Name- Harshit Agrawal SUID: 912538842

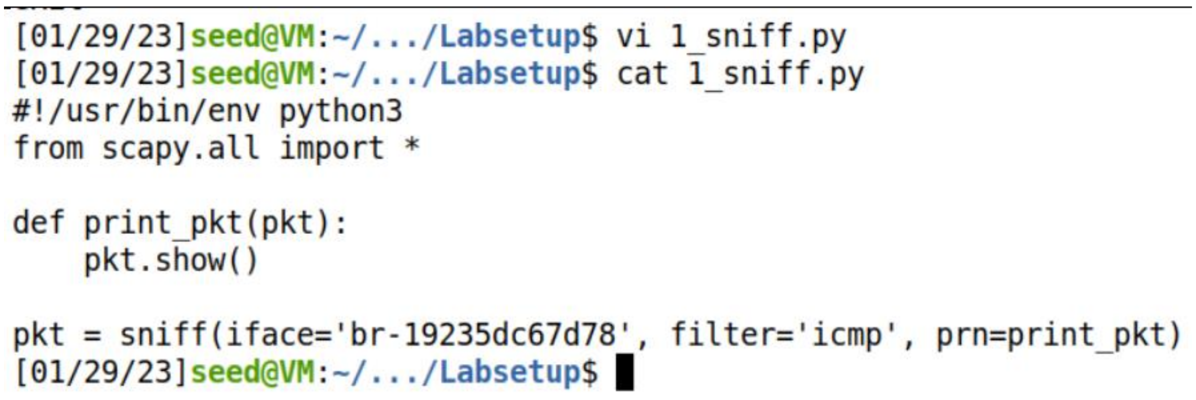
CIS – 644

Packet Sniffing and Spoofing Lab

Task 1: Using Scapy to Sniff and Spoof Packets

1.1 Sniffing Packet –

Using the below Scrapy library in the code to sniff the packets from the network.

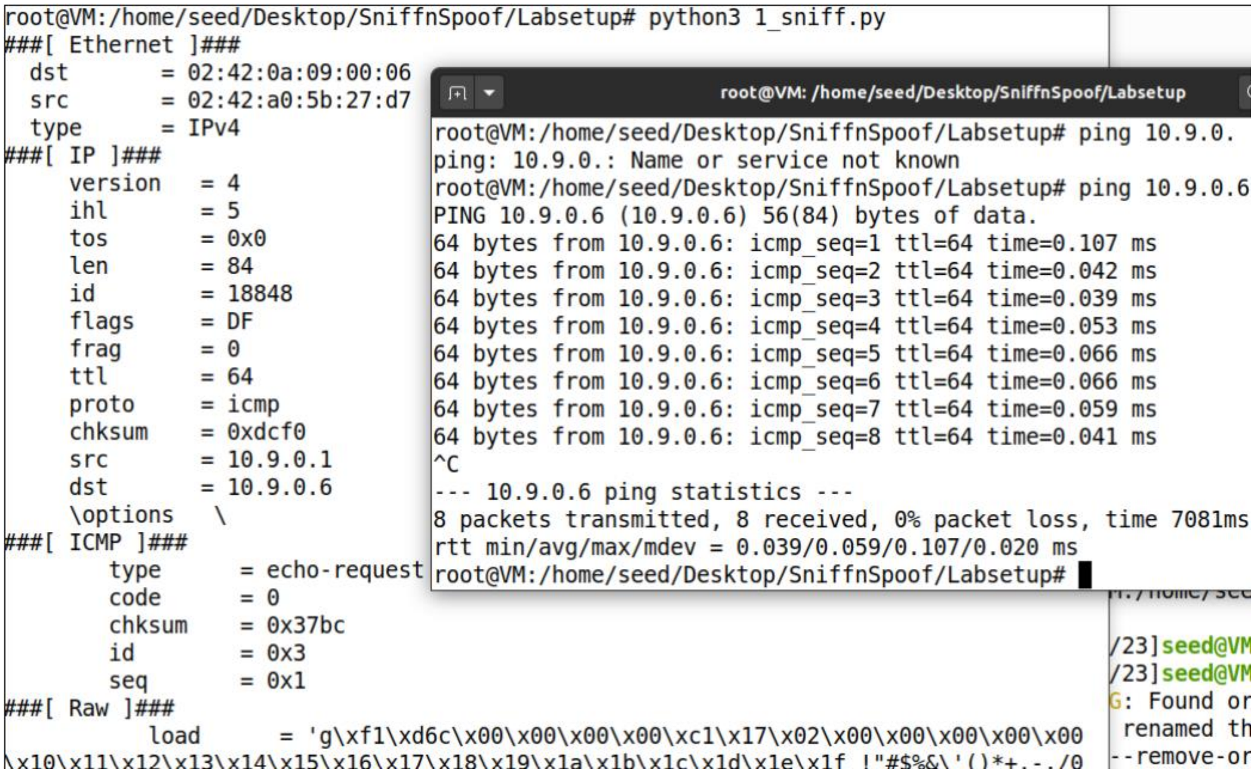


```
[01/29/23]seed@VM:~/.../Labsetup$ vi 1_sniff.py
[01/29/23]seed@VM:~/.../Labsetup$ cat 1_sniff.py
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-19235dc67d78', filter='icmp', prn=print_pkt)
[01/29/23]seed@VM:~/.../Labsetup$ 
```

We use the function sniff() to capture and monitor the interface on the ip addr 10.9.0.1 and filter the ICMP packets. As ‘ping’ uses ICMP packet to operate.

Task 1.1 A:

Initially we run this program with root privilege:



```
root@VM:/home/seed/Desktop/SniffnSpoof/Labsetup# python3 1_sniff.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:06
  src       = 02:42:a0:5b:27:d7
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 18848
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xdcf0
     src       = 10.9.0.1
     dst       = 10.9.0.6
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x37bc
        id        = 0x3
        seq       = 0x1
###[ Raw ]###
           load      = 'g\xf1\xd6c\x00\x00\x00\x00\xc1\x17\x02\x00\x00\x00\x00\x00
\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./0
```

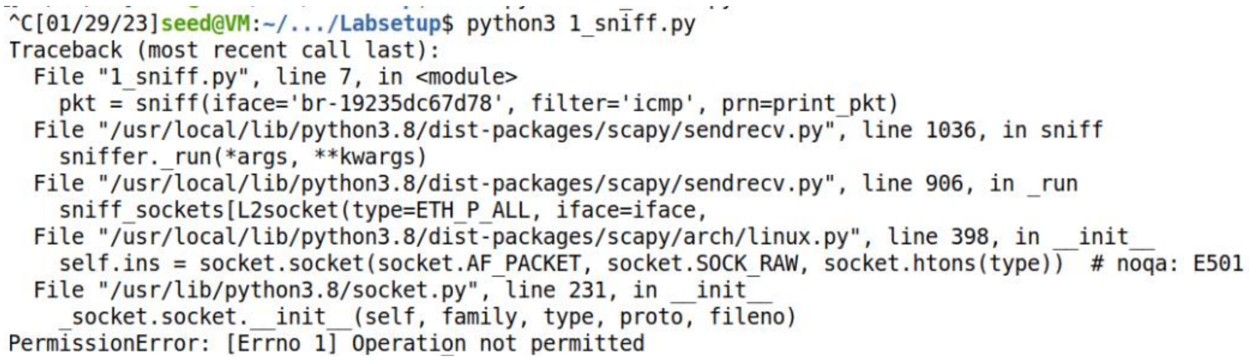
root@VM: /home/seed/Desktop/SniffnSpoof/Labsetup

```
root@VM:/home/seed/Desktop/SniffnSpoof/Labsetup# ping 10.9.0.
ping: 10.9.0.: Name or service not known
root@VM:/home/seed/Desktop/SniffnSpoof/Labsetup# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.107 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.042 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.039 ms
64 bytes from 10.9.0.6: icmp_seq=4 ttl=64 time=0.053 ms
64 bytes from 10.9.0.6: icmp_seq=5 ttl=64 time=0.066 ms
64 bytes from 10.9.0.6: icmp_seq=6 ttl=64 time=0.066 ms
64 bytes from 10.9.0.6: icmp_seq=7 ttl=64 time=0.059 ms
64 bytes from 10.9.0.6: icmp_seq=8 ttl=64 time=0.041 ms
^C
--- 10.9.0.6 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7081ms
rtt min/avg/max/mdev = 0.039/0.059/0.107/0.020 ms
root@VM:/home/seed/Desktop/SniffnSpoof/Labsetup#
```

```
/23]seed@VM
/23]seed@VM
G: Found or
 renamed th
--remove-or
```

This proves that we were able to capture the network packets when the root privileged prog was executed.

We use the same procedure without the root privileges and we can see the error of permissions which tells that it donot have the required privilege. As, when the sniff funtn is called it calls for raw socket initialization which also enables promiscuous mode. However OS do require the root privilege to execute this operation. Hence, we can see the following error.

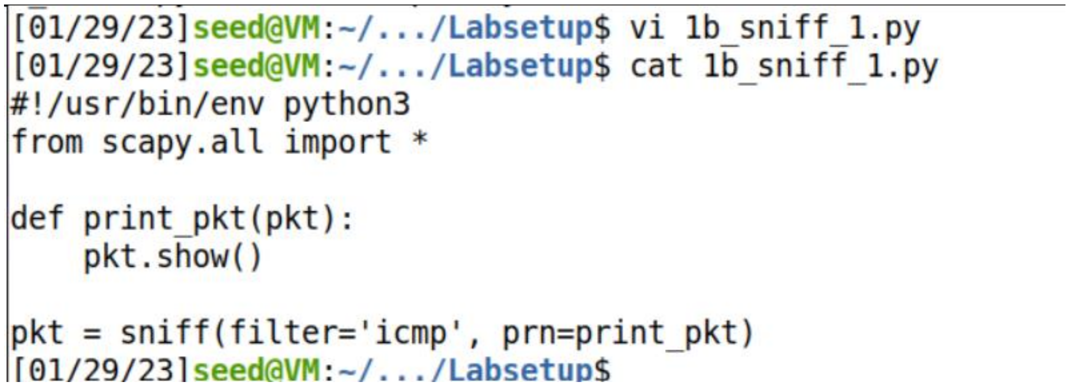


```
^C[01/29/23]seed@VM:~/.../Labsetup$ python3 1_sniff.py
Traceback (most recent call last):
  File "1_sniff.py", line 7, in <module>
    pkt = sniff(iface='br-19235dc67d78', filter='icmp', prn=print_pkt)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", line 1036, in sniff
    sniffer._run(*args, **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
```

Task 1.1 B:

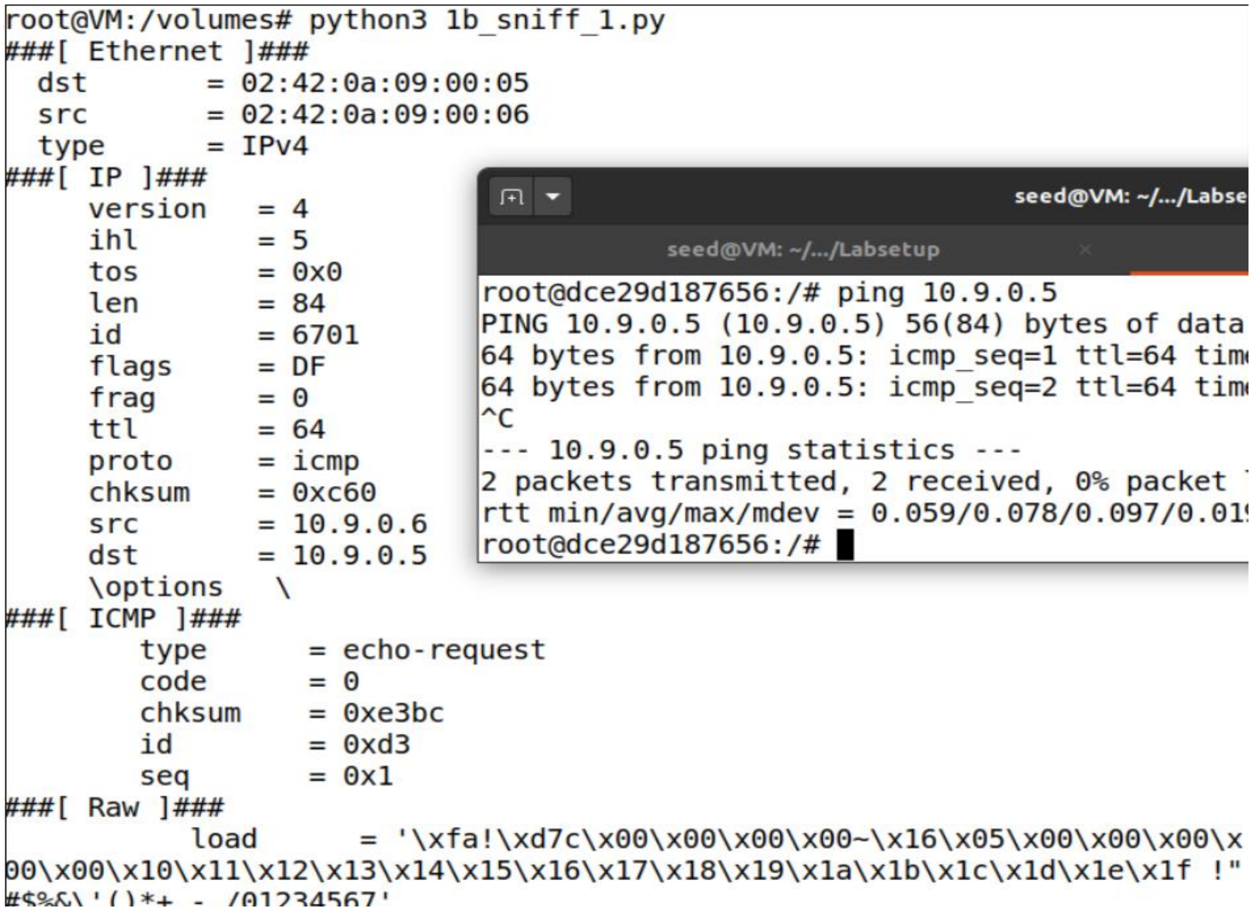
1) Capture only ICMP packet:

For this task we will be using the code given below:



```
[01/29/23]seed@VM:~/.../Labsetup$ vi 1b_sniff_1.py
[01/29/23]seed@VM:~/.../Labsetup$ cat 1b_sniff_1.py
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(filter='icmp', prn=print_pkt)
[01/29/23]seed@VM:~/.../Labsetup$
```

The program is executed with root privilege while ICMP packets was generated,
The below output would show the result of only ICMP packets were captured.

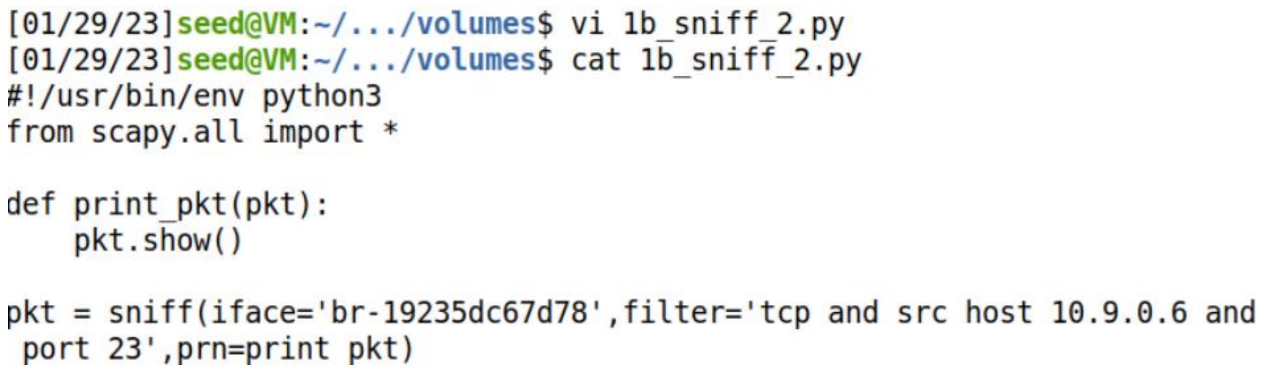


```
root@VM:/volumes# python3 1b_sniff_1.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 6701
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xc60
     src       = 10.9.0.6
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xe3bc
        id        = 0xd3
        seq       = 0x1
###[ Raw ]###
           load      = '\xfa!\xd7c\x00\x00\x00\x00~\x16\x05\x00\x00\x00\x
00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"
#$%&\'()*+,-./01234567'
```

```
root@dce29d187656:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 tim
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 tim
^C
--- 10.9.0.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet
rtt min/avg/max/mdev = 0.059/0.078/0.097/0.01
root@dce29d187656:/#
```

Above, we can see that the pung command was run by the host having the ip addr 10.9.0.6 to the host 10.9.0.5 and the sniffer program was executed on the attacker host end.

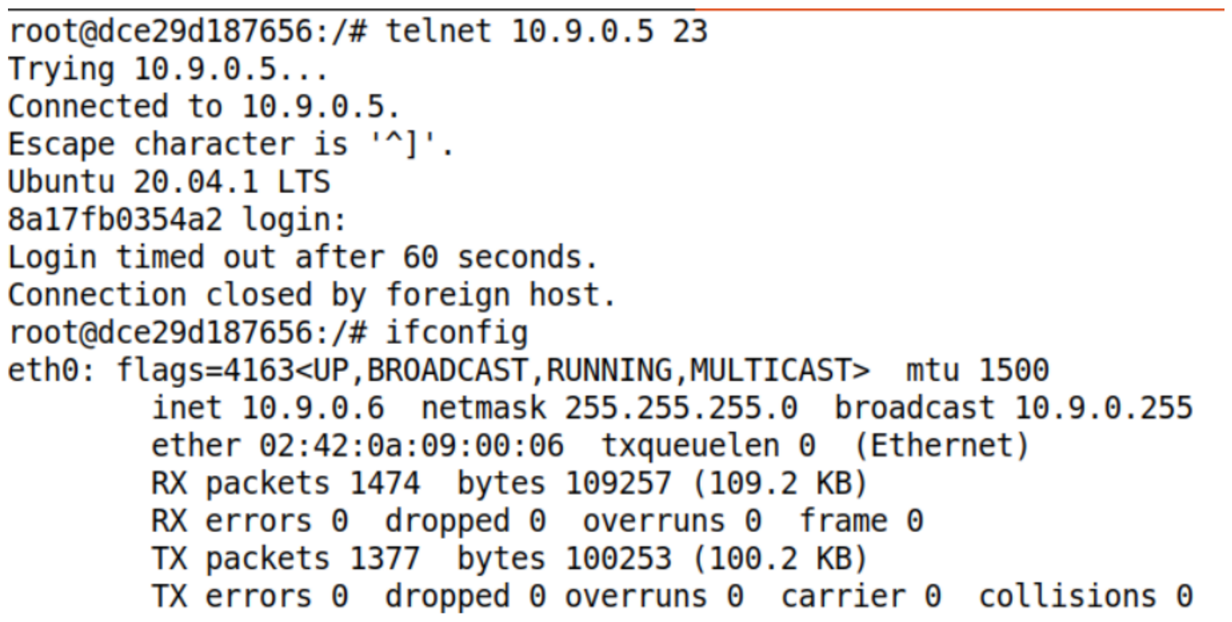
2) Capture any TCP packet that comes from a particular IP and with a destination port number 23.



```
[01/29/23]seed@VM:~/.../volumes$ vi 1b_sniff_2.py
[01/29/23]seed@VM:~/.../volumes$ cat 1b_sniff_2.py
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(iface='br-19235dc67d78',filter='tcp and src host 10.9.0.6 and
 port 23',prn=print_pkt)
```

This code captures the tcp packets that are coming from the ip addr 10.9.0.6 and trying to connect with port 23 of the destination host.

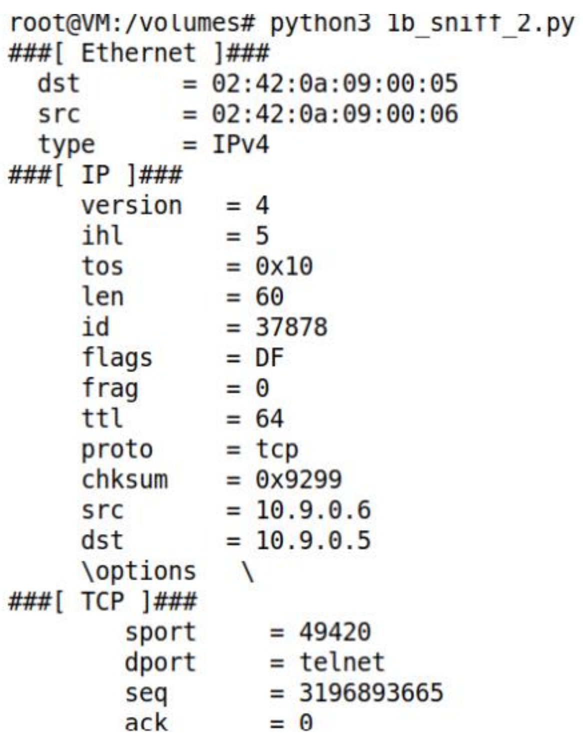
Now, initializing the telnet connection from the ip add 10.9.0.6



```
root@dce29d187656:/# telnet 10.9.0.5 23
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
8a17fb0354a2 login:
Login timed out after 60 seconds.
Connection closed by foreign host.
root@dce29d187656:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.6  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:06  txqueuelen 0  (Ethernet)
        RX packets 1474  bytes 109257 (109.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1377  bytes 100253 (100.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

The connection was initiated for the ip 10.9.0.5 which was on the port 23

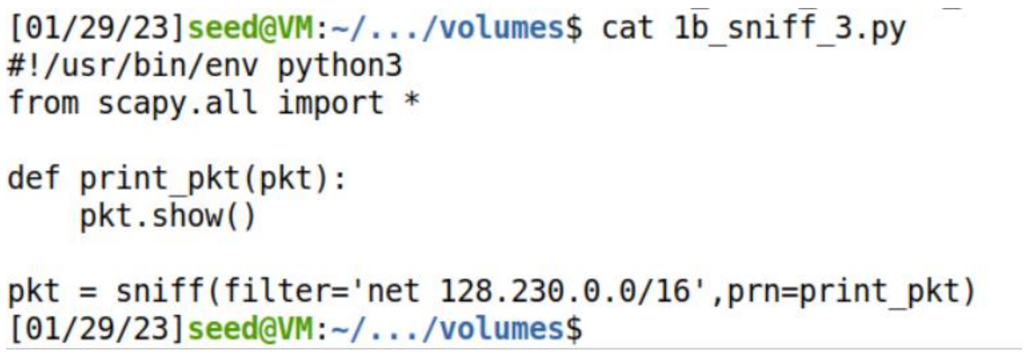
The attacker's system captures the packer as shown below:



```
root@VM:/volumes# python3 1b_sniff_2.py
###[ 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       = 60
     id        = 37878
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x9299
     src       = 10.9.0.6
     dst       = 10.9.0.5
     \options   \
###[ TCP ]###
        sport     = 49420
        dport     = telnet
        seq       = 3196893665
        ack       = 0
```

3) Capture packets comes from or to go to a particular subnet:

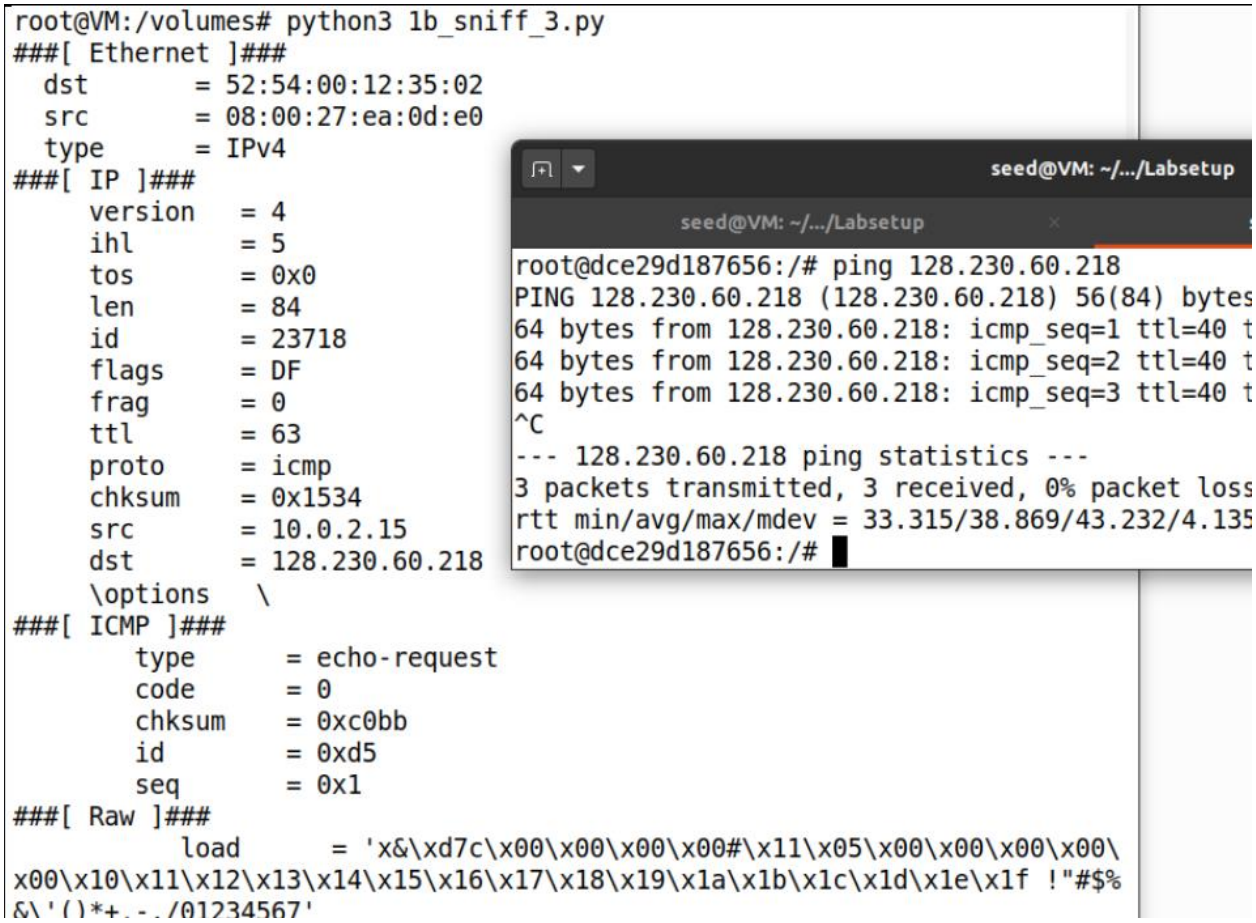
The code for this will be as given below:



```
[01/29/23]seed@VM:~/.../volumes$ cat 1b_sniff_3.py
#!/usr/bin/env python3
from scapy.all import *

def print_pkt(pkt):
    pkt.show()

pkt = sniff(filter='net 128.230.0.0/16',prn=print_pkt)
[01/29/23]seed@VM:~/.../volumes$
```

With the filter used above any connection that comes and goes from the 128.230.0.0/16 subnet will be captured.

The request for the ping will be made from the Host B(10.9.0.6) to 128.230.60.218 when also the sniffer program will be running on the attacker host side.

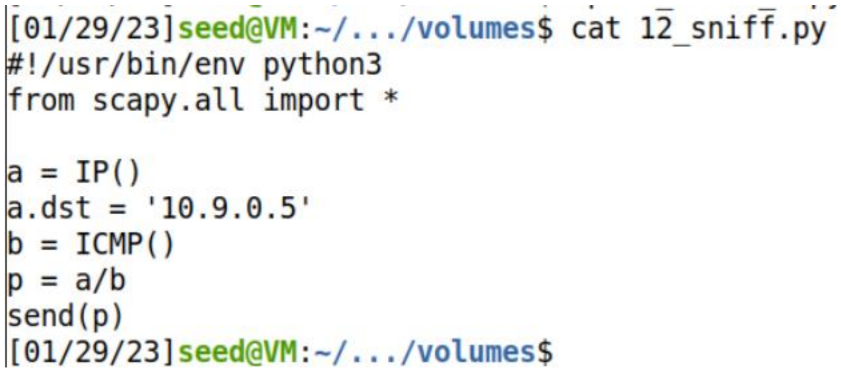


```
root@VM:/volumes# python3 1b_sniff_3.py
###[ Ethernet ]###
  dst       = 52:54:00:12:35:02
  src       = 08:00:27:ea:0d:e0
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 23718
     flags     = DF
     frag      = 0
     ttl       = 63
     proto     = icmp
     chksum    = 0x1534
     src       = 10.0.2.15
     dst       = 128.230.60.218
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xc0bb
        id        = 0xd5
        seq       = 0x1
###[ Raw ]###
           load      = 'x&\xd7c\x00\x00\x00\x00#\x11\x05\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

```
root@dce29d187656:/# ping 128.230.60.218
PING 128.230.60.218 (128.230.60.218) 56(84) bytes
64 bytes from 128.230.60.218: icmp_seq=1 ttl=40 t
64 bytes from 128.230.60.218: icmp_seq=2 ttl=40 t
64 bytes from 128.230.60.218: icmp_seq=3 ttl=40 t
^C
--- 128.230.60.218 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss
rtt min/avg/max/mdev = 33.315/38.869/43.232/4.135
root@dce29d187656:/#
```

1.2: Spoofing the ICMP Packets:

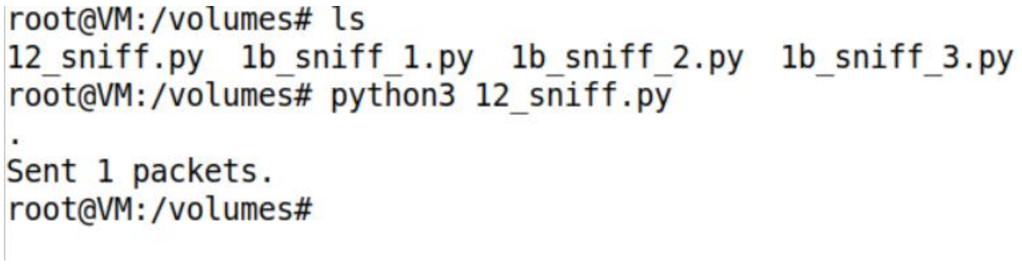
In this we will use the Scrapy to disguise the identity and also sent the packets out. As seen below:



```
[01/29/23]seed@VM:~/.../volumes$ cat 12_sniff.py
#!/usr/bin/env python3
from scapy.all import *

a = IP()
a.dst = '10.9.0.5'
b = ICMP()
p = a/b
send(p)
[01/29/23]seed@VM:~/.../volumes$
```

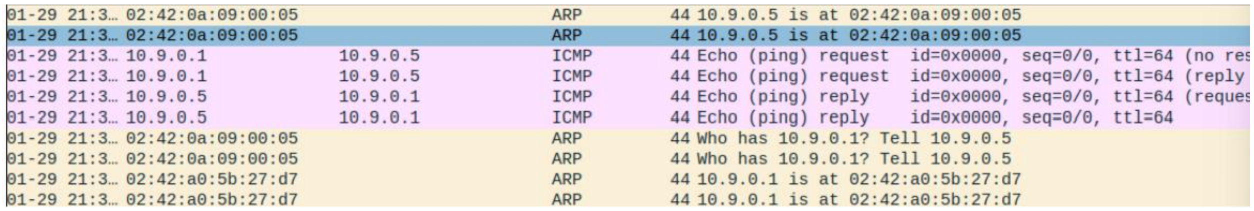
So, here a.dst is the destination ip addr where our packet will be sent. Also we have created the ICMP header type for using the overload operator that stacks the layers in p and then later also sends the packets.

Here, the spoofed packets were sent from the attacker's host machine:



```
root@VM:/volumes# ls
12_sniff.py  1b_sniff_1.py  1b_sniff_2.py  1b_sniff_3.py
root@VM:/volumes# python3 12_sniff.py
.
Sent 1 packets.
root@VM:/volumes#
```

This can also be confirmed from using the Wireshark tool.



```
01-29 21:3… 02:42:0a:09:00:05                    ARP    44 10.9.0.5 is at 02:42:0a:09:00:05
01-29 21:3… 02:42:0a:09:00:05                    ARP    44 10.9.0.5 is at 02:42:0a:09:00:05
01-29 21:3… 10.9.0.1          10.9.0.5           ICMP   44 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (no res
01-29 21:3… 10.9.0.1          10.9.0.5           ICMP   44 Echo (ping) request  id=0x0000, seq=0/0, ttl=64 (reply
01-29 21:3… 10.9.0.5          10.9.0.1           ICMP   44 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64 (reques
01-29 21:3… 10.9.0.5          10.9.0.1           ICMP   44 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64
01-29 21:3… 02:42:0a:09:00:05                    ARP    44 Who has 10.9.0.1? Tell 10.9.0.5
01-29 21:3… 02:42:0a:09:00:05                    ARP    44 Who has 10.9.0.1? Tell 10.9.0.5
01-29 21:3… 02:42:a0:5b:27:d7                    ARP    44 10.9.0.1 is at 02:42:a0:5b:27:d7
01-29 21:3… 02:42:a0:5b:27:d7                    ARP    44 10.9.0.1 is at 02:42:a0:5b:27:d7
```

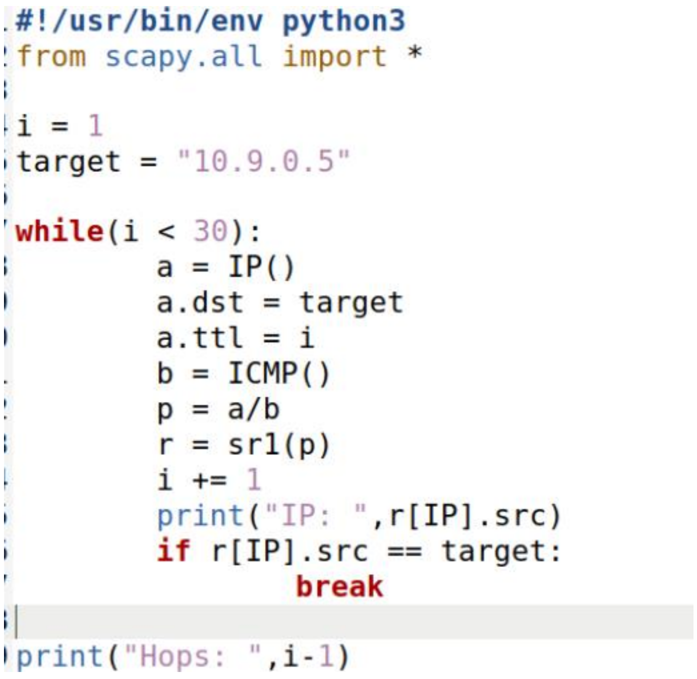
Hence, we see that the response was successful also proving that the spoofed packet was sent successfully.

1.3: Traceroute:

Here, we will build traceroute function using the scrapy.

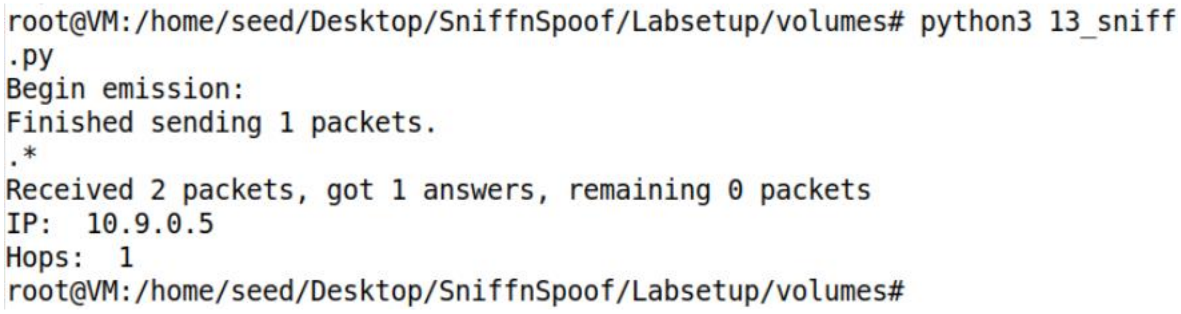
So, for this we will have to put the destination address in the target variable. Later, we will traverse through every TTL value using the loop till we reach the destination machine.

The code used in this is given below:



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

i = 1
target = "10.9.0.5"

while(i < 30):
        a = IP()
        a.dst = target
        a.ttl = i
        b = ICMP()
        p = a/b
        r = sr1(p)
        i += 1
        print("IP: ",r[IP].src)
        if r[IP].src == target:
                break

print("Hops: ",i-1)
```

As, shown below is the execution of the given code:

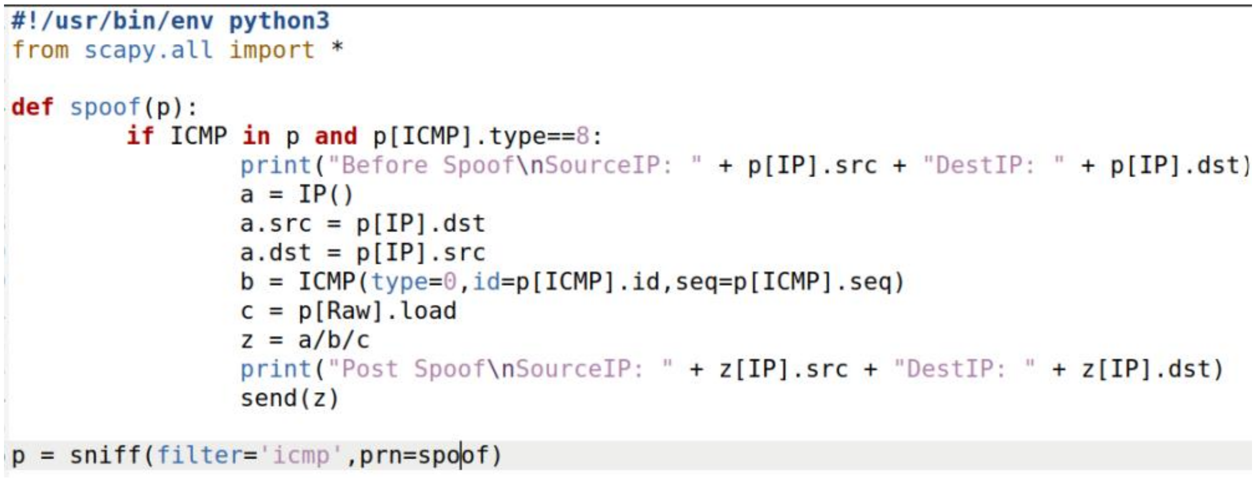


```
root@VM:/home/seed/Desktop/SniffnSpoof/Labsetup/volumes# python3 13_sniff
.py
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
IP:  10.9.0.5
Hops:  1
root@VM:/home/seed/Desktop/SniffnSpoof/Labsetup/volumes#
```

Hence, after execution we can see that in the attackers host machine traceroute to the ip addr 10.9.0.5 will be just 1 hop.

1.4: Sniffing and-then Spoofing:

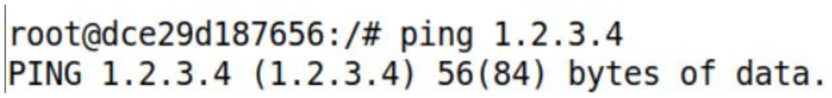
For this execution we will have to combine both the sniff and spoof programs as shown below:



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

def spoof(p):
        if ICMP in p and p[ICMP].type==8:
                print("Before Spoof\nSourceIP: " + p[IP].src + "DestIP: " + p[IP].dst)
                a = IP()
                a.src = p[IP].dst
                a.dst = p[IP].src
                b = ICMP(type=0,id=p[ICMP].id,seq=p[ICMP].seq)
                c = p[Raw].load
                z = a/b/c
                print("Post Spoof\nSourceIP: " + z[IP].src + "DestIP: " + z[IP].dst)
                send(z)

p = sniff(filter='icmp',prn=spoof)
```

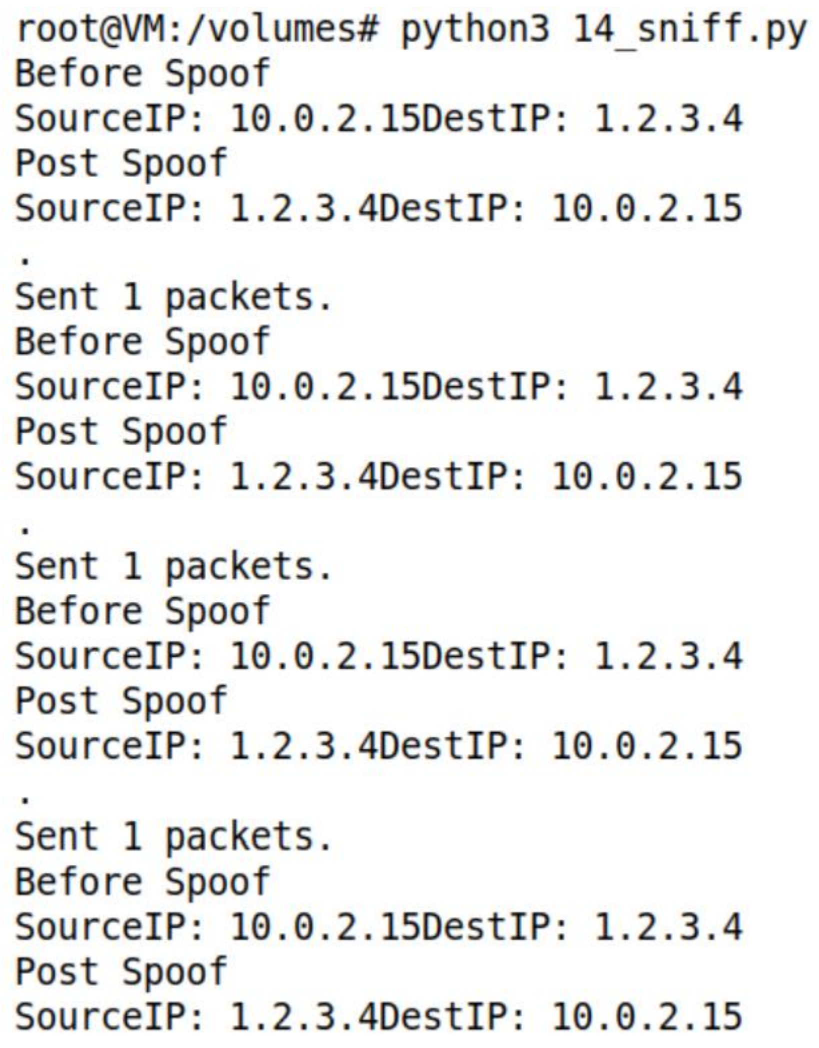
1) Using the Ping 1.2.3.4

User side:



```
root@dce29d187656:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
```

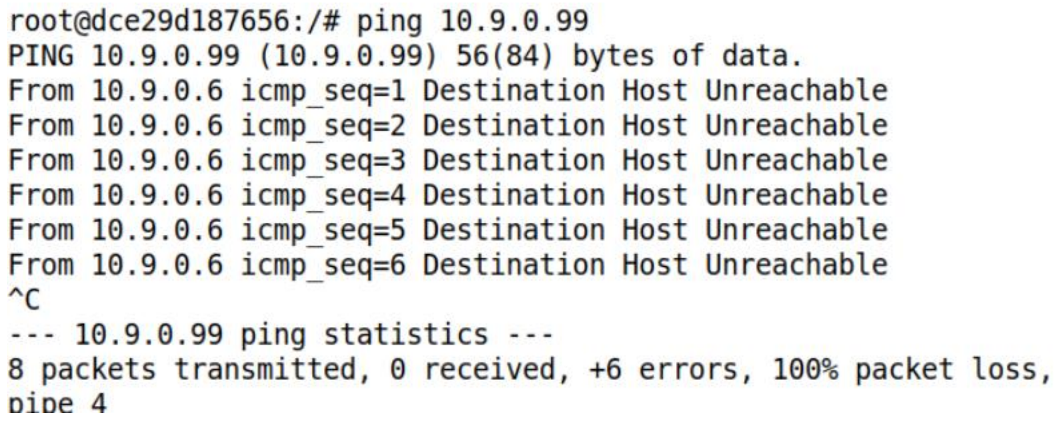
Attacker side:



```
root@VM:/volumes# python3 14_sniff.py
Before Spoof
SourceIP: 10.0.2.15DestIP: 1.2.3.4
Post Spoof
SourceIP: 1.2.3.4DestIP: 10.0.2.15
.
Sent 1 packets.
Before Spoof
SourceIP: 10.0.2.15DestIP: 1.2.3.4
Post Spoof
SourceIP: 1.2.3.4DestIP: 10.0.2.15
.
Sent 1 packets.
Before Spoof
SourceIP: 10.0.2.15DestIP: 1.2.3.4
Post Spoof
SourceIP: 1.2.3.4DestIP: 10.0.2.15
.
Sent 1 packets.
Before Spoof
SourceIP: 10.0.2.15DestIP: 1.2.3.4
Post Spoof
SourceIP: 1.2.3.4DestIP: 10.0.2.15
```

2) Ping 10.9.0.99

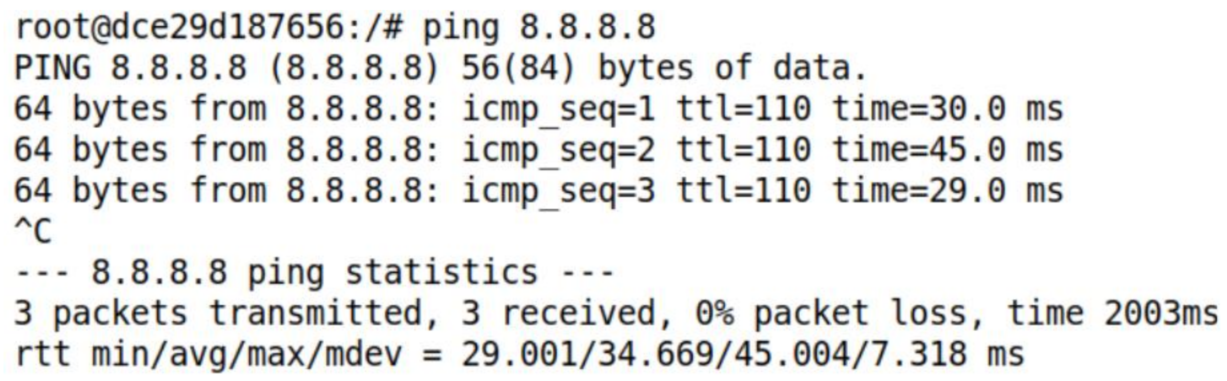
User Side:



```
root@dce29d187656:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.6 icmp_seq=1 Destination Host Unreachable
From 10.9.0.6 icmp_seq=2 Destination Host Unreachable
From 10.9.0.6 icmp_seq=3 Destination Host Unreachable
From 10.9.0.6 icmp_seq=4 Destination Host Unreachable
From 10.9.0.6 icmp_seq=5 Destination Host Unreachable
From 10.9.0.6 icmp_seq=6 Destination Host Unreachable
^C
--- 10.9.0.99 ping statistics ---
8 packets transmitted, 0 received, +6 errors, 100% packet loss,
pipe 4
```

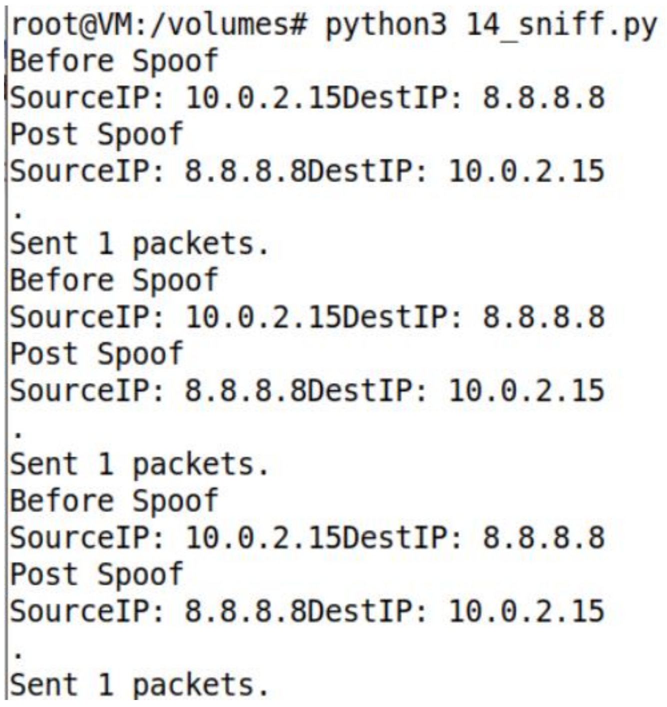
3) Ping 8.8.8.8

User Side:



```
root@dce29d187656:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=110 time=30.0 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=110 time=45.0 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=110 time=29.0 ms
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 29.001/34.669/45.004/7.318 ms
```

Attacker Side:



```
root@VM:/volumes# python3 14_sniff.py
Before Spoof
SourceIP: 10.0.2.15DestIP: 8.8.8.8
Post Spoof
SourceIP: 8.8.8.8DestIP: 10.0.2.15
.
Sent 1 packets.
Before Spoof
SourceIP: 10.0.2.15DestIP: 8.8.8.8
Post Spoof
SourceIP: 8.8.8.8DestIP: 10.0.2.15
.
Sent 1 packets.
Before Spoof
SourceIP: 10.0.2.15DestIP: 8.8.8.8
Post Spoof
SourceIP: 8.8.8.8DestIP: 10.0.2.15
.
Sent 1 packets.
```

So, in case 2 a non-existing host gets encountered on the LAN as shows that the host is unreachable. However, in cases 1 and 3 the spoofed packet was sent successfully. This happens because the IP on the LAN which is nor currently active fails to connect as APR resolution do not take place then therefore, ICMP echo request will also not be generated so the host becomes unreachable.