Sniffing and Spoofing Lab

Submitted by: Mahika Gupta

SRN: PES1UG20CS243

Table of Contents:

Lab Environment Setup

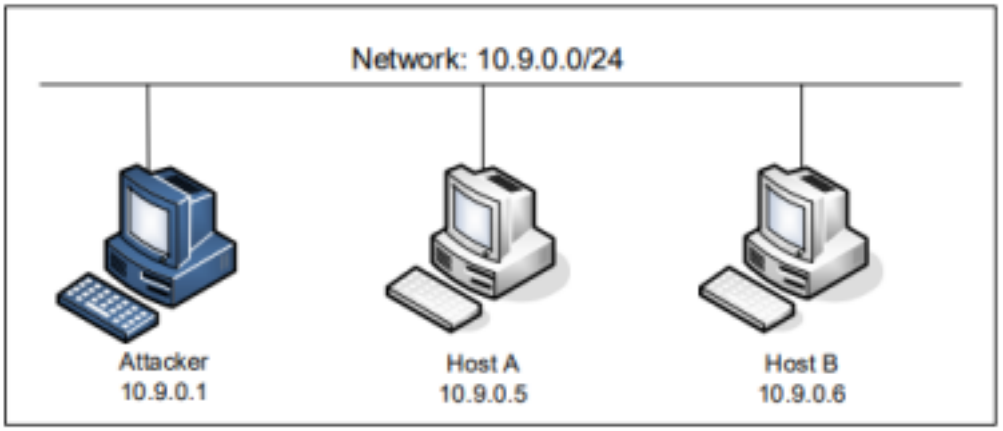
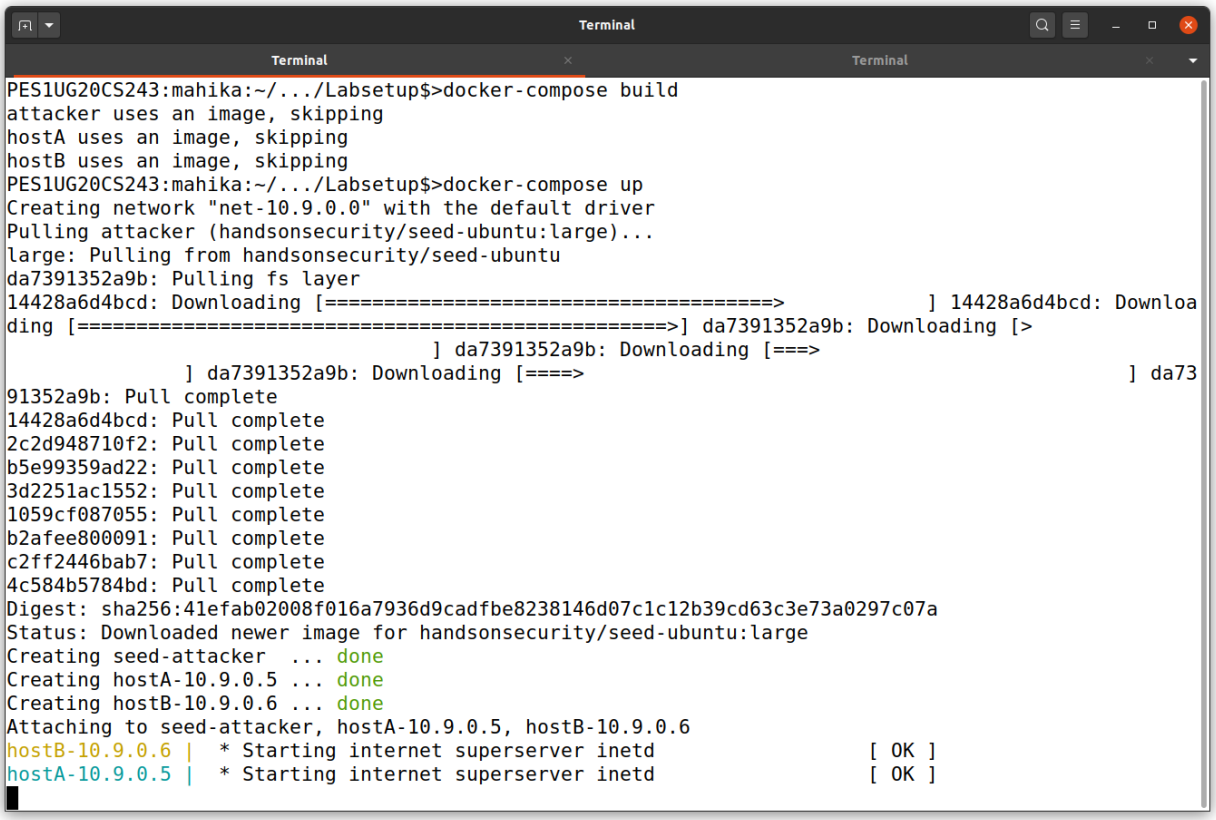
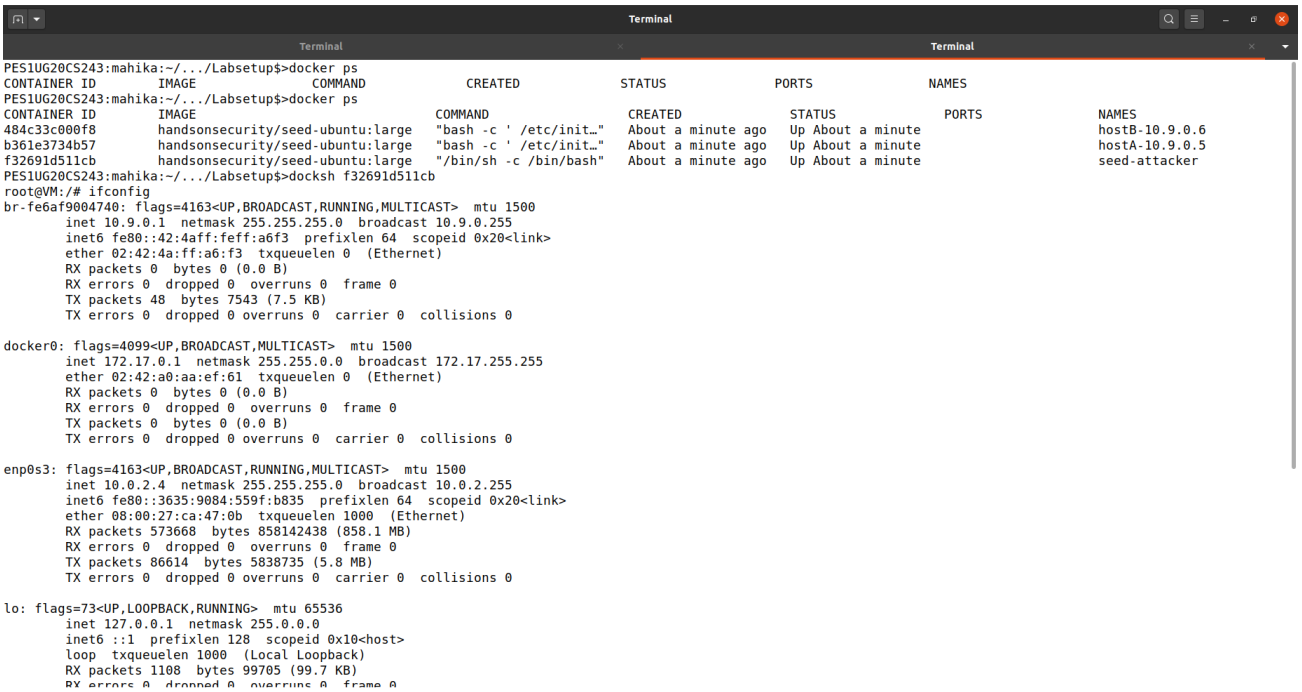


