

# CSC 5290 : Cyber Security Practices

## Lab1

Winter 2023

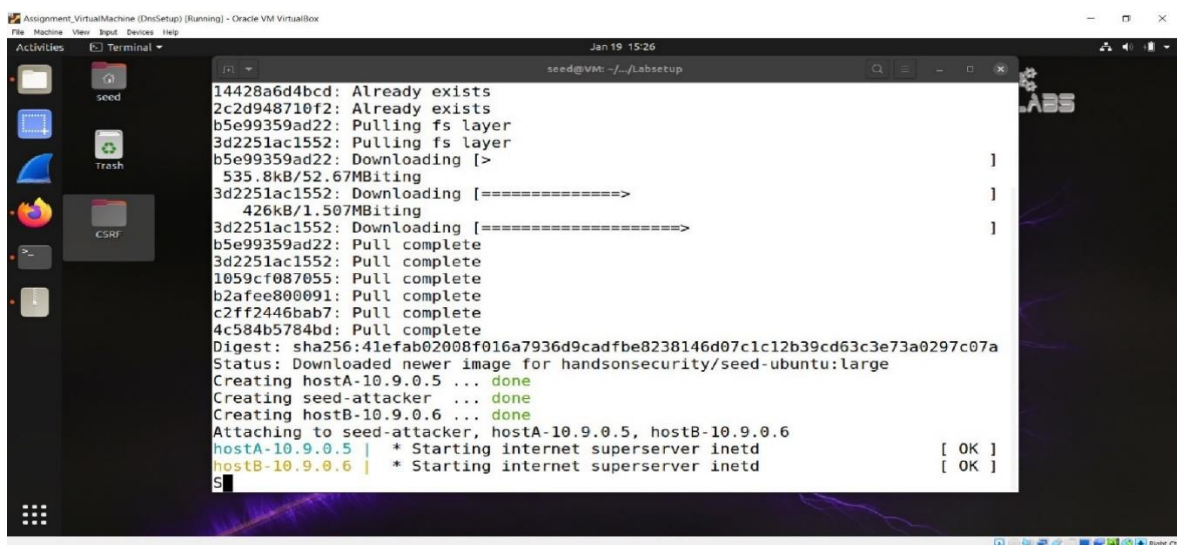
Student name: Kanchan Chopde

### Packet Sniffing and Spoofing

Lab setup :

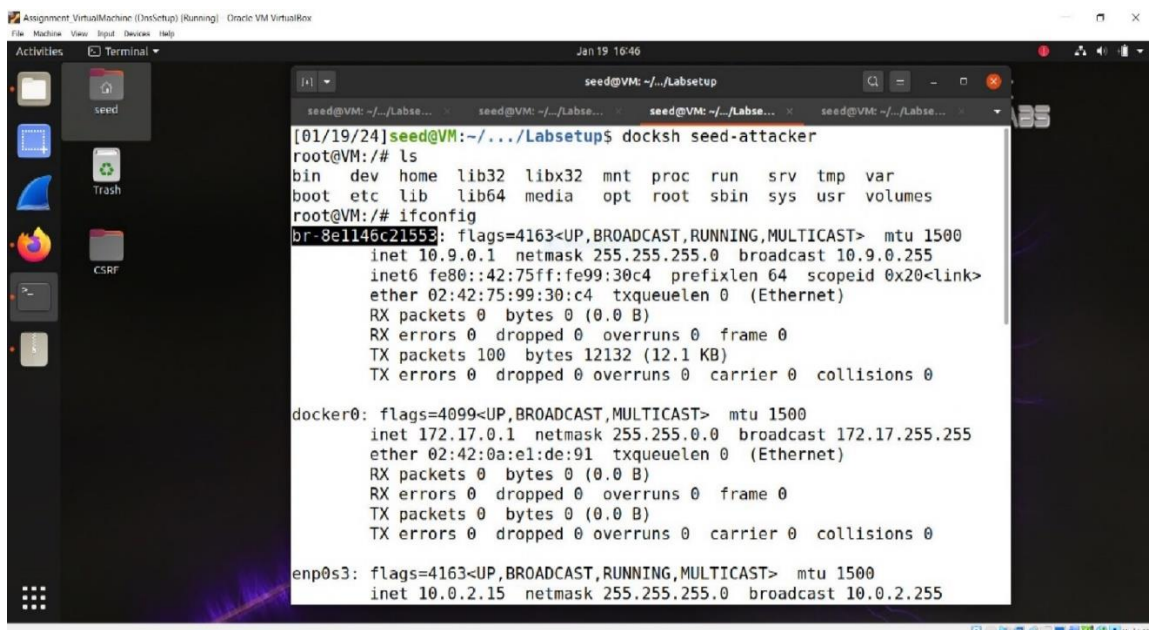
This screenshot shows we are making use of dcup command i.e. docker-compose up to start the services mentioned in docker-compose.yml.

Here it is creating and starting seed-attacker, Host A and Host B.



```
seed@VM: ~/Labsetup
14428a6d4bcd: Already exists
2c2d948710f2: Already exists
b5e99359ad22: Pulling fs layer
3d2251ac1552: Pulling fs layer
b5e99359ad22: Downloading [>
  535.8kB/52.67MBbiting
3d2251ac1552: Downloading [=====>
  426kB/1.507MBbiting
3d2251ac1552: Downloading [=====>
b5e99359ad22: Pull complete
3d2251ac1552: Pull complete
1059cf087055: Pull complete
b2afee800091: Pull complete
c2ff2446bab7: Pull complete
4c584b5784bd: Pull complete
Digest: sha256:41efab02008f016a7936d9cadf8e8238146d07c1c12b39cd63c3e73a0297c07a
Status: Downloaded newer image for handsontest/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 ]
S
```

Below we use ifconfig command to get name of corresponding network interface starting with br- and the ID by docker.

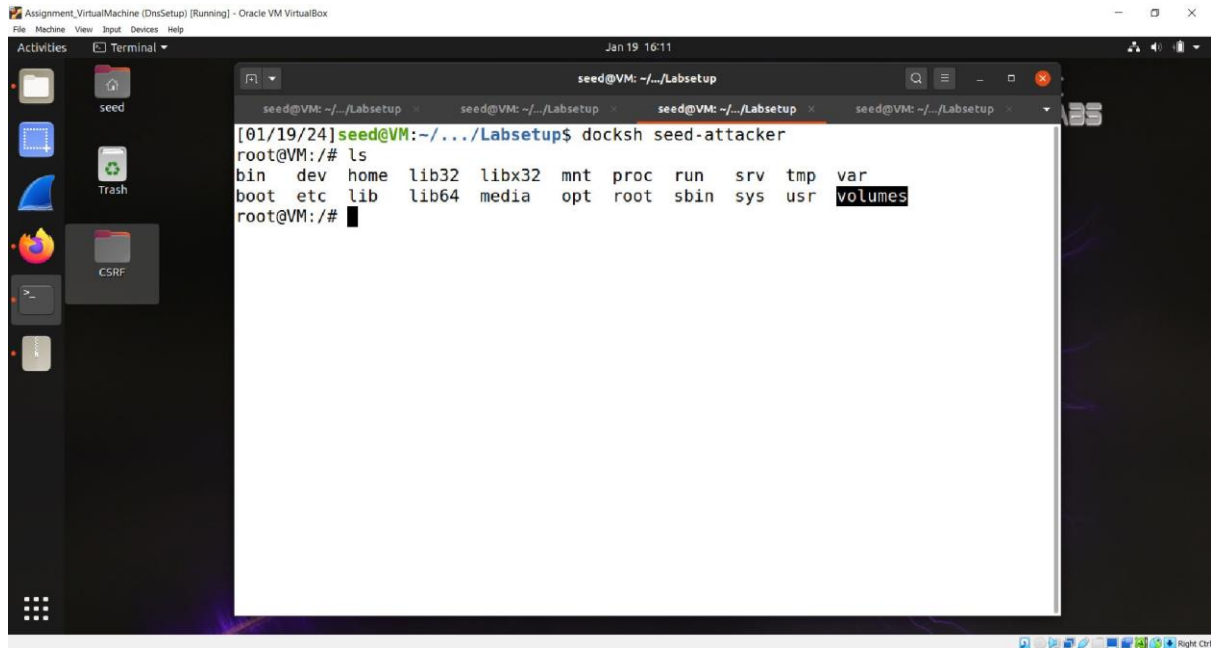


```
seed@VM: ~/Labsetup
[01/19/24]seed@VM:~/.../Labsetup$ docksh seed-attacker
root@VM:/# ls
bin dev home lib32 libx32 mnt proc run srv tmp var
boot etc lib lib64 media opt root sbin sys usr volumes
root@VM:/# ifconfig
br-8e1146c21553: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.1 netmask 255.255.255.0 broadcast 10.9.0.255
    inet6 fe80::42:75ff:fe99:30c4 prefixlen 64 scopeid 0x20<link>
    ether 02:42:75:99:30:c4 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 100 bytes 12132 (12.1 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
    ether 02:42:0a:e1:de:91 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 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
```

Inside attacker we can see here volumes directory.

