314078023
204420590

Sniffing Spoofing Lab Report

Tasks 1.1A:

In this following screenshot(1), it is evident that using the python script (2) with root privilege we were able to capture and document the packets and their content however, in the next screenshot (3) we got a permission error. Opening a raw socket requires higher privilege then given in this task.

1.

```python
#!/usr/bin/env python3

from scapy.all import *


def print_pkt(pkt):
    pkt.show()


pkt = sniff(iface="br-c93733e9f913", filter="icmp", prn=print_pkt)
```

2.

```
x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f
!"#$%&\'()*+,-./01234567'

###[ Ethernet ]###
  dst       = 08:00:27:34:68:d3
  src       = 52:54:00:12:35:02
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 7054
     flags     =
     frag      = 0
     ttl       = 111
     proto     = icmp
     chksum    = 0x13fd
     src       = 8.8.8.8
     dst       = 10.0.2.15
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0x9bac
        id        = 0x8
        seq       = 0x3c
###[ Raw ]###
           load      = '\xd2\x9f*`\x00\x00\x00\x00\x9a<\x0e\x00\x00\x00\
x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f
!"#$%&\'()*+,-./01234567'

^Z
[4]+  Stopped                 sudo python3 sniff.py
```

```
64 bytes from 8.8.8.8: icmp_seq=29 ttl=111 time=78.6 ms
64 bytes from 8.8.8.8: icmp_seq=30 ttl=111 time=78.0 ms
64 bytes from 8.8.8.8: icmp_seq=31 ttl=111 time=79.0 ms
64 bytes from 8.8.8.8: icmp_seq=32 ttl=111 time=79.6 ms
64 bytes from 8.8.8.8: icmp_seq=33 ttl=111 time=78.6 ms
64 bytes from 8.8.8.8: icmp_seq=34 ttl=111 time=79.5 ms
64 bytes from 8.8.8.8: icmp_seq=35 ttl=111 time=79.5 ms
64 bytes from 8.8.8.8: icmp_seq=36 ttl=111 time=78.1 ms
64 bytes from 8.8.8.8: icmp_seq=37 ttl=111 time=78.8 ms
64 bytes from 8.8.8.8: icmp_seq=38 ttl=111 time=78.8 ms
64 bytes from 8.8.8.8: icmp_seq=39 ttl=111 time=78.2 ms
64 bytes from 8.8.8.8: icmp_seq=40 ttl=111 time=78.4 ms
64 bytes from 8.8.8.8: icmp_seq=41 ttl=111 time=78.2 ms
64 bytes from 8.8.8.8: icmp_seq=42 ttl=111 time=78.1 ms
64 bytes from 8.8.8.8: icmp_seq=43 ttl=111 time=78.5 ms
64 bytes from 8.8.8.8: icmp_seq=44 ttl=111 time=78.0 ms
64 bytes from 8.8.8.8: icmp_seq=45 ttl=111 time=79.7 ms
64 bytes from 8.8.8.8: icmp_seq=46 ttl=111 time=79.7 ms
64 bytes from 8.8.8.8: icmp_seq=47 ttl=111 time=81.3 ms
64 bytes from 8.8.8.8: icmp_seq=48 ttl=111 time=79.6 ms
64 bytes from 8.8.8.8: icmp_seq=49 ttl=111 time=79.8 ms
64 bytes from 8.8.8.8: icmp_seq=50 ttl=111 time=78.6 ms
64 bytes from 8.8.8.8: icmp_seq=51 ttl=111 time=78.6 ms
64 bytes from 8.8.8.8: icmp_seq=52 ttl=111 time=79.4 ms
64 bytes from 8.8.8.8: icmp_seq=53 ttl=111 time=78.8 ms
64 bytes from 8.8.8.8: icmp_seq=54 ttl=111 time=79.4 ms
64 bytes from 8.8.8.8: icmp_seq=55 ttl=111 time=79.2 ms
64 bytes from 8.8.8.8: icmp_seq=56 ttl=111 time=78.4 ms
64 bytes from 8.8.8.8: icmp_seq=57 ttl=111 time=78.4 ms
64 bytes from 8.8.8.8: icmp_seq=58 ttl=111 time=77.7 ms
64 bytes from 8.8.8.8: icmp_seq=59 ttl=111 time=79.8 ms
64 bytes from 8.8.8.8: icmp_seq=60 ttl=111 time=79.0 ms
^Z
[8]+  Stopped                 ping 8.8.8.8
```

3.

```
[02/15/21]seed@VM:~/Documents$ python3 sniff.py
Traceback (most recent call last):
  File "sniff.py", line 9, in <module>
    pkt = sniff(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", lin
e 398, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.h
tons(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
[02/15/21]seed@VM:~/Documents$
```

314078023
204420590

Tasks 1.1B:

Using BPF[1] to capture:

