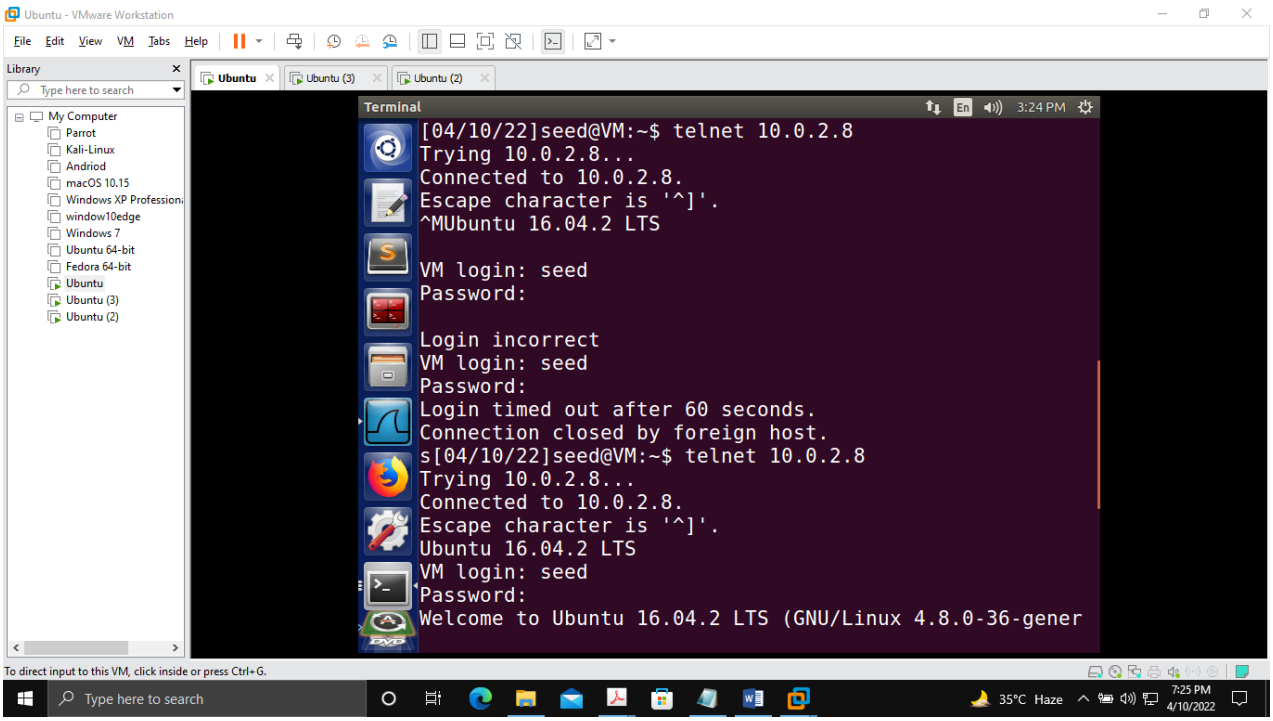


On performing the attack again, we see that the network statistics changes again from LISTEN state to multiple SYN_RECV state. This indicates that multiple half-open connections are established.

Now, the Wireshark trace of the attack looks similar to the one seen before with multiple SYN packets going from random IP addresses to the victim machine on port 23. Also, we see some RST ACK going from the spoofed source IP to the victim indicating that they had never started the connection and wants the connection closed. This will remove the entry from the queue.



Now, in order to check if our attack was successful, we try to start a telnet connection from the client machine to the server i.e. the victim. We see that the connection is not established and there is a time out. This indicates that our attack was successful.



We notice that the attack was not successful when SYN cookie was turned on. The SYN cookie can effectively prevent the server from SYN flood attack because it does not allocate resources when it receives the SYN packet, it allocates resources only if the server receives the final ACK packet. This prevents from having the queue as a bottleneck, and instead consume resources only for the established connections.

SYN cookies also prevents an ACK flood attack (since it's now consuming resources for ACK packet received), by calculating an initial sequence number using a key (known only to the server) on certain parameters of the received SYN packet and sending it in SYN ACK packet. This sequence number + 1 is sent back in the ACK packet in the acknowledgment field. The server verifies the acknowledgement number and ensures that it was a result of a SYN ACK packet. Since the server is the only one who knows the key calculating the value, it restricts the attackers from having a valid SYN cookie i.e. initial sequence number from the server to client. This prevents any system from the SYN flood attacks.

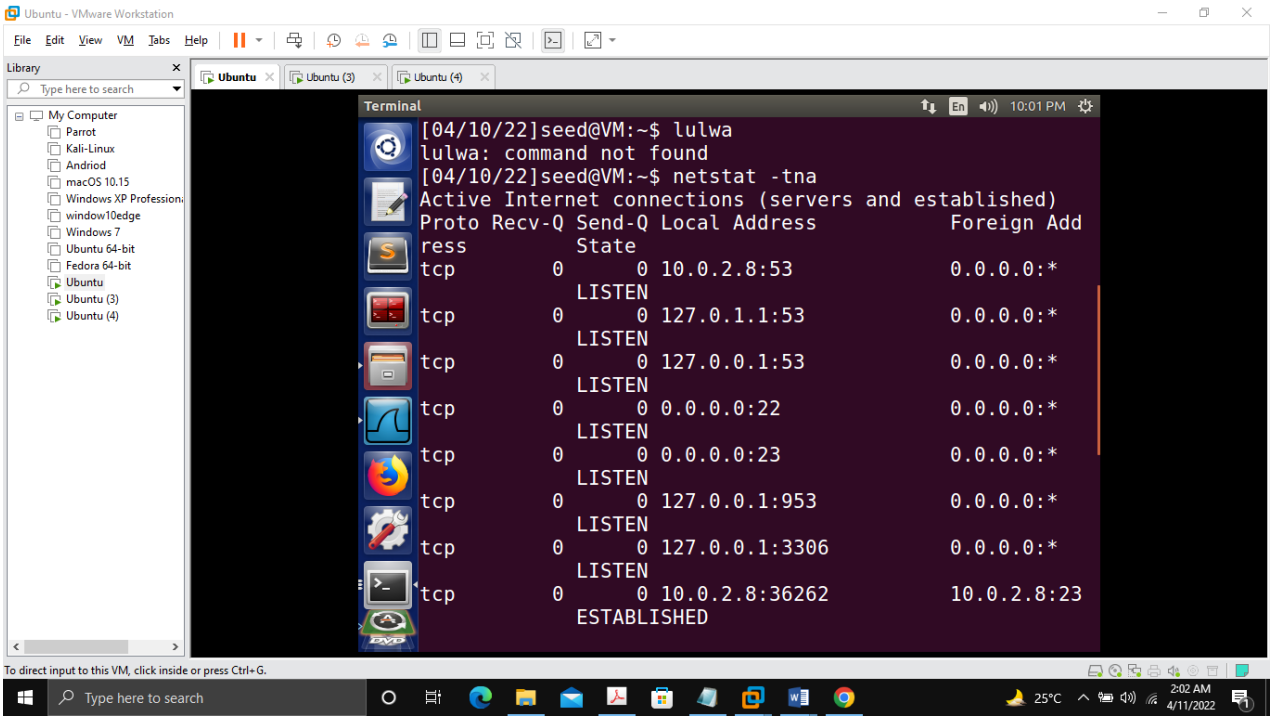
Task 2: TCP RST Attacks on telnet and SSH Connections

Breaking a Telnet connection:

Server i.e. 10.0.2.8 has the telnet port open and in the LISTEN state.

Using Netwox:

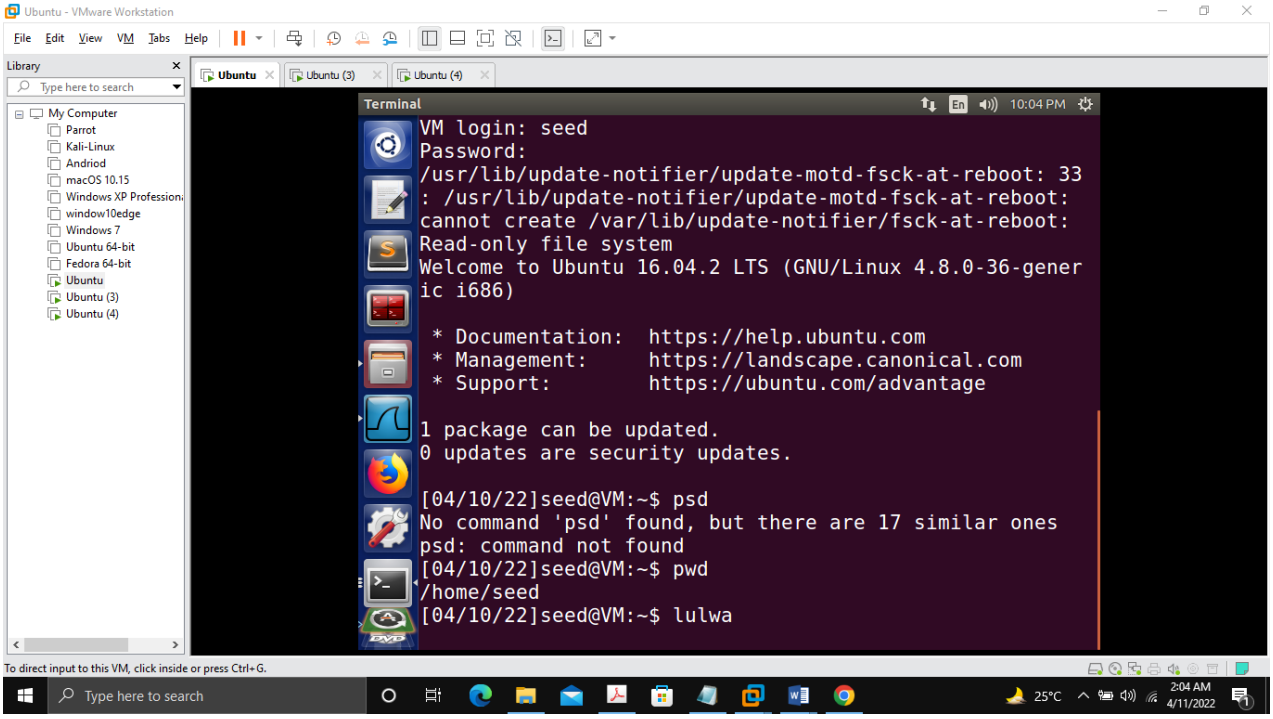
We establish a telnet connection from the client 10.0.2.10 (A) to the server 10.0.2.8 (B):



