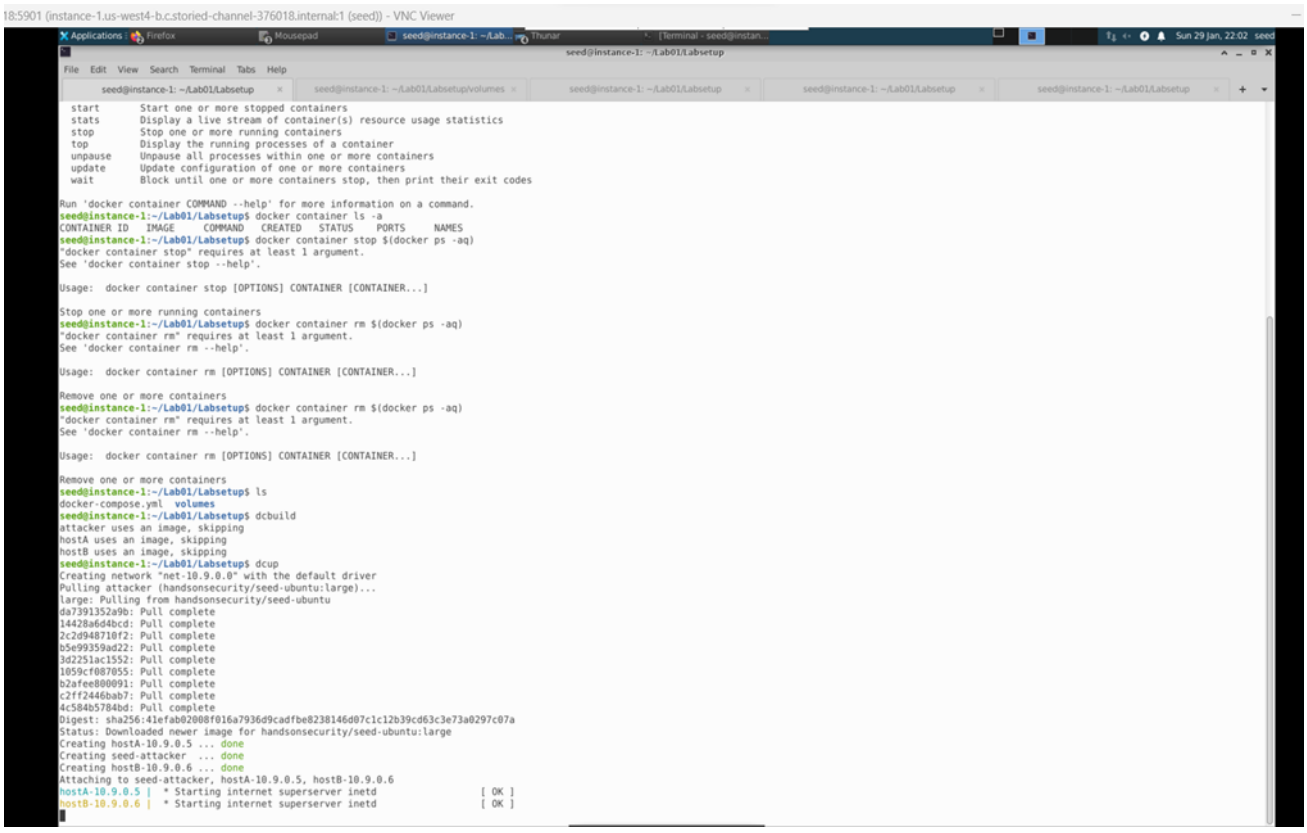
Packet Sniffing & Spoofing Lab

1. Environment Set-up using container.

Created the source and the destination host

Host A= 10.9.0.5

Host B=10.9.0.6



Build the volumes folder in the following directory: - /home/seed/Lab01/Labsetup/volumes

```
seed@instance-1:~/Lab01/Labsetup$ ls
docker-compose.yml  volumes
seed@instance-1:~/Lab01/Labsetup$ dcbuild
attacker uses an image, skipping
hostA uses an image, skipping
hostB uses an image, skipping
seed@instance-1:~/Lab01/Labsetup$ dcup
Creating network "net-10.9.0.0" with the default driver
Pulling attacker (handsonsecurity/seed-ubuntu:large)...
large: Pulling from handsonsecurity/seed-ubuntu
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 hostA-10.9.0.5 ... done
Creating seed-attacker  ... done
Creating hostB-10.9.0.6 ... done
Attaching to seed-attacker, hostA-10.9.0.5, hostB-10.9.0.6
hostA-10.9.0.5 |  * Starting internet superserver inetd                 [ OK ]
hostB-10.9.0.6 |  * Starting internet superserver inetd                 [ OK ]
```

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

In the Network address is 10.9.0.0/24 and in the Network there are following address

Attacker address is 10.9.0.1

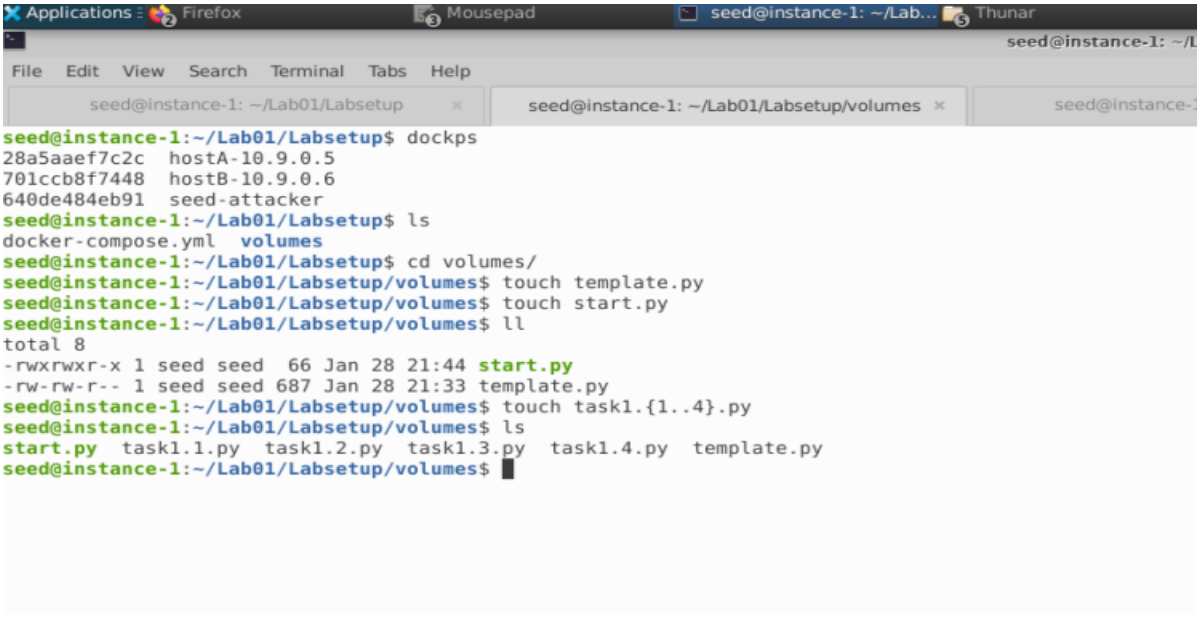
Host A address is 10.9.0.5

Host B address is 10.9.0.6

In the above screenshot we have used the two commands. The two commands are building and starting the Lab environment.

- dcbuild command. It is an alias for docker-compose build.
- dcup command-It is an alias for docker-compose up. Here the dcup command creates a network-10.9.0.0. As seen in the diagram the three containers are running.

Here we are using the dockerps command.

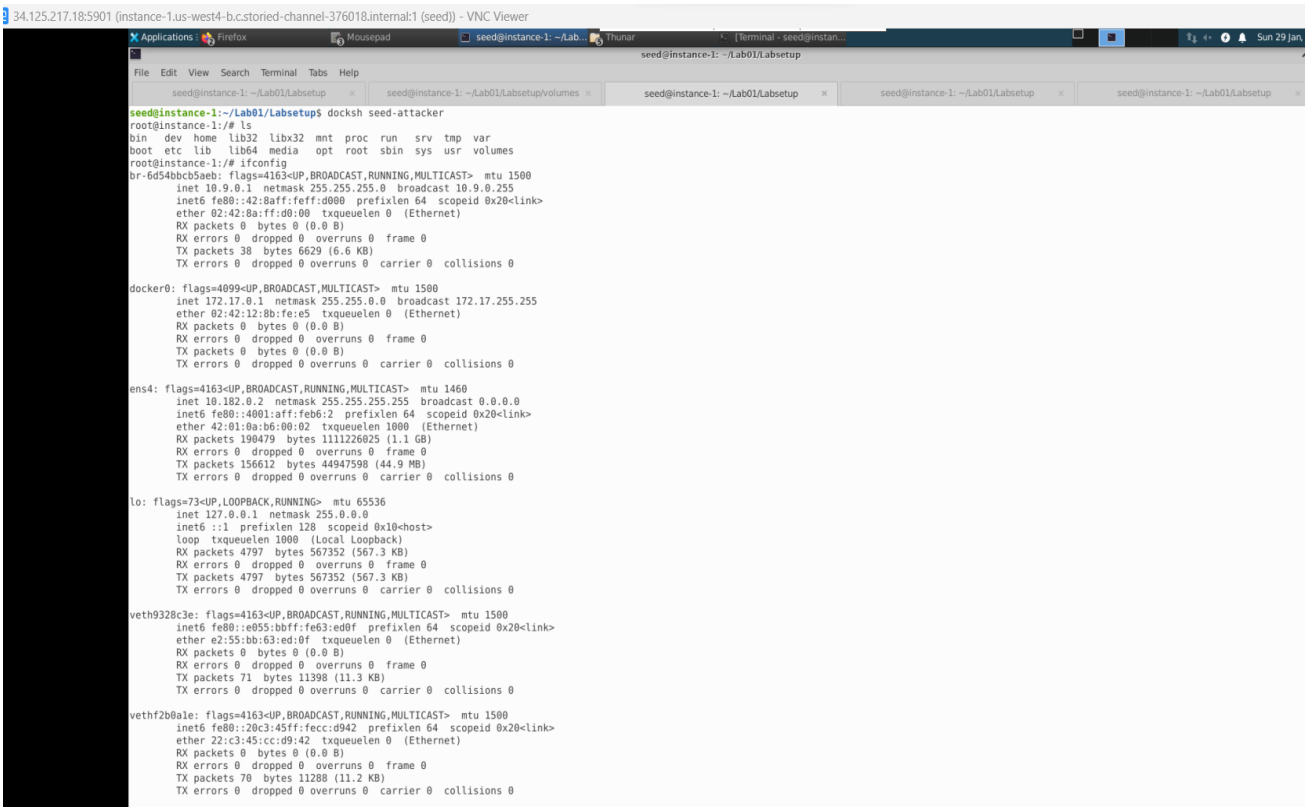


In the above screenshot, we have mentioned the docker ps

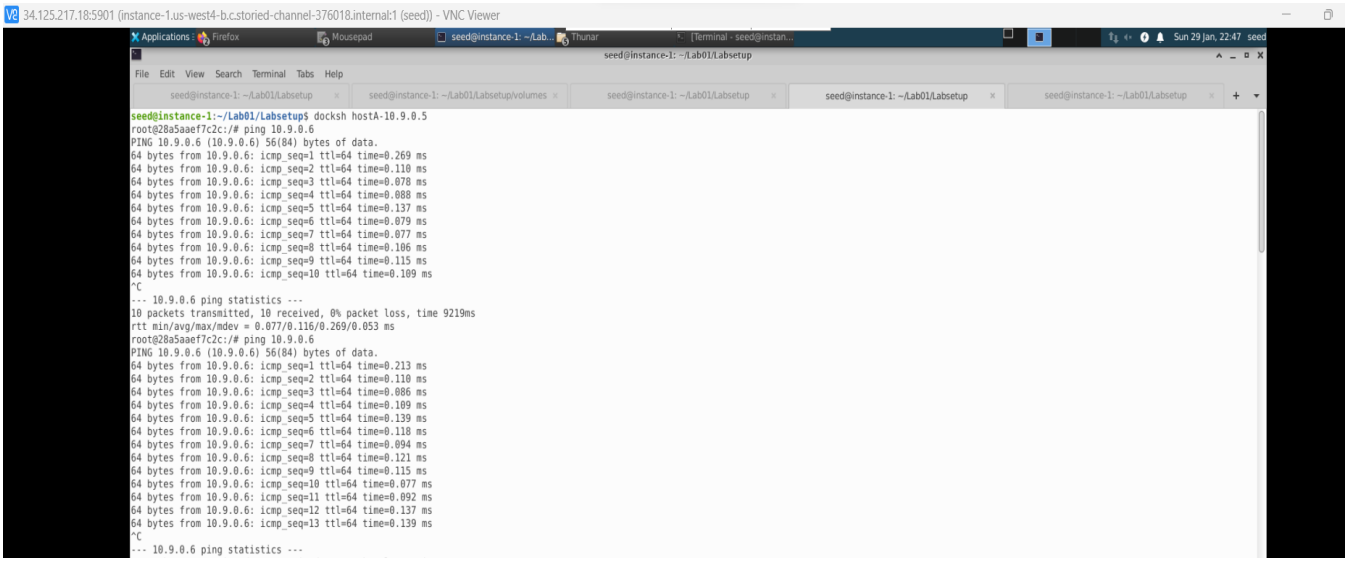
dockerps- This command is available inside the container, is used to see the status of the process. This is like the standard ps command in the Linux environment and is not a docker ps command that we run on the Docker host machine.

