LAB 3

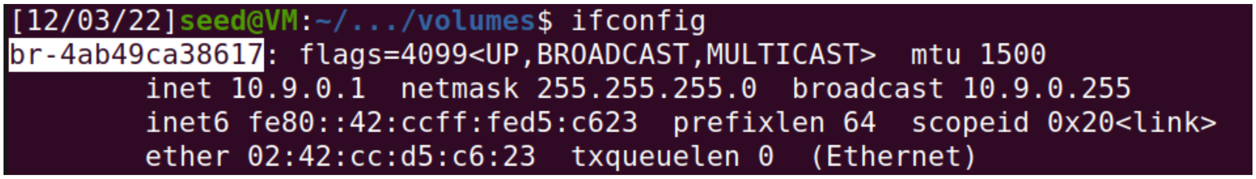
Spoorthi Badikala | sbadi006 | 862324457

Task-1 Using Scapy to Sniff and Spoof Packets:

Task-1.1 Sniffing Packets:

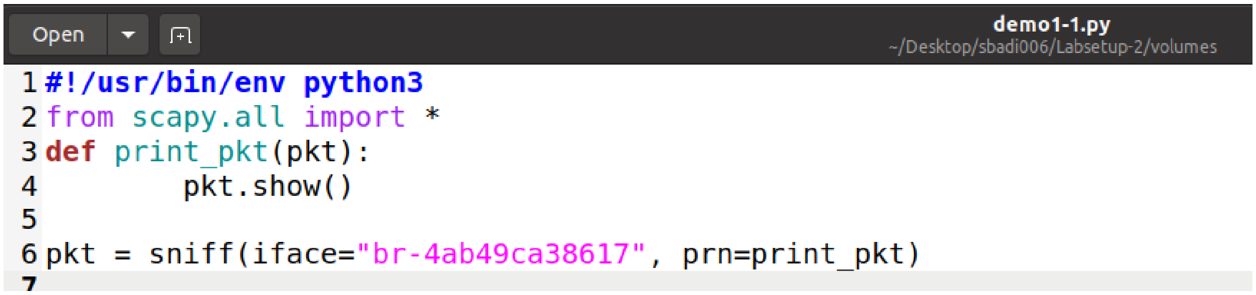
Task 1.1 A:

Using the ifconfig command in the attacker terminal, the iface value was fetched.



```
[12/03/22]seed@VM:~/.../volumes$ ifconfig
br-4ab49ca38617: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:ccff:fed5:c623  prefixlen 64  scopeid 0x20<link>
        ether 02:42:cc:d5:c6:23  txqueuelen 0  (Ethernet)
```

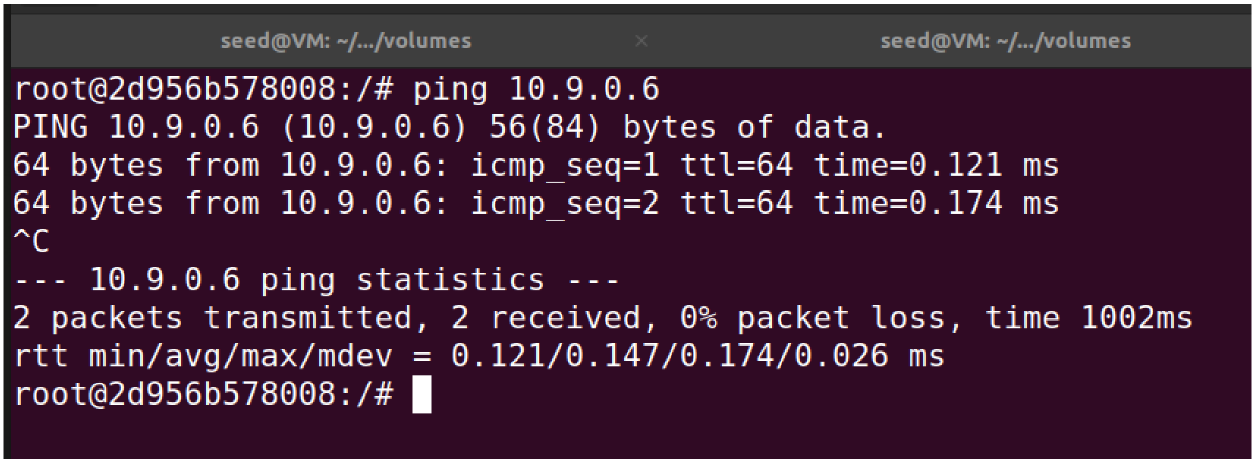
After following the instructions to setup, I created a file named ‘demo1-1.py’ and copied the scapy code mentioned in the instructions.



```
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
        pkt.show()

pkt = sniff(iface="br-4ab49ca38617", prn=print_pkt)
```

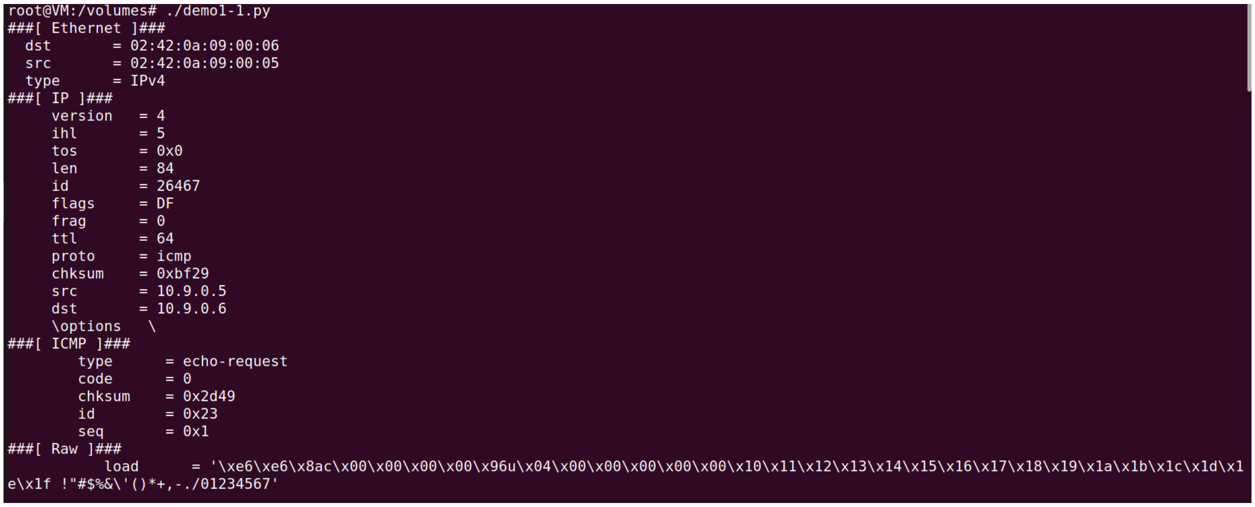
Initially I ran this code on the attacker side with root access and was able to get the packets being transmitted by hostA

–HostA terminal:



```
root@2d956b578008:/# 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=0.121 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.174 ms
^C
--- 10.9.0.6 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.121/0.147/0.174/0.026 ms
root@2d956b578008:/# 
```

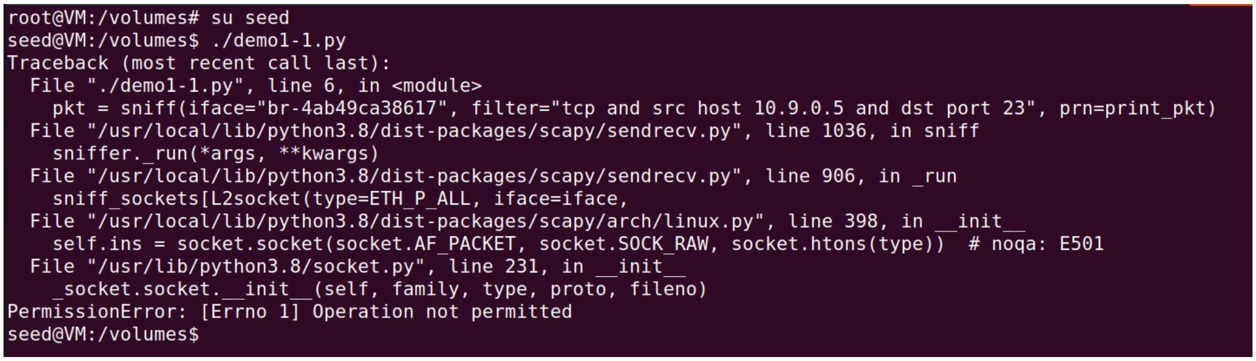
–Attacker terminal:



```
root@VM:/volumes# ./demo1-1.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        = 26467
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xbf29
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x2d49
        id        = 0x23
        seq       = 0x1
###[ Raw ]###
           load      = '\xe6\xe6\x8ac\x00\x00\x00\x00\x96u\x04\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1
e\x1f !"#$%&\'()*+,-./01234567'
```

To observe how the code behaves without root access, it was disabled using su seed command. The attacker code could not attack the host without root access.

