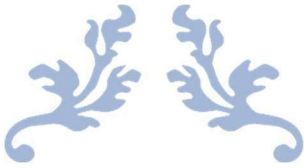


PACKET SNIFFING AND SPOOFING

JANUARY 2024

Abdulfatah Abdillahi

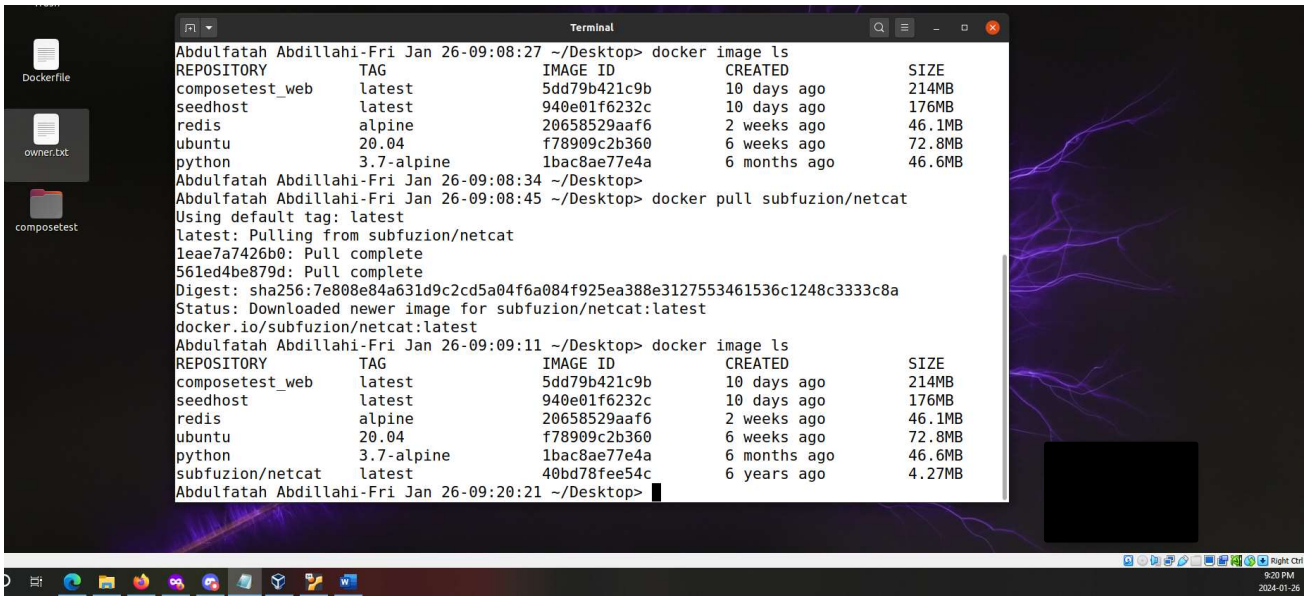
Contents

Introduction

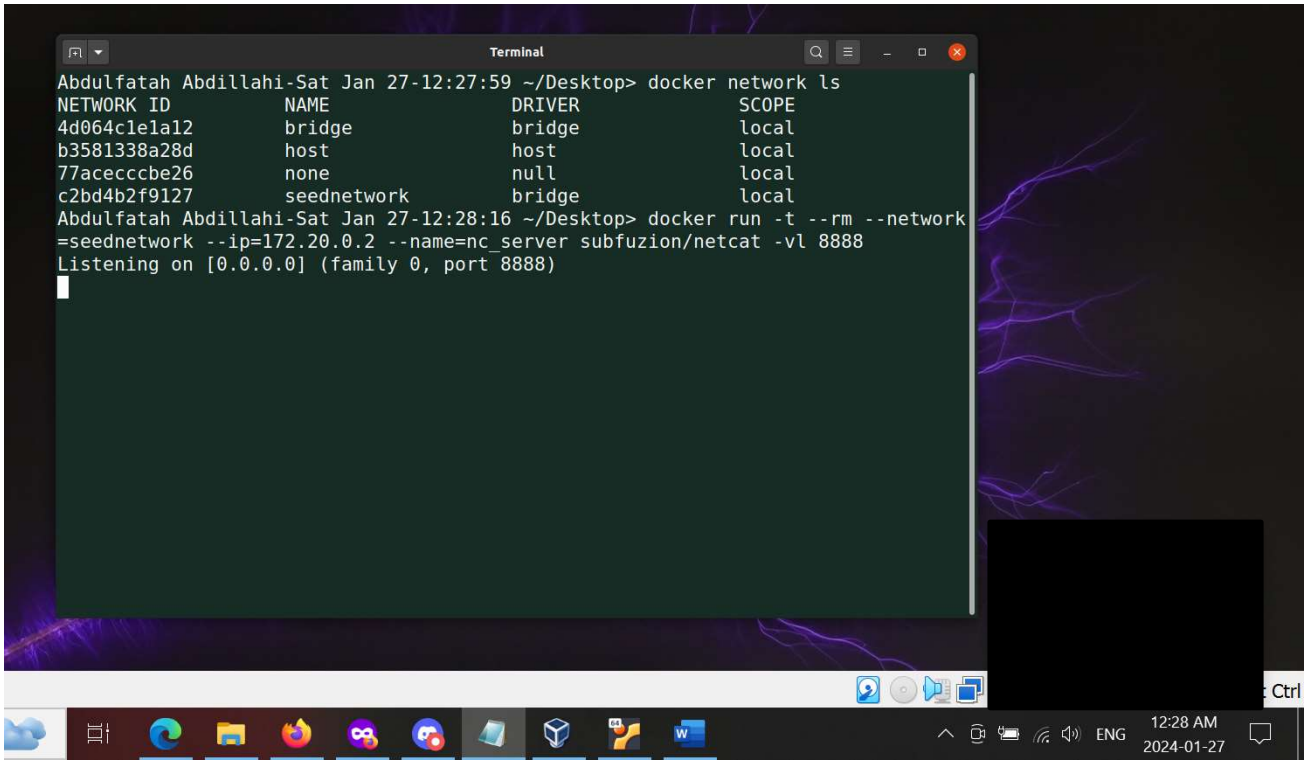
In this lab we explored some topics like Netcat and packet sniffing and spoofing using docker containers.

Part 1: Netcat

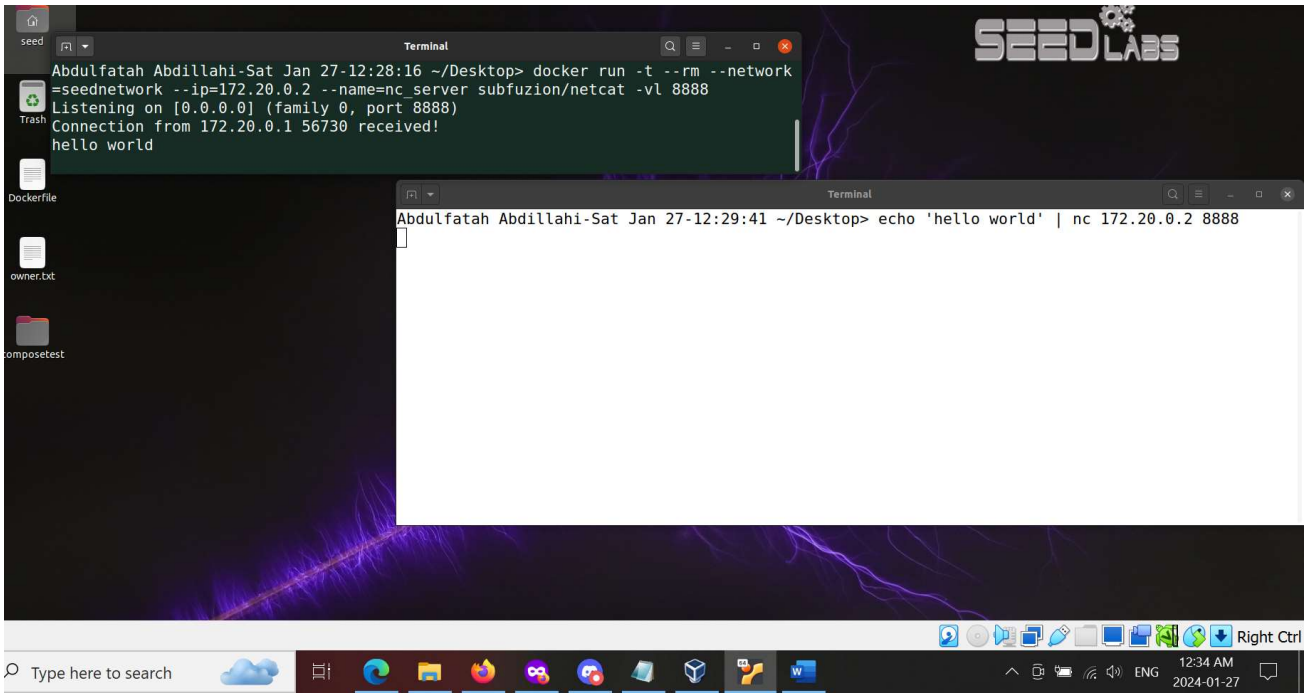
Here, I'm pulling the netcat image (as specified in the lab instruction) from Docker Hub.



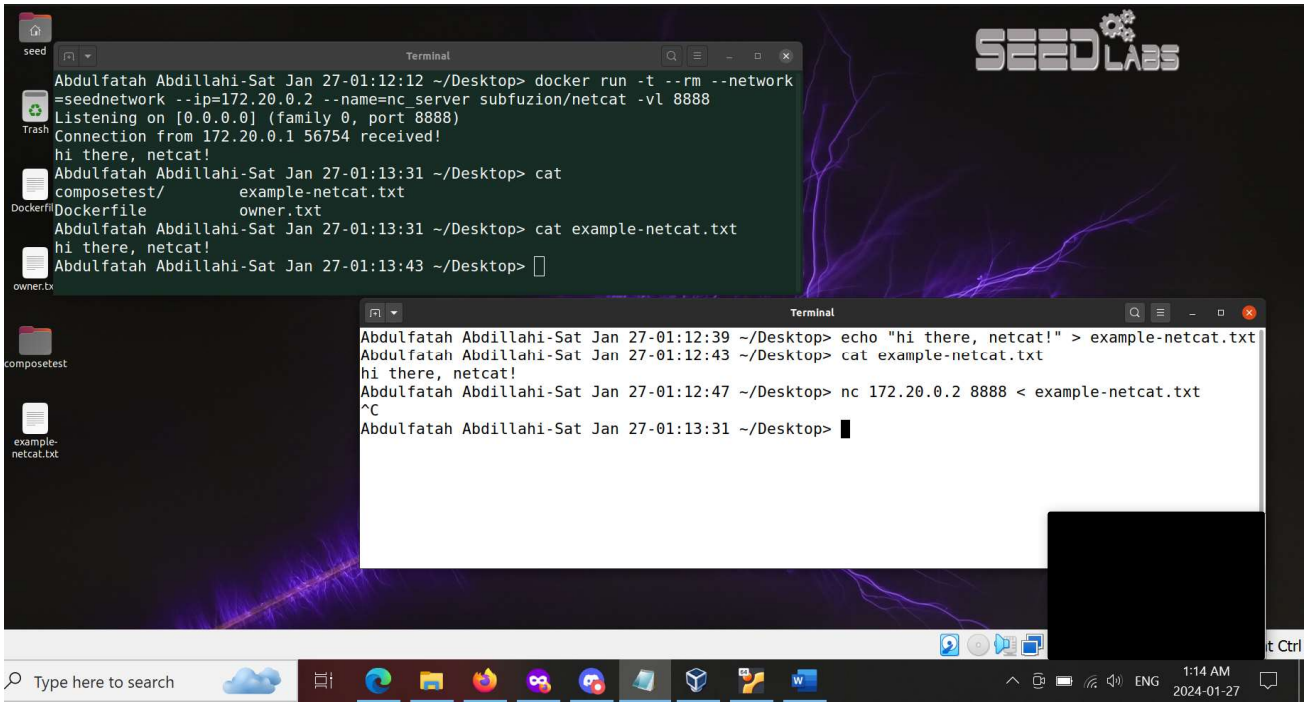
Here, I was able to run the Netcat server container using the command *docker run -t --rm --network=seednetwork --ip=172.20.0.2 --name=nc_server subfuzion/netcat -vl 8888* where I utilized the previously made network in lab 0 (seednetwork). Also, the ip assigned to this server is 172.20.0.2 with a port of 8888.