Figure 1 : Lab environment setup


Terminal
Terminal
Terminal
PES1UG20CS243:mahika:~/.../Labsetup$>docker-compose build
attacker uses an image, skipping
hostA uses an image, skipping
hostB uses an image, skipping
PES1UG20CS243:mahika:~/.../Labsetup$>docker-compose up
Creating network "net-10.9.0.0" with the default driver
Pulling attacker (handsonsecurity/seed-ubuntu:large)...
large: Pulling from handsonsecurity/seed-ubuntu
da7391352a9b: Pulling fs layer
14428a6d4bcd: Downloading [==========>] 14428a6d4bcd: Downloading [==========>] da7391352a9b: Downloading [>
] da7391352a9b: Downloading [===>
] da7391352a9b: Downloading [====>] da7391352a9b: Pull complete
14428a6d4bcd: Pull complete
2c2d948710f2: Pull complete
b5e99359ad22: Pull complete
3d2251ac1552: Pull complete
1059cf087055: Pull complete
b2afee800091: Pull complete
c2ff2446bab7: Pull complete
4c584b5784bd: Pull complete
Digest: sha256:41efab02008f016a7936d9cadfbe8238146d07c1c12b39cd63c3e73a0297c07a
Status: Downloaded newer image for handsonsecurity/seed-ubuntu:large
Creating seed-attacker ... done
Creating hostA-10.9.0.5 ... done
Creating hostB-10.9.0.6 ... done
Attaching to seed-attacker, hostA-10.9.0.5, hostB-10.9.0.6
hostB-10.9.0.6 | * Starting internet superserver inetd [OK]
hostA-10.9.0.5 | * Starting internet superserver inetd [OK]


Terminal
Terminal
Terminal
PES1UG20CS243:mahika:~/.../Labsetup$>docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
PES1UG20CS243:mahika:~/.../Labsetup$>docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
484c33c000f8 handsonsecurity/seed-ubuntu:large "bash -c ' /etc/init…" About a minute ago Up About a minute hostB-10.9.0.6
b361e3734b57 handsonsecurity/seed-ubuntu:large "bash -c ' /etc/init…" About a minute ago Up About a minute hostA-10.9.0.5
f32691d511cb handsonsecurity/seed-ubuntu:large "/bin/sh -c /bin/bash" About a minute ago Up About a minute seed-attacker
PES1UG20CS243:mahika:~/.../Labsetup$>docksh f32691d511cb
root@VM:/# ifconfig
br-fe6af9004740: 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::42:4aff:feff:a6f3 prefixlen 64 scopeid 0x20<link>
ether 02:42:4a:ff:a6:f3 txqueuelen 0 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 48 bytes 7543 (7.5 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:a0:aa:ef:61 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
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 10.0.2.4 netmask 255.255.255.0 broadcast 10.0.2.255
inet6 fe80::3635:9084:559f:b835 prefixlen 64 scopeid 0x20<link>
ether 08:00:27:ca:47:0b txqueuelen 1000 (Ethernet)
RX packets 573668 bytes 858142438 (858.1 MB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 86614 bytes 5838735 (5.8 MB)
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
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 1108 bytes 99705 (99.7 KB)

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

Task 1.1 : Sniffing Packets

The objective of this task is to learn how to use Scapy to do packet sniffing in Python programs.

Task 1.1 A : Sniff IP packets using Scapy

The program ,for each captured packet, the callback function print pkt() will be invoked; this function will print out some of the information about the packet. Run the program with the root privilege and demonstrate that you can indeed capture packets. After that, run the program again, but without using the root privilege; describe and explain your observations.

NOTE

Check the **Lab setup instructions document** for detailed instructions on how to find out the interface of your attacker machine. **Replace the interface in the code wherever required**.

On the Attacker terminal run the command:

 # python3 Task1.1A.py

Explain on which VM you ran this command and why? Provide a screenshot of your observations.

```
Terminal
Terminal                                                    Terminal
seed-attacker:PES1UG20CS243:Mahika:/volumes
$>python3 Task1.1A.py
SNIFFING PACKETS...
###[ Ethernet ]###
  dst       = ff:ff:ff:ff:ff:ff
  src       = 02:42:0a:09:00:05
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = 6
     plen      = 4
     op        = who-has
     hwsrc     = 02:42:0a:09:00:05
     psrc      = 10.9.0.5
     hwdst     = 00:00:00:00:00:00
     pdst      = 10.9.0.1

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:4a:ff:a6:f3
  type      = ARP
###[ ARP ]###
     hwtype    = 0x1
     ptype     = IPv4
     hwlen     = 6
     plen      = 4
     op        = is-at
     hwsrc     = 02:42:4a:ff:a6:f3
     psrc      = 10.9.0.1
     hwdst     = 02:42:0a:09:00:05
     pdst      = 10.9.0.5

###[ Ethernet ]###
```

ANS: This command was run on the seed-attacker VM as this is a program to sniff packets, and the attacker machine is sniffing the packets of the host machine with IP address 10.9.0.1.

From the **host A** machine's terminal ping a random IP address(8.8.8.8) **On the Host A terminal run the command:**

 # ping 8.8.8.8

```
Terminal
Terminal                                                    Terminal
PES1UG20CS243:mahika:~$>docker ps
CONTAINER ID        IMAGE                                   COMMAND
 CREATED             STATUS              PORTS               NAMES
484c33c000f8        handsonsecurity/seed-ubuntu:large   "bash -c ' /etc/init…"
 About an hour ago   Up About an hour                        hostB-10.9.0.6
b361e3734b57        handsonsecurity/seed-ubuntu:large   "bash -c ' /etc/init…"
 About an hour ago   Up About an hour                        hostA-10.9.0.5
f32691d511cb        handsonsecurity/seed-ubuntu:large   "/bin/sh -c /bin/bash"
 About an hour ago   Up About an hour                        seed-attacker
PES1UG20CS243:mahika:~$>docksh b361e3734b57
root@b361e3734b57:/# 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=58 time=13.5 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=58 time=13.8 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=58 time=12.9 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=58 time=12.9 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=58 time=12.9 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=58 time=13.0 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=58 time=12.7 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=58 time=13.1 ms
```

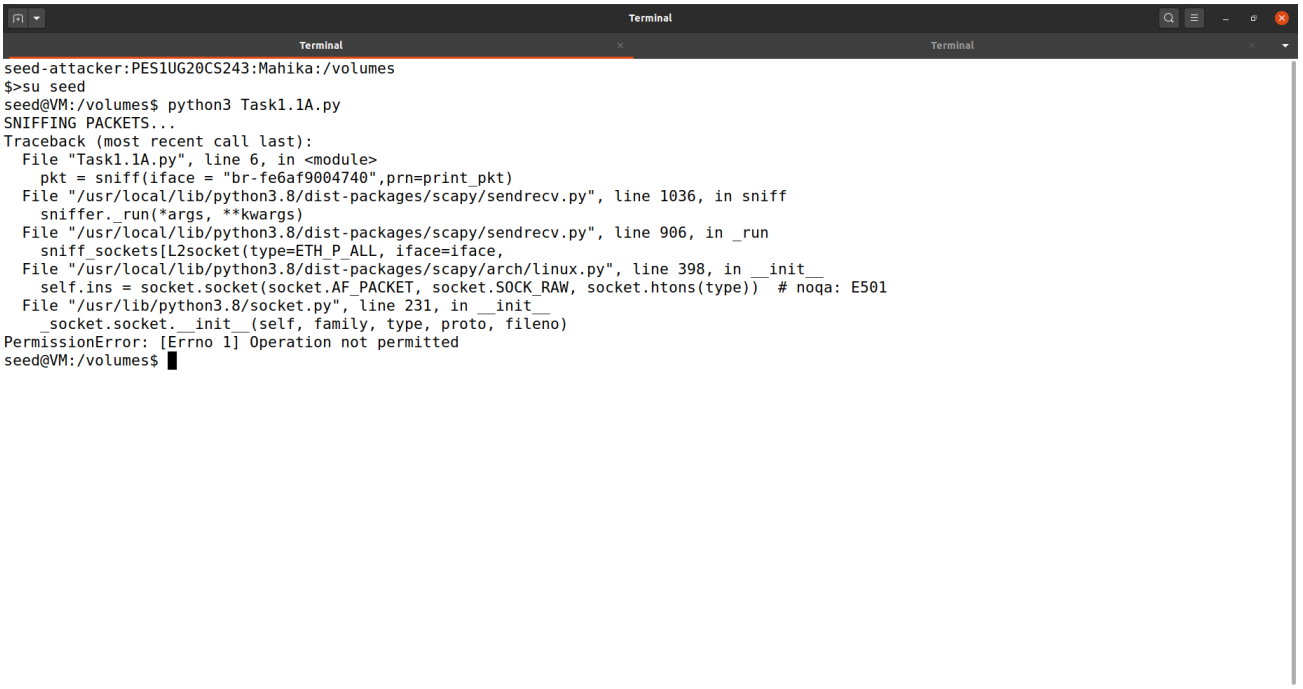
Now, we run the same program without root privileges. Do you find any issues? If so, why? Provide a screenshot of your observations.'

On the Attacker terminal run the command:

 # su seed

$ python3 Task1.1A.py

Running with root privileges:



On running the program with root privileges, the code shows an error saying the operation is not permitted. This is because in SEED the root user does not the privileges to sniff packets.

Task 1.1 B : Capturing ICMP, TCP packet and Subnet

Usually, when we sniff packets, we are only interested in certain types of packets. We can do

that by setting filters in sniffing. Scapy's filter uses the BPF (Berkeley Packet Filter) syntax; you can find the BPF manual from the Internet. Please set the following filters and demonstrate your sniffer program again (each filter should be set separately):

- Capture only ICMP packets.

- Capture any TCP packet that comes from a particular IP and with a destination port number 23.
- Capture packets comes from or to go to a particular subnet. You can pick any subnet, such as 128.230.0.0/16; you should not pick the subnet that your VM is attached to.

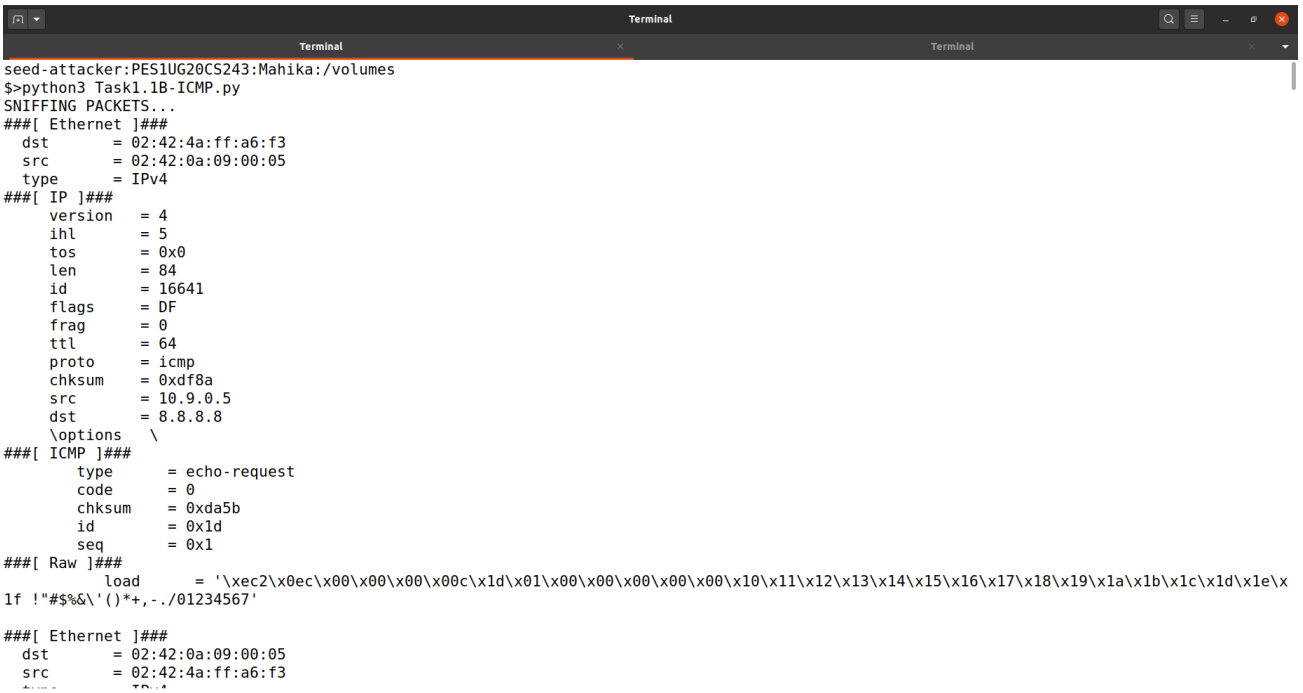
Capture only the ICMP packet

The ICMP packets are captured by the Scapy sniffer program. Hence, when some machine on the same network sends ping requests, the packets get captured by the sniffer.

Fill in the interface of the attacker machine in the given program and run the code. On the Attacker terminal run the command:

 # python3 Task1.1B-ICMP.py

Provide a screenshot of your observations



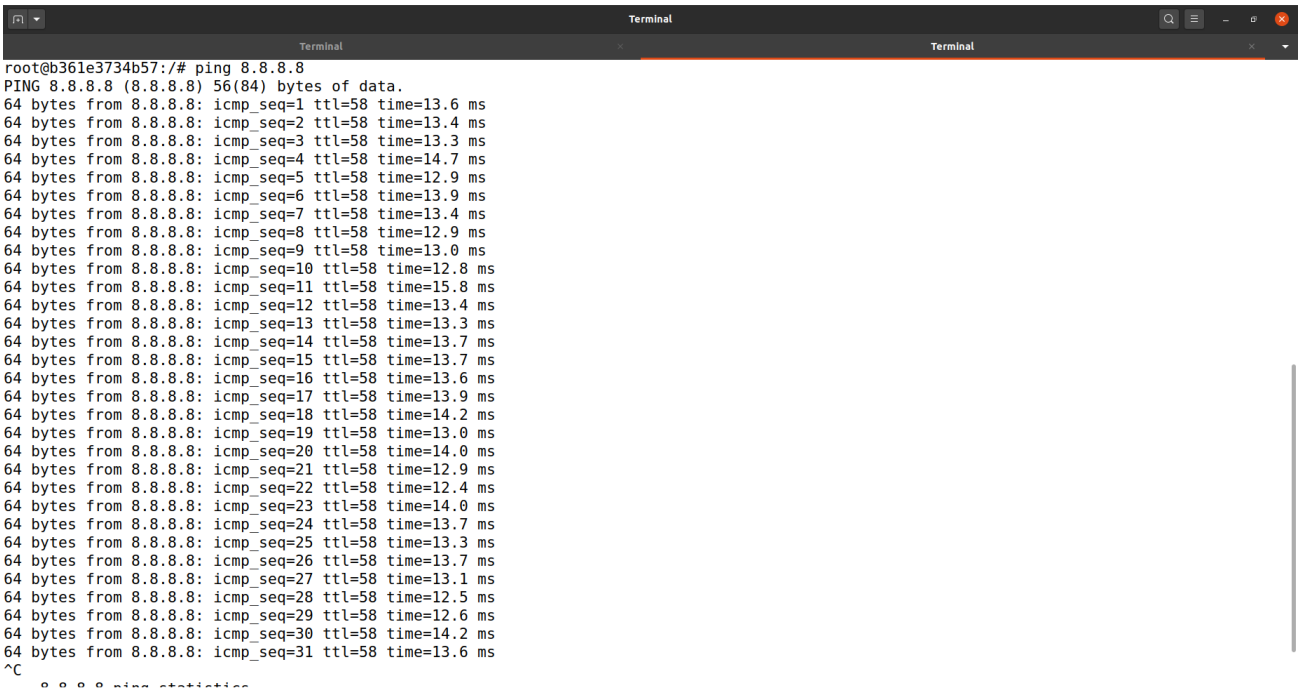
Here we observe that an ICMP echo request (type 0) sent by host A to 8.8.8.8, is sniffed by the attacker.

From the **host A** machine's terminal ping a random IP

address(8.8.8.8) **On the Host A terminal run the command:**

 # ping 8.8.8.8

The ICMP packets are captured by the sniffer program. Provide a screenshot of your observations.



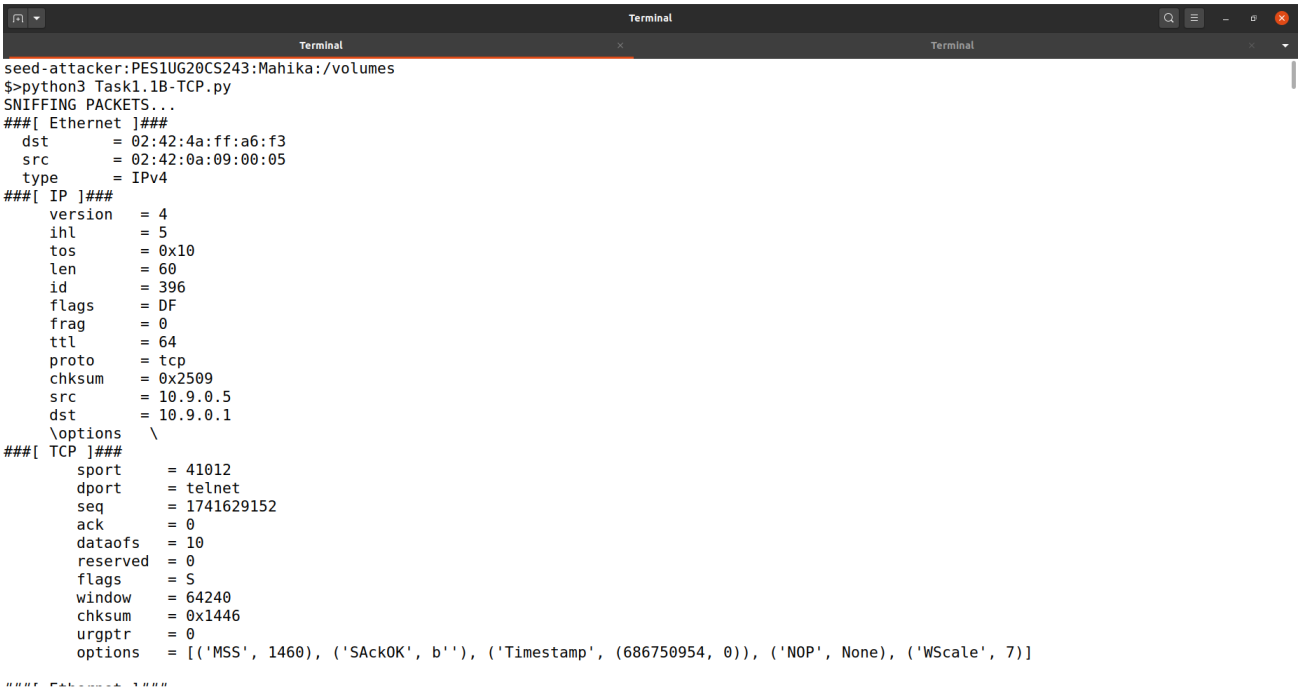
Capture any TCP packet that comes from a particular IP and with a destination port number 23

The program must capture the TCP packets being sent from the specified IP address on the port 23.
Fill in the interface of the attacker machine in the given program and run the code.

On the Attacker terminal run the command:

 # python3 Task1.1B-TCP.py

Provide a screenshot of your observations



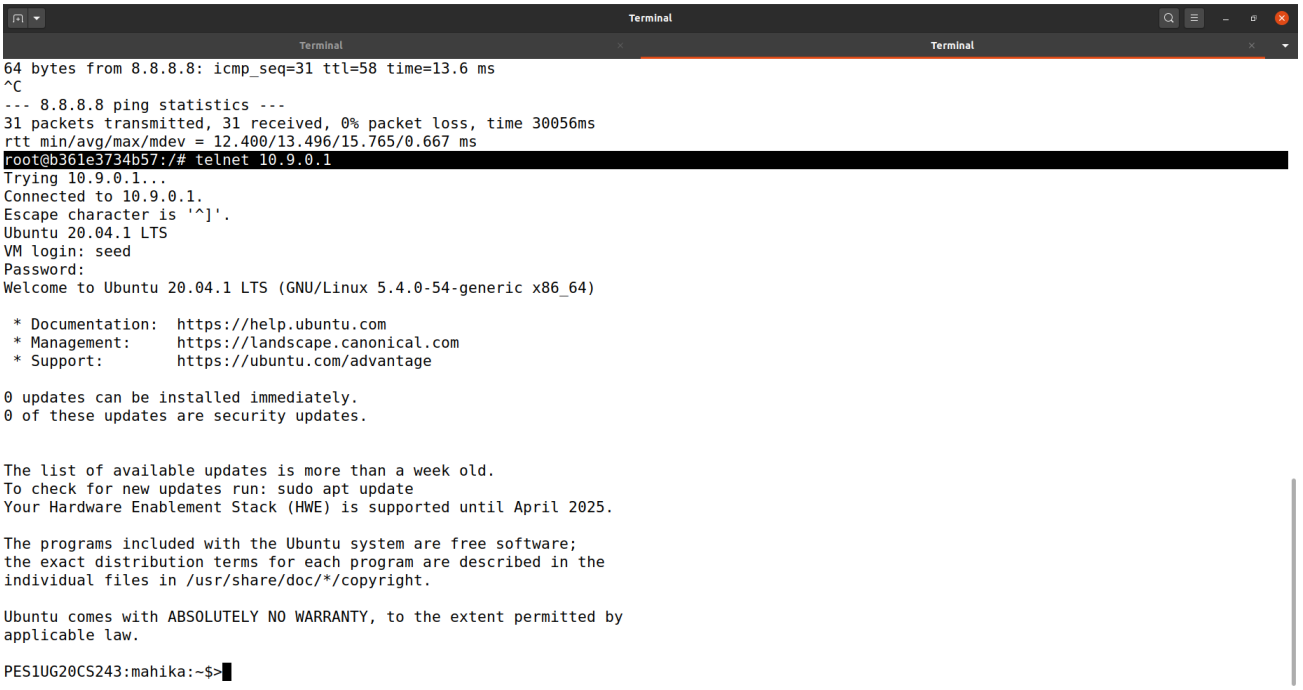
Telnet has port number 23.

From the **host A** machine's terminal telnet to a random IP address.

On the Host A terminal run the command::

 # telnet 10.9.0.1

Provide screenshots of your observations.



We use telnet command to ping as Telnet has port number 23.

Capture packets that come from or go to a particular subnet

You can pick any subnet, such as 192.168.254.0/24; you should not pick the subnet that your VM is attached to.

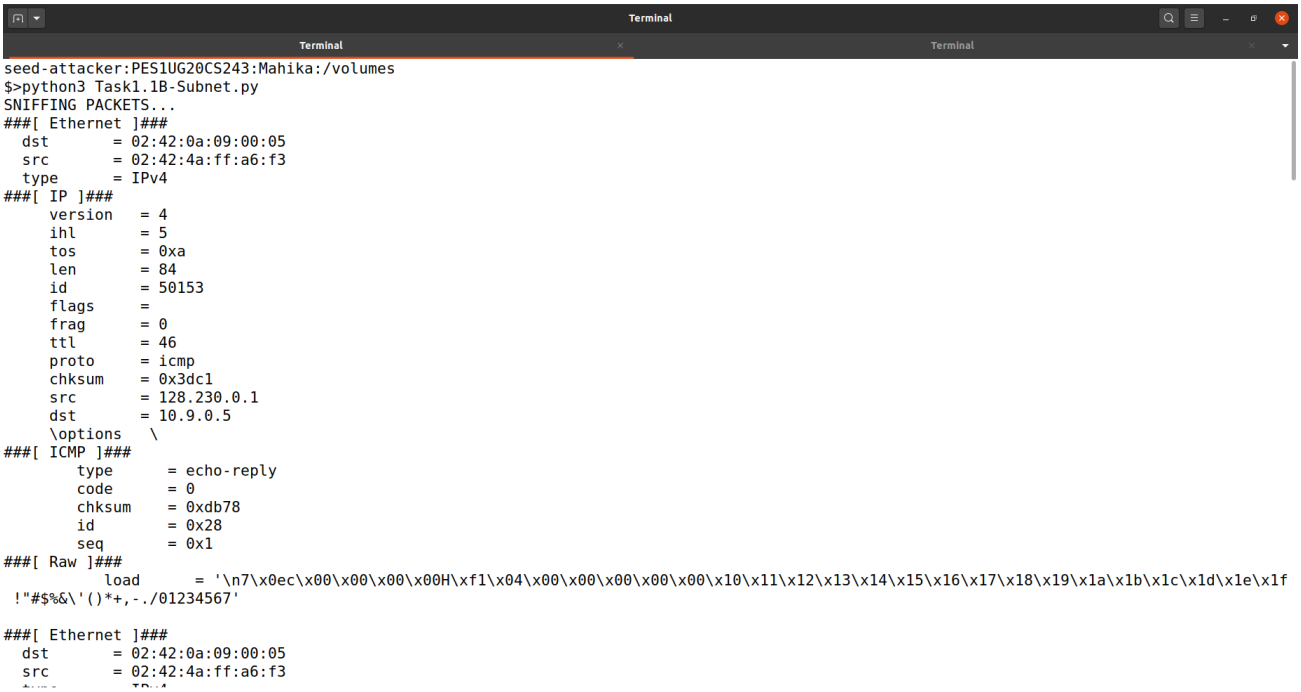
Show that on sending ICMP packets to 192.168.254.1, the sniffer program captures the packets sent out from 192.168.254.1 .

Fill in the interface of the attacker machine in the given program and run the code.

On the Attacker terminal run the command:

 # python3 Task1.1B-Subnet.py

Provide a screenshot of your observations



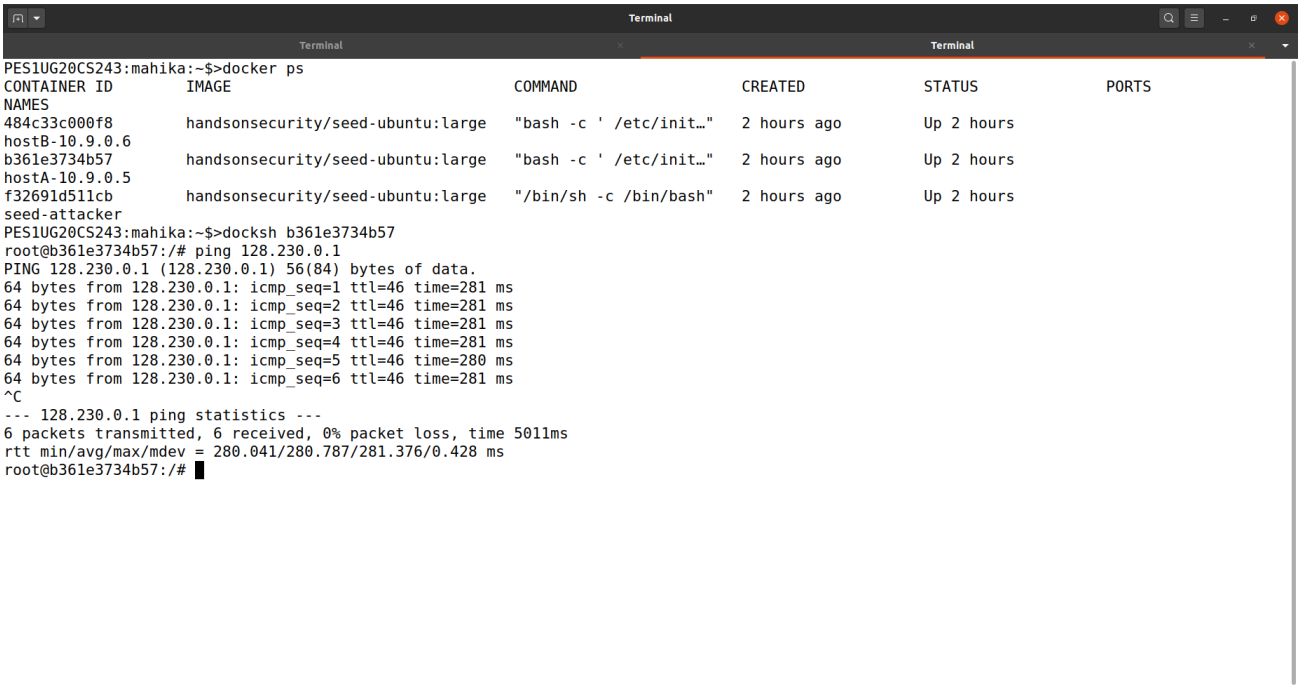
Here the subnet is taken as 128.230.0.0/16

From the **host A** machine's terminal, ping a random IP address on the chosen

subnet. **On the Host A terminal run the command::**

 # ping 192.168.254.1

Provide screenshots of your observations.



The IP address 128.230.0.1 is pinged, and all the packets sent/received from that IP address are sniffed by the attacker.

Task 1.2 : Spoofing

The objective of this task is to spoof IP packets with an arbitrary source IP address. We will spoof **ICMP echo request packets** and send them to another VM on the same network. We will use Wireshark to observe whether our request will be accepted by the receiver. If it is accepted, an echo reply packet will be sent to the spoofed IP address. Below shows the code to create the ICMP packet. The spoofed request is formed by creating our own packet with the header specifications.

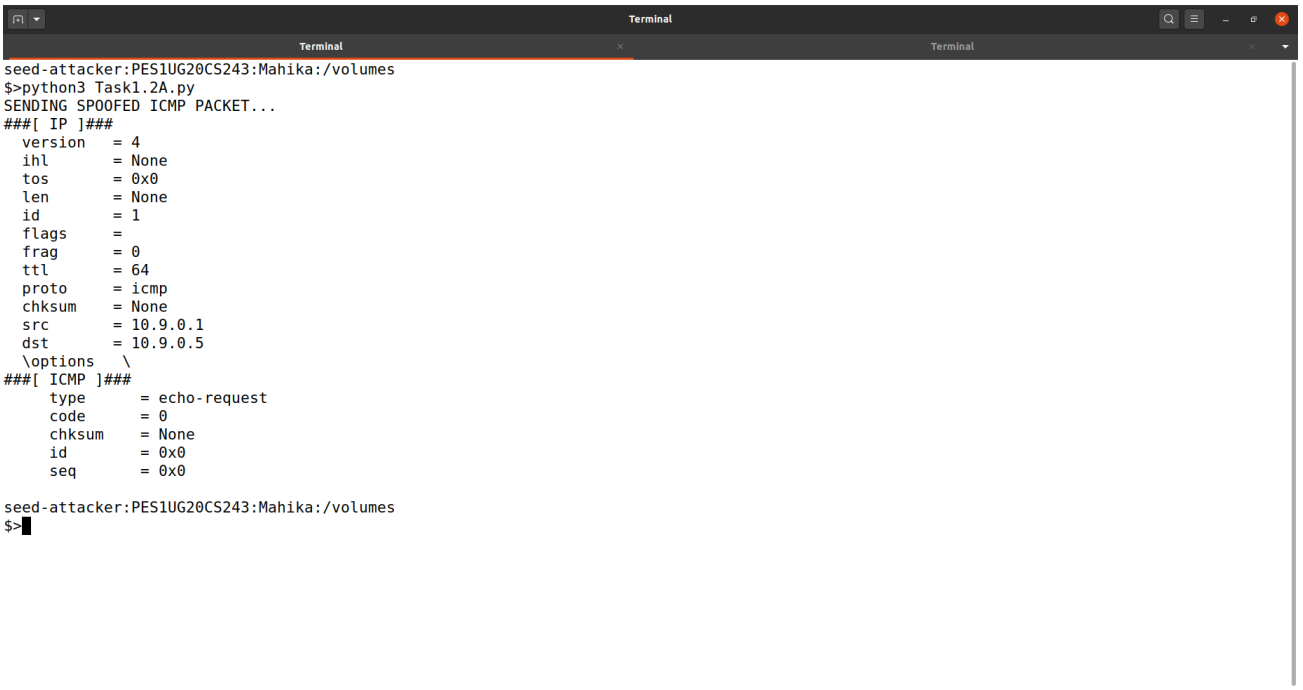
Similarly, we fill the IP header with source IP address of any machine within the local network and destination IP address of any remote machine on the internet which is alive.

Please keep wireshire open before you execute the program. Show that Wireshark captures the live machine sending back an ICMP response.

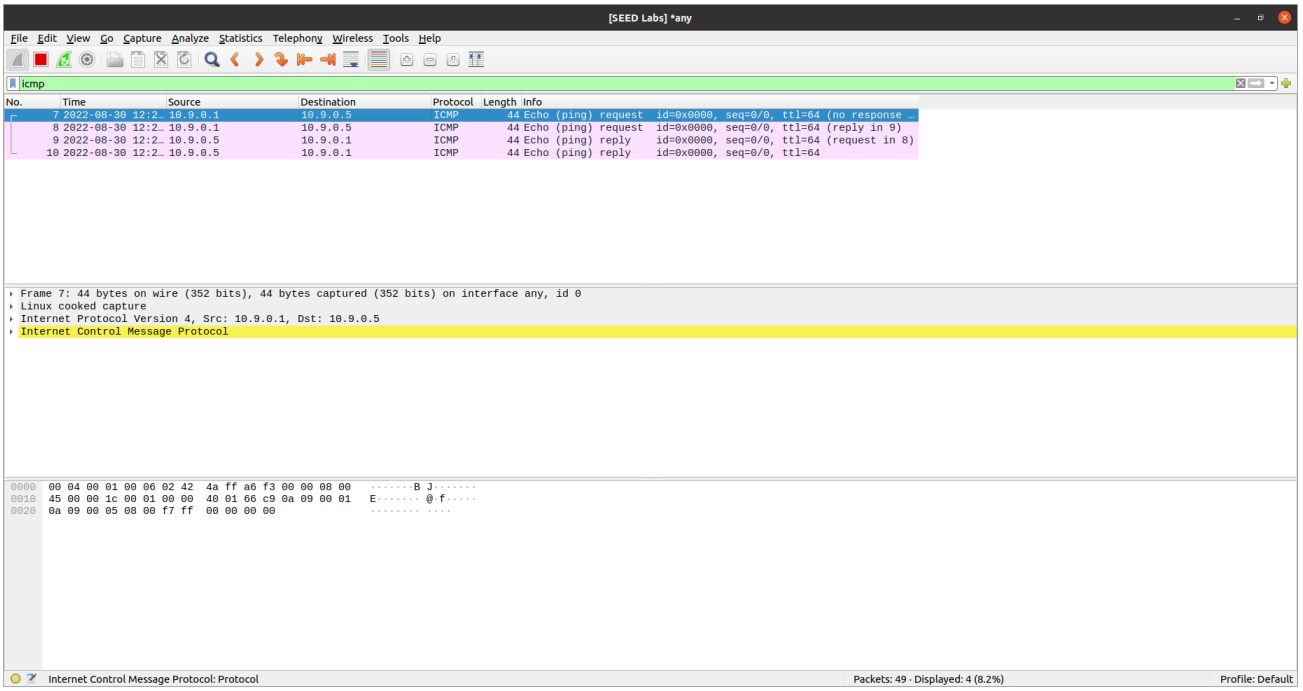
On the Attacker terminal run the command:

 # python Task1.2A.py

Provide a screenshot of your observations.



Demonstrate that you can spoof an ICMP echo request packet with an **arbitrary source IP address**. Open Wireshark and observe the ICMP packets as they are being captured.

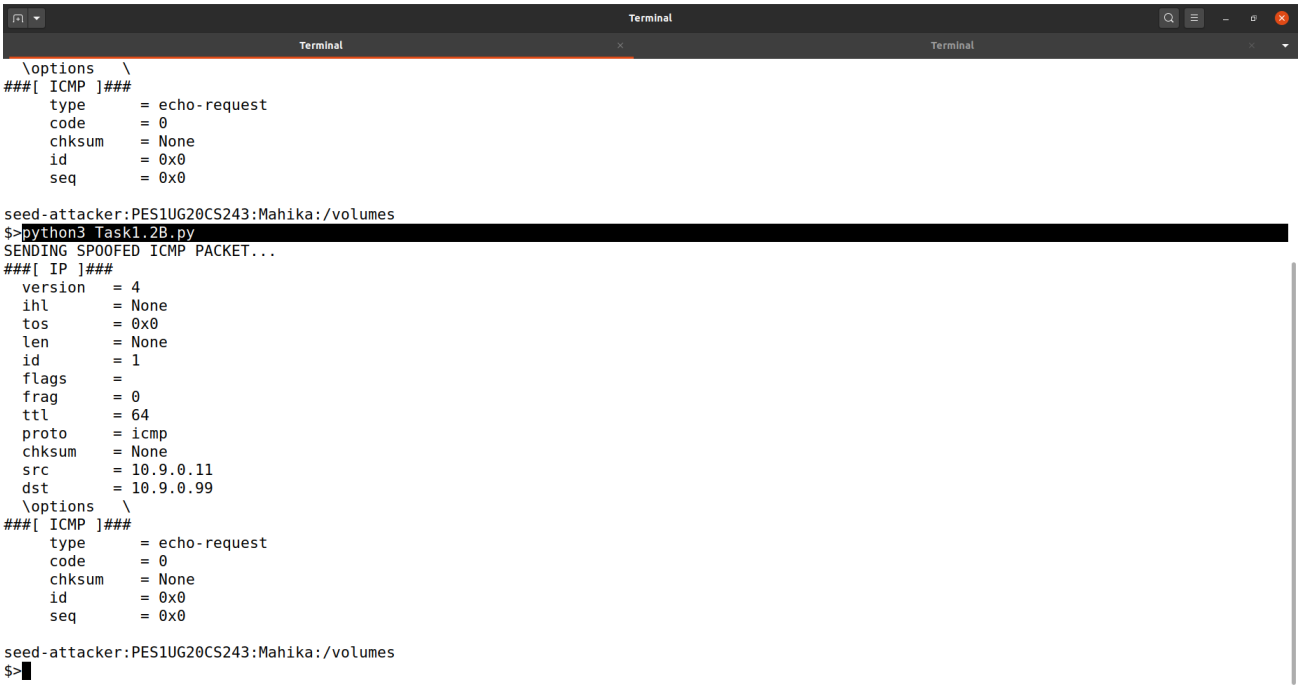


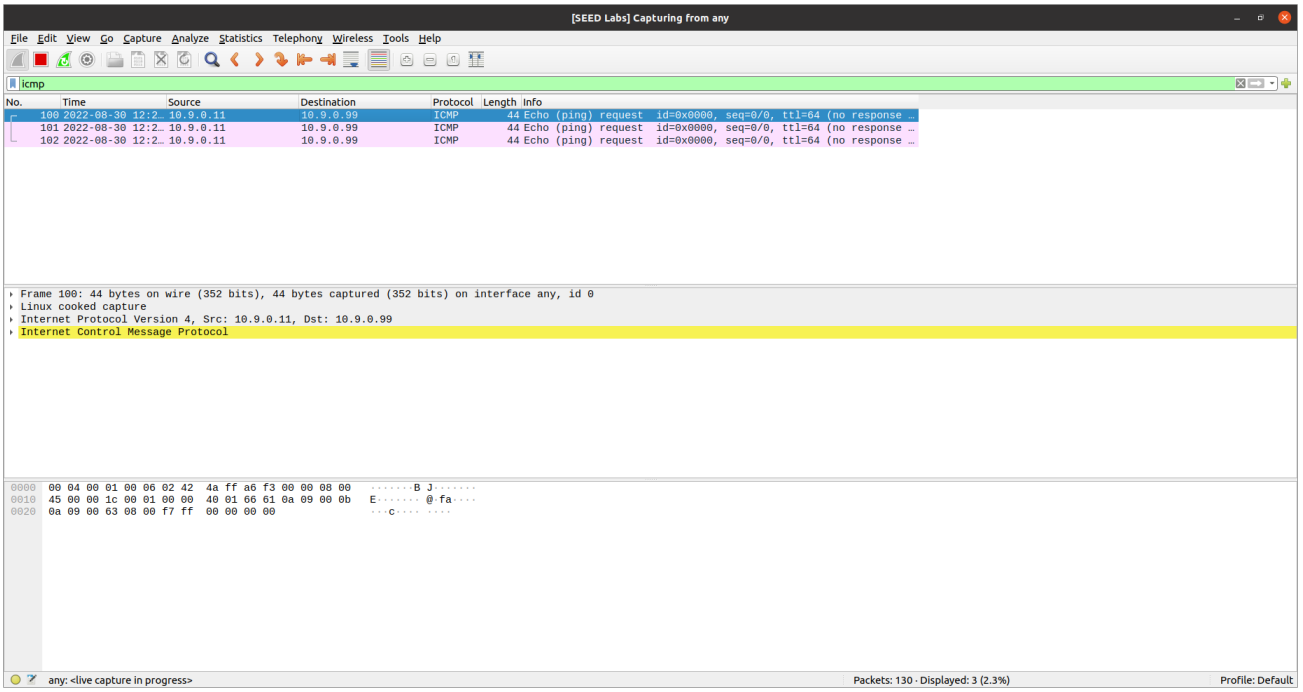
The requests are sent and echo replies are received by the destination IP address.

On the Attacker terminal run the command:

 # python Task1.2B.py

Provide a screenshot of your observations.





A spoofed ICMP echo request is sent to the host using a fake src IP and an echo reply is not received.

Task 1.3 : Traceroute

The objective of this task is to implement a simple traceroute tool using Scapy to estimate the distance, in terms of number of routers, between your VM and a selected destination.

The below code is a simple traceroute implementation using Scapy. It takes hostname or IP address as the input. We create an IP packet with destination address and TTL value and ICMP packet. We send the packet using function sr1(). This function waits for the reply from the destination. If the ICMP reply type is 0, we receive an echo response from the destination, else we increase the TTL value and resend the packet.

Provide a screenshot of the Wireshark capture that shows the ICMP requests sent with increasing TTL and the error response from the routers with a message as “Time to live exceeded”.

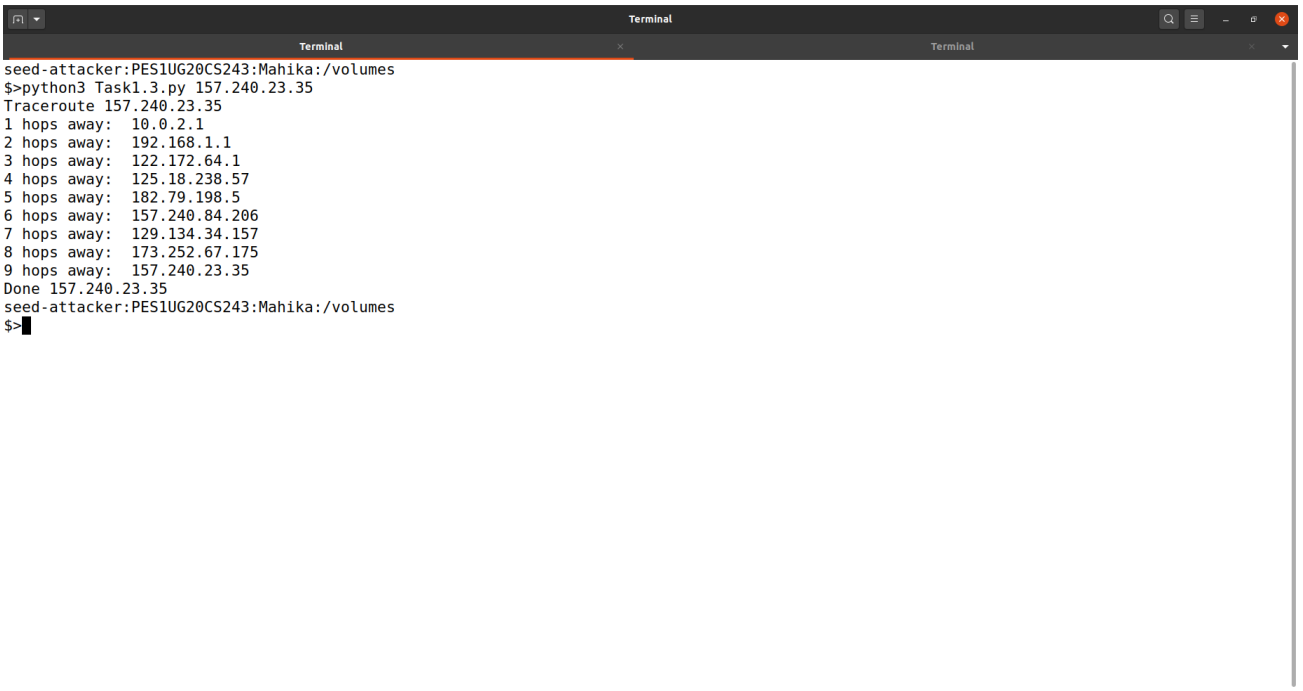
On the Attacker terminal run the command:

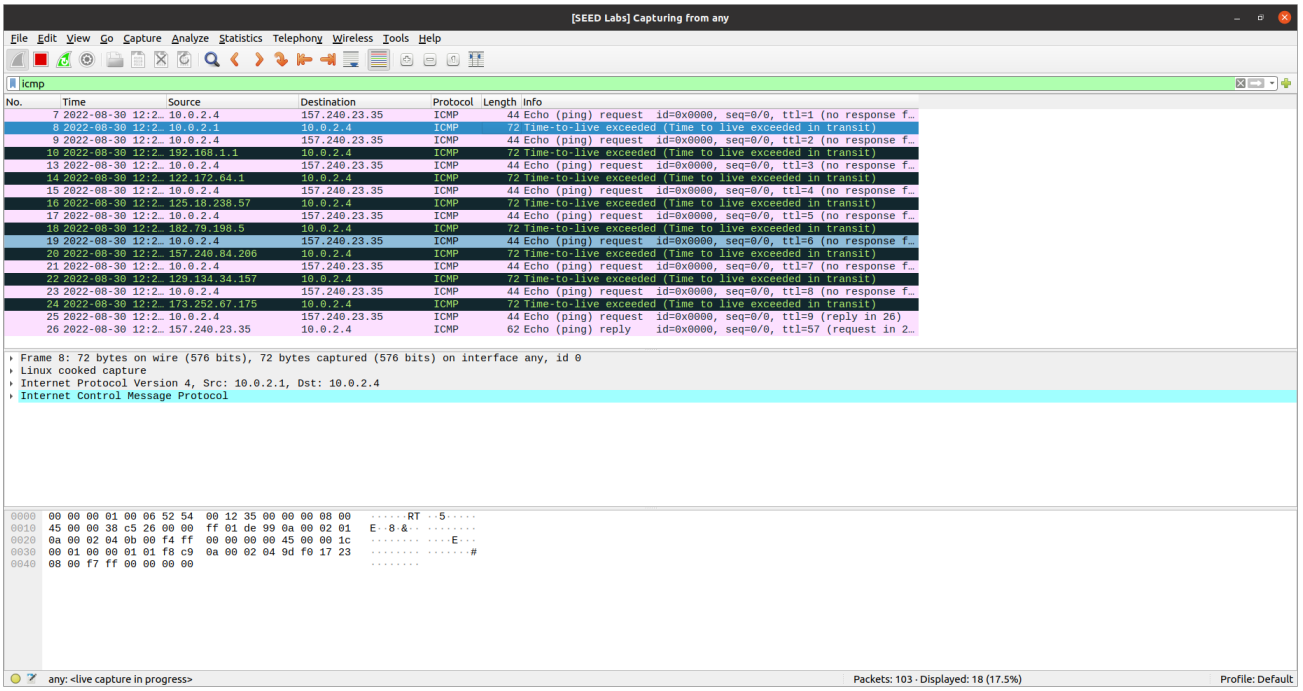
 # python3 Task1.3.py 157.240.23.35

157.240.23.35 is the IP address for facebook.com

On running the above python code, provide a screenshot of the

response.





With every hop if ICMP reply is not received, TTL value is increased, and the packet is sent again. TTL exceeds at some points when this happens.

Task 1.4 : Sniffing and-then Spoofing

In this task, the victim machine pings a non-existing IP address “1.2.3.4”. As the attacker machine is on the same network, it sniffs the request packet, creates a new echo reply packet with IP and ICMP header and sends it to the victim machine. Hence, the user will always receive an echo reply from a non-existing IP address indicating that the machine is alive.

The below code sniffs ICMP packets sent out by the victim machine. Using the callback function, we can use the packets to send the spoofed packets. We retrieve source IP and destination IP from the sniffed packet and create a new IP packet. The new source IP of the spoofed packet is the sniffed packet’s destination IP address and vice versa. We also generate ICMP packets with id and sequence number. In the new packet, ICMP type should be 0 (ICMP reply). To avoid truncated packets, we also add the data to the new packet.

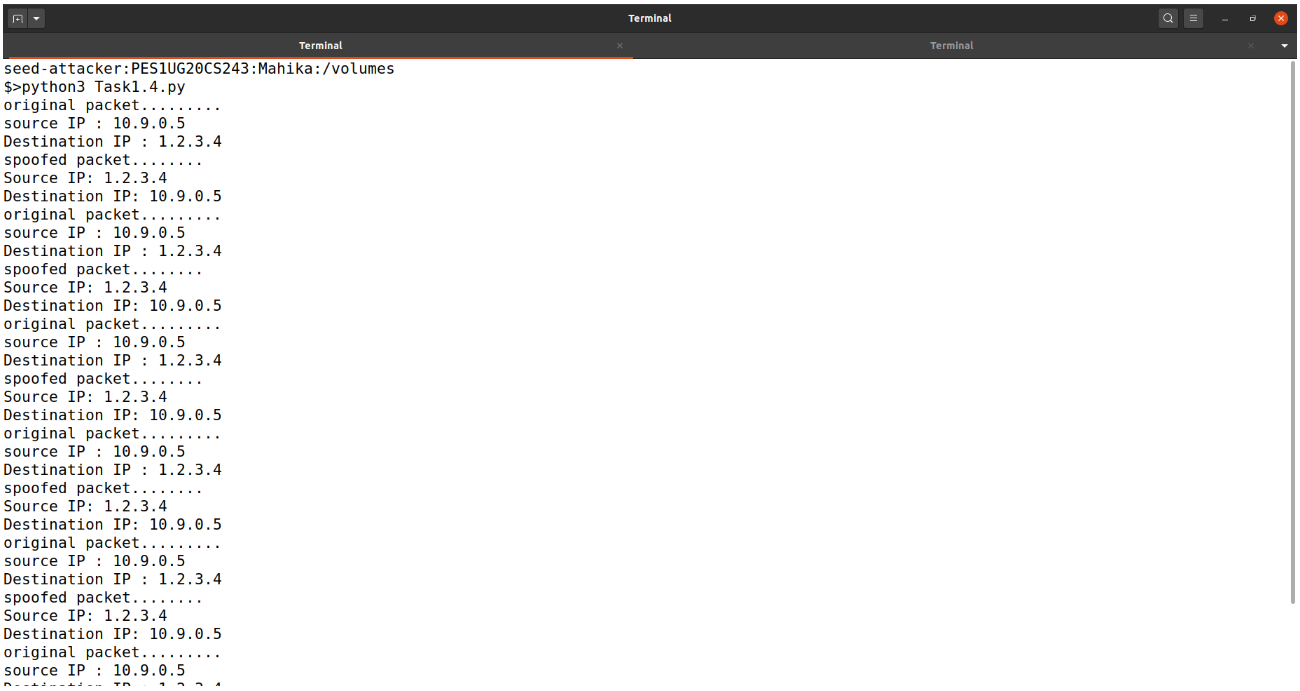
Keep wireshark open before running the python program. Provide wireshark screenshots of the spoofed packets being sent.

Fill in the interface of the attacker machine in the given program and run the code.

On the Attacker terminal run the command:

 # python3 Task1.4.py

Provide a screenshot of your observations.



The source and destination IP of the original packets are switched to make the spoofed packet, to make it appear that ICMP echo reply was sent from the same destination IP address.

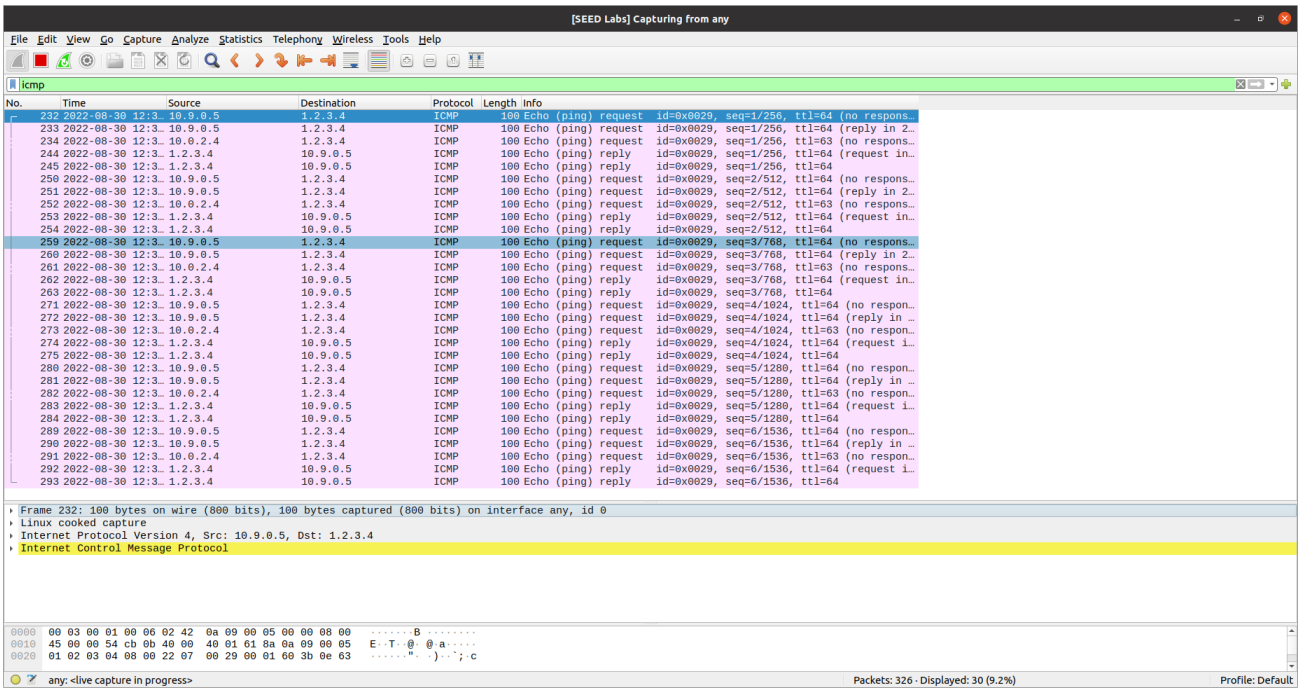
From the **host A** machine's terminal ping 1.2.3.4

On the Host A terminal run the command::

 # ping 1.2.3.4

Provide a screenshot of your observations.

```
root@b361e3734b57:/# 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=57.7 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=13.9 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=15.1 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=29.7 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=13.2 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=16.1 ms
^C
--- 1.2.3.4 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5009ms
rtt min/avg/max/mdev = 13.163/24.279/57.733/15.975 ms
root@b361e3734b57:/#
```



ICMP echo request is sent to the fake IP address, and an echo reply packet is spoofed and sent to host A, with source IP as the fake IP. This notifies Host A that the IP address 1.2.3.4 is alive, when in fact it is the attacker spoofing the packet.