Without root access:

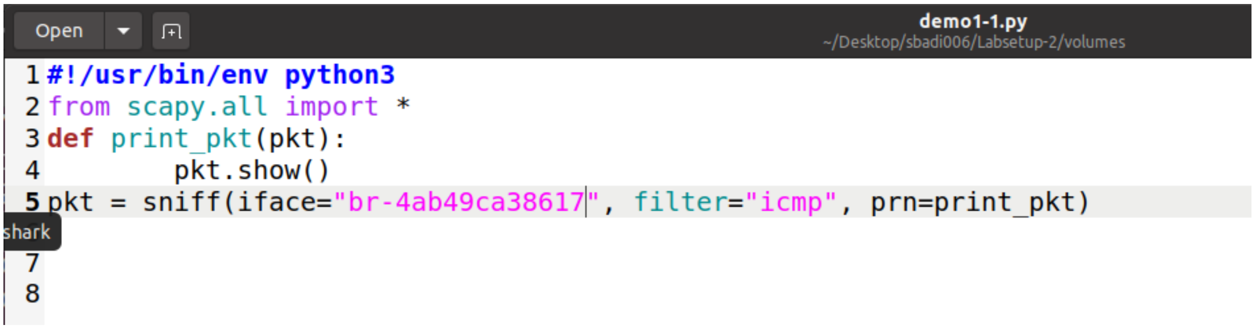


```
root@VM:/volumes# su seed
seed@VM:/volumes$ ./demo1-1.py
Traceback (most recent call last):
  File "./demo1-1.py", line 6, in <module>
    pkt = sniff(iface="br-4ab49ca38617", filter="tcp and src host 10.9.0.5 and dst port 23", prn=print_pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", 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
seed@VM:/volumes$
```

Task-1.1 B: Capture only the ICMP packet

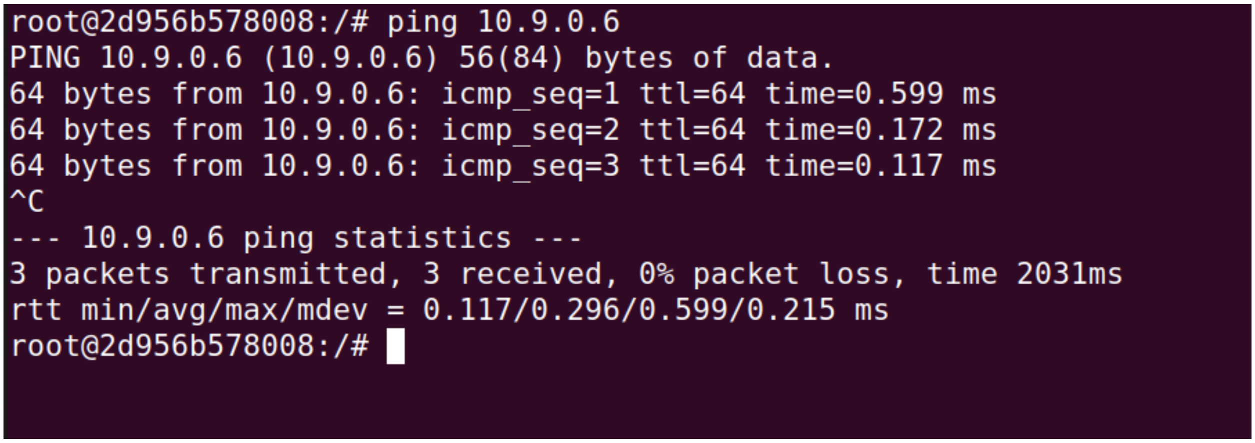
To capture only the icmp packet a filter should be applied in the sniff function. In the filter I mentioned which packets the attacker has to capture from the host.

–Attacking code:



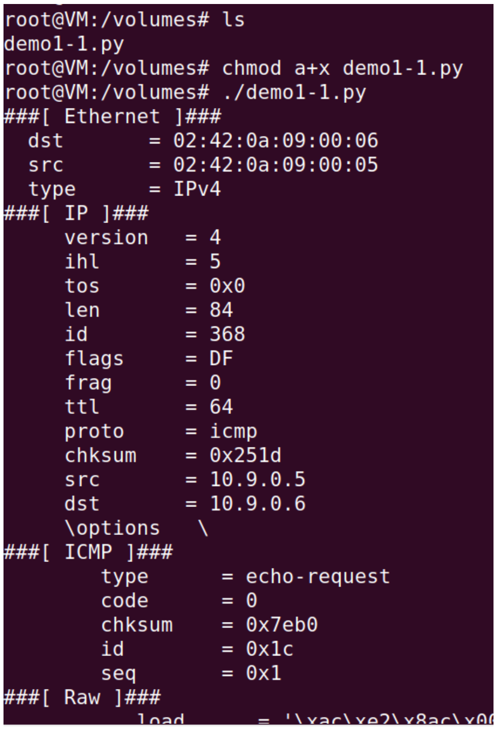
```
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
        pkt.show()
pkt = sniff(iface="br-4ab49ca38617", filter="icmp", prn=print_pkt)
```

–HostA terminal:



```
root@2d956b578008:/# 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=0.599 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.172 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.117 ms
^C
--- 10.9.0.6 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2031ms
rtt min/avg/max/mdev = 0.117/0.296/0.599/0.215 ms
root@2d956b578008:/# 
```

–Attacker terminal:



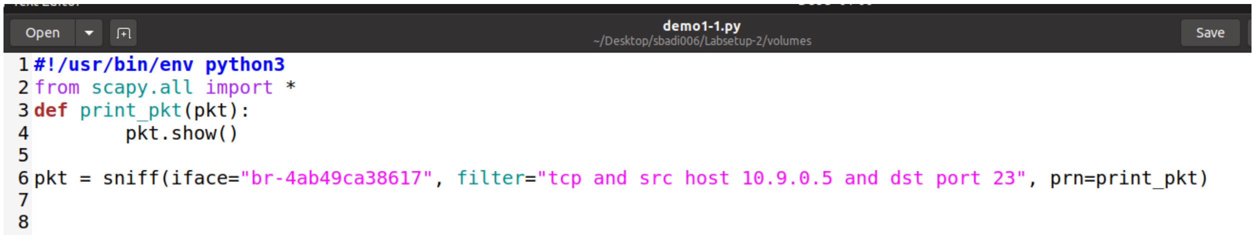
```
root@VM:/volumes# ls
demo1-1.py
root@VM:/volumes# chmod a+x demo1-1.py
root@VM:/volumes# ./demo1-1.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 368
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x251d
     src       = 10.9.0.5
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x7eb0
        id        = 0x1c
        seq       = 0x1
###[ Raw ]###
           load      = '\xac\xe2\x8ac\x00
```

The attacker was able to capture all the icmp packets being transmitted by hostA to hostB.

Task-1.1B: Capture any TCP packet that comes from a particular IP and with a destination port number 23

To capture TCP packets the filter in sniff function has been changed to ‘tcp’. The IP it listens to is the hostA ip address i.e, 10.9.0.5 and the destination port was also mentioned as 23 in the filter.

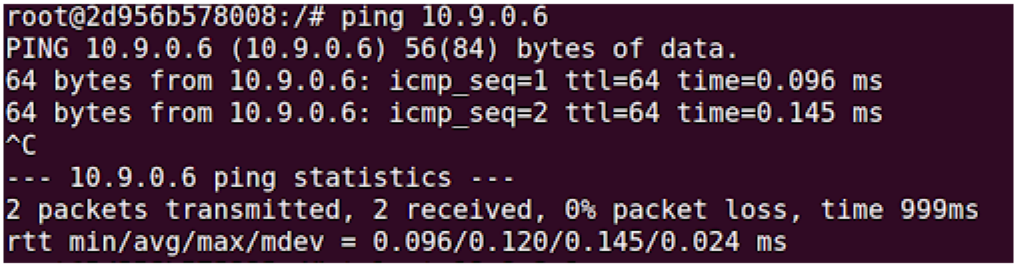
–Code:



```
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
        pkt.show()

pkt = sniff(iface="br-4ab49ca38617", filter="tcp and src host 10.9.0.5 and dst port 23", prn=print_pkt)


```

–HostA terminal:

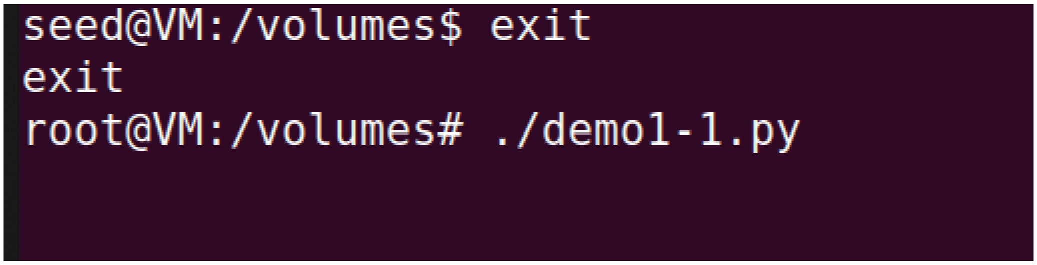
With the general ping command from the host, tcp packets cannot be read.



```
root@2d956b578008:/# 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=0.096 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.145 ms
^C
--- 10.9.0.6 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.096/0.120/0.145/0.024 ms
```

–Attacker Terminal:

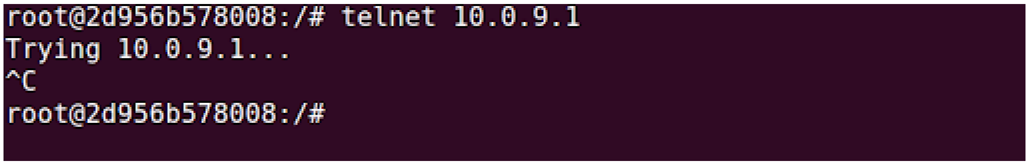
Attacker could not sniff the transmitted tcp packets as we did not use the right command.



```
seed@VM:/volumes$ exit
exit
root@VM:/volumes# ./demo1-1.py
```

–HostA terminal:

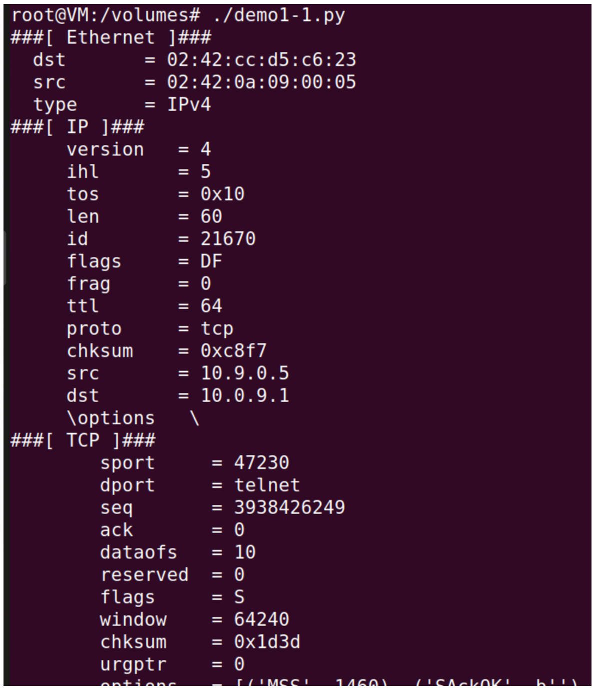
Using a telnet command which uses the attacker address.



```
root@2d956b578008:/# telnet 10.0.9.1
Trying 10.0.9.1...
^C
root@2d956b578008:/#
```

–Attacker terminal:

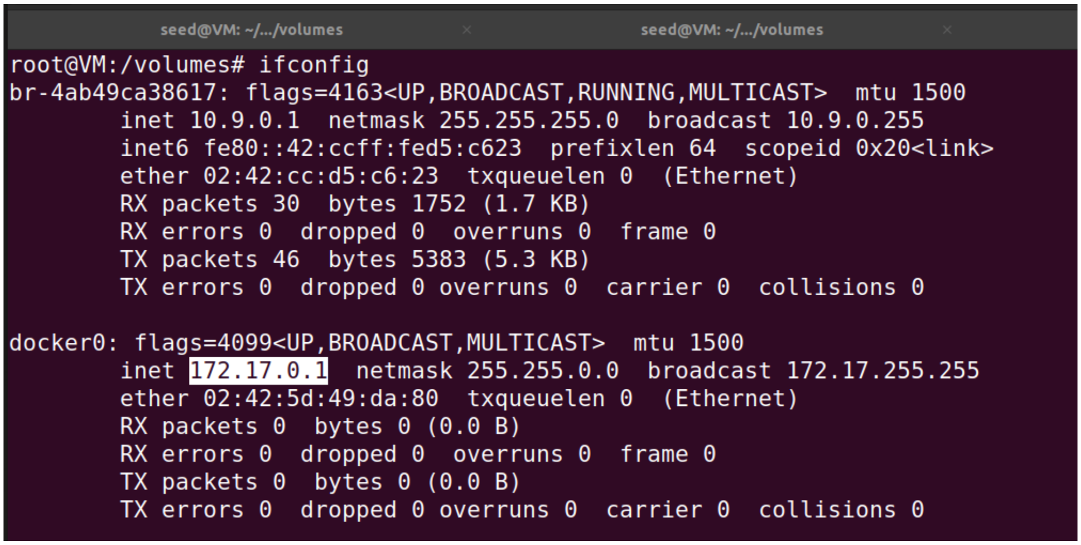
Attacker was able to sniff the transmitted tcp packets when the host used telnet command instead of ping to connect.



```
root@VM:/volumes# ./demo1-1.py
###[ Ethernet ]###
  dst       = 02:42:cc:d5:c6:23
  src       = 02:42:0a:09:00:05
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x10
     len       = 60
     id        = 21670
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0xc8f7
     src       = 10.9.0.5
     dst       = 10.0.9.1
     \options   \
###[ TCP ]###
        sport     = 47230
        dport     = telnet
        seq       = 3938426249
        ack       = 0
        dataofs   = 10
        reserved  = 0
        flags     = S
        window    = 64240
        chksum    = 0x1d3d
        urgptr    = 0
        options   = [('MSS', 1460), ('SAckOK', b'')
```

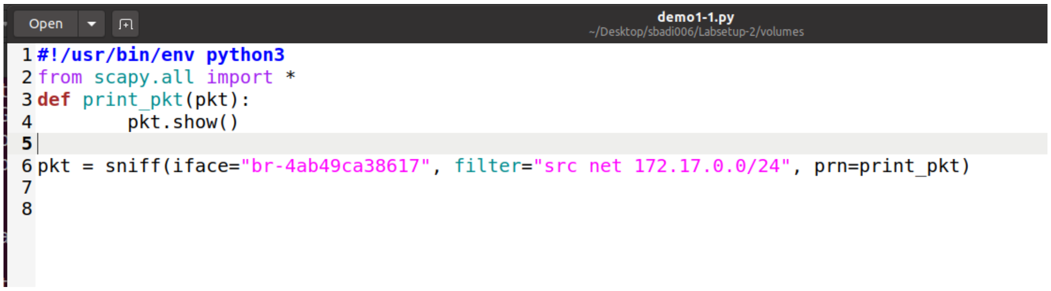
Task-1.1B Capture packets comes from or to go to a particular subnet:

Attacker ifconfig



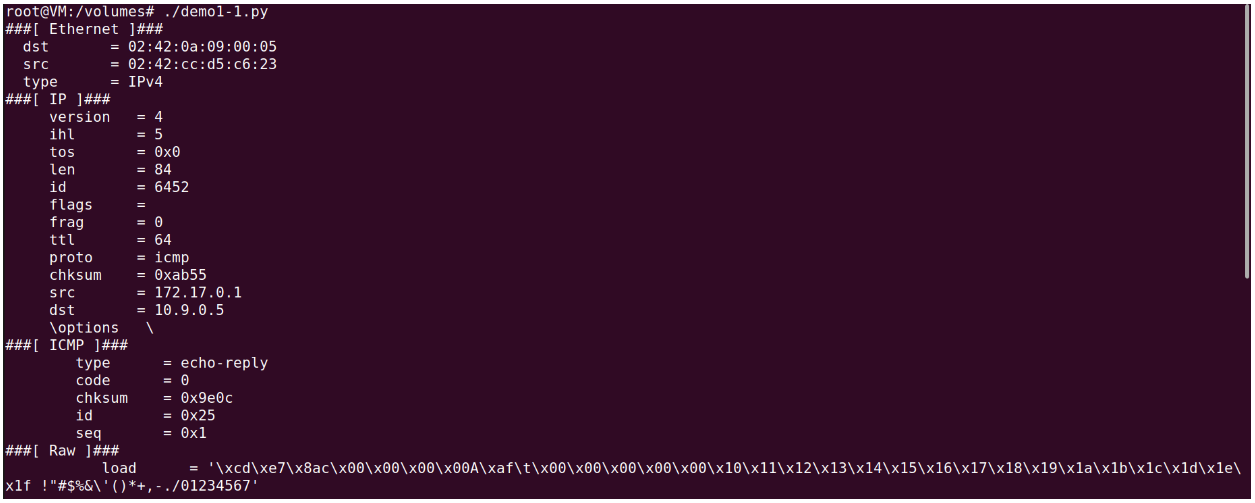
```
root@VM:/volumes# ifconfig
br-4ab49ca38617: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:ccff:fed5:c623  prefixlen 64  scopeid 0x20<link>
        ether 02:42:cc:d5:c6:23  txqueuelen 0  (Ethernet)
        RX packets 30  bytes 1752 (1.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 46  bytes 5383 (5.3 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 02:42:5d:49:da:80  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
```

Code:



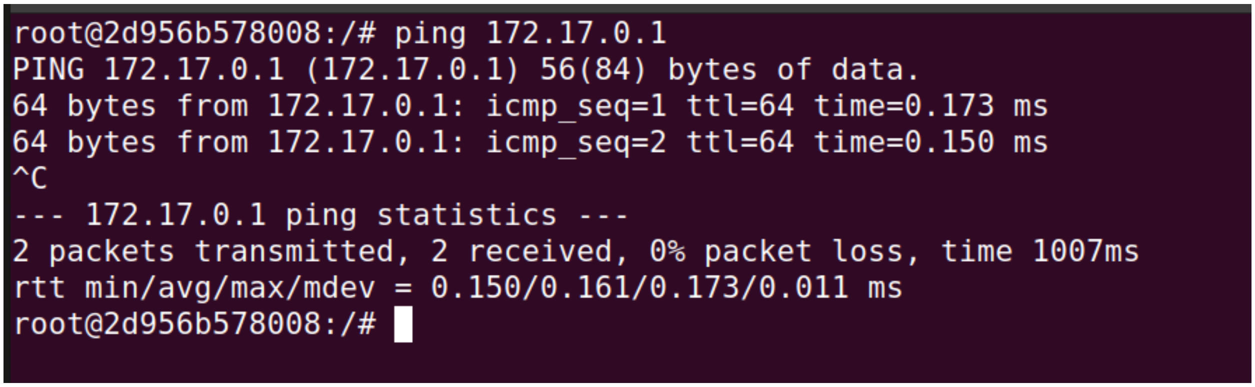
```python
#!/usr/bin/env python3
from scapy.all import *
def print_pkt(pkt):
        pkt.show()

pkt = sniff(iface="br-4ab49ca38617", filter="src net 172.17.0.0/24", prn=print_pkt)


```

Attacker:



```
root@VM:/volumes# ./demo1-1.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:cc:d5:c6:23
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 6452
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xab55
     src       = 172.17.0.1
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0x9e0c
        id        = 0x25
        seq       = 0x1
###[ Raw ]###
           load      = '\xcd\xe7\x8ac\x00\x00\x00\x00A\xaf\t\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\
x1f !"#$%&\'()*+,-./01234567'
```

Host:

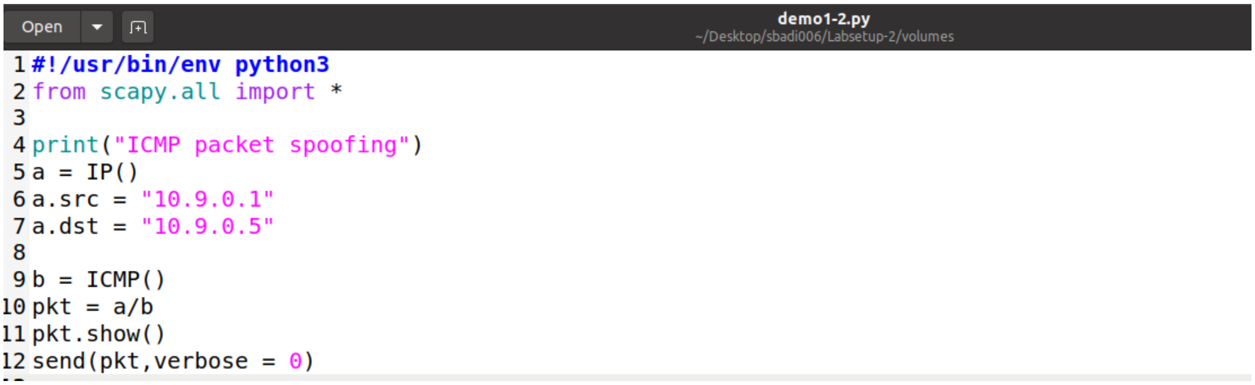


```
root@2d956b578008:/# ping 172.17.0.1
PING 172.17.0.1 (172.17.0.1) 56(84) bytes of data.
64 bytes from 172.17.0.1: icmp_seq=1 ttl=64 time=0.173 ms
64 bytes from 172.17.0.1: icmp_seq=2 ttl=64 time=0.150 ms
^C
--- 172.17.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1007ms
rtt min/avg/max/mdev = 0.150/0.161/0.173/0.011 ms
root@2d956b578008:/# 
```

Task 1.2: Spoofing ICMP Packets

To compare how scapy works with fields of IP packets set to arbitrary values, the scapy code was run with fields of IP packets set to known values. (hostA and attacker IP addresses)

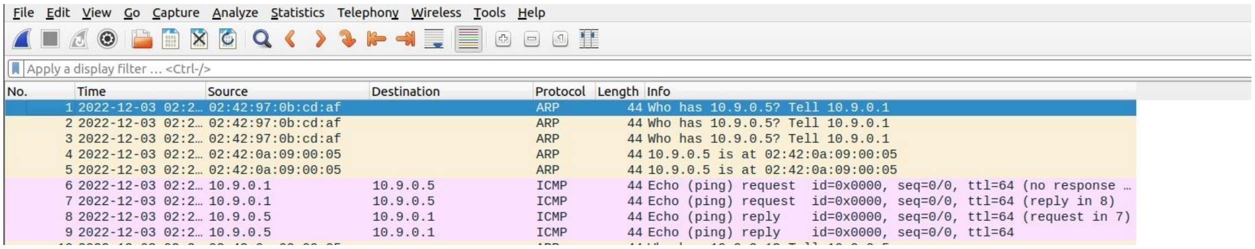
– Code with valid values:



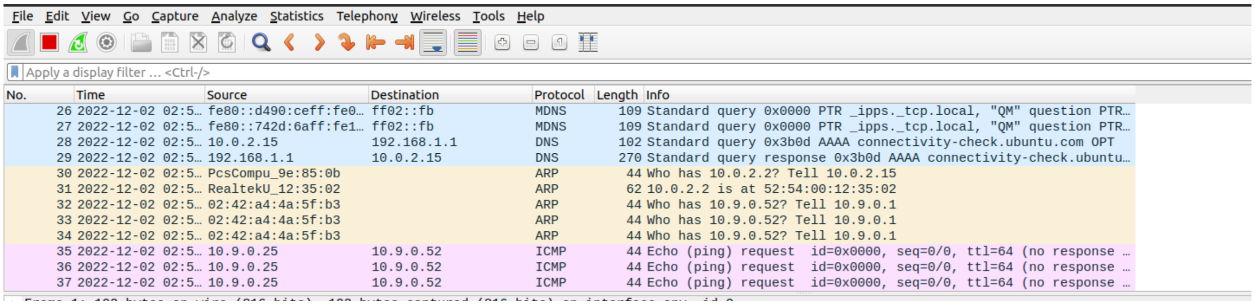
```python
#!/usr/bin/env python3
from scapy.all import *

print("ICMP packet spoofing")
a = IP()
a.src = "10.9.0.1"
a.dst = "10.9.0.5"

b = ICMP()
pkt = a/b
pkt.show()
send(pkt,verbose = 0)
```

–Wireshark:

As you can see when I run wireshark, it documents the packages being transmitted.


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

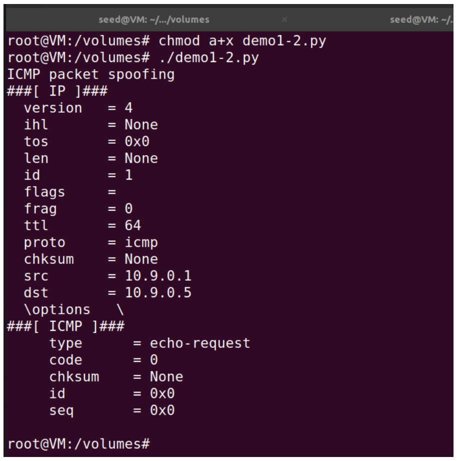
No.	Time	Source	Destination	Protocol	Length	Info
1	2022-12-03 02:2…	02:42:97:0b:cd:af		ARP	44	Who has 10.9.0.5? Tell 10.9.0.1
2	2022-12-03 02:2…	02:42:97:0b:cd:af		ARP	44	Who has 10.9.0.5? Tell 10.9.0.1
3	2022-12-03 02:2…	02:42:97:0b:cd:af		ARP	44	Who has 10.9.0.5? Tell 10.9.0.1
4	2022-12-03 02:2…	02:42:0a:09:00:05		ARP	44	10.9.0.5 is at 02:42:0a:09:00:05
5	2022-12-03 02:2…	02:42:0a:09:00:05		ARP	44	10.9.0.5 is at 02:42:0a:09:00:05
6	2022-12-03 02:2…	10.9.0.1	10.9.0.5	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response …
7	2022-12-03 02:2…	10.9.0.1	10.9.0.5	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 8)
8	2022-12-03 02:2…	10.9.0.5	10.9.0.1	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 7)
9	2022-12-03 02:2…	10.9.0.5	10.9.0.1	ICMP	44	Echo (ping) reply id=0x0000, seq=0/0, ttl=64


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
26	2022-12-02 02:5…	fe80::d490:ceff:fe0…	ff02::fb	MDNS	109	Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR…
27	2022-12-02 02:5…	fe80::742d:6aff:fe1…	ff02::fb	MDNS	109	Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR…
28	2022-12-02 02:5…	10.0.2.15	192.168.1.1	DNS	102	Standard query 0x3b0d AAAA connectivity-check.ubuntu.com OPT
29	2022-12-02 02:5…	192.168.1.1	10.0.2.15	DNS	270	Standard query response 0x3b0d AAAA connectivity-check.ubuntu…
30	2022-12-02 02:5…	PcsCompu_9e:85:0b		ARP	44	Who has 10.0.2.2? Tell 10.0.2.15
31	2022-12-02 02:5…	RealtekU_12:35:02		ARP	62	10.0.2.2 is at 52:54:00:12:35:02
32	2022-12-02 02:5…	02:42:a4:4a:5f:b3		ARP	44	Who has 10.9.0.52? Tell 10.9.0.1
33	2022-12-02 02:5…	02:42:a4:4a:5f:b3		ARP	44	Who has 10.9.0.52? Tell 10.9.0.1
34	2022-12-02 02:5…	02:42:a4:4a:5f:b3		ARP	44	Who has 10.9.0.52? Tell 10.9.0.1
35	2022-12-02 02:5…	10.9.0.25	10.9.0.52	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response …
36	2022-12-02 02:5…	10.9.0.25	10.9.0.52	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response …
37	2022-12-02 02:5…	10.9.0.25	10.9.0.52	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (no response …

Attacker:

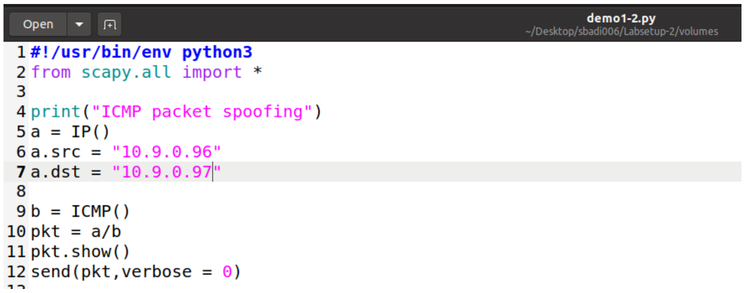
The attacker was also able to fetch the transmitted packets.



```
root@VM:/volumes# chmod a+x demo1-2.py
root@VM:/volumes# ./demo1-2.py
ICMP packet spoofing
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = None
  src       = 10.9.0.1
  dst       = 10.9.0.5
  \options   \
###[ ICMP ]###
     type      = echo-request
     code      = 0
     chksum    = None
     id        = 0x0
     seq       = 0x0

root@VM:/volumes#
```

Now I am saving arbitrary values instead of known.

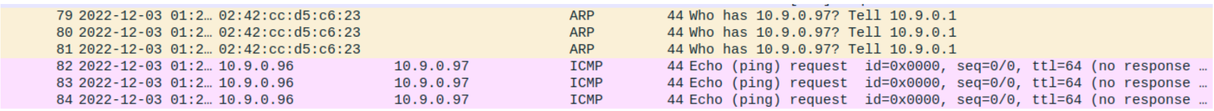
–Code:



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

print("ICMP packet spoofing")
a = IP()
a.src = "10.9.0.96"
a.dst = "10.9.0.97"

b = ICMP()
pkt = a/b
pkt.show()
send(pkt,verbose = 0)
```

–Wireshark:

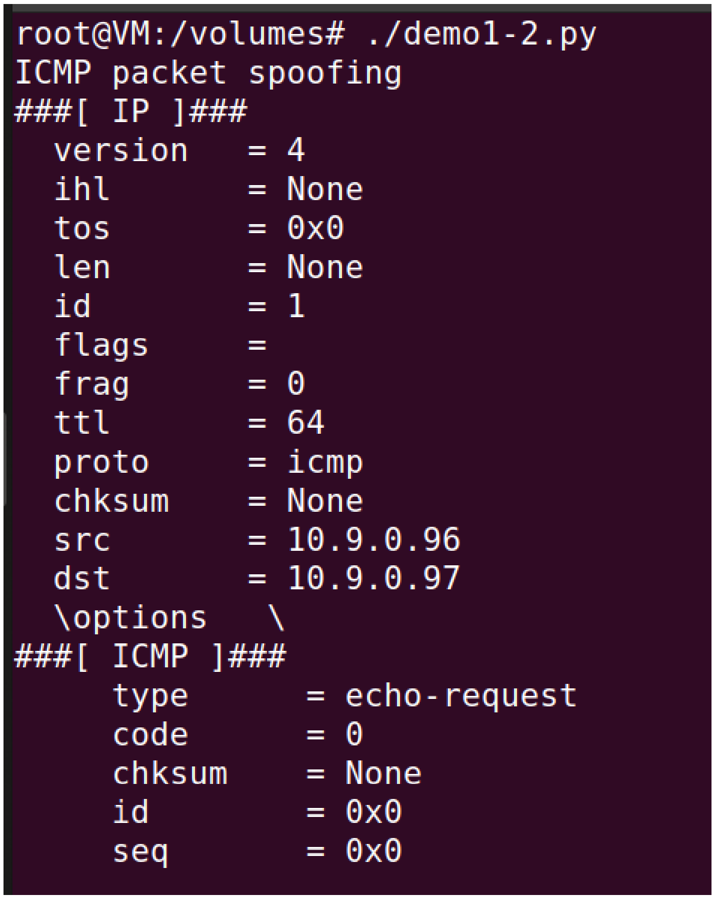
Wireshark will only document requests and no responses as the ip addresses used were invalid.



```
79 2022-12-03 01:2… 02:42:cc:d5:c6:23                    ARP        44 Who has 10.9.0.97? Tell 10.9.0.1
80 2022-12-03 01:2… 02:42:cc:d5:c6:23                    ARP        44 Who has 10.9.0.97? Tell 10.9.0.1
81 2022-12-03 01:2… 02:42:cc:d5:c6:23                    ARP        44 Who has 10.9.0.97? Tell 10.9.0.1
82 2022-12-03 01:2… 10.9.0.96          10.9.0.97         ICMP       44 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response …
83 2022-12-03 01:2… 10.9.0.96          10.9.0.97         ICMP       44 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response …
84 2022-12-03 01:2… 10.9.0.96          10.9.0.97         ICMP       44 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no response …
```

–Attacker:

The attacker won't be able to capture the packets as the ip address does not exist



```
root@VM:/volumes# ./demo1-2.py
ICMP packet spoofing
###[ IP ]###
  version   = 4
  ihl       = None
  tos       = 0x0
  len       = None
  id        = 1
  flags     =
  frag      = 0
  ttl       = 64
  proto     = icmp
  chksum    = None
  src       = 10.9.0.96
  dst       = 10.9.0.97
  \options   \
###[ ICMP ]###
     type      = echo-request
     code      = 0
     chksum    = None
     id        = 0x0
     seq       = 0x0
```

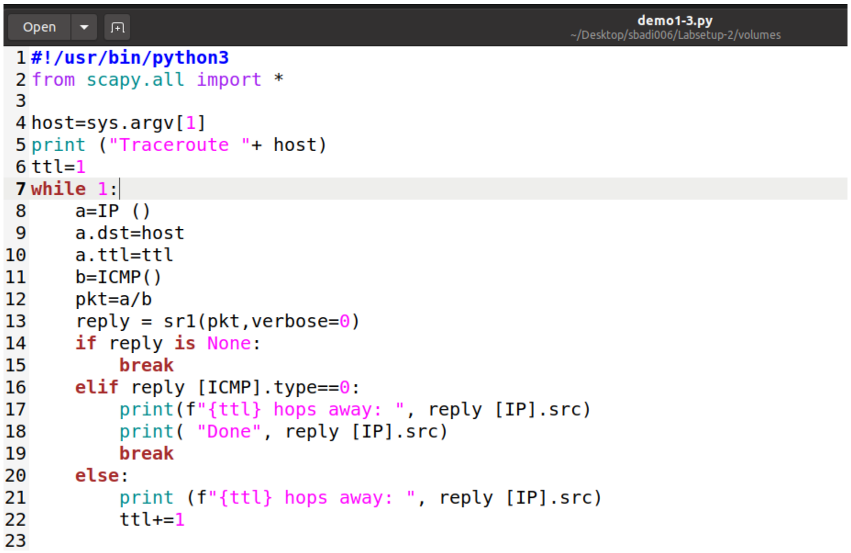
Task-1.3 Traceroute:

In this task we are using Scapy to estimate the distance, in terms of number of routers, between our VM and a selected destination

–Code:

The below code was used to estimate the distance between the VM and destination, in our case is hostA and google.com website.

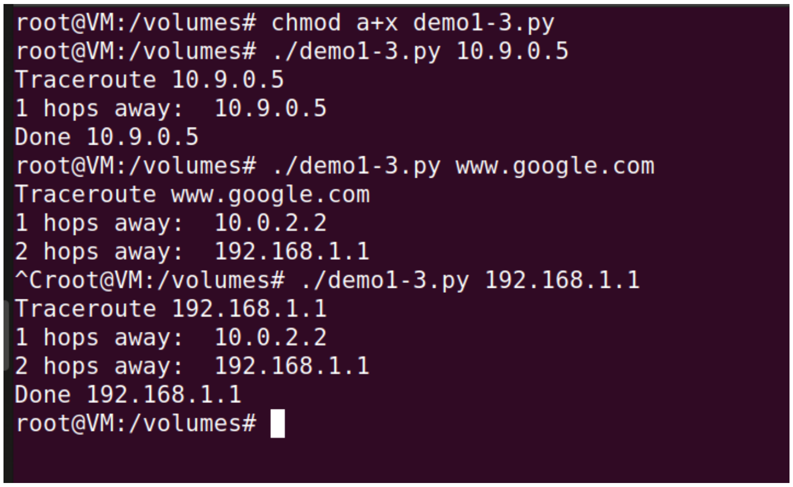
The code uses functions like srl(waits for destinations reply) by setting time to live factor to 1 and running an infinite loop to create and send packets till the destination is 0 hops away.



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

host=sys.argv[1]
print ("Traceroute "+ host)
ttl=1
while 1:
    a=IP ()
    a.dst=host
    a.ttl=ttl
    b=ICMP()
    pkt=a/b
    reply = sr1(pkt,verbose=0)
    if reply is None:
        break
    elif reply [ICMP].type==0:
        print(f"{ttl} hops away: ", reply [IP].src)
        print( "Done", reply [IP].src)
        break
    else:
        print (f"{ttl} hops away: ", reply [IP].src)
        ttl+=1

```

–Attacker:

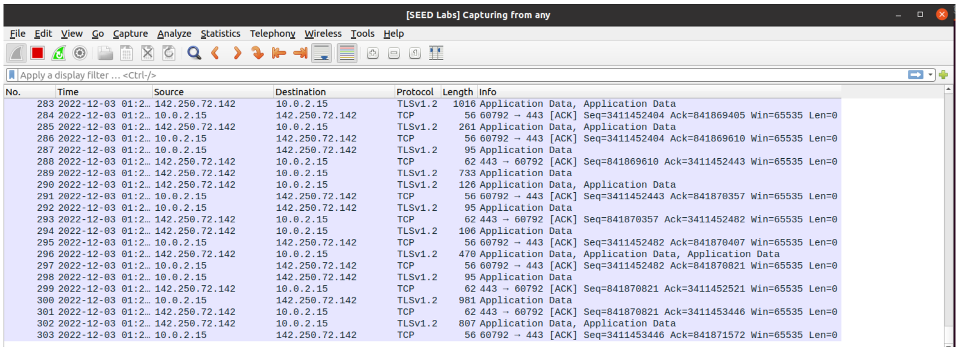
This code was implemented or tested with two inputs: one with hostA IP address and ‘www.google.com’ as destinations. The output is as follows.



```
root@VM:/volumes# chmod a+x demo1-3.py
root@VM:/volumes# ./demo1-3.py 10.9.0.5
Traceroute 10.9.0.5
1 hops away:  10.9.0.5
Done 10.9.0.5
root@VM:/volumes# ./demo1-3.py www.google.com
Traceroute www.google.com
1 hops away:  10.0.2.2
2 hops away:  192.168.1.1
^Croot@VM:/volumes# ./demo1-3.py 192.168.1.1
Traceroute 192.168.1.1
1 hops away:  10.0.2.2
2 hops away:  192.168.1.1
Done 192.168.1.1
root@VM:/volumes# 
```

–Wireshark:

Wireshark will document the transmitted packets.


[SEED Labs] Capturing from any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

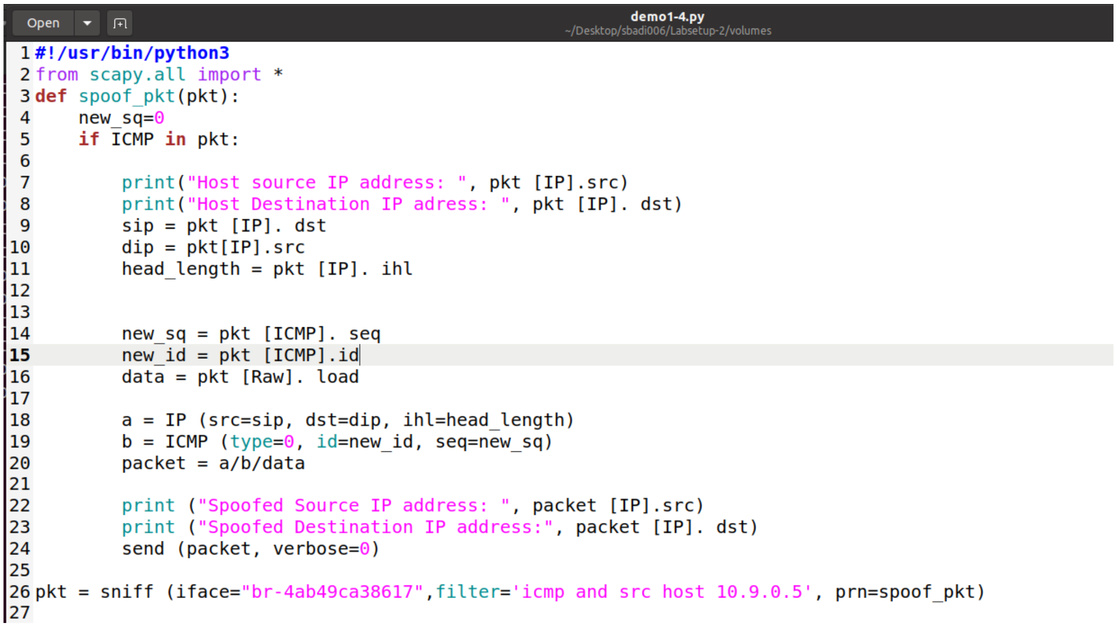
No.	Time	Source	Destination	Protocol	Length	Info
283	2022-12-03 01:2…	142.250.72.142	10.0.2.15	TLSv1.2	1016	Application Data, Application Data
284	2022-12-03 01:2…	10.0.2.15	142.250.72.142	TCP	56	60792 → 443 [ACK] Seq=3411452404 Ack=841869405 Win=65535 Len=0
285	2022-12-03 01:2…	142.250.72.142	10.0.2.15	TLSv1.2	261	Application Data, Application Data
286	2022-12-03 01:2…	10.0.2.15	142.250.72.142	TCP	56	60792 → 443 [ACK] Seq=3411452404 Ack=841869610 Win=65535 Len=0
287	2022-12-03 01:2…	10.0.2.15	142.250.72.142	TLSv1.2	95	Application Data
288	2022-12-03 01:2…	142.250.72.142	10.0.2.15	TCP	62	443 → 60792 [ACK] Seq=841869610 Ack=3411452443 Win=65535 Len=0
289	2022-12-03 01:2…	142.250.72.142	10.0.2.15	TLSv1.2	733	Application Data
290	2022-12-03 01:2…	142.250.72.142	10.0.2.15	TLSv1.2	126	Application Data, Application Data
291	2022-12-03 01:2…	10.0.2.15	142.250.72.142	TCP	56	60792 → 443 [ACK] Seq=3411452443 Ack=841870357 Win=65535 Len=0
292	2022-12-03 01:2…	10.0.2.15	142.250.72.142	TLSv1.2	95	Application Data
293	2022-12-03 01:2…	142.250.72.142	10.0.2.15	TCP	62	443 → 60792 [ACK] Seq=841870357 Ack=3411452482 Win=65535 Len=0
294	2022-12-03 01:2…	142.250.72.142	10.0.2.15	TLSv1.2	106	Application Data
295	2022-12-03 01:2…	10.0.2.15	142.250.72.142	TCP	56	60792 → 443 [ACK] Seq=3411452482 Ack=841870407 Win=65535 Len=0
296	2022-12-03 01:2…	142.250.72.142	10.0.2.15	TLSv1.2	470	Application Data, Application Data, Application Data
297	2022-12-03 01:2…	10.0.2.15	142.250.72.142	TCP	56	60792 → 443 [ACK] Seq=3411452482 Ack=841870821 Win=65535 Len=0
298	2022-12-03 01:2…	10.0.2.15	142.250.72.142	TLSv1.2	95	Application Data
299	2022-12-03 01:2…	142.250.72.142	10.0.2.15	TCP	62	443 → 60792 [ACK] Seq=841870821 Ack=3411452521 Win=65535 Len=0
300	2022-12-03 01:2…	10.0.2.15	142.250.72.142	TLSv1.2	981	Application Data
301	2022-12-03 01:2…	142.250.72.142	10.0.2.15	TCP	62	443 → 60792 [ACK] Seq=841870821 Ack=3411453446 Win=65535 Len=0
302	2022-12-03 01:2…	142.250.72.142	10.0.2.15	TLSv1.2	807	Application Data, Application Data
303	2022-12-03 01:2…	10.0.2.15	142.250.72.142	TCP	56	60792 → 443 [ACK] Seq=3411453446 Ack=841871572 Win=65535 Len=0

Task-1.4 Sniffing and-then Spoofing:

In this task, we will combine the sniffing and spoofing techniques to implement the following sniff-and then-spoof program.

–Code:

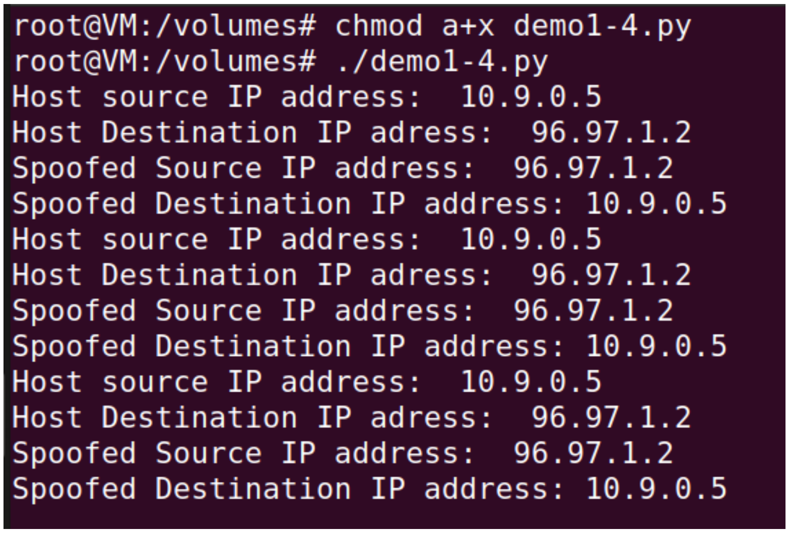
This program takes a sniffed packet as an argument and tries to send new packets back to the source. To do this, the code swaps the destination and source address and adds payload to the data. It will then send it back to the source.



```
#!/usr/bin/python3
from scapy.all import *
def spoof_pkt(pkt):
    new_sq=0
    if ICMP in pkt:

        print("Host source IP address: ", pkt [IP].src)
        print("Host Destination IP adress: ", pkt [IP]. dst)
        sip = pkt [IP]. dst
        dip = pkt[IP].src
        head_length = pkt [IP]. ihl


        new_sq = pkt [ICMP]. seq
        new_id = pkt [ICMP].id
        data = pkt [Raw]. load

        a = IP (src=sip, dst=dip, ihl=head_length)
        b = ICMP (type=0, id=new_id, seq=new_sq)
        packet = a/b/data

        print ("Spoofed Source IP address: ", packet [IP].src)
        print ("Spoofed Destination IP address:", packet [IP]. dst)
        send (packet, verbose=0)

pkt = sniff (iface="br-4ab49ca38617",filter='icmp and src host 10.9.0.5', prn=spoof_pkt)
```

–Attacker:

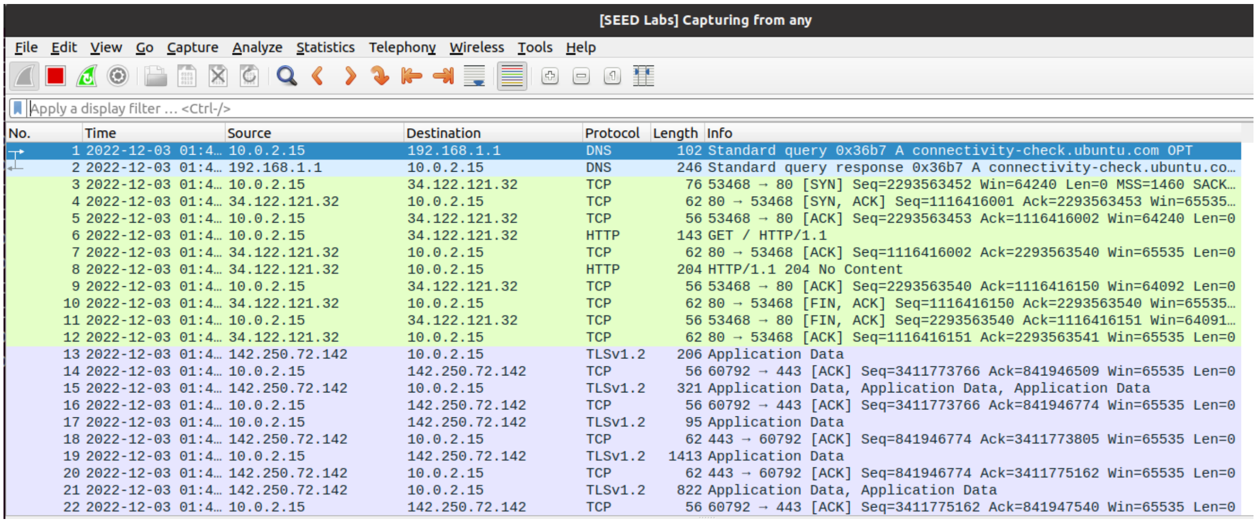
As you can see the attacker successfully sent the packets back to the source from the destination of the spoofed packets.



```
root@VM:/volumes# chmod a+x demo1-4.py
root@VM:/volumes# ./demo1-4.py
Host source IP address:  10.9.0.5
Host Destination IP adress:  96.97.1.2
Spoofed Source IP address:  96.97.1.2
Spoofed Destination IP address: 10.9.0.5
Host source IP address:  10.9.0.5
Host Destination IP adress:  96.97.1.2
Spoofed Source IP address:  96.97.1.2
Spoofed Destination IP address: 10.9.0.5
Host source IP address:  10.9.0.5
Host Destination IP adress:  96.97.1.2
Spoofed Source IP address:  96.97.1.2
Spoofed Destination IP address: 10.9.0.5
```

–Wireshark:

Wireshark also documents the same.


[SEED Labs] Capturing from any
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	2022-12-03 01:4…	10.0.2.15	192.168.1.1	DNS	102	Standard query 0x36b7 A connectivity-check.ubuntu.com OPT
2	2022-12-03 01:4…	192.168.1.1	10.0.2.15	DNS	246	Standard query response 0x36b7 A connectivity-check.ubuntu.co…
3	2022-12-03 01:4…	10.0.2.15	34.122.121.32	TCP	76	53468 → 80 [SYN] Seq=2293563452 Win=64240 Len=0 MSS=1460 SACK…
4	2022-12-03 01:4…	34.122.121.32	10.0.2.15	TCP	62	80 → 53468 [SYN, ACK] Seq=1116416001 Ack=2293563453 Win=65535…
5	2022-12-03 01:4…	10.0.2.15	34.122.121.32	TCP	56	53468 → 80 [ACK] Seq=2293563453 Ack=1116416002 Win=64240 Len=0
6	2022-12-03 01:4…	10.0.2.15	34.122.121.32	HTTP	143	GET / HTTP/1.1
7	2022-12-03 01:4…	34.122.121.32	10.0.2.15	TCP	62	80 → 53468 [ACK] Seq=1116416002 Ack=2293563540 Win=65535 Len=0
8	2022-12-03 01:4…	34.122.121.32	10.0.2.15	HTTP	204	HTTP/1.1 204 No Content
9	2022-12-03 01:4…	10.0.2.15	34.122.121.32	TCP	56	53468 → 80 [ACK] Seq=2293563540 Ack=1116416150 Win=64092 Len=0
10	2022-12-03 01:4…	34.122.121.32	10.0.2.15	TCP	62	80 → 53468 [FIN, ACK] Seq=1116416150 Ack=2293563540 Win=65535…
11	2022-12-03 01:4…	10.0.2.15	34.122.121.32	TCP	56	53468 → 80 [FIN, ACK] Seq=2293563540 Ack=1116416151 Win=64091…
12	2022-12-03 01:4…	34.122.121.32	10.0.2.15	TCP	62	80 → 53468 [ACK] Seq=1116416151 Ack=2293563541 Win=65535 Len=0
13	2022-12-03 01:4…	142.250.72.142	10.0.2.15	TLSv1.2	206	Application Data
14	2022-12-03 01:4…	10.0.2.15	142.250.72.142	TCP	56	60792 → 443 [ACK] Seq=3411773766 Ack=841946509 Win=65535 Len=0
15	2022-12-03 01:4…	142.250.72.142	10.0.2.15	TLSv1.2	321	Application Data, Application Data, Application Data
16	2022-12-03 01:4…	10.0.2.15	142.250.72.142	TCP	56	60792 → 443 [ACK] Seq=3411773766 Ack=841946774 Win=65535 Len=0
17	2022-12-03 01:4…	10.0.2.15	142.250.72.142	TLSv1.2	95	Application Data
18	2022-12-03 01:4…	142.250.72.142	10.0.2.15	TCP	62	443 → 60792 [ACK] Seq=841946774 Ack=3411773805 Win=65535 Len=0
19	2022-12-03 01:4…	10.0.2.15	142.250.72.142	TLSv1.2	1413	Application Data
20	2022-12-03 01:4…	142.250.72.142	10.0.2.15	TCP	62	443 → 60792 [ACK] Seq=841946774 Ack=3411775162 Win=65535 Len=0
21	2022-12-03 01:4…	142.250.72.142	10.0.2.15	TLSv1.2	822	Application Data, Application Data
22	2022-12-03 01:4…	10.0.2.15	142.250.72.142	TCP	56	60792 → 443 [ACK] Seq=3411775162 Ack=841947540 Win=65535 Len=0