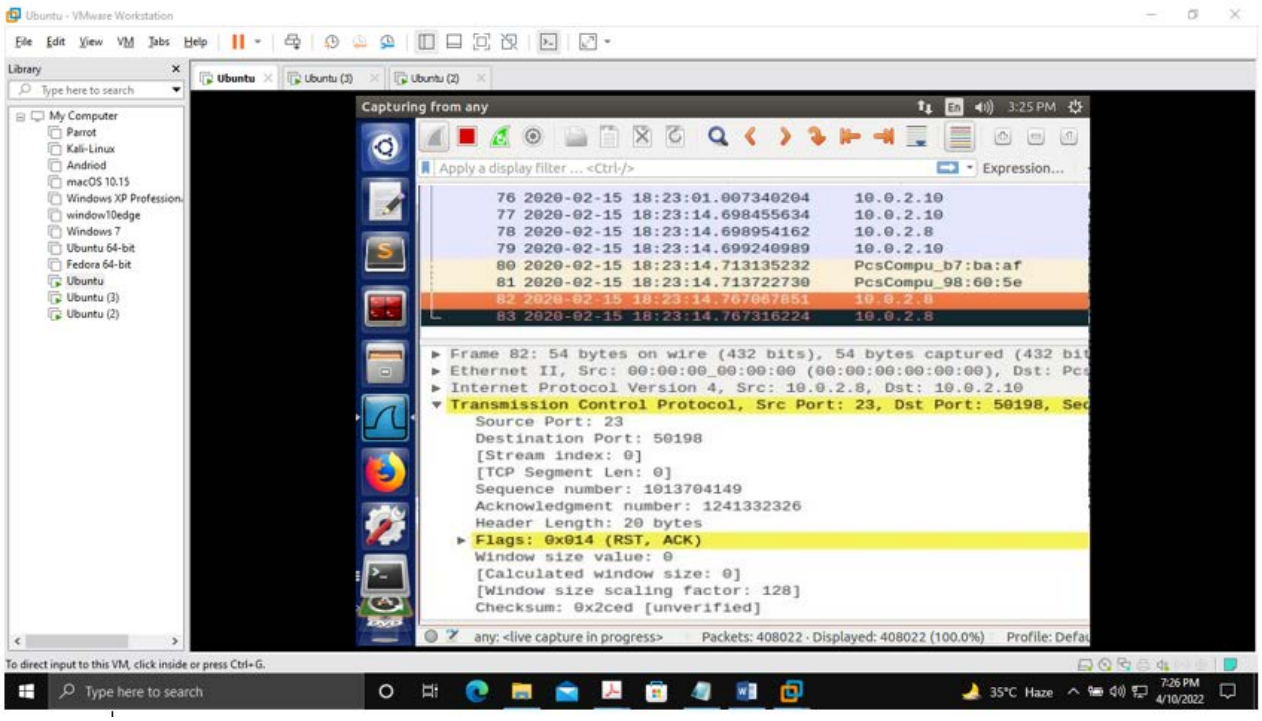
We then use the netwox tool on the Attacker's machine to launch the RST Attack using the following:

sudo netwox 78 --filter "src host 10.0.2.10 and dst port 23"

The above command sends an RST packet as soon as something is sent from A to B on the telnet connection. After establishing the connection and entering a pwd command once, we run the above command. Then we again start typing pwd and see the following on A:



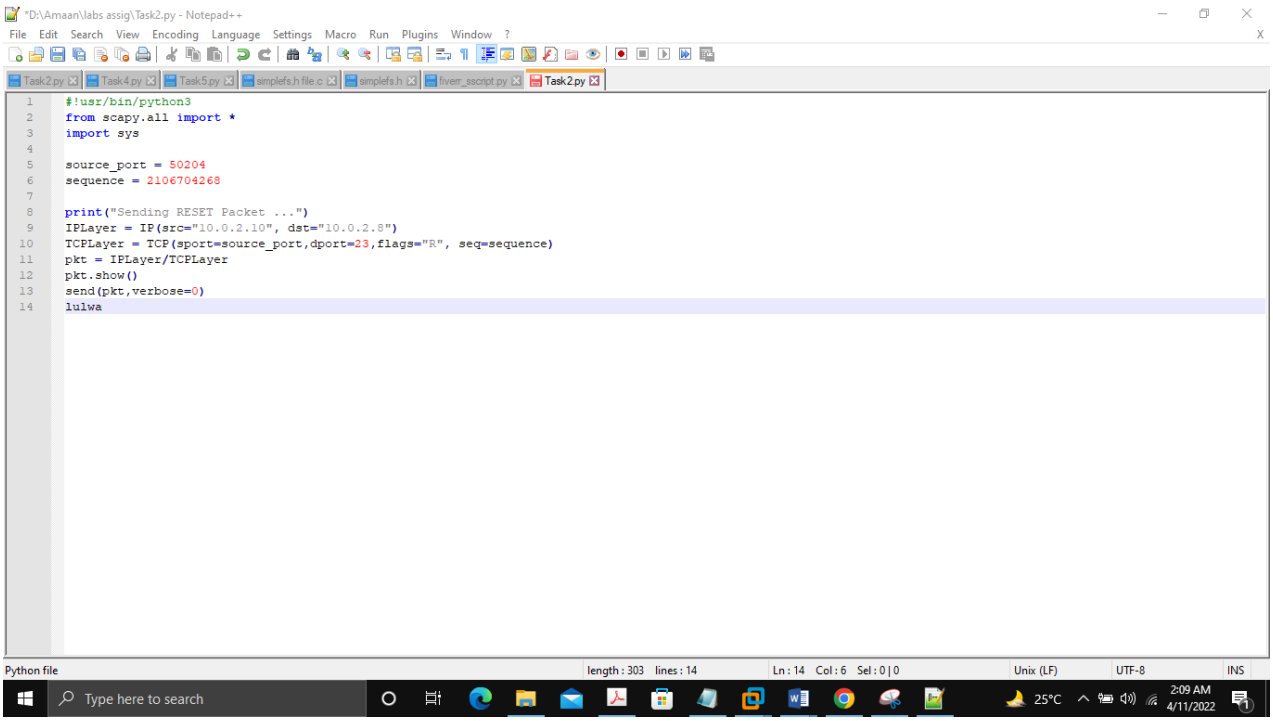
The following Wireshark trace shows the spoofed RST packet from B to A:



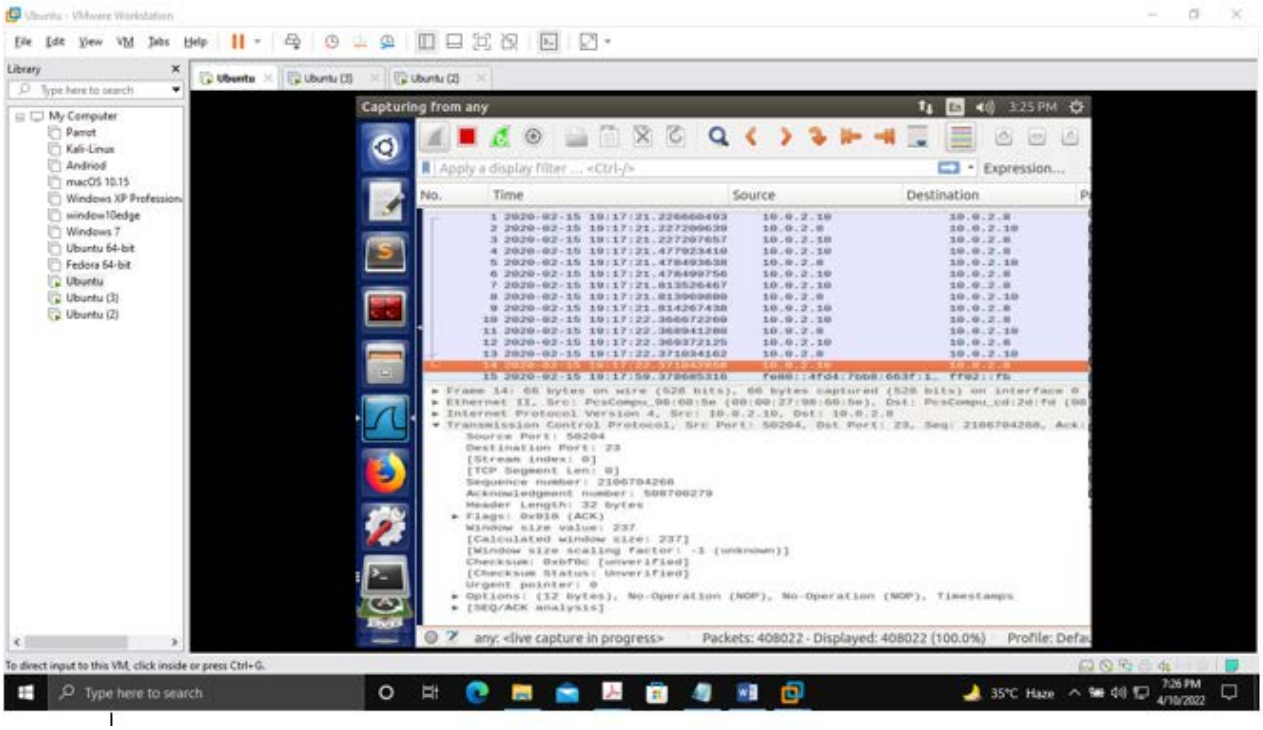
This indicates that we were able to close an established connection between A and B by spoofing an RST packet from B to A.

