

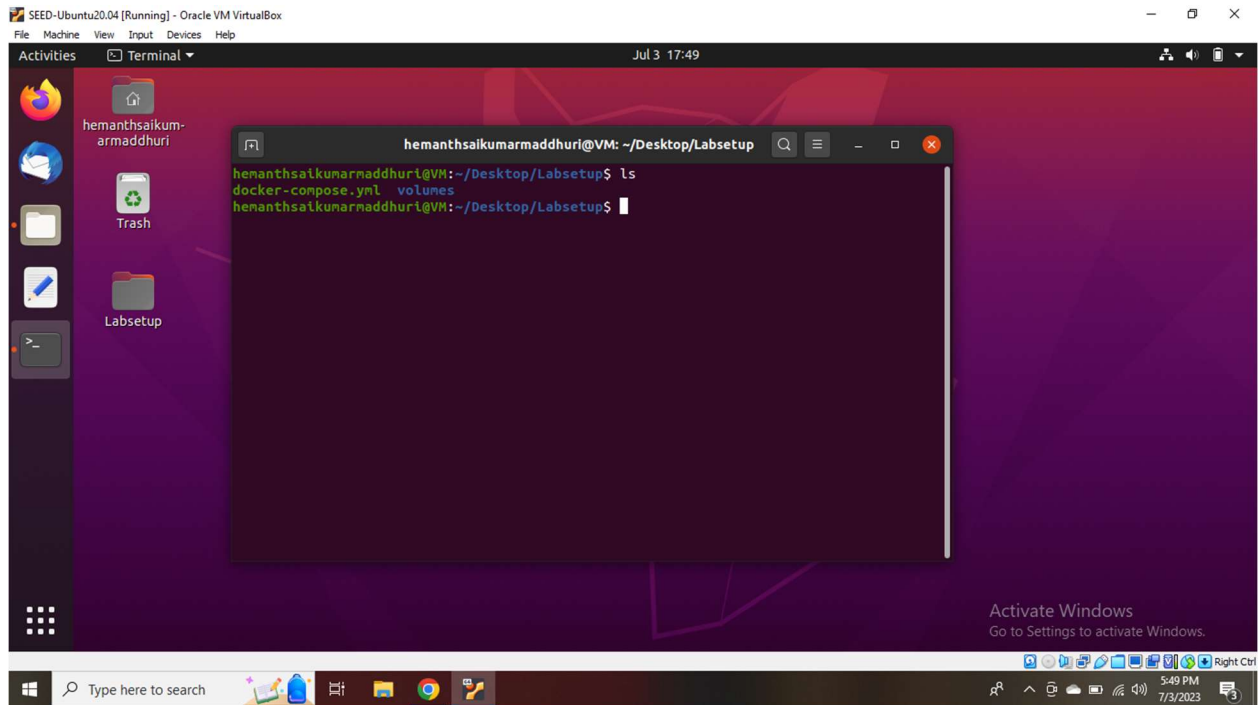
Information and Networking Security

Quiz - 5

Name: Hemanth Sai Kumar Maddhuri

ID: 999902480

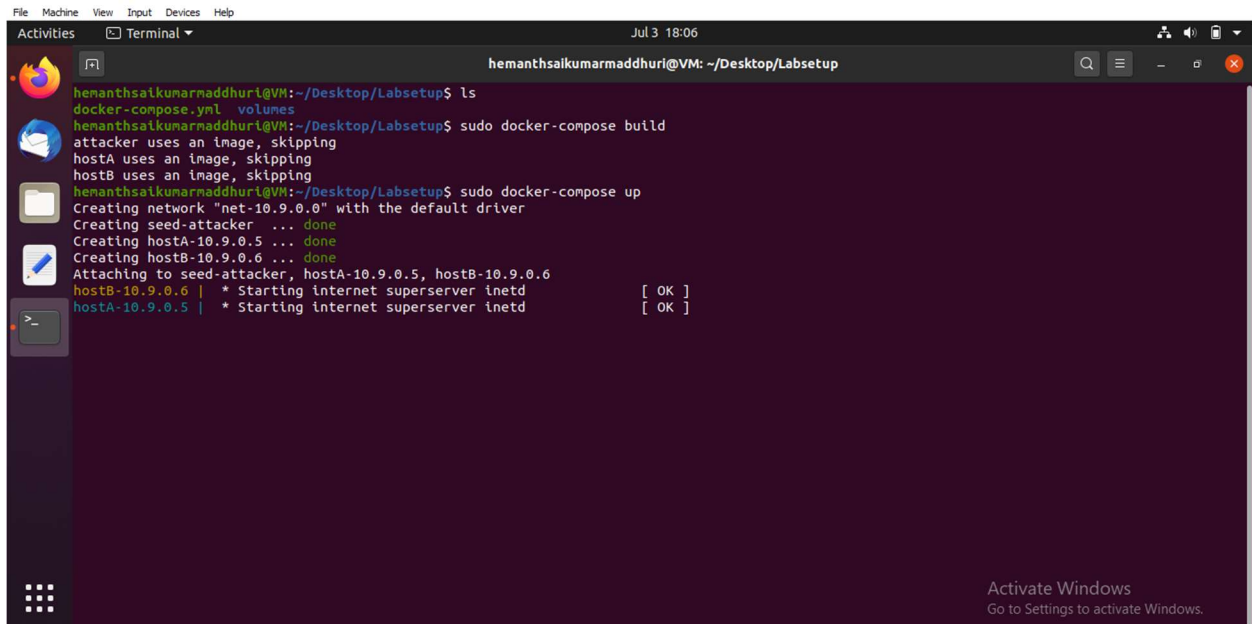
As Instructed I have downloaded Labsetup files required for Packet Sniffing and Spoofing Lab from the following URL https://seedsecuritylabs.org/Labs_20.04/Networking/Sniffing_Spoofing/.



2 Environment Setup using Container

2.1 Container Setup and Commands

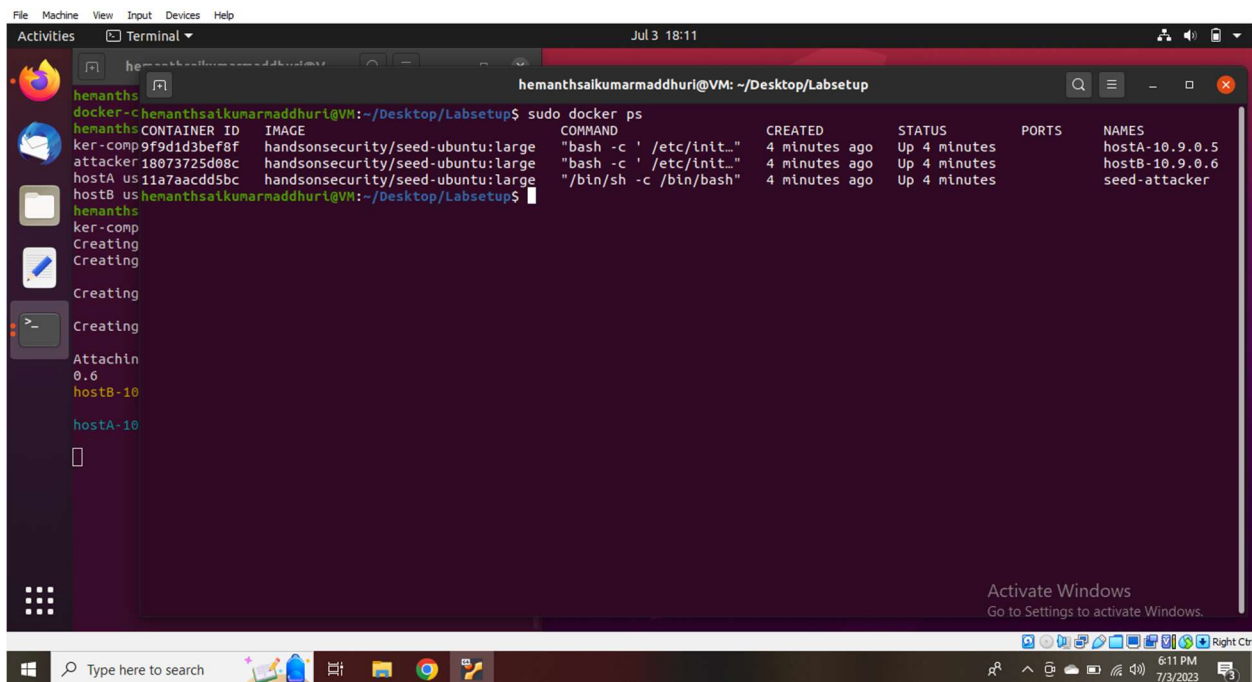
Here we are building the docker and turning it up using the commands shown in the screenshot below.



A terminal window titled 'hemanthsalkumarmaddhuri@VM: ~/Desktop/Labsetup' showing the execution of Docker Compose commands. The user runs 'ls' to show 'docker-compose.yml' and 'volumes'. Then 'sudo docker-compose build' is run, showing that images for 'attacker', 'hostA', and 'hostB' are being skipped. Finally, 'sudo docker-compose up' is run, creating a network 'net-10.9.0.0' and containers 'seed-attacker', 'hostA-10.9.0.5', and 'hostB-10.9.0.6'. The containers are attached to 'seed-attacker' and 'inetd' is started on both hostA and hostB.

```
hemanthsalkumarmaddhuri@VM:~/Desktop/Labsetup$ ls
docker-compose.yml  volumes
hemanthsalkumarmaddhuri@VM:~/Desktop/Labsetup$ sudo docker-compose build
attacker uses an image, skipping
hostA uses an image, skipping
hostB uses an image, skipping
hemanthsalkumarmaddhuri@VM:~/Desktop/Labsetup$ sudo docker-compose up
Creating network "net-10.9.0.0" with the default driver
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 ]
```

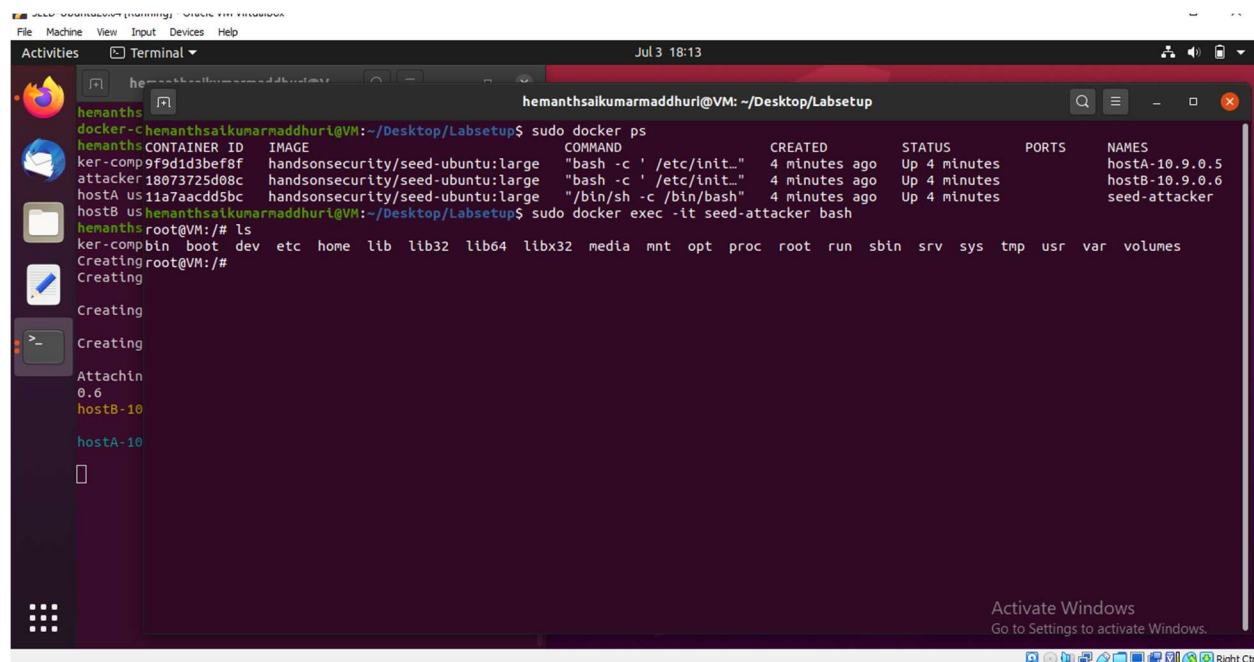
Simultaneously in the other terminal we run the command “**sudo docker ps**” to find out the ID’s of the containers.