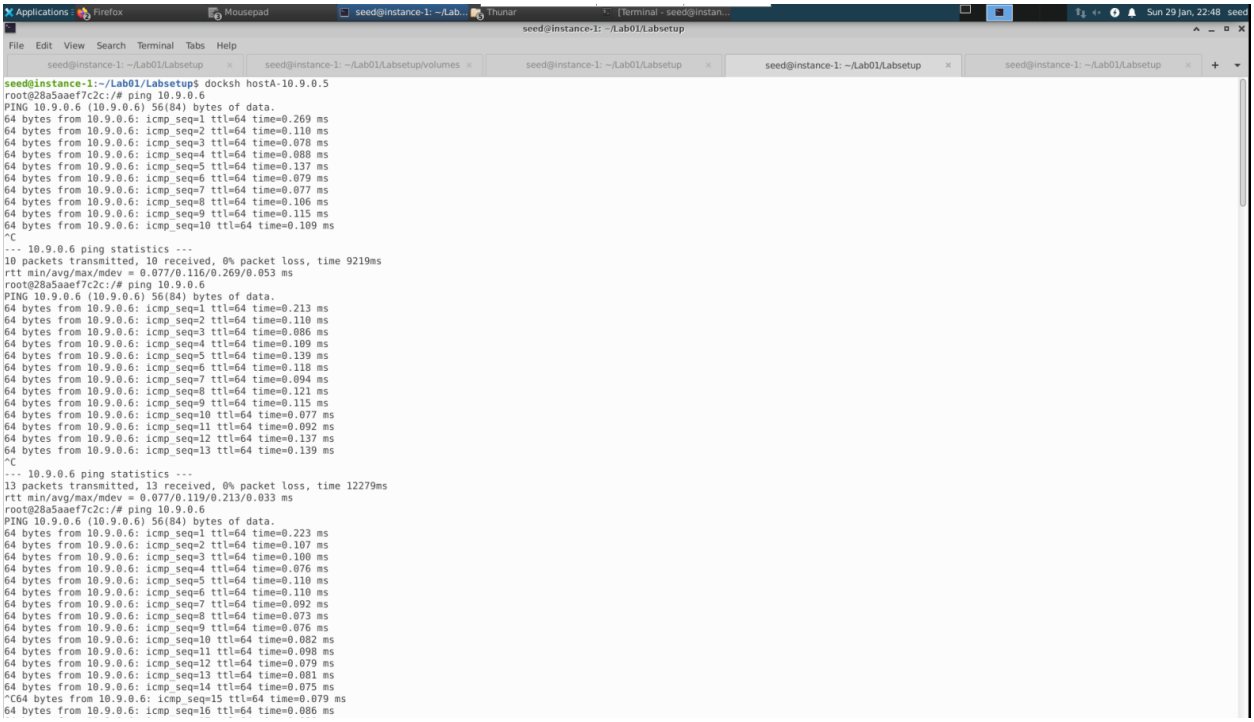
Here we have created the two python files in the volumes folder, namely start.py and template.py.

Additionally, we have created task1.1 to task1.4 in the volumes folder.



Docksh command- specifies the file which contains the name of the image to spin up the dockshcontainer form. Here we are able to see volumes folder by typing the **ls** command.ls command specifies the files or directories.

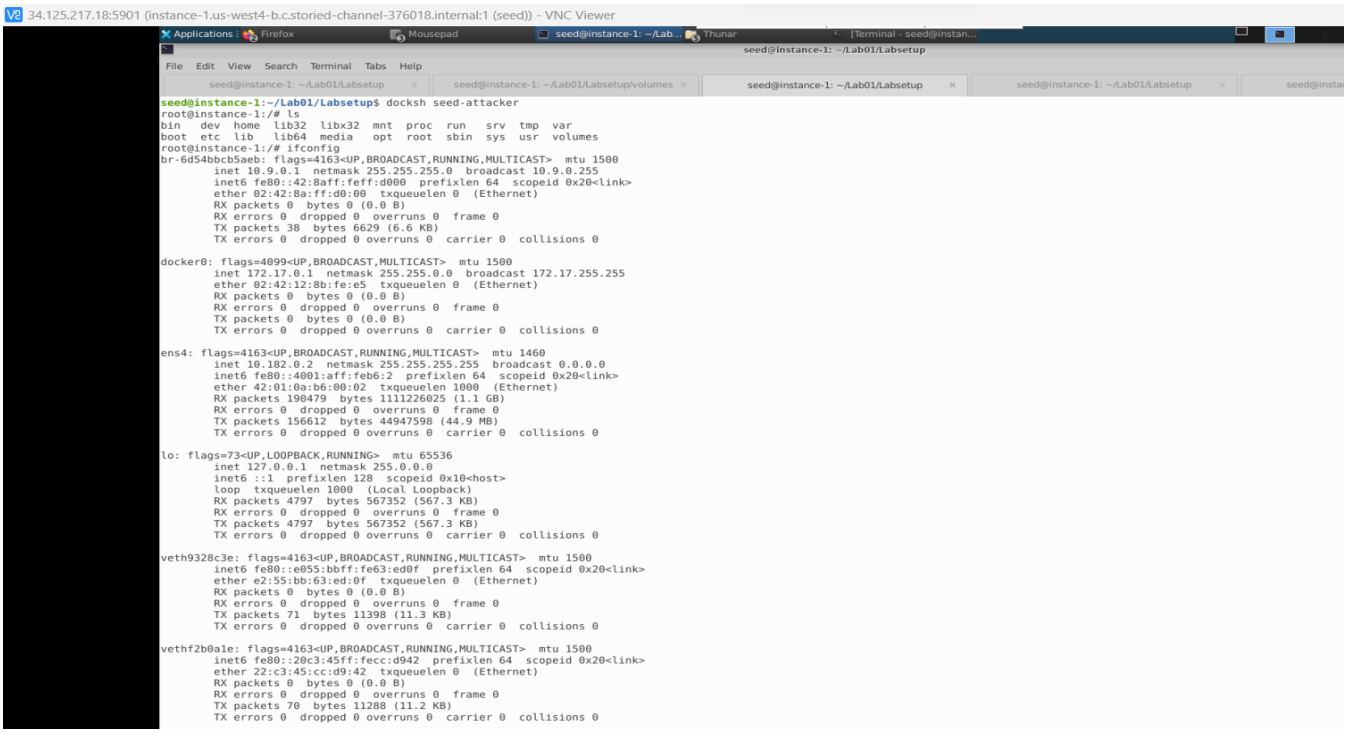




For getting the network interface name we need to type “ifconfig”

The network interface name is specified as below

“br-6d54bbcb5aeb” in the below file



Interface host name mentioned in the above screenshot is **“ens4 ”**

Lab Task Set 1: Using Scapy to Sniff and Spoof packets.

As this is a lab based on scapy libararies we will be working on scapy

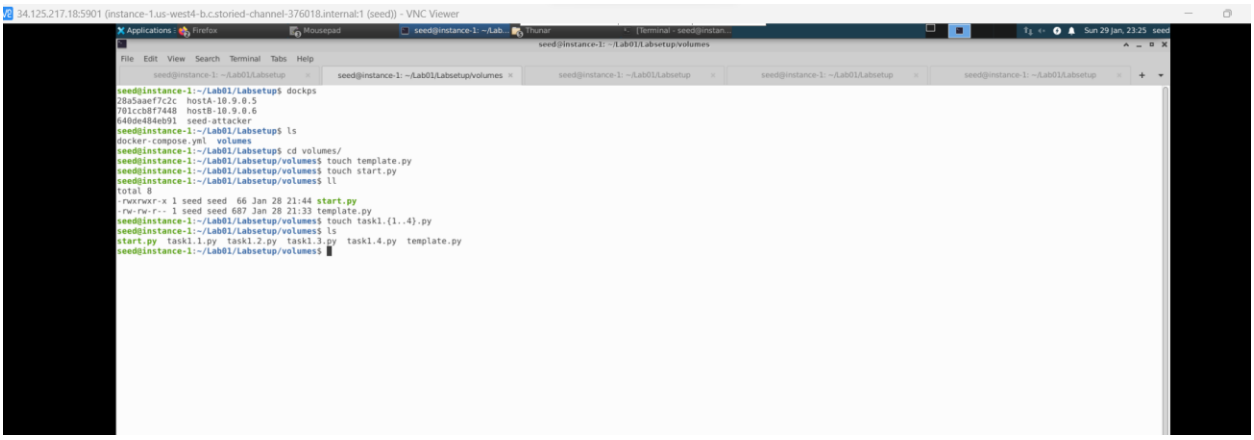
```
root@instance-1:/# scapy
INFO: Can't import matplotlib. Won't be able to plot.
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
INFO: Can't import python-cryptography v1.7+. Disabled WEP decryption/encryption. (Dot11)
INFO: Can't import python-cryptography v1.7+. Disabled IPsec encryption/authentication.
WARNING: IPython not available. Using standard Python shell instead.
AutoCompletion, History are disabled.

                    aSPY//YASa
            apyyyyCY//////////YCa       |
           sY//////YSpcs  scpCY//Pp     | Welcome to Scapy
 ayp ayyyyyyySCP//Pp           syY//C    | Version 2.4.4
 AYAsAYYYYYYYY///Ps              cY//S   |
         pCCCCY//p          cSSps y//Y   | https://github.com/secdev/scapy
         SPPPP///a          pP///AC//Y   |
              A//A            cyP////C   | Have fun!
              p///Ac            sC///a   |
              P////YCpc           A//A   | We are in France, we say Skappee.
       sccccp///pSP///p           p//Y   | OK? Merci.
      sY/////////y  caa           S//P   |              -- Sebastien Chabal
       cayCyayP//Ya              pY/Ya   |
        sY/PsY////YCc           aC//Yp
         sc  sccaCY//PCypaapyCP//YSs
                  spCPY//////YPSps
                       ccaacs

>>> a.IP()
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'a' is not defined
>>> a = IP()
>>> a.show()
###[ 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\
```

In the above screenshot by, by typing the below lines, the default values are shown such as IP version, checksum.

a = IP ()

a.show()

We created two files template.py and start.py.



Additionally, we will also create task1.1 to task1.4 python executable files.

```
seed@instance-1:~/Lab01/Labsetup$ ls
docker-compose.yml  volumes
seed@instance-1:~/Lab01/Labsetup$ cd volumes/
seed@instance-1:~/Lab01/Labsetup/volumes$ touch template.py
seed@instance-1:~/Lab01/Labsetup/volumes$ touch start.py
seed@instance-1:~/Lab01/Labsetup/volumes$ ll
total 8
-rwxrwxr-x 1 seed seed  66 Jan 28 21:44 start.py
-rw-rw-r-- 1 seed seed 687 Jan 28 21:33 template.py
seed@instance-1:~/Lab01/Labsetup/volumes$ touch task1.{1..4}.py
seed@instance-1:~/Lab01/Labsetup/volumes$ ls
start.py  task1.1.py  task1.2.py  task1.3.py  task1.4.py  template.py
seed@instance-1:~/Lab01/Labsetup/volumes$
```

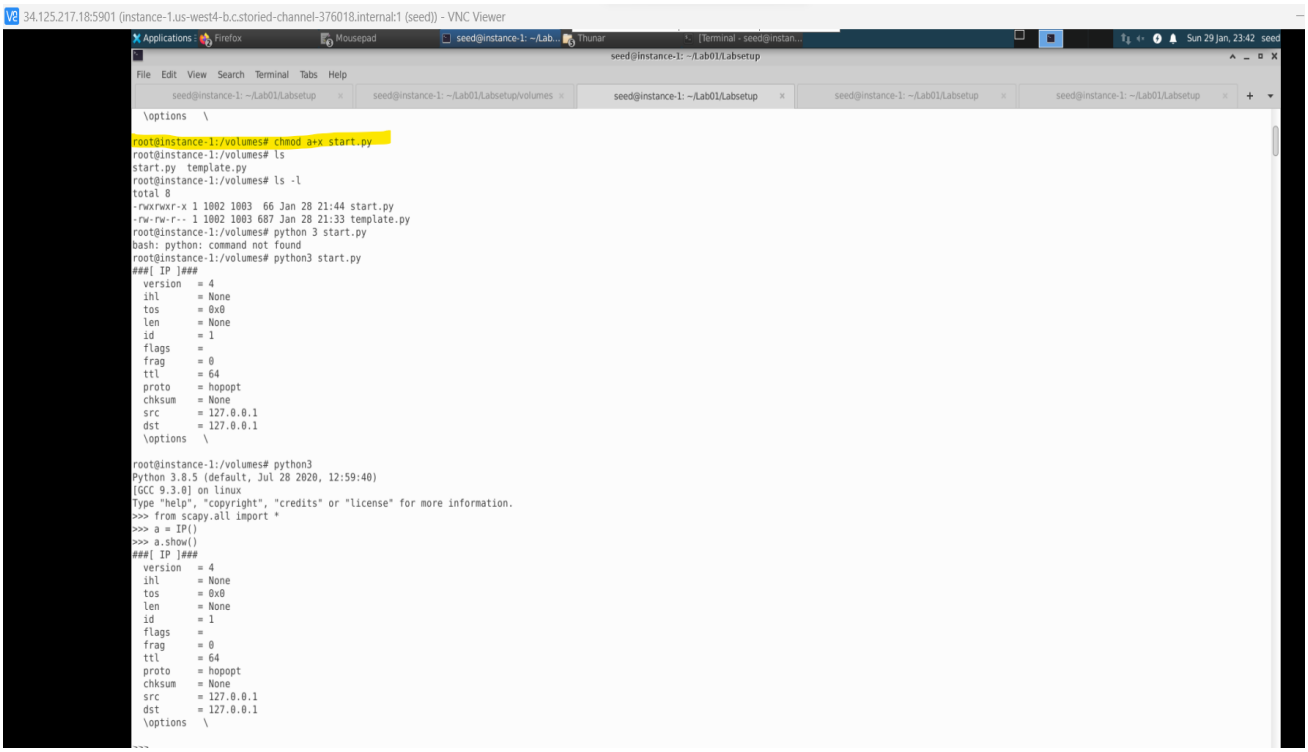
After creating two files we execute the start.py file by writing the command

“python3 start.py” for which we get the following IP specification.

```
root@instance-1:/volumes# python3 start.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   \
```

Now we can make the start.py executable file by typing the command.

“chmod a+x start.py”




```
root@instance-1:/volumes# chmod a+x start.py
root@instance-1:/volumes# ls
start.py  template.py
root@instance-1:/volumes# ls -l
total 8
-rwxrwxr-x 1 1002 1003  66 Jan 28 21:44 start.py
-rw-rw-r-- 1 1002 1003 687 Jan 28 21:33 template.py
root@instance-1:/volumes# python 3 start.py
bash: python: command not found
root@instance-1:/volumes# python3 start.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   \
```


Now we will be importing the scapy.all functions by using the following command.

>from scapy.all import *

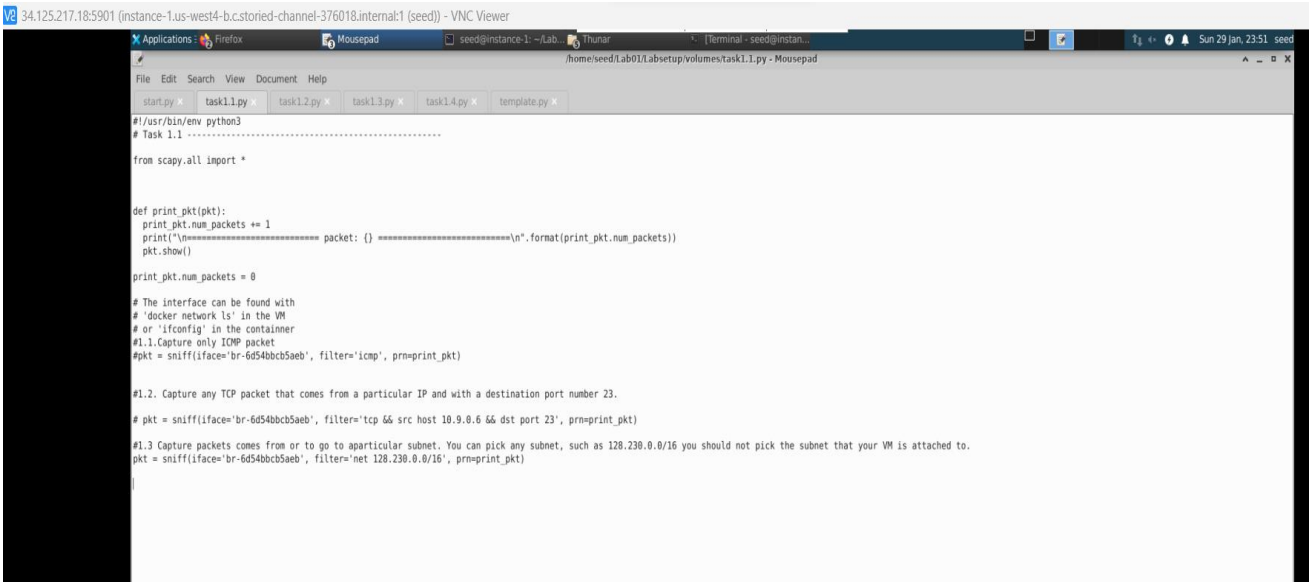
>a = IP ()

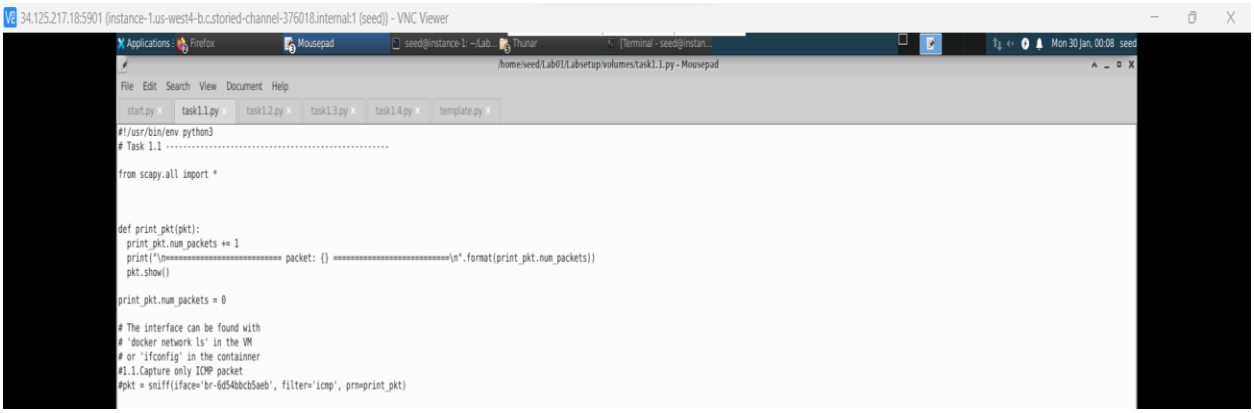
>a.show()

```
root@instance-1:/volumes# python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from scapy.all import *
>>> a = IP()
>>> a.show()
###[ 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   \

>>>
```

Task1.1. Sniffing Packets

Programs: -



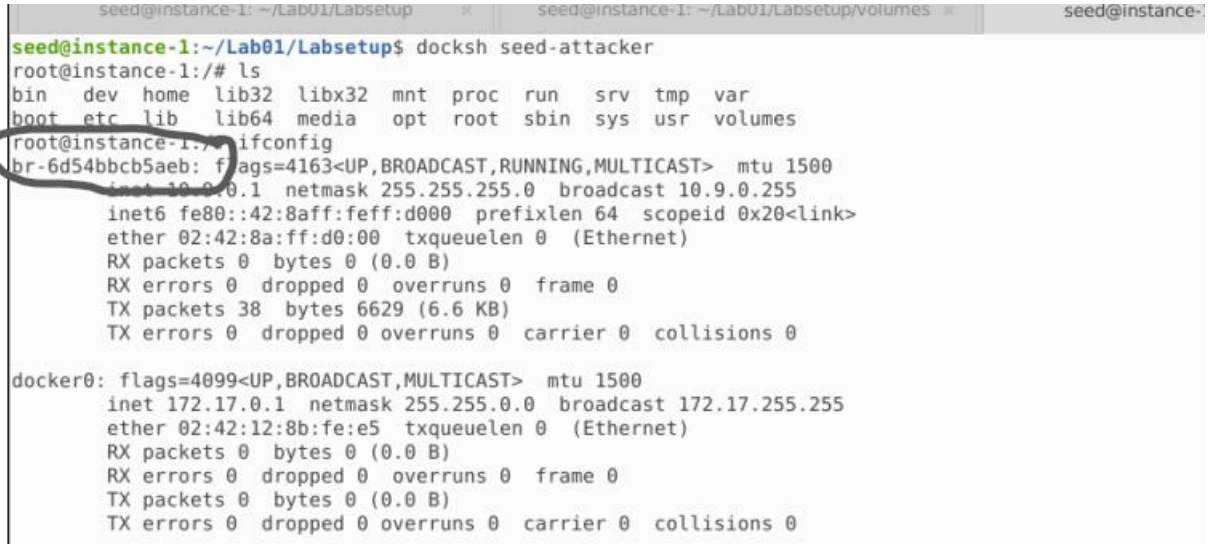


The objective of this task is to learn how to use Scapy to do packet sniffing in Python programs.

We need to change the interface name. Here we can get the attacker interface name by tying the **"ifconfig"** command. Took the interfaces using the command 'ifconfig'in the terminal and mentioned them in the code within the line

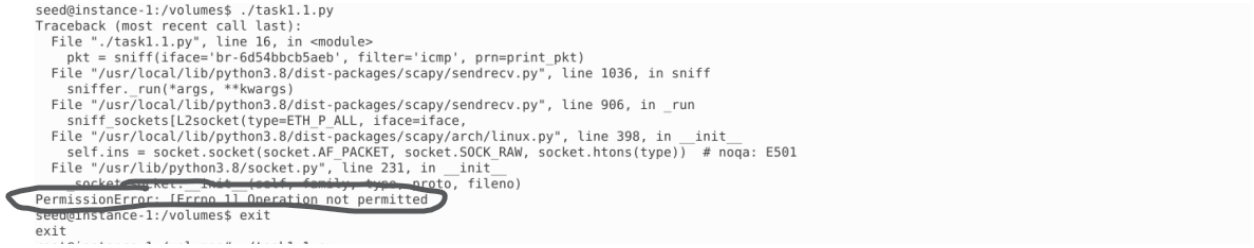
pkt = sniff(iface= br-6d54bbcb5aeb' , filter ='icmp', prn= print_pkt)

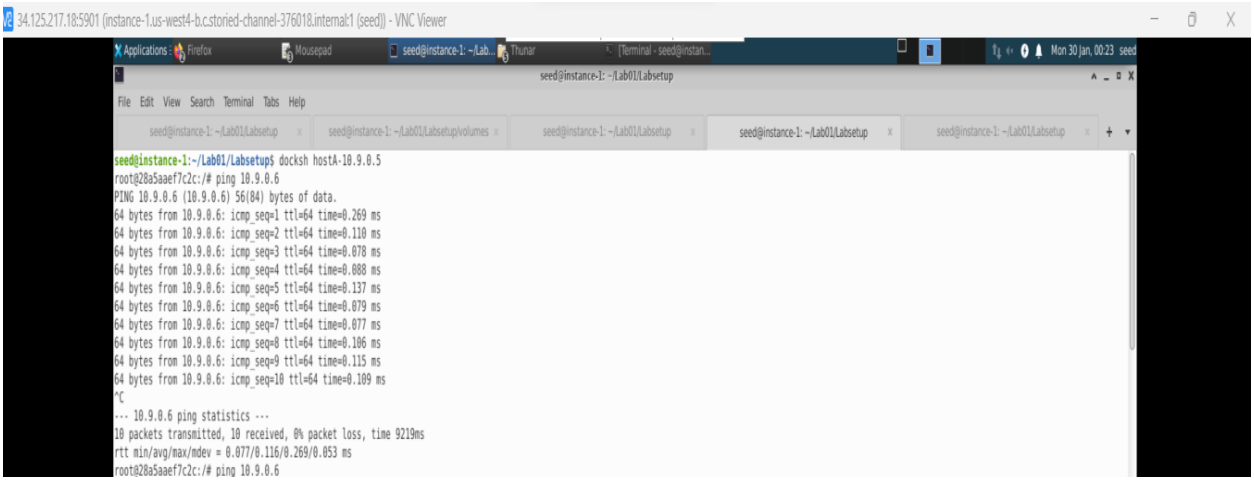
We can see that scapy will show the ICMP packets in the output.



We need to copy the interface name and paste it in the task1.1.py program as

iface = ‘br-6d54bbcb5aeb’





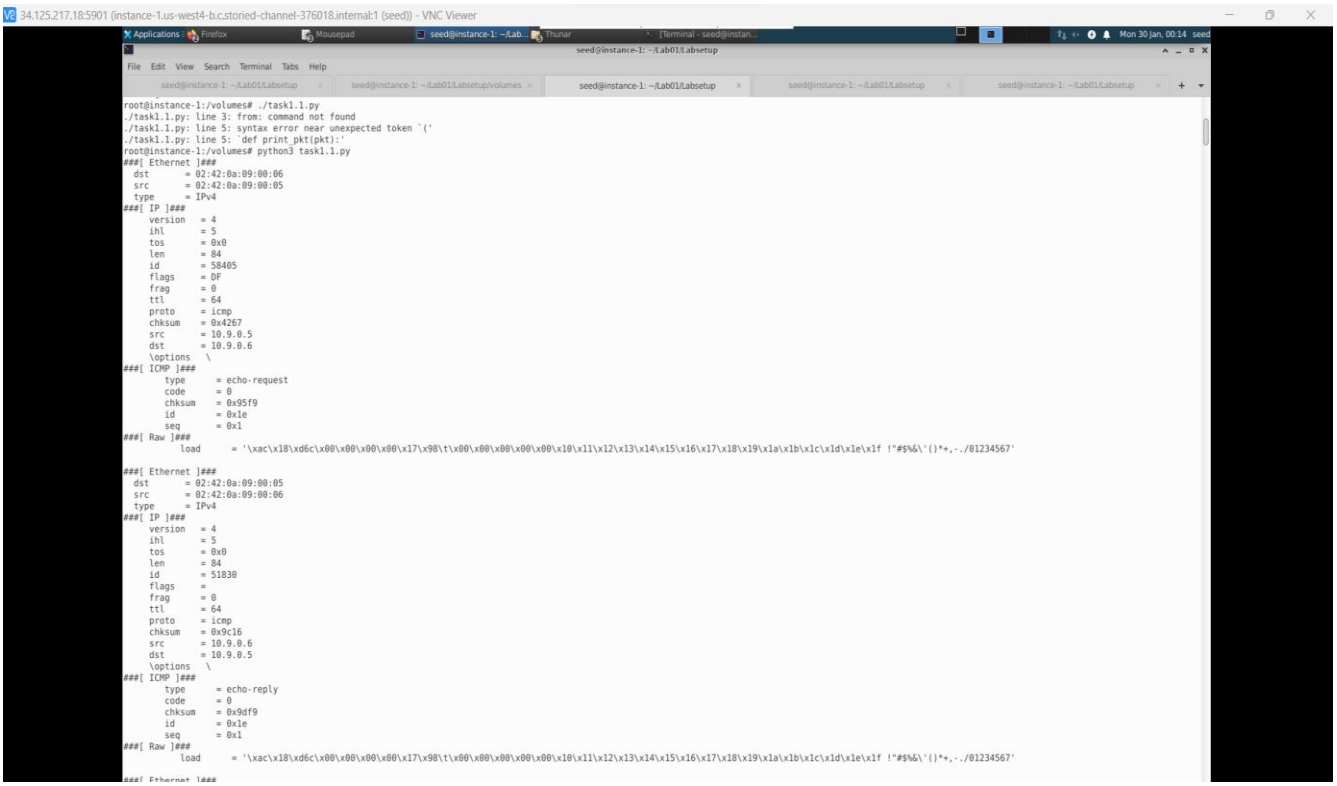
start.py | task1.1.py | task1.2.py | task1.3.py | task1.4.py | template.py

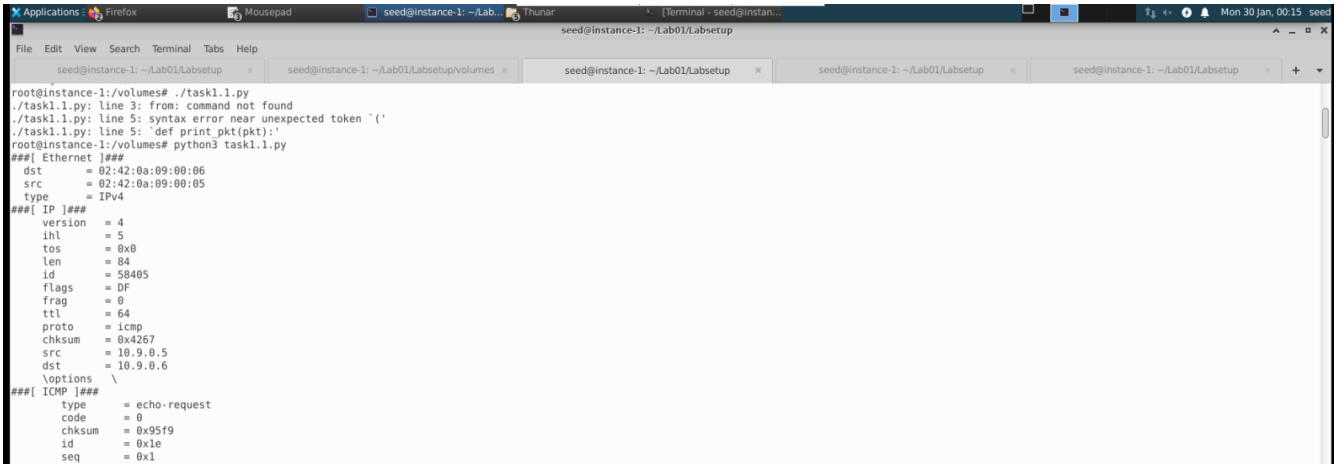
```
#!/usr/bin/env python3
# Task 1.1 ------------------------------------------------------

from scapy.all import *


def print_pkt(pkt):
  print_pkt.num_packets += 1
  print("\n============================ packet: {} ============================\n".format(print_pkt.num_packets))
  pkt.show()

print_pkt.num_packets = 0

# The interface can be found with
# 'docker network ls' in the VM
# or 'ifconfig' in the containner
#1.1.Capture only ICMP packet
#pkt = sniff(iface='br-6d54bbcb5aeb', filter='icmp', prn=print_pkt)
```

In the above program, here each packet for the call back function print_pkt() will be invoked; this function will print out some information about the packet.Here the scapy will show us only ICMP packets.

Once we run the program we can see we are getting the Permission error-“operation not permitted”.

If we even want to sniff packets we need root privileges to whether see there is traffic or not and to capture the relevant packets.


34.125.217.18:5901 (instance-1.us-west4-b.c.storied-channel-376018.internal:1 (seed)) - VNC Viewer
Applications
Firefox
Mousepad
seed@instance-1: ~/Lab...
Thunar
[Terminal - seed@instan...
Mon 30 Jan, 00:14
seed
seed@instance-1: ~/Lab01/Labsetup
File Edit View Search Terminal Tabs Help
seed@instance-1: ~/Lab01/Labsetup
seed@instance-1: ~/Lab01/Labsetup/volumes
seed@instance-1: ~/Lab01/Labsetup
seed@instance-1: ~/Lab01/Labsetup
seed@instance-1: ~/Lab01/Labsetup
root@instance-1:/volumes# ./task1.1.py
./task1.1.py: line 3: from: command not found
./task1.1.py: line 5: syntax error near unexpected token `('
./task1.1.py: line 5: `def print_pkt(pkt):'
root@instance-1:/volumes# python3 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 = 0x0
len = 84
id = 58405
flags = DF
frag = 0
ttl = 64
proto = icmp
chksum = 0x4267
src = 10.9.0.5
dst = 10.9.0.6
\options \
###[ICMP]###
type = echo-request
code = 0
chksum = 0x95f9
id = 0x1e
seq = 0x1
###[Raw]###
load = '\xac\x18\xd6c\x00\x00\x00\x00\x17\x98\t\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
###[Ethernet]###
dst = 02:42:0a:09:00:05
src = 02:42:0a:09:00:06
type = IPv4
###[IP]###
version = 4
ihl = 5
tos = 0x0
len = 84
id = 51830
flags =
frag = 0
ttl = 64
proto = icmp
chksum = 0x9c16
src = 10.9.0.6
dst = 10.9.0.5
\options \
###[ICMP]###
type = echo-reply
code = 0
chksum = 0x9df9
id = 0x1e
seq = 0x1
###[Raw]###
load = '\xac\x18\xd6c\x00\x00\x00\x00\x17\x98\t\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
###[Ethernet]###


Applications
Firefox
Mousepad
seed@instance-1: ~/Lab...
Thunar
[Terminal - seed@instan...
Mon 30 Jan, 00:15
seed
seed@instance-1: ~/Lab01/Labsetup
File Edit View Search Terminal Tabs Help
seed@instance-1: ~/Lab01/Labsetup
seed@instance-1: ~/Lab01/Labsetup/volumes
seed@instance-1: ~/Lab01/Labsetup
seed@instance-1: ~/Lab01/Labsetup
seed@instance-1: ~/Lab01/Labsetup
root@instance-1:/volumes# ./task1.1.py
./task1.1.py: line 3: from: command not found
./task1.1.py: line 5: syntax error near unexpected token `('
./task1.1.py: line 5: `def print_pkt(pkt):'
root@instance-1:/volumes# python3 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 = 0x0
len = 84
id = 58405
flags = DF
frag = 0
ttl = 64
proto = icmp
chksum = 0x4267
src = 10.9.0.5
dst = 10.9.0.6
\options \
###[ICMP]###
type = echo-request
code = 0
chksum = 0x95f9
id = 0x1e
seq = 0x1

Task1.1 B

- **Capture only the ICMP Packet**

Explanation

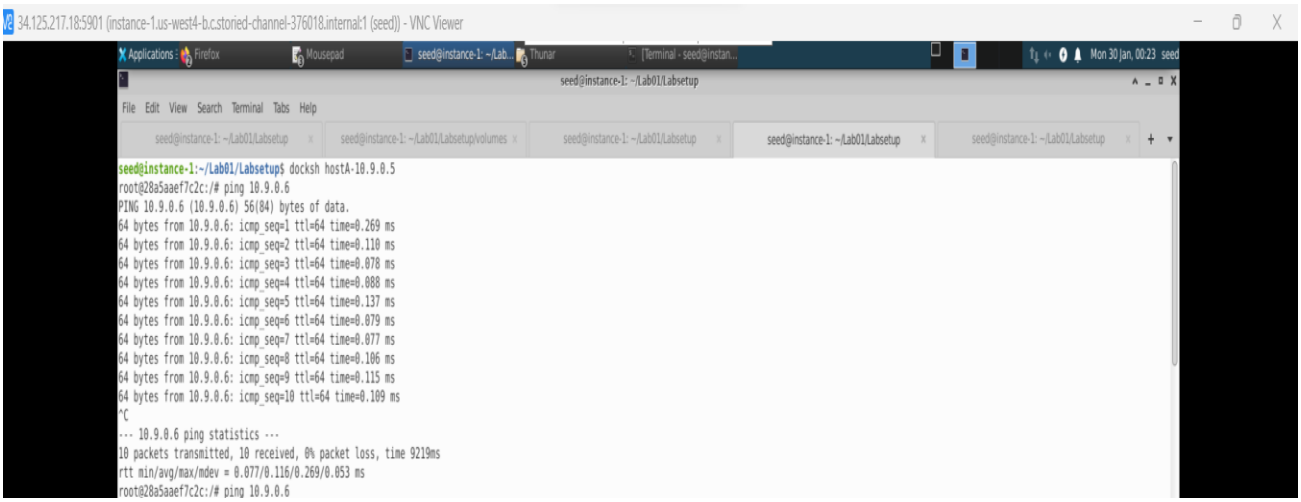
We need to ping host A to host B. We observed that 10 packets were transmitted and 10 received.We can see the ICMP packets and the raw packets.

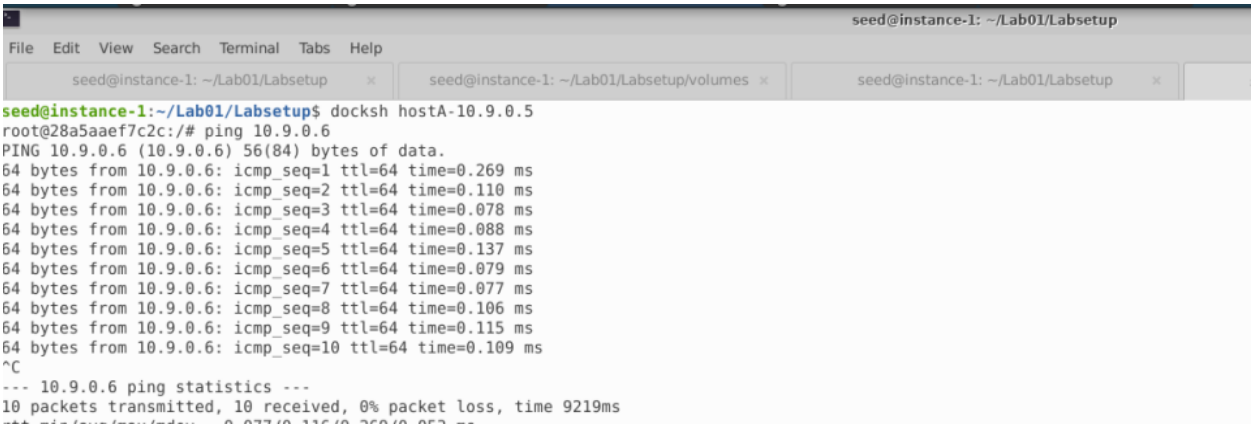
Used the ping command with host B for ICMP echo request and reply packets

Program-task1.1.py

start.py | task1.1.py | task1.2.py | task1.3.py | task1.4.py | template.py

```
#!/usr/bin/env python3
# Task 1.1 ------------------------------------------------------

from scapy.all import *


def print_pkt(pkt):
  print_pkt.num_packets += 1
  print("\n=========================== packet: {} ===========================\n".format(print_pkt.num_packets))
  pkt.show()

print_pkt.num_packets = 0

# The interface can be found with
# 'docker network ls' in the VM
# or 'ifconfig' in the containner
#1.1.Capture only ICMP packet
#pkt = sniff(iface='br-6d54bbcb5aeb', filter='icmp', prn=print_pkt)
```



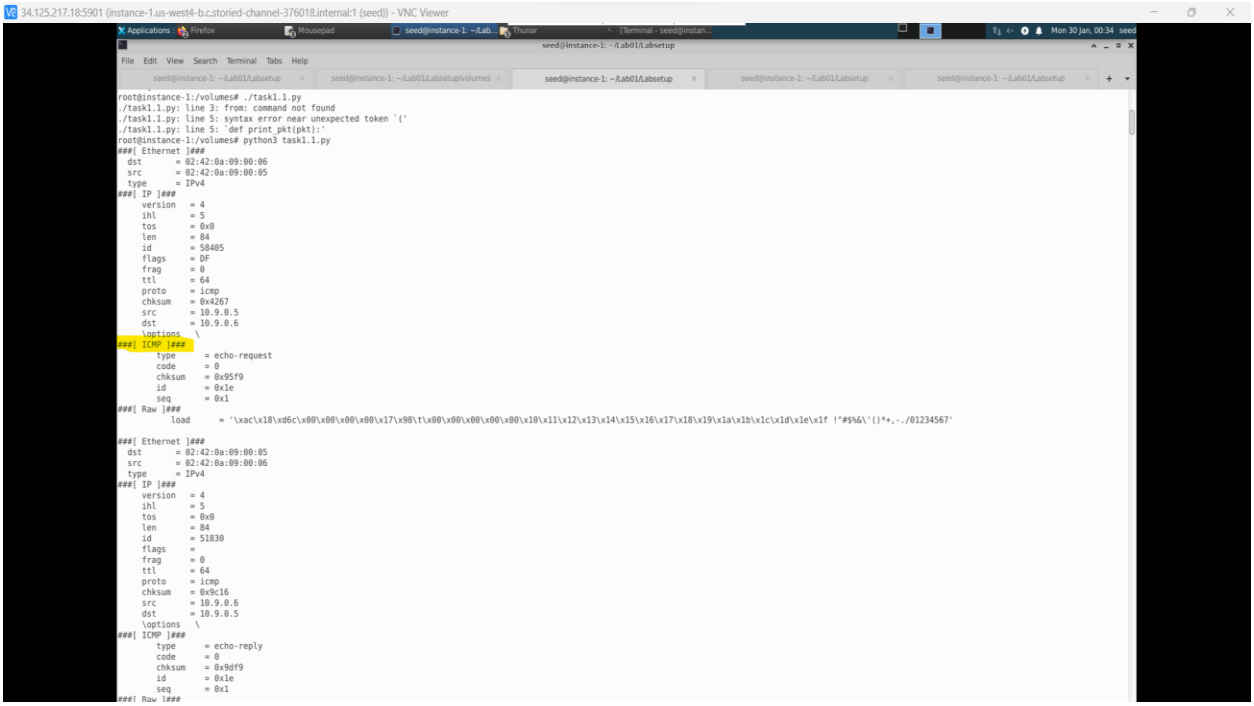


Packet wise transmission

1. Made use of filter like the last code which accepts the syntax of Berkeley Packet Filter.The output is printed with IP and ICMP type packets.

2. Ping is sent to host B. This generates an ICMP echo request packet. And from the opposite side we get means the program will receive an echo reply and print out the response.

```
root@instance-1:/volumes# ./task1.1.py

########## packet: 1 ##########

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 36720
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x971c
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xdfd
        id        = 0x21
        seq       = 0xc
###[ Raw ]###
           load      = '\x00 \xd6c\x00\x00\x00\x00M\x7f\x07\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'

########## packet: 2 ##########

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 5011
     flags     =
     frag      = 0
     ttl       = 64
```



34.125.217.18:5901 (instance-1.us-west4-b.c.storied-channel-376018.internal:1 (seed)) - VNC Viewer

Applications · Firefox · Mousepad · seed@instance-1: ~/Lab... · Thunar · [Terminal - seed@instan... · Mon 30 Jan, 00:34 seed

seed@instance-1: ~/Lab01/Labsetup

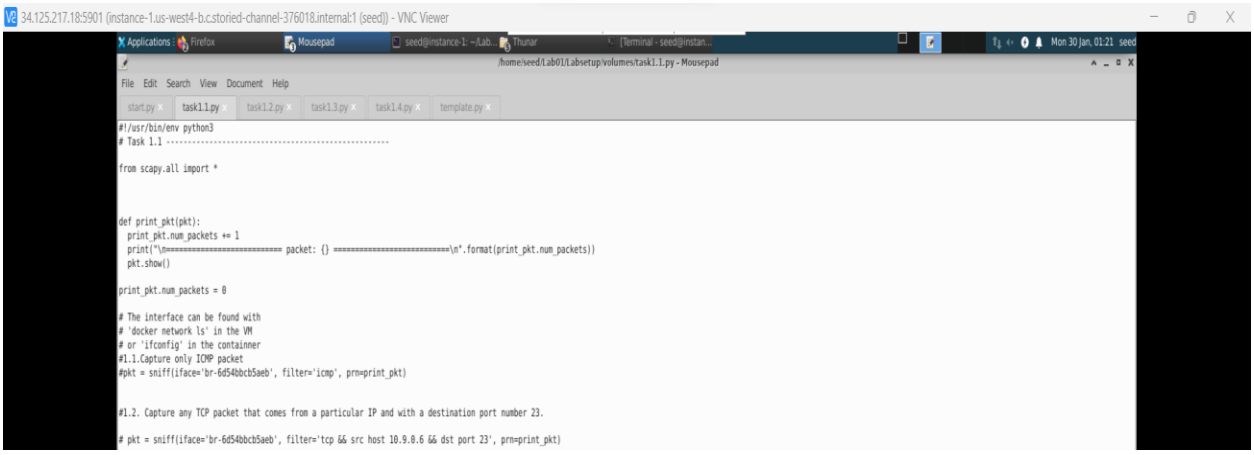
File Edit View Search Terminal Tabs Help

seed@instance-1: ~/Lab01/Labsetup | seed@instance-1: ~/Lab01/Labsetup/volumes | seed@instance-1: ~/Lab01/Labsetup | seed@instance-1: ~/Lab01/Labsetup | seed@instance-1: ~/Lab01/Labsetup

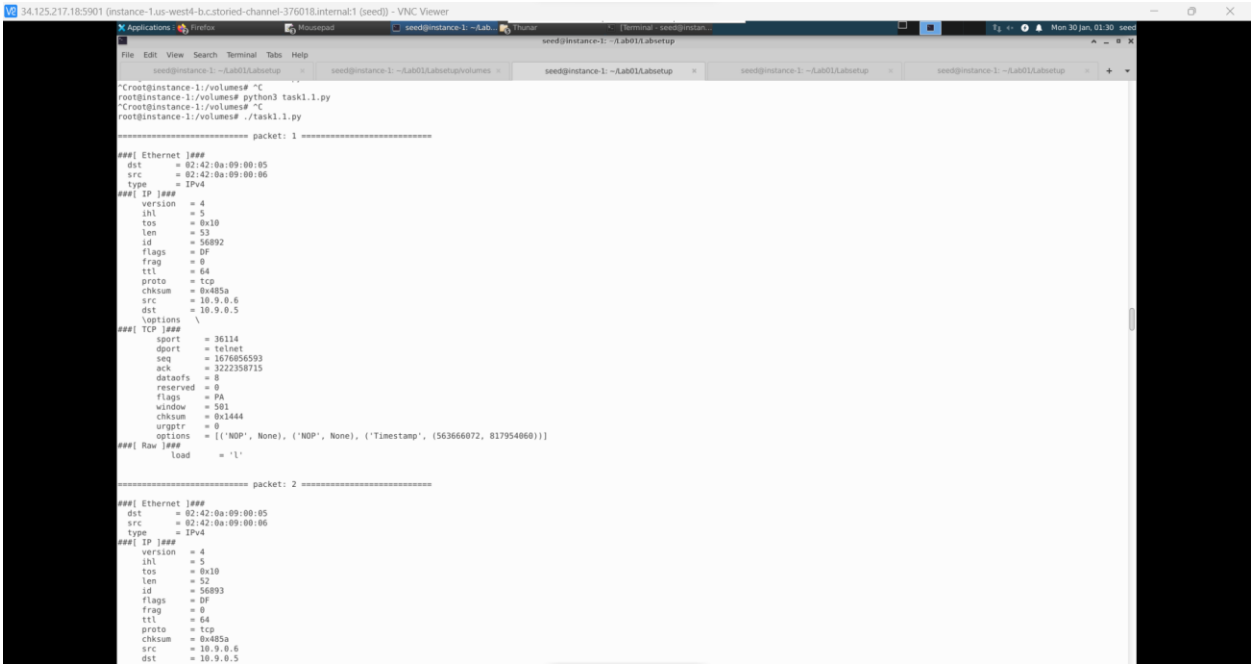
```
root@instance-1:/volumes# ./task1.1.py
./task1.1.py: line 3: from: command not found
./task1.1.py: line 5: syntax error near unexpected token `('
./task1.1.py: line 5: `def print_pkt(pkt):'
root@instance-1:/volumes# python3 task1.1.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 58405
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x4267
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x95f9
        id        = 0x1e
        seq       = 0x1
###[ Raw ]###
           load      = '\xac\x18\xd6c\x00\x00\x00\x00\x17\x98\t\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'

###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 51830
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x9c16
     src       = 10.9.0.6
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0x9df9
        id        = 0x1e
        seq       = 0x1
###[ Raw ]###
```

Task1.1 B

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



Output: -



Explanation:

1. Filtered tcp port 23 and src host 10.9.0.5.Syntax used is from BPF syntax website.

2.My destination and source address are below

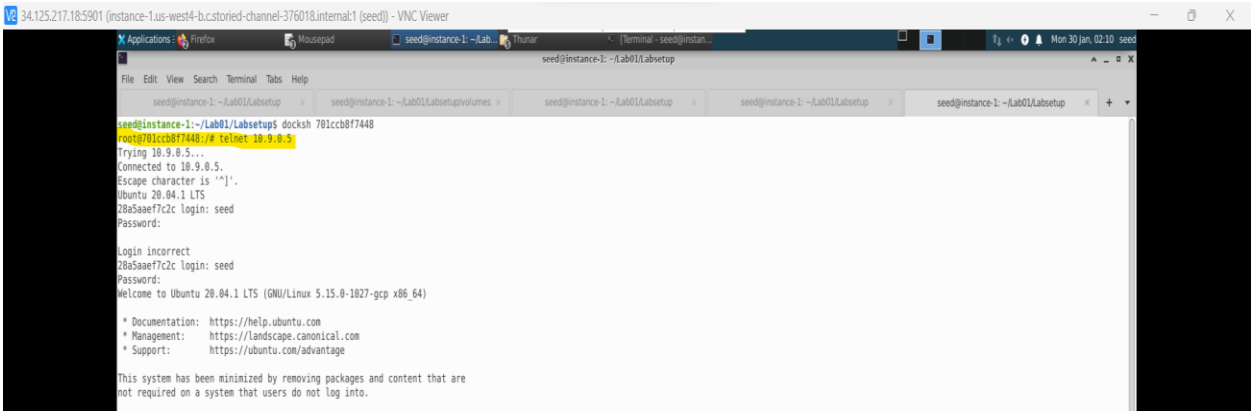
host A-10.9.0.5, host B-10.9.0.6

3. Here in the below screenshot I have done the telnet from host A from host B

4.This way I have send the telnet command to my another machine and the program “tcp_task1.1.py’ sniffed the TCP packets

5.We used telnet here because the protocol is used for establishing a connection to TCP port number 23.


```
root@instance-1:/volumes# ./task1.1.py
TCP Packet==========
    Source: 10.9.0.6
    Destination: 10.9.0.5
    TCP Source port: 36114
    TCP Destination port: 23
TCP Packet==========
    Source: 10.9.0.6
    Destination: 10.9.0.5
    TCP Source port: 36114
    TCP Destination port: 23
TCP Packet==========
    Source: 10.9.0.6
    Destination: 10.9.0.5
    TCP Source port: 36114
    TCP Destination port: 23
TCP Packet==========
```



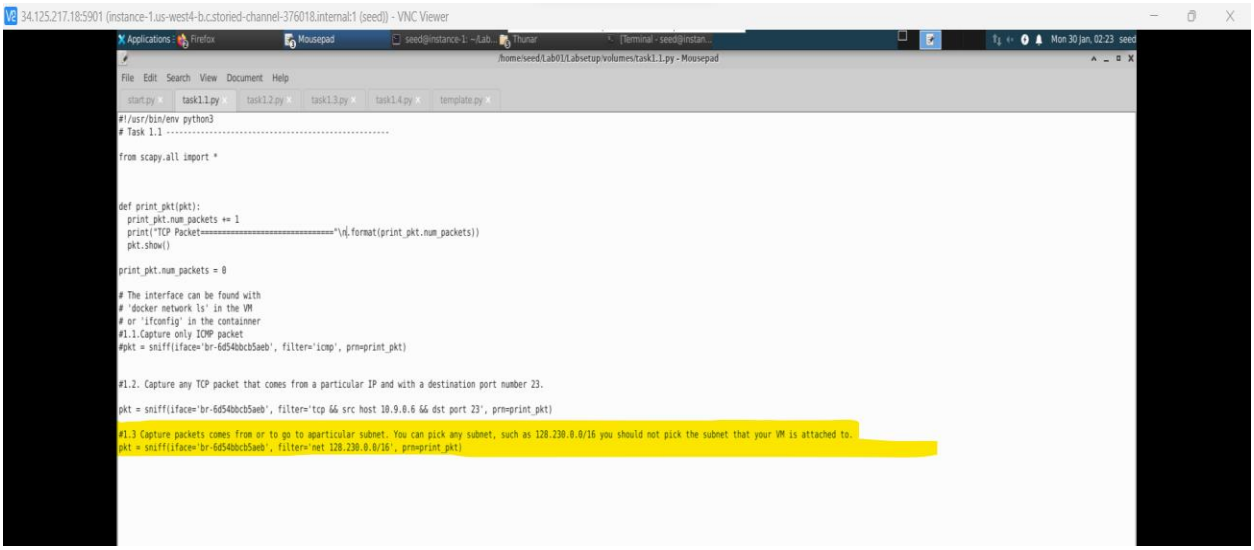
Task 1.1. B

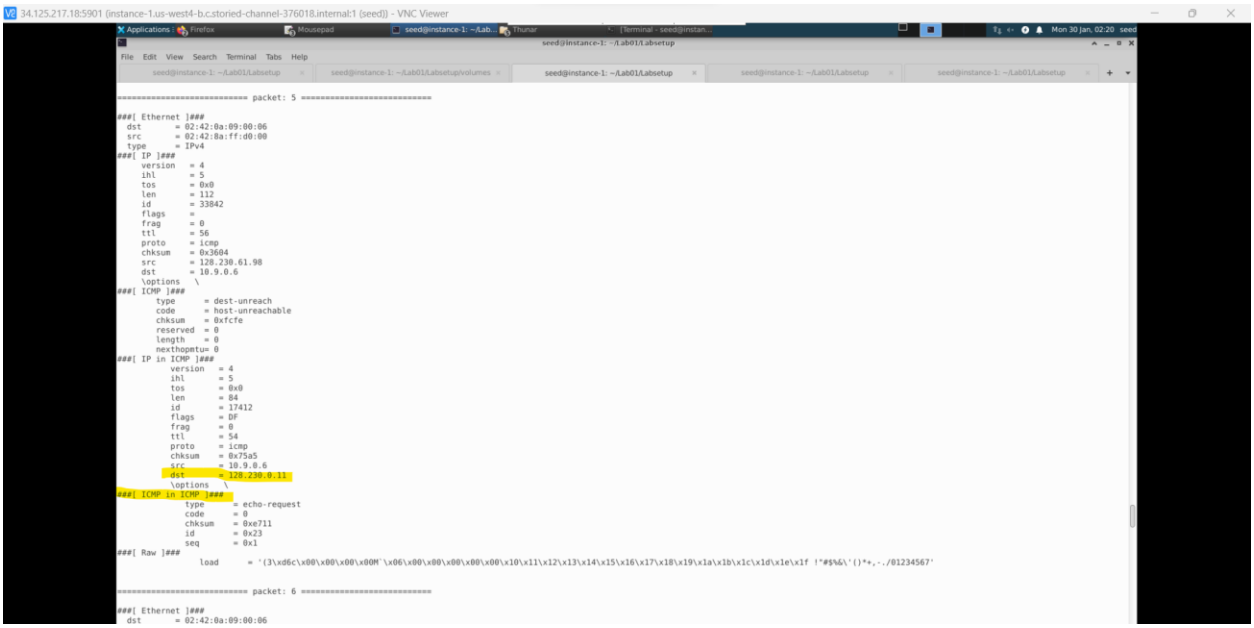
- Capture packets comes from or to go to a particular subnet. You can pick any subnet, such as 128.230.0.0/16; you should not pick the subnet that your VM is attached to.

Explanation:

1.A normal packet was sent to a particular IP address, or a subnet and the program sniffed the packets that were sent from source '10.9.0.6' to the destination IP from the other IP address.

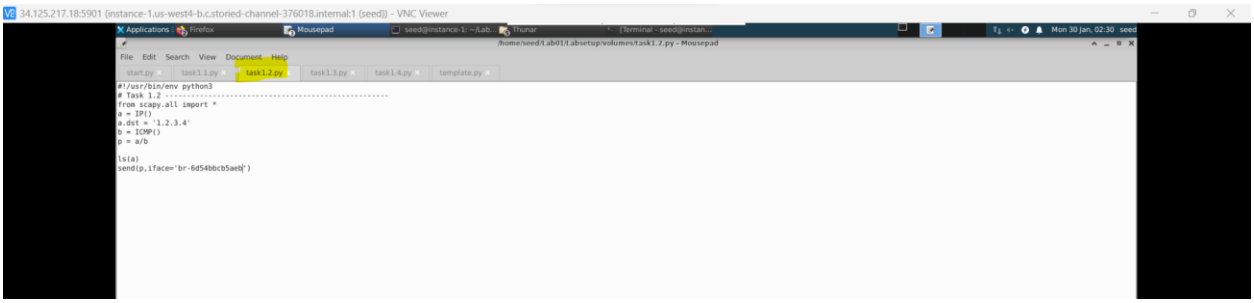
2. Here I have taken the destination as 128.230.0.0/16 subnet's IP address.



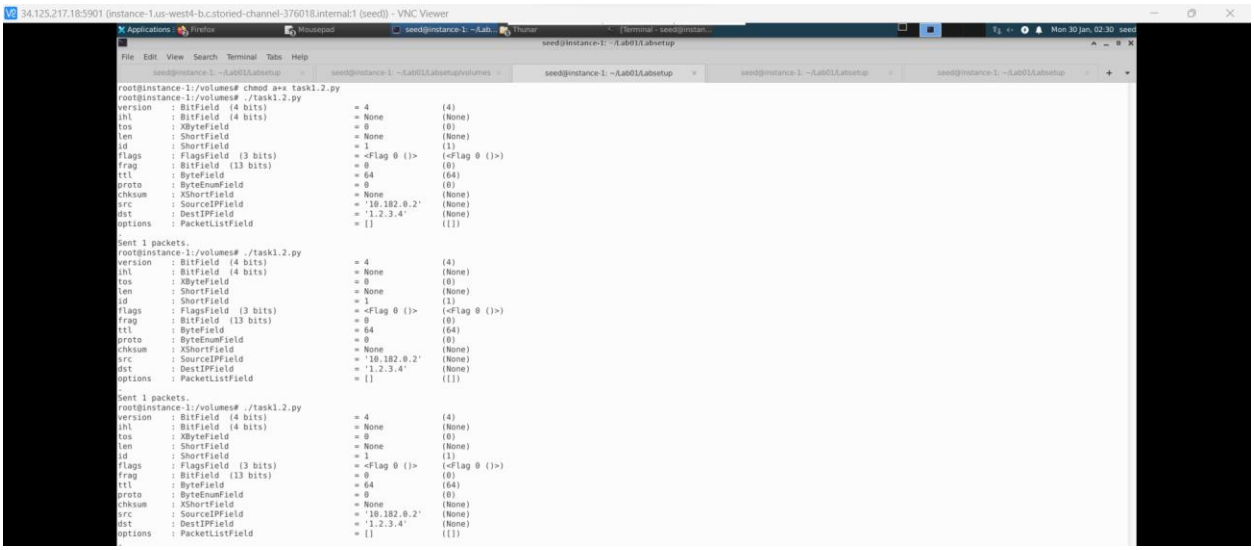


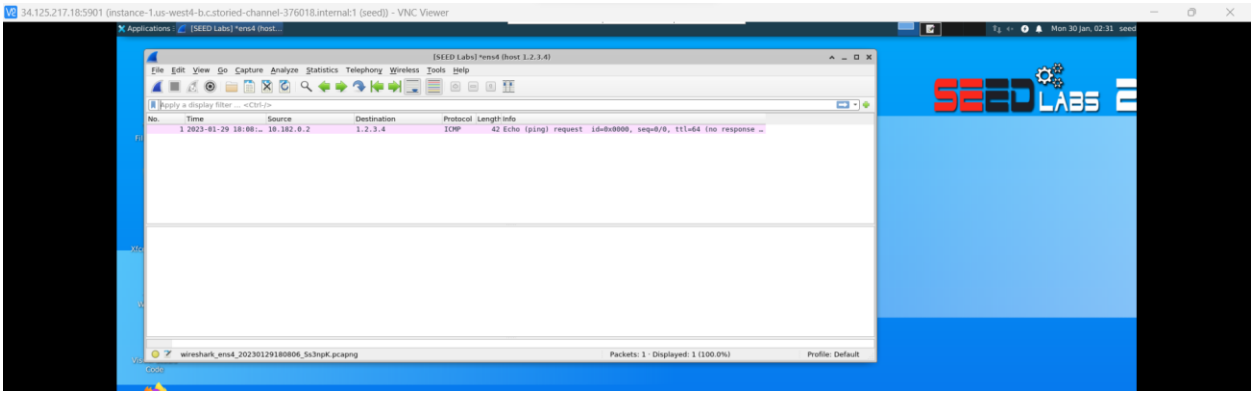
Task 1.2: Spoofing ICMP Packets

Program- task1.2.py



Output:-





Explanation: -

1.Used other IP address with IP as destination-‘1.2.3.4’ and sent an ICMP packet using a random IP source ‘10.182.0.2’

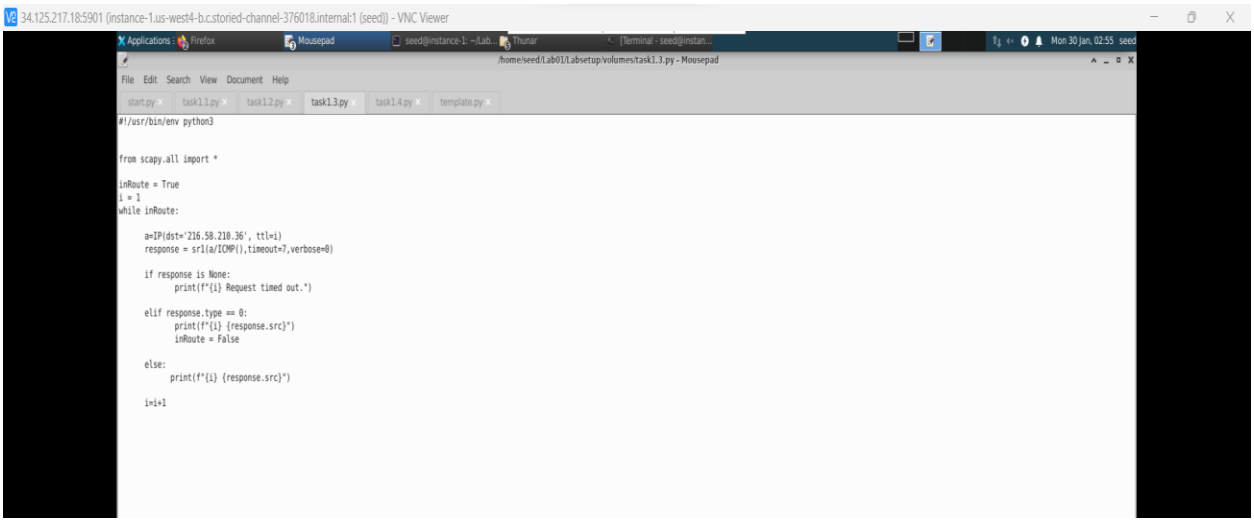
2. Spoofed ICMP echo request packet, and sent it to another machine on the same subnet.And then we used Wireshark to check whether our request will be accepted by the receiver.

3.As here Scapy comes into picture, the source ip was overwritten and the original IP was used,10.182.0.2 and sent the packet to destination 1.2.3.4.

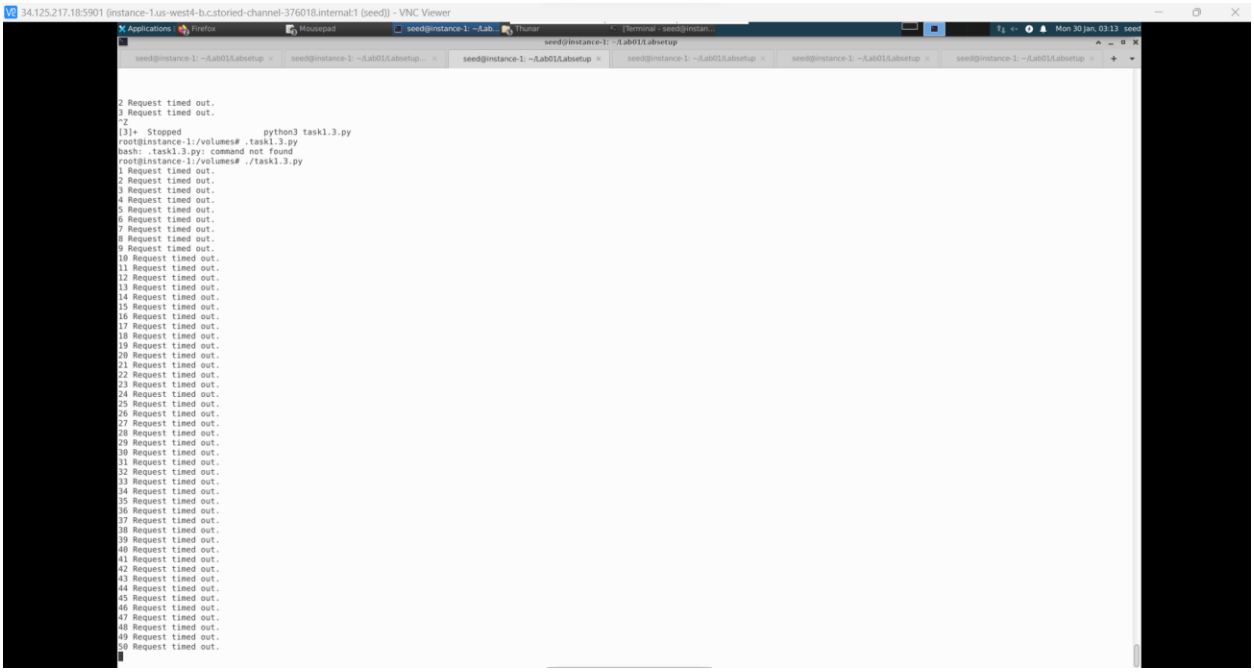
4.Packet was received and an echo reply back was sent to 10.182.0.2

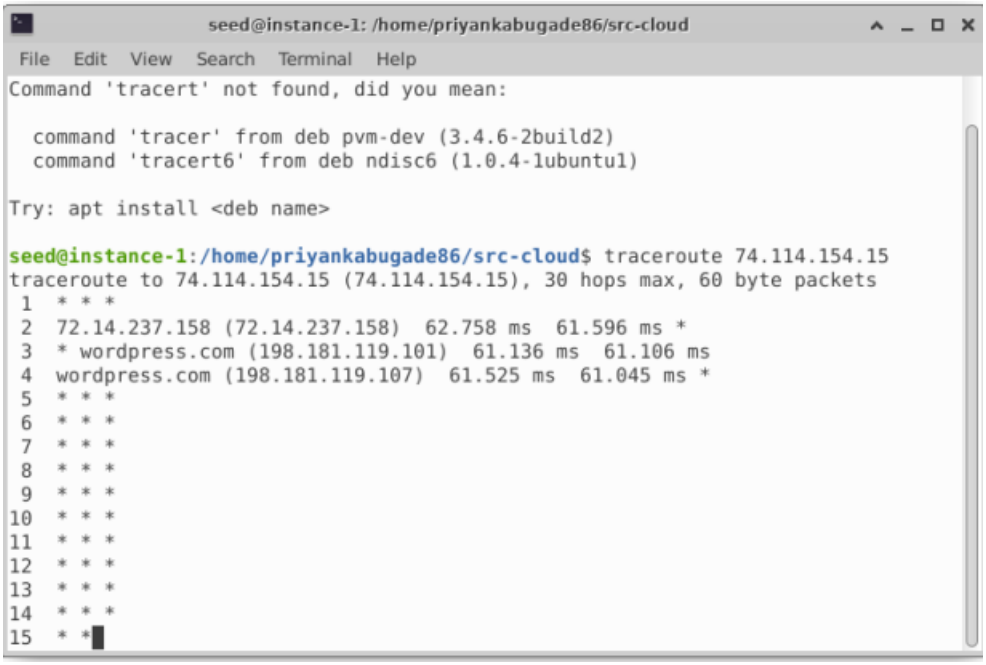
Task 1.3: Traceroute

Program-task1.3.py


34.125.217.18:5901 (instance-1.us-west4-b.c.storied-channel-376018.internal:1 (seed)) - VNC Viewer
Applications
Firefox
Mousepad
Thunar
Mon 30 Jan, 02:55
/home/seed/Lab01/Labsetup/volumes/task1.3.py - Mousepad
File Edit Search View Document Help
start.py
task1.1.py
task1.2.py
task1.3.py
task1.4.py
template.py
#!/usr/bin/env python3
from scapy.all import *
inRoute = True
i = 1
while inRoute:
a=IP(dst='216.58.210.36', ttl=i)
response = sr1(a/ICMP(),timeout=7,verbose=0)
if response is None:
print(f"{i} Request timed out.")
elif response.type == 0:
print(f"{i} {response.src}")
inRoute = False
else:
print(f"{i} {response.src}")
i=i+1


root@instance-1:/volumes# ./task1.3.py
1 Request timed out.
2 Request timed out.
3 Request timed out.
4 Request timed out.
5 Request timed out.
6 Request timed out.
7 Request timed out.
8 Request timed out.
9 Request timed out.
10 Request timed out.
11 Request timed out.


34.125.217.18:5901 (instance-1.us-west4-b.c.storied-channel-376018.internal:1 (seed)) - VNC Viewer
Applications
Firefox
Mousepad
Thunar
Mon 30 Jan, 03:13
seed@instance-1: ~/Lab01/Labsetup
2 Request timed out.
3 Request timed out.
^Z
[3]+ Stopped python3 task1.3.py
root@instance-1:/volumes# .task1.3.py
bash: .task1.3.py: command not found
root@instance-1:/volumes# ./task1.3.py
1 Request timed out.
2 Request timed out.
3 Request timed out.
4 Request timed out.
5 Request timed out.
6 Request timed out.
7 Request timed out.
8 Request timed out.
9 Request timed out.
10 Request timed out.
11 Request timed out.
12 Request timed out.
13 Request timed out.
14 Request timed out.
15 Request timed out.
16 Request timed out.
17 Request timed out.
18 Request timed out.
19 Request timed out.
20 Request timed out.
21 Request timed out.
22 Request timed out.
23 Request timed out.
24 Request timed out.
25 Request timed out.
26 Request timed out.
27 Request timed out.
28 Request timed out.
29 Request timed out.
30 Request timed out.
31 Request timed out.
32 Request timed out.
33 Request timed out.
34 Request timed out.
35 Request timed out.
36 Request timed out.
37 Request timed out.
38 Request timed out.
39 Request timed out.
40 Request timed out.
41 Request timed out.
42 Request timed out.
43 Request timed out.
44 Request timed out.
45 Request timed out.
46 Request timed out.
47 Request timed out.
48 Request timed out.
49 Request timed out.
50 Request timed out.



Explanation:

1.The programs explains about how many routers or hops it takes to send out that packet all the way to the IP address destination. Here while packet is travelling each path has a different router.

2. Request timed out message is because at the beginning/middle of a traceroute is very common.

3.This is when the device typically doesn't respond to ICMP or traceroute requests.

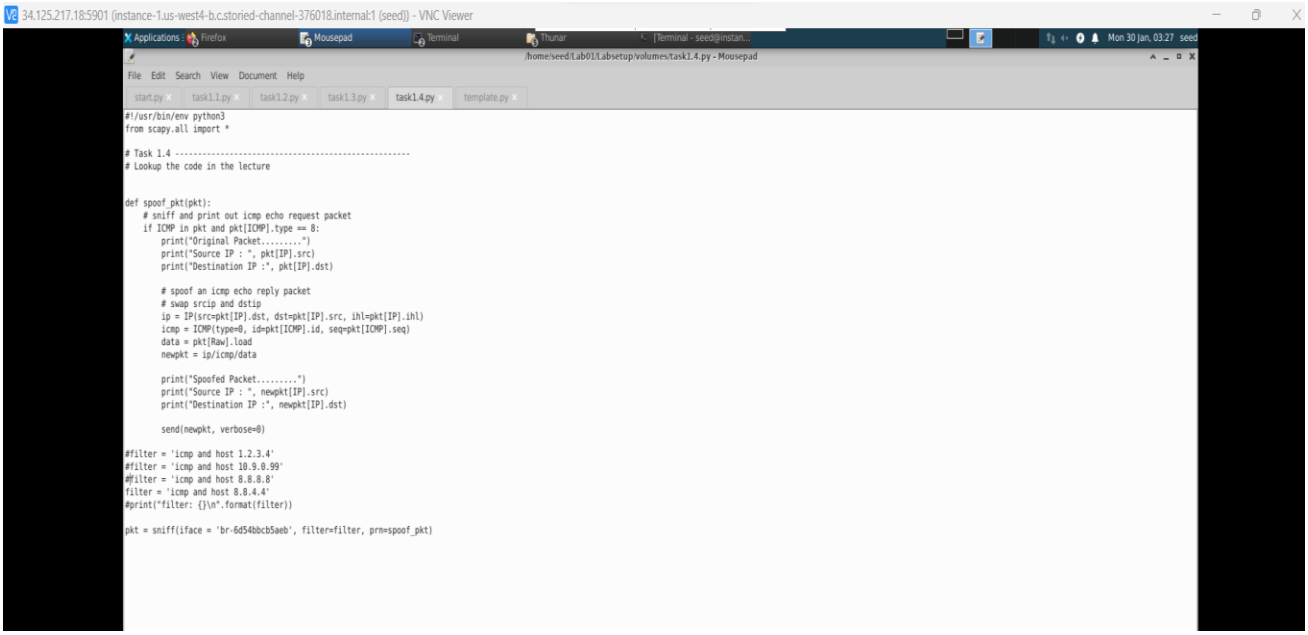
4. Here, I have used traceroute to destination IP '74.114.154.15' which is Tumblr microblogging site.

5. The flag 'ttl' of the packet is increasing by one in each given packet.

6.While loop is used till it keeps routing. The sr1() method od Scay is a method which will listen and wait for a packet to respond.

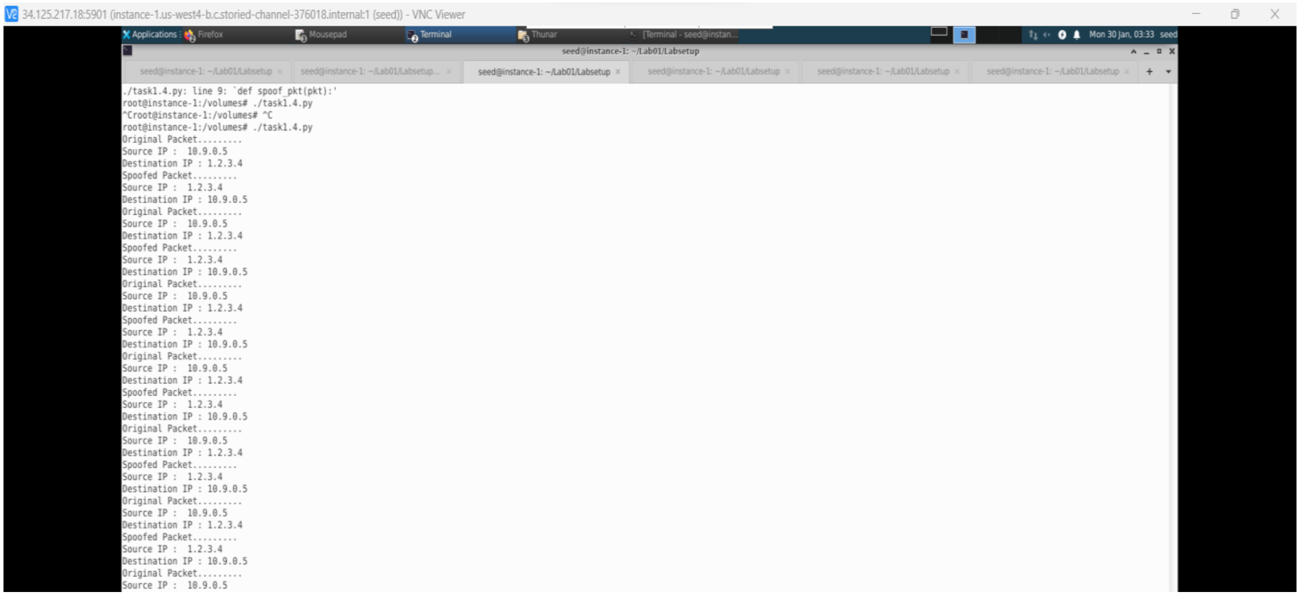
Task 1.4: Sniffing and-then Spoofing

Program-task1.4.py

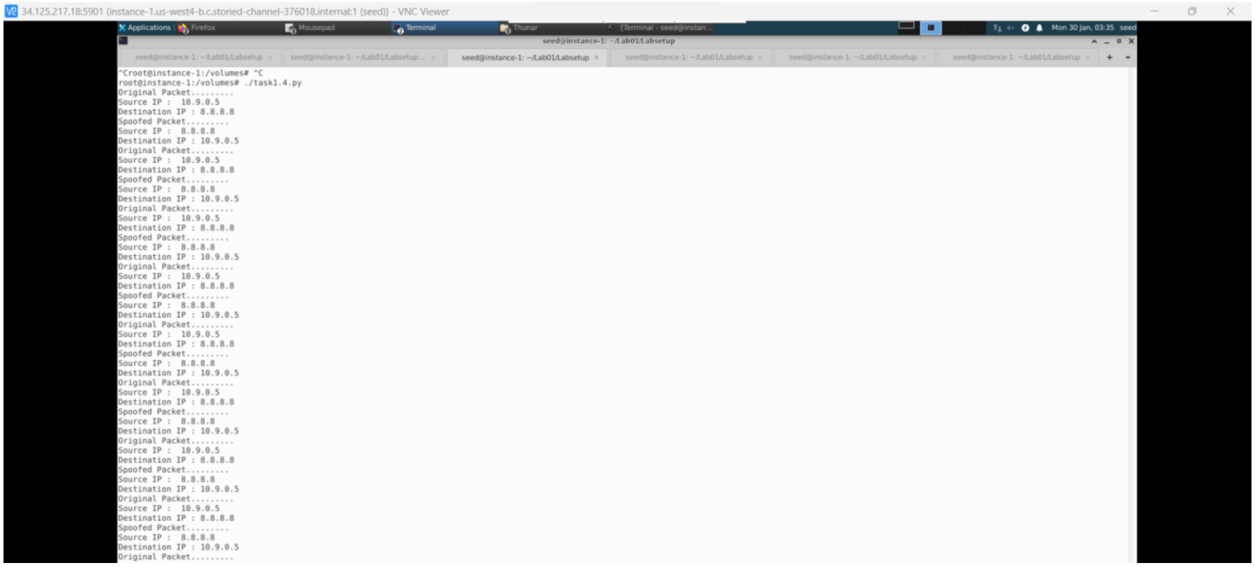



```
#!/usr/bin/env python3
from scapy.all import *

# Task 1.4 ----------------------------------------------------
# Lookup the code in the lecture


def spoof_pkt(pkt):
    # sniff and print out icmp echo request packet
    if ICMP in pkt and pkt[ICMP].type == 8:
        print("Original Packet.....")
        print("Source IP : ", pkt[IP].src)
        print("Destination IP :", pkt[IP].dst)

        # spoof an icmp echo reply packet
        # swap srcip and dstip
        ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
        icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
        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)

#filter = 'icmp and host 1.2.3.4'
#filter = 'icmp and host 10.9.0.99'
#filter = 'icmp and host 8.8.8.8'
filter = 'icmp and host 8.8.4.4'
#print("filter: {}\n".format(filter))

pkt = sniff(iface = 'br-6d54bbcb5aeb', filter=filter, prn=spoof_pkt)
```

Output:




```
root@instance-1:/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
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
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
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
```




```
^Croot@instance-1:/volumes# ^C
root@instance-1:/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
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
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.........
```

```
root@instance-1:/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
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
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
```

```
--- 8.8.4.4 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19422ms
rtt min/avg/max/mdev = 0.338/0.498/1.504/0.242 ms
root@28a5aaef7c2c:/# ./task1.4.py
bash: ./task1.4.py: No such file or directory
root@28a5aaef7c2c:/# ping 8.8.4.4
PING 8.8.4.4 (8.8.4.4) 56(84) bytes of data.
64 bytes from 8.8.4.4: icmp_seq=1 ttl=114 time=1.41 ms
64 bytes from 8.8.4.4: icmp_seq=1 ttl=64 time=65.9 ms (DUP!)
64 bytes from 8.8.4.4: icmp_seq=2 ttl=114 time=0.339 ms
64 bytes from 8.8.4.4: icmp_seq=2 ttl=64 time=18.3 ms (DUP!)
64 bytes from 8.8.4.4: icmp_seq=3 ttl=114 time=0.365 ms
64 bytes from 8.8.4.4: icmp_seq=3 ttl=64 time=23.7 ms (DUP!)
64 bytes from 8.8.4.4: icmp_seq=4 ttl=114 time=0.378 ms
64 bytes from 8.8.4.4: icmp_seq=4 ttl=64 time=19.0 ms (DUP!)
64 bytes from 8.8.4.4: icmp_seq=5 ttl=114 time=0.326 ms
64 bytes from 8.8.4.4: icmp_seq=5 ttl=64 time=16.8 ms (DUP!)
64 bytes from 8.8.4.4: icmp_seq=6 ttl=114 time=0.372 ms
64 bytes from 8.8.4.4: icmp_seq=6 ttl=64 time=19.8 ms (DUP!)
64 bytes from 8.8.4.4: icmp_seq=7 ttl=114 time=0.357 ms
64 bytes from 8.8.4.4: icmp_seq=7 ttl=64 time=18.5 ms (DUP!)
64 bytes from 8.8.4.4: icmp_seq=8 ttl=114 time=0.299 ms
64 bytes from 8.8.4.4: icmp_seq=8 ttl=64 time=17.4 ms (DUP!)
^C
```

Explanation:

1.ARP is a communication protocol used for discovering MAC address with a particular given internet layer address.

2. Here the if block is checking the ICMP request if it is true or not.

3.If it is true the reply packet we be derived from the original packet but if it is not true it will flip destination and source address whenever it will see an ICMP echo request, regardless whatever the target IP address is, here the program should send an echo request.

4. Scenario-ping 1.2.3.4 # a non-existing host on the Internet

-Here the machine senda a ping to'1.2.3.4'.

-Packetloss is present here, this why because it will never return to it's source destination.

-Here the attacker returns with an answer to the machine1 and the ICMP packet reply is coming back to machine2.

Scenario- ping 10.9.0.99 # a non-existing host on the LAN

-Here the host doesn't exist hence we are not getting request and reply packets.

-It is a non-existing host on the LAN.

Scenario-ping 8.8.4.4 # an existing host on the Internet

- This scenario is different as host exists on the net.
- Here we're having duplicate responses, the reason is because the original or the real destination is responding to the source.