Using Scapy:

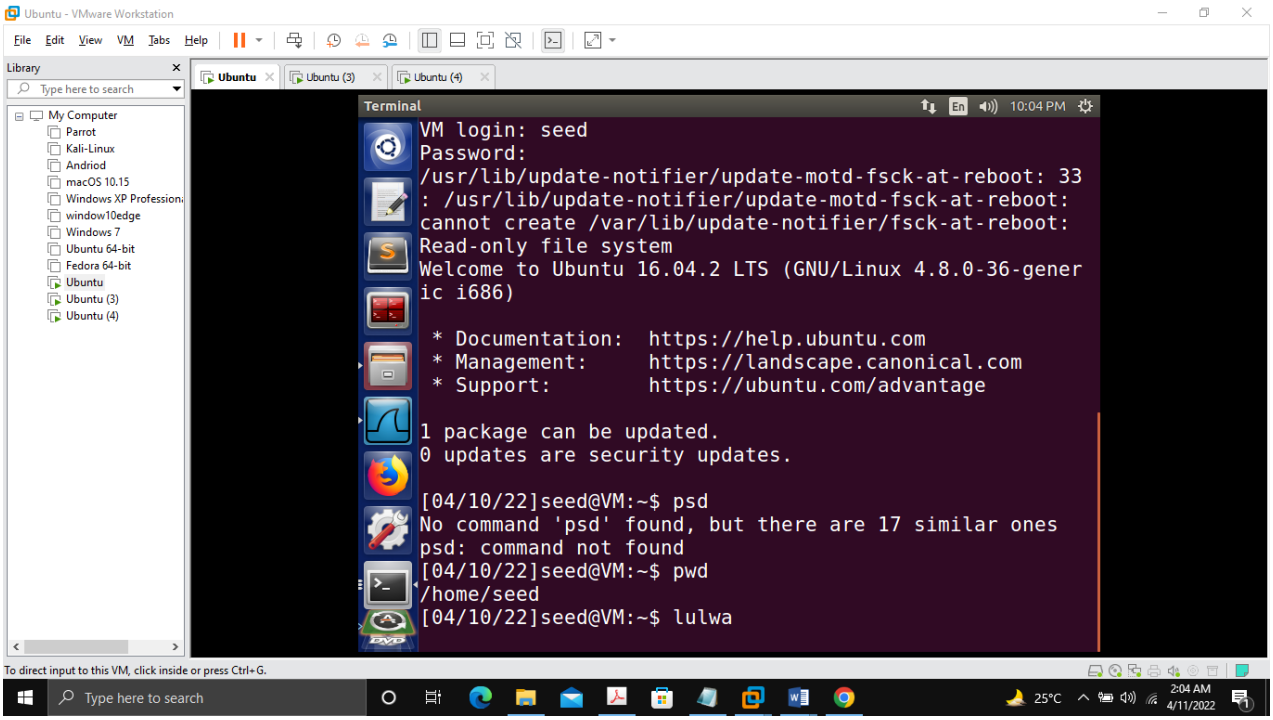
Now we perform the same RST Attack on a telnet connection using the following scapy program:



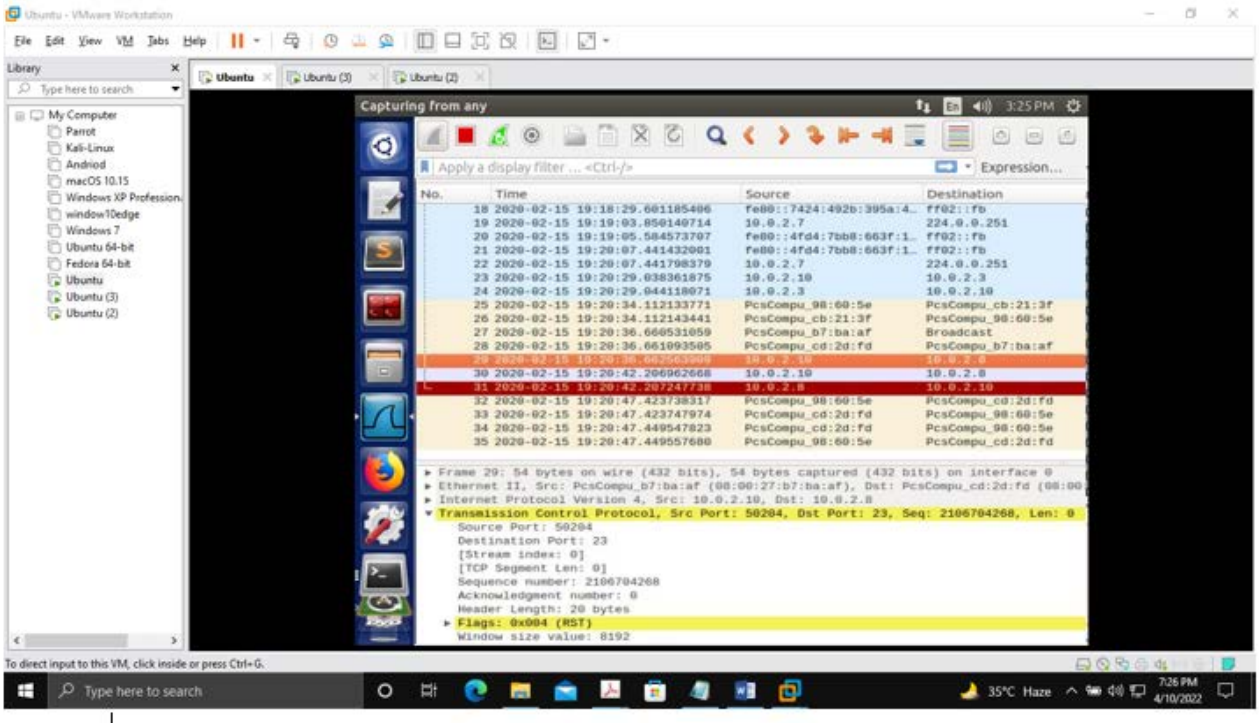
After establishing the connection and verifying the established connection by sending a pwd command, we sniff the network to find the sequence number and source port of the last sent packet from 10.0.2.10 (A) to 10.0.2.8 (B):



In order for our attack to be successful, we need to make sure that the sequence number is exactly what is next expected by the server or else our attack will fail. Then we run the program on the attacker machine and see that the connection closes on the client machine:



The following shows that an RST packet is sent from A to B and the source MAC address is of the Attacker. This proves that we were able to successfully perform an RST attack:



Hence, we were able to successfully launch a TCP RST attack on a telnet connection using netwox tool and scapy.

Breaking an SSH connection:

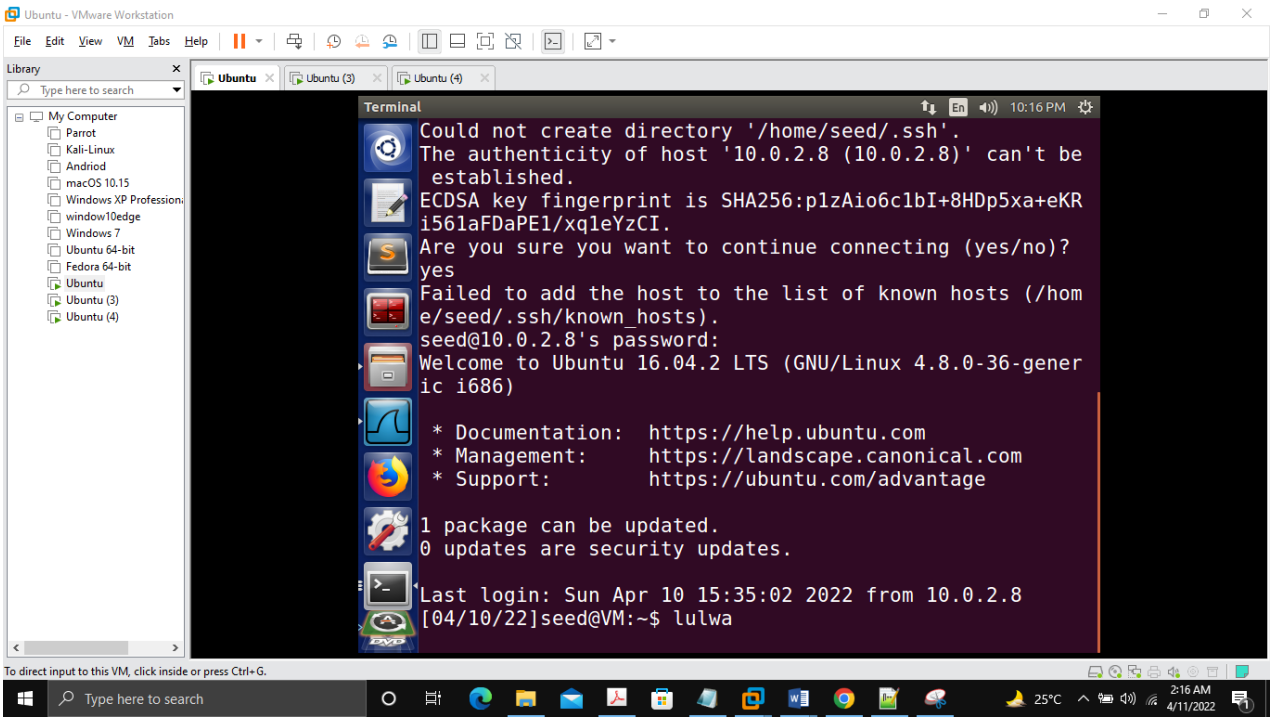
Server i.e. 10.0.2.8 has the SSH port open and in the LISTEN state.

Using Netwox:

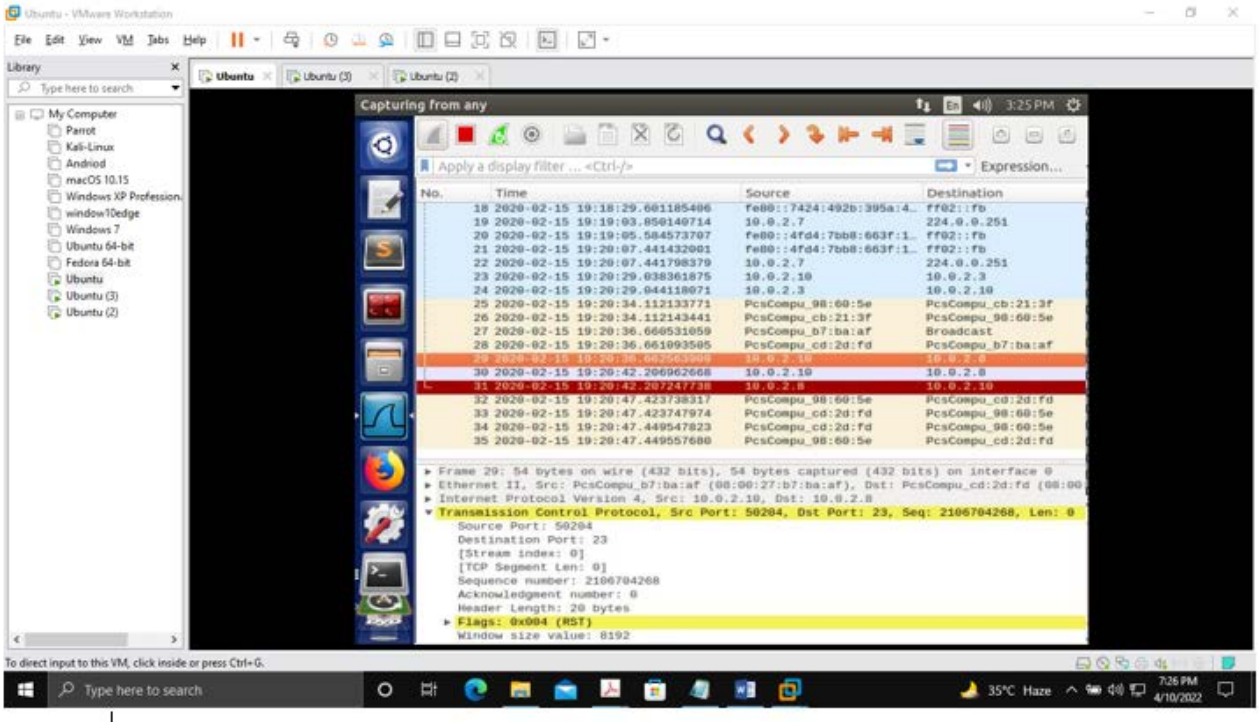
We establish an SSH connection from the client 10.0.2.10 (A) to the server 10.0.2.8 (B). We then use the netwox tool on the Attacker's machine to launch the RST Attack using the following command:

sudo netwox 78 --filter "src host 10.0.2.10 and dst port 22"