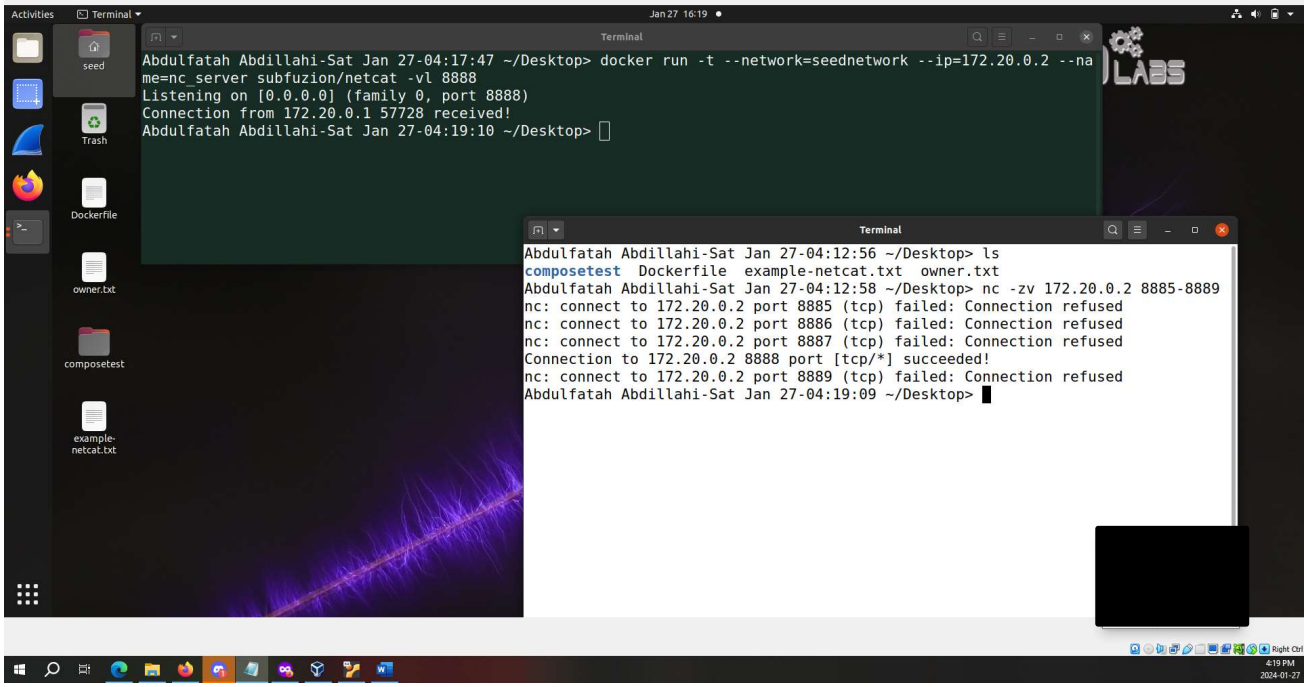
Here, I'm sending "Hello World" to the netcat server with IP 172.20.0.2 and port 8888.



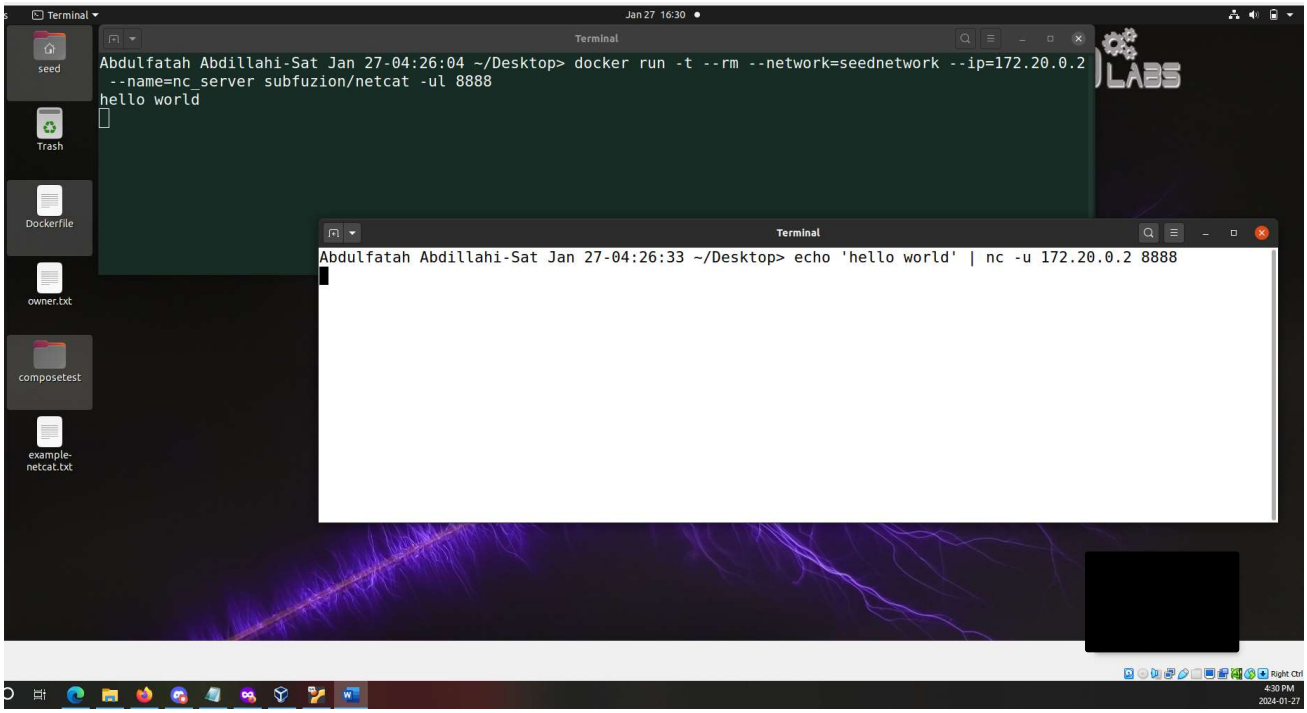
Here, I created a file called "example-netcat.txt" on the client side and sent it to the listener on the other side. On the left you can see a confirmation that the file was indeed received along with its contents.



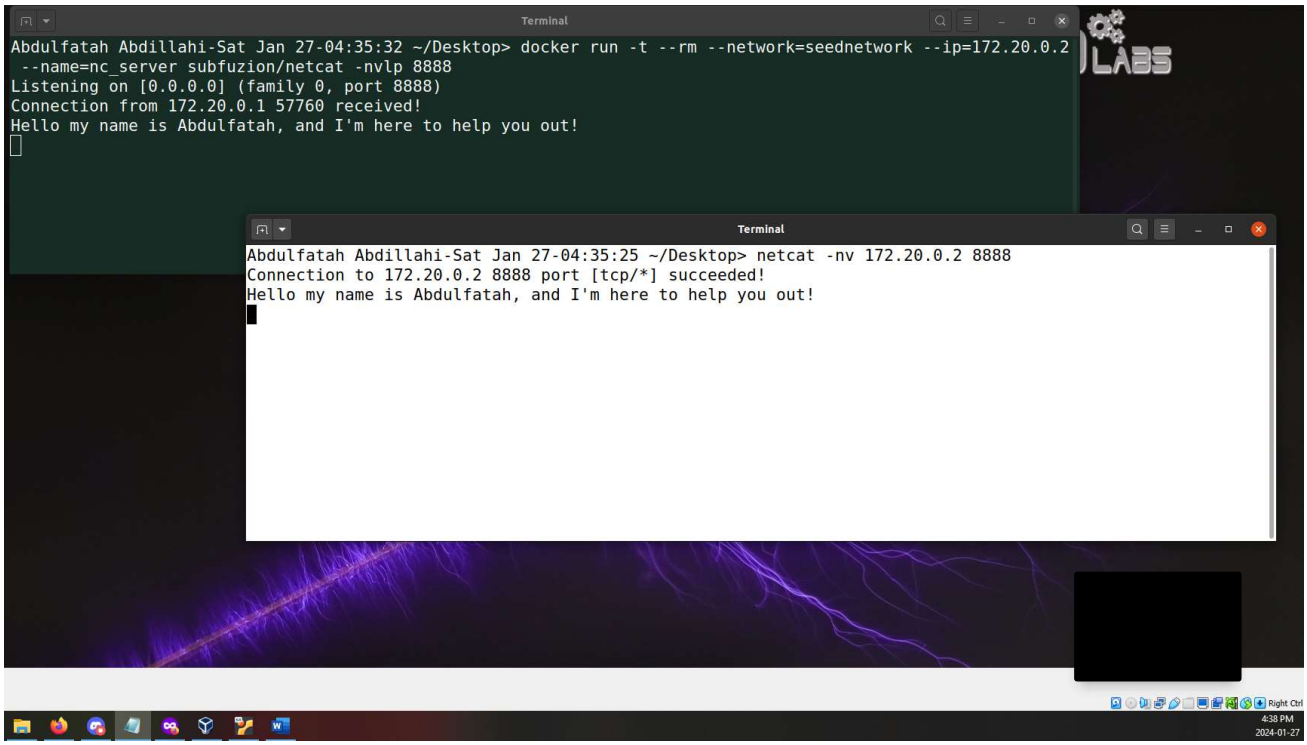
Here, we are attempting to use netcat for port scanning.



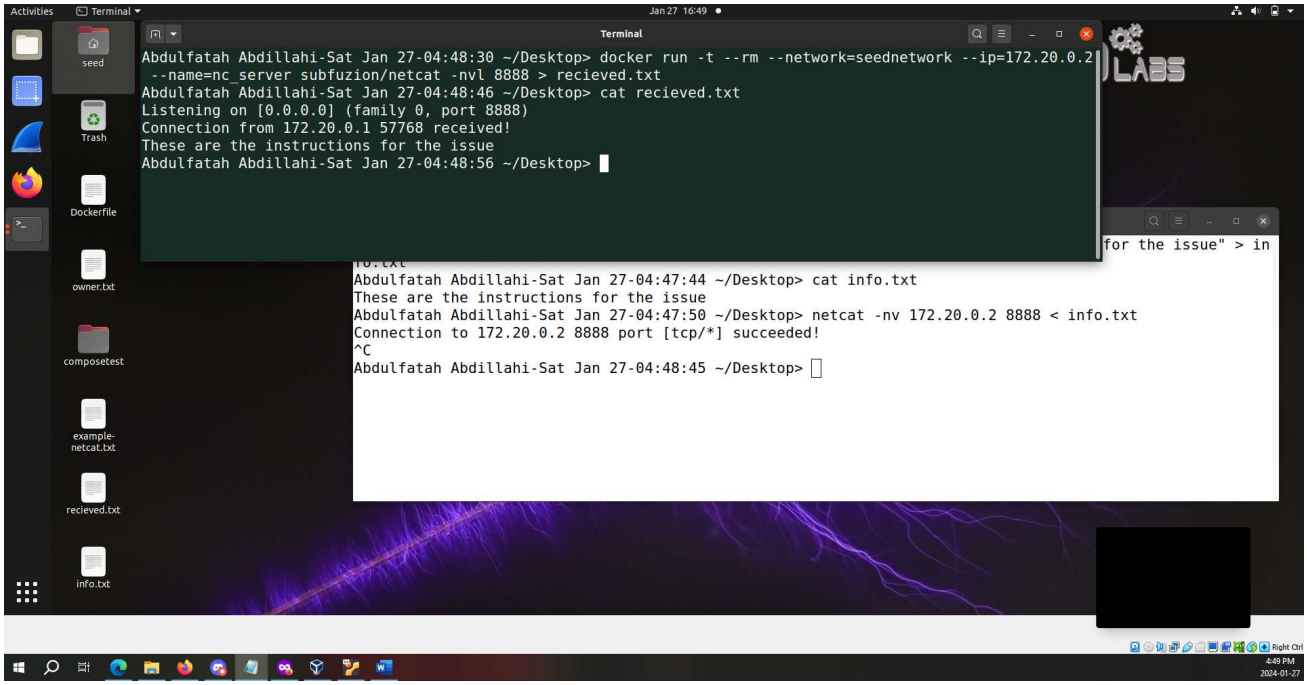
Here, we are attempting to send UDP packets witch nc (as it is commonly used with tcp)



Here, we are mimicking what connecting to a client may look like via netcat

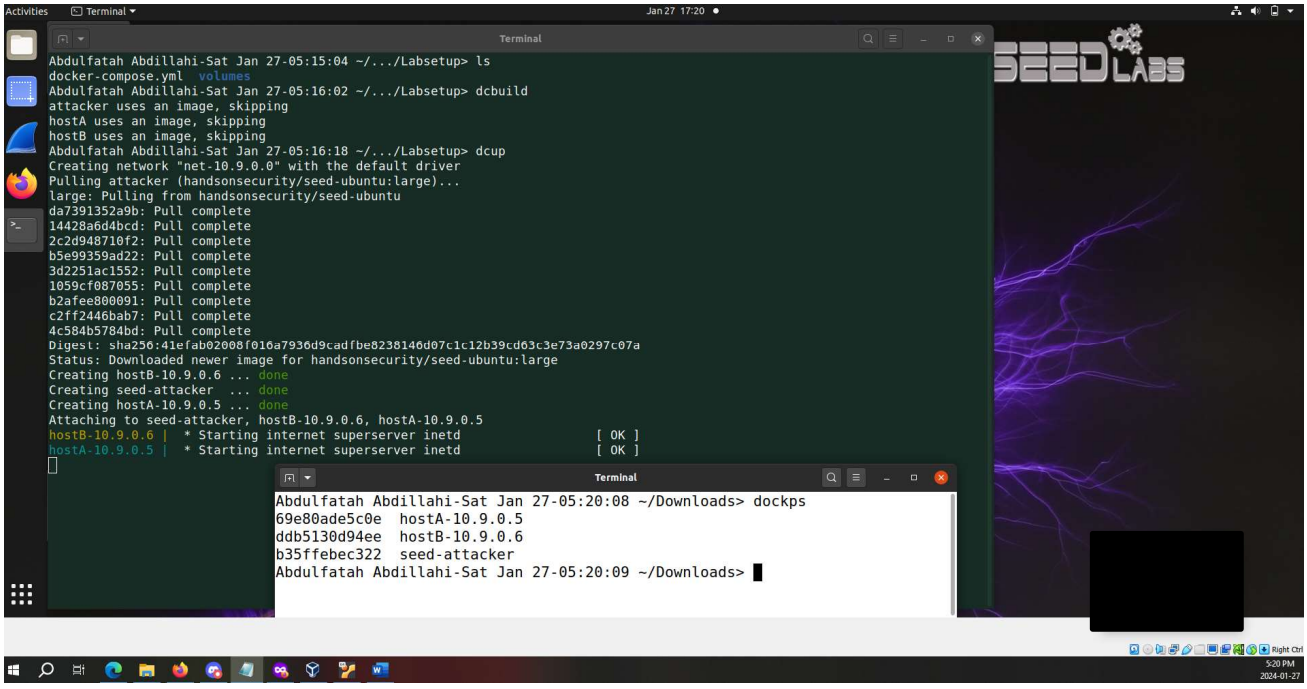


Here, in this scenario we are pretending to send and receive files to help clients with an issue for example.

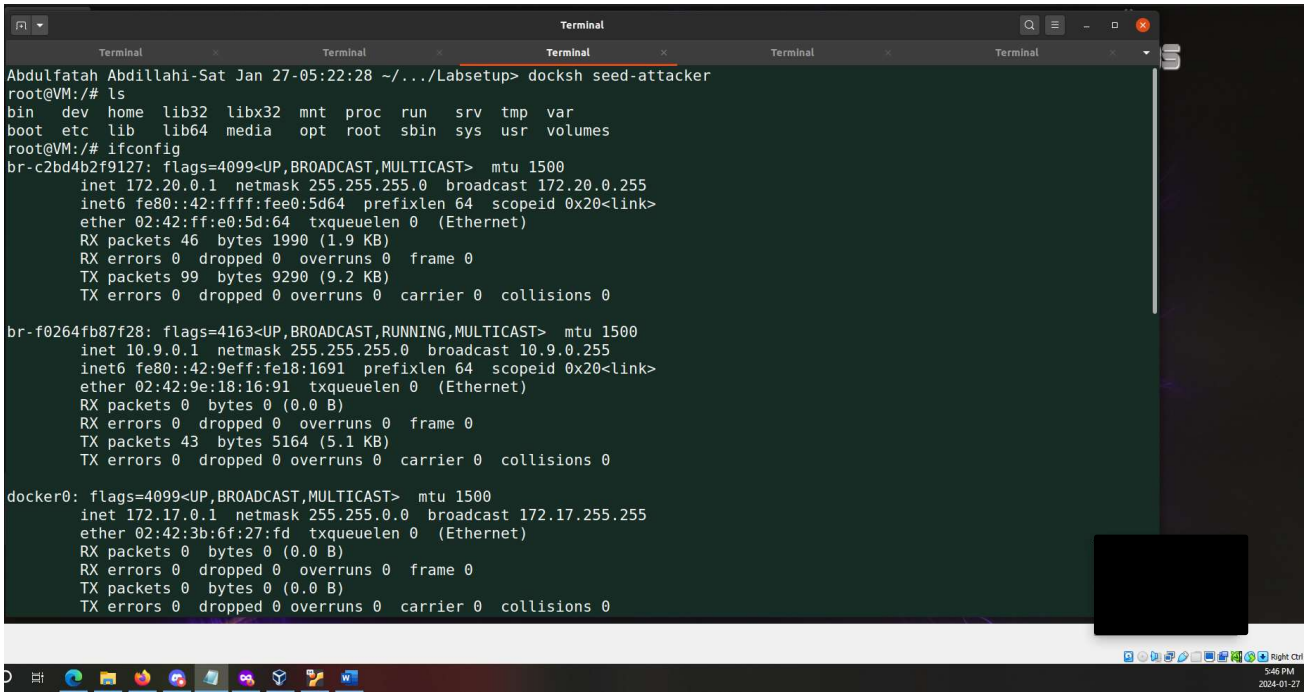


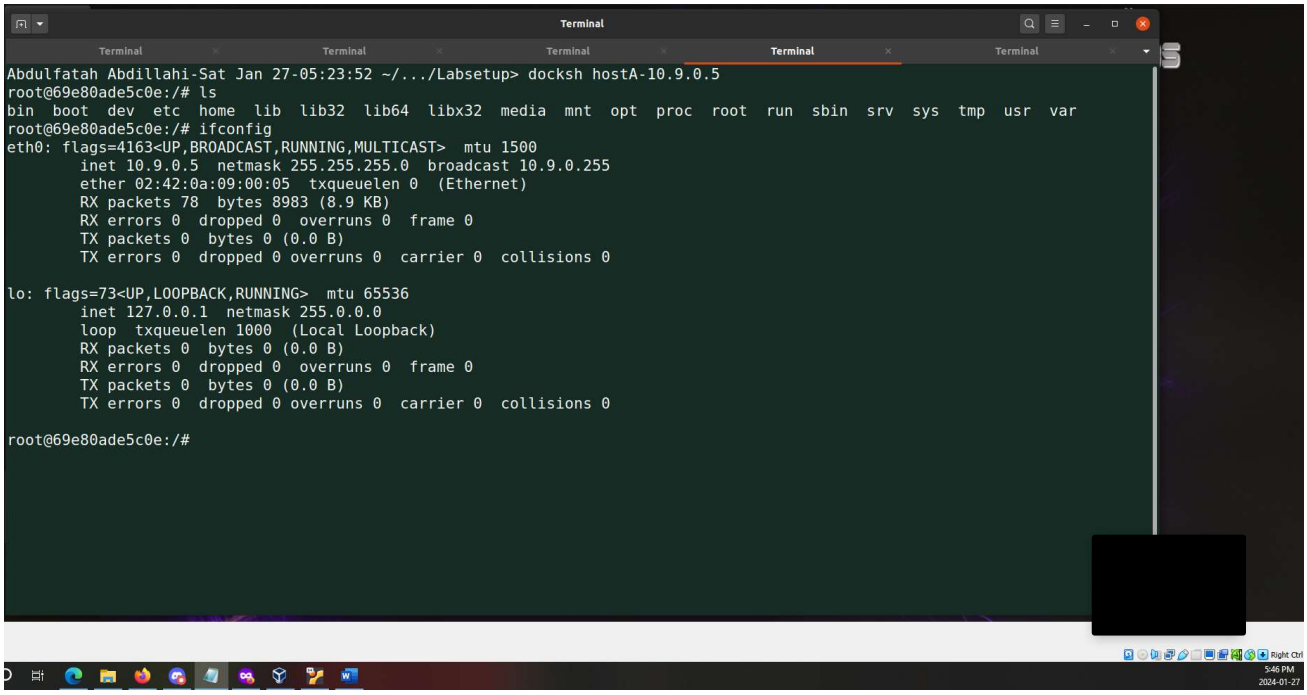
Part 2: SEED: Packet Sniffing and Spoofing Lab

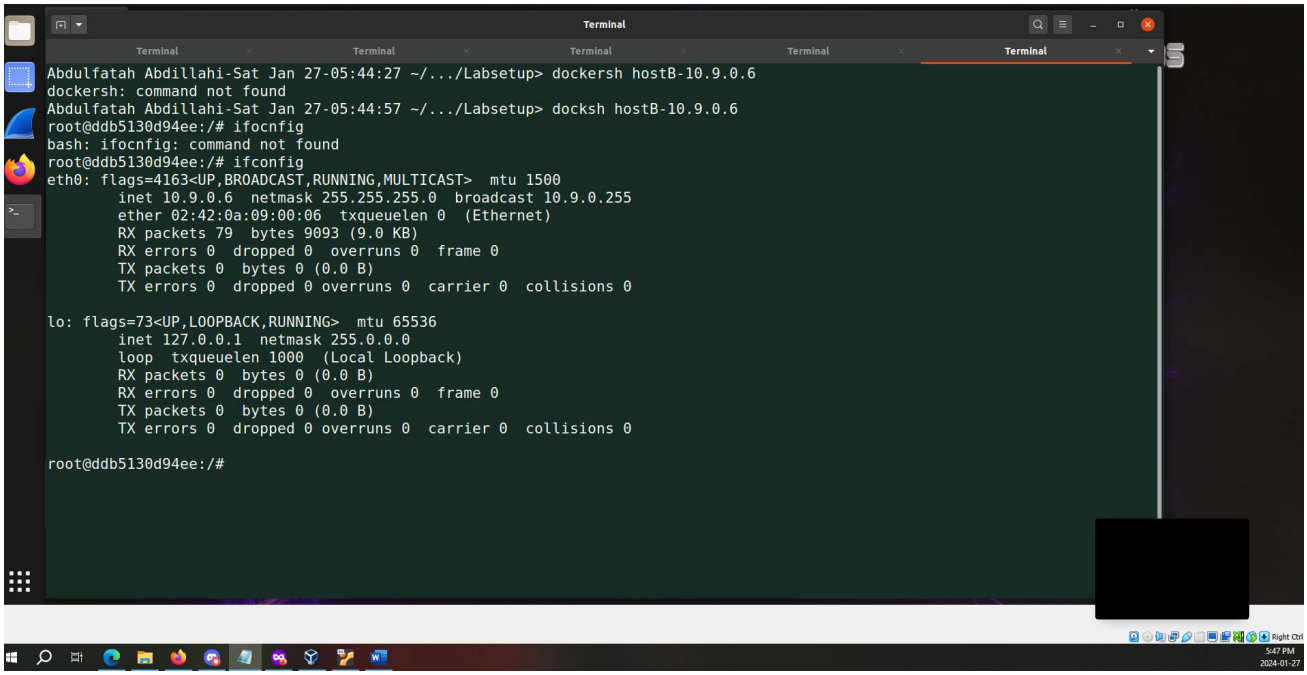
Creating three docker containers



Checking the IP addresses for each of the three containers (seed-container, hostA, and hostB).

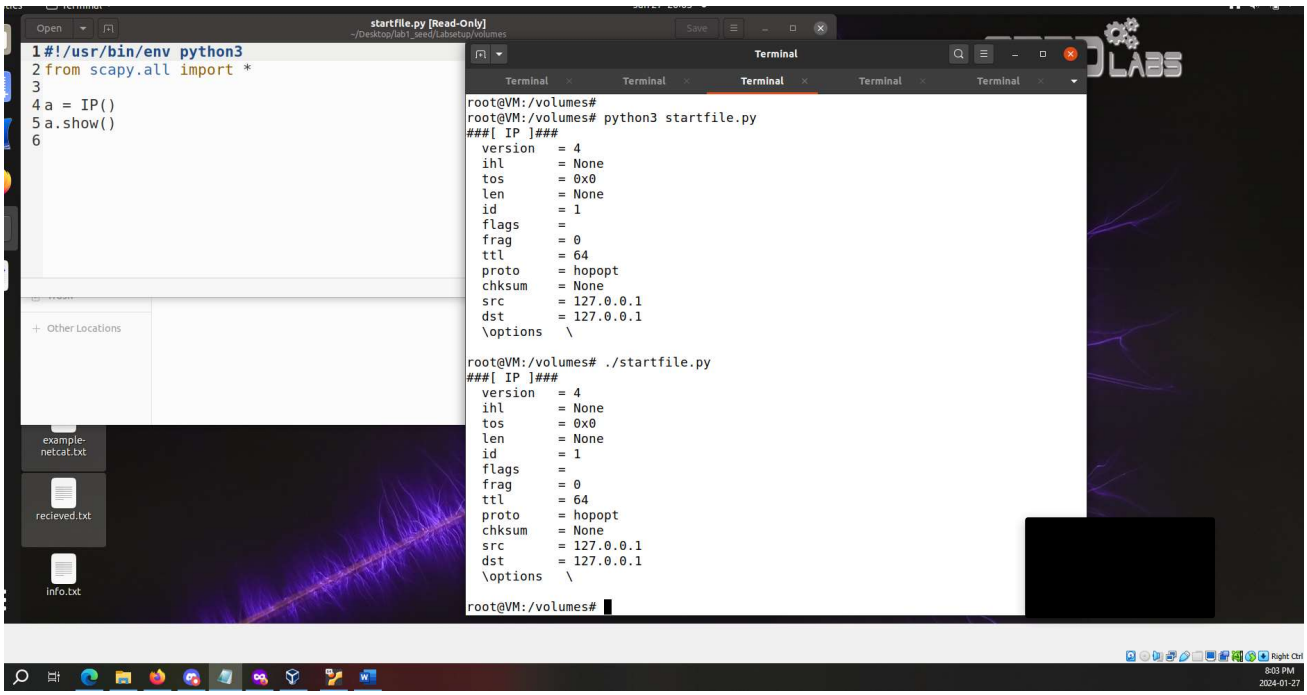

Terminal
Terminal
Terminal
Terminal
Terminal
Terminal
Abdulfatah Abdillahi-Sat Jan 27-05:22:28 ~/.../Labsetup> docksh seed-attacker
root@VM:/# ls
bin dev home lib32 libx32 mnt proc run srv tmp var
boot etc lib lib64 media opt root sbin sys usr volumes
root@VM:/# ifconfig
br-c2bd4b2f9127: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
inet 172.20.0.1 netmask 255.255.255.0 broadcast 172.20.0.255
inet6 fe80::42:ffff:fee0:5d64 prefixlen 64 scopeid 0x20<link>
ether 02:42:ff:e0:5d:64 txqueuelen 0 (Ethernet)
RX packets 46 bytes 1990 (1.9 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 99 bytes 9290 (9.2 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
br-f0264fb87f28: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.9.0.1 netmask 255.255.255.0 broadcast 10.9.0.255
inet6 fe80::42:9eff:fe18:1691 prefixlen 64 scopeid 0x20<link>
ether 02:42:9e:18:16:91 txqueuelen 0 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 43 bytes 5164 (5.1 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
ether 02:42:3b:6f:27:fd txqueuelen 0 (Ethernet)
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


Terminal
Terminal
Terminal
Terminal
Terminal
Terminal
Abdulfatah Abdillahi-Sat Jan 27-05:23:52 ~/.../Labsetup> docksh hostA-10.9.0.5
root@69e80ade5c0e:/# ls
bin boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr var
root@69e80ade5c0e:/# 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 78 bytes 8983 (8.9 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
root@69e80ade5c0e:/#



Lab Task Set 1: Using Scapy to Sniff and Spoof Packets

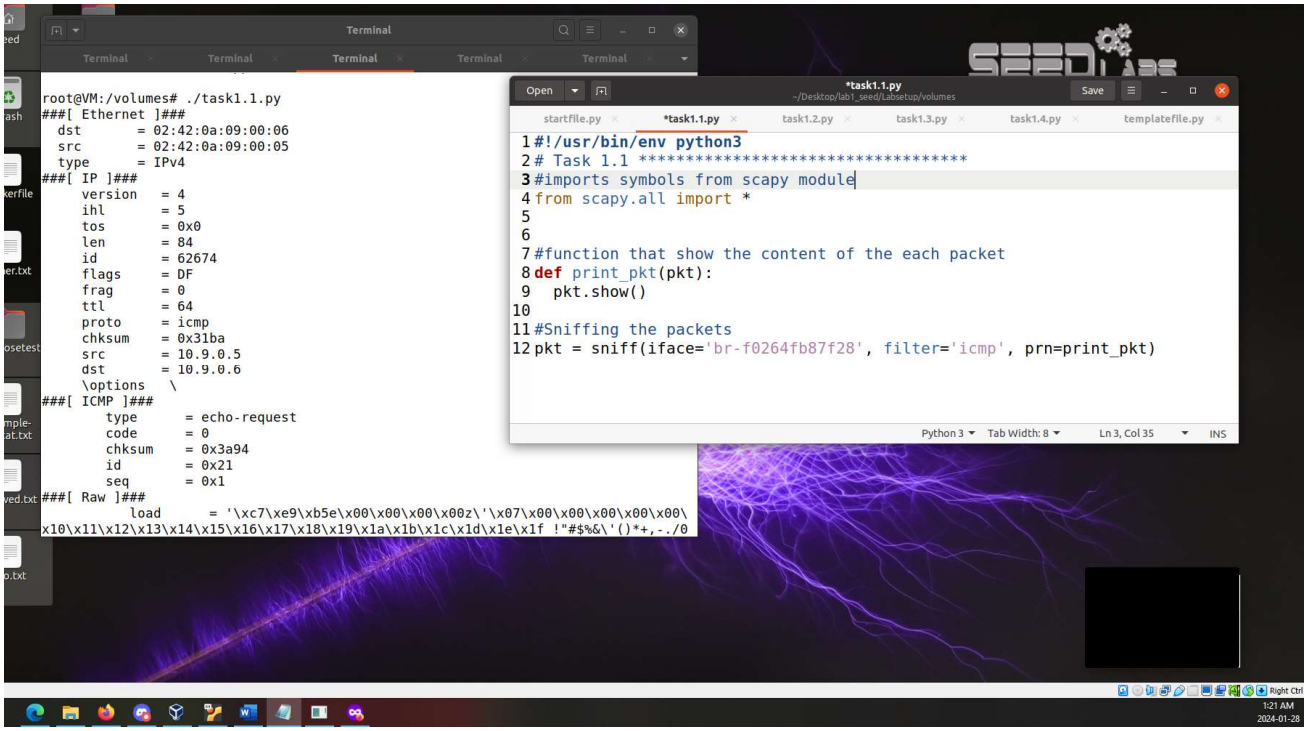
Typically, when using Scapy, we can write a python program and execute it using python. This is and example. Here, I am showing the program and its respective output. You can run the code using *python3* or as an executable.

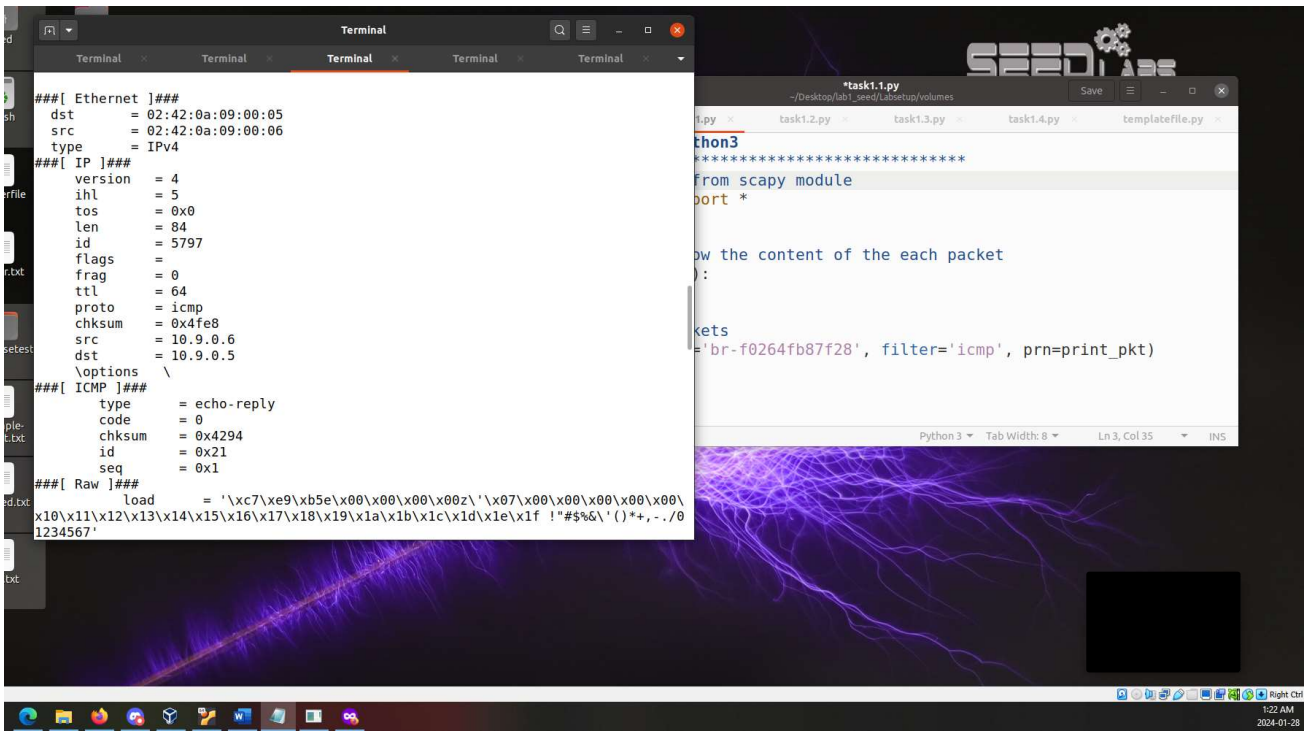


Task 1.1: Sniffing packets

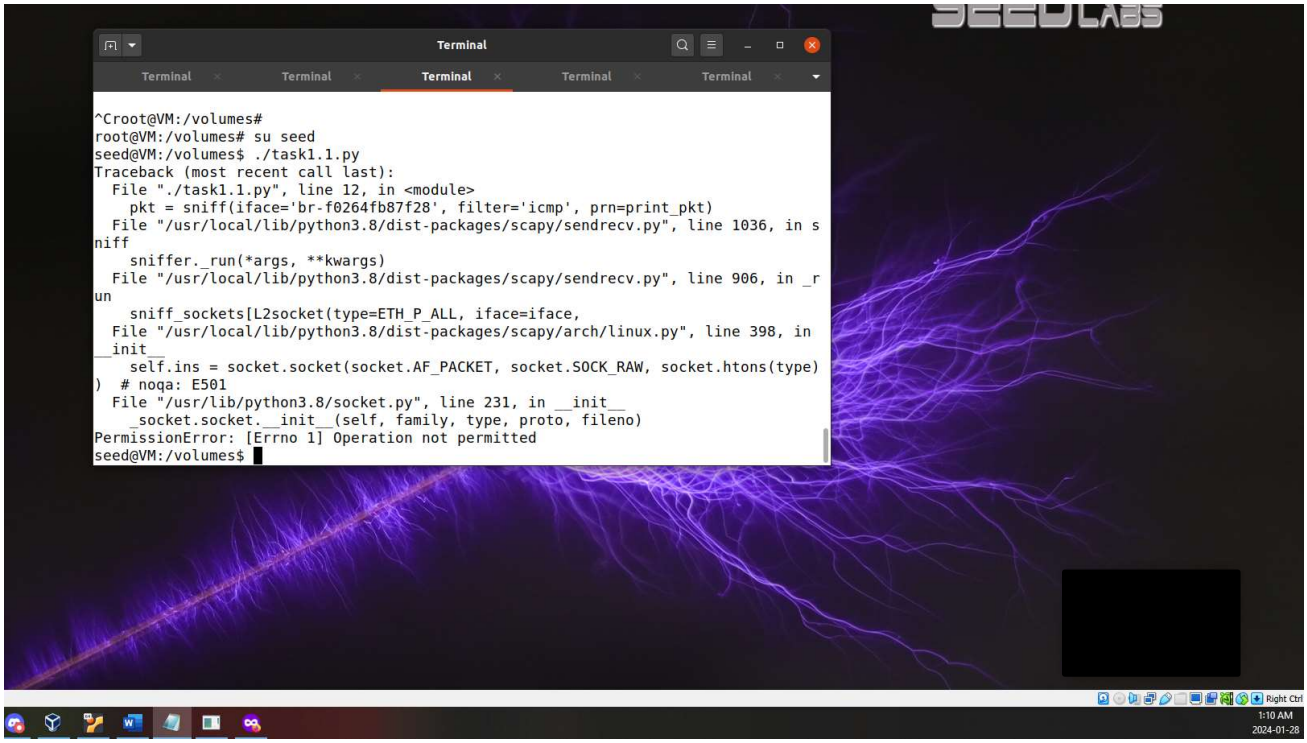
Task 1.1 A:

Here, I used the attacker container to sniff the ICMP packets going between Host A and B. Below you will se both the echo-request and the echo-reply associated with the communication as well as the program used.



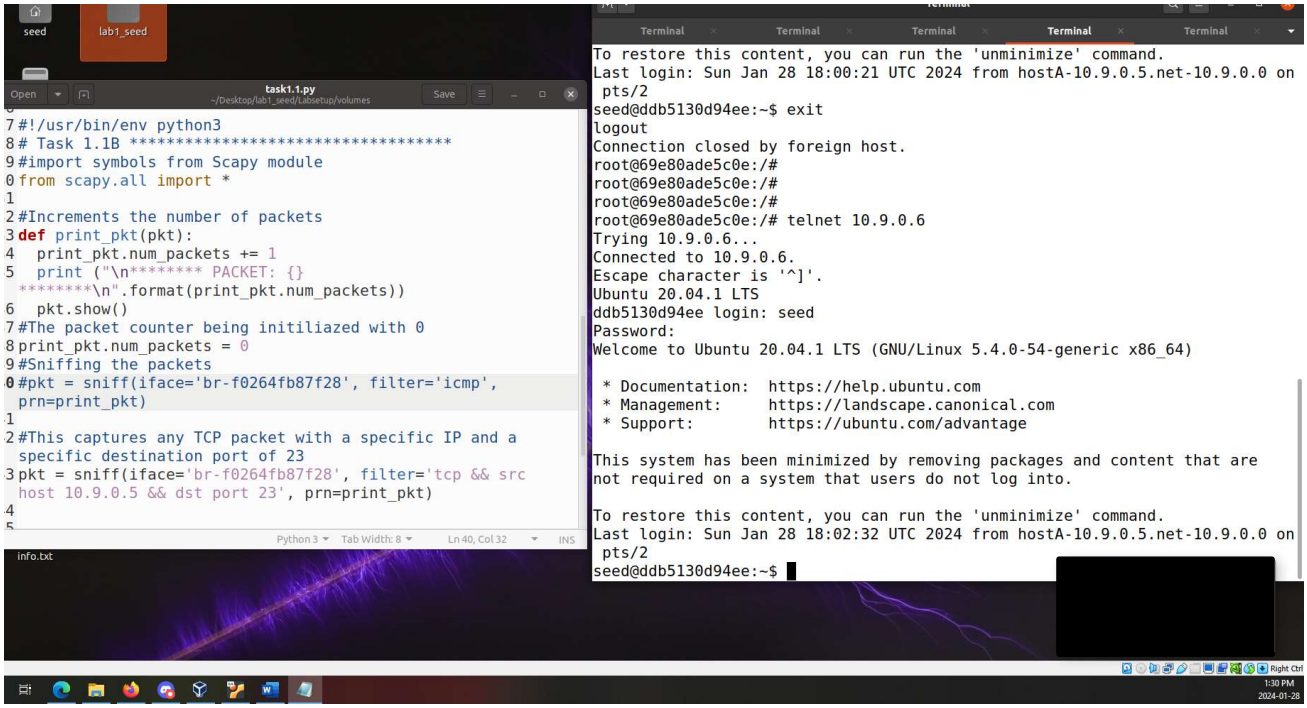


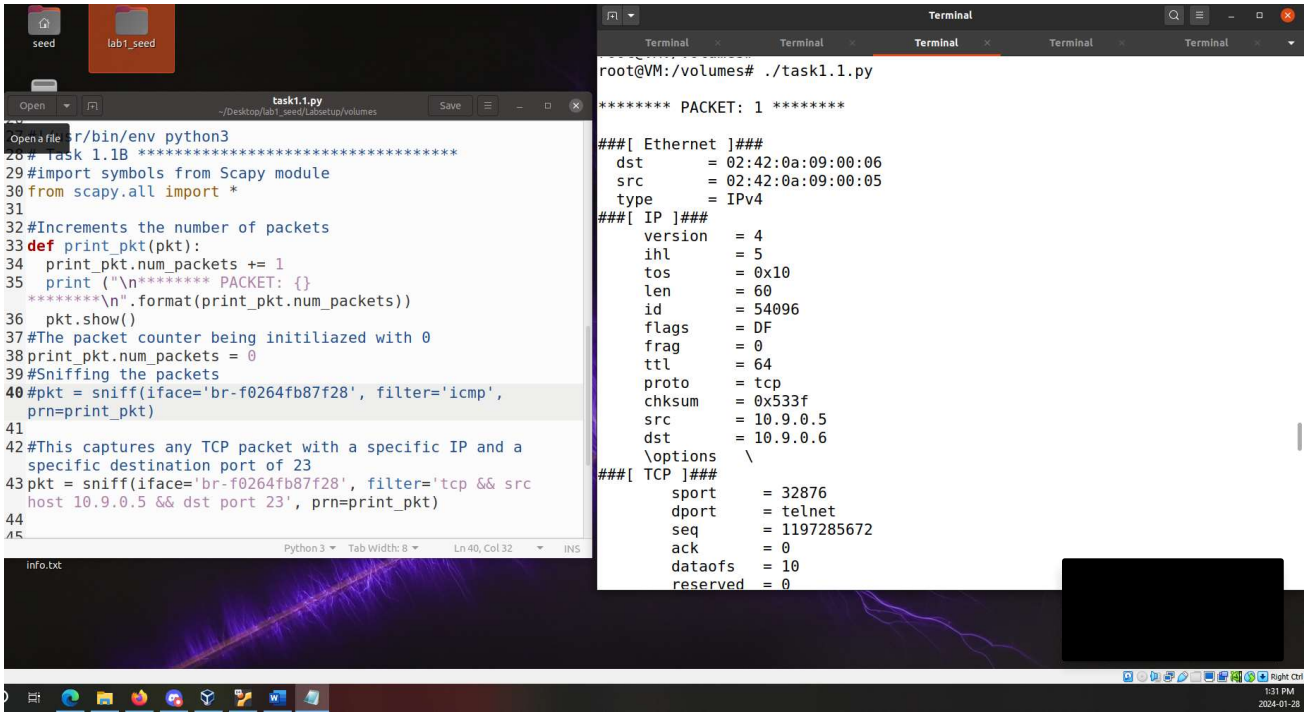
Here, I'm trying to run the same code again but this time without root privileges. I'm doing this by logging into the (unprivileged) SEED account. As you can see this did not work due to permission error.



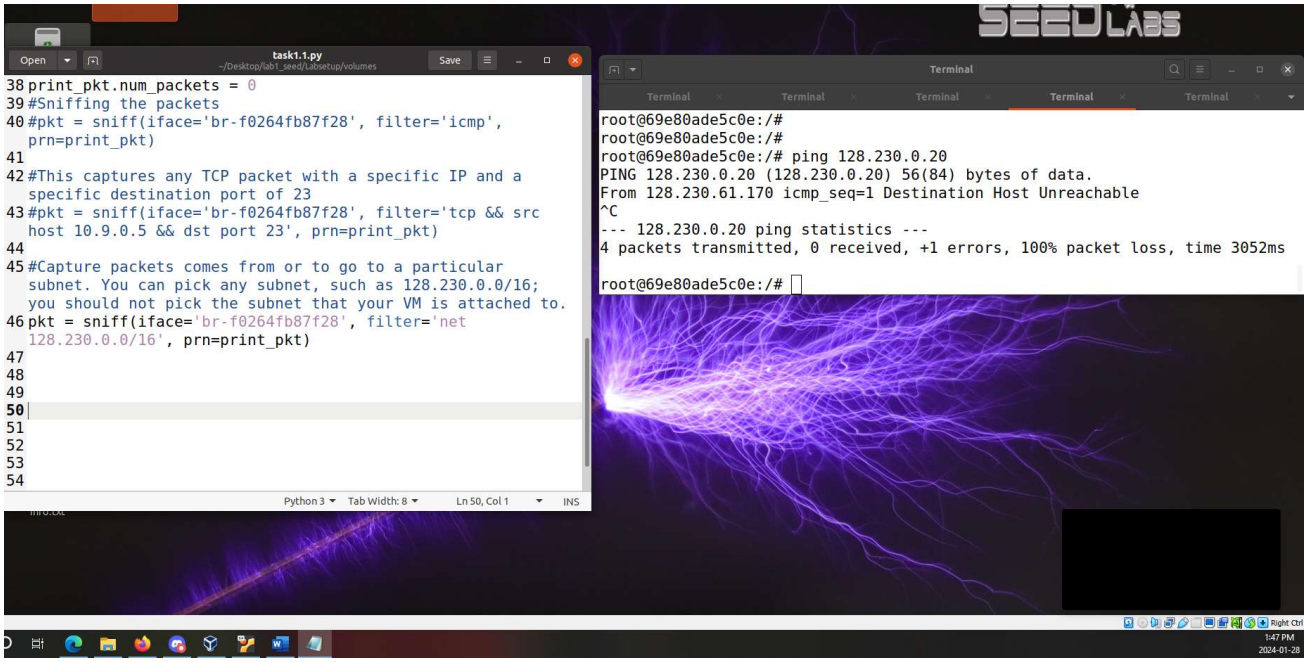
Task 1.1 B:

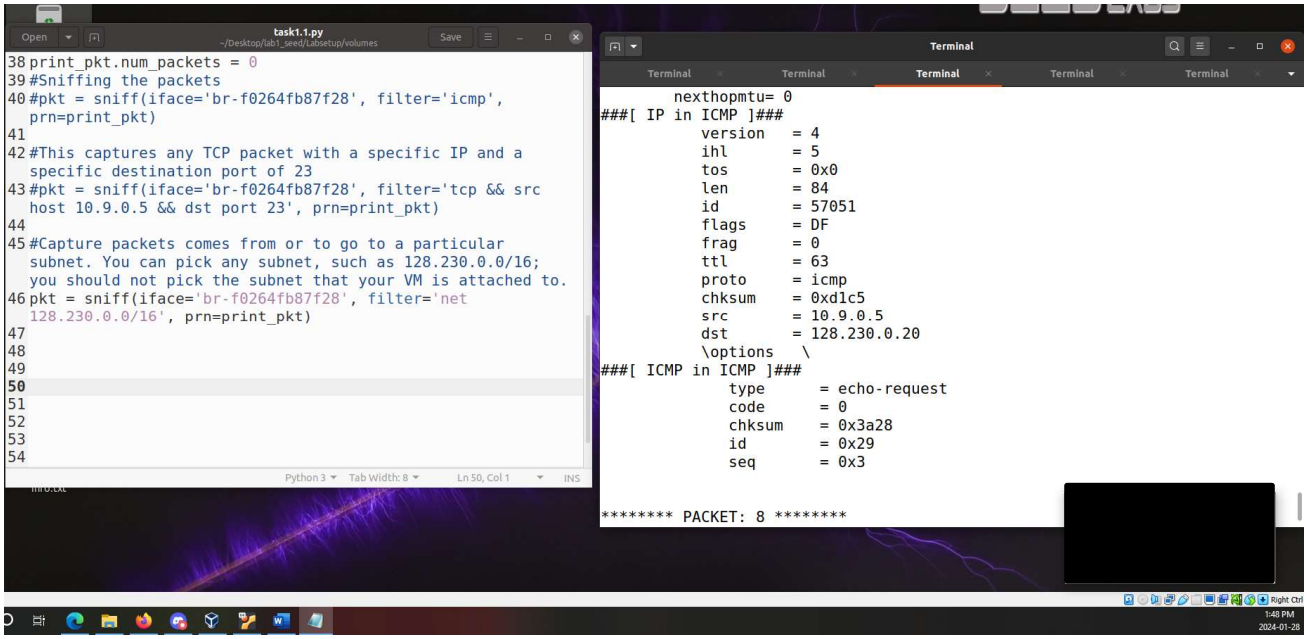
Here, I modified the original program used by the seed attacker to capture any TCP packet with a specific IP and a specific destination port of 23. In this case, I'm capturing packets from host A (10.9.0.5) communicating on telnet (port 23).





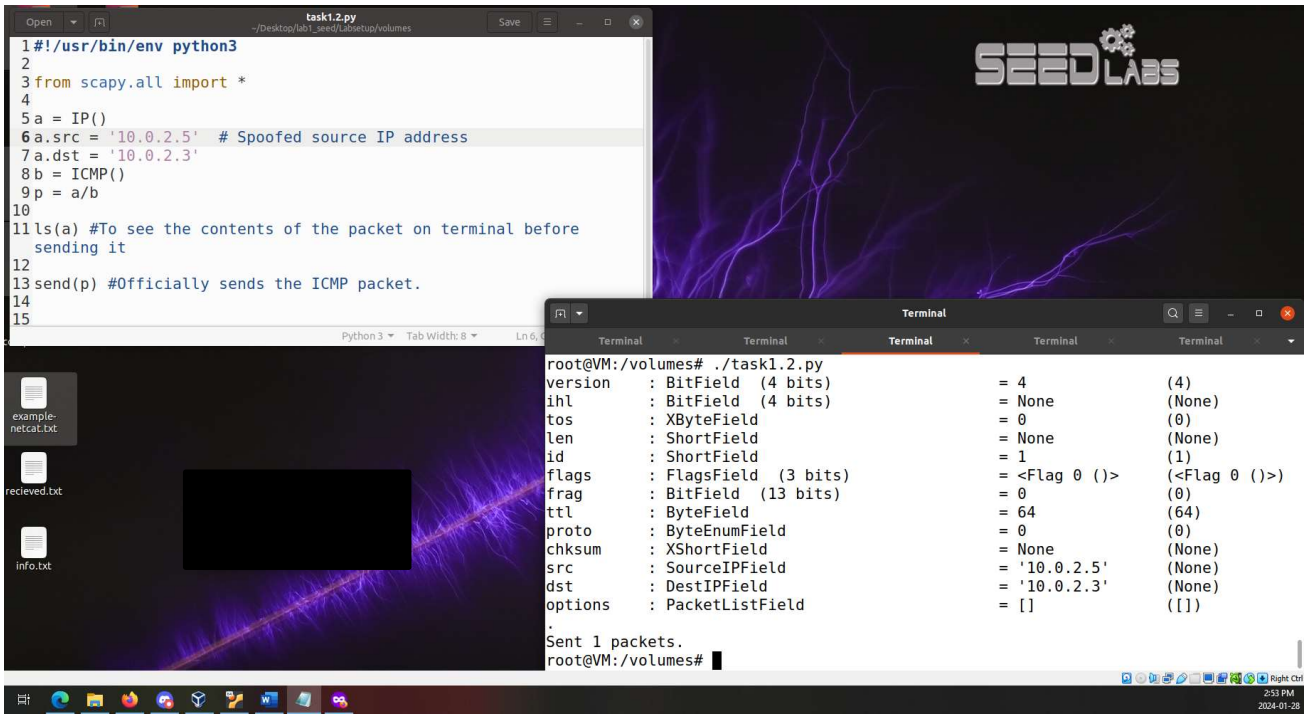
Here, I modified the program to capture packets coming from or going to a particular subnet. In this case, like the lab instruction suggested, I'm going with 128.230.0.0/16 network. I then proceeded to run the program on the attacker machine and ping the specified subnet from host A (but this could have also be done from host B).

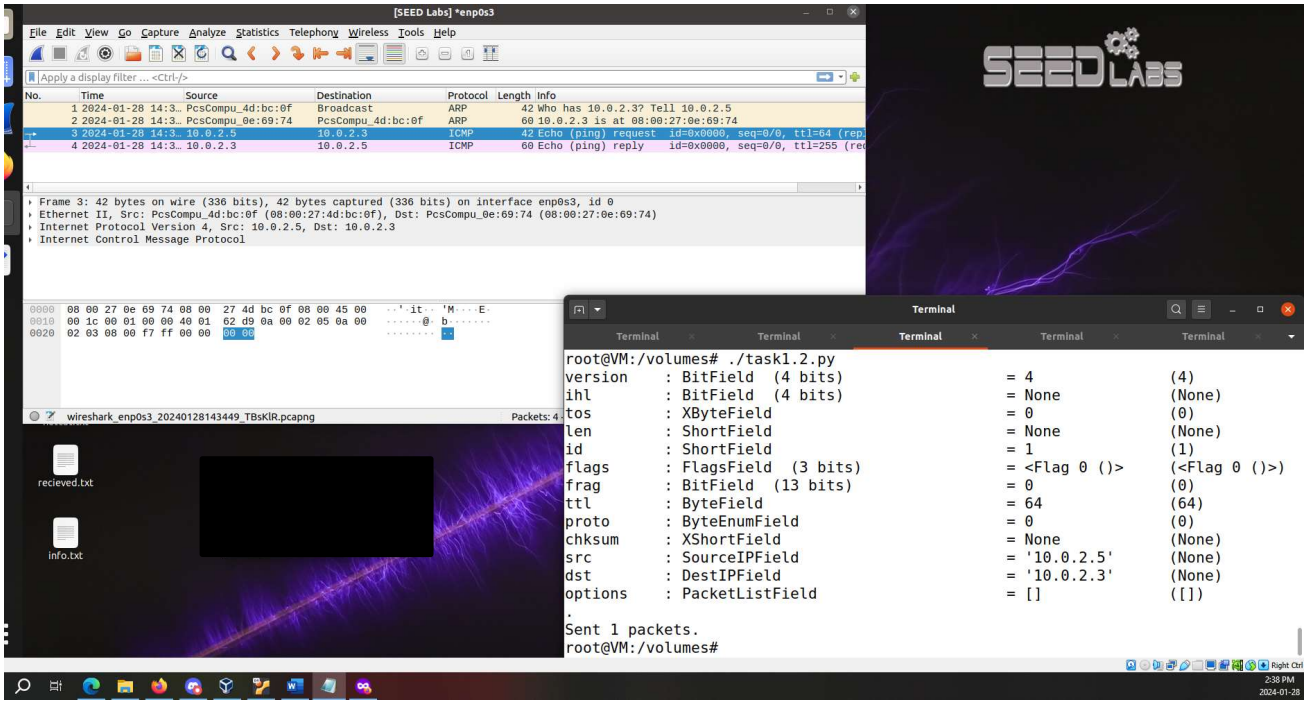




Task 1.2: Spoofing ICMP packets

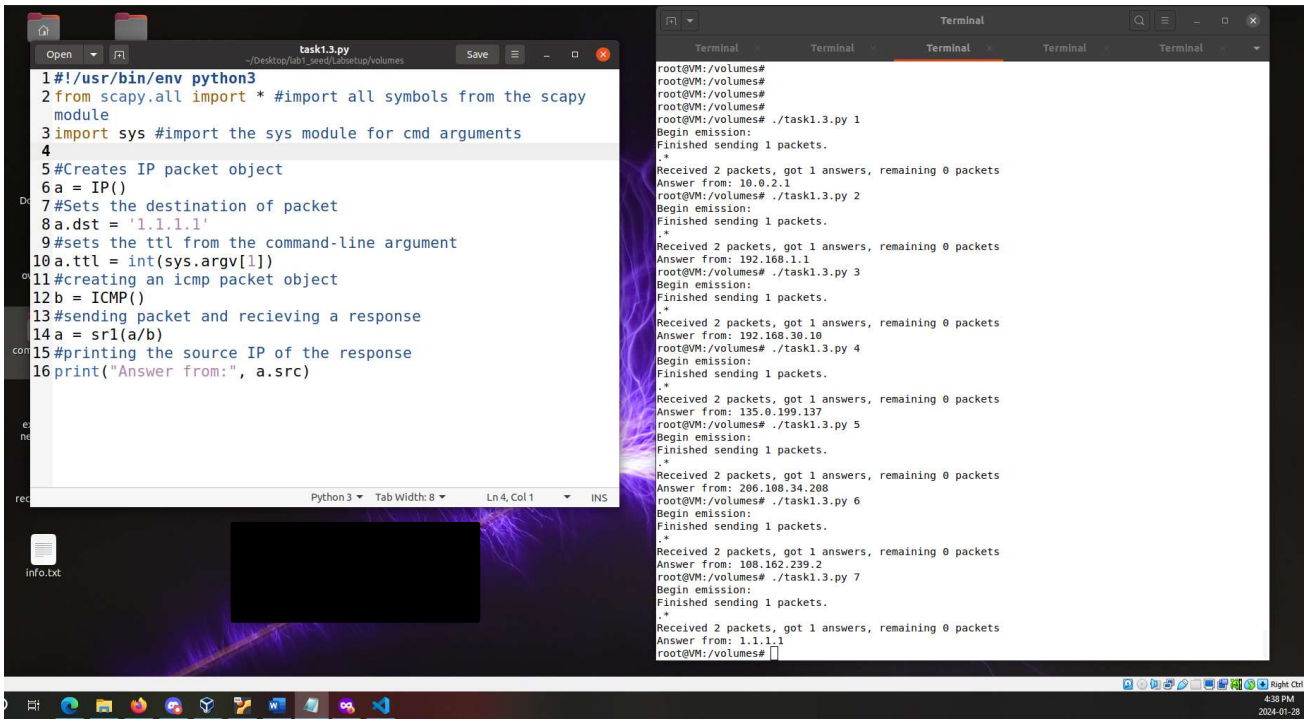
Here, I made the necessary changes from the sample code given in the lab to spoof our IP and send a casual ICMP request to the target. Using Wireshark, I have determined that this was successful by observing the echo-request and echo-reply to packets.

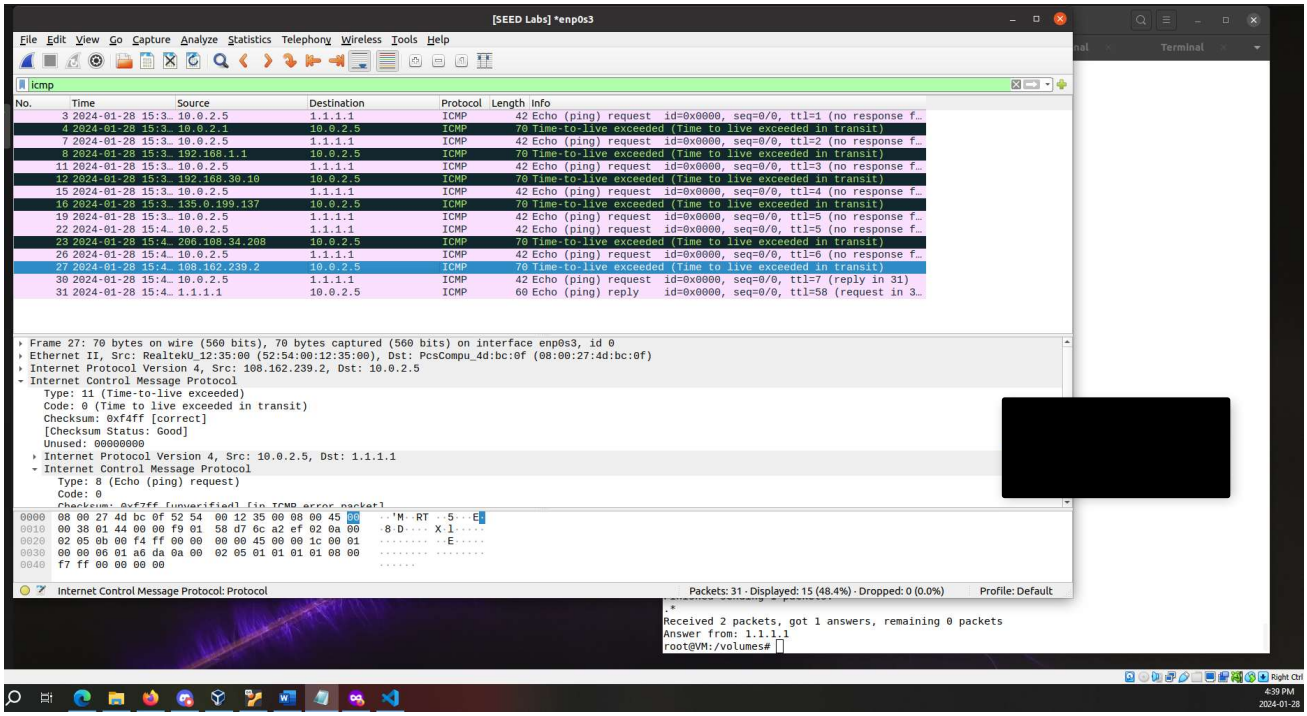




Task 1.3: Traceroute

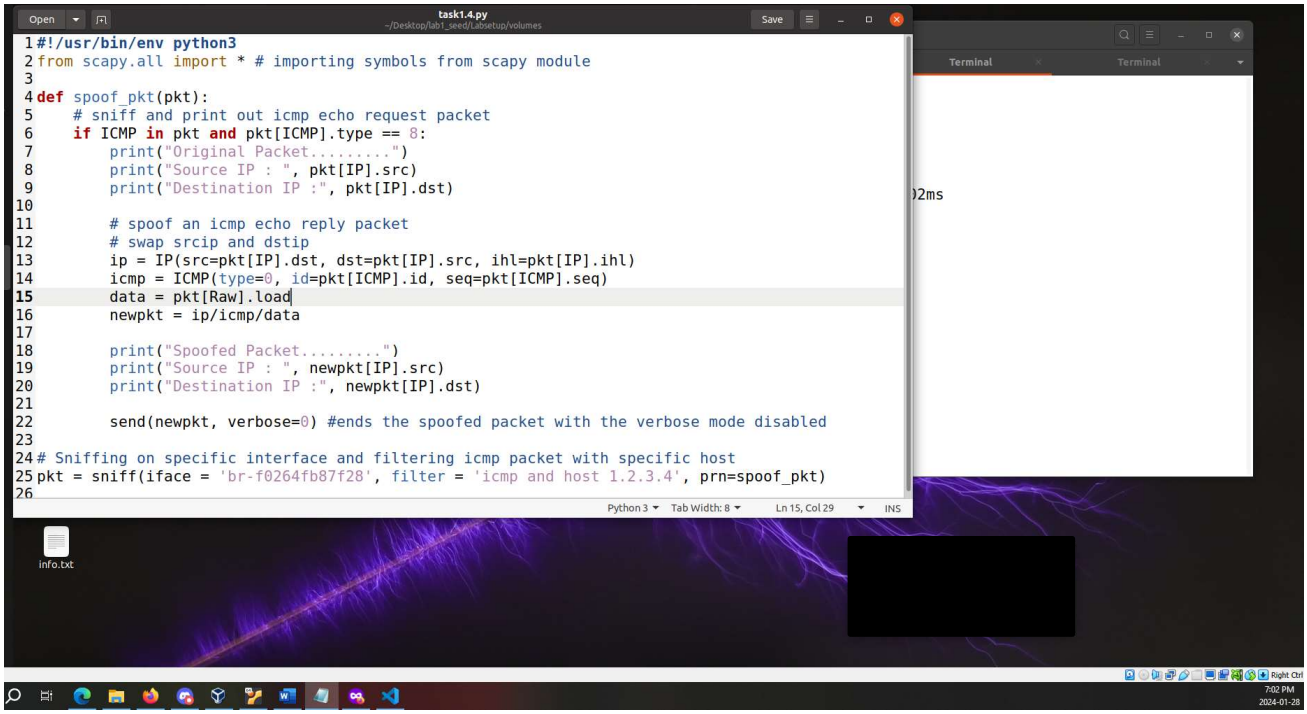
Here, I have slightly modified the original code as outlined in the lab instructions to a traceroute and find out the number of routers present between the source machine and the destination. In my case, my destination was Google's public DNS (1.1.1.1). I kept running the program until I got an answer from the specified destination. It took a total of 7 tries to get a response from the destination IP. In other words, there are 7 hops that the packet must go through to reach its destination. I have also shown the what the traffic looks like on Wireshark as this is happening.

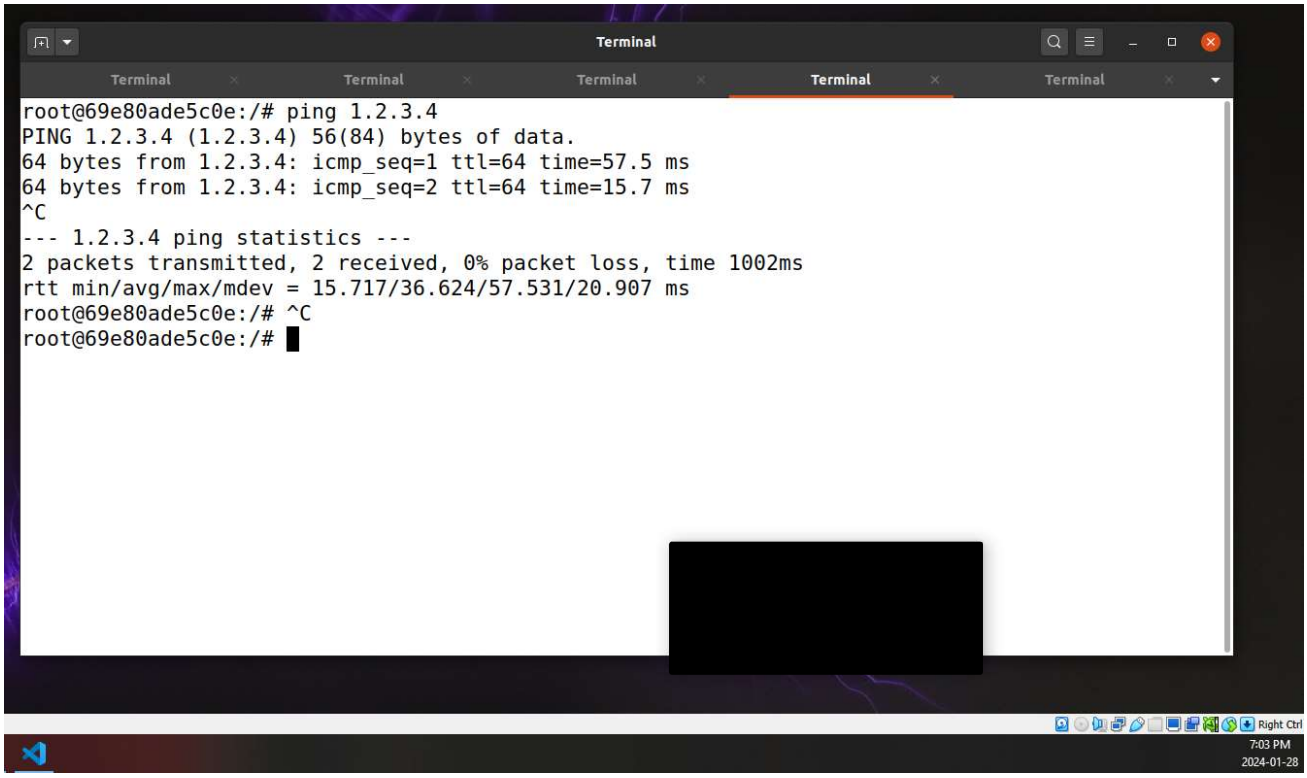


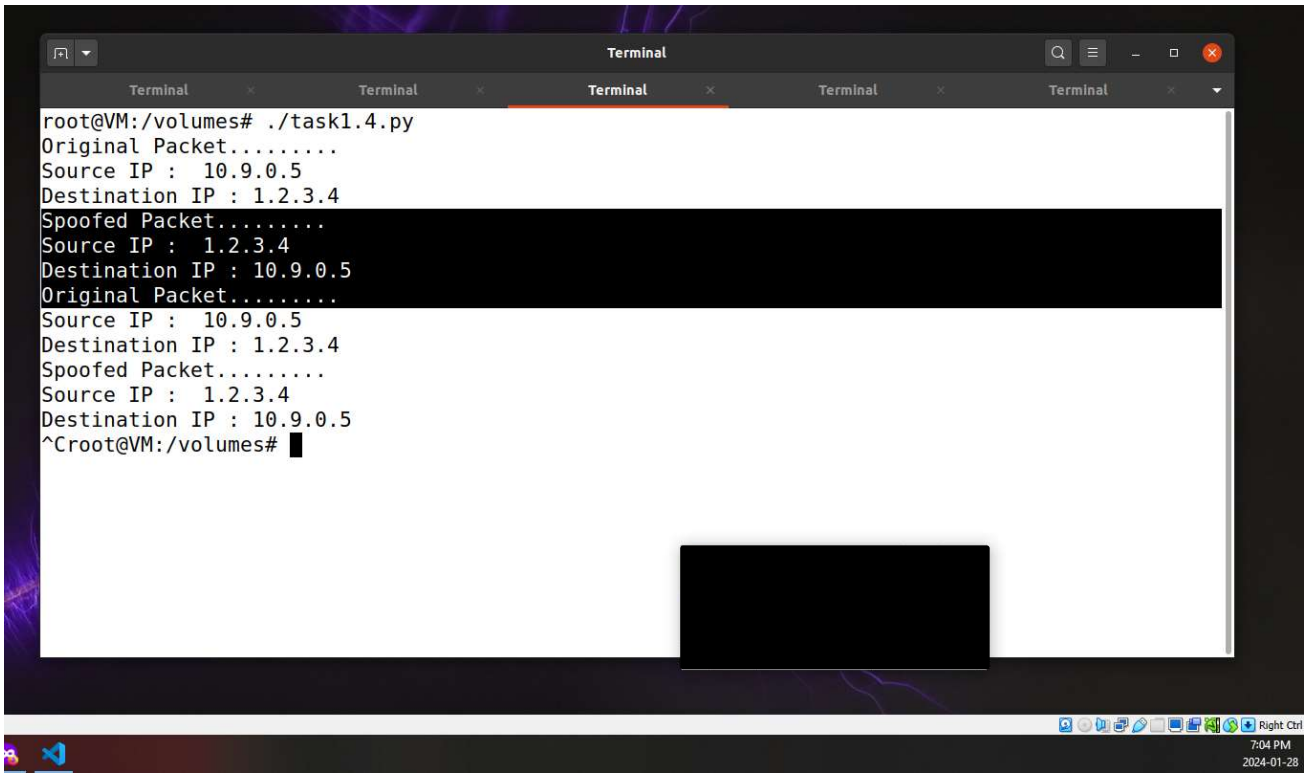


Task 1.4: Sniffing and-then Spoofing

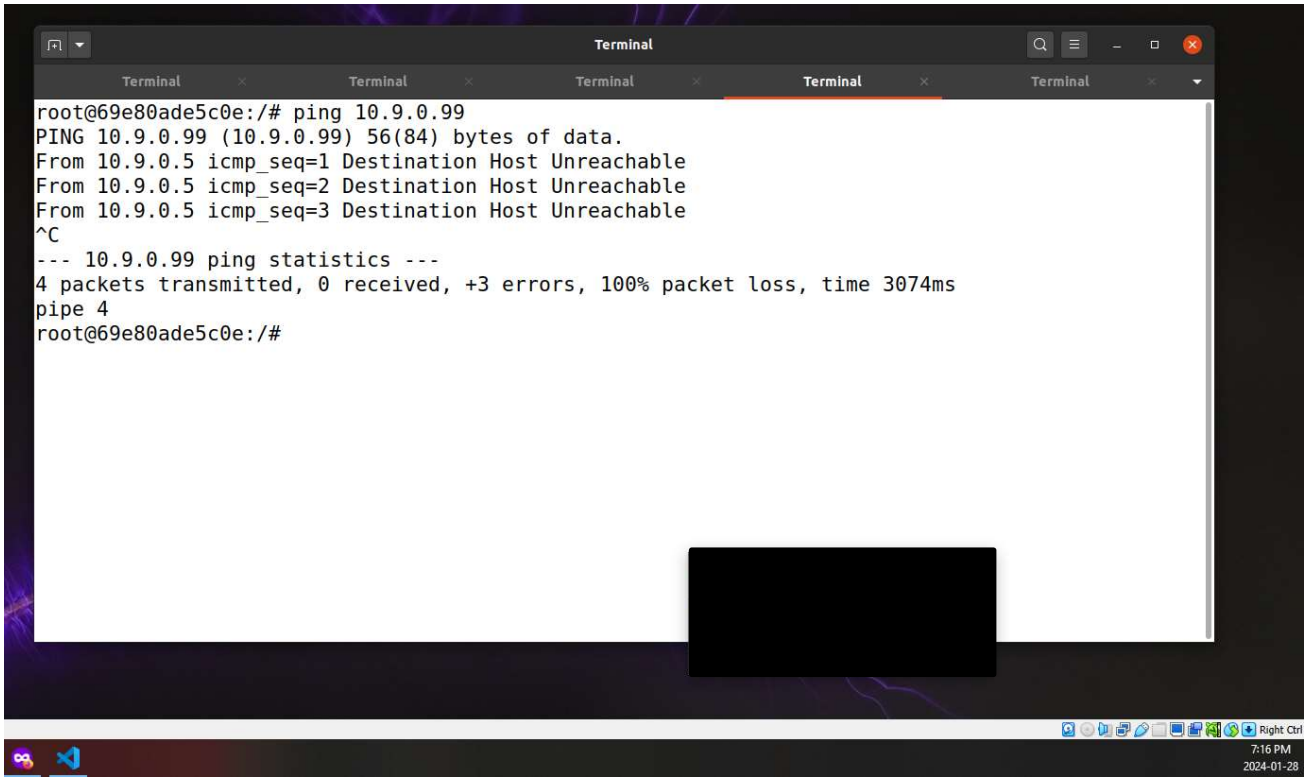
Here, I implemented the sniffing and spoofing program. The sniffing and spoofing was conducted on the attacker container while the ping was done on the host A container. For this example, if you look below, you will find that I have ran program specifying the IP 1.2.3.4 and successfully spoofed a reply. If you look closely, you will see that there were 2 echo-replies both of which were created by the attacker vm.

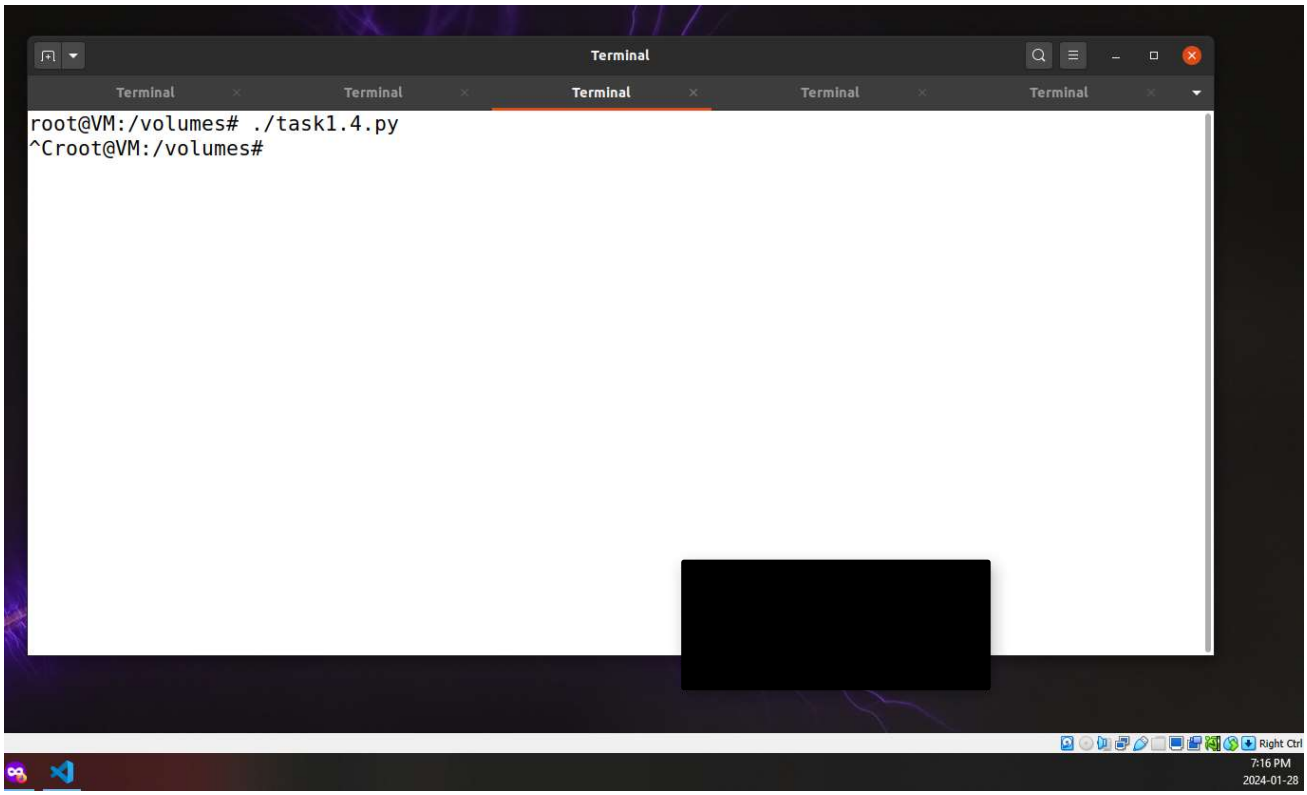




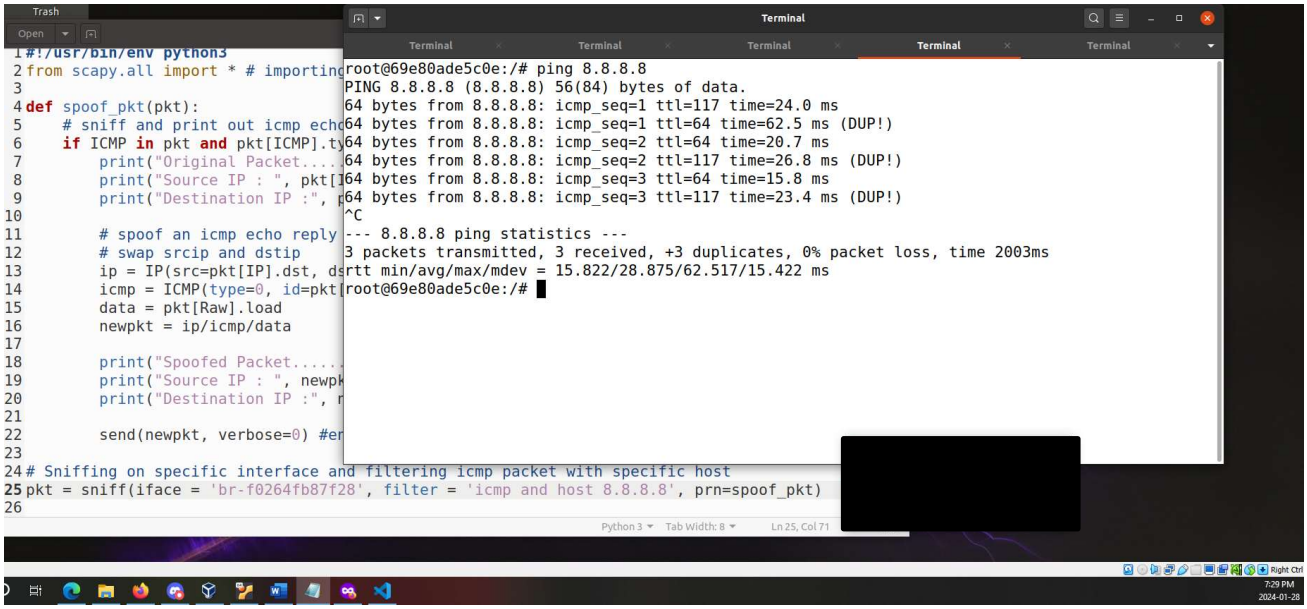


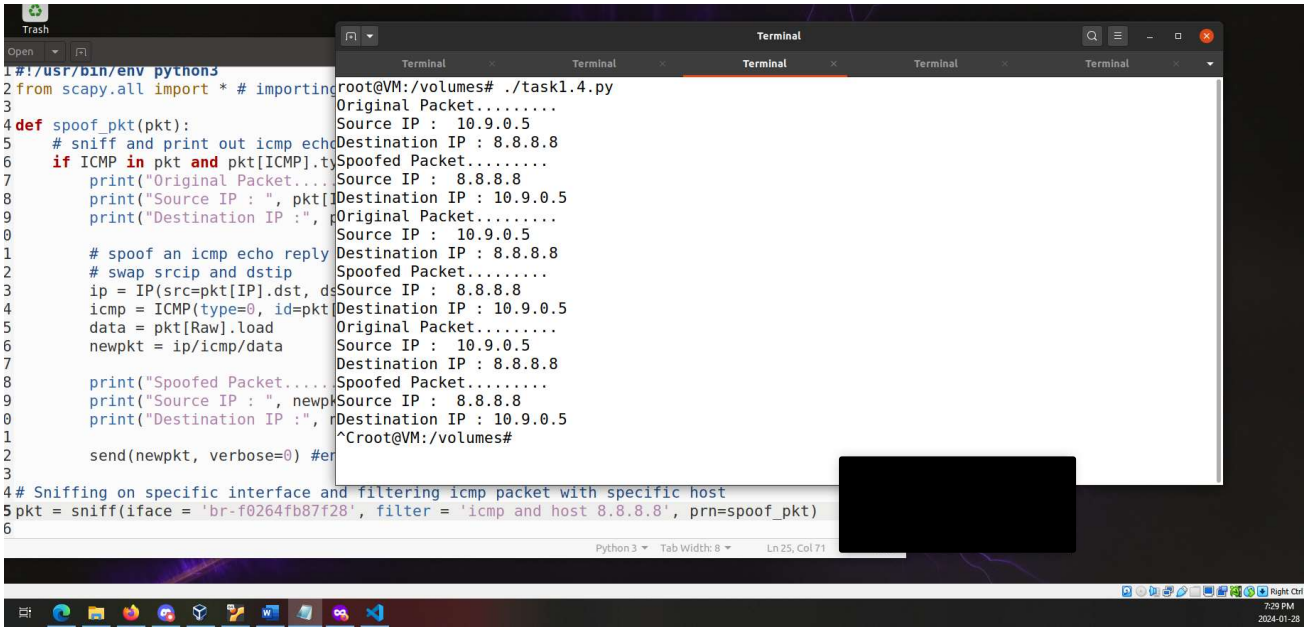
Here, I used the same program but this time to ping a non-existent host on the LAN (10.9.0.99). This resulted in errors such as “Destination Host Unreachable”. This is due to the fact that there is no ARP record mapping a mac-address to the IP 10.90.0.99 in the LAN. Similarly, the attacker container wasn’t also able to sniff any packets.





Here, I used the same program but this time to ping an existing host on the internet (8.8.8.8). Because this si a real, existing address we received 2 replies instead of just one (one from the spoofed packet done by attacker vm and one from the real host on the internet). This is why you see "(DUP!)" meaning duplicate on the output of the terminal.


Trash
Open
#!/usr/bin/env python3
from scapy.all import * # importing
def spoof_pkt(pkt):
# sniff and print out icmp echo
if ICMP in pkt and pkt[ICMP].ty
print("Original Packet....
print("Source IP : ", pkt[I
print("Destination IP :",
# spoof an icmp echo reply
# swap srcip and dstip
ip = IP(src=pkt[IP].dst, ds
icmp = ICMP(type=0, id=pkt[
data = pkt[Raw].load
newpkt = ip/icmp/data
print("Spoofed Packet.....
print("Source IP : ", newpk
print("Destination IP :",
send(newpkt, verbose=0) #er
# Sniffing on specific interface and filtering icmp packet with specific host
pkt = sniff(iface = 'br-f0264fb87f28', filter = 'icmp and host 8.8.8.8', prn=spoof_pkt)
Terminal
root@69e80ade5c0e:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=117 time=24.0 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=64 time=62.5 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=20.7 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=117 time=26.8 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=15.8 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=117 time=23.4 ms (DUP!)
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, +3 duplicates, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 15.822/28.875/62.517/15.422 ms
root@69e80ade5c0e:/#
Python 3
Tab Width: 8
Ln 25, Col 71
7:29 PM
2024-01-28


Trash
Open
#!/usr/bin/env python3
from scapy.all import * # importing
def spoof_pkt(pkt):
# sniff and print out icmp echo
if ICMP in pkt and pkt[ICMP].ty
print("Original Packet....
print("Source IP : ", pkt[I
print("Destination IP :",
# spoof an icmp echo reply
# swap srcip and dstip
ip = IP(src=pkt[IP].dst, ds
icmp = ICMP(type=0, id=pkt[
data = pkt[Raw].load
newpkt = ip/icmp/data
print("Spoofed Packet.....
print("Source IP : ", newpk
print("Destination IP :",
send(newpkt, verbose=0) #er
# Sniffing on specific interface and filtering icmp packet with specific host
pkt = sniff(iface = 'br-f0264fb87f28', filter = 'icmp and host 8.8.8.8', prn=spoof_pkt)
Terminal
root@VM:/volumes# ./task1.4.py
Original Packet.....
Source IP : 10.9.0.5
Destination IP : 8.8.8.8
Spoofed Packet.....
Source IP : 8.8.8.8
Destination IP : 10.9.0.5
Original Packet.....
Source IP : 10.9.0.5
Destination IP : 8.8.8.8
Spoofed Packet.....
Source IP : 8.8.8.8
Destination IP : 10.9.0.5
Original Packet.....
Source IP : 10.9.0.5
Destination IP : 8.8.8.8
Spoofed Packet.....
Source IP : 8.8.8.8
Destination IP : 10.9.0.5
^Croot@VM:/volumes#
Python 3
Tab Width: 8
Ln 25, Col 71
7:29 PM
2024-01-28

References

Aria. (2018, September 11). *Netcat tutorial | introduction to netcat | cybersecurity certification training | edureka*. YouTube. https://www.youtube.com/watch?v=rdgv-EqyeAU

Du, W. (2006). *Packet sniffing and Spoofing lab*. seedsecuritylabs. https://seedsecuritylabs.org/Labs_20.04/Files/Sniffing_Spoofing/Sniffing_Spoofing.pdf

Introduction — Scapy 2.4.4 documentation. (n.d.). Scapy.readthedocs.io. https://scapy.readthedocs.io/en/latest/introduction.html#about-scapy

subfuzion. (2018). *subfuzion/netcat*. Docker. https://hub.docker.com/r/subfuzion/netcat

Tan, Z. H. (2017, April 19). *A gentle introduction to netcat*. Medium. https://medium.com/@tzhenghao/a-gentle-introduction-to-netcat-3e0b105dfbd9