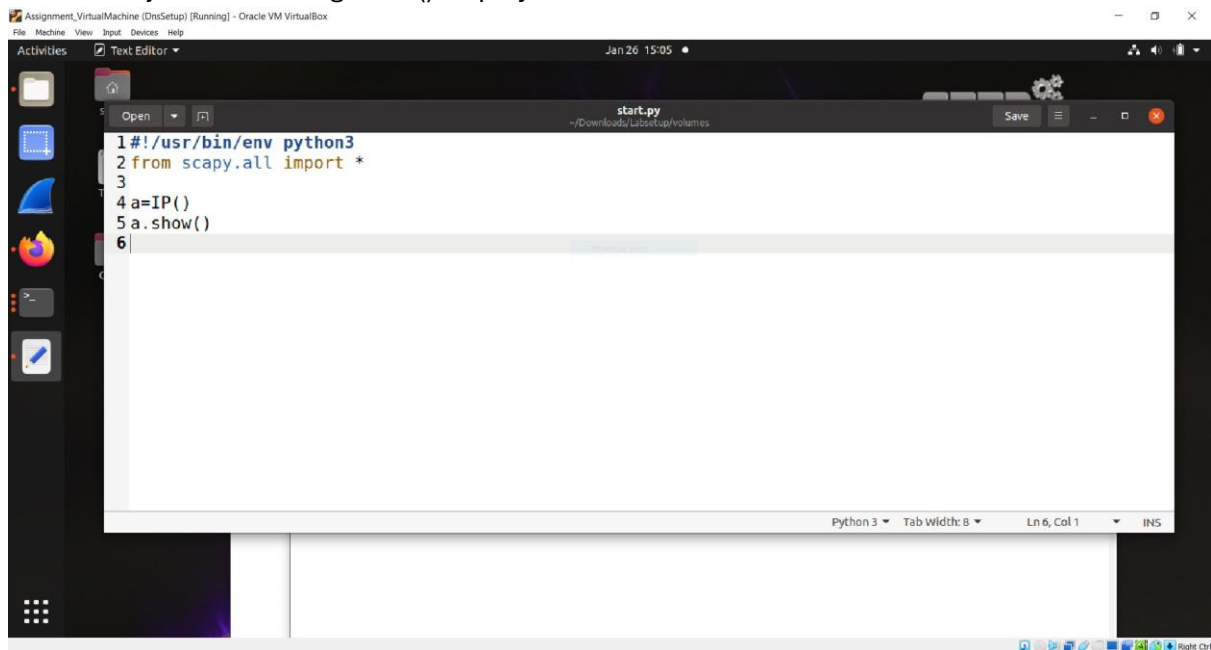


The screenshot shows a terminal window titled 'seed@VM: ~/.../Labsetup' with a date of Jan 19 16:11. The user 'seed@VM' runs the command 'docksh seed-attacker'. The prompt changes to 'root@VM:/#'. The user then runs 'ls', which displays the following directory listing:

```
bin  dev  home  lib32  libx32  mnt  proc  run  srv  tmp  var
boot  etc  lib   lib64  media  opt  root  sbin  sys  usr  volumes
```

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

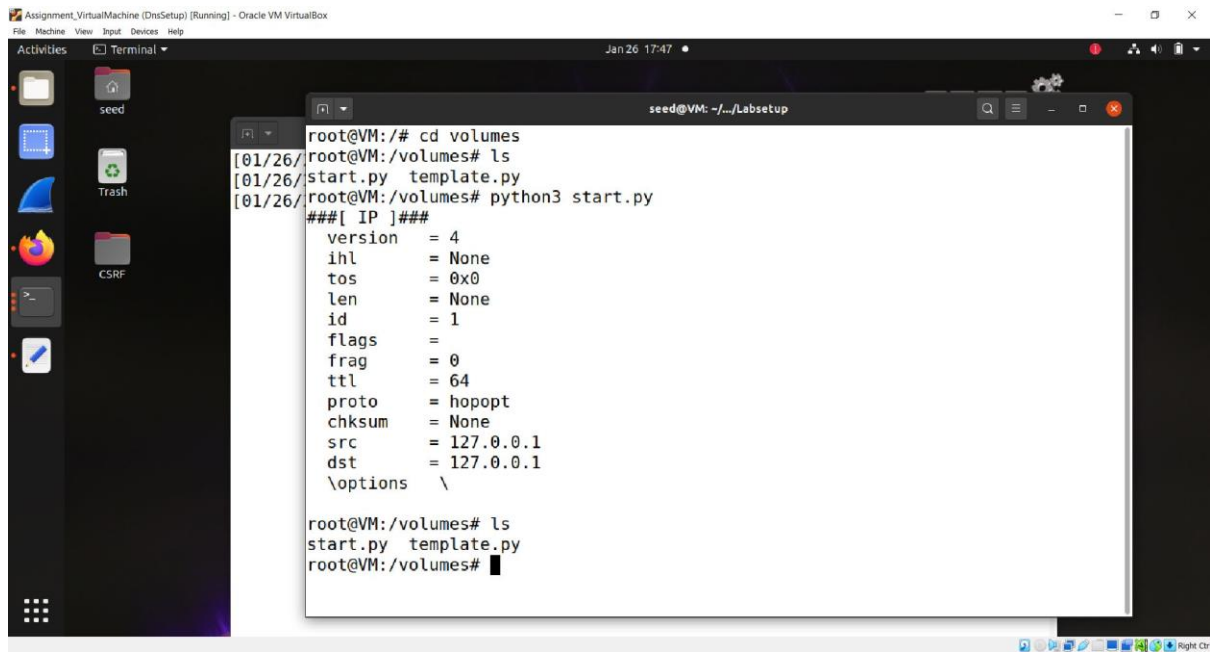
We create file name: start.py file and add below code to it in volumes folder.  
make IP object a and using show() display.



The screenshot shows a text editor window titled 'start.py' with a date of Jan 26 15:05. The code in the file is as follows:

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

To run start.py file following commands are used: python3 start.py  
Here we can see output IP object is displayed.

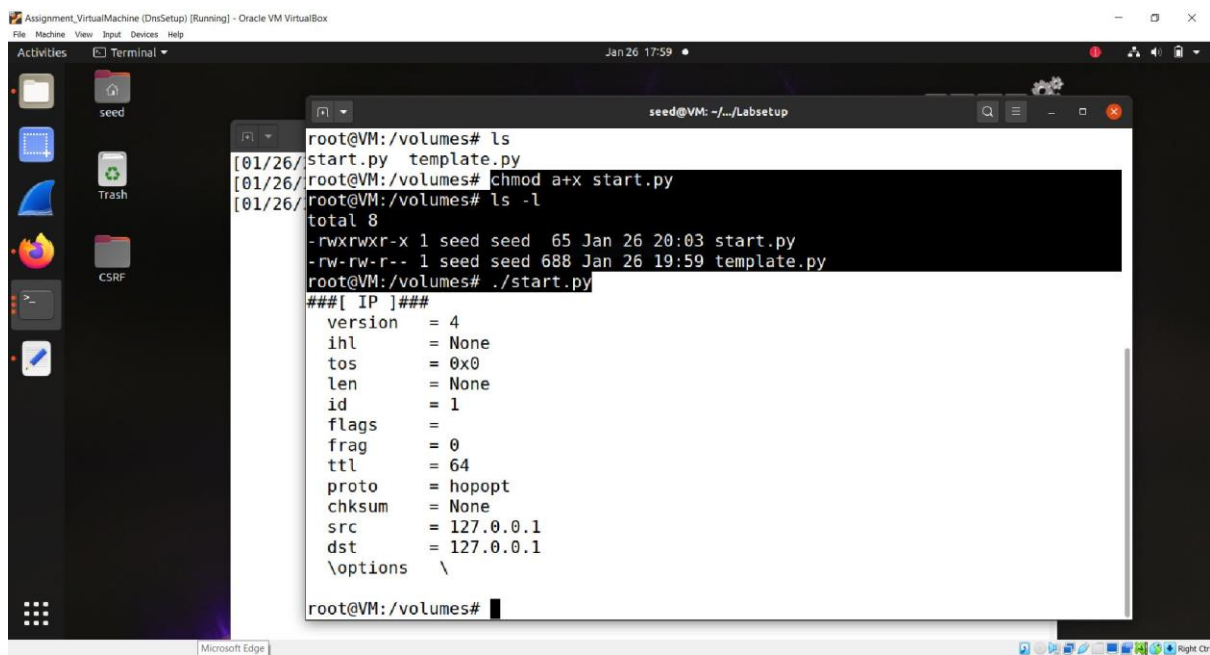


The screenshot shows a VirtualBox window titled "Assignment\_VirtualMachine (DnsSetup) [Running] - Oracle VM VirtualBox". Inside, a terminal window titled "seed@VM: ~/Labsetup" is open. The terminal shows the following commands and output:

```
root@VM:/# cd volumes
root@VM:/volumes# ls
[01/26/2023 17:47] start.py template.py
root@VM:/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 \

root@VM:/volumes# ls
start.py template.py
root@VM:/volumes#
```

We can also use command: `chmod a+x start.py` and `./start.py`



The screenshot shows the same VirtualBox window and terminal. The terminal output continues from the previous state:

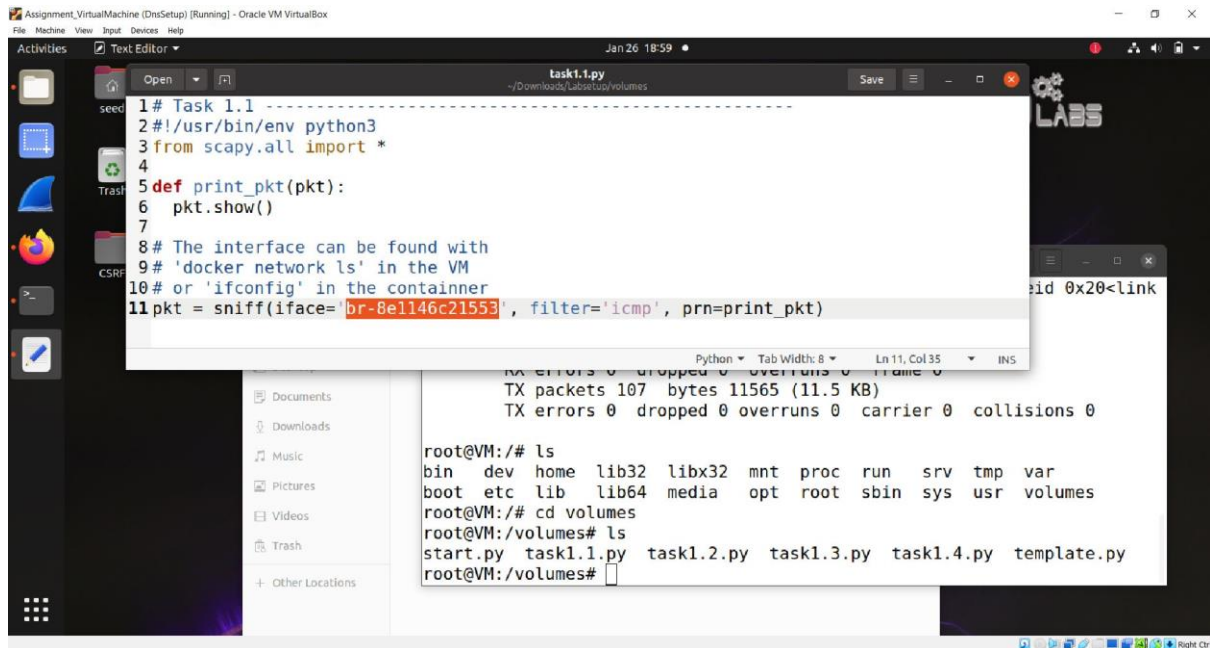
```
root@VM:/volumes# ls
start.py template.py
[01/26/2023 17:59] root@VM:/volumes# chmod a+x start.py
[01/26/2023 17:59] root@VM:/volumes# ls -l
total 8
-rwxrwxr-x 1 seed seed 65 Jan 26 20:03 start.py
-rw-rw-r-- 1 seed seed 688 Jan 26 19:59 template.py
root@VM:/volumes# ./start.py
###[ IP ]###
version = 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 \

root@VM:/volumes#
```

## Task 1.1: Sniffing Packets

Create task1.1.py file and add the code displayed below.

here we use sniff function with iface value as our interface and filter as icmp and function call to print\_pkt() which displayed pkt which we created.



The screenshot shows a VirtualBox VM window titled "Assignment\_VirtualMachine (DnsSetup) [Running] - Oracle VM VirtualBox". Inside the VM, a text editor window titled "task1.1.py" is open, displaying the following Python code:

```
1 # Task 1.1 -----
2#!/usr/bin/env python3
3from scapy.all import *
4
5def print_pkt(pkt):
6    pkt.show()
7
8# The interface can be found with
9# 'docker network ls' in the VM
10# or 'ifconfig' in the container
11pkt = sniff(iface='br-8e1146c21553', filter='icmp', prn=print_pkt)
```

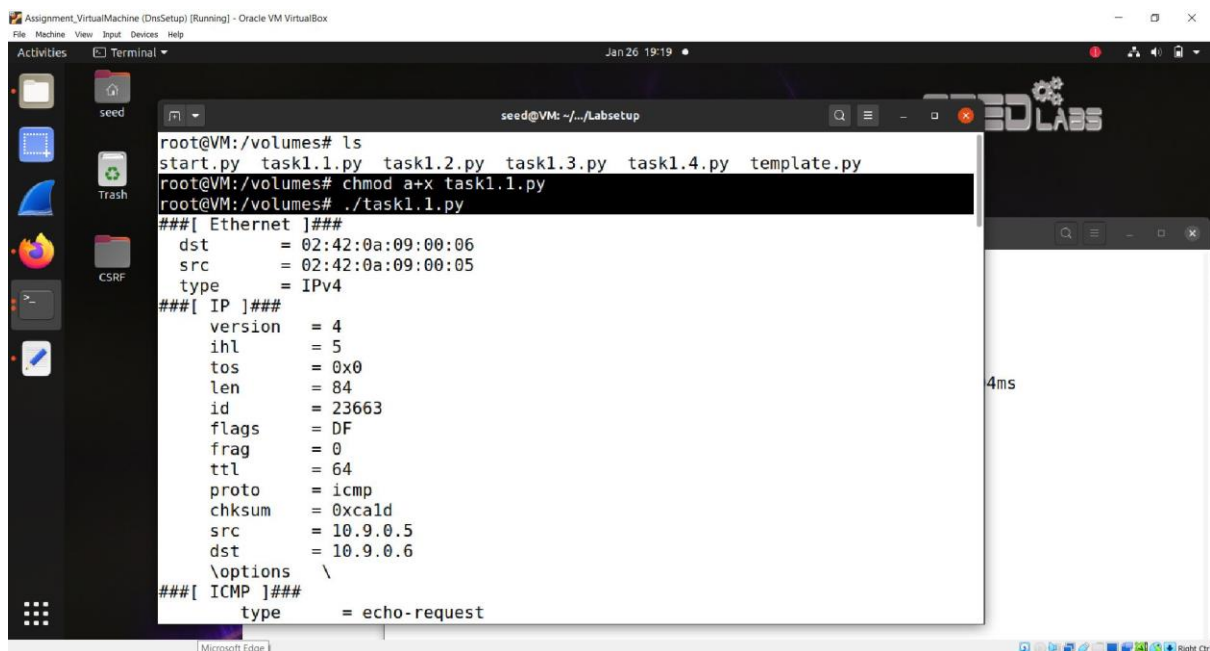
Below the text editor, a terminal window shows the execution of the script:

```
root@VM:/# ls
bin dev home lib32 libx32 mnt proc run srv tmp var
boot etc lib lib64 media opt root sbin sys usr volumes
root@VM:/# cd volumes
root@VM:/volumes# ls
start.py task1.1.py task1.2.py task1.3.py task1.4.py template.py
root@VM:/volumes#
```

run task 1.1 using command on seed-attacker

chmod a+x task1.1.py

./task1.1.py



The screenshot shows a VirtualBox VM window titled "Assignment\_VirtualMachine (DnsSetup) [Running] - Oracle VM VirtualBox". Inside the VM, a terminal window titled "seed@VM: ~/../Labsetup" is open, displaying the following commands and output:

```
root@VM:/volumes# ls
start.py task1.1.py task1.2.py task1.3.py task1.4.py template.py
root@VM:/volumes# chmod a+x task1.1.py
root@VM:/volumes# ./task1.1.py
###[ Ethernet ]###
  dst      = 02:42:0a:09:00:06
  src      = 02:42:0a:09:00:05
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 23663
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = 0xcald
  src      = 10.9.0.5
  dst      = 10.9.0.6
  \options \
###[ ICMP ]###
  type     = echo-request
```

Assignment\_VirtualMachine (DnsSetup) [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Activities Terminal Jan 26 19:17

seed

```

ihl      = 5
tos      = 0x0
len      = 84
id       = 13314
flags    =
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0x328
src      = 10.9.
dst      = 10.9.
\options \
###[ ICMP ]###
        type    = ec
        code    = 0
        chksum   = 0x
        id      = 0x
        seq     = 0x
###[ Raw ]###
        load    =
00\x10\x11\x12\x13\x14
-./01234567'

```

seed@VM: ~/../Labsetup

```

root@5dc7fd52121e:/# 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=0.100 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.168 ms
^C
--- 10.9.0.6 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1004ms
rtt min/avg/max/mdev = 0.100/0.134/0.168/0.034 ms
root@5dc7fd52121e:/#

```

Microsoft Edge

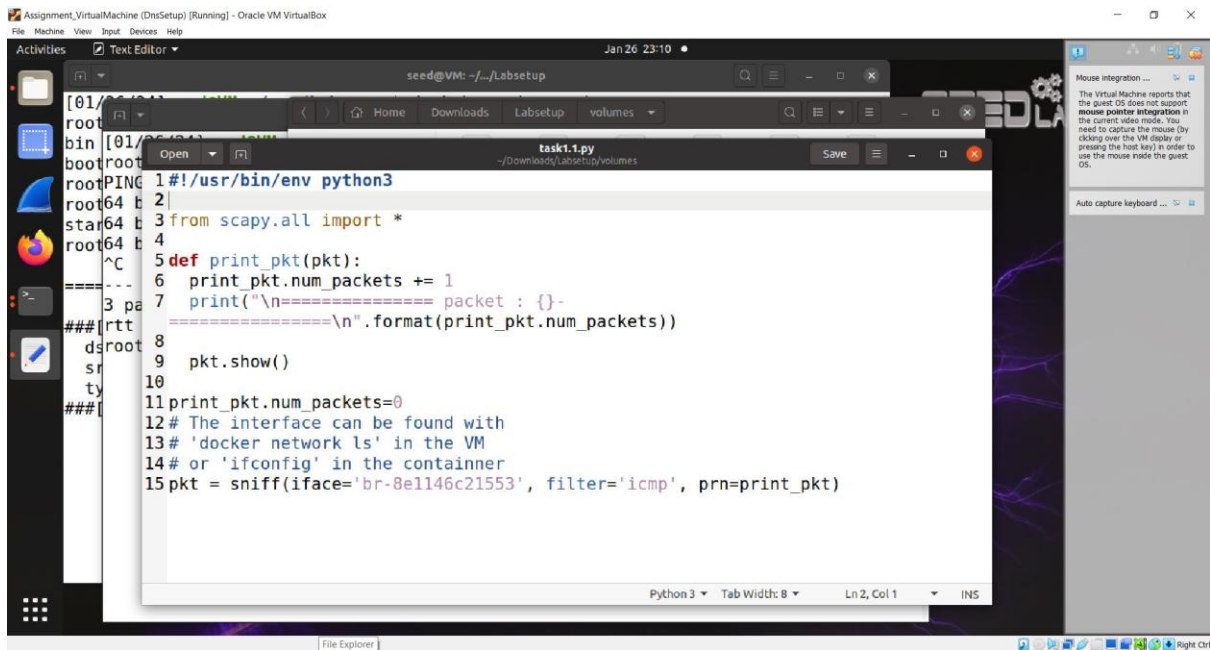
```

root@kali:~/netcat# nc -l -p 4444
Ncat: Version 0.9.24 (2020-01-21)
Ncat listening on 0.0.0.0 port 4444
10.10.10.10:~
root@kali:~/netcat# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
games:x:5:12:games:/usr/games:/usr/sbin/nologin
ftp:x:14:14:ftp:/var:/usr/sbin/nologin
root:x:0:0:root:/root:/bin/bash

```

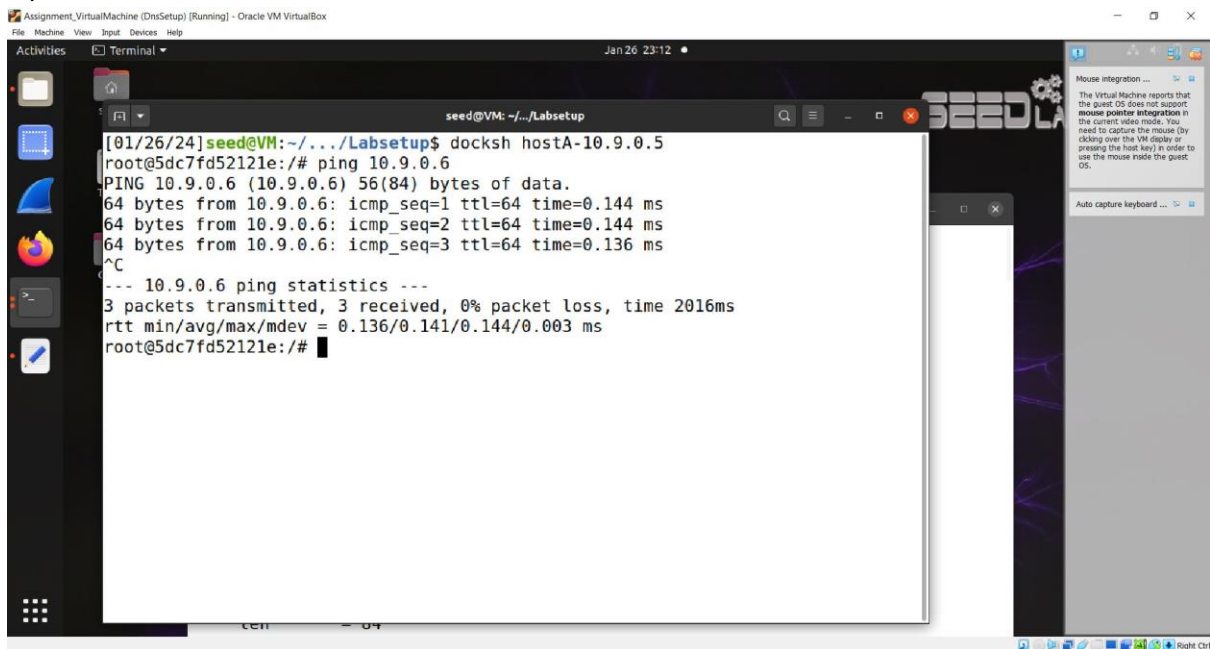


Now we make changes in code to get packet information printed.



```
seed@VM: ~/.../Labsetup
[01/26/24] root@5dc7fd52121e:~# cat task1.1.py
1#!/usr/bin/env python3
2
3from scapy.all import *
4
5def print_pkt(pkt):
6    print_pkt.num_packets += 1
7    print("\n===== packet : {}-".format(print_pkt.num_packets))
8
9    pkt.show()
10
11print_pkt.num_packets=0
12# The interface can be found with
13# 'docker network ls' in the VM
14# or 'ifconfig' in the container
15pkt = sniff(iface='br-8e1146c21553', filter='icmp', prn=print_pkt)
```

again we run it on attacker and listen from host A we ping to hostB:  
we observe following :  
3 packets transmitted and 3 received.



```
Assignment_VirtualMachine (DnsSetup) [Running] - Oracle VM VirtualBox
Activities Terminal
seed@VM: ~/.../Labsetup
[01/26/24] seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@5dc7fd52121e:~# 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=0.144 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.144 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.136 ms
^C
--- 10.9.0.6 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2016ms
rtt min/avg/max/mdev = 0.136/0.141/0.144/0.003 ms
root@5dc7fd52121e:~#
```

Here, we can see after seeing the results of running modified code to print packet number there are exactly 6 packets in total.

```

seed@VM: ~/Labsetup
f !"#%&'()*+,-./01234567

===== packet : 5=====

###[ 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       = 22968
flags    = DF
frag     = 0
ttl      = 64
proto    = icmp
chksum   = 0xccc4
src      = 10.9.0.5
dst      = 10.9.0.6

\options \
###[ ICMP ]###
type     = echo-request
code     = 0
chksum   = 0x8199
id       = 0x27
seq      = 0x3

###[ Raw ]###
load     = '\x0f\x82\xb4e\x00\x00\x00\x00\xee\x81\x05\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&'()*+,-./01234567'

===== packet : 6=====

```

## Task 1.1B.

Please set the following filters and demonstrate your sniffer program again (each filter should be set separately):

- Capture only the ICMP packet

Task1.1.py file has filter set as “icmp”

```

task1.1.py
~/Downloads/LabSetup/volumes

1#!/usr/bin/env python3
2
3from scapy.all import *
4
5def print_pkt(pkt):
6    print_pkt.num_packets += 1
7    print("\n===== packet : {}-
8    =====\n".format(print_pkt.num_packets))
9
10    pkt.show()
11
12print_pkt.num_packets=0
13# The interface can be found with
14# 'docker network ls' in the VM
15# or 'ifconfig' in the container
16pkt = sniff(iface='br-8e1146c21553', filter='icmp', prn=print_pkt)

```

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

Here we set tcp protocol, src host as 10.9.0.6 and destination port is 23.

After checking found port 23 corresponds to telnet.

```

3 from scapy.all import *
4
5 def print_pkt(pkt):
6     print_pkt.num_packets += 1
7     print("\n===== packet : {} =====\n".format(print_pkt.num_packets))
8
9     pkt.show()
10
11 print_pkt.num_packets=0
12 # The interface can be found with
13 # 'docker network ls' in the VM
14 # or 'ifconfig' in the container
15 # 1.1 capture only icmp packet
16 #pkt = sniff(iface='br-8e1146c21553', filter='icmp', prn=print_pkt)
17
18 # Capture any TCP packet that comes from a particular IP and with a destination port number
19 # 23.
20 pkt = sniff(iface='br-8e1146c21553', filter='tcp && src host 10.9.0.6 && dst port 23',
21 prn=print_pkt)
  
```

Here we can see seed-attacker gets src= 10.9.0.6 and dst=10.9.0.5

Host B uses telnet 10.9.0.5 and attacker gets information of TCP protocol with Ips and dport as telnet which is port no. 23

```

===== packet : 71=====
###[ Ethernet ]###
  dst      = 02:42:0a:09:00:05
  src      = 02:42:0a:09:00:06
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x10
  len      = 52
  id       = 43201
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = 0x7dd6
  src      = 10.9.0.6
  dst      = 10.9.0.5
  \options \
###[ TCP ]###
  sport    = 48134
  dport    = telnet
  seq      = 3778944331
  ack      = 182314270
  dataofs  = 0
  reserved = 0
  flags    = A
  window   = 501
  chksum   = 0x1443
  urgptr   = 0
  options  = [('NOP', None), ('NOP', None), ('Timestamp', None)]
  
```

```

64 bytes from 10.9.0.6: icmp_seq=16 ttl=64 time=0.113 ms
64 bytes from 10.9.0.6: icmp_seq=17 ttl=64 time=0.307 ms
64 bytes from 10.9.0.6: icmp_seq=18 ttl=64 time=0.134 ms
64 bytes from 10.9.0.6: icmp_seq=19 ttl=64 time=0.234 ms
64 bytes from 10.9.0.6: icmp_seq=20 ttl=64 time=0.145 ms
64 bytes from 10.9.0.6: icmp_seq=21 ttl=64 time=0.084 ms
64 bytes from 10.9.0.6: icmp_seq=22 ttl=64 time=0.093 ms
64 bytes from 10.9.0.6: icmp_seq=23 ttl=64 time=0.085 ms
64 bytes from 10.9.0.6: icmp_seq=24 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=25 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=26 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=27 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=28 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=29 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=30 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=31 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=32 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=33 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=34 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=35 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=36 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=37 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=38 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=39 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=40 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=41 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=42 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=43 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=44 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=45 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=46 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=47 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=48 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=49 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=50 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=51 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=52 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=53 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=54 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=55 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=56 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=57 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=58 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=59 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=60 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=61 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=62 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=63 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=64 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=65 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=66 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=67 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=68 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=69 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=70 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=71 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=72 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=73 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=74 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=75 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=76 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=77 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=78 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=79 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=80 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=81 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=82 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=83 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=84 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=85 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=86 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=87 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=88 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=89 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=90 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=91 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=92 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=93 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=94 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=95 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=96 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=97 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=98 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=99 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=100 ttl=64 time=0.064 ms
  
```

```

seed@5dc7fd52121e:~$ telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
5dc7fd52121e login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

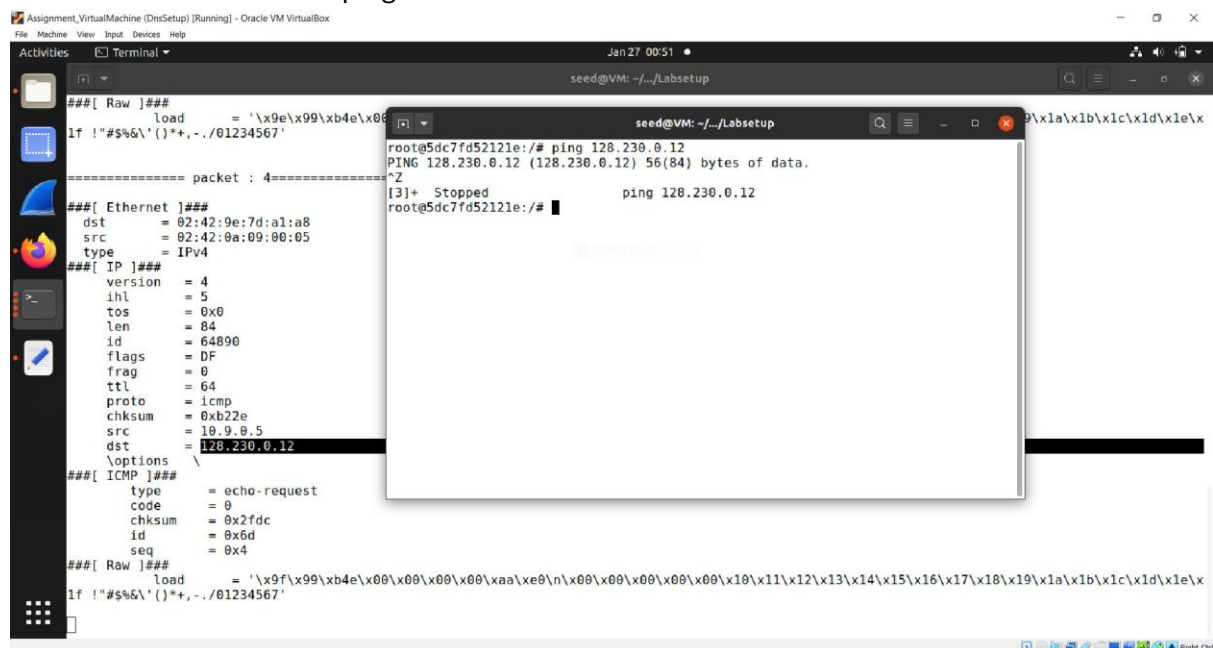
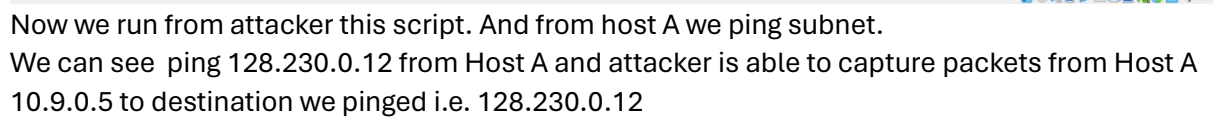
 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

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

To restore this content, you can run the 'unminimize' command.
Last login: Sat Jan 27 05:15:56 UTC 2024 from 5dc7fd52121e on pts/5
seed@5dc7fd52121e:~$
  
```

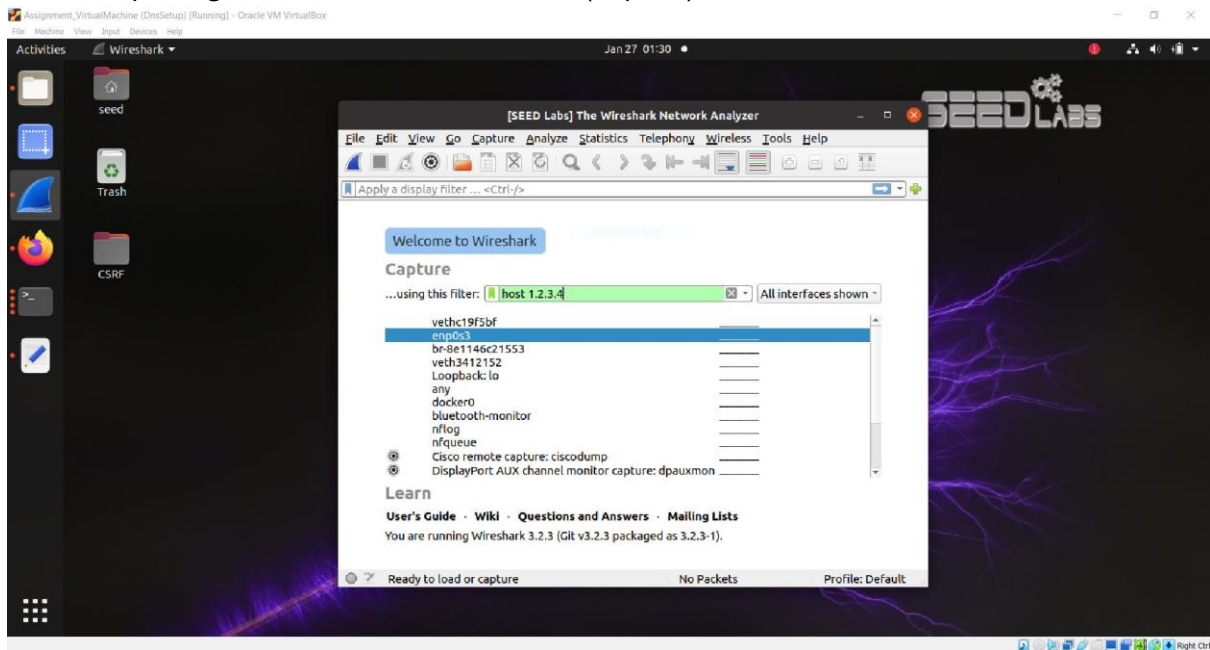


- Added “net 128.230.0.0/16” in the filter to start capturing from the particular subnet mentioned.

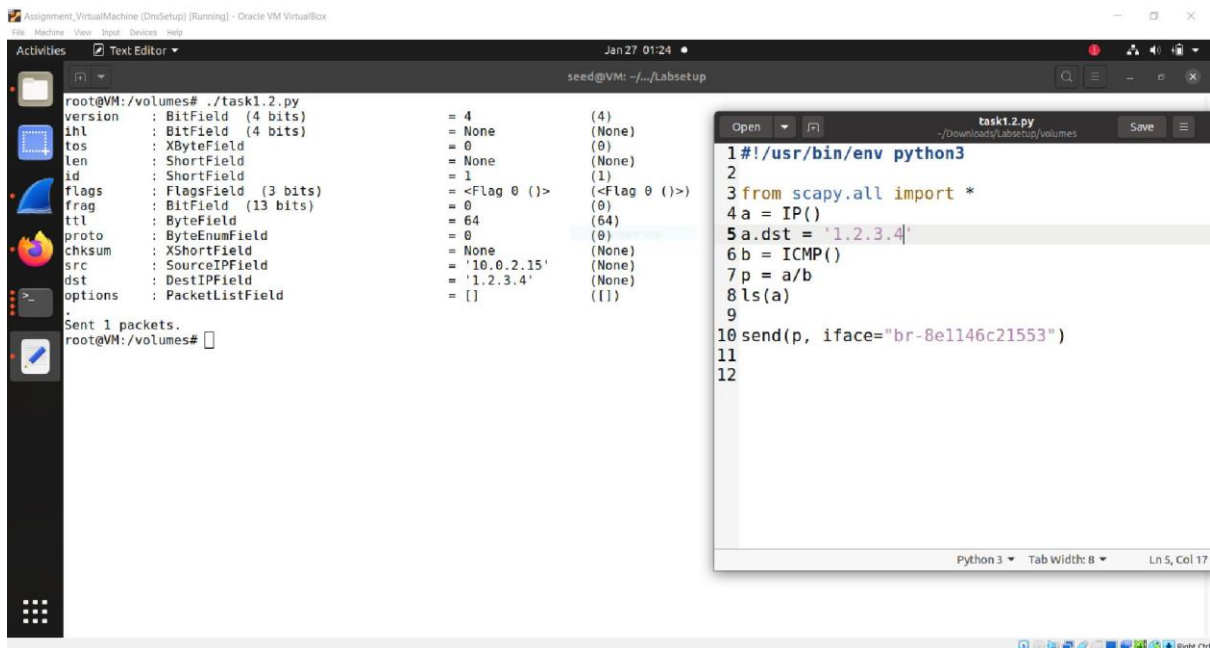


## Task 1.2: Spoofing ICMP Packets

We are capturing from our interface 10.0.2.15 (enp0s3) and host 1.2.3.4



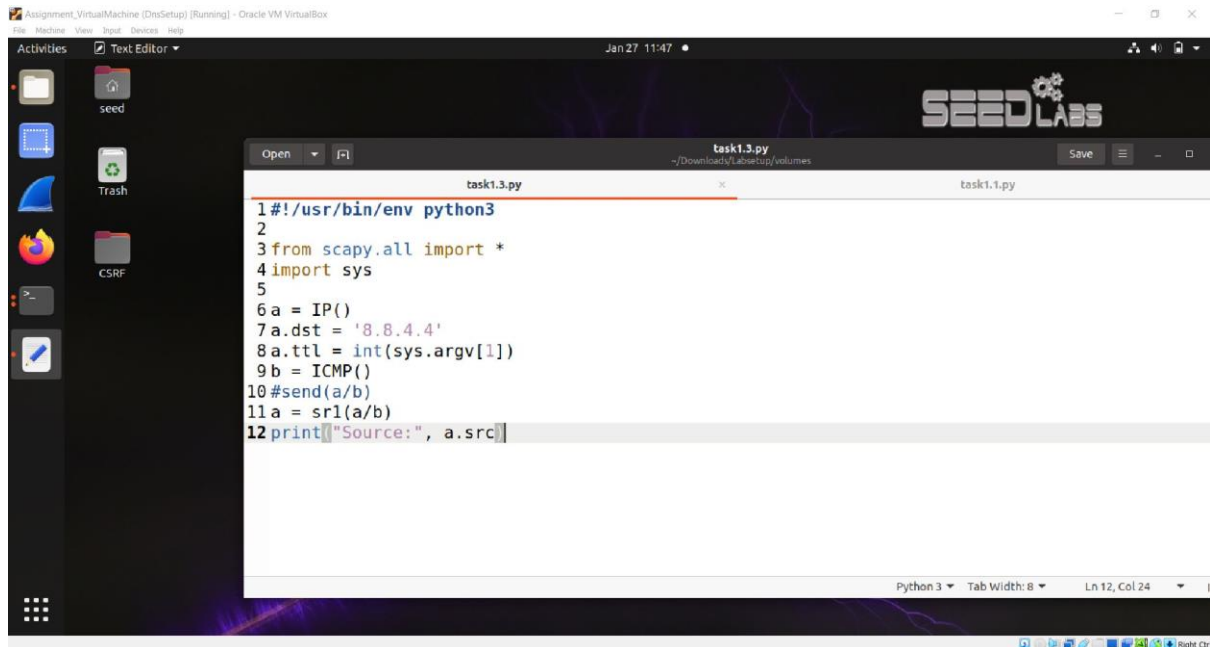
Now we add interface name in send () and destination as 1.2.3.4



### Task 1.3: Traceroute

Here for tracing route we write this program: set destination IP = 8.8.4.4 and take from command line TTL (Time to Live). We make use of `sr1()` from `scapy` which returns one packet that answered the packet sent. In this way, we can track the route of packets until we get destination.

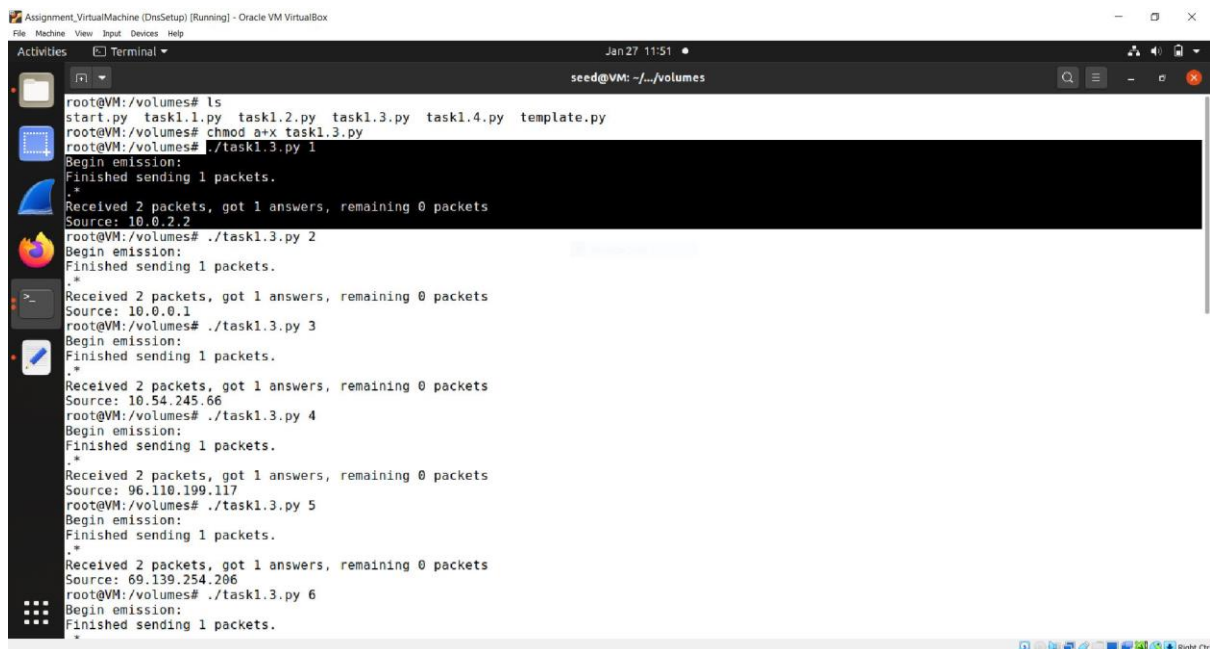
Also print the source, so we can verify the route.

A screenshot of a virtual machine window titled "Assignment\_VirtualMachine (DnsSetup) [Running] - Oracle VM VirtualBox". The window shows a desktop environment with a sidebar containing icons for "seed", "Trash", and "CSRF". A text editor window titled "task1.3.py" is open, displaying the following Python code:

```
1#!/usr/bin/env python3
2
3from scapy.all import *
4import sys
5
6a = IP()
7a.dst = '8.8.4.4'
8a.ttl = int(sys.argv[1])
9b = ICMP()
10#send(a/b)
11a = sr1(a/b)
12print("Source:", a.src)
```

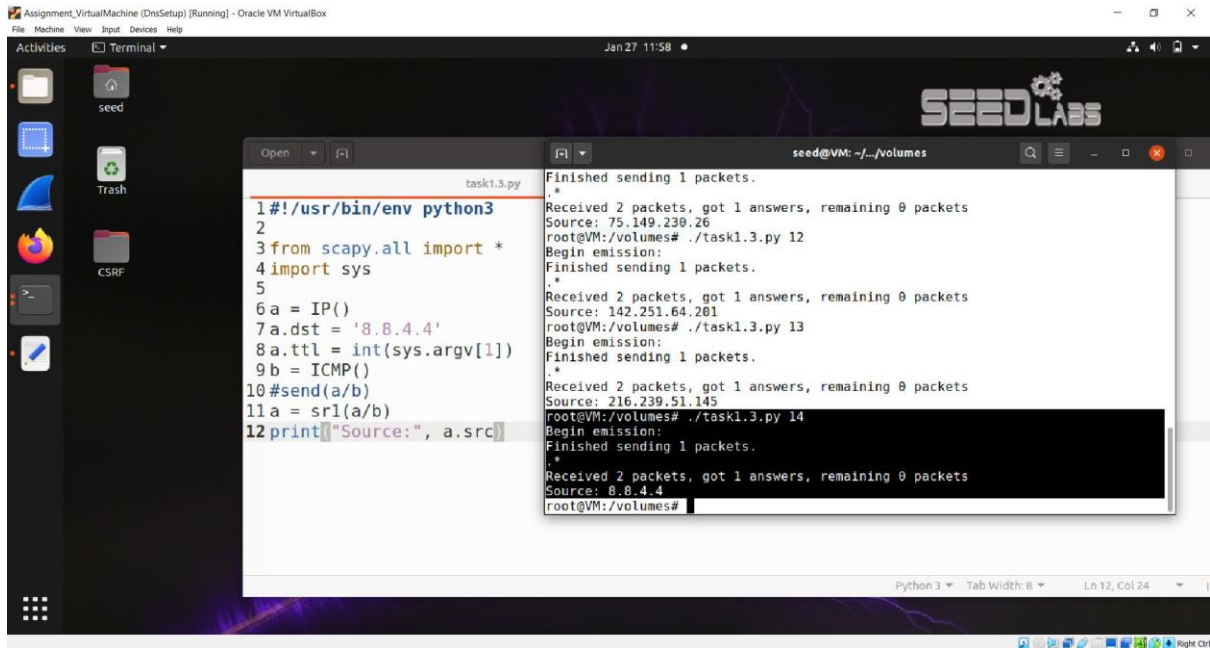
The status bar at the bottom of the text editor indicates "Python 3", "Tab Width: 8", and "Ln 12, Col 24".

After running the code we see: src is 10.0.2.2 and we keep giving argument from command line until we reach destination.

A screenshot of a terminal window titled "Assignment\_VirtualMachine (DnsSetup) [Running] - Oracle VM VirtualBox". The terminal shows the execution of the `task1.3.py` script with different TTL values. The output for each run is as follows:

```
root@VM:/volumes# ls
start.py task1.1.py task1.2.py task1.3.py task1.4.py template.py
root@VM:/volumes# chmod a+x task1.3.py
root@VM:/volumes# ./task1.3.py 1
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 10.0.2.2
root@VM:/volumes# ./task1.3.py 2
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 10.0.0.1
root@VM:/volumes# ./task1.3.py 3
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 10.54.245.66
root@VM:/volumes# ./task1.3.py 4
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 96.110.199.117
root@VM:/volumes# ./task1.3.py 5
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 69.139.254.206
root@VM:/volumes# ./task1.3.py 6
Begin emission:
Finished sending 1 packets.
.*
```

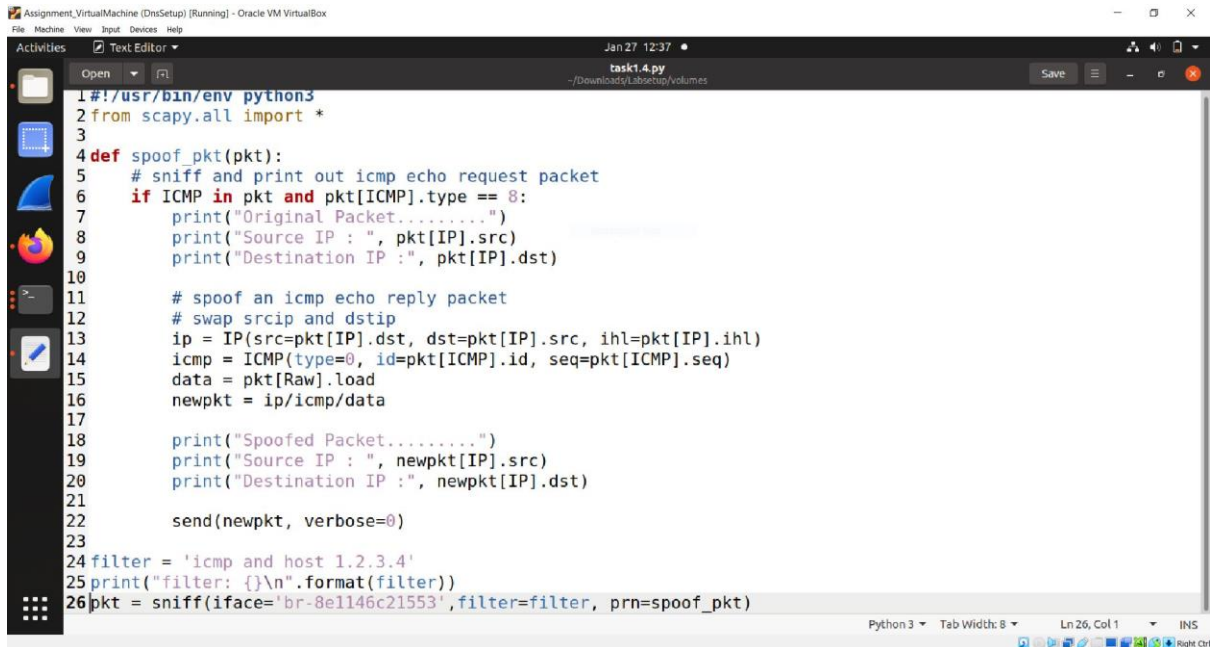
We can see it reached destination after giving 14 as TTL.



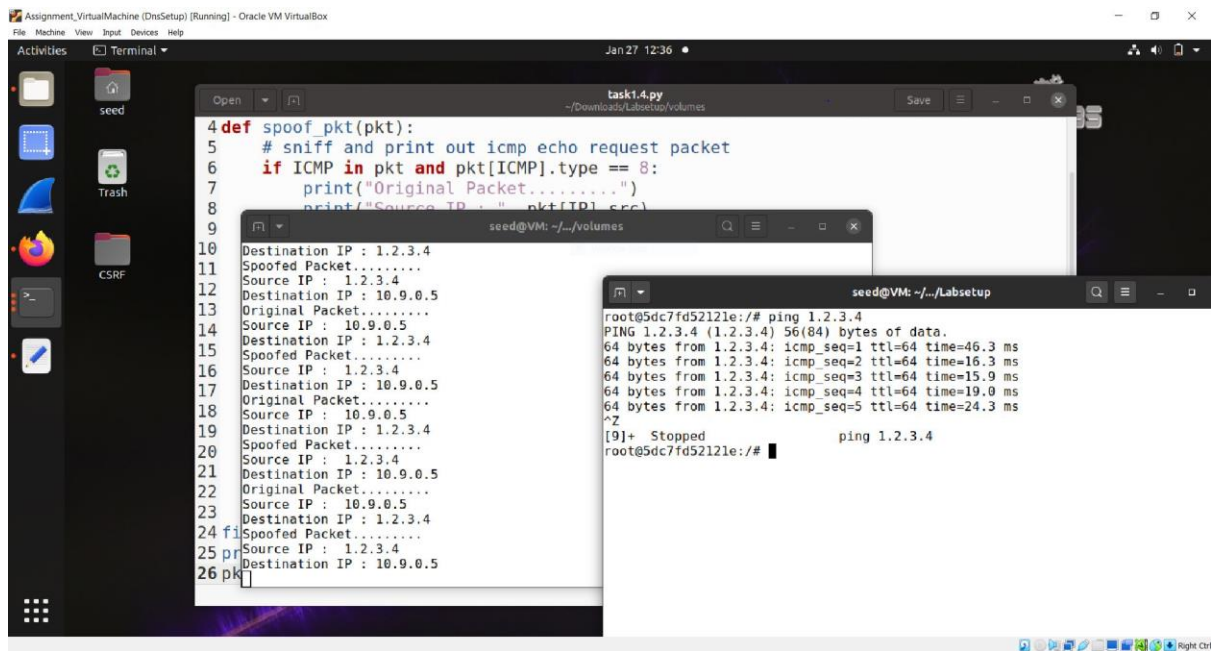
```
Assignment_VirtualMachine (DnsSetup) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal Jan 27 11:58
seed
Trash
CSRF
task1.3.py
1#!/usr/bin/env python3
2
3from scapy.all import *
4import sys
5
6a = IP()
7a.dst = '8.8.4.4'
8a.ttl = int(sys.argv[1])
9b = ICMP()
10#send(a/b)
11a = sr1(a/b)
12print("Source:", a.src)
Finished sending 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 75.149.230.26
root@VM: /volumes# ./task1.3.py 12
Begin emission:
Finished sending 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 142.251.64.201
root@VM: /volumes# ./task1.3.py 13
Begin emission:
Finished sending 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 216.239.51.145
root@VM: /volumes# ./task1.3.py 14
Begin emission:
Finished sending 1 packets.
*
Received 2 packets, got 1 answers, remaining 0 packets
Source: 8.8.4.4
root@VM: /volumes#
```

#### **Task 1.4: Sniffing and-then Spoofing**

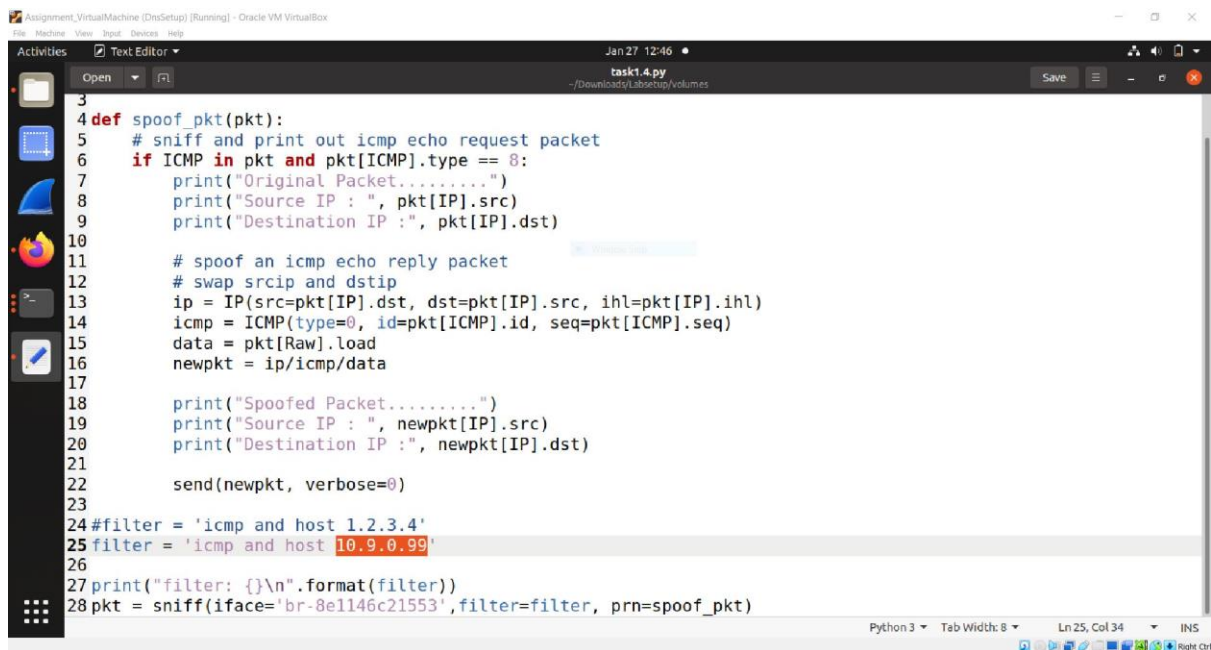
ping 1.2.3.4 # a non-existing host on the Internet



```
Assignment_VirtualMachine (DnsSetup) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Text Editor Jan 27 12:37
task1.4.py
1#!/usr/bin/env python3
2from scapy.all import *
3
4def spoof_pkt(pkt):
5    # sniff and print out icmp echo request packet
6    if ICMP in pkt and pkt[ICMP].type == 8:
7        print("Original Packet.....")
8        print("Source IP : ", pkt[IP].src)
9        print("Destination IP :", pkt[IP].dst)
10
11    # spoof an icmp echo reply packet
12    # swap srcip and dstip
13    ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
14    icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
15    data = pkt[Raw].load
16    newpkt = ip/icmp/data
17
18    print("Spoofed Packet.....")
19    print("Source IP : ", newpkt[IP].src)
20    print("Destination IP :", newpkt[IP].dst)
21
22    send(newpkt, verbose=0)
23
24filter = 'icmp and host 1.2.3.4'
25print("filter: {}\n".format(filter))
26pkt = sniff(iface='br-8e1146c21553', filter=filter, prn=spoof_pkt)
```

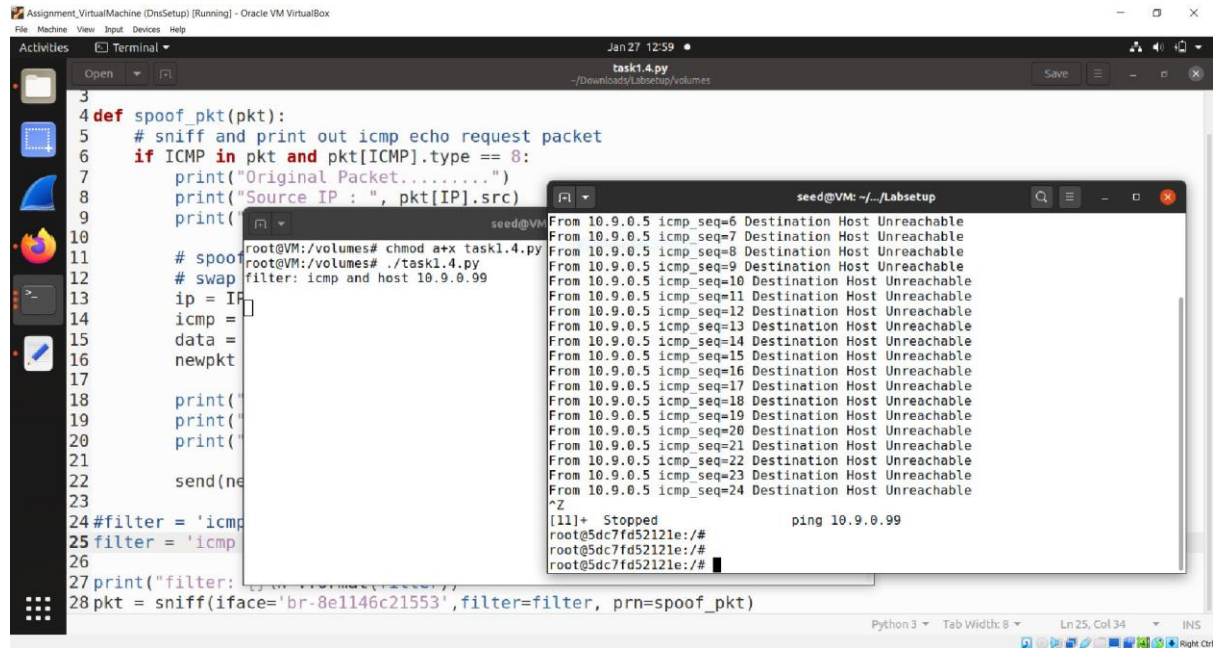


ping 10.9.0.99 # a non-existing host on the LAN





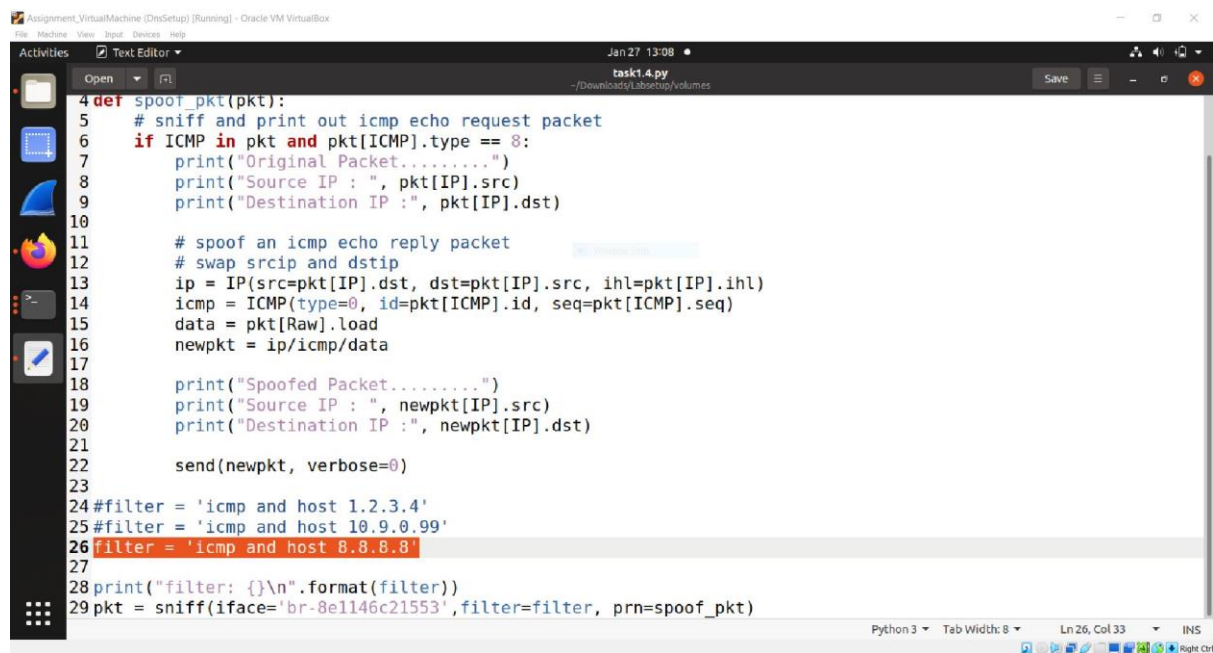
results: Destination Host is not reachable in this case



```
3 def spoof_pkt(pkt):
4     # sniff and print out icmp echo request packet
5     if ICMP in pkt and pkt[ICMP].type == 8:
6         print("Original Packet.....")
7         print("Source IP : ", pkt[IP].src)
8         print("Destination IP : ", pkt[IP].dst)
9
10    # spoof an icmp echo reply packet
11    # swap srcip and dstip
12    ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
13    icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
14    data = pkt[Raw].load
15    newpkt = ip/icmp/data
16
17    print("Spoofed Packet.....")
18    print("Source IP : ", newpkt[IP].src)
19    print("Destination IP : ", newpkt[IP].dst)
20
21    send(newpkt, verbose=0)
22
23 #filter = 'icmp and host 1.2.3.4'
24 #filter = 'icmp and host 10.9.0.99'
25 filter = 'icmp and host 8.8.8.8'
26 print("filter: {}".format(filter))
27 pkt = sniff(iface='br-8e1146c21553', filter=filter, prn=spoof_pkt)
```

ping 8.8.8.8 # an existing host on the Internet

Code changes to added IP 8.8.8.8 as host



```
4 def spoof_pkt(pkt):
5     # sniff and print out icmp echo request packet
6     if ICMP in pkt and pkt[ICMP].type == 8:
7         print("Original Packet.....")
8         print("Source IP : ", pkt[IP].src)
9         print("Destination IP : ", pkt[IP].dst)
10
11    # spoof an icmp echo reply packet
12    # swap srcip and dstip
13    ip = IP(src=pkt[IP].dst, dst=pkt[IP].src, ihl=pkt[IP].ihl)
14    icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
15    data = pkt[Raw].load
16    newpkt = ip/icmp/data
17
18    print("Spoofed Packet.....")
19    print("Source IP : ", newpkt[IP].src)
20    print("Destination IP : ", newpkt[IP].dst)
21
22    send(newpkt, verbose=0)
23
24 #filter = 'icmp and host 1.2.3.4'
25 #filter = 'icmp and host 10.9.0.99'
26 filter = 'icmp and host 8.8.8.8'
27 print("filter: {}".format(filter))
28 pkt = sniff(iface='br-8e1146c21553', filter=filter, prn=spoof_pkt)
```

We get results as duplicate packets received i.e. DUP! we do not know which one is spoofed at host side but at attacker we have printed information.

The screenshot shows a terminal window with a Python script named `task1.4.py` running. The script is designed to sniff and print out ICMP echo request packets. It uses a filter to capture packets from 8.8.8.8 to 10.9.0.5. The script prints the original packet details and then creates a spoofed packet with the same destination IP but a different source IP (8.8.8.8). The packet capture window shows a list of captured packets, many of which are marked as duplicates (DUP!). The terminal output shows the script's execution, including the filter being applied and the packets being captured.

```

3
4 def spoof_pkt(pkt):
5     # sniff and print out icmp echo request packet
6     if ICMP in pkt and pkt[ICMP].type == 8:
7         print("Original Packet.....")
8         print("Source IP : ", pkt[IP].src)
9         print("Destination IP : ", pkt[IP].dst)
10
11 # spoof
12 # swap
13 ip = IP
14 icmp = ICMP
15 data = Spoofed Packet.....
16 Source IP : 8.8.8.8
17 Destination IP : 10.9.0.5
18 Original Packet.....
19 Source IP : 10.9.0.5
20 Destination IP : 8.8.8.8
21 Spoofed Packet.....
22 Source IP : 8.8.8.8
23 Destination IP : 10.9.0.5
24 #filter = 'icmp and host 8.8.8.8'
25 #filter = 'icmp and host 10.9.0.5'
26 filter = 'icmp and host 8.8.8.8'
27
28 print("filter: {}".format(filter))

```

Packet capture output (seed@VM: ~/Labsetup):

```

64 bytes from 8.8.8.8: icmp_seq=13 ttl=64 time=16.8 ms
64 bytes from 8.8.8.8: icmp_seq=13 ttl=53 time=20.3 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=14 ttl=64 time=19.2 ms
64 bytes from 8.8.8.8: icmp_seq=14 ttl=53 time=23.2 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=15 ttl=64 time=16.4 ms
64 bytes from 8.8.8.8: icmp_seq=15 ttl=53 time=23.2 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=16 ttl=64 time=18.5 ms
64 bytes from 8.8.8.8: icmp_seq=16 ttl=53 time=23.0 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=17 ttl=64 time=18.6 ms
64 bytes from 8.8.8.8: icmp_seq=17 ttl=53 time=19.0 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=18 ttl=64 time=17.3 ms
64 bytes from 8.8.8.8: icmp_seq=18 ttl=53 time=19.0 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=19 ttl=64 time=14.6 ms
64 bytes from 8.8.8.8: icmp_seq=19 ttl=53 time=16.5 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=20 ttl=64 time=26.9 ms
64 bytes from 8.8.8.8: icmp_seq=20 ttl=53 time=28.3 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=21 ttl=64 time=18.2 ms
64 bytes from 8.8.8.8: icmp_seq=21 ttl=53 time=21.7 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=22 ttl=64 time=20.7 ms
64 bytes from 8.8.8.8: icmp_seq=23 ttl=53 time=16.8 ms
64 bytes from 8.8.8.8: icmp_seq=24 ttl=53 time=10.3 ms

```

## Task 2.1A: Understanding How a Sniffer Works

**Question 1.** Please use your own words to describe the sequence of the library calls that are essential for sniffer programs. This is meant to be a summary, not a detailed explanation like the one in the tutorial or book.

For sniffer program in c using pcap api following is the sequence of library calls:

- 1) using `pcap_open_live()` function to capture packet .
- 2) using `pcap_compile()` to compile filter expression to convert it into BPF code (Berkely filter packet)
- 3) using `pcap_setfilters()` to set the compiled filters to capture only specific packets based filter expression.
- 4) using `pcap_loop()` to capture packets with few parameters stating when to stop loop or otherwise.

**Question 2.** Why do you need the root privilege to run a sniffer program? Where does the program fail if it is executed without the root privilege?

Without root privileges it will not execute and will display operation not permitted error.

**Question 3.** The value 1 of the third parameter in `pcap_open_live()` turns on the promiscuous mode (use 0 to turn it off). Can you tell the difference when this mode is on and off ?

Promiscuous mode allows a network interface to receive and process all incoming traffic, regardless of whether it is intended for the interface's MAC address or not.

To turn on promiscuous mode in a sniffer program, you set the third parameter of `pcap_open_live()` to 1. To turn it off, you set the parameter to 0