The above command sends an RST packet as soon as something is sent from A to B on the SSH connection. After establishing the connection and entering a pwd command once, we run the above command. Then we again start typing pwd and see the following on A:



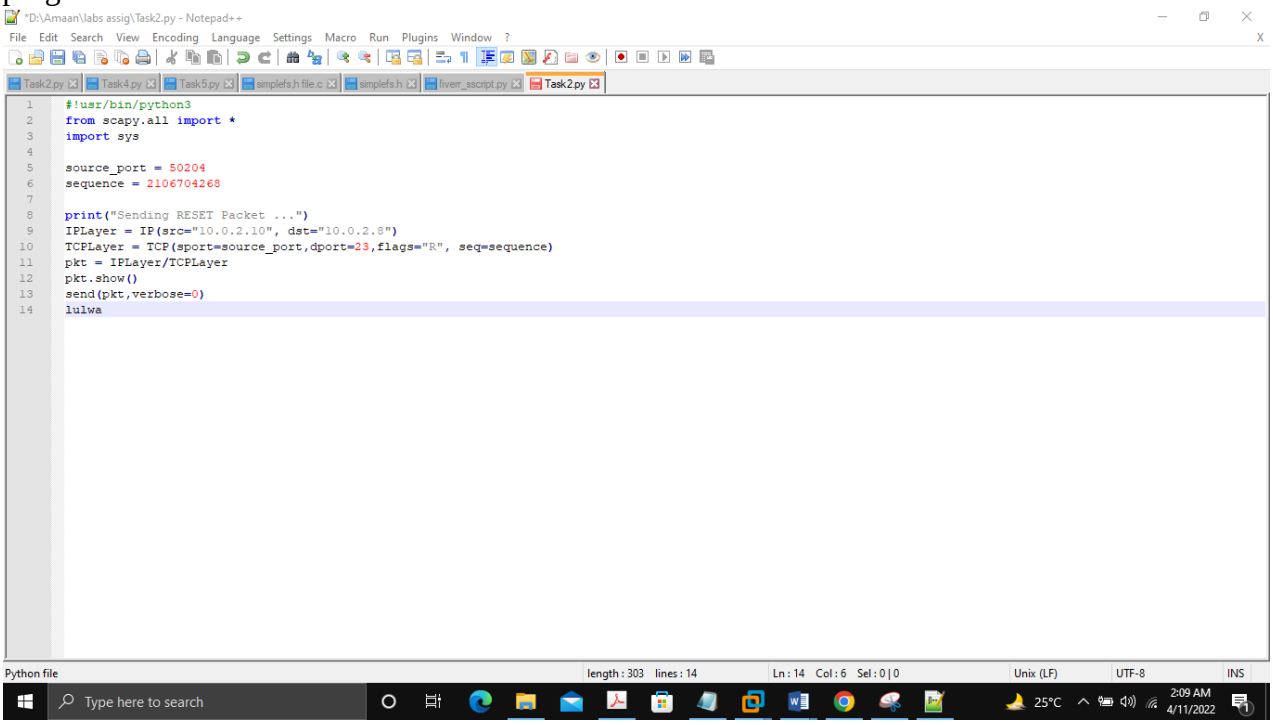
The following Wireshark trace shows the spoofed RST packet from B to A:



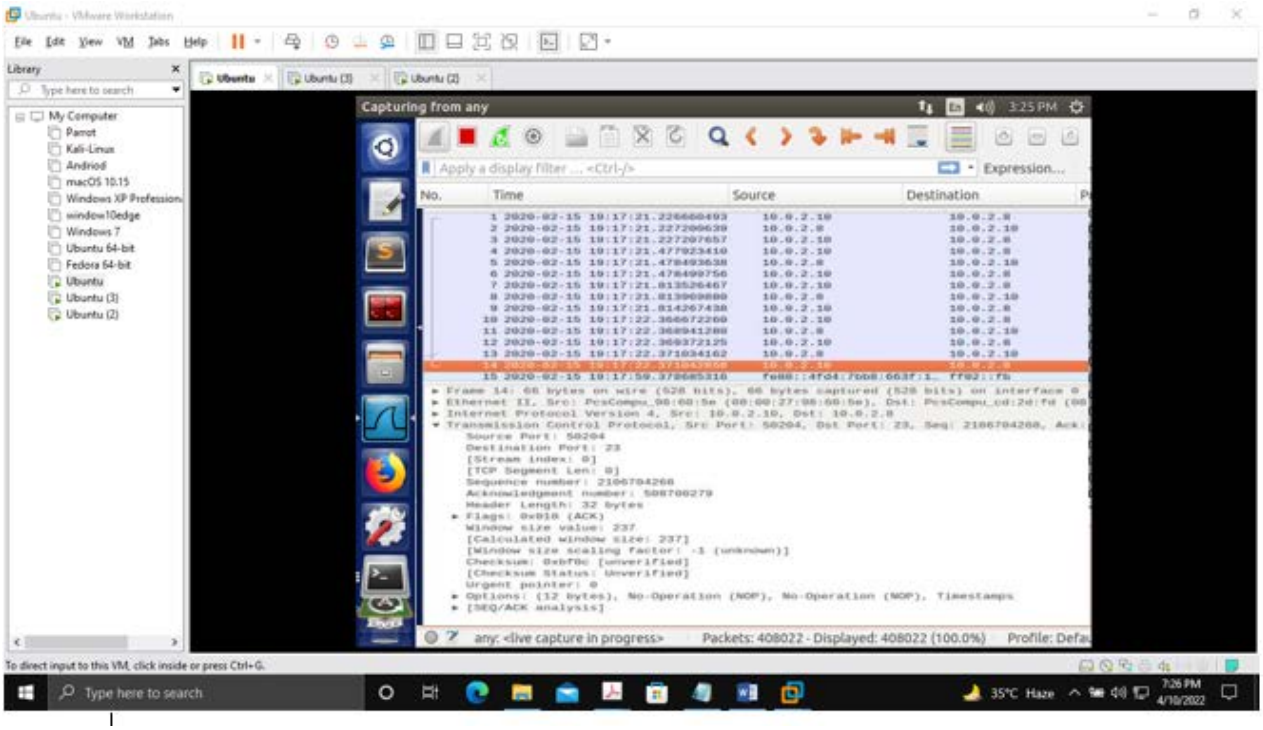
This indicates that we were able to close an established connection between A and B by spoofing an RST packet from B to A.

Using Scapy:

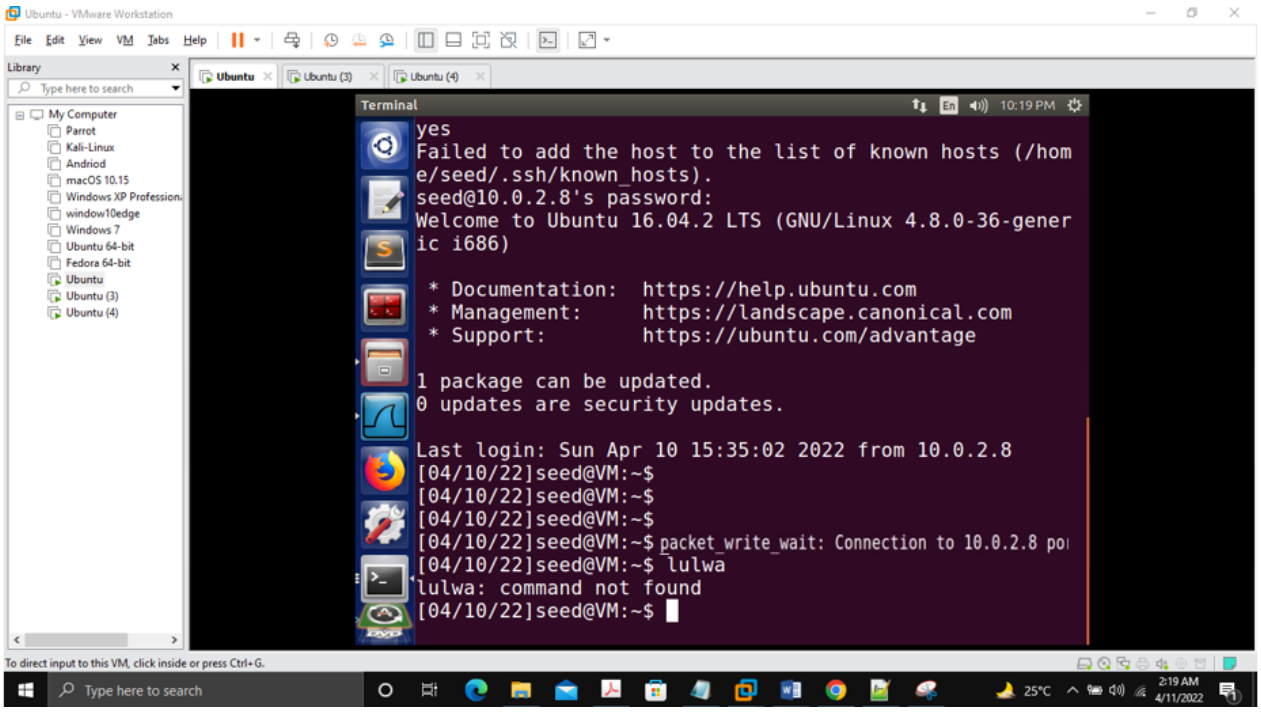
Now we perform the same RST Attack on an SSH connection using the following scapy program:



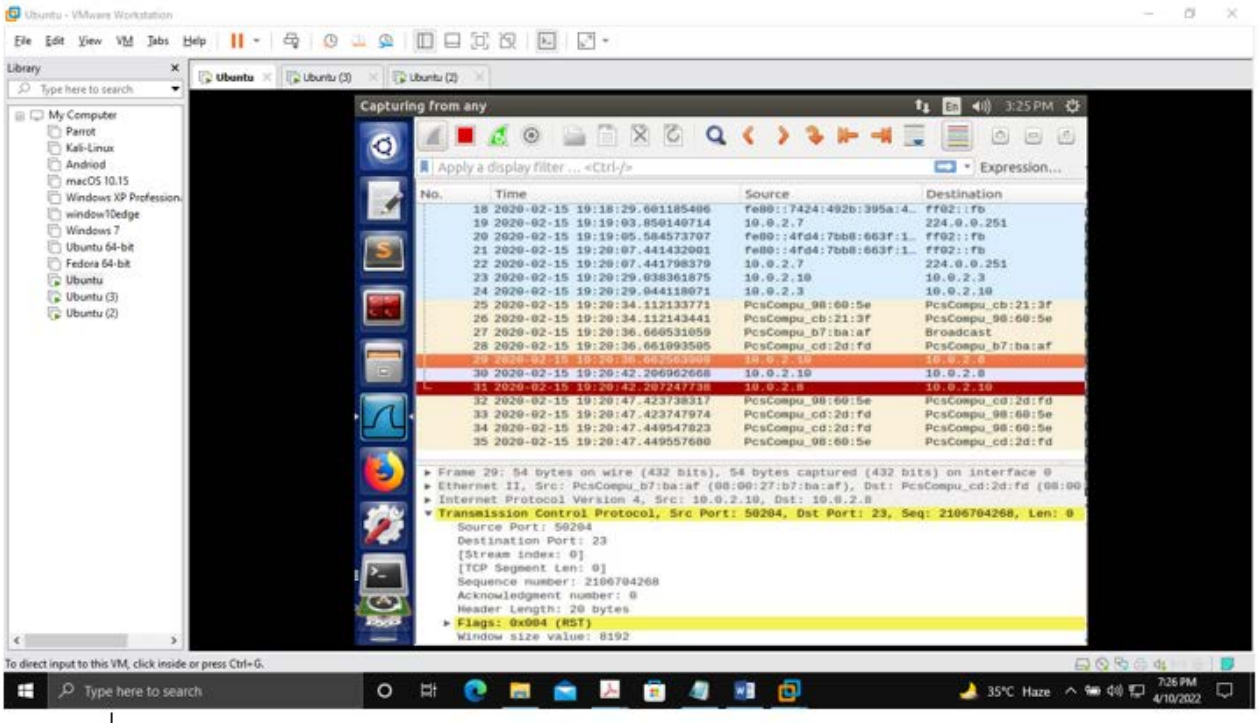
After establishing the connection and verifying the established connection by sending a pwd command, we sniff the network to find the sequence number and source port of the last sent packet from 10.0.2.10 (A) to 10.0.2.8 (B):



In order for our attack to be successful, we need to make sure that the sequence number is exactly what is next expected by the server or else our attack will fail. Then we run the program on the attacker machine and see that the connection closes on the client machine:



The following shows that an RST packet is sent from A to B and the source MAC address is of the Attacker. This proves that we were able to successfully perform an RST attack:

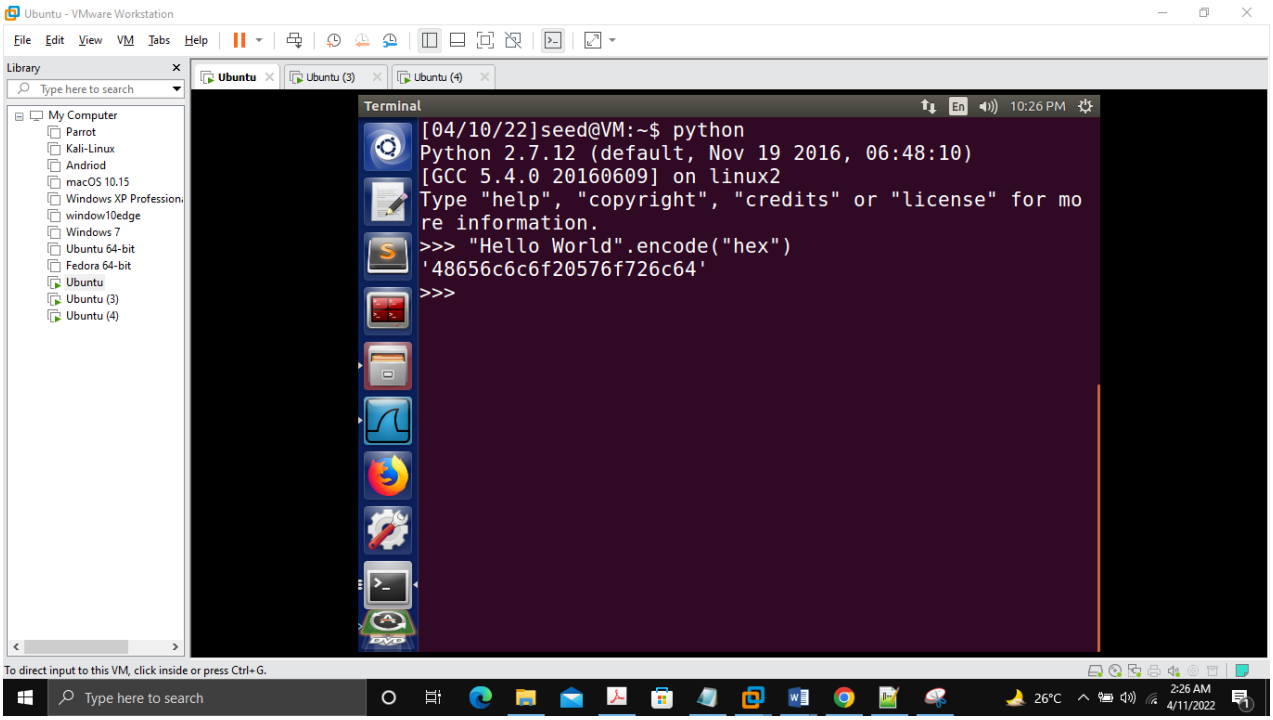


Hence, we were able to successfully launch a TCP RST attack on an SSH connection using netwox tool and scapy.

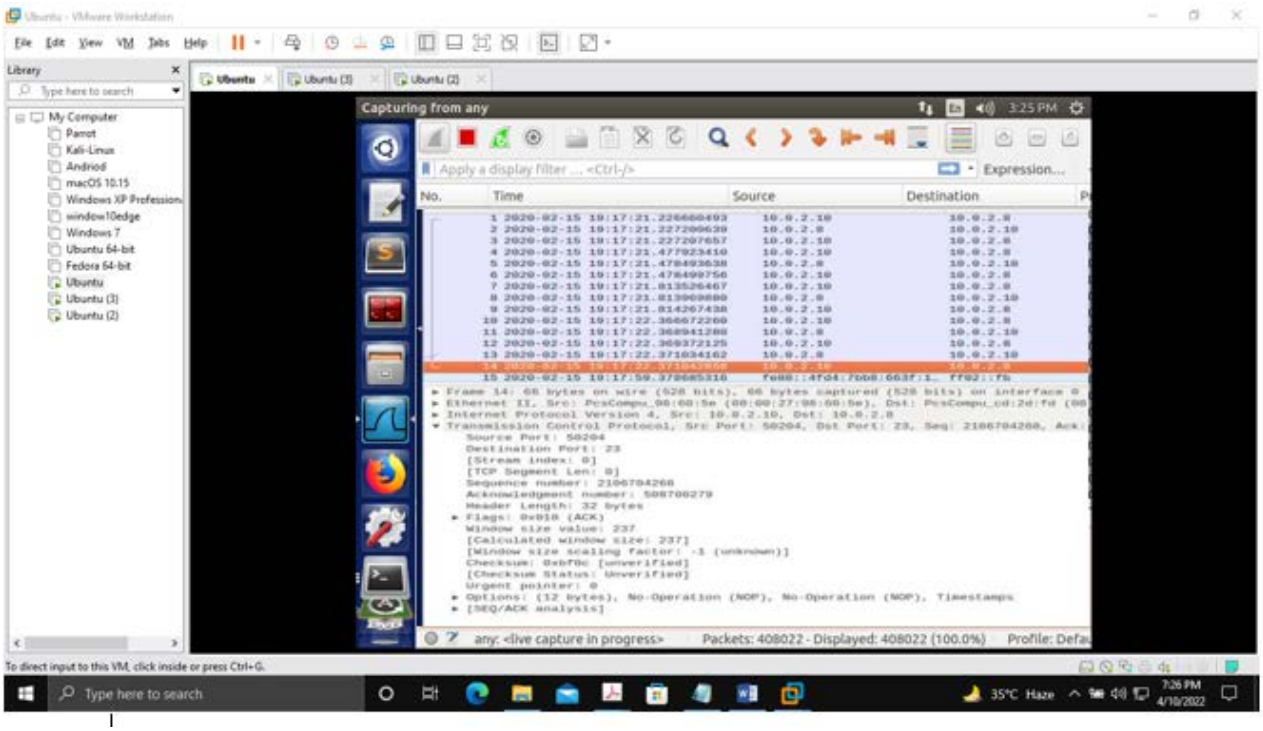
Task 4: TCP Session Hijacking

Using Netwox:

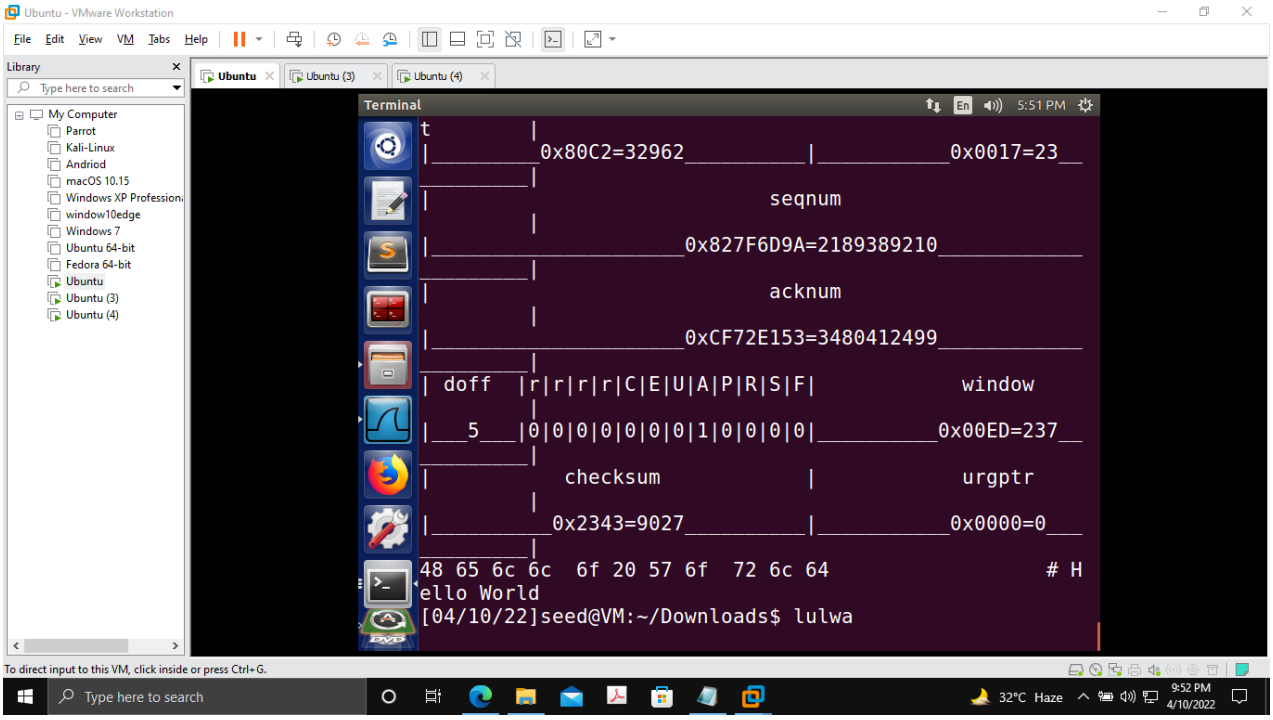
We first convert the data to be put in the packet to Hex string from an ASCII string as follows:



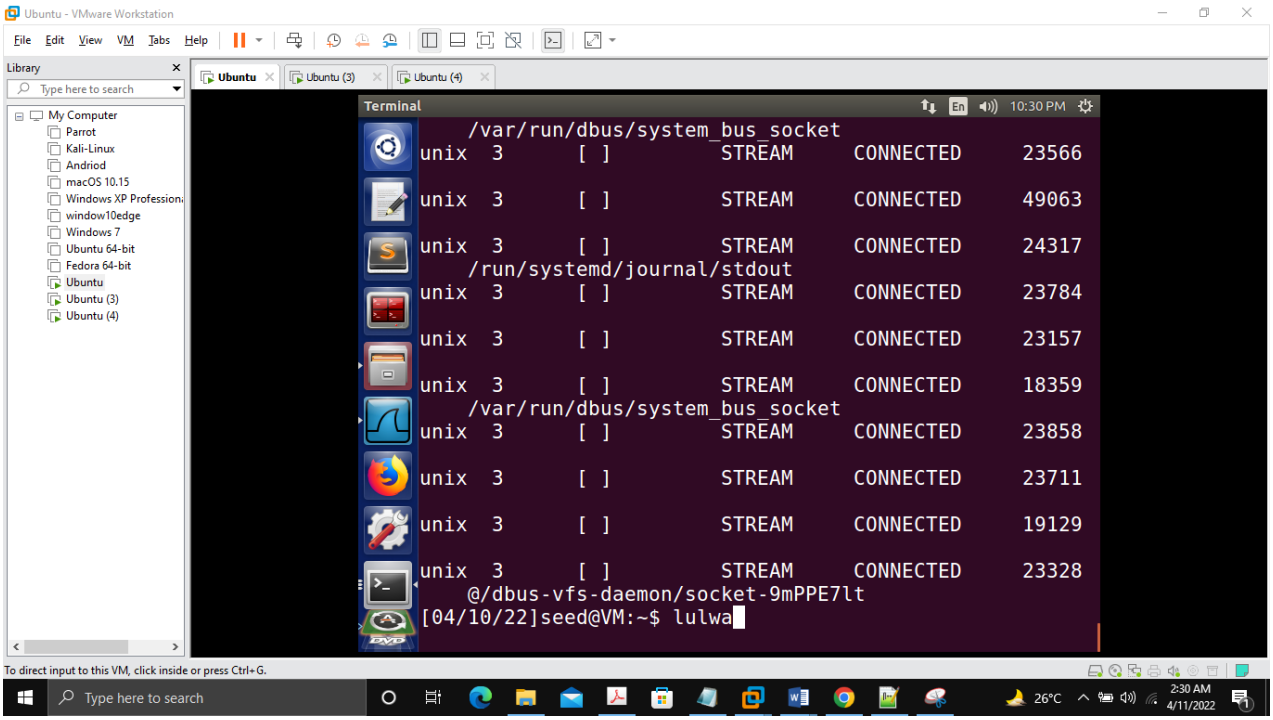
We then establish a connection between the client and server and sniff the packets in order to find the latest sent packet. The details of this packet will be used to construct the spoofed packet:



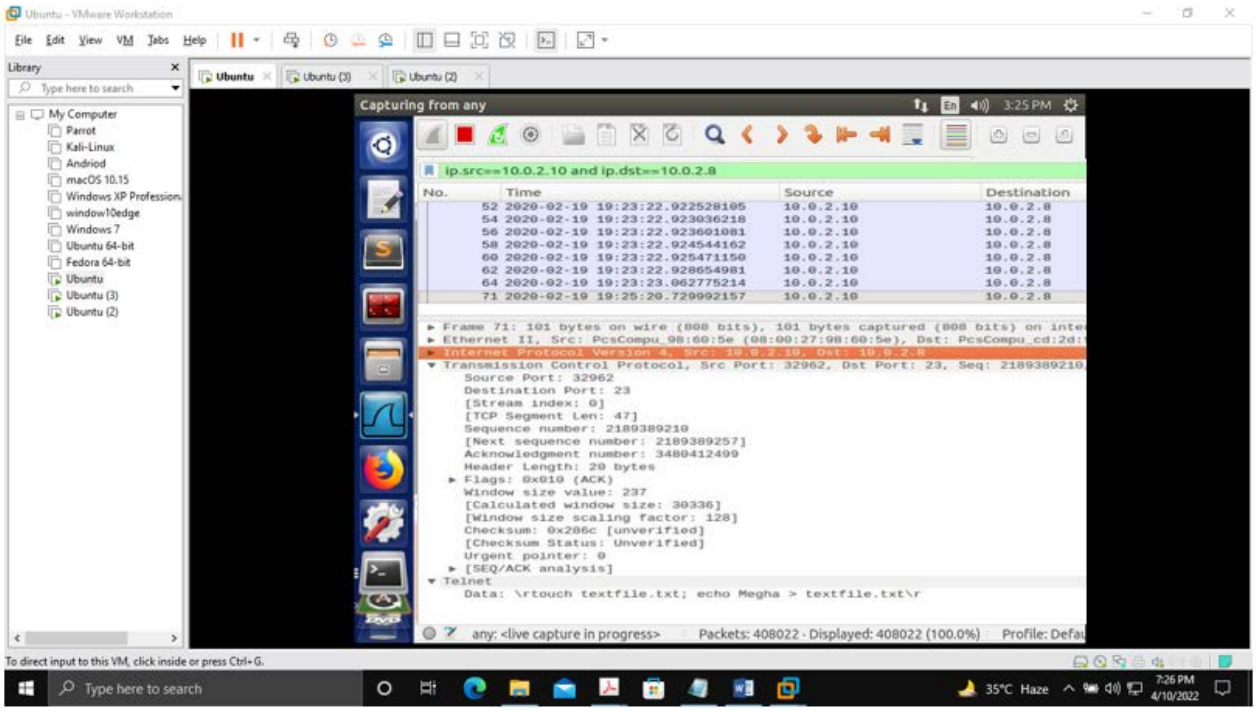
By running the netwox tool 40, we then spoof a packet from 10.0.2.10 to 10.0.2.8 such that it contains a command to create a file and write to it. This command could be more harmful such as deleting all the files in the current directory. However, for demonstration purposes we just create a file and write to it. The sequence number, acknowledgement number and the source port are obtained from the last packet. We set all the required fields in order to send the packet without it being dropped or flagged due to missing field. The following show the command and the output of the command:



The following shows the output on the server. We see that initially there was no file containing text in their name and then a telnet connection is established, and the attack program is run. On checking for the file again, we see that the file is created, and the content is also as expected.



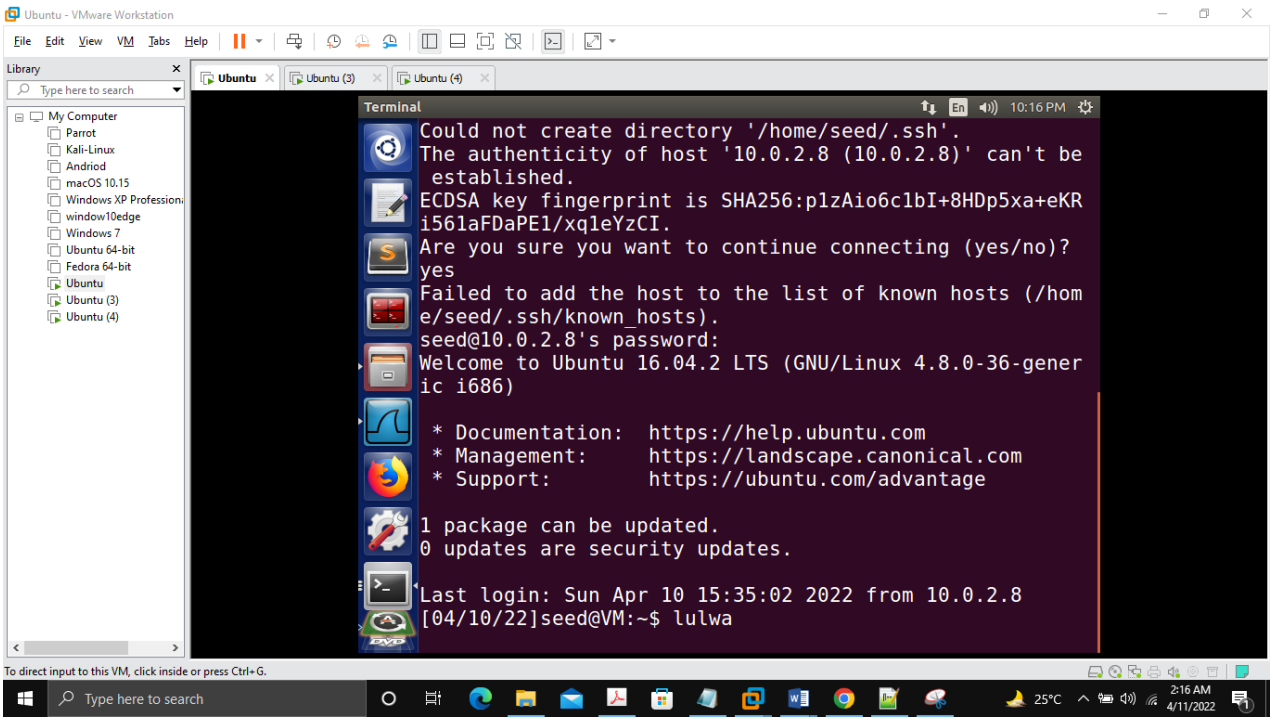
This indicates that we were able to hijack the session between the client and server and sent a command from the attacker's machine in a way that it seemed to be coming from the client. The following shows the sent packet in the Wireshark trace:



We see that the connection freezes. This is because after the spoofed packet is sent, if the actual client sends something, it is sent with the same sequence number as that of the spoofed packet. Now since the server has already received a packet with that sequence number, it just drops it. Telnet being a TCP connection, the client keeps sending the packet until it receives an acknowledgement.