A terminal window titled 'hemanthsalkumarmaddhuri@VM: ~/Desktop/Labsetup' showing the output of the 'sudo docker ps' command. The output is a table with columns: CONTAINER ID, IMAGE, COMMAND, CREATED, STATUS, PORTS, and NAMES. It lists three containers: 'ker-comp9f9d1d3bef8f', 'attacker18073725d08c', and 'hostA us11a7aacdd5bc', all using 'handsonsecurity/seed-ubuntu:large' images and running '/bin/sh -c /bin/bash'.

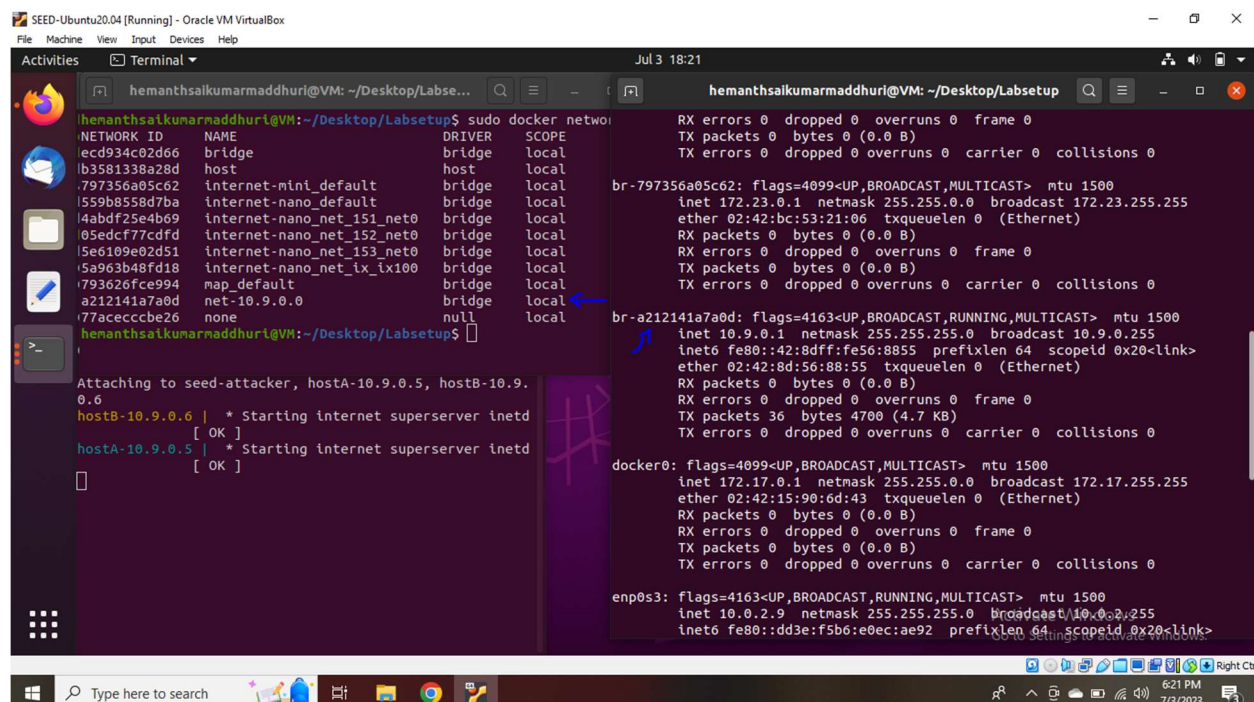
```
hemanthsalkumarmaddhuri@VM:~/Desktop/Labsetup$ sudo docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS   NAMES
ker-comp9f9d1d3bef8f   handsonsecurity/seed-ubuntu:large   "bash -c ' /etc/init..." 4 minutes ago   Up 4 minutes           hostA-10.9.0.5
attacker18073725d08c   handsonsecurity/seed-ubuntu:large   "bash -c ' /etc/init..." 4 minutes ago   Up 4 minutes           hostB-10.9.0.6
hostA us11a7aacdd5bc   handsonsecurity/seed-ubuntu:large   "/bin/sh -c /bin/bash"    4 minutes ago   Up 4 minutes           seed-attacker
```

Using the displayed docker names we now try to get into one of the host machines using the command “**sudo docker exec -it seed-attacker bash**”. As shown below I have logged in and listed the files present in the host machines.



```
hemanths@hemanthsalkumarmaddhuriVM: ~/Desktop/Labsetup$ sudo docker ps
hemanths CONTAINER ID        IMAGE               COMMAND             CREATED   STATUS    PORTS     NAMES
ker-comp9f9d1d3bef8f        handsonsecurity/seed-ubuntu:large   "bash -c ' /etc/init..." 4 minutes ago Up 4 minutes           hostA-10.9.0.5
attacker18073725d08c        handsonsecurity/seed-ubuntu:large   "bash -c ' /etc/init..." 4 minutes ago Up 4 minutes           hostB-10.9.0.6
hostA us11a7acdd5bc         handsonsecurity/seed-ubuntu:large   "/bin/sh -c /bin/bash" 4 minutes ago Up 4 minutes           seed-attacker
hemanths root@VM: /# ls
ker-compbn boot dev etc home lib lib32 lib64 libx32 media mnt opt proc root run sbin srv sys tmp usr var volumes
Creating root@VM: /#
Creating
Creating
Attaching to seed-attacker, hostA-10.9.0.5, hostB-10.9.0.6
hostB-10.9.0.6
hostA-10.9.0.5
```

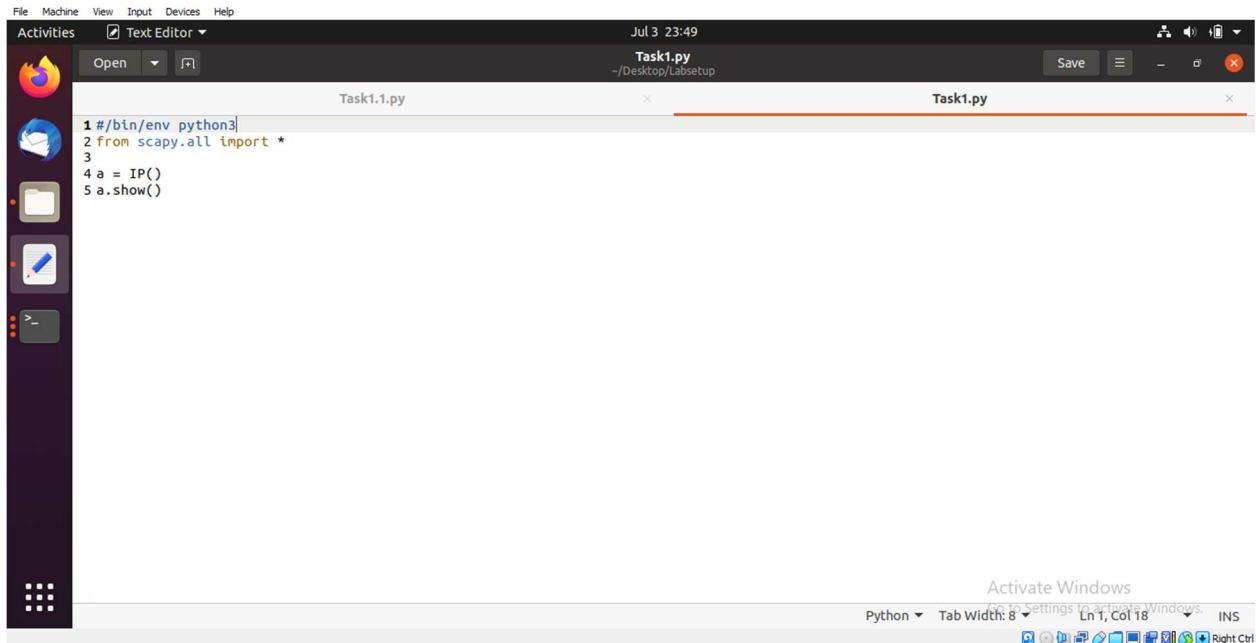
Getting the bridge ID for the host IP and in my case, it is “**br-a212141a7a0d**”.



```
hemanths@hemanthsalkumarmaddhuriVM: ~/Desktop/Labsetup$ sudo docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
ecd934c02d66        bridge             bridge             local
1b3581338a28d        host              host              local
797356a05c62        Internet-mini_default bridge            local
1559b8558d7ba        Internet-nano_default bridge            local
14abdf25e4b69        Internet-nano_net_151_net0 bridge            local
05edcf77cdfd        Internet-nano_net_152_net0 bridge            local
15e6109e02d51        Internet-nano_net_153_net0 bridge            local
15a963b48fd18        Internet-nano_net_1x100 bridge            local
793626fce994        map_default        bridge            local
a212141a7a0d        net-10.9.0.0       bridge            local
77aceccbe26         none              null              local
hemanths@hemanthsalkumarmaddhuriVM: ~/Desktop/Labsetup$
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 ]
```

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

The python code for Task 1 is as shown in the figure.

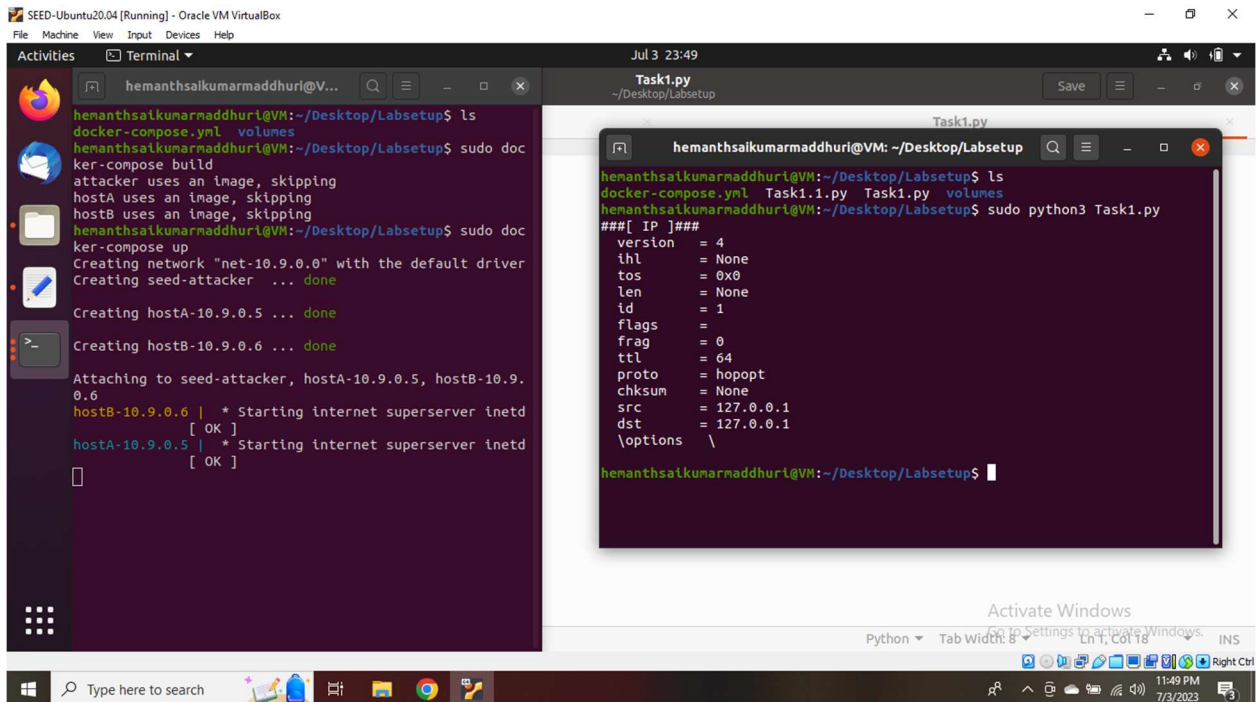


A screenshot of a text editor window titled "Task1.py" showing the following Python code:

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

The editor interface includes a menu bar (File, Machine, View, Input, Devices, Help), a toolbar with "Open" and "Save" buttons, and a status bar at the bottom indicating "Python", "Tab Width: 8", "Ln 1, Col 18", and "INS".

Here we are running the code for Task 1 using the command “**sudo python3 Task1.py**” the information of IP is printed to the command line after the execution of Task1.py.



A screenshot of a terminal window showing the execution of the command `sudo python3 Task1.py`. The output displays the IP structure details:

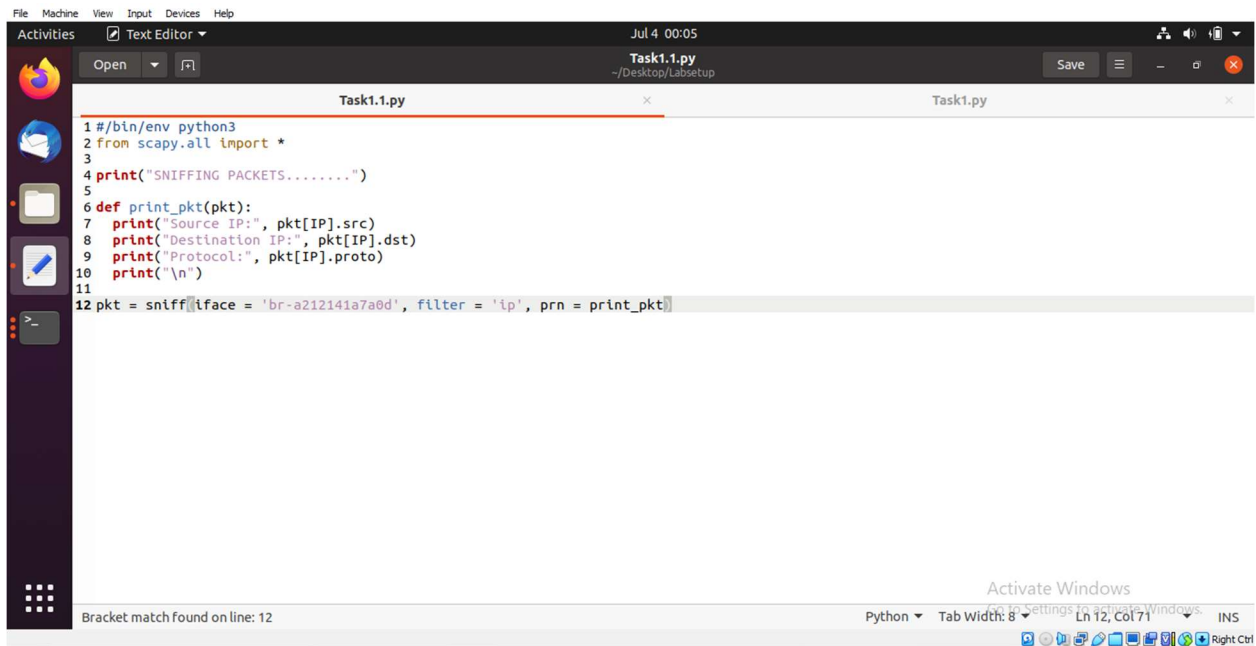
```
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup$ ls
docker-compose.yml Task1.1.py Task1.py volumes
hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ sudo python3 Task1.py
###[ IP ]###
version      = 4
ihl          = None
tos          = 0x0
len          = None
id           = 1
flags        =
frag         = 0
ttl          = 64
proto        = hopopt
chksum       = None
src          = 127.0.0.1
dst          = 127.0.0.1
\options     \

hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$
```

The terminal window also shows the output of `ls` and the execution of `docker-compose build` and `docker-compose up` commands.

3.1 Task 1.1: Sniffing Packets

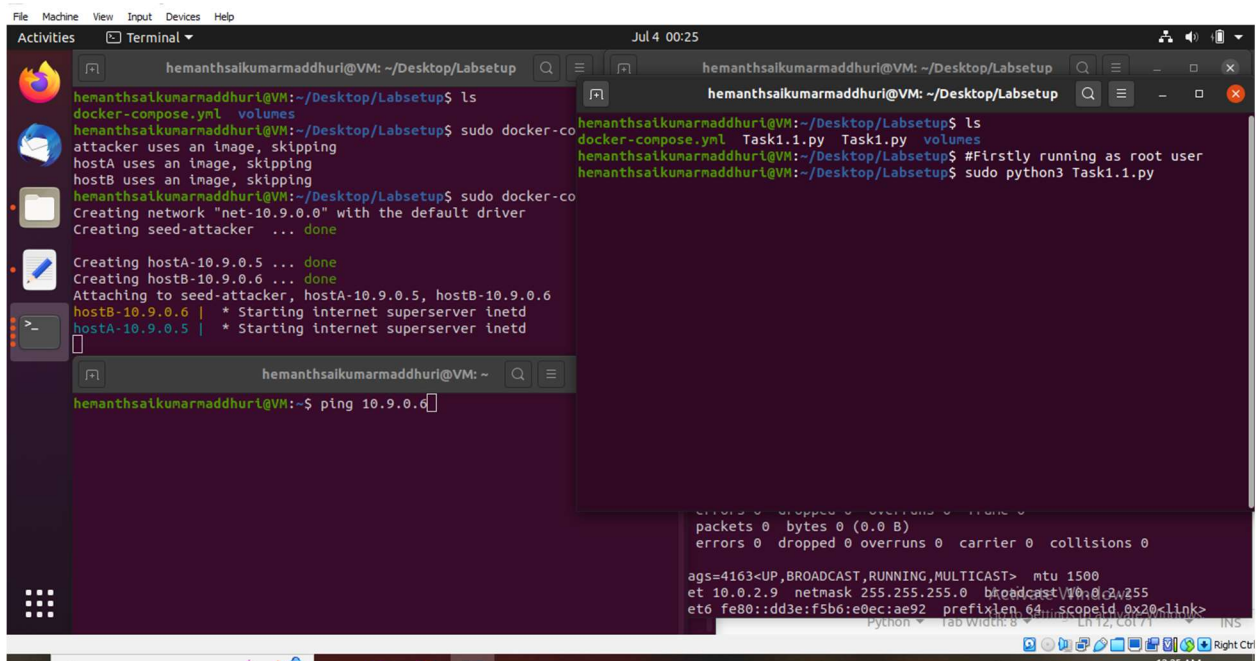
The python code for Task 1.1 is as shown below.



```
1 #!/bin/env python3
2 from scapy.all import *
3
4 print("SNIFFING PACKETS.....")
5
6 def print_pkt(pkt):
7     print("Source IP:", pkt[IP].src)
8     print("Destination IP:", pkt[IP].dst)
9     print("Protocol:", pkt[IP].proto)
10    print("\n")
11
12 pkt = sniff(iface = 'br-a212141a7a0d', filter = 'ip', prn = print_pkt)
```

Task 1.1 A

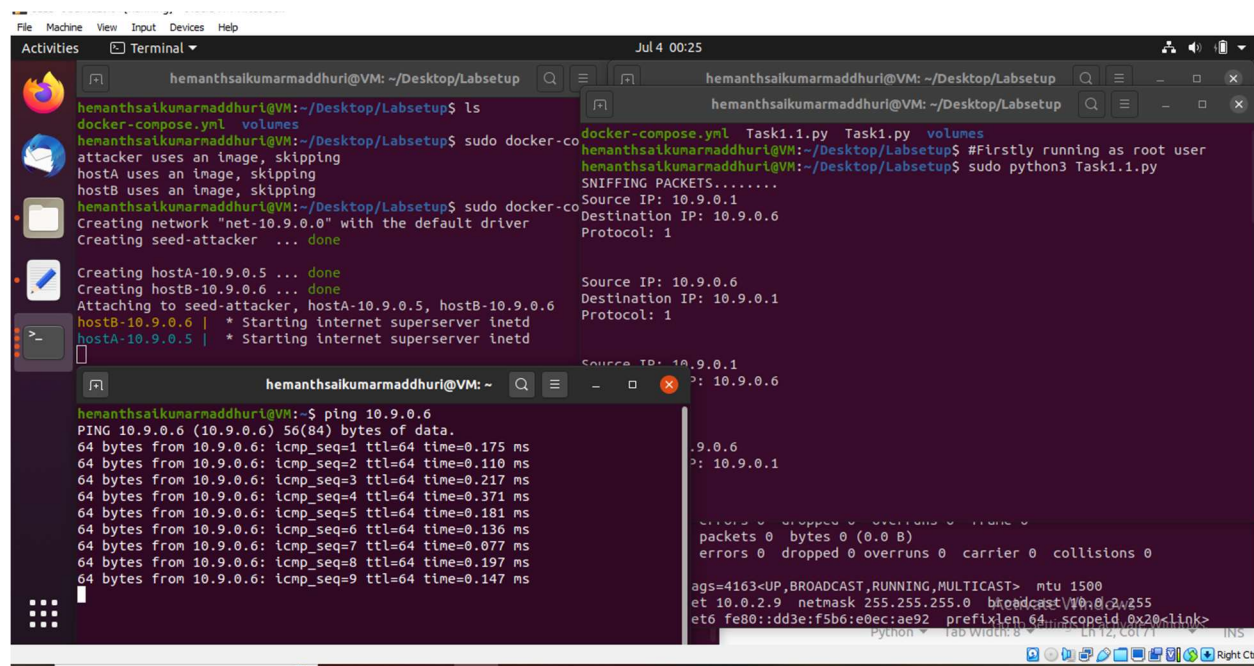
Here we are running the python file required for Task 1.1 as a root user, using the command “**sudo python3 Task1.1A.py**” and in the other terminal we are trying to create some traffic by using the command “**ping 10.9.0.0**”.



```
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup$ ls
docker-compose.yml  volumes
hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ sudo docker-co
attacker uses an image, skipping
hostA uses an image, skipping
hostB uses an image, skipping
hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ sudo docker-co
Creating network "net-10.9.0.0" with the default driver
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
hostA-10.9.0.5 | * Starting internet superserver inetd
hemanthsaikumarmaddhuri@VM: ~
hemanthsaikumarmaddhuri@VM:~$ ping 10.9.0.0

hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ ls
docker-compose.yml  Task1.1.py  volumes
hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ #Firstly running as root user
hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ sudo python3 Task1.1.py
```


After the execution of the above commands, we see the output as shown in the screenshot below. We can see that the packet information such as source IP, destination IP and protocol used is being printed to command line when we run python file as root user.



```
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup$ ls
docker-compose.yml  volumes
hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ sudo docker-co
attacker uses an image, skipping
hostA uses an image, skipping
hostB uses an image, skipping
hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ sudo docker-co
Creating network "net-10.9.0.0" with the default driver
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
hostA-10.9.0.5 | * Starting internet superserver inetd

hemanthsaikumarmaddhuri@VM:~$ ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.175 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.110 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.217 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.371 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.181 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.136 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=64 time=0.077 ms
64 bytes from 10.9.0.6: icmp_seq=8 ttl=64 time=0.197 ms
64 bytes from 10.9.0.6: icmp_seq=9 ttl=64 time=0.147 ms

hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ python3 Task1.1.py
#Firstly running as root user
hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ sudo python3 Task1.1.py
SNIFFING PACKETS.....
Source IP: 10.9.0.1
Destination IP: 10.9.0.6
Protocol: 1

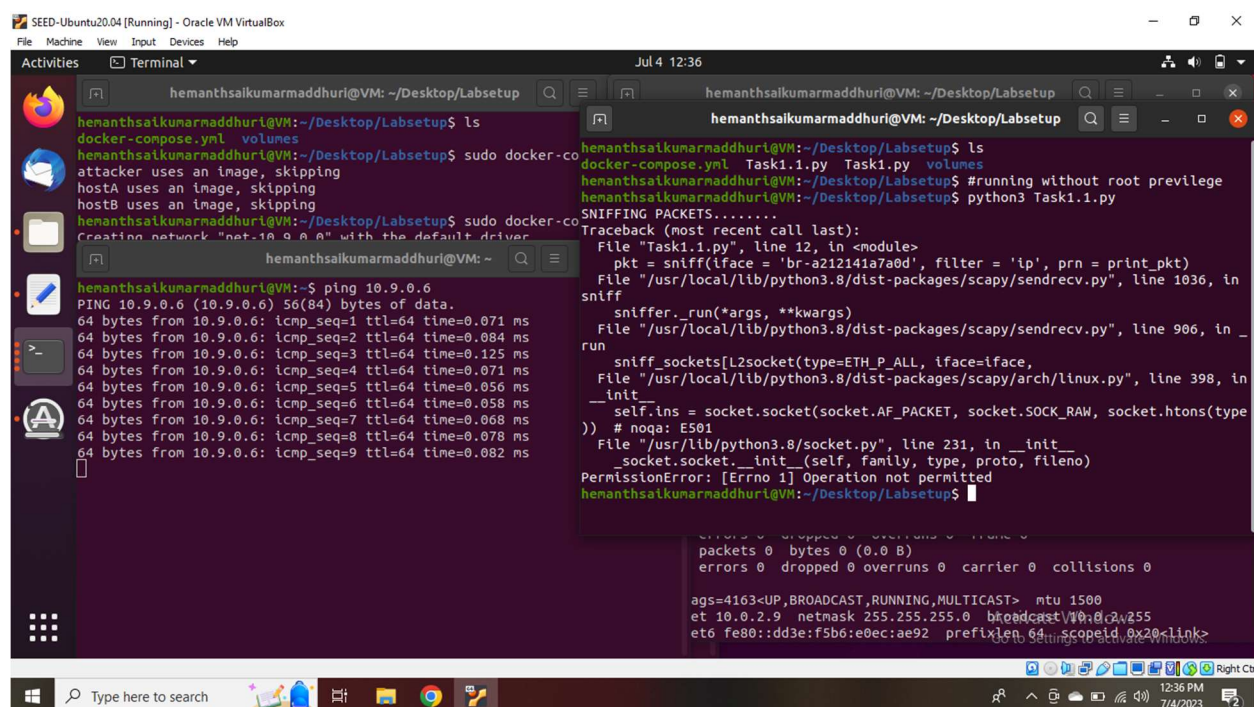
Source IP: 10.9.0.6
Destination IP: 10.9.0.1
Protocol: 1

Source IP: 10.9.0.1
Destination IP: 10.9.0.6
Protocol: 1

errors 0 dropped 0 overruns 0 frame 0
packets 0 bytes 0 (0.0 B)
errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
et 10.0.2.9 netmask 255.255.255.0 broadcast/10.0.2.255
et6 fe80::dd3e:f5b6:e0ec:ae92 prefixlen 64 scopeid 0x20<link>
```

I have observed that when we try to run the python file required for Task 1.1A as non-root user it prints out that **“operation is not permitted”**. Here we can get into the execution but unable to sniff as non-root user.

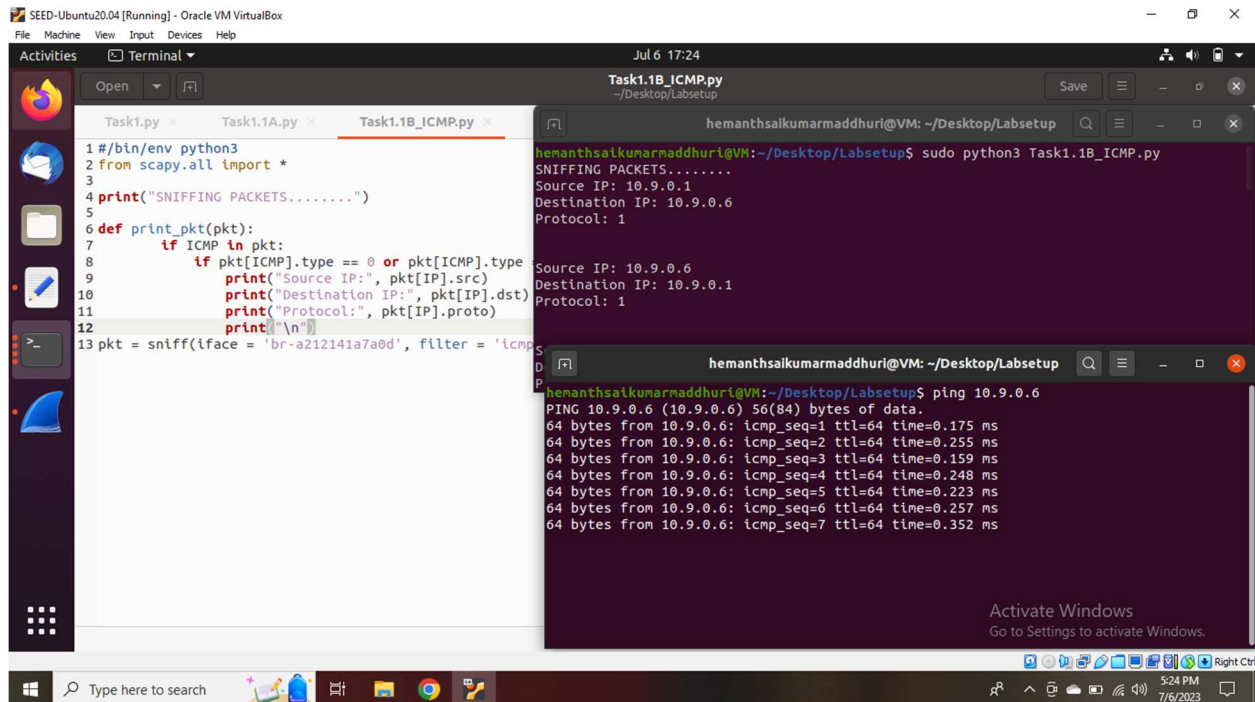


```
hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ ls
docker-compose.yml  volumes
hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ sudo docker-co
attacker uses an image, skipping
hostA uses an image, skipping
hostB uses an image, skipping
hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ sudo docker-co
Creating network "net-10.9.0.0" with the default driver
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
hostA-10.9.0.5 | * Starting internet superserver inetd

hemanthsaikumarmaddhuri@VM:~$ ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.071 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.084 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.125 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.071 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.056 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.058 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=64 time=0.068 ms
64 bytes from 10.9.0.6: icmp_seq=8 ttl=64 time=0.078 ms
64 bytes from 10.9.0.6: icmp_seq=9 ttl=64 time=0.082 ms

hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ python3 Task1.1.py
#running without privilege
hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$ python3 Task1.1.py
SNIFFING PACKETS.....
Traceback (most recent call last):
  File "Task1.1.py", line 12, in <module>
    pkt = sniff(iface = 'br-a212141a7a0d', filter = 'ip', prn = print_pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 1036, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 906, in _run
    sniff_sockets[L2socket(type=ETH_P_ALL, iface=iface,
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py", line 398, in _init
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(type
)) # noqa: E501
  File "/usr/lib/python3.8/socket.py", line 231, in _init
    _socket.socket._init(self, family, type, proto, fileno)
PermissionError: [Errno 1] Operation not permitted
hemanthsaikumarmaddhuri@VM:~/Desktop/Labsetup$
```

In the screenshot below, we are running Task1.1B_ICMP.py to capture only the ICMP packets. I am running the code in one of the terminals while creating traffic on the other one.

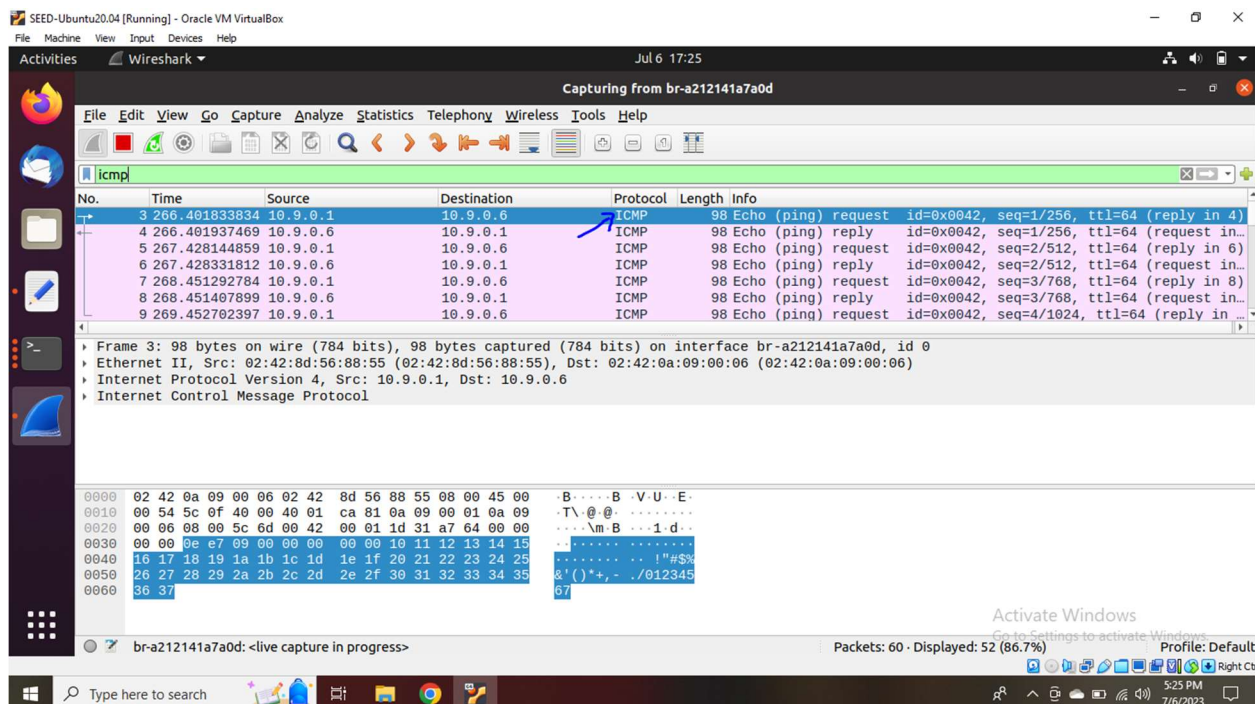


```
Task1.py Task1.1A.py Task1.1B_ICMP.py
1 #/bin/env python3
2 from scapy.all import *
3
4 print("SNIFFING PACKETS.....")
5
6 def print_pkt(pkt):
7     if ICMP in pkt:
8         if pkt[ICMP].type == 0 or pkt[ICMP].type == 8:
9             print("Source IP:", pkt[IP].src)
10            print("Destination IP:", pkt[IP].dst)
11            print("Protocol:", pkt[IP].proto)
12            print("\n")
13 pkt = sniff(iface = 'br-a212141a7a0d', filter = 'icmp')

hemanthsakumarmaddhuri@VM: ~/Desktop/Labsetup
hemanthsakumarmaddhuri@VM:~/Desktop/Labsetup$ sudo python3 Task1.1B_ICMP.py
SNIFFING PACKETS.....
Source IP: 10.9.0.1
Destination IP: 10.9.0.6
Protocol: 1
Source IP: 10.9.0.6
Destination IP: 10.9.0.1
Protocol: 1

hemanthsakumarmaddhuri@VM: ~/Desktop/Labsetup
hemanthsakumarmaddhuri@VM:~/Desktop/Labsetup$ ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data:
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.175 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.255 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.159 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.248 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.223 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.257 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=64 time=0.352 ms
```

Screenshot from the Wireshark is attached here to show that we have captured ICMP packet successfully.



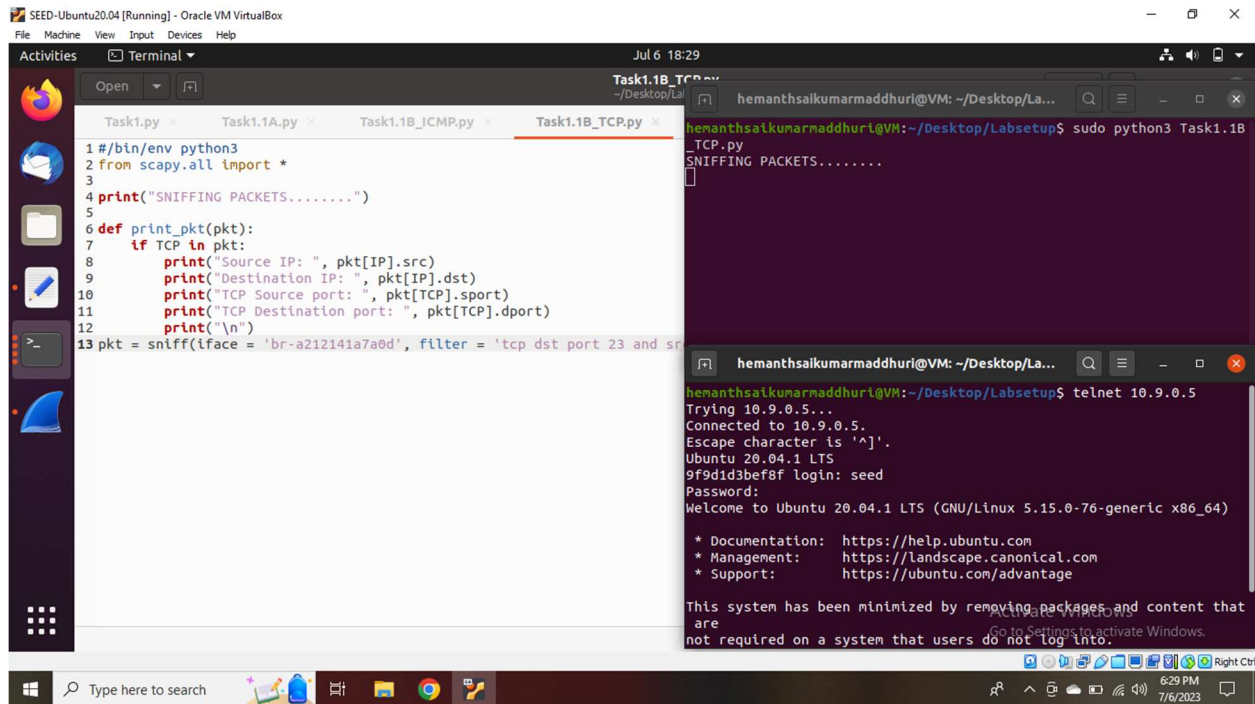
```
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Capturing from br-a212141a7a0d

icmp
No. Time Source Destination Protocol Length Info
3 266.401833834 10.9.0.1 10.9.0.6 ICMP 98 Echo (ping) request id=0x0042, seq=1/256, ttl=64 (reply in 4)
4 266.401937469 10.9.0.6 10.9.0.1 ICMP 98 Echo (ping) reply id=0x0042, seq=1/256, ttl=64 (request in...)
5 267.428144859 10.9.0.1 10.9.0.6 ICMP 98 Echo (ping) request id=0x0042, seq=2/512, ttl=64 (reply in 6)
6 267.428331812 10.9.0.6 10.9.0.1 ICMP 98 Echo (ping) reply id=0x0042, seq=2/512, ttl=64 (request in...)
7 268.451292784 10.9.0.1 10.9.0.6 ICMP 98 Echo (ping) request id=0x0042, seq=3/768, ttl=64 (reply in 8)
8 268.451407899 10.9.0.6 10.9.0.1 ICMP 98 Echo (ping) reply id=0x0042, seq=3/768, ttl=64 (request in...)
9 269.452702397 10.9.0.1 10.9.0.6 ICMP 98 Echo (ping) request id=0x0042, seq=4/1024, ttl=64 (reply in...)

Frame 3: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface br-a212141a7a0d, id 0
  Ethernet II, Src: 02:42:8d:56:88:55 (02:42:8d:56:88:55), Dst: 02:42:0a:09:00:06 (02:42:0a:09:00:06)
  Internet Protocol Version 4, Src: 10.9.0.1, Dst: 10.9.0.6
  Internet Control Message Protocol

0000 02 42 0a 09 00 06 02 42 8d 56 88 55 08 00 45 00  B....B.V.U..E.
0010 00 54 5c 0f 40 00 40 01 ca 81 0a 09 00 01 0a 09  .T.@.@.....
0020 00 06 08 00 5c 6d 00 42 00 01 1d 31 a7 64 00 00  ....m.B...1.d...
0030 00 00 0e e7 09 00 00 00 00 00 10 11 12 13 14 15  .....!#$%&()*+,-./012345
0040 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25  .....
0050 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35  .....
0060 36 37 67
```

In the below screenshot, we are running the Task1.1B_TCP.py python file in one of the terminals and other terminal we are creating traffic using command “telnet 10.9.0.5”.



```
Task1.1B_TCP.py
1 #!/bin/env python3
2 from scapy.all import *
3
4 print("SNIFFING PACKETS.....")
5
6 def print_pkt(pkt):
7     if TCP in pkt:
8         print("Source IP: ", pkt[IP].src)
9         print("Destination IP: ", pkt[IP].dst)
10        print("TCP Source port: ", pkt[TCP].sport)
11        print("TCP Destination port: ", pkt[TCP].dport)
12        print("\n")
13 pkt = sniff(iface = 'br-a212141a7a0d', filter = 'tcp dst port 23 and src ip 10.9.0.1')
14
```

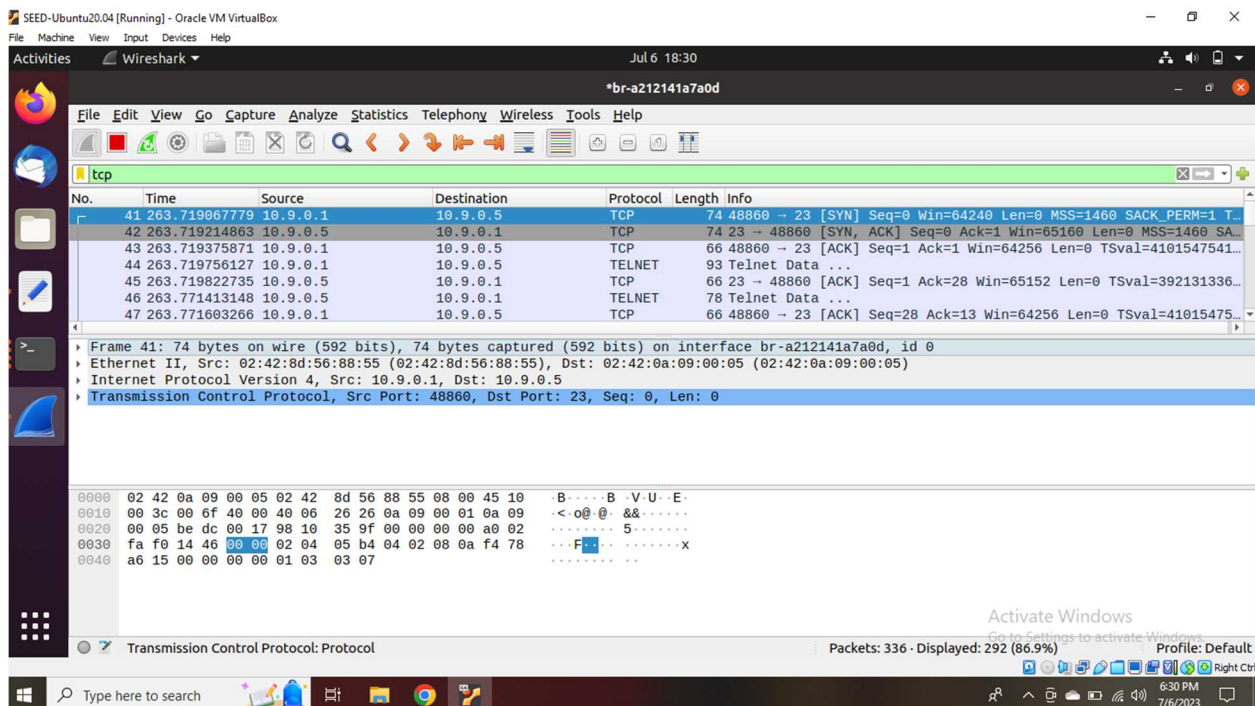
```
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup$ sudo python3 Task1.1B_TCP.py
SNIFFING PACKETS.....

hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup$ telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
9f9did3bef8f login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.15.0-76-generic x86_64)

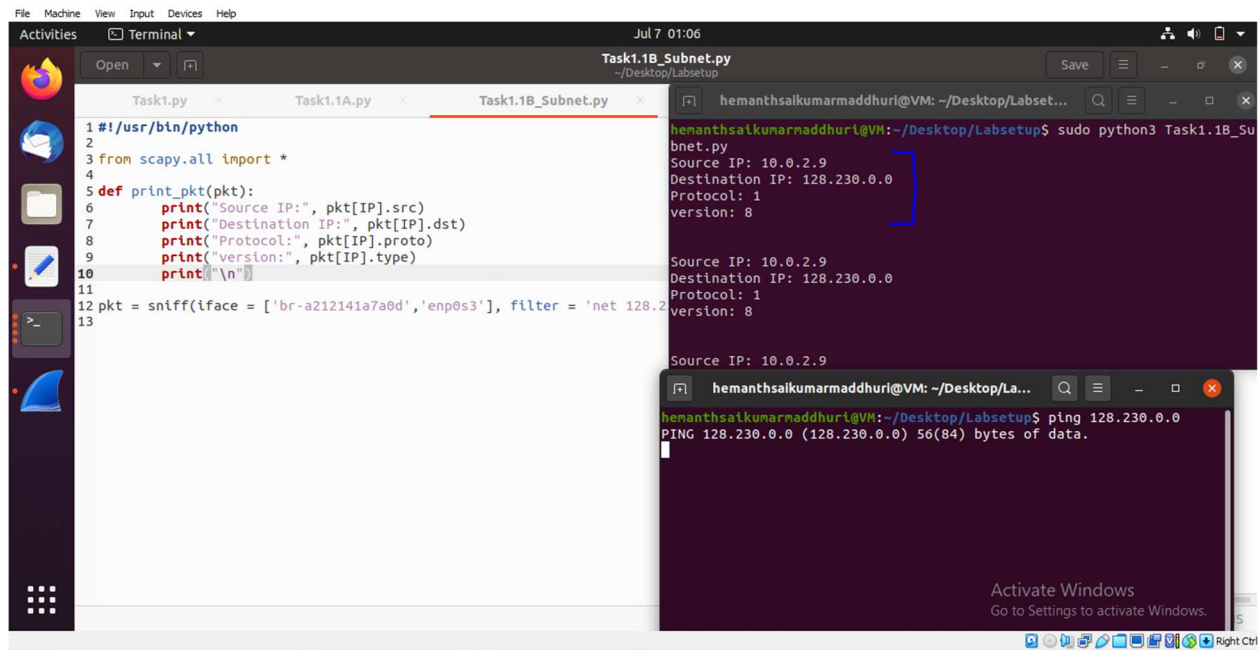
* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:        https://ubuntu.com/advantage

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

The same is captured using the Wireshark application as shown below.

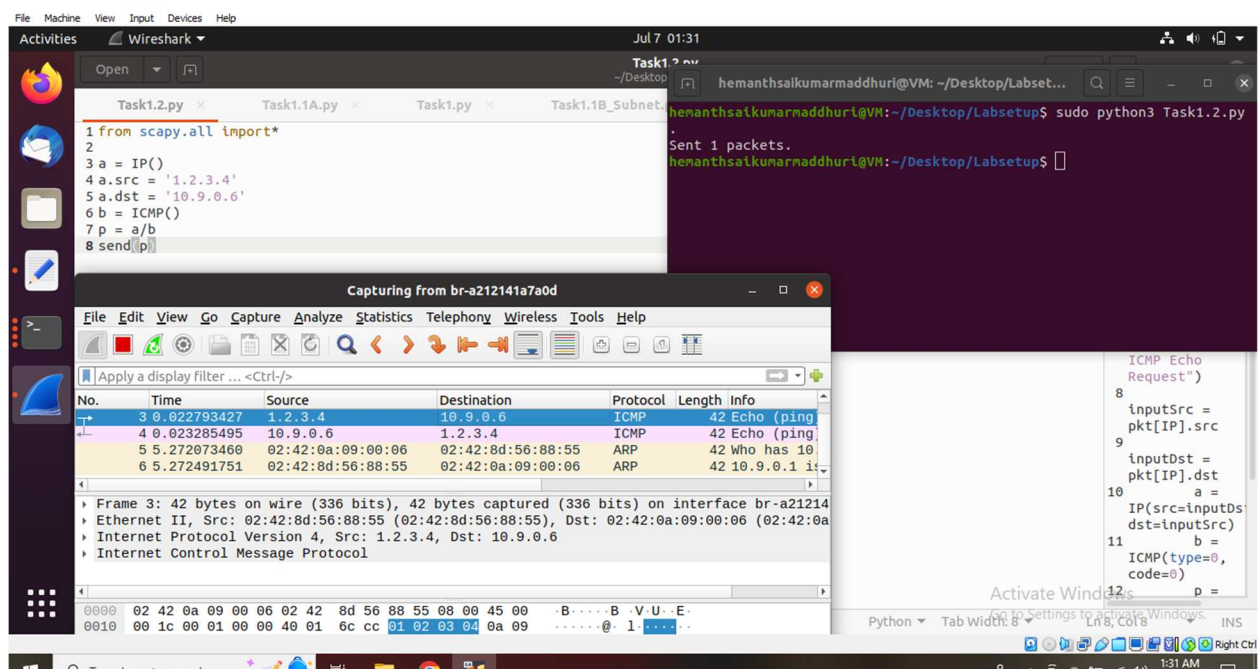


In the below screenshot, we are running the Task1.1B_Subnet.py python file in one of the terminals and other terminals we are creating traffic using command “ping 128.230.0.0”.



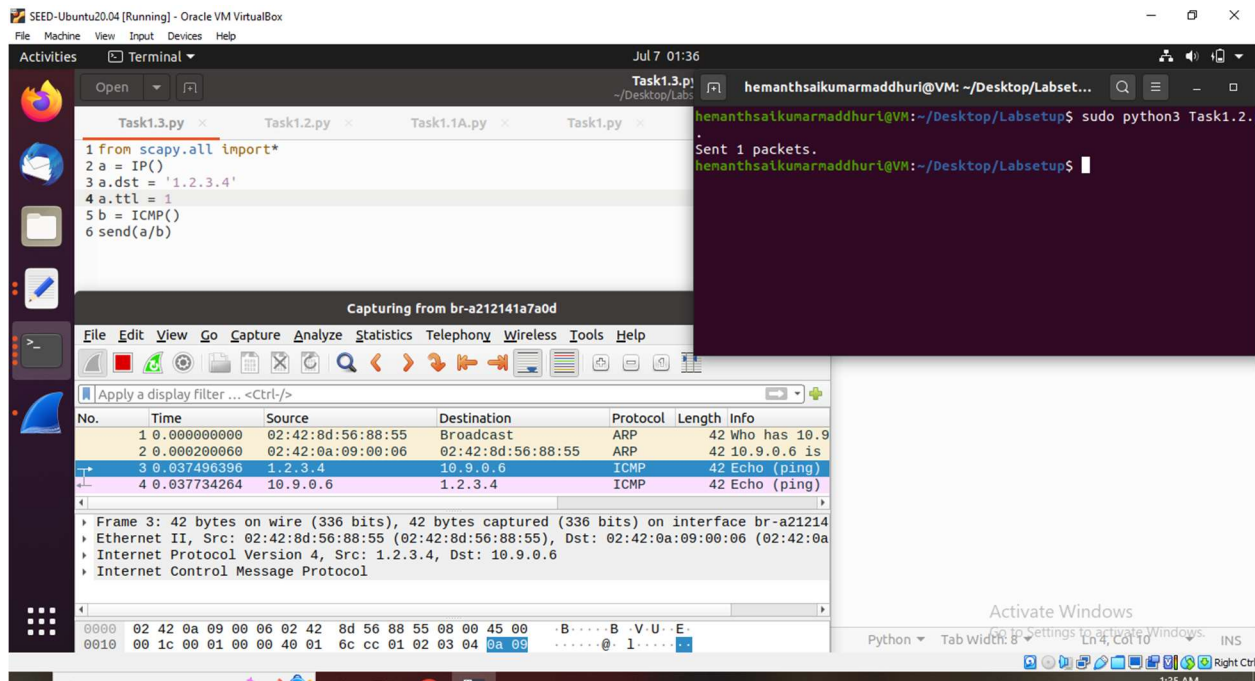
3.2 Task 1.2: Spoofing ICMP Packets

Here we are trying to spoof an ICMP echo request packet with an arbitrary source IP address. As shown in the screenshot below I have run the python file and we can see that output as “sent 1 packet” similarly in the Wireshark application too.

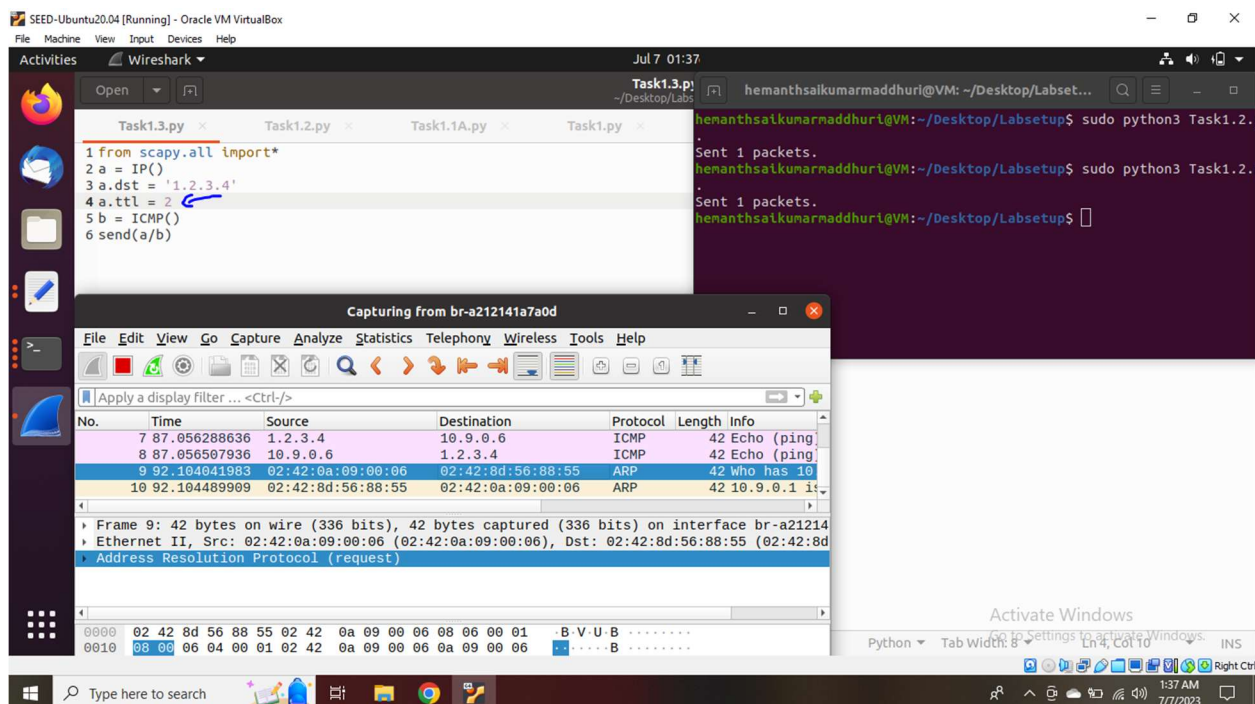


3.3 Task 1.3: Traceroute

Here we are trying to estimate the distance between the routers. So I have run the below code with TTL value set to “1” and in this instance the packet is being delivered.



As instructed, I have next set the TTL value to “2” whereas even in this case the packet was delivered and the same can be found in Wireshark application too.



As we did in the previous steps, I have set the TTL value to “3” here even in this case the packet was delivered successfully and I hope the distance between routers in my case can be 1, 2 or 3.

The screenshot shows a virtual machine environment with a terminal window and a Wireshark packet capture window. The terminal window displays the execution of a Python script named Task1.3.py, which sends ICMP echo requests with a TTL of 3. The Wireshark window shows the captured packets, including the ICMP echo requests and replies.

```
Task1.3.py
1 from scapy.all import *
2 a = IP()
3 a.dst = '1.2.3.4'
4 a.ttl = 3
5 b = ICMP()
6 send(a/b)
```

```
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup$ sudo python3 Task1.2.
Sent 1 packets.
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup$ sudo python3 Task1.2.
Sent 1 packets.
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup$ sudo python3 Task1.2.
Sent 1 packets.
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup$
```

Wireshark packet capture details:

No.	Time	Source	Destination	Protocol	Length	Info
13	173.762859525	1.2.3.4	10.9.0.6	ICMP	42	Echo (ping)
14	173.763174930	10.9.0.6	1.2.3.4	ICMP	42	Echo (ping)
15	178.687766099	02:42:0a:56:88:55	02:42:8d:56:88:55	ARP	42	Who has 10
16	178.888043598	02:42:8d:56:88:55	02:42:0a:09:00:06	ARP	42	10.9.0.1

Frame 13: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface br-a2121
Ethernet II, Src: 02:42:8d:56:88:55 (02:42:8d:56:88:55), Dst: 02:42:0a:09:00:06 (02:42:0a:
Internet Protocol Version 4, Src: 1.2.3.4, Dst: 10.9.0.6
Internet Control Message Protocol

3.4 Task 1.4 Sniffing and then Spoofing

As per given hint I have tried the command “ip route get 1.2.3.4” and understood its characteristics.

The screenshot shows a virtual machine environment with a terminal window and a Python script named Task1.4.py. The terminal window displays the execution of the script, which sniffs and spoofs ICMP echo requests. The terminal also shows the output of the command "ip route get 1.2.3.4".

```
Task1.4.py
1#!/usr/bin/python
2
3from scapy.all import *
4
5def sniff_spoof(pkt):
6    if pkt[ICMP].type == 8:
7        print("Received ICMP Echo Request")
8        a = IP(src = pkt[IP].dst, dst = pkt[IP].src)
9        b = ICMP()
10       p = a / b
11       send(p)
12       print("Spoofed ICMP Echo Reply Sent")
13 sniff(filter="icmp", prn=sniff_spoof)
```

```
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup$ ip route get 1.2.3.4
1.2.3.4 via 10.0.2.1 dev enp0s3 src 10.0.2.9 uid 1001
cache
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup$
```

The screenshot shows a Windows desktop with a virtual machine running Ubuntu 20.04. The terminal window displays a Python script for sniffing ICMP Echo requests and replies. The script uses Scapy to sniff on the interface 'br-a212141a7a0d1' and prints the source and destination IP addresses of the packets. The output shows several packets being received and spoofed.

```

1 #!/usr/bin/python
2
3 from scapy.all import *
4
5 def sniff_spoof(pkt):
6     if pkt[ICMP].type == 8:
7         print("Received ICMP Echo Request")
8         a = IP(src = pkt[IP].dst, dst = pkt[IP].src)
9         b = ICMP(type = 0, id = pkt[ICMP].id, seq = pkt[ICMP].seq)
10        p = a / b / pkt[3].load #pkt is printed as array in hex
11        send(p)
12        print("Spoofed ICMP Echo Reply Sent")
13 pkt = sniff(iface = 'br-a212141a7a0d1', filter="icmp", count=0)
  
```

The terminal output shows the following sequence of events:

- Received ICMP Echo Request
- Sent 1 packets.
- Spoofed ICMP Echo Reply Sent
- Received ICMP Echo Request
- Sent 1 packets.
- Spoofed ICMP Echo Reply Sent
- Received ICMP Echo Request
- Sent 1 packets.
- Spoofed ICMP Echo Reply Sent

The terminal also shows the output of a ping command:

```

hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup$ 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=34.3 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=65.8 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=60.6 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=37.9 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=36.2 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=55.0 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=64 time=49.2 ms
64 bytes from 1.2.3.4: icmp_seq=8 ttl=64 time=50.9 ms
64 bytes from 1.2.3.4: icmp_seq=9 ttl=64 time=48.0 ms
64 bytes from 1.2.3.4: icmp_seq=10 ttl=64 time=56.2 ms
64 bytes from 1.2.3.4: icmp_seq=11 ttl=64 time=56.9 ms
64 bytes from 1.2.3.4: icmp_seq=12 ttl=64 time=48.8 ms
^C
--- 1.2.3.4 ping statistics ---
12 packets transmitted, 12 received, 0% packet loss, time 11914ms
rtt min/avg/max/mdev = 34.294/49.981/65.832/9.426 ms
  
```

The image shows a Windows 10 desktop with a virtual machine (VM) running Ubuntu 20.04. The VM is titled "SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox". The Ubuntu desktop environment includes a sidebar with application icons (Firefox, Mail, Files, etc.) and a top bar with system status and window controls. A terminal window is open, displaying a Python script for sniffing ICMP echo requests. The script is as follows:

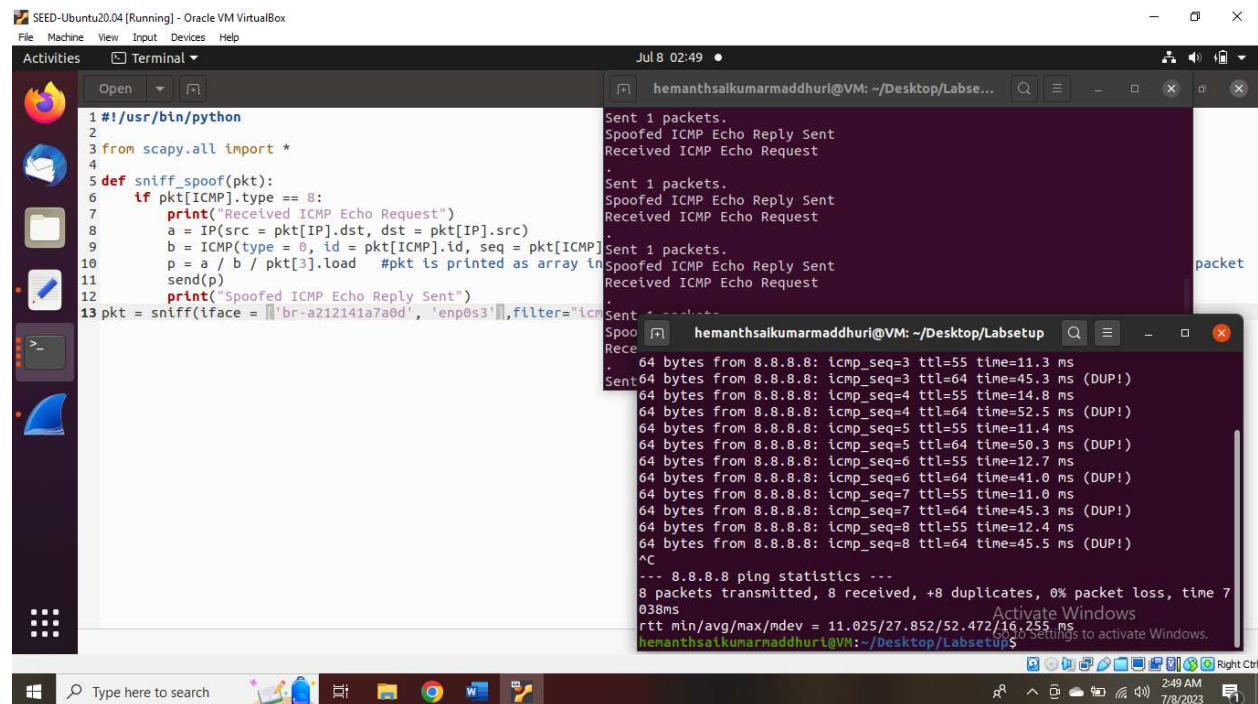
```
1 #!/usr/bin/python
2
3 from scapy.all import *
4
5 def sniff_spoof(pkt):
6     if pkt[ICMP].type == 8:
7         print("Received ICMP Echo Request")
8         a = IP(src = pkt[IP].dst, dst = pkt[IP].src)
9         b = ICMP(type = 0, id = pkt[ICMP].id, seq = pkt[ICMP].seq)
10        p = a / b / pkt[3].load #pkt is printed as array in packet
11        send(p)
12        print("Spoofed ICMP Echo Reply Sent")
13 pkt = sniff(iface = 'br-a212141a7a0d', 'enp0s3', filter="icmp")
```

The terminal output shows the script running successfully, receiving and spoofing ICMP echo requests. Below the script, the user runs a ping command:

```
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup$ 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=34.3 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=65.8 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=60.6 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=37.9 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=36.2 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=55.0 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=64 time=49.2 ms
64 bytes from 1.2.3.4: icmp_seq=8 ttl=64 time=50.9 ms
64 bytes from 1.2.3.4: icmp_seq=9 ttl=64 time=48.0 ms
64 bytes from 1.2.3.4: icmp_seq=10 ttl=64 time=56.2 ms
64 bytes from 1.2.3.4: icmp_seq=11 ttl=64 time=56.9 ms
64 bytes from 1.2.3.4: icmp_seq=12 ttl=64 time=48.8 ms
^C
--- 1.2.3.4 ping statistics ---
12 packets transmitted, 12 received, 0% packet loss, time 11914ms
rtt min/avg/max/mdev = 34.294/49.981/65.832/9.426 ms
```

The terminal window is titled "hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup". The Windows taskbar at the bottom shows the Start button, a search bar, and several pinned applications including File Explorer, Google Chrome, and Microsoft Word. The system clock in the bottom right corner indicates the time is 2:47 AM on 7/8/2023.

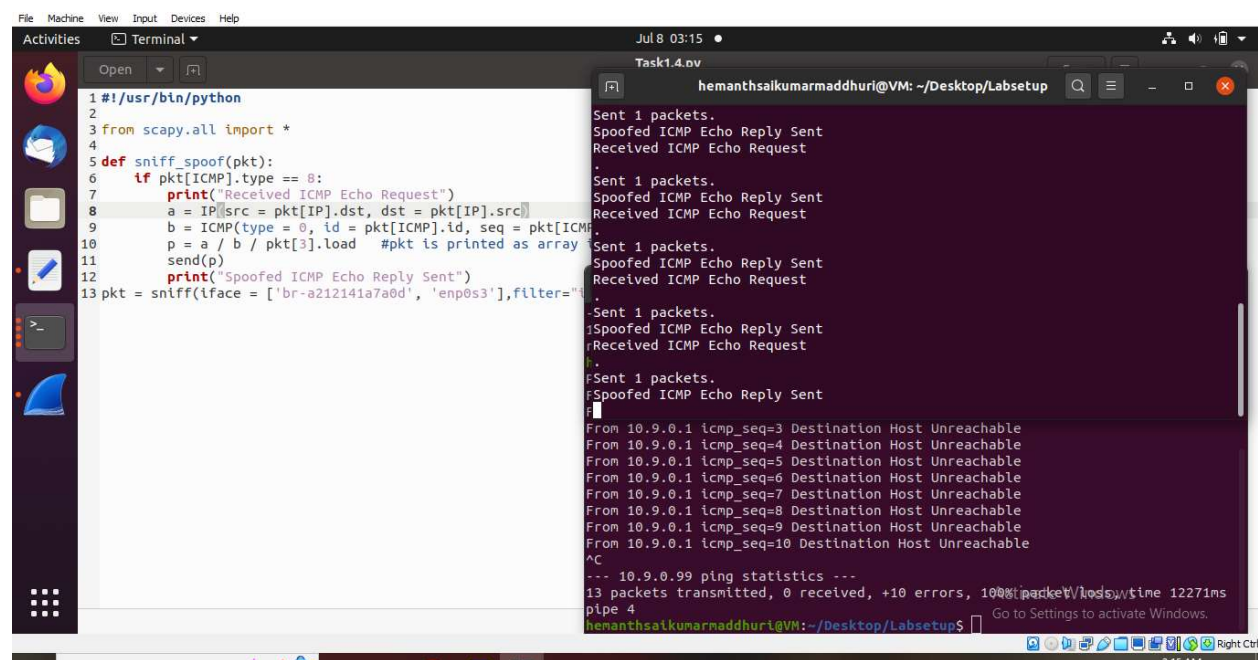
While the above process is still going on, in one of the terminals we run the command “**ping 8.8.8.8**” as instructed in the lab manual and we can clearly see that the packet transmission is successful, and we have successfully sniffed and spoofed the packet.



```
1#!/usr/bin/python
2
3from scapy.all import *
4
5def sniff_spoof(pkt):
6    if pkt[ICMP].type == 8:
7        print("Received ICMP Echo Request")
8        a = IP(src = pkt[IP].dst, dst = pkt[IP].src)
9        b = ICMP(type = 0, id = pkt[ICMP].id, seq = pkt[ICMP].seq)
10       p = a / b / pkt[3].load #pkt is printed as array in hex
11       send(p)
12       print("Spoofed ICMP Echo Reply Sent")
13 pkt = sniff(iface = ['br-a212141a7a0d', 'enp0s3'], filter="icmp")
```

```
Sent 1 packets.
Spoofed ICMP Echo Reply Sent
Received ICMP Echo Request
.
Sent 1 packets.
Spoofed ICMP Echo Reply Sent
Received ICMP Echo Request
.
Sent 1 packets.
Spoofed ICMP Echo Reply Sent
Received ICMP Echo Request
.
Sent 1 packets.
Spoofed ICMP Echo Reply Sent
Received ICMP Echo Request
.
64 bytes from 8.8.8.8: icmp_seq=3 ttl=55 time=11.3 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=45.3 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=55 time=14.8 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=64 time=52.5 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=5 ttl=55 time=11.4 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=64 time=50.3 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=6 ttl=55 time=12.7 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=64 time=41.0 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=7 ttl=55 time=11.0 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=64 time=45.3 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=8 ttl=55 time=12.4 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=64 time=45.5 ms (DUP!)
^C
--- 8.8.8.8 ping statistics ---
8 packets transmitted, 8 received, +8 duplicates, 0% packet loss, time 7038ms
rtt min/avg/max/mdev = 11.025/27.852/52.472/16.255 ms
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup$
```

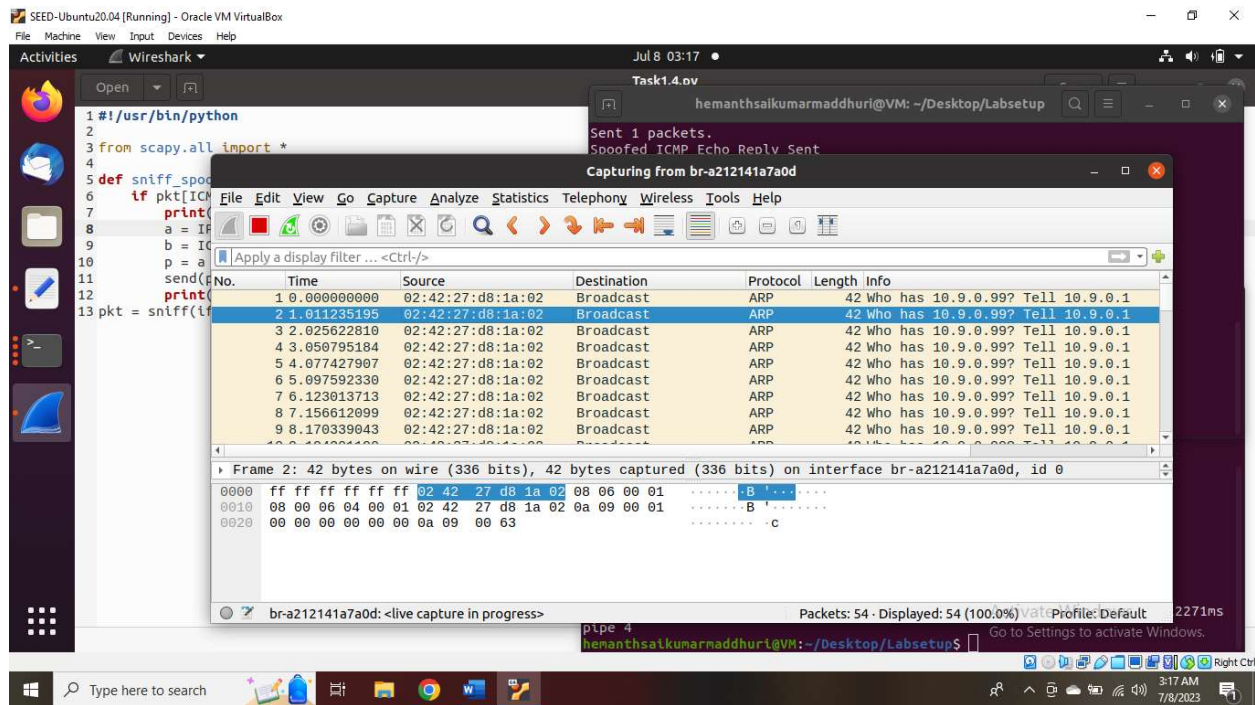
As instructed, we are trying to ping 10.9.0.99 using command “**ping 10.9.0.99**” from one of the terminals and I have observed that when to try to ping 10.9.0.99 it is unreachable as it does not exist.



```
1#!/usr/bin/python
2
3from scapy.all import *
4
5def sniff_spoof(pkt):
6    if pkt[ICMP].type == 8:
7        print("Received ICMP Echo Request")
8        a = IP(src = pkt[IP].dst, dst = pkt[IP].src)
9        b = ICMP(type = 0, id = pkt[ICMP].id, seq = pkt[ICMP].seq)
10       p = a / b / pkt[3].load #pkt is printed as array in hex
11       send(p)
12       print("Spoofed ICMP Echo Reply Sent")
13 pkt = sniff(iface = ['br-a212141a7a0d', 'enp0s3'], filter="icmp")
```

```
Sent 1 packets.
Spoofed ICMP Echo Reply Sent
Received ICMP Echo Request
.
Sent 1 packets.
Spoofed ICMP Echo Reply Sent
Received ICMP Echo Request
.
Sent 1 packets.
Spoofed ICMP Echo Reply Sent
Received ICMP Echo Request
.
Sent 1 packets.
Spoofed ICMP Echo Reply Sent
Received ICMP Echo Request
.
Sent 1 packets.
Spoofed ICMP Echo Reply Sent
Received ICMP Echo Request
.
From 10.9.0.1 icmp_seq=3 Destination Host Unreachable
From 10.9.0.1 icmp_seq=4 Destination Host Unreachable
From 10.9.0.1 icmp_seq=5 Destination Host Unreachable
From 10.9.0.1 icmp_seq=6 Destination Host Unreachable
From 10.9.0.1 icmp_seq=7 Destination Host Unreachable
From 10.9.0.1 icmp_seq=8 Destination Host Unreachable
From 10.9.0.1 icmp_seq=9 Destination Host Unreachable
From 10.9.0.1 icmp_seq=10 Destination Host Unreachable
^C
--- 10.9.0.99 ping statistics ---
13 packets transmitted, 0 received, +10 errors, 100% packet loss, time 12271ms
pipe 4
hemanthsaikumarmaddhuri@VM: ~/Desktop/Labsetup$
```

The Wireshark Screenshot for the same is attached below.



Summary

In this lab I have learnt how to sniff and spoof data to host or target system. I have learnt about a new library of python named Scapy with is useful for network security. Also, I have observed various things like the packet that we print to command line is printed as array, I have tested with giving names and numbers, yet I got the same result while debugging. Overall it was a good lab and understanding the topics like spoofing and sniffing is very important according to my opinion.