1. Only ICMP packets- In this screenshot we used BPF for icmp and the result in the console of the packets that were captured.

```python
#!/usr/bin/env python3

from scapy.all import *


def print_pkt(pkt):
    pkt.show()


pkt = sniff(filter="icmp", prn=print_pkt)
```

```
seed@VM: ~/Documents
    ttl       = 64
    proto     = icmp
    chksum    = 0xc1a6
    src       = 10.0.2.15
    dst       = 8.8.8.8
    \options   \
###[ ICMP ]###
       type      = echo-request
       code      = 0
       chksum    = 0xa0d4
       id        = 0x3
       seq       = 0x6
###[ Raw ]###
          load      = '\xd2\x04-`\x00\x00\x00\x00\x91\xea\x07\x00\x00\x
00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x
1f !"#$%&\'()*+,-./01234567'

###[ Ethernet ]###
  dst       = 08:00:27:34:68:d3
  src       = 52:54:00:12:35:02
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 894
     flags     =
     frag      = 0
     ttl       = 111
     proto     = icmp
     chksum    = 0x2c0d
     src       = 8.8.8.8
     dst       = 10.0.2.15
     \options   \
```

```
seed@VM: ~
64 bytes from 8.8.8.8: icmp_seq=3 ttl=111 time=81.7 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=111 time=77.6 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=111 time=77.5 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=111 time=79.6 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=111 time=78.2 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=111 time=102 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=111 time=78.7 ms
64 bytes from 8.8.8.8: icmp_seq=10 ttl=111 time=80.6 ms
64 bytes from 8.8.8.8: icmp_seq=11 ttl=111 time=78.2 ms
64 bytes from 8.8.8.8: icmp_seq=12 ttl=111 time=78.4 ms
64 bytes from 8.8.8.8: icmp_seq=13 ttl=111 time=80.9 ms
64 bytes from 8.8.8.8: icmp_seq=14 ttl=111 time=79.1 ms
64 bytes from 8.8.8.8: icmp_seq=15 ttl=111 time=79.3 ms
64 bytes from 8.8.8.8: icmp_seq=16 ttl=111 time=78.4 ms
^Z
[1]+  Stopped                 ping 8.8.8.8
[02/17/21]seed@VM:~$ 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=111 time=79.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=111 time=78.5 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=111 time=78.2 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=111 time=77.6 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=111 time=78.2 ms
^Z
[2]+  Stopped                 ping 8.8.8.8
[02/17/21]seed@VM:~$ 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=111 time=78.7 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=111 time=78.0 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=111 time=78.6 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=111 time=80.3 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=111 time=79.2 ms
^Z
[3]+  Stopped                 ping 8.8.8.8
[02/17/21]seed@VM:~$
```

[1] https://en.wikipedia.org/wiki/Berkeley_Packet_Filter

314078023
204420590

2. Any TCP packets from a particular IP with destination port 23- Because telnet is over TCP , we used it to sniff the TCP packets with the destination 23, for 172.217.169.37.

```python
#!/usr/bin/env python3

from scapy.all import *


def print_pkt(pkt):
    pkt.show()


pkt = sniff(filter="tcp dst port 23 and dst host 172.127.169.69", prn=print_pkt)
```

Activities Terminal Feb 15 12:56

seed@VM: ~/Documents

```
454429, 0)), ('NOP', None), ('WScale', 7)]

###[ Ethernet ]###
  dst       = 52:54:00:12:35:02
  src       = 08:00:27:34:68:d3
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x10
     len       = 60
     id        = 64967
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0xdad6
     src       = 10.0.2.15
     dst       = 172.217.169.37
     \options   \
###[ TCP ]###
        sport     = 32862
        dport     = telnet
        seq       = 1497807325
        ack       = 0
        dataofs   = 10
        reserved  = 0
        flags     = S
        window    = 64240
        chksum    = 0x623c
        urgptr    = 0
        options   = [('MSS', 1460), ('SAckOK', b''), ('Timestamp', (2743
458621, 0)), ('NOP', None), ('WScale', 7)]
```

seed@VM: ~

```
[02/15/21]seed@VM:~$ telnet gmail.com
Trying 216.58.204.5...
^Z^C
[02/15/21]seed@VM:~$ telnet gmail.com
Trying 216.58.204.5...
^Z^C
[02/15/21]seed@VM:~$ telnet gmail.com
Trying 216.58.204.5...
^Z^C
[02/15/21]seed@VM:~$ telnet gmail.com
Trying 172.217.169.37...
^Z^C
[02/15/21]seed@VM:~$
```

314078023
204420590

3. Capture packets comes from or to a particular subnet- We used the subnet adrress 10.0.2.0/25 to be sniffed.

```python
#!/usr/bin/env python3

from scapy.all import *

def print_pkt(pkt):
    pkt.show()


pkt = sniff(filter="dst net 10.0.2.0/25", prn=print_pkt)
```

```
x1f !"#$%&\'()*+,-./01234567'

###[ Ethernet ]###
  dst       = 08:00:27:86:d6:05
  src       = 52:54:00:12:35:00
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 3660
     flags     =
     frag      = 0
     ttl       = 116
     proto     = icmp
     chksum    = 0x1c49
     src       = 8.8.8.8
     dst       = 10.0.2.5
     \options   \
###[ ICMP ]###
        type      = echo-reply
        code      = 0
        chksum    = 0xa7ac
        id        = 0x1b
        seq       = 0x3
###[ Raw ]###
           load      = '\x00\x00F`\x00\x00\x00\x00D\x02\x0f\x00\x00\x0
0\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\
x1f !"#$%&\'()*+,-./01234567'

^Z
[8]+  Stopped                 python3 1_1b.py
root@VM:/volumes/Sniff_Spoof/Python Code#
```

```
[10]+  Stopped                 ping 8.8.8.8
root@0dc37fd4c48f:/# 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=115 time=74.6 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=74.2 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=74.7 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=115 time=75.7 ms
^Z
[11]+  Stopped                 ping 8.8.8.8
root@0dc37fd4c48f:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.9.0.5  netmask 255.255.255.0  broadcast 10.9.0.255
        ether 02:42:0a:09:00:05  txqueuelen 0  (Ethernet)
        RX packets 288  bytes 30747 (30.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 103  bytes 8806 (8.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 4  bytes 448 (448.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4  bytes 448 (448.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@0dc37fd4c48f:/# 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=115 time=75.7 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=74.3 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=78.4 ms
^Z
[12]+  Stopped                 ping 8.8.8.8
root@0dc37fd4c48f:/#
```

314078023
204420590

<u>Tasks 1.2</u>:

In the next screenshot (1) you can see we used the previous sniffer to sniff the ICMP packet we spoofed. We set the destination of the the packet as the VM we were working on and the source as google public DNS server. And as you can see the screenshot (2) the sniffer see's the packet source as 8.8.8.8.

1.

```
#!/usr/bin/env python3

import ...

a = IP()
a.dst = "10.0.2.3"
a.src = "8.8.8.8"
b = ICMP()
p = a/b
send(p)
ls(a)
```

2.

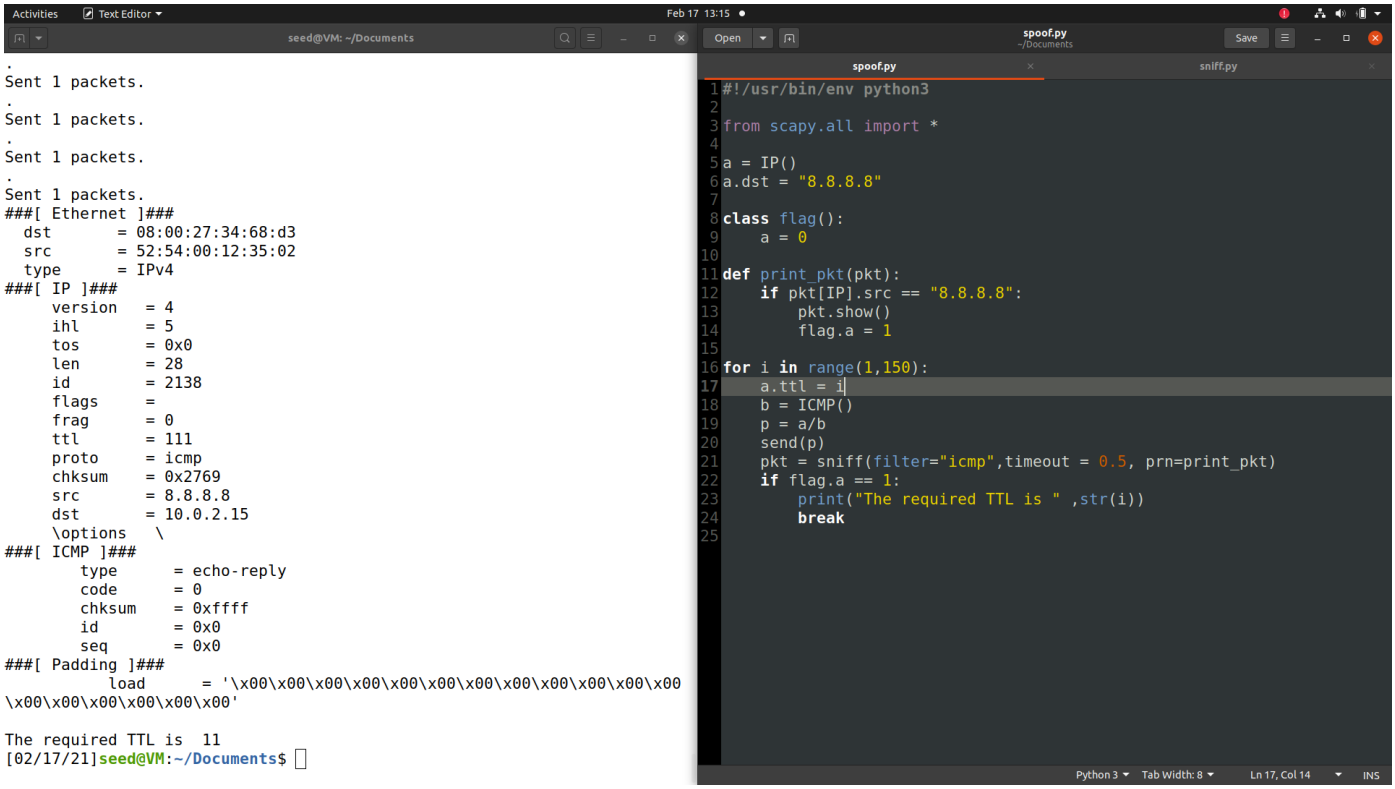
```
[02/17/21]seed@VM:~/Documents$ sudo python3 mycode.py
.
Sent 1 packets.
version    : BitField  (4 bits)                = 4               (4)
ihl        : BitField  (4 bits)                = None            (None
)
tos        : XByteField                        = 0               (0)
len        : ShortField                        = None            (None
)
id         : ShortField                        = 1               (1)
flags      : FlagsField  (3 bits)              = <Flag 0 ()>     (<Fla
g 0 ()>)
frag       : BitField  (13 bits)               = 0               (0)
ttl        : ByteField                         = 64              (64)
proto      : ByteEnumField                     = 0               (0)
chksum     : XShortField                       = None            (None
)
src        : SourceIPField                     = '8.8.8.8'       (None
)
dst        : DestIPField                       = '10.0.2.3'      (None
)
options    : PacketListField                   = []              ([])
[02/17/21]seed@VM:~/Documents$
```

```
906, in _run
    sniff_sockets[L2socket(type=ETH_P_ALL, iface=iface,
  File "/usr/local/lib/python3.8/dist-packages/scapy/arch/linux.py", lin
e 398, in __init__
    self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.h
tons(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
[02/17/21]seed@VM:~/Documents$ sudo python3 s
sniff.py  spoof.py
[02/17/21]seed@VM:~/Documents$ sudo python3 sniff.py
###[ Ethernet ]###
  dst       = 52:54:00:12:35:03
  src       = 08:00:27:34:68:d3
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 28
     id        = 1
     flags     =
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x5ece
     src       = 8.8.8.8
     dst       = 10.0.2.3
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xf7ff
        id        = 0x0
        seq       = 0x0
```

314078023
204420590

<u>Tasks 1.3</u>:

Traceroute - In the next python script we were asked to estimate the distance in terms of network nodes between our VM and a selected destination. We chose Google public DNS server. We made a Boolean flag to know when our packet got to the destination. We used a for loop (with an arbitrary amount of iterations that we knew would be sufficient) and pinged the server, once the response came from 8.8.8.8 we changed the flag to 1, and that's how we made sure it came back from google. In the console screenshot, it is visible that the number of routers was 11.



314078023
204420590

Task 1.4:

1. **10.9.0.99:**
   Because this should be an IP in our own LAN our computer will try to reach it through it's MAC address. Because the MAC will not e available in the OS's Mac table it will result in us sending an ARP request to get the MAC. Those are broadcast requests sent to the whole LAN asking for the computer with the requested IP address to send over his MAC address. We sniffed the packets and after receiving suck a request we spoofed the ARP reply and than the ICMP reply.

```
seed@VM: ~/.../Labsetup    seed@VM: ~/.../Labsetup
root@VM:/volumes/Sniff_Spoof/Python Code# python3 1_4_sniff_spoof_arp.
py
.
Sent 1 packets.
.
Sent 1 packets.
```

```
seed@VM: ~    seed@VM: ~
root@0dc37fd4c48f:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
64 bytes from 10.9.0.99: icmp_seq=1 ttl=64 time=60.5 ms
64 bytes from 10.9.0.99: icmp_seq=2 ttl=64 time=23.5 ms
64 bytes from 10.9.0.99: icmp_seq=3 ttl=64 time=13.7 ms
64 bytes from 10.9.0.99: icmp_seq=4 ttl=64 time=20.7 ms
```

2. **8.8.8.8:**
   Because we also receive a response from the real server and from our spoofing code we get a DUP (duplicated) response.

```
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
^Croot@VM:/volumes/Sniff_Spoof/Python Code#
```

```
root@0dc37fd4c48f:/# 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=64 time=32.2 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=76.6 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=64 time=25.1 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=75.5 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=64 time=21.9 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=75.0 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=64 time=15.7 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=115 time=74.1 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=5 ttl=64 time=23.3 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=115 time=75.7 ms (DUP!)
^C
```

3. **1.2.3.4:**
   Because this IP does not exists the only response we receive is from our spoofing code

```
root@VM:/volumes/Sniff_Spoof/Python Code# python3 1_4_sniff_spoof_arp.
py
.
Sent 1 packets.
.
Sent 1 packets.
```

```
root@0dc37fd4c48f:/# 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=50.1 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=27.9 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=64 time=17.0 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=64 time=16.1 ms
```

314078023
204420590

<u>Task 2.1A:</u>

1. Library calls:
    a. **pcap_open_live**: Starts sniffing on the chosen network interface. In here we define the buffer size, promisc mode on/off and delay in milliseconds. This needs root privileges.pcap_datalink: Returns the kind of device we're capturing on.
    b. **pcap_compile**: Compiles the filter expression stored in a regular string format to binary and sets in on the sniffing handle.
    c. **pcap_setfilter:** Starts the above compiled filter,pcap_freecode: Frees up allocated memory generated by pcap_compile.
    d. **pcap_loop:** Starts the actual sniffing session on the sniffing handle we opened previously with the open_live function. For each receiving packet this loop will call the got_packet function that will process each and every packet according to our settings.
    e. **pcap_close:** Closes the sniffing session.

2. **Root access:**
   We need root access because pcap library needs access to low level network interface functions like CAP_NET_RAW[2]. Any access like that needs root privilege in order to access low level and hardware drivers, that an OS security feature.

[2] https://linux.die.net/man/7/capabilities

314078023
204420590

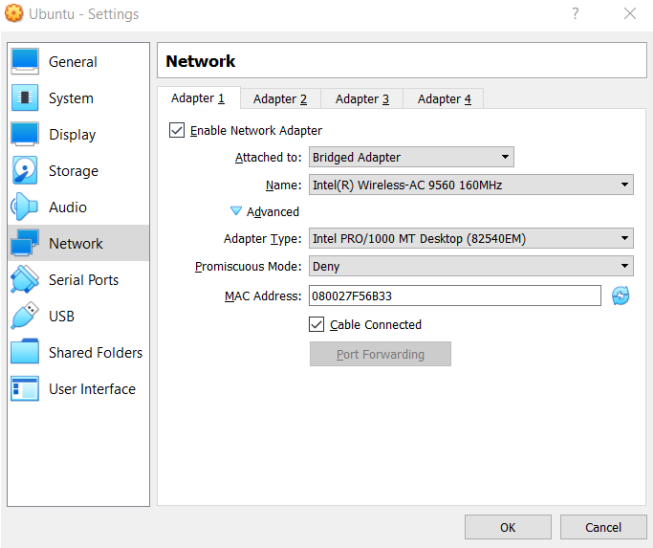
3. **Promiscuous mode:**
   In order to turn Promiscuous the mode on/off we simply change the third int value of pcap_open_live to 1/0 accrodingly:
   on -

```
handle = pcap_open_live(capture_device, 65536, 1, 100, errbuf);
```

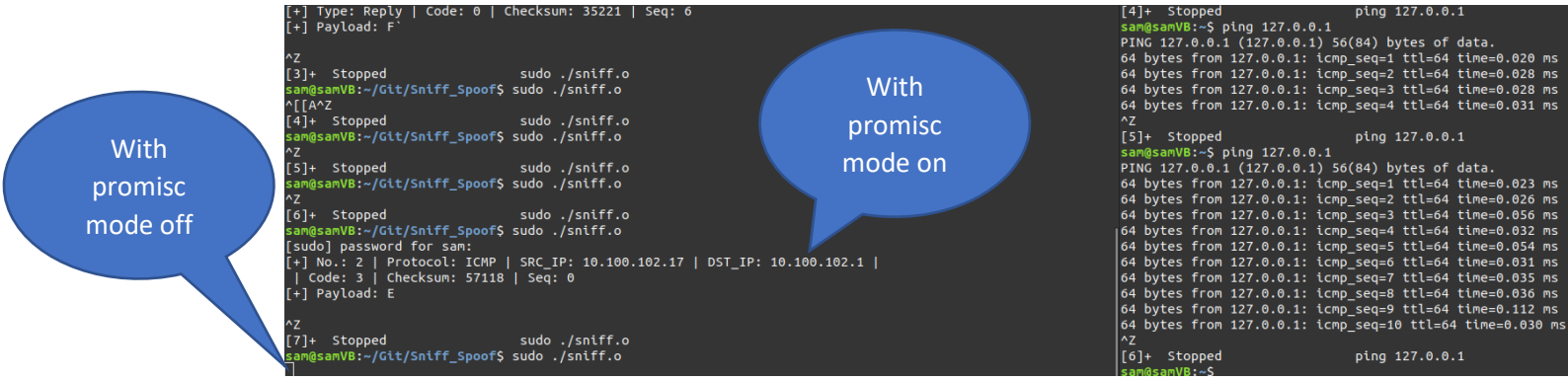
   off -

```
handle = pcap_open_live(capture_device, 65536, 0, 100, errbuf);
```

   Promiscuous mode allows a network interface to access data and communication that it not intended to and directly access hardware interface and drivers. We will demonstrate by first changing our VM network setting to disable Promiscuous mode by default than we set networking to bridged.



   We will send a ping to 127.0.0.1 which should be sniffed from loopback ("lo") interface and will try to sniff them from the main "enp03" interface from the VM. We can clearly see that when we try without the Promiscuous enabled in the code we cannot sniff those pakcets but as soon as we enable them we can sniff the loopback packets from a non-intended interface.



314078023
204420590

Task 2.1B:

ICMP Between to specific hosts:

In the next screenshot we captured ICMP packets between 8.8.8.8 to 10.9.0.5

```
static char filter_exp[] = "icmp and (src host 10.9.0.5 and dst host 8.8.8.8) or (src host 8.8.8.8 and dst host 10.9.0.5)";
```

as we can see in the next screenshot no packet that is not netween those IP's is being sniffed no matter the command

```
[+] Payload: ⓪SF`

[+] No.: 5 | Protocol: ICMP | SRC_IP: 8.8.8.8 | DST_IP: 10.9.0.5 |
[+] Type: Reply | Code: 0 | Checksum: 16379 | Seq: 2
[+] Payload: ⓪SF`

^Z
[30]+  Stopped                 ./sniff.o
root@VM:/volumes/Sniff_Spoof# ./sniff.o
[+] No.: 2 | Protocol: ICMP | SRC_IP: 10.9.0.5 | DST_IP: 8.8.8.8 |
[+] Type: Request | Code: 0 | Checksum: 53199 | Seq: 1
[+] Payload: ⓪SF`

[+] No.: 3 | Protocol: ICMP | SRC_IP: 8.8.8.8 | DST_IP: 10.9.0.5 |
[+] Type: Reply | Code: 0 | Checksum: 55247 | Seq: 1
[+] Payload: ⓪SF`

[+] No.: 4 | Protocol: ICMP | SRC_IP: 10.9.0.5 | DST_IP: 8.8.8.8 |
[+] Type: Request | Code: 0 | Checksum: 22727 | Seq: 2
[+] Payload: ⓪SF`

[+] No.: 5 | Protocol: ICMP | SRC_IP: 8.8.8.8 | DST_IP: 10.9.0.5 |
[+] Type: Reply | Code: 0 | Checksum: 24775 | Seq: 2
[+] Payload: ⓪SF`

[+] No.: 6 | Protocol: ICMP | SRC_IP: 10.9.0.5 | DST_IP: 8.8.8.8 |
[+] Type: Request | Code: 0 | Checksum: 22698 | Seq: 3
[+] Payload: ⓪SF`

[+] No.: 7 | Protocol: ICMP | SRC_IP: 8.8.8.8 | DST_IP: 10.9.0.5 |
[+] Type: Reply | Code: 0 | Checksum: 24746 | Seq: 3
[+] Payload: ⓪SF`
```

```
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=74.3 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=74.3 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=115 time=74.6 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=115 time=75.4 ms
^Z
[27]+  Stopped                 ping 8.8.8.8
root@0dc37fd4c48f:/# 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=115 time=75.1 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=73.9 ms
^Z
[28]+  Stopped                 ping 8.8.8.8
root@0dc37fd4c48f:/# 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=115 time=75.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=75.6 ms
^Z
[29]+  Stopped                 ping 8.8.8.8
root@0dc37fd4c48f:/# ping 22.22.22.22
PING 22.22.22.22 (22.22.22.22) 56(84) bytes of data.
^Z
[30]+  Stopped                 ping 22.22.22.22
root@0dc37fd4c48f:/# pint 10.9.0.1
bash: pint: command not found
root@0dc37fd4c48f:/# ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data.
64 bytes from 10.9.0.1: icmp_seq=1 ttl=64 time=0.092 ms
64 bytes from 10.9.0.1: icmp_seq=2 ttl=64 time=0.069 ms
64 bytes from 10.9.0.1: icmp_seq=3 ttl=64 time=0.069 ms
64 bytes from 10.9.0.1: icmp_seq=4 ttl=64 time=0.086 ms
64 bytes from 10.9.0.1: icmp_seq=5 ttl=64 time=0.068 ms
^Z
[31]+  Stopped                 ping 10.9.0.1
root@0dc37fd4c48f:/#
```

314078023
204420590

TCP PORT Range:

In the next expression we sniffed tcp packets between ports 10 to 100

```
static char filter_exp[] = "tcp portrange dst 10-100";
```

As we can see in the next screenshot when we send a telnet (port 23) packet it is being sniffed, but when we surf via HTTPS (port 443) there are no packets being sniffed:

```
[+] Payload: 	SF`

^Z
[31]+  Stopped                 ./sniff.o
root@VM:/volumes/Sniff_Spoof# ./sniff.o
[+] No.: 2 | Protocol: TCP | SRC_PORT 32904 | DST_PORT 23
[+] SRC_IP: 10.9.0.5 | DST_IP: 172.217.169.69 | Checksum 24667
[+] Payload:

[+] No.: 3 | Protocol: TCP | SRC_PORT 32904 | DST_PORT 23
[+] SRC_IP: 10.9.0.5 | DST_IP: 172.217.169.69 | Checksum 24667
[+] Payload:

[+] No.: 4 | Protocol: TCP | SRC_PORT 32904 | DST_PORT 23
[+] SRC_IP: 10.9.0.5 | DST_IP: 172.217.169.69 | Checksum 24667
[+] Payload:

[+] No.: 5 | Protocol: TCP | SRC_PORT 32904 | DST_PORT 23
[+] SRC_IP: 10.9.0.5 | DST_IP: 172.217.169.69 | Checksum 24667
[+] Payload:

[+] No.: 6 | Protocol: TCP | SRC_PORT 32904 | DST_PORT 23
[+] SRC_IP: 10.9.0.5 | DST_IP: 172.217.169.69 | Checksum 24667
[+] Payload:

[+] No.: 7 | Protocol: TCP | SRC_PORT 32904 | DST_PORT 23
[+] SRC_IP: 10.9.0.5 | DST_IP: 172.217.169.69 | Checksum 24667
[+] Payload:

[+] No.: 8 | Protocol: TCP | SRC_PORT 32904 | DST_PORT 23
[+] SRC_IP: 10.9.0.5 | DST_IP: 172.217.169.69 | Checksum 24667
[+] Payload:

]
```

314078023
204420590

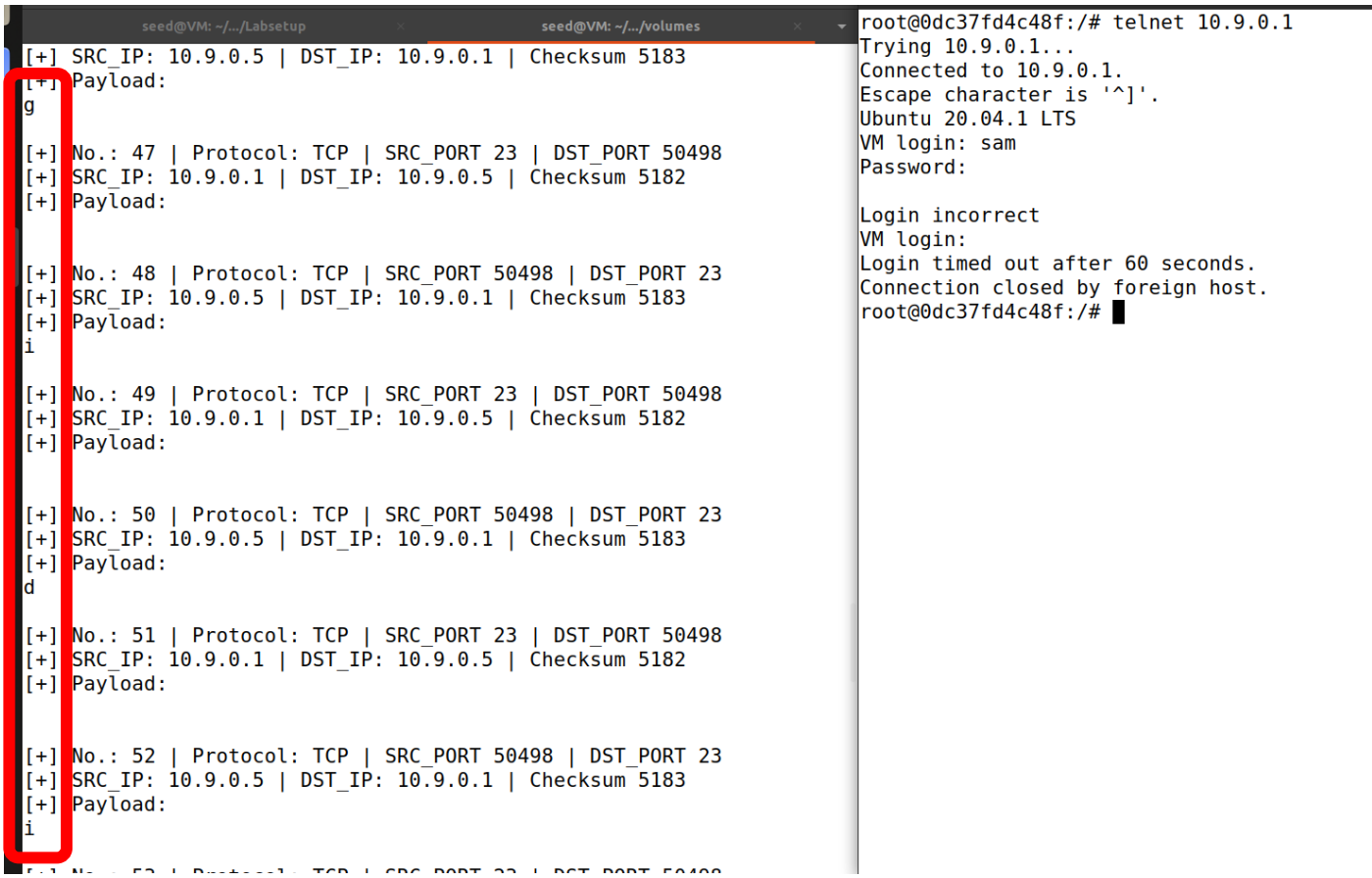
Task 2.1C:

At the end of every Telnet packet there is a payload which contains the transferred data in plain text. Because Telnet is not encrypted, we can sniff this data and print it out with our sniffer program. We will try to connect to our own VM via telnet with these credentials:

Username: sam

Password: gidi

Telnet splits the passwords and transfers it in different packets, one for every char, so in our case, we will have four packets containing the word “gidi” as show in the next output:



314078023
204420590

Task 2.2A:

As we can see in the next screenshot we ping a non-existent IP and get a response from our spoofing program that sniffs and awaits the request.

```
################################
       Spoofing ICMP Packet
################################

[+] No.: 5 | Protocol: ICMP | SRC_IP: 1.2.3.4 | DST_IP: 10.9.0.5 |
[+] Type: Reply | Code: 0 | Checksum: 57933 | Seq: 2
[+] Payload: ⍰YF`

[+] No.: 6 | Protocol: ICMP | SRC_IP: 10.9.0.5 | DST_IP: 1.2.3.4 |
[+] Type: Request | Code: 0 | Checksum: 45893 | Seq: 3
[+] Payload: ⍰YF`

################################
       Spoofing ICMP Packet
################################

[+] No.: 7 | Protocol: ICMP | SRC_IP: 1.2.3.4 | DST_IP: 10.9.0.5 |
[+] Type: Reply | Code: 0 | Checksum: 47941 | Seq: 3
[+] Payload: ⍰YF`

[+] No.: 8 | Protocol: ICMP | SRC_IP: 10.9.0.5 | DST_IP: 1.2.3.4 |
[+] Type: Request | Code: 0 | Checksum: 22326 | Seq: 4
[+] Payload: ⍰YF`

################################
       Spoofing ICMP Packet
################################

[+] No.: 9 | Protocol: ICMP | SRC_IP: 1.2.3.4 | DST_IP: 10.9.0.5 |
[+] Type: Reply | Code: 0 | Checksum: 24374 | Seq: 4
[+] Payload: ⍰YF`
```

```
root@0dc37fd4c48f:/# 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=128 time=44.1 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=128 time=95.1 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=128 time=38.9 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=128 time=80.7 ms
^Z
[32]+  Stopped                 ping 1.2.3.4
root@0dc37fd4c48f:/#
```

314078023
204420590

Task 2.2B:

As we can see in our spoofed ICMP request we have sent google a spoofed request from one of our containers (the host) which is in IP 10.9.0.5.
We see the packet being sent to 10.0.2.5 because it is the host VM that passes the packet down to the container which. The proof to that is that the TTL is being reduced accordingly an we can also see the detail in the Ethernet layer via the MAC addresses passing down those packets:

icmp

No.	Time	Source	Destination	Protocol	Length	Info
10	1.3612527…	10.0.2.5	8.8.8.8	ICMP	64	Echo (ping) request id=0x0000, seq=0/0, ttl=20 (reply in 11)
11	1.4360323…	8.8.8.8	10.0.2.5	ICMP	64	Echo (ping) reply id=0x0000, seq=0/0, ttl=116 (request in 10)
12	1.4360657…	8.8.8.8	10.9.0.5	ICMP	64	Echo (ping) reply id=0x0000, seq=0/0, ttl=115
13	1.4360826…	8.8.8.8	10.9.0.5	ICMP	64	Echo (ping) reply id=0x0000, seq=0/0, ttl=115

Questions:

4. **Can we set arbitrary IP length?**
   No we can't. because the IP header have specific RFC specifications for size and content, we can't just choose any size we want because the the OS won't process the packet. The IP header length should be the IP header + ICMP header

5. **Do we have to calculate checksum?**
   No the OS calculates the checksum automatically before transmitting it regardless of the value.

6. **Why do we need root access?**
   as we mentioned before, any program that required access to hardware and drivers needs root access (Like the example we gave earlier of CAP_NET_RAW library). Without root access we cannot fully control the network driver and interfaces and insert the information and data we want.

314078023
204420590

Task2.3

As we can see at the next screenshot out code send a response to any icmp request regardless the destination IP (we gave 3 examples)

```
[+] Payload: }F`

################################
       Spoofing ICMP Response
################################
[+] No.: 19 | Protocol: ICMP | SRC_IP: 1.2.3.4 | DST_IP: 10.9.0.5 |
[+] Type: Reply | Code: 0 | Checksum: 35999 | Seq: 3
[+] Payload: }F`

[+] No.: 20 | Protocol: ICMP | SRC_IP: 10.9.0.5 | DST_IP: 1.2.3.4 |
[+] Type: Request | Code: 0 | Checksum: 52635 | Seq: 4
[+] Payload: }F`

################################
       Spoofing ICMP Response
################################
[+] No.: 21 | Protocol: ICMP | SRC_IP: 1.2.3.4 | DST_IP: 10.9.0.5 |
[+] Type: Reply | Code: 0 | Checksum: 54683 | Seq: 4
[+] Payload: }F`

[+] No.: 22 | Protocol: ICMP | SRC_IP: 10.9.0.5 | DST_IP: 1.2.3.4 |
[+] Type: Request | Code: 0 | Checksum: 45457 | Seq: 5
[+] Payload: }F`

################################
       Spoofing ICMP Response
################################
[+] No.: 23 | Protocol: ICMP | SRC_IP: 1.2.3.4 | DST_IP: 10.9.0.5 |
[+] Type: Reply | Code: 0 | Checksum: 47505 | Seq: 5
[+] Payload: }F`

^Z
[1]+  Stopped                 ./sniff.o
root@VM:/volumes/Sniff_Spoof#
```

```
[03/08/21]seed@VM:~$ docksh 0
root@0dc37fd4c48f:/# 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=128 time=72.6 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=77.0 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=2 ttl=128 time=13.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=74.7 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=3 ttl=128 time=71.2 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=74.5 ms (DUP!)
64 bytes from 8.8.8.8: icmp_seq=4 ttl=128 time=23.4 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=115 time=80.1 ms (DUP!)
^Z
[1]+  Stopped                 ping 8.8.8.8
root@0dc37fd4c48f:/# 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=128 time=19.4 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=128 time=104 ms
64 bytes from 1.2.3.4: icmp_seq=3 ttl=128 time=92.5 ms
64 bytes from 1.2.3.4: icmp_seq=4 ttl=128 time=28.1 ms
64 bytes from 1.2.3.4: icmp_seq=5 ttl=128 time=69.4 ms
^Z
[2]+  Stopped                 ping 1.2.3.4
root@0dc37fd4c48f:/# ping 127.234.111.111
PING 127.234.111.111 (127.234.111.111) 56(84) bytes of data.
64 bytes from 127.234.111.111: icmp_seq=1 ttl=64 time=0.028 ms
64 bytes from 127.234.111.111: icmp_seq=2 ttl=64 time=0.032 ms
64 bytes from 127.234.111.111: icmp_seq=3 ttl=64 time=0.032 ms
64 bytes from 127.234.111.111: icmp_seq=4 ttl=64 time=0.035 ms
64 bytes from 127.234.111.111: icmp_seq=5 ttl=64 time=0.039 ms
64 bytes from 127.234.111.111: icmp_seq=6 ttl=64 time=0.034 ms
64 bytes from 127.234.111.111: icmp_seq=7 ttl=64 time=0.042 ms
^Z
[3]+  Stopped                 ping 127.234.111.111
root@0dc37fd4c48f:/#
```