Also, the server sends an ACK to the actual client for the spoofed packet and since the client did not send anything, it just discards the received ACK. The server is expecting an ACK in return and until it receives one, it keeps sending more and more ACK packets.

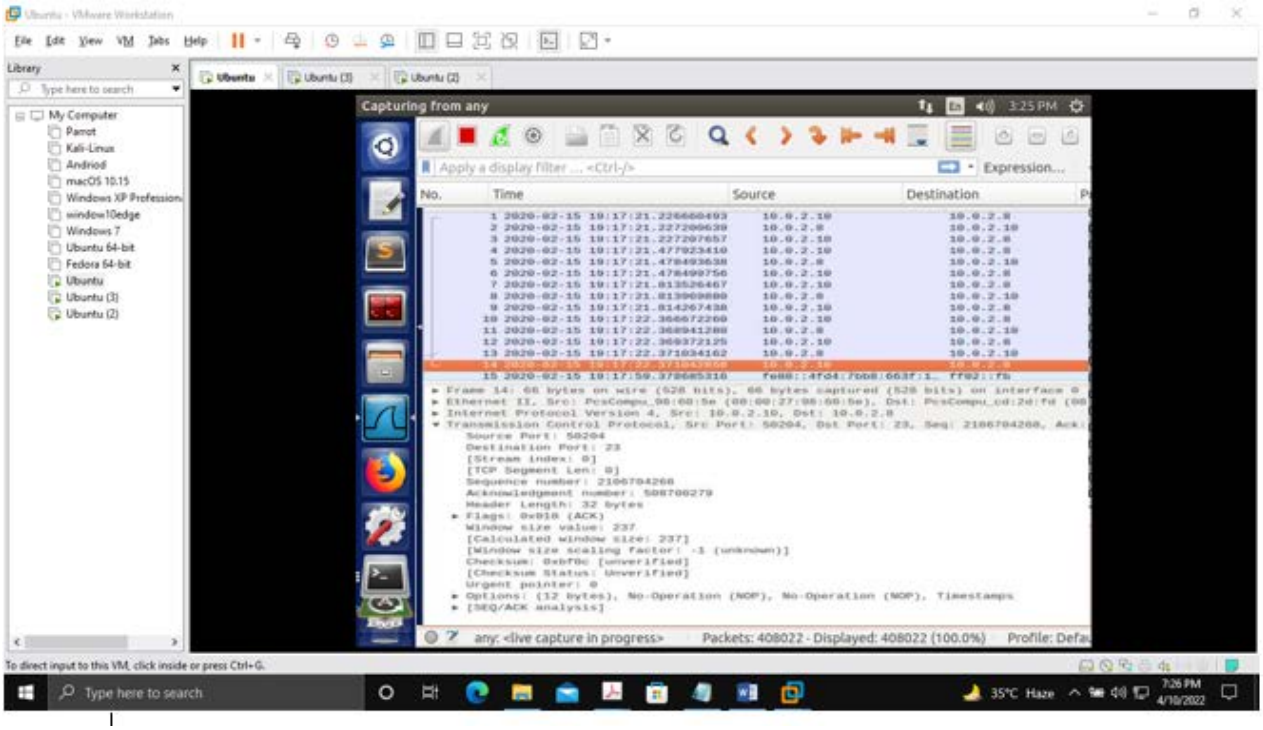
This leads to a deadlock and eventually freezes this connection as seen:



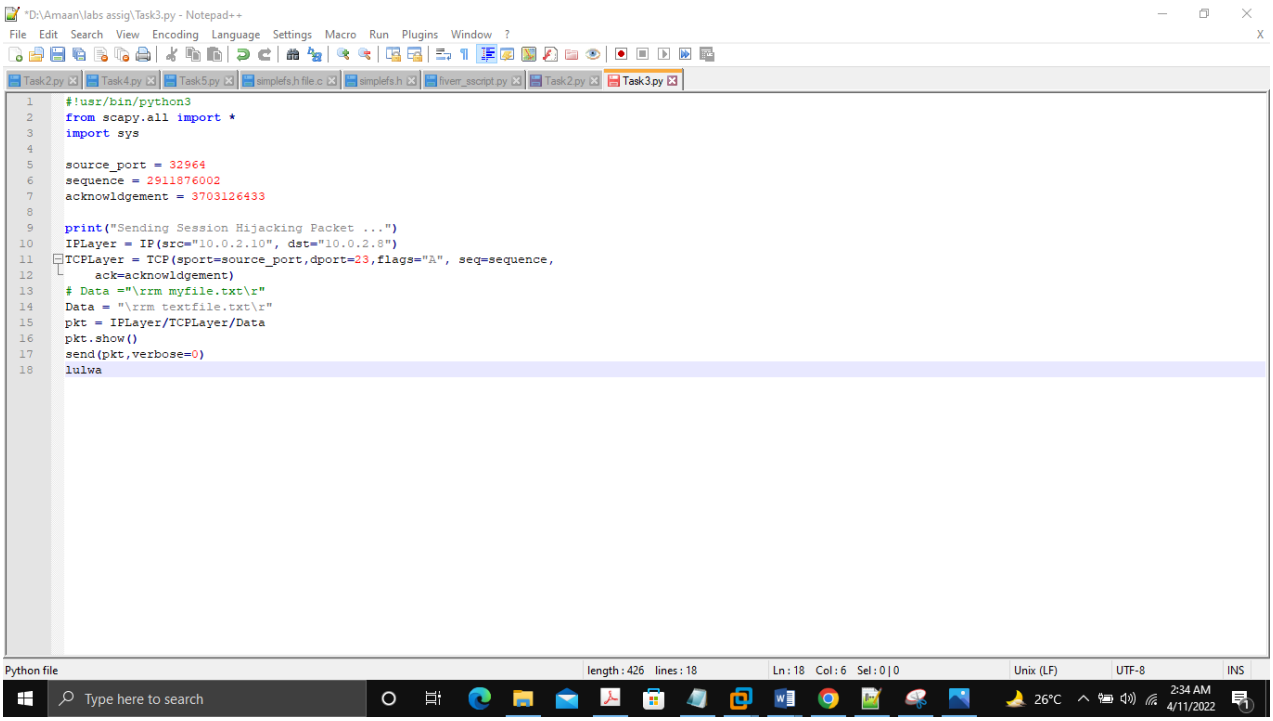
Instead of just creating a file, we could edit files such as /etc/passwd and others using session hijacking.

Using Scapy:

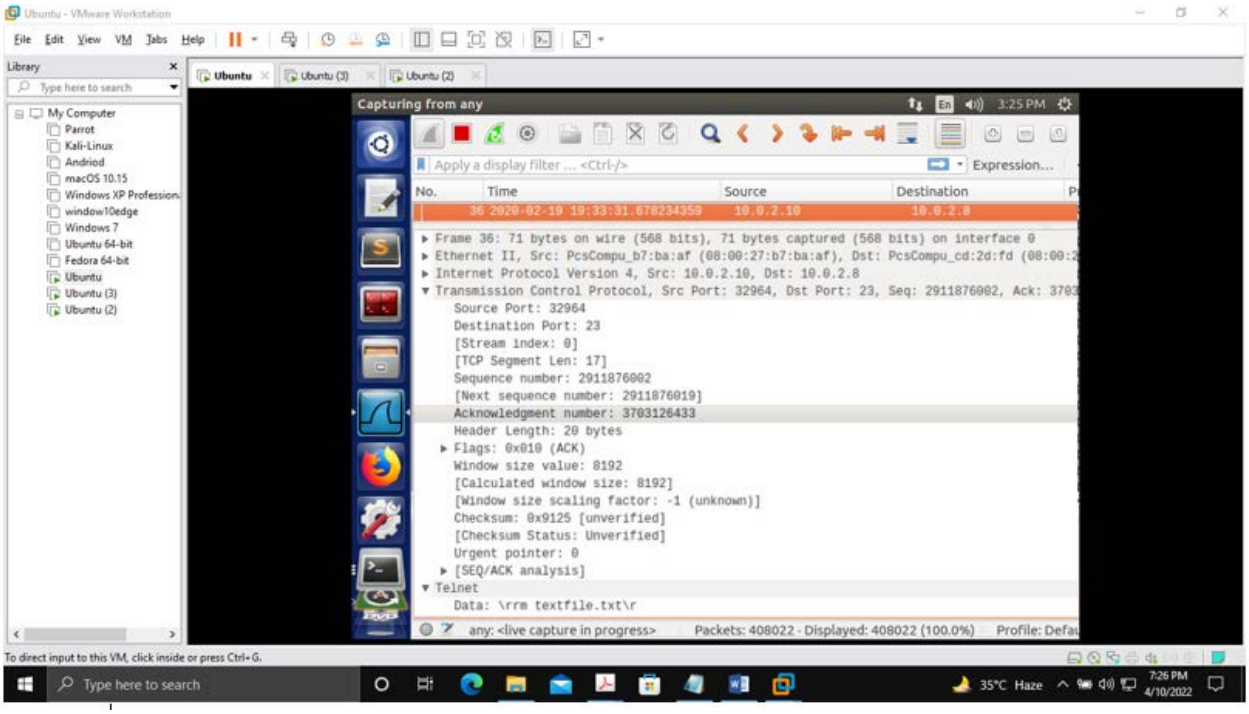
A Telnet connection is first established between the client and the server and we sniff this traffic. The following shows the Wireshark trace:



The details of the last sent packet is used to construct the spoofed packet. We perform session hijacking using the following program that sends a packet from the client to the server and deletes a file named textfile.txt in the current directory. This file is the one created in session hijacking attack using netwox:


*D:\Amaan\labs assig\Task3.py - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Task2.py Task4.py Task5.py simplefs.h file.c simplefs.h fiverr_sscript.py Task2.py Task3.py
#!usr/bin/python3
from scapy.all import *
import sys
source_port = 32964
sequence = 2911876002
acknowldgement = 3703126433
print("Sending Session Hijacking Packet ...")
IPLayer = IP(src="10.0.2.10", dst="10.0.2.8")
TCPLayer = TCP(sport=source_port,dport=23,flags="A", seq=sequence,
ack=acknowldgement)
# Data ="\rrm myfile.txt\r"
Data = "\rrm textfile.txt\r"
pkt = IPLayer/TCPLayer/Data
pkt.show()
send(pkt,verbose=0)
lulwa
Python file
length : 426 lines : 18
Ln : 18 Col : 6 Sel : 0 | 0
Unix (LF)
UTF-8
INS
Type here to search
26°C
2:34 AM
4/11/2022

The following are the packet details of the spoofed packet:



The following shows the output at the Server. We see that after the connection is established and the program is run, the file is deleted on the server.

