



COMPUTER NETWORK SECURITY LAB

LAB 1: SNIFFING AND SPOOFING

OBJECTIVE

- a) Understanding the concepts of Sniffing and Spoofing
- b) Using tools to Sniff and Spoof packets using Scapy

INTRODUCTION

Packet sniffing and spoofing are the two important concepts in network security; they are two major threats in network communication. Being able to understand these two threats is essential for understanding security measures in networking. There are many packets sniffing and spoofing tools, such as Wireshark, Tcpdump, Netwox, etc.

Sniffing is the process in which all the data packets passing in the network are monitored. Sniffers are usually used by network administrators to monitor and troubleshoot the network traffic. Whereas attackers use Sniffers to monitor and capture data packets to steal sensitive information containing password and user accounts.

Spoofing is the process in which an intruder introduces fake traffic and pretends to be someone else (legal source or the legitimate entity). Spoofing is done by sending packets with incorrect source address over the network.

EXECUTION

Using Tools to Sniff and Spoof Packets using Scapy

Attacker Machine: 10.0.2.9

Victim Machine: 10.0.2.10

2.1 Task 1: Sniffing Packets

2.1.1 Task 1.1 Sniff IP packets using Scapy

Attacker:

```
Terminal
[01/30/21]seed@Ankitha_PES1201801491:~/CNS$ sudo python sample.py
SNIFFING PACKETS...
##[ Ethernet ]##
  dst      = 08:00:27:22:6a:96
  src      = 08:00:27:e8:b9:1d
  type     = 0x800
##[ IP ]##
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 23308
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  checksum = 0xc78a
  src      = 10.0.2.10
  dst      = 10.0.2.9
  \options \
##[ ICMP ]##
  type     = echo-request
  code     = 0
  checksum = 0x803c
  id       = 0x11fc
  seq      = 0x1
##[ Raw ]##
  load     = '\x07\x15'\n\\\x00\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&'()*+,-./01234567'
##[ Ethernet ]##
```

Victim:

```
Terminal
[01/30/21]seed@Ankitha_PES1201801491:~$ ping 10.0.2.9
PING 10.0.2.9 (10.0.2.9) 56(84) bytes of data.
64 bytes from 10.0.2.9: icmp_seq=1 ttl=64 time=0.657 ms
64 bytes from 10.0.2.9: icmp_seq=2 ttl=64 time=1.60 ms
64 bytes from 10.0.2.9: icmp_seq=3 ttl=64 time=1.26 ms
64 bytes from 10.0.2.9: icmp_seq=4 ttl=64 time=1.58 ms
64 bytes from 10.0.2.9: icmp_seq=5 ttl=64 time=1.19 ms
64 bytes from 10.0.2.9: icmp_seq=6 ttl=64 time=1.13 ms
64 bytes from 10.0.2.9: icmp_seq=7 ttl=64 time=0.990 ms
^Z
[1]+  Stopped                  ping 10.0.2.9
[01/30/21]seed@Ankitha_PES1201801491:~$
```

OBSERVATIONS:

- The program code acts as a sniffer and captures the ping requests sent by another machine on the same network.
- The callback function in the sniff function displays the source, destination IP address and the protocol.

Explain on which VM you ran the command “sudo python sample.py “ and why?

The command “sudo python sample.py” is run on the Attacker side. The program code acts as a sniffer and captures the ping requests sent by the victim machine. i.e in this case, the victim(10.0.2.1) pings to the machine 10.0.2.9 and the attacker sniffs all the packets by the victim displaying all the packet details.

Now, we run the same program without root privileges. Do you find any issues? If so, why?

When we try to run the same program without root privileges, it does not execute successfully. This is because certain applications, files, packages etc has certain security privileges that does not

anyone in the system to open the raw sockets. Thus can be accessed only with root privileges. The following image shows the error

```
Terminal
[01/30/21]seed@Ankitha_PES1201801491:~/CNS$ python sample.py
SNIFFING PACKETS...
Traceback (most recent call last):
  File "sample.py", line 6, in <module>
    pkt = sniff(filter="icmp",prn=print pkt)
  File "/home/seed/.local/lib/python2.7/site-packages/scapy/sendrecv.py", line 731, in sniff
    *arg, **karg)] = iface
  File "/home/seed/.local/lib/python2.7/site-packages/scapy/arch/linux.py", line 567, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(type))
  File "/usr/lib/python2.7/socket.py", line 191, in __init__
    sock = _realsocket(family, type, proto)
socket.error: [Errno 1] Operation not permitted
[01/30/21]seed@Ankitha_PES1201801491:~/CNS$
```

2.1.2 Task 1.2 Capturing ICMP, TCP packet and Subnet

2.1.2.1 Capture only the ICMP packet

When victim pings to 10.0.2.9

Attacker:

```
Terminal
[01/30/21]seed@Ankitha_PES1201801491:~/CNS$ sudo python sample.py
SNIFFING PACKETS...
###[ Ethernet ]###
  dst      = 52:54:00:12:35:00
  src      = 08:00:27:e8:b9:1d
  type     = 0x800
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 6741
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = 0x43b
  src      = 10.0.2.10
  dst      = 8.8.8.8
  \options \
###[ ICMP ]###
  type     = echo-request
  code     = 0
  chksum   = 0x9485
  id       = 0x125a
  seq      = 0x1
###[ Raw ]###
  load     = '\xb2\x0b\x15'\x93\xb0\x0b\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&'()*+,-./01234567'
```

Victim:

```
Terminal
[01/30/21]seed@Ankitha_PES1201801491:~$ 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=116 time=84.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=116 time=421 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=116 time=42.5 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=116 time=68.0 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=116 time=40.8 ms
^Z
[1]+  Stopped                  ping 8.8.8.8
[01/30/21]seed@Ankitha_PES1201801491:~$
```

When attacker pings to 8.8.8.8

Attacker:

```
Terminal
[01/30/21]seed@Ankitha_PES1201801491:~/CNS$ sudo python sample.py
SNIFFING PACKETS...
##[ Ethernet ]##
  dst      = 52:54:00:12:35:00
  src      = 08:00:27:22:6a:96
  type     = 0x800
##[ IP ]##
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 60327
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  chksum   = 0x32e9
  src      = 10.0.2.9
  dst      = 8.8.8.8
  \options \
##[ ICMP ]##
  type     = echo-request
  code     = 0
  chksum   = 0x5903
  id       = 0x1302
  seq      = 0x1
##[ Raw ]##
  load     = '\xe3\n\x15'\x9f\x8b\t\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&'()*+,-./01234567'
```

Victim:

```
Terminal
[01/30/21]seed@Ankitha_PES1201801491:~$ 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=116 time=452 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=116 time=625 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=116 time=544 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=116 time=568 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=116 time=591 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=116 time=597 ms
^Z
[1]+  Stopped                  ping 8.8.8.8
[01/30/21]seed@Ankitha_PES1201801491:~$
```

OBSERVATIONS:

- The sniffer program code on the Attacker side has filter is only for the ICMP packets. Hence, when some machine on the same network sends ping requests, the packets get captured by the sniffer.
- Here when the victim sends a ping request to 8.8.8.8, the attacker sniffs all the icmp packets and gets all the packet details.

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

The attacker sniffer code sniffs the TCP traffic from a specific host (10.0.2.10) to port 23.

Victim:

```
Terminal
[01/30/21]seed@Ankitha_PES1201801491:~$ telnet 10.0.2.9
Trying 10.0.2.9...
Connected to 10.0.2.9.
Escape character is '^]'.
Ubuntu 16.04.2 LTS
Ankitha_PES1201801491 login: seed
Password:
Last login: Sat Jan 30 02:39:14 EST 2021 from 10.0.2.10 on pts/18
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.8.0-36-generic i686)

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

1 package can be updated.
0 updates are security updates.

[01/30/21]seed@Ankitha_PES1201801491:~$
```

Attacker:

```
Terminal
###[ Ethernet ]###
dst      = 08:00:27:22:6a:96
src      = 08:00:27:e8:b9:1d
type     = 0x800
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x10
len      = 52
id       = 33950
flags    = DF
frag     = 0
ttl      = 64
proto    = tcp
chksum   = 0x9e03
src      = 10.0.2.10
dst      = 10.0.2.9
\options \
###[ TCP ]###
sport    = 41630
dport    = telnet
seq      = 2974173451L
ack      = 2948533755L
dataofs  = 8
reserved = 0
flags    = A
window   = 245
chksum   = 0xe6af
urgptr   = 0
options  = [('NOP', None), ('NOP', None), ('Timestamp', (1980548, 1958788))]
```

OBSERVATIONS:

- The attacker sniffs all the TCP traffic from the victim that sends a telnet request to 10.0.2.9.
- From the sniffed packet details we observe the following:

Protocol: TCP

IP version: 4

Sport: 41630

Dport: telnet (port no 23)

Explain where you will run Telnet.

The telnet is run on the victim machine which sends a telnet request to the ip 10.0.2.9 over the port number 23.

2.1.2.2.1 Capture packets that comes from or go to a particular subnet

The subnet chosen for trial is the subnet of which host machine is part of. i.e 192.168.0.0/16

```
sniff1.py (~/.CNS) - gedit
#!/usr/bin/python
from scapy.all import *
print("SNIFFING PACKETS...");
def print_pkt(pkt):
    pkt.show()
pkt = sniff(filter='src net 192.168.0.0/16',
prn=print_pkt)
```

Attacker:

```
Terminal
[01/30/21]seed@Ankitha_PES1201801491:~/CNS$ sudo python sniff1.py
SNIFFING PACKETS...
##[ Ethernet ]##
  dst      = 08:00:27:e8:b9:1d
  src      = 52:54:00:12:35:00
  type     = 0x800
##[ IP ]##
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 28523
  flags    =
  frag     = 0
  ttl      = 127
  proto    = icmp
  checksum = 0xc78a
  src      = 192.168.56.1
  dst      = 10.0.2.10
  \options \
##[ ICMP ]##
  type     = echo-reply
  code     = 0
  checksum = 0x3efe
  id       = 0x17e5
  seq      = 0x1
##[ Raw ]##
  load     = 'X3\x15\C\x85\r\x00\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&'()*+,-./01234567'
##[ Ethernet ]##
  dst      = 08:00:27:e8:b9:1d
  src      = 52:54:00:12:35:00
  type     = 0x800
##[ IP ]##
```

Victim:

```
Terminal
[01/30/21]seed@Ankitha_PES1201801491:~$ ping 192.168.56.1
PING 192.168.56.1 (192.168.56.1) 56(84) bytes of data.
64 bytes from 192.168.56.1: icmp_seq=1 ttl=127 time=2.80 ms
64 bytes from 192.168.56.1: icmp_seq=2 ttl=127 time=0.825 ms
64 bytes from 192.168.56.1: icmp_seq=3 ttl=127 time=1.73 ms
64 bytes from 192.168.56.1: icmp_seq=4 ttl=127 time=1.94 ms
64 bytes from 192.168.56.1: icmp_seq=5 ttl=127 time=1.72 ms
64 bytes from 192.168.56.1: icmp_seq=6 ttl=127 time=0.528 ms
64 bytes from 192.168.56.1: icmp_seq=7 ttl=127 time=1.00 ms
64 bytes from 192.168.56.1: icmp_seq=8 ttl=127 time=1.83 ms
^Z
[1]+  Stopped                  ping 192.168.56.1
[01/30/21]seed@Ankitha_PES1201801491:~$
```

OBSERVATIONS:

- When the victim sends ICMP packets to 192.168.254.1, the sniffer program captures the packets sent out from 192.168.254.1 and thus retrieves the packet details

2.1.3 Task 2: Spoofing

The program code spoofs ICMP echo request packets and sends them to another VM on the same network. The spoofed request is formed by creating our own packet with the header specifications. Here we create an ICMP header. Similarly, we fill the IP header with source IP address of any machine within the local network and destination IP address of any remote machine on the internet which is alive.

Attacker:

```
Terminal
[01/30/21]seed@Ankitha PES1201801491:~/CNS$ sudo python spoof.py
SENDING SPOOFED ICMP PACKET...
###[ IP ]###
version      = 4
ihl          = None
tos          = 0x0
len          = None
id           = 1
flags        = 
frag         = 0
ttl          = 64
proto        = icmp
chksum       = None
src          = 10.0.2.10
dst          = 192.168.56.1
\options     \
###[ ICMP ]###
type         = echo-request
code         = 0
chksum       = None
id           = 0x0
seq          = 0x0
[01/30/21]seed@Ankitha PES1201801491:~/CNS$
```

The image shows a Wireshark packet capture window. The top pane displays a list of five packets. Packet 4 is highlighted, showing an ICMP Echo (ping) request from 192.168.56.1 to 10.0.2.10. Packet 5 is the corresponding Echo (ping) reply from 10.0.2.10 back to 192.168.56.1. The bottom pane shows the detailed view of packet 4, identifying it as an Internet Protocol Version 4 (IP) packet with a spoofed source address of 192.168.56.1 and a destination of 10.0.2.10. The ICMP field is also expanded, showing it is an echo request with ID 0 and sequence number 0.

No.	Time	Source	Destination	Protocol	Length	Info
1	2021-01-30 05:36:06.4595181...	::1	::1	UDP	64	41435 → 36254 Len=0
2	2021-01-30 05:36:07.7476427...	PcsCompu_22:6a:96		ARP	44	Who has 10.0.2.1? Tell 10.0.2.9
3	2021-01-30 05:36:07.7480176...	RealtekU_12:35:00		ARP	62	10.0.2.1 is at 52:54:00:12:35:00
4	2021-01-30 05:36:07.7503702...	10.0.2.10	192.168.56.1	ICMP	44	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 5)
5	2021-01-30 05:36:07.7512258...	192.168.56.1	10.0.2.10	ICMP	62	Echo (ping) reply id=0x0000, seq=0/0, ttl=127 (request in 4)

Frame 5: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 192.168.56.1, Dst: 10.0.2.10
0100 = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 28
Identification: 0x6f74 (28532)
Flags: 0x00
Fragment offset: 0
Time to live: 127
Protocol: ICMP (1)
Header checksum: 0xc7b9 [validation disabled]
[Header checksum status: Unverified]
Source: 192.168.56.1
Destination: 10.0.2.10
[Source GeoIP: Unknown]
[Destination GeoIP: Unknown]
Internet Control Message Protocol
VSS-Monitoring ethernet trailer, Source Port: 0

Internet Protocol Version 4 (Ip), 20 bytes
Packets: 5 - Displayed: 5 (100.0%)
Profile: Default

Victim:

```
Terminal
[01/30/21]seed@Ankitha_PES1201801491:~$ ping 10.0.2.9
PING 10.0.2.9 (10.0.2.9) 56(84) bytes of data.
64 bytes from 10.0.2.9: icmp_seq=1 ttl=64 time=0.605 ms
64 bytes from 10.0.2.9: icmp_seq=2 ttl=64 time=1.46 ms
64 bytes from 10.0.2.9: icmp_seq=3 ttl=64 time=1.29 ms
64 bytes from 10.0.2.9: icmp_seq=4 ttl=64 time=1.20 ms
64 bytes from 10.0.2.9: icmp_seq=5 ttl=64 time=0.461 ms
^Z
[1]+  Stopped                  ping 10.0.2.9
[01/30/21]seed@Ankitha_PES1201801491:~$
```

Wireshark packet capture showing ICMP Echo (ping) requests and replies between 10.0.2.10 and 10.0.2.9. The capture shows 13 packets. The first 11 packets are ICMP Echo (ping) requests and replies. The 12th packet is an ARP request from 10.0.2.10 to 10.0.2.9. The 13th packet is an ARP reply from 10.0.2.9 to 10.0.2.10.

No.	Time	Source	Destination	Protocol	Length	Info
1	2021-01-30 05:40:37.942857...	:::1	:::1	UDP	64	41057 → 37798 Len=0
2	2021-01-30 05:40:42.1180323...	10.0.2.10	10.0.2.9	ICMP	100	Echo (ping) request id=0x1951, seq=1/256, ttl=64 (reply in 3)
3	2021-01-30 05:40:42.1186256...	10.0.2.9	10.0.2.10	ICMP	100	Echo (ping) reply id=0x1951, seq=1/256, ttl=64 (request in 2)
4	2021-01-30 05:40:43.1481558...	10.0.2.10	10.0.2.9	ICMP	100	Echo (ping) request id=0x1951, seq=2/512, ttl=64 (reply in 5)
5	2021-01-30 05:40:43.1495981...	10.0.2.9	10.0.2.10	ICMP	100	Echo (ping) reply id=0x1951, seq=2/512, ttl=64 (request in 4)
6	2021-01-30 05:40:44.1500403...	10.0.2.10	10.0.2.9	ICMP	100	Echo (ping) request id=0x1951, seq=3/768, ttl=64 (reply in 7)
7	2021-01-30 05:40:44.1512998...	10.0.2.9	10.0.2.10	ICMP	100	Echo (ping) reply id=0x1951, seq=3/768, ttl=64 (request in 6)
8	2021-01-30 05:40:45.1516528...	10.0.2.10	10.0.2.9	ICMP	100	Echo (ping) request id=0x1951, seq=4/1024, ttl=64 (reply in 9)
9	2021-01-30 05:40:45.1528272...	10.0.2.9	10.0.2.10	ICMP	100	Echo (ping) reply id=0x1951, seq=4/1024, ttl=64 (request in 8)
10	2021-01-30 05:40:46.1520822...	10.0.2.10	10.0.2.9	ICMP	100	Echo (ping) request id=0x1951, seq=5/1280, ttl=64 (reply in 11)
11	2021-01-30 05:40:46.1523307...	10.0.2.9	10.0.2.10	ICMP	100	Echo (ping) reply id=0x1951, seq=5/1280, ttl=64 (request in 10)
12	2021-01-30 05:40:47.2450255...	PcsCompu_22:6a:96	10.0.2.9	ARP	62	Who has 10.0.2.10? Tell 10.0.2.9
13	2021-01-30 05:40:47.2450359...	PcsCompu_e8:b9:1d	10.0.2.10	ARP	44	10.0.2.10 is at 08:00:27:e8:b9:1d

Reply received by the victim from host with ip 192.168.56.1 which had received a spoof request from the attacker.

Wireshark packet capture showing a spoofed ICMP Echo (ping) request from 192.168.56.1 to 10.0.2.9. The capture shows 4 packets. The first packet is a UDP packet from 10.0.2.10 to 10.0.2.9. The second packet is an ARP request from 10.0.2.10 to 10.0.2.9. The third packet is an ICMP Echo (ping) request from 192.168.56.1 to 10.0.2.9. The fourth packet is a UDP packet from 10.0.2.10 to 10.0.2.9.

No.	Time	Source	Destination	Protocol	Length	Info
1	2021-01-30 05:35:57.1984574...	:::1	:::1	UDP	64	41057 → 37798 Len=0
2	2021-01-30 05:36:07.7468985...	PcsCompu_22:6a:96	10.0.2.10	ARP	62	Who has 10.0.2.10? Tell 10.0.2.9
3	2021-01-30 05:36:07.7502601...	192.168.56.1	10.0.2.9	ICMP	62	Echo (ping) request id=0x0000, seq=0/0, ttl=127
4	2021-01-30 05:36:17.7536407...	:::1	:::1	UDP	64	41057 → 37798 Len=0

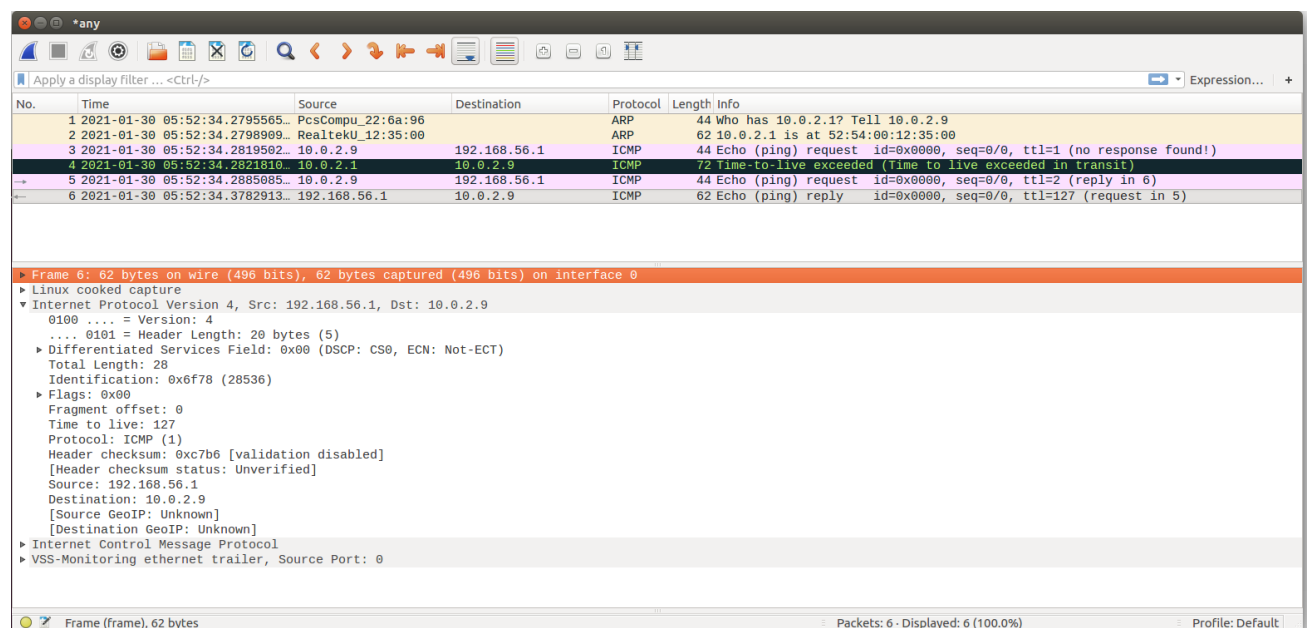
OBSERVATIONS:

- The spoof ICMP echo request packets sent to the host with ip 192.168.56.1 was accepted and an echo reply packet was sent to the spoofed IP address(i.e victim: 10.0.2.10)
- The reply from the spoofed packet was observed in the wireshark of the victim and attacker machine.

2.1.4 Task 3: Traceroute

Implemented a simple traceroute tool using Scapy to estimate the distance, in terms of number of routers, between your VM and a selected destination.

```
Terminal
[01/30/21]seed@Ankitha_PES1201801491:~/CNS$ sudo python traceroute.py 192.168.56.1
Traceroute 192.168.56.1
('1 hops away: ', '10.0.2.1')
('2 hops away: ', '192.168.56.1')
('Done', '192.168.56.1')
[01/30/21]seed@Ankitha_PES1201801491:~/CNS$
```



OBSERVATIONS:

- The packet will be dropped by the first router, which will send us an ICMP error message, telling that the TTL has exceeded.
- The wireshark capture shows the ICMP requests sent with increasing TTL (i.e the second packet is sent out with TTL value of 2)

2.1.5 Task 4: Sniffing and-then Spoofing

In this task, victim machine pings a non-existing IP address "1.2.3.4". As the attacker machine is on the same network, it sniffs the request packet, creates a new echo reply packet with IP and ICMP header and sends it to the victim machine. Hence, the user will always receive an echo reply from a non-existing IP address indicating that the machine is alive.

VM 1 (Victim): 10.0.2.10

Ping X: 1.2.3.4

VM 2 (attacker with sniffer-spoofing running): 10.0.2.9

Attacker:

```
Terminal
[01/30/21]seed@Ankitha_PES1201801491:~/CNS$ sudo python sniffspoof.py
original packet.....
('source IP:', '10.0.2.10')
('Destination IP:', '1.2.3.4')
spoofed packet.....
('Source IP:', '1.2.3.4')
('Destination IP:', '10.0.2.10')
original packet.....
('source IP:', '10.0.2.10')
('Destination IP:', '1.2.3.4')
spoofed packet.....
('Source IP:', '1.2.3.4')
('Destination IP:', '10.0.2.10')
original packet.....
('source IP:', '10.0.2.10')
('Destination IP:', '1.2.3.4')
spoofed packet.....
('Source IP:', '1.2.3.4')
('Destination IP:', '10.0.2.10')
original packet.....
('source IP:', '10.0.2.10')
('Destination IP:', '1.2.3.4')
spoofed packet.....
('Source IP:', '1.2.3.4')
```

Victim:

```
Terminal
[01/30/21]seed@Ankitha_PES1201801491:~$ ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=12.5 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=25.2 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=6.13 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=6.00 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=64 time=5.99 ms
64 bytes from 1.2.3.4: icmp_seq=6 ttl=64 time=25.1 ms
64 bytes from 1.2.3.4: icmp_seq=7 ttl=64 time=33.6 ms
64 bytes from 1.2.3.4: icmp_seq=8 ttl=64 time=27.1 ms
^Z
[1]+  Stopped                  ping 1.2.3.4
[01/30/21]seed@Ankitha_PES1201801491:~$
```

No.	Time	Source	Destination	Protocol	Length	Info
2	2021-01-30 06:06:25.6446590	10.0.2.10	1.2.3.4	ICMP	100	Echo (ping) request id=0x19eb, seq=1/256, ttl=64 (reply in 5)
5	2021-01-30 06:06:25.6571850	1.2.3.4	10.0.2.10	ICMP	100	Echo (ping) reply id=0x19eb, seq=1/256, ttl=64 (request in 2)
6	2021-01-30 06:06:26.6471025	10.0.2.10	1.2.3.4	ICMP	100	Echo (ping) request id=0x19eb, seq=2/512, ttl=64 (reply in 7)
7	2021-01-30 06:06:26.6722740	1.2.3.4	10.0.2.10	ICMP	100	Echo (ping) reply id=0x19eb, seq=2/512, ttl=64 (request in 6)
8	2021-01-30 06:06:27.6484354	10.0.2.10	1.2.3.4	ICMP	100	Echo (ping) request id=0x19eb, seq=3/768, ttl=64 (reply in 9)
9	2021-01-30 06:06:27.6545541	1.2.3.4	10.0.2.10	ICMP	100	Echo (ping) reply id=0x19eb, seq=3/768, ttl=64 (request in 8)
10	2021-01-30 06:06:28.6500412	10.0.2.10	1.2.3.4	ICMP	100	Echo (ping) request id=0x19eb, seq=4/1024, ttl=64 (reply in 11)
11	2021-01-30 06:06:28.6560350	1.2.3.4	10.0.2.10	ICMP	100	Echo (ping) reply id=0x19eb, seq=4/1024, ttl=64 (request in 10)
12	2021-01-30 06:06:29.6518720	10.0.2.10	1.2.3.4	ICMP	100	Echo (ping) request id=0x19eb, seq=5/1280, ttl=64 (reply in 13)
13	2021-01-30 06:06:29.6578511	1.2.3.4	10.0.2.10	ICMP	100	Echo (ping) reply id=0x19eb, seq=5/1280, ttl=64 (request in 12)
14	2021-01-30 06:06:30.6537331	10.0.2.10	1.2.3.4	ICMP	100	Echo (ping) request id=0x19eb, seq=6/1536, ttl=64 (reply in 15)
15	2021-01-30 06:06:30.6788479	1.2.3.4	10.0.2.10	ICMP	100	Echo (ping) reply id=0x19eb, seq=6/1536, ttl=64 (request in 14)
17	2021-01-30 06:06:31.6544605	10.0.2.10	1.2.3.4	ICMP	100	Echo (ping) request id=0x19eb, seq=7/1792, ttl=64 (reply in 18)

▶ Frame 5: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface 0
 ▶ Linux cooked capture
 ▼ Internet Protocol Version 4, Src: 1.2.3.4, Dst: 10.0.2.10
 0100 = Version: 4
 0101 = Header Length: 20 bytes (5)
 ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
 Total Length: 84
 Identification: 0x0001 (1)
 ▶ Flags: 0x00
 Fragment offset: 0
 Time to live: 64
 Protocol: ICMP (1)
 Header checksum: 0x6a99 [validation disabled]
 [Header checksum status: Unverified]
 Source: 1.2.3.4
 Destination: 10.0.2.10
 ▶ [Source GeoIP: Mukilteo, WA, United States, 47.912998, -122.304199]
 [Destination GeoIP: Unknown]
 ▶ Internet Control Message Protocol

Internet Control Message Protocol: Protocol
 Packets: 31 · Displayed: 16 (51.6%)
 Profile: Default

OBSERVATIONS:

- We observe that spoofer program sends spoofed ICMP responses to the ICMP requests set by the victim machine.
- The victim machine pings a non-existing IP address, but gets back ICMP response.