



PACKET SNIFFING AND SPOOFING



JANUARY 2024
Abdulfatah Abdillahi

Contents

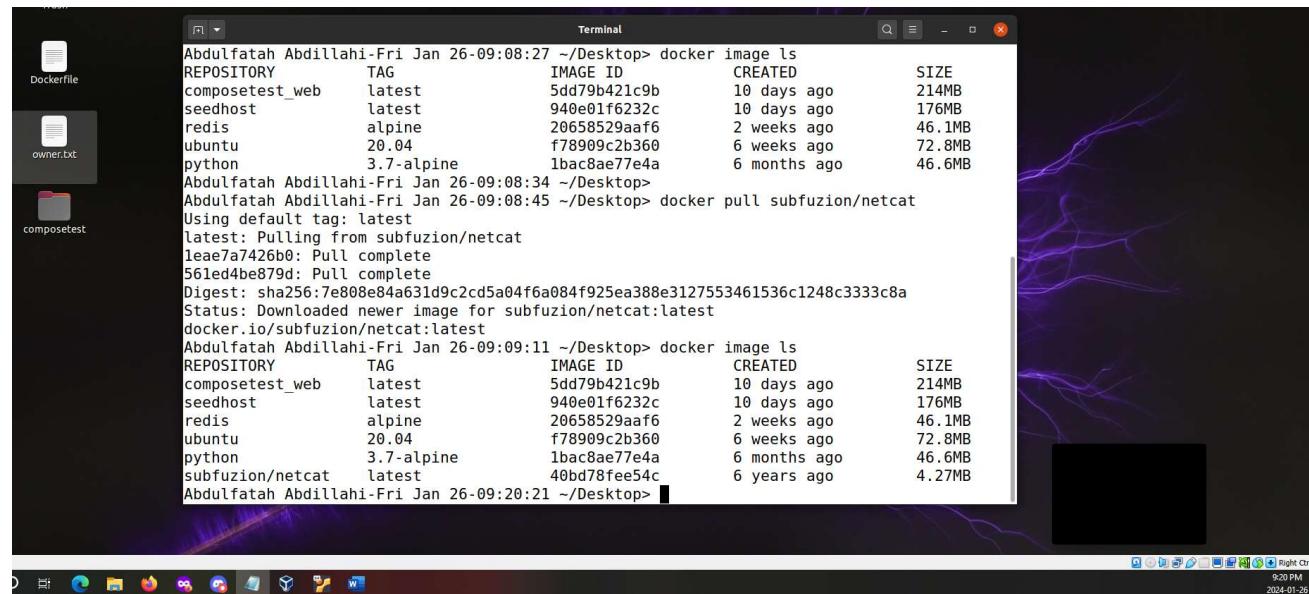
Introduction	3
Part 1: Netcat	3
Part 2: SEED: Packet Sniffing and Spoofing Lab	7
References	19

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.



The screenshot shows a Linux desktop environment with a terminal window open. The terminal window displays the following command and its output:

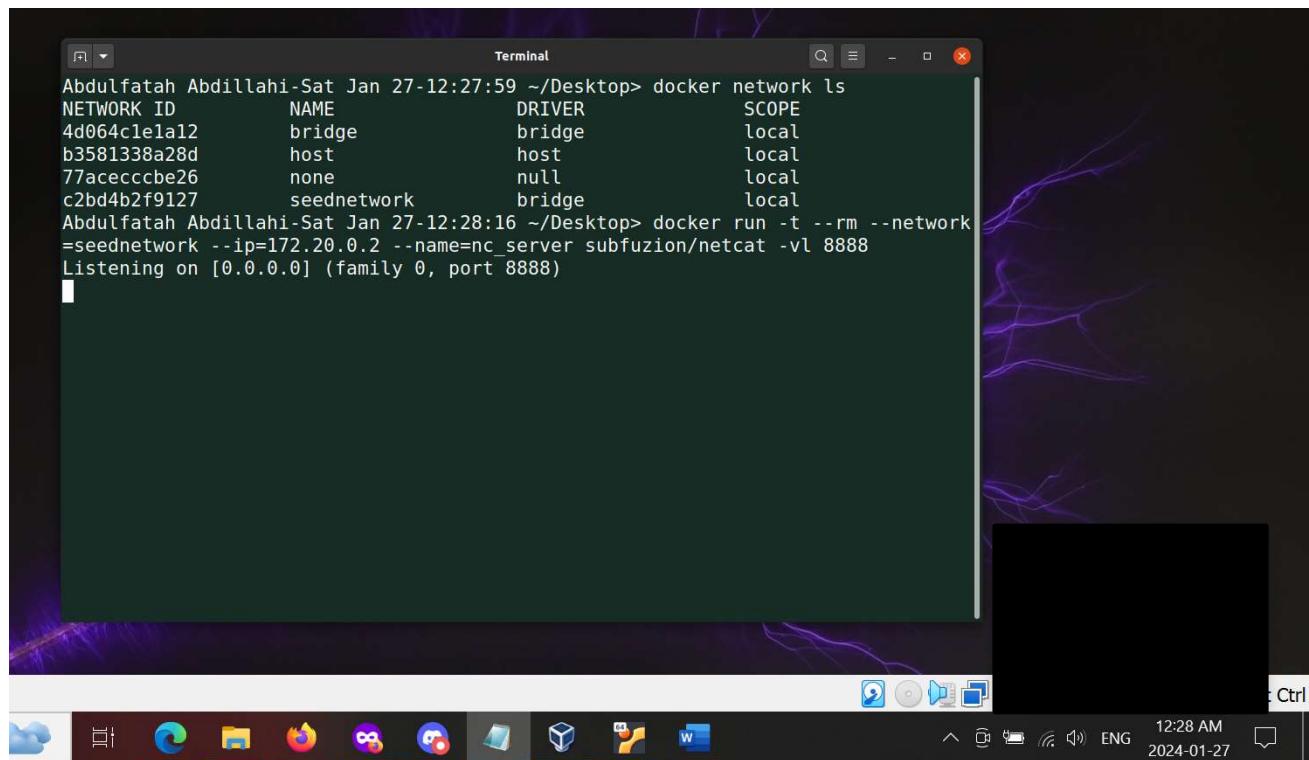
```
Abdulfatah Abdillahi-Fri Jan 26-09:08:27 ~/Desktop> docker image ls
REPOSITORY      TAG        IMAGE ID       CREATED          SIZE
composetest_web latest     5dd79b421c9b   10 days ago    214MB
seedhost         latest     940e01f6232c   10 days ago    176MB
redis            alpine     20658529aaaf6  2 weeks ago    46.1MB
ubuntu           20.04     f78909c2b360   6 weeks ago    72.8MB
python           3.7-alpine 1bac8ae77e4a   6 months ago   46.6MB

Abdulfatah Abdillahi-Fri Jan 26-09:08:34 ~/Desktop>
Abdulfatah Abdillahi-Fri Jan 26-09:08:45 ~/Desktop> docker pull subfuzion/netcat
Using default tag: latest
latest: Pulling from subfuzion/netcat
1eaef7a7426b0: Pull complete
561ed4be879d: Pull complete
Digest: sha256:7e808e84a631d9c2cd5a04f6a084f925ea388e3127553461536c1248c3333c8a
Status: Downloaded newer image for subfuzion/netcat:latest
docker.io/subfuzion/netcat:latest

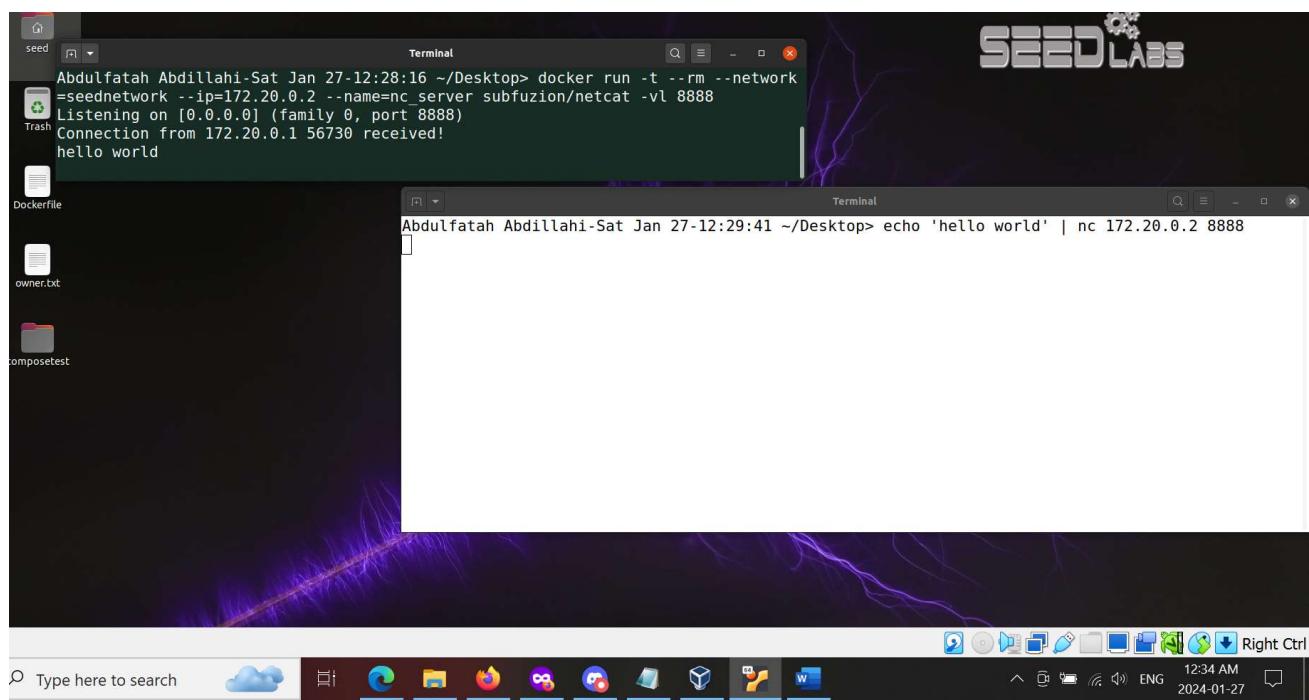
Abdulfatah Abdillahi-Fri Jan 26-09:09:11 ~/Desktop> docker image ls
REPOSITORY      TAG        IMAGE ID       CREATED          SIZE
composetest_web latest     5dd79b421c9b   10 days ago    214MB
seedhost         latest     940e01f6232c   10 days ago    176MB
redis            alpine     20658529aaaf6  2 weeks ago    46.1MB
ubuntu           20.04     f78909c2b360   6 weeks ago    72.8MB
python           3.7-alpine 1bac8ae77e4a   6 months ago   46.6MB
subfuzion/netcat latest     40bd78fee54c   6 years ago    4.27MB

Abdulfatah Abdillahi-Fri Jan 26-09:20:21 ~/Desktop>
```

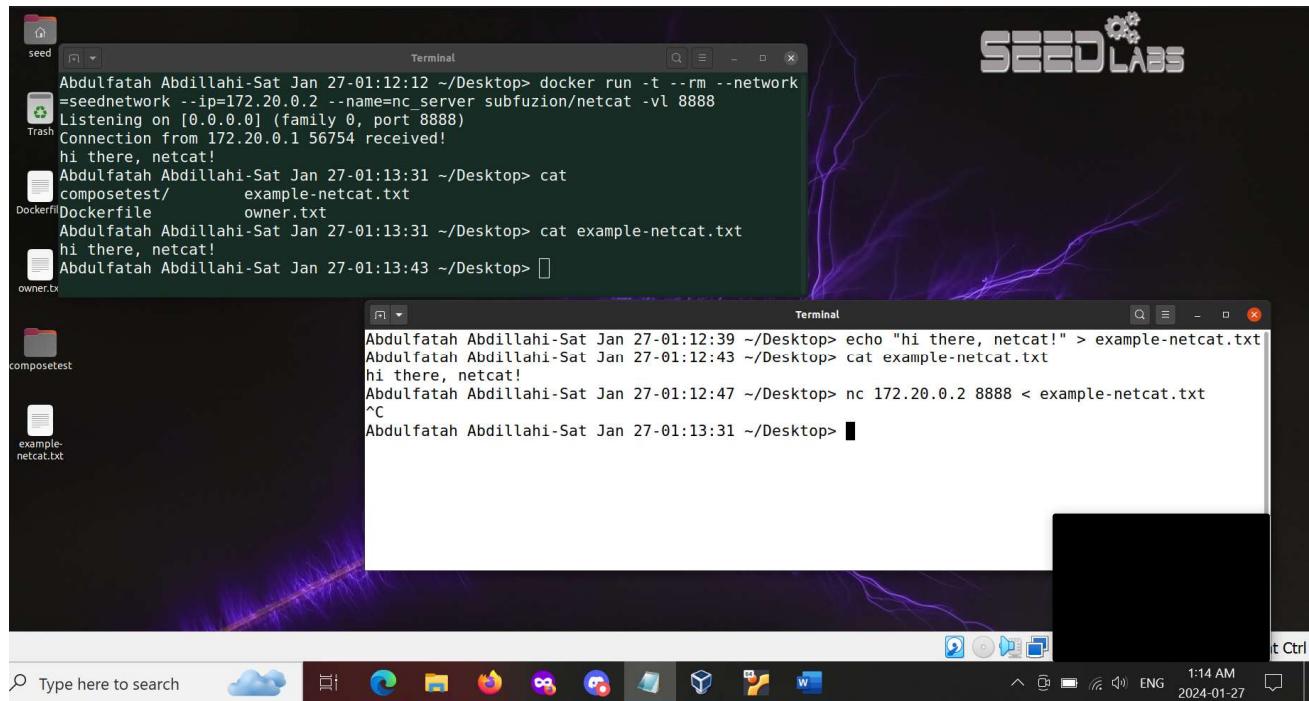
Here, I was able to run the Netcat server container using the command `docker run -t --rm --network=seednetwork --ip=172.20.0.2 --name=nc_server subfuzion/netcat -vl 8888` where I utilized the previously made network in lab 0 (seednetwork). Also, the ip assigned to this server is 172.20.0.2 with a port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



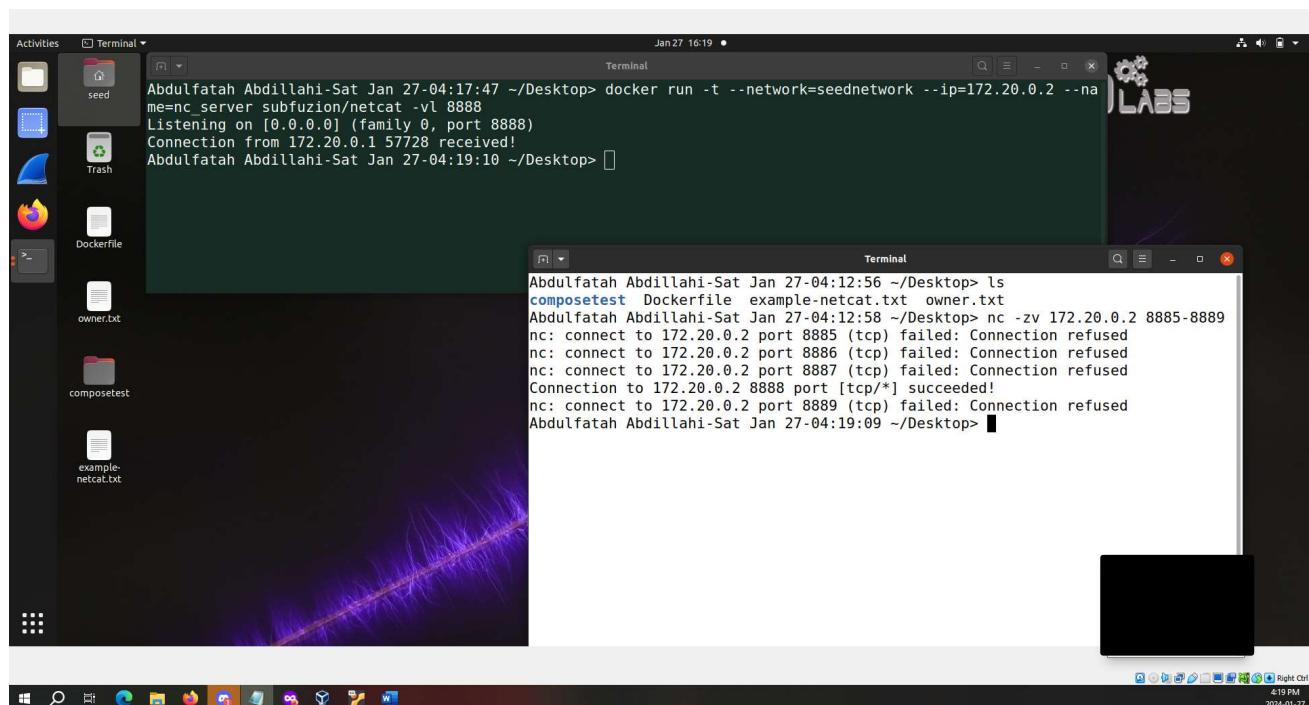
Here, I'm sending “Hello World” to the netcat server with IP 172.20.0.2 and port 8888.



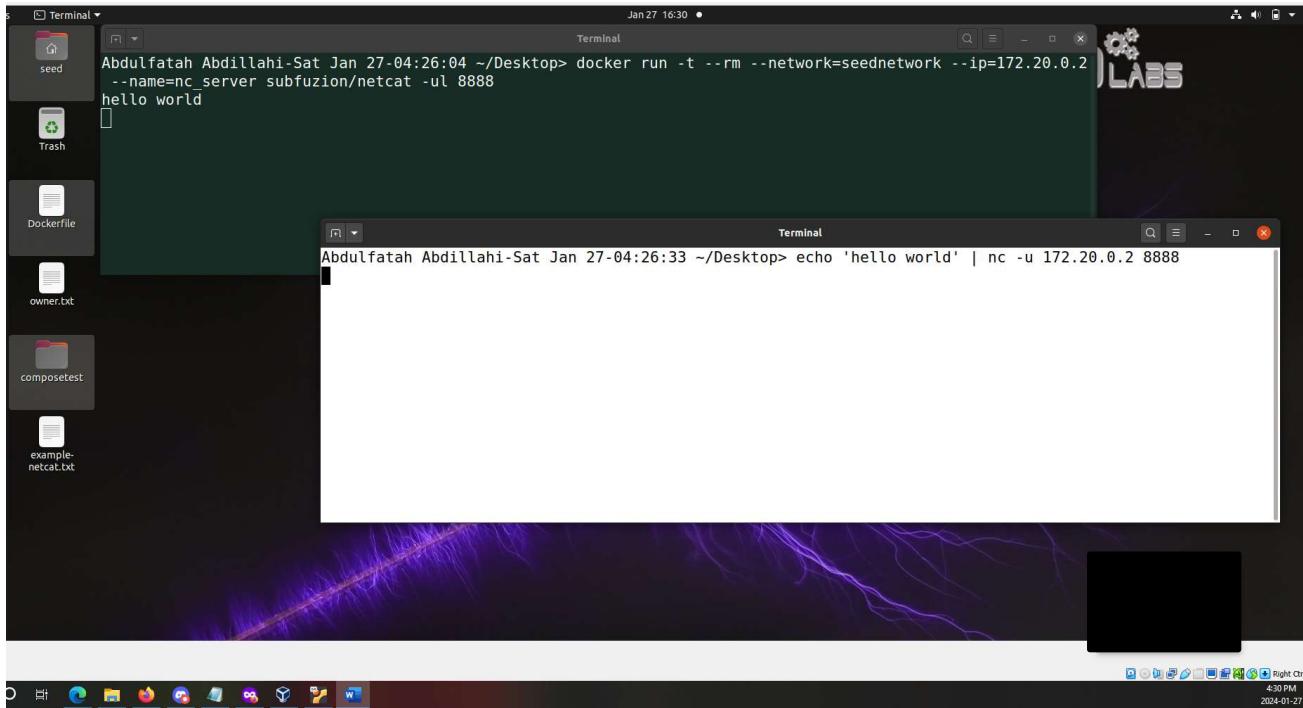
Here, I created a file called “example-netcat.txt” on the client side and sent it to the listener on the other side. On the left you can see a confirmation that the file was indeed received along with its contents.



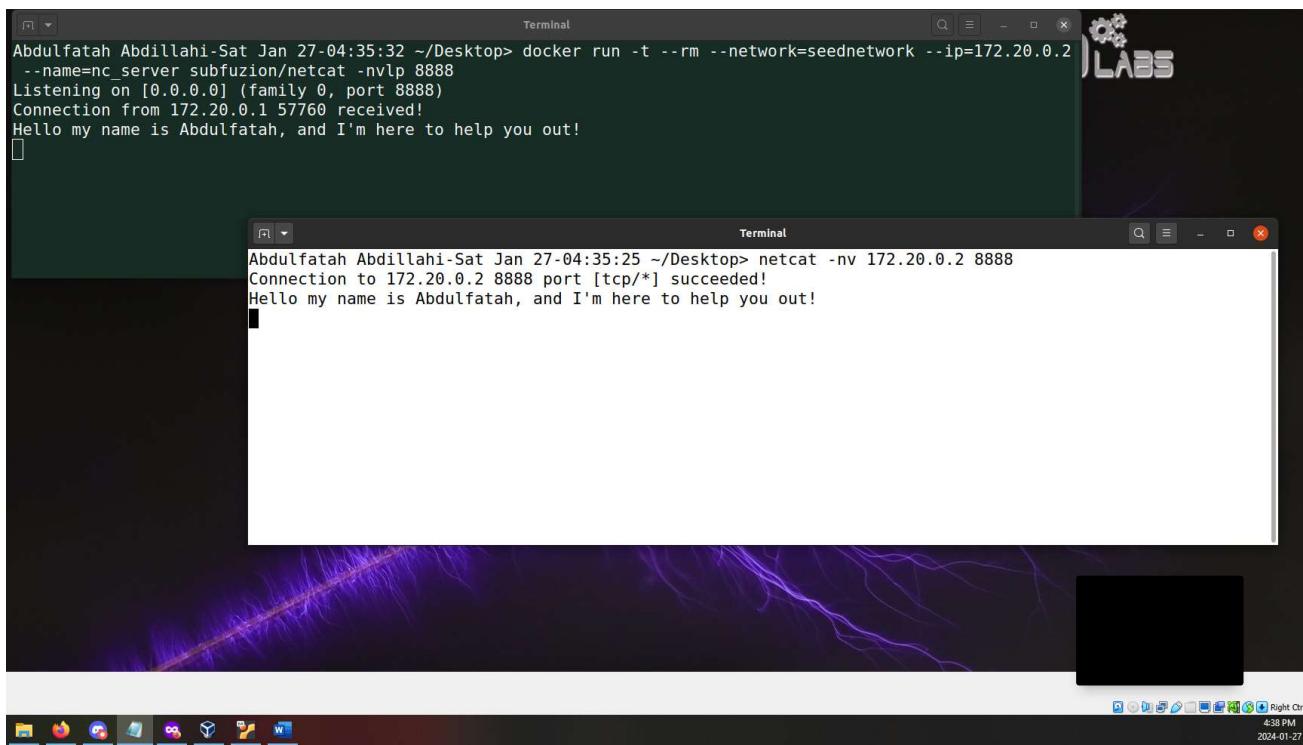
Here, we are attempting to use netcat for port scanning.



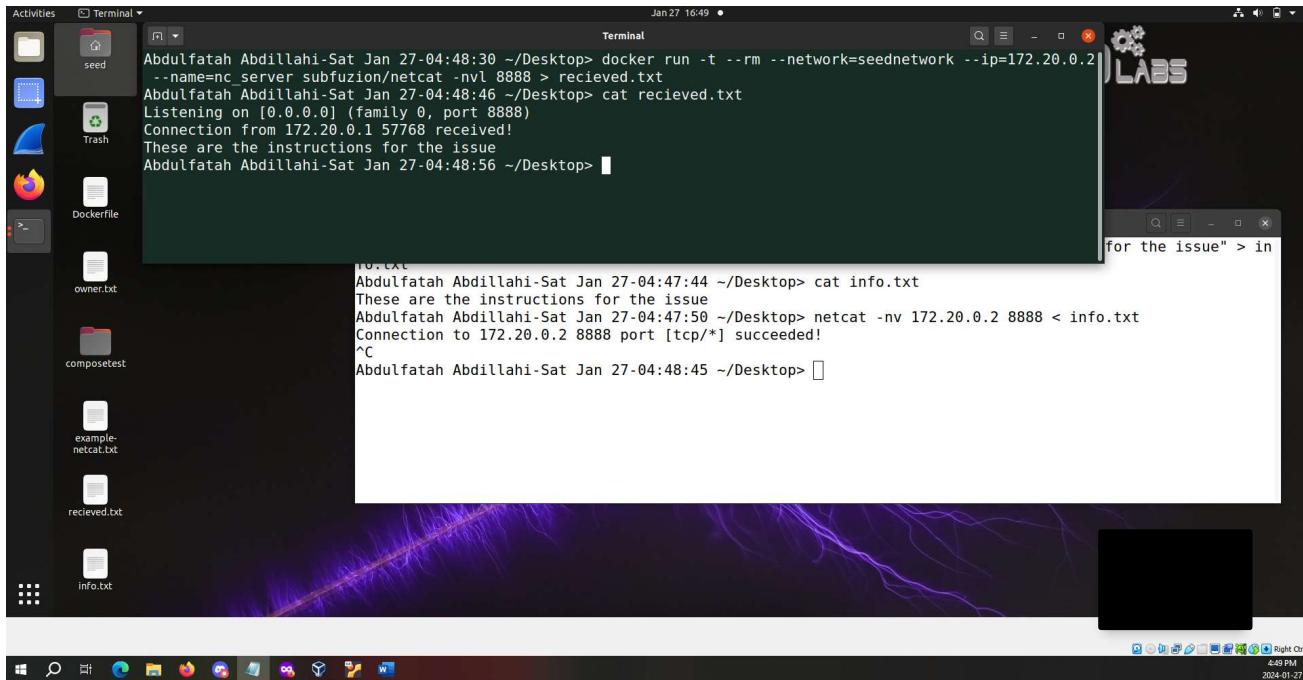
Here, we are attempting to send UDP packets with nc (as it is commonly used with tcp)



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

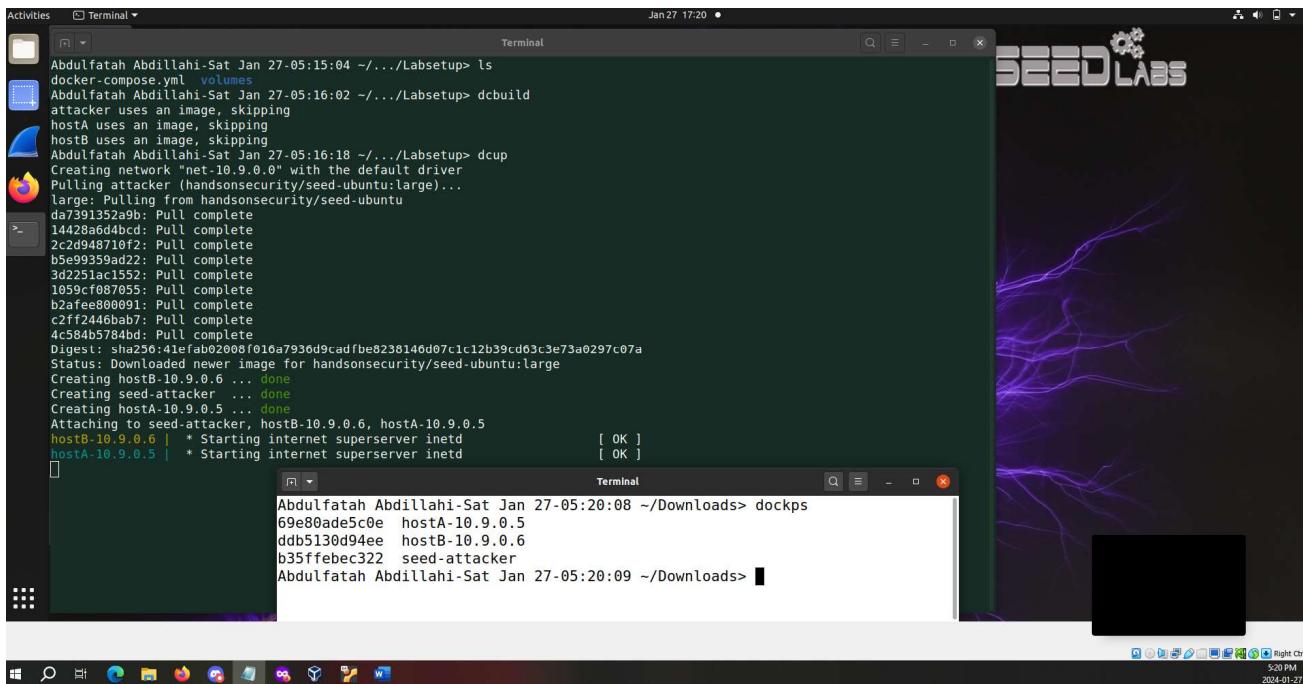


Here, in this scenario we are pretending to send and receive files to help clients with an issue for example.

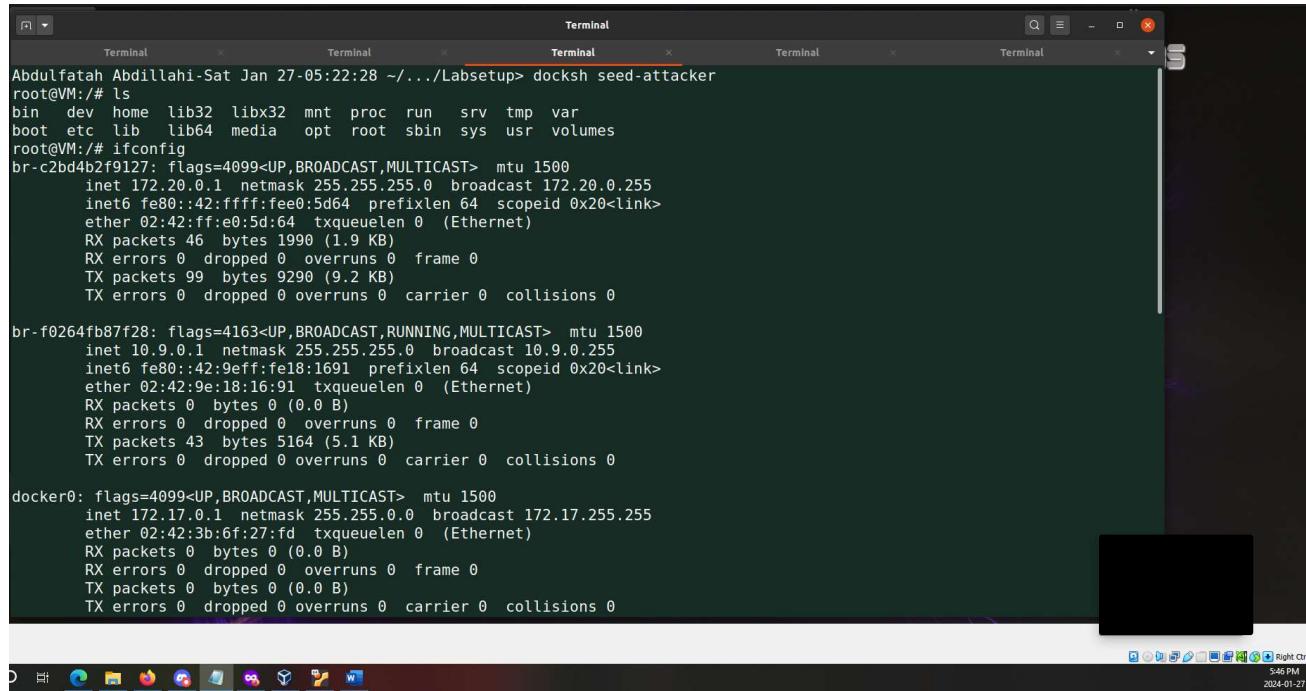


Part 2: SEED: Packet Sniffing and Spoofing Lab

Creating three docker containers



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



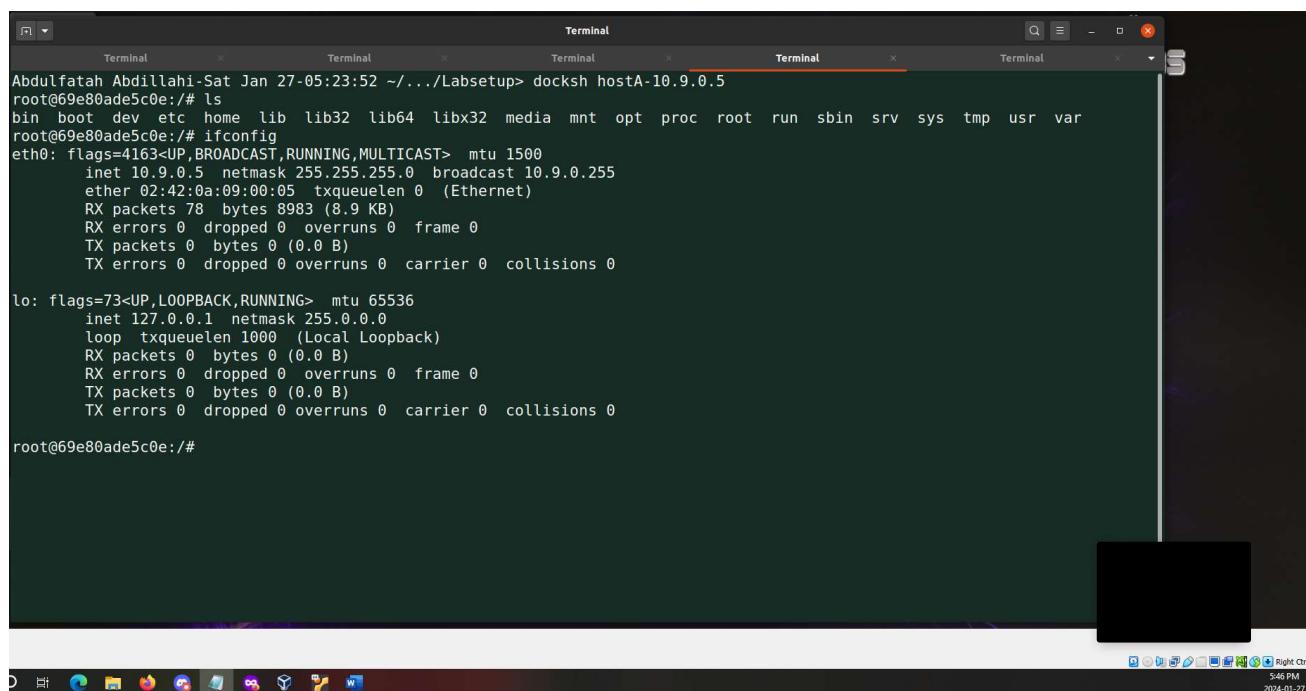
```
Terminal Terminal Terminal Terminal Terminal
Abdulfatah Abdillahi-Sat Jan 27-05:22:28 ~/.../Labsetup> docksh seed-attacker
root@VM:/# ls
bin dev home lib32 libx32 mnt proc run srv tmp var
boot etc lib lib64 media opt root sbin sys usr volumes
root@VM:/# ifconfig
br-c2bd4b2f9127: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.20.0.1 netmask 255.255.255.0 broadcast 172.20.0.255
        inet6 fe80::42:ffff:fe0:5d64 prefixlen 64 scopeid 0x20<link>
            ether 02:42:ff:0e: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

docksh0: 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

Right Ctrl
2024-01-27
```



```
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:/#
```

```

Terminal
Terminal
Terminal
Terminal
Terminal
Terminal

Abdulfatah Abdillahi-Sat Jan 27 05:44:27 ~.../Labsetup> dockersh hostB-10.9.0.6
dockersh: command not found
Abdulfatah Abdillahi-Sat Jan 27 05:44:57 ~.../Labsetup> docksh hostB-10.9.0.6
root@ddb5130d94ee:/# ifocnfig
bash: ifocnfig: command not found
root@ddb5130d94ee:/# 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 79 bytes 9093 (9.0 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@ddb5130d94ee:/#

```

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 an example. Here, I am showing the program and its respective output. You can run the code using `python3` or as an executable.

```

startfile.py [Read-Only]
~/Desktop/lab1_sniff/Labsetup/volumes

1#!/usr/bin/env python3
2from scapy.all import *
3
4a = IP()
5a.show()
6

root@VM:/volumes#
root@VM:/volumes# python3 startfile.py
###[ IP ]###
version = 4
ihl = None
tos = 0x0
len = None
id = 1
flags =
frag = 0
ttl = 64
proto = hopopt
checksum = None
src = "127.0.0.1"
dst = "127.0.0.1"
\options \
root@VM:/volumes# ./startfile.py
###[ IP ]###
version = 4
ihl = None
tos = 0x0
len = None
id = 1
flags =
frag = 0
ttl = 64
proto = hopopt
checksum = None
src = "127.0.0.1"
dst = "127.0.0.1"
\options \
root@VM:/volumes#

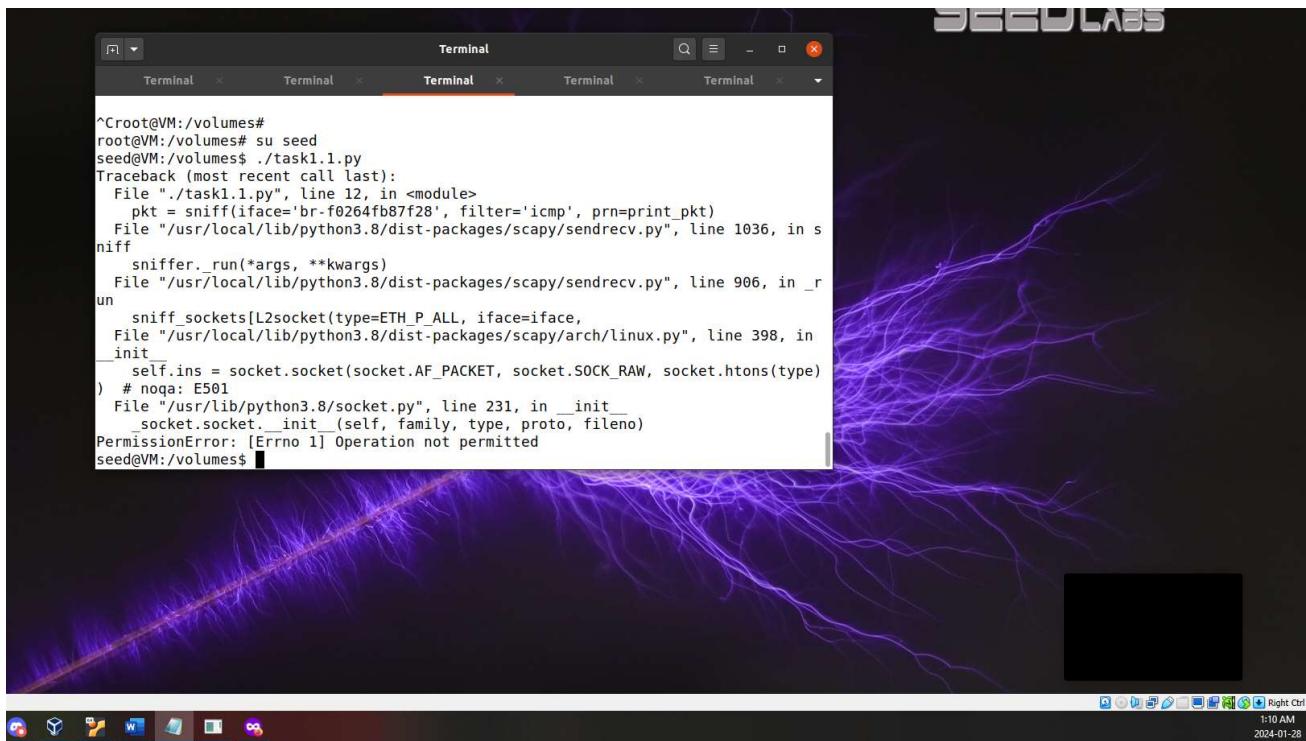
```

Task 1.1: Sniffing packets

Task 1.1 A:

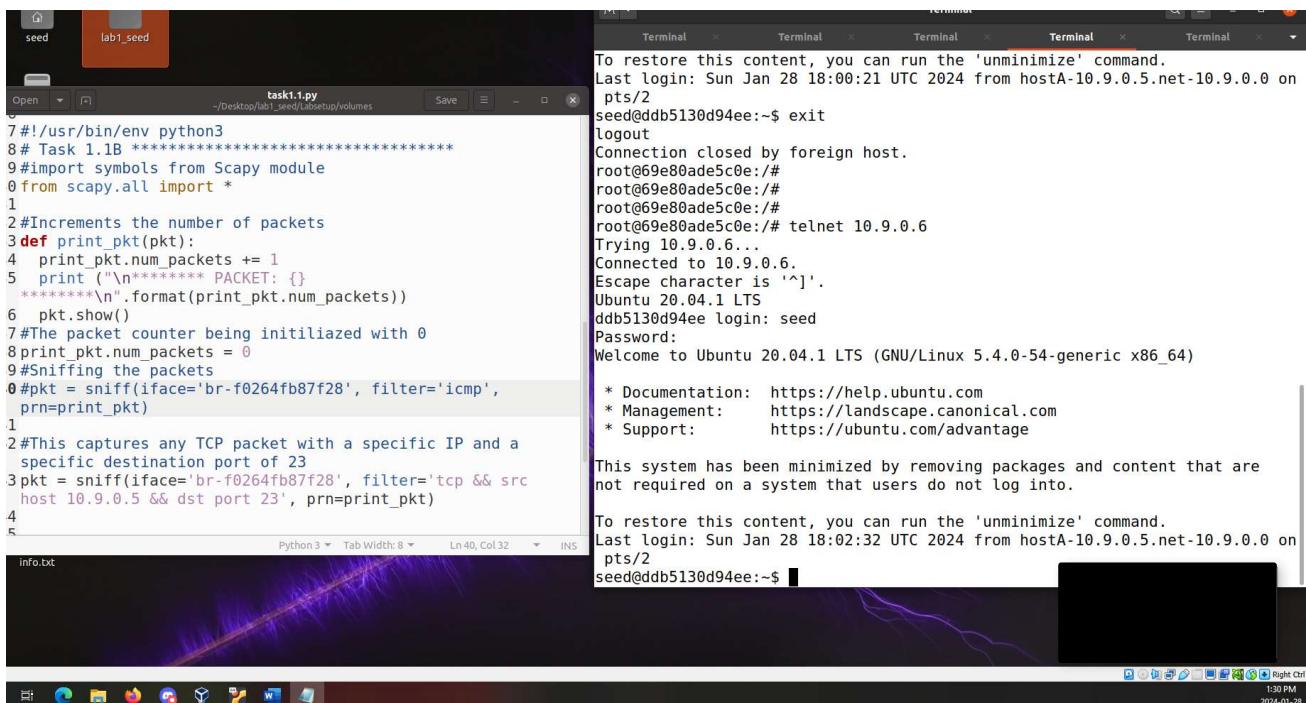
Here, I used the attacker container to sniff the ICMP packets going between Host A and B. Below you will see both the echo-request and the echo-reply associated with the communication as well as the program used.

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).



The screenshot shows a Linux desktop environment with a dark theme. In the top left, there are icons for 'seed' and 'lab1_seed'. A code editor window titled 'task1.1.py' is open, displaying Python code for packet capture. A terminal window titled 'root@VM:/volumes#' is running the script, showing detailed information about a captured Ethernet and TCP packet.

```

Open a file /r/bin/env python3
20 # Task 1.1B *****
21 #import symbols from Scapy module
22 from scapy.all import *
23
24 #Increments the number of packets
25 def print_pkt(pkt):
26     print_pkt.num_packets += 1
27     print ("***** PACKET: {}"
28           .format(print_pkt.num_packets))
29     pkt.show()
30 #The packet counter being initialized with 0
31 print_pkt.num_packets = 0
32 #Sniffing the packets
33 pkt = sniff(iface='br-f0264fb87f28', filter='icmp',
34             prn=print_pkt)
35
36 #This captures any TCP packet with a specific IP and a
37 #specific destination port of 23
38 pkt = sniff(iface='br-f0264fb87f28', filter='tcp && src
39 host 10.9.0.5 && dst port 23', prn=print_pkt)
40
41
42 #Capture packets comes from or to go to a particular
43 #subnet. You can pick any subnet, such as 128.230.0.0/16;
44 #you should not pick the subnet that your VM is attached to.
45 pkt = sniff(iface='br-f0264fb87f28', filter='net
46 128.230.0.0/16', prn=print_pkt)
47
48
49
50
51
52
53
54

```

```

root@VM:/volumes# ./task1.1.py
*****
###[ Ethernet ]###
dst      = 02:42:0a:09:00:06
src      = 02:42:0a:09:00:05
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x10
len      = 60
id       = 54096
flags    = DF
frag     = 0
ttl      = 64
proto    = tcp
chksum   = 0x533f
src      = 10.9.0.5
dst      = 10.9.0.6
\options 
###[ TCP ]###
sport    = 32876
dport    = telnet
seq      = 1197285672
ack      = 0
dataofs  = 10
reserved = 0

```

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 been done from host B).

The screenshot shows a Linux desktop environment with a dark theme. In the top right, there is a 'SEDU LABS' logo. A code editor window titled 'task1.1.py' is open, displaying Python code for packet capture, similar to the previous one but with a different subnet filter. A terminal window titled 'root@69e80ade5c0e:' is running the script, showing the output of a ping command to the specified subnet.

```

38 print_pkt.num_packets = 0
39 #Sniffing the packets
40 pkt = sniff(iface='br-f0264fb87f28', filter='icmp',
41             prn=print_pkt)
42
43 #This captures any TCP packet with a specific IP and a
44 #specific destination port of 23
45 pkt = sniff(iface='br-f0264fb87f28', filter='tcp && src
46 host 10.9.0.5 && dst port 23', prn=print_pkt)
47
48
49
50
51
52
53
54

```

```

root@69e80ade5c0e:/#
root@69e80ade5c0e:/#
root@69e80ade5c0e:/# ping 128.230.0.20
PING 128.230.0.20 (128.230.0.20) 56(84) bytes of data.
From 128.230.61.170 icmp_seq=1 Destination Host Unreachable
^C
--- 128.230.0.20 ping statistics ---
4 packets transmitted, 0 received, +1 errors, 100% packet loss, time 3052ms
root@69e80ade5c0e:/#

```

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.

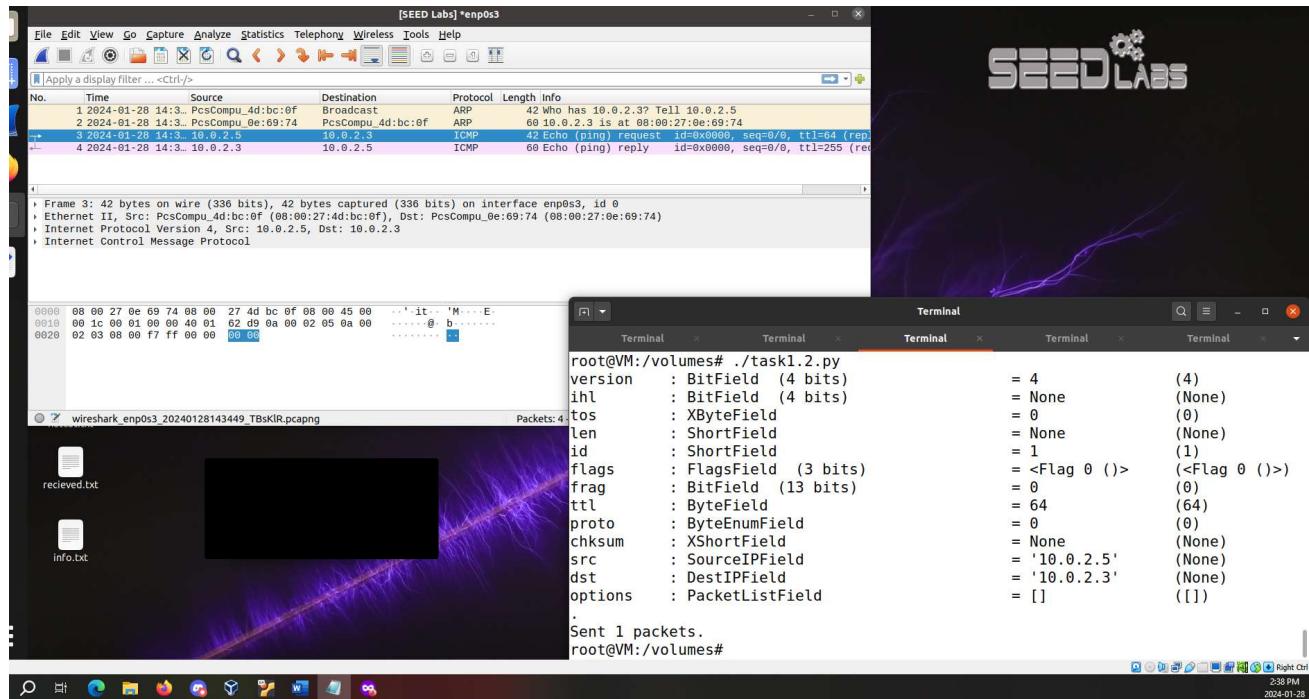
The screenshot shows a Kali Linux desktop environment with several open windows:

- A terminal window titled "Terminal" showing the output of running the Python script: "root@VM:/volumes# ./task1.2.py". The output details the fields and their values of the generated ICMP packet.
- A code editor window titled "task1.2.py" containing the Python scapy code for crafting an ICMP echo request.
- A file manager window showing files like "example-netcat.txt", "recieved.txt", and "info.txt".

The terminal output is as follows:

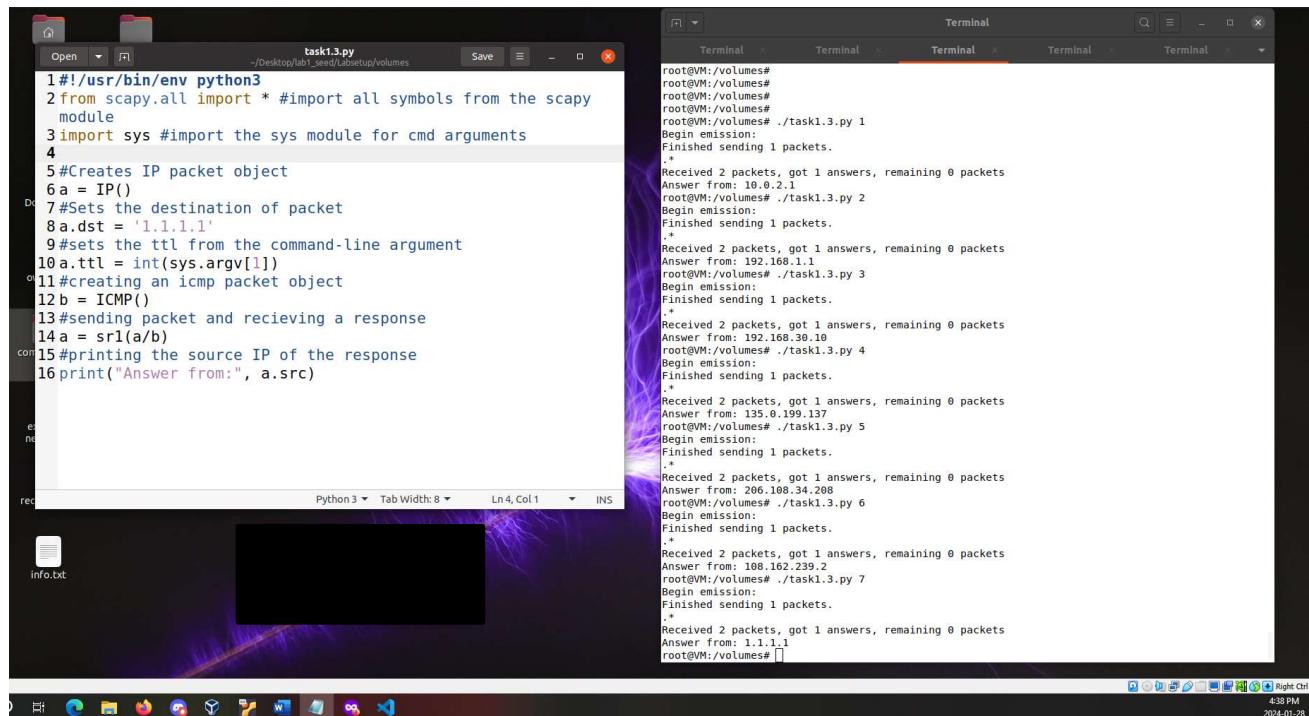
```
root@VM:/volumes# ./task1.2.py
version      : BitField (4 bits)          = 4          (4)
ihl         : BitField (4 bits)          = None       (None)
tos         : XByteField               = 0          (0)
len         : ShortField              = None       (None)
id          : ShortField              = 1          (1)
flags        : FlagsField (3 bits)        = <Flag 0 ()> (<Flag 0 ()>)
frag        : BitField (13 bits)         = 0          (0)
ttl          : ByteField                = 64         (64)
proto        : ByteEnumField           = 0          (0)
chksum       : XShortField             = None       (None)
src          : SourceIPField           = '10.0.2.5' (None)
dst          : DestIPField              = '10.0.2.3' (None)
options      : PacketListField          = []         ([])

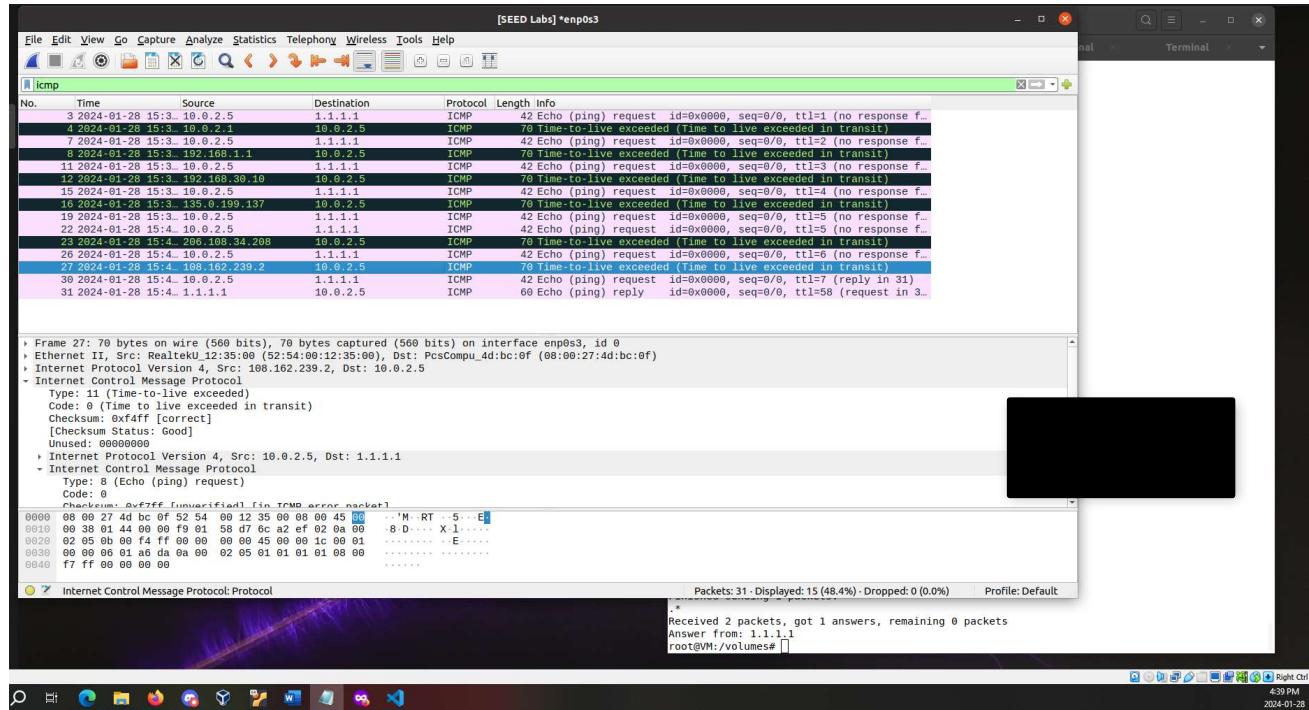
.
Sent 1 packets.
root@VM:/volumes#
```



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

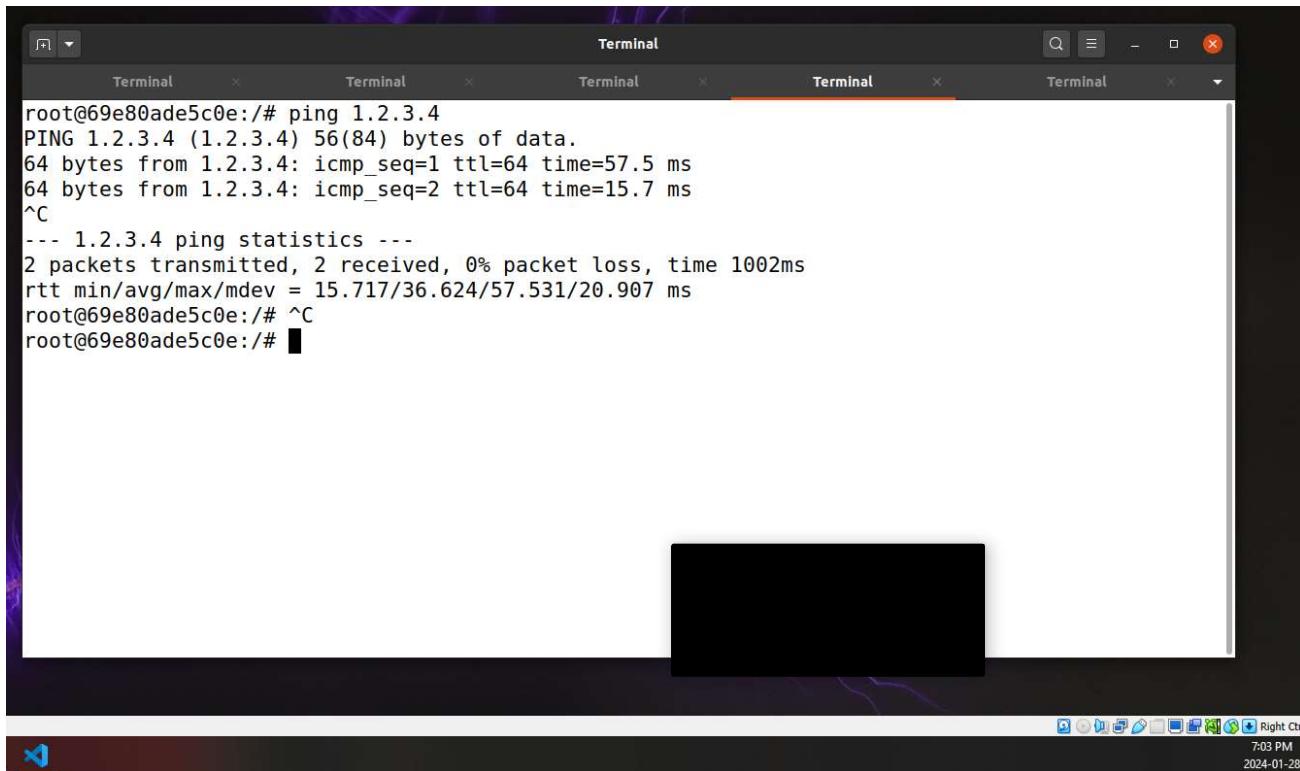
Here, I implemented the sniffing and spoofing program. The sniffing and spoofing was conducted on the attacker container while the ping was done on the host A container. For this example, if you look below, you will find that I have ran program specifying the IP 1.2.3.4 and successfully spoofed a reply.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

```

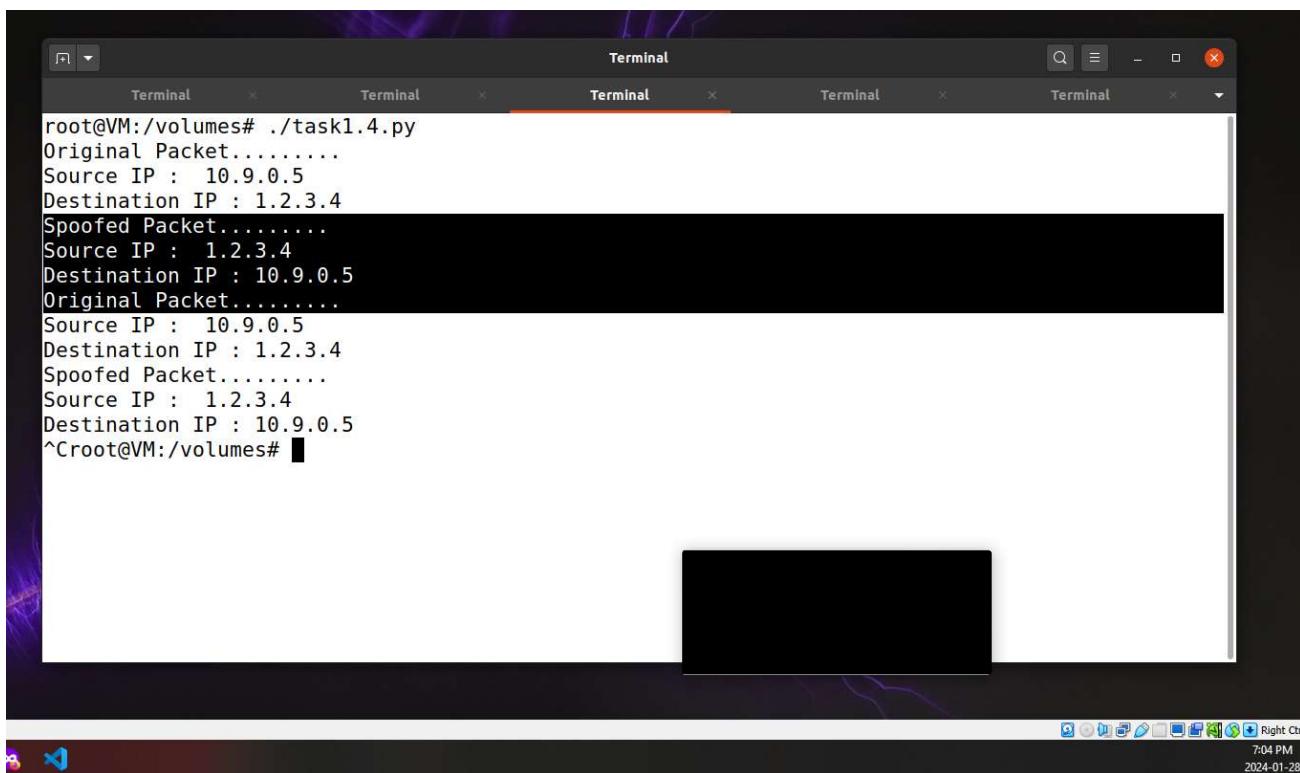
task1.4.py
~/Desktop/rishabh_Seed/lanSetup/volumes

1 #!/usr/bin/env python3
2 from scapy.all import * # importing symbols from scapy module
3
4 def spoof_pkt(pkt):
5     # sniff and print out icmp echo request packet
6     if ICMP in pkt and pkt[ICMP].type == 8:
7         print("Original Packet.....")
8         print("Source IP : ", pkt[IP].src)
9         print("Destination IP : ", pkt[IP].dst)
10
11        # spoof an icmp echo reply packet
12        # swap srcip and dstip
13        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
14        icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
15        data = pkt[Raw].load
16        newpkt = ip/icmp/data
17
18        print("Spoofed Packet.....")
19        print("Source IP : ", newpkt[IP].src)
20        print("Destination IP : ", newpkt[IP].dst)
21
22        send(newpkt, verbose=0) #ends the spoofed packet with the verbose mode disabled
23
24# Sniffing on specific interface and filtering icmp packet with specific host
25 pkt = sniff(iface = 'br-f0264fb87f28', filter = 'icmp and host 1.2.3.4', prn=spoof_pkt)
26

```

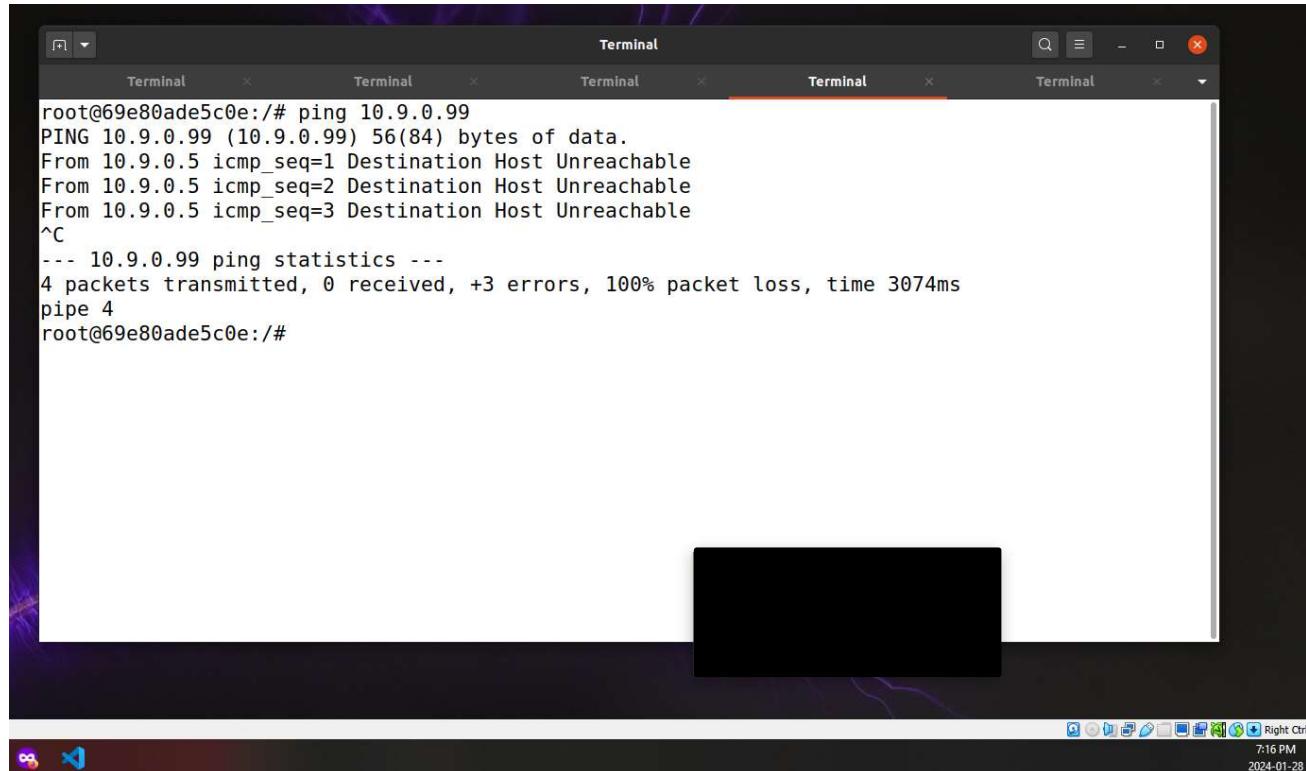


```
root@69e80ade5c0e:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=57.5 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=15.7 ms
^C
--- 1.2.3.4 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 15.717/36.624/57.531/20.907 ms
root@69e80ade5c0e:/# ^C
root@69e80ade5c0e:/#
```

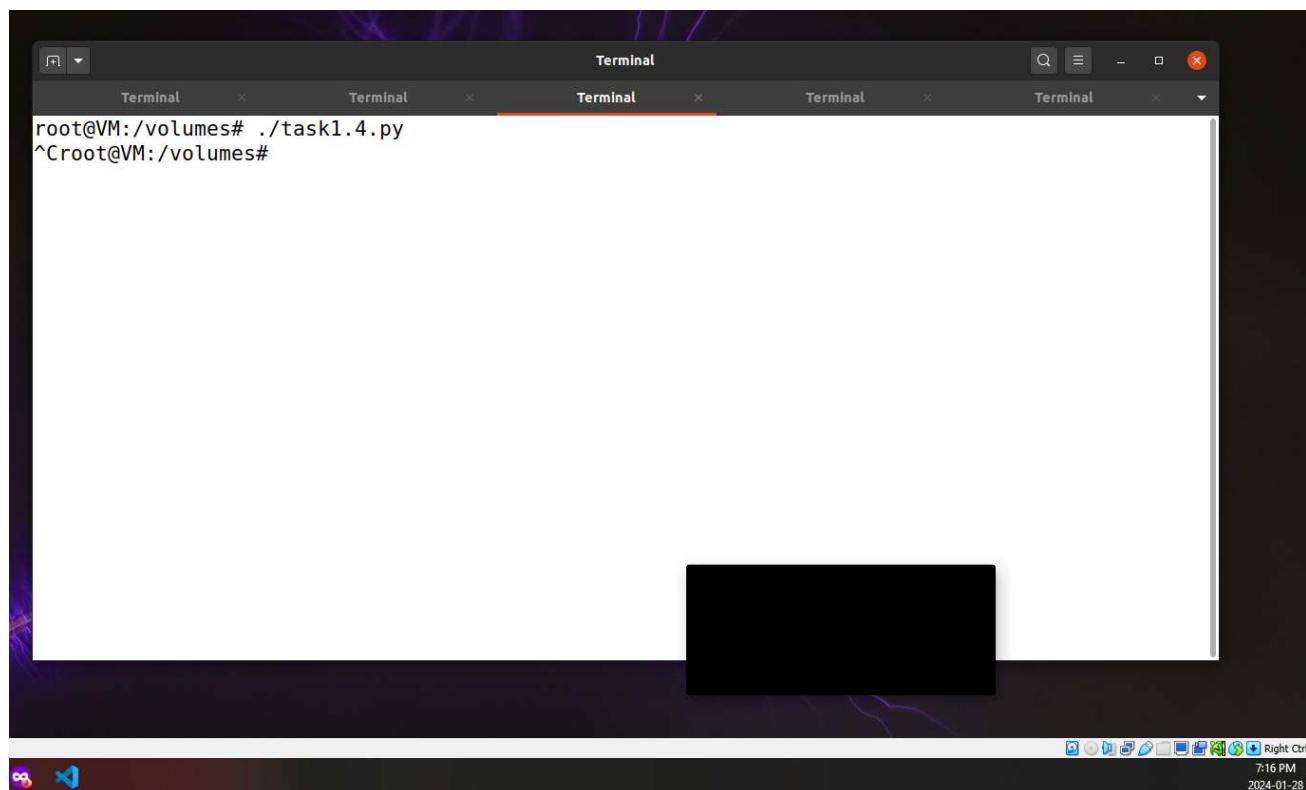


```
root@VM:/volumes# ./task1.4.py
Original Packet.....
Source IP : 10.9.0.5
Destination IP : 1.2.3.4
Spoofed Packet.....
Source IP : 1.2.3.4
Destination IP : 10.9.0.5
Original Packet.....
Source IP : 10.9.0.5
Destination IP : 1.2.3.4
Spoofed Packet.....
Source IP : 1.2.3.4
Destination IP : 10.9.0.5
^Croot@VM:/volumes#
```

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.

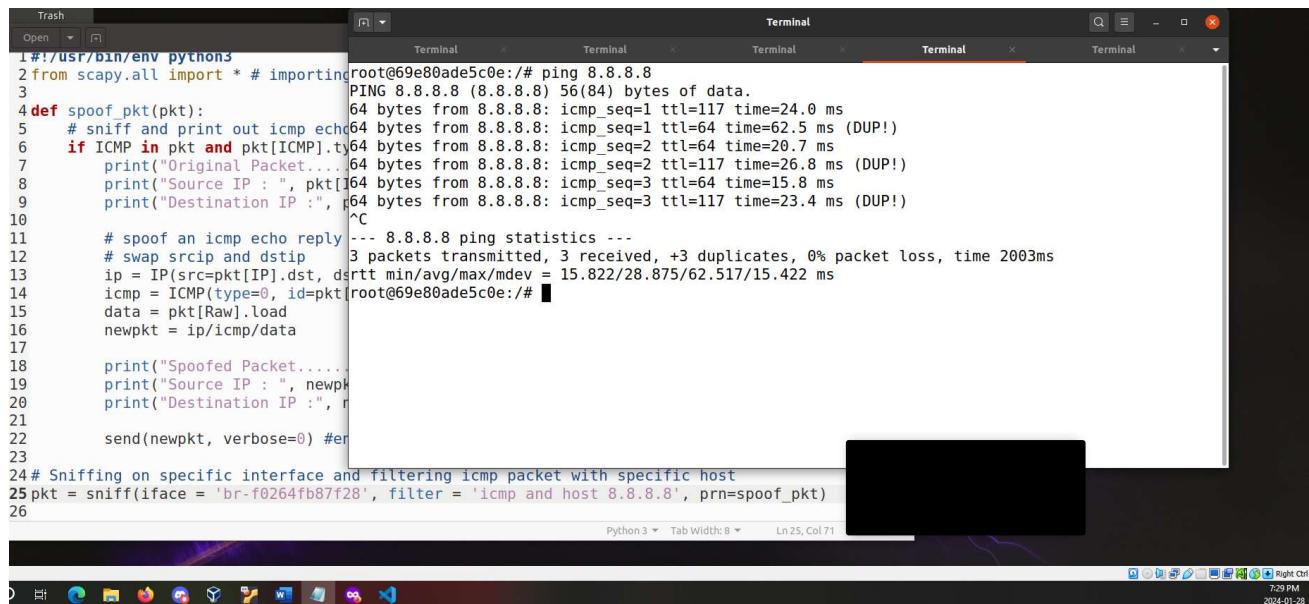


```
root@69e80ade5c0e:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.5 icmp_seq=1 Destination Host Unreachable
From 10.9.0.5 icmp_seq=2 Destination Host Unreachable
From 10.9.0.5 icmp_seq=3 Destination Host Unreachable
^C
--- 10.9.0.99 ping statistics ---
4 packets transmitted, 0 received, +3 errors, 100% packet loss, time 3074ms
pipe 4
root@69e80ade5c0e:/#
```



```
root@VM:/volumes# ./task1.4.py
^Croot@VM:/volumes#
```

Here, I used the same program but this time to ping an existing host on the internet (8.8.8.8). Because this is 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.

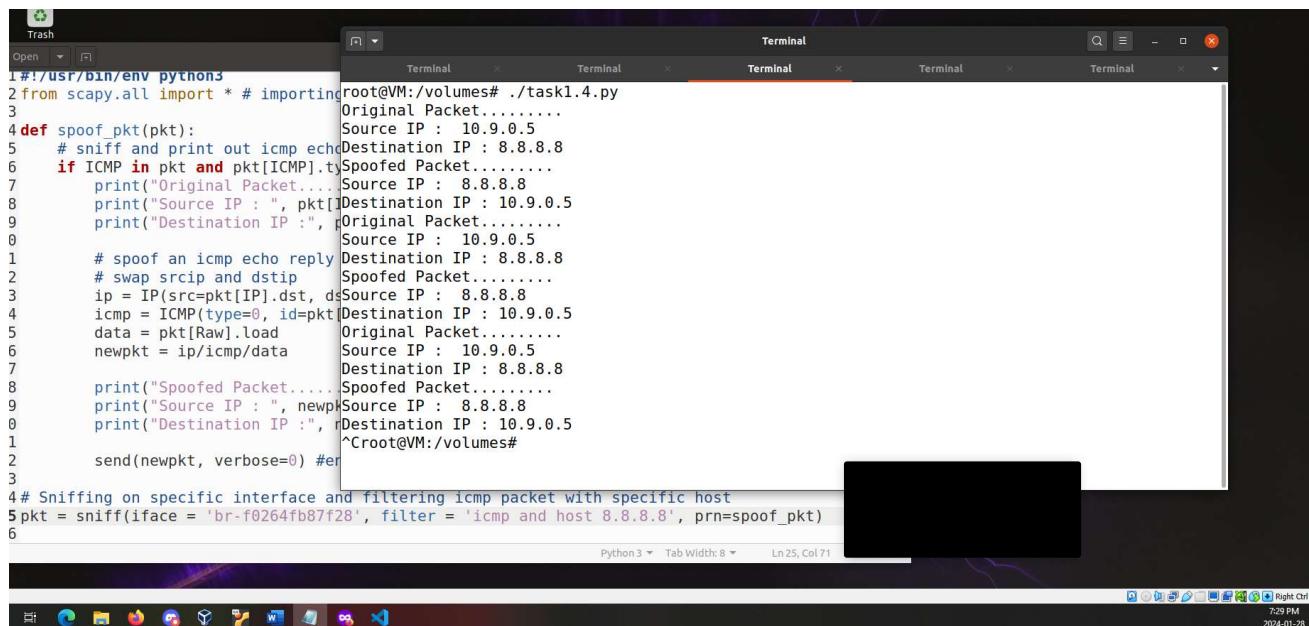


```

1#!/usr/bin/env python3
2from scapy.all import * # importing
3
4def spoof_pkt(pkt):
5    # sniff and print out icmp echo
6    if ICMP in pkt and pkt[ICMP].type == 8:
7        print("Original Packet....")
8        print("Source IP : ", pkt[IP].src)
9        print("Destination IP : ", pkt[IP].dst)
10
11    # spoof an icmp echo reply --- 8.8.8.8 ping statistics ---
12    # swap srcip and dstip
13    ip = IP(src=pkt[IP].dst, dst=pkt[IP].src)
14    icmp = ICMP(type=0, id=pkt[ICMP].id)
15    data = pkt[Raw].load
16    newpkt = ip/icmp/data
17
18    print("Spoofed Packet.....")
19    print("Source IP : ", newpkt[IP].src)
20    print("Destination IP : ", newpkt[IP].dst)
21
22    send(newpkt, verbose=0) #er
23
24# Sniffing on specific interface and filtering icmp packet with specific host
25pkt = sniff(iface = 'br-f0264fb87f28', filter = 'icmp and host 8.8.8.8', prn=spoof_pkt)
26

```

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
3 packets transmitted, 3 received, +3 duplicates, 0% packet loss, time 2003ms
ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, id=pkt[ICMP].id)
icmp = ICMP(type=0, id=pkt[ICMP].id)
data = pkt[Raw].load
newpkt = ip/icmp/data
print("Spoofed Packet.....")
print("Source IP : ", newpkt[IP].src)
print("Destination IP : ", newpkt[IP].dst)
send(newpkt, verbose=0) #er
root@69e80ade5c0e:/#



```

1#!/usr/bin/env python3
2from scapy.all import * # importing
3
4def spoof_pkt(pkt):
5    # sniff and print out icmp echo
6    if ICMP in pkt and pkt[ICMP].type == 8:
7        print("Original Packet....")
8        print("Source IP : 10.9.0.5")
9        print("Destination IP : 8.8.8.8")
10
11    # spoof an icmp echo reply
12    # swap srcip and dstip
13    ip = IP(src=pkt[IP].dst, dst=pkt[IP].src)
14    icmp = ICMP(type=0, id=pkt[ICMP].id)
15    data = pkt[Raw].load
16    newpkt = ip/icmp/data
17
18    print("Spoofed Packet.....")
19    print("Source IP : ", newpkt[IP].src)
20    print("Destination IP : 10.9.0.5")
21
22    send(newpkt, verbose=0) #er
23
24# Sniffing on specific interface and filtering icmp packet with specific host
25pkt = sniff(iface = 'br-f0264fb87f28', filter = 'icmp and host 8.8.8.8', prn=spoof_pkt)
26

```

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#

References

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