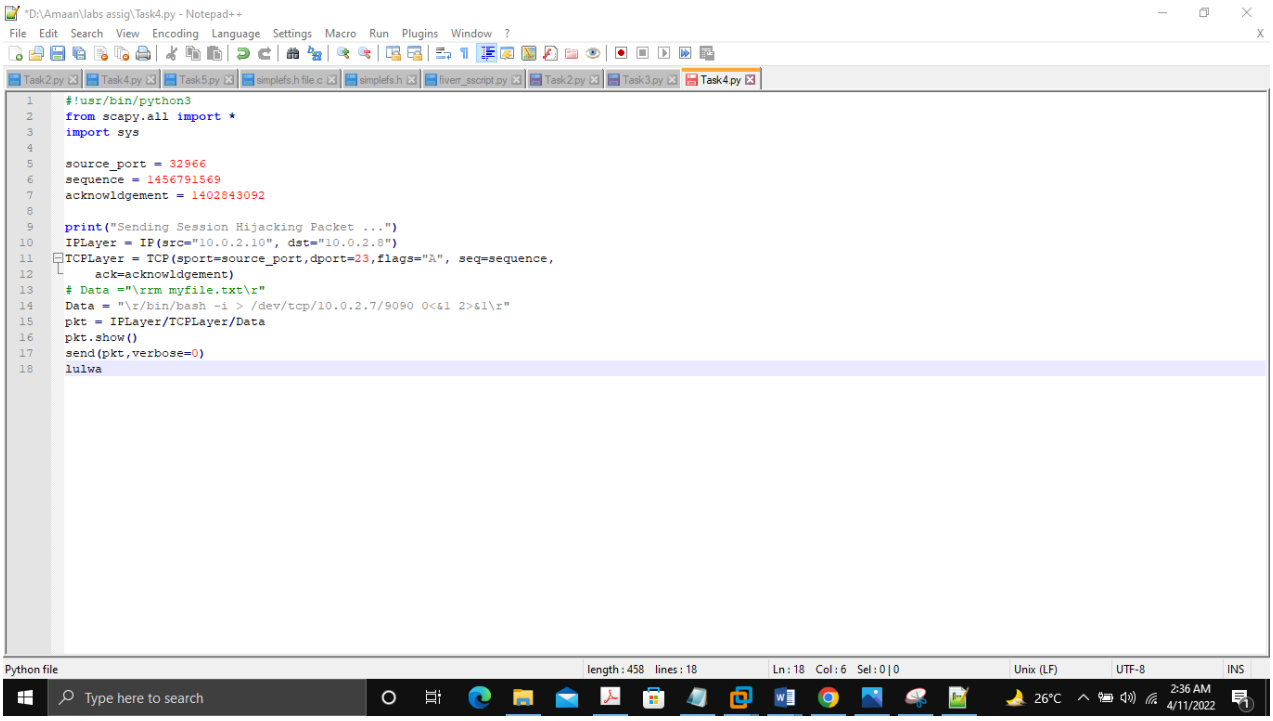
This completes Session Hijacking attack using netwox and scapy.

Task 5: Creating Reverse Shell using TCP Session Hijacking

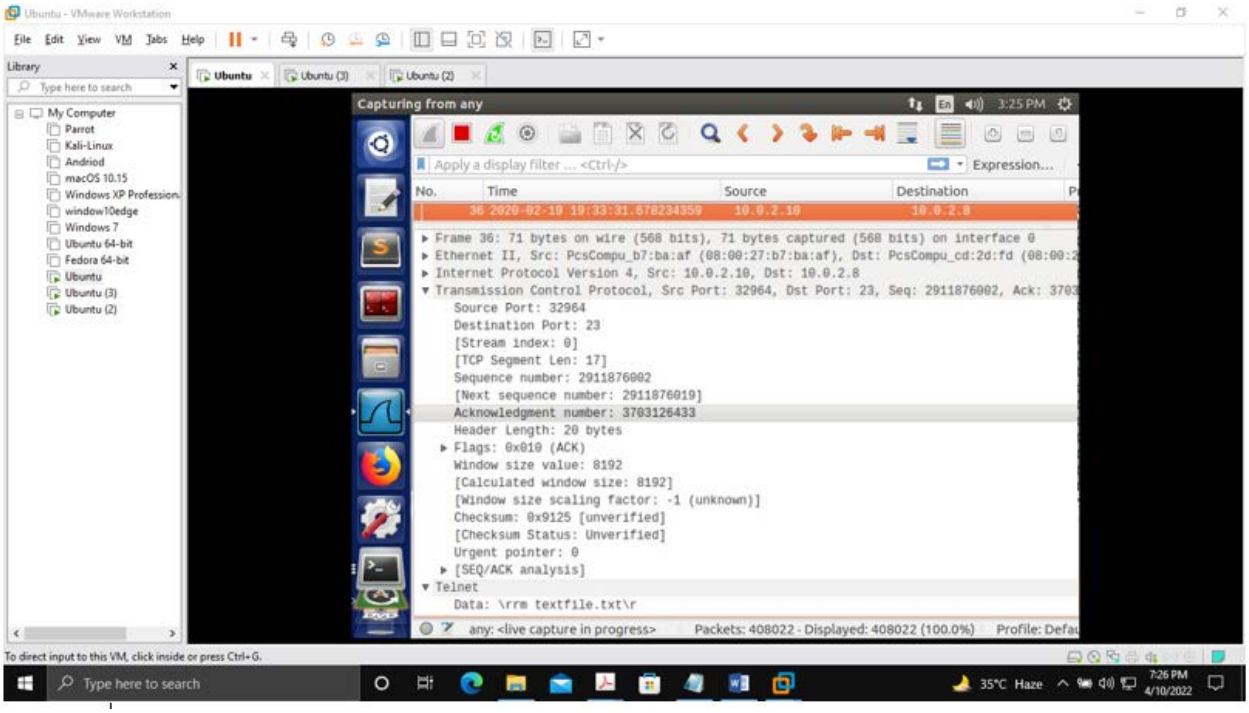
Using the Session Hijacking attack, we create a reverse shell from the server to the attacker's machine, giving attacker the access to the entire server machine to run commands. In this attack, we send a command in the packet's data to run the bash program and redirect its input, output and error devices to the remote TCP connection.

The following is the program to perform the session hijacking attack. The flow of the task is as follows:

1. Establish a telnet connection between the client 10.0.2.10 and server 10.0.2.8.
2. Sniff the traffic and find the last packet sent from client to the server. The details of this packet are used to spoof the attack packet.
3. Start a TCP connection listening to port 9090 on the attacker's machine.
4. Run the Session Hijacking program on the attacker's machine

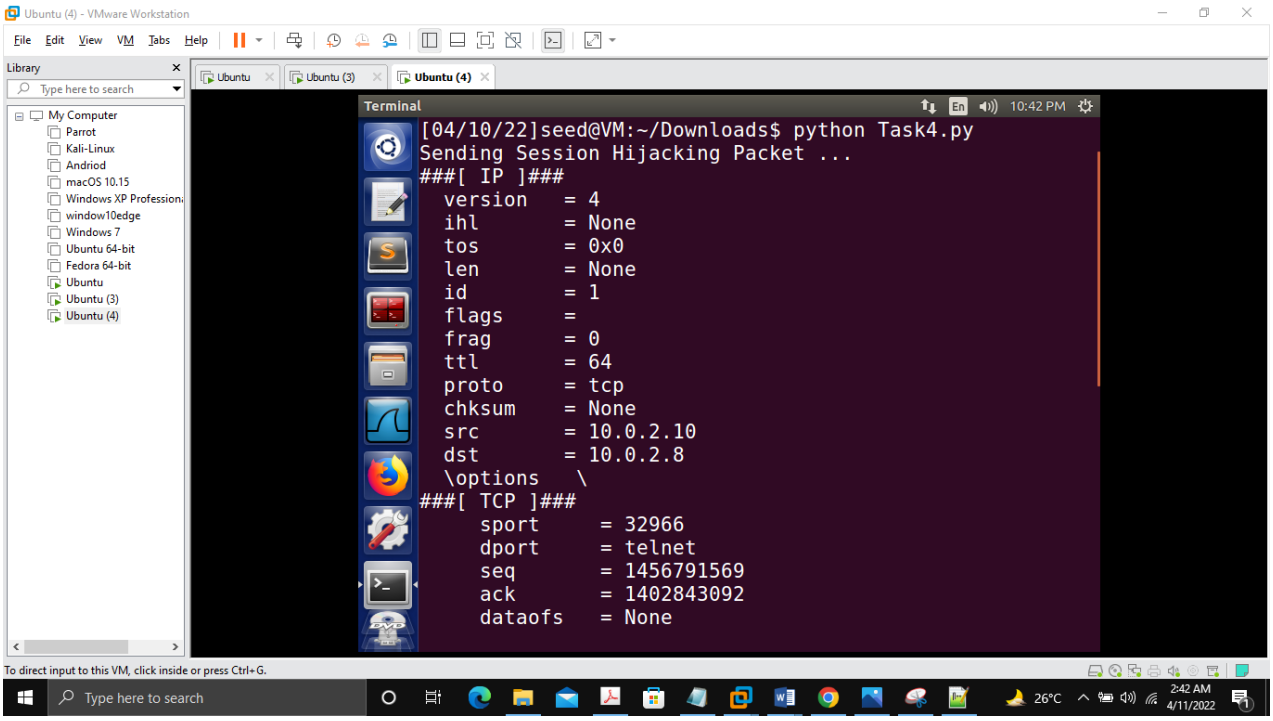


The following Wireshark trace show the spoofed packet sent. Notice that the source and destination are of
client and server and MAC source is of the attacker's machine.



The following show the output on the attacker's machine. We see that the packet sent is the same as one captured in Wireshark. Also, another terminal with a TCP connection listening to port 9090 has successfully established a reverse shell. This can be proven because before running the netcat server, we switched to the downloads folder, hence the current directory was /home/seed/Downloads. After the netcat command, on looking for the current directory, we see that it's changed to /home/seed. This is the directory of the telnet connection, as seen. Hence, we were able to create a reverse shell by performing session hijacking attacks.

Output on the Attacker's machine:



Output on the Server machine: