CENX586 Network Security

Project #1
SEED Labs: Sniffing and Spoofing

Submitted By

Mohammed Shahzad

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:

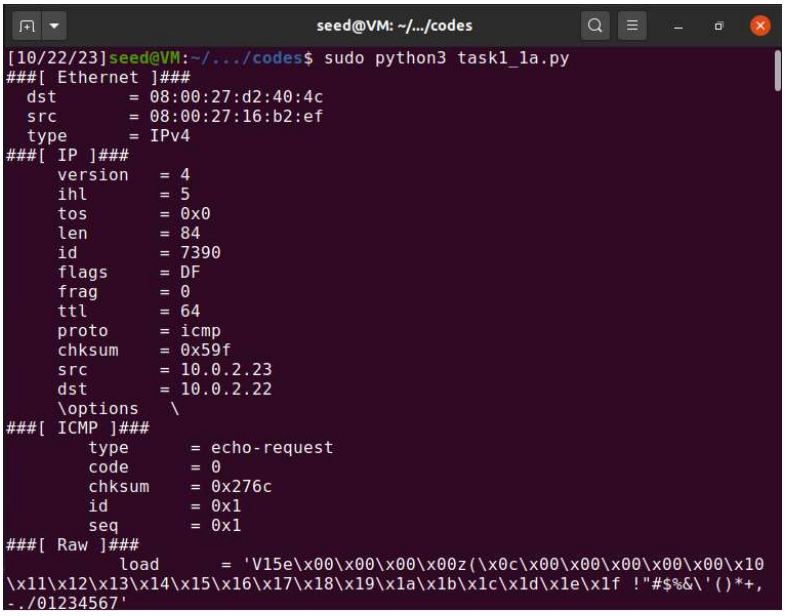
Host IP Address: 10.0.2.22

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:



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:

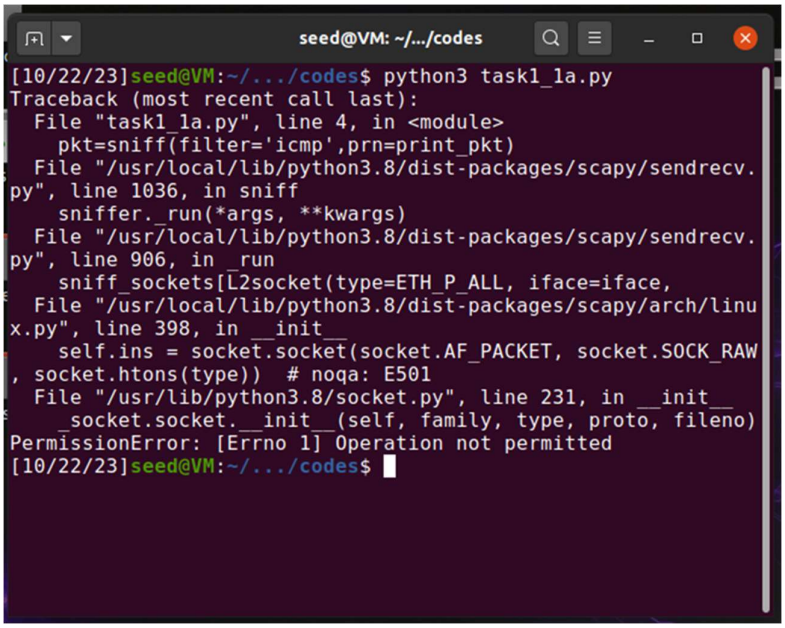
```
from scapy.all import *

def print_pkt(pkt):

        pkt.show() #show packet details

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

Evidence:



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.

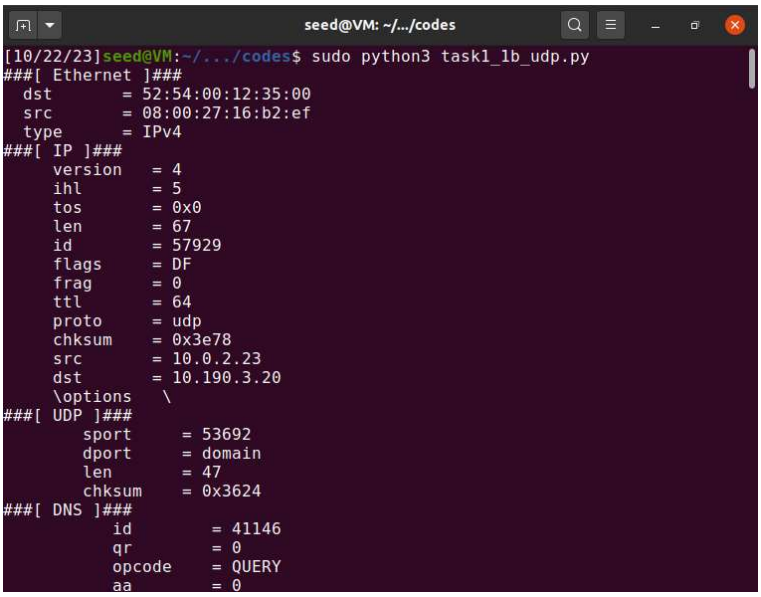
➔ Capture UDP packets:

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:



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:

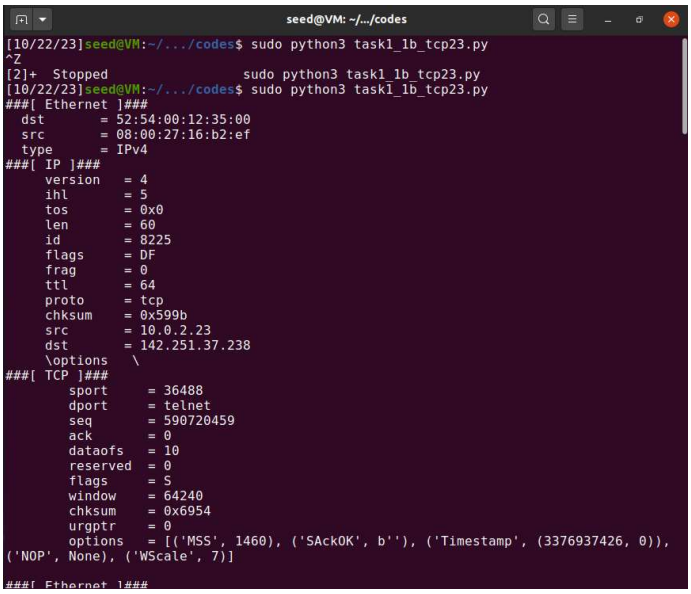
Generate traffic at Victim machine: $ nc -t google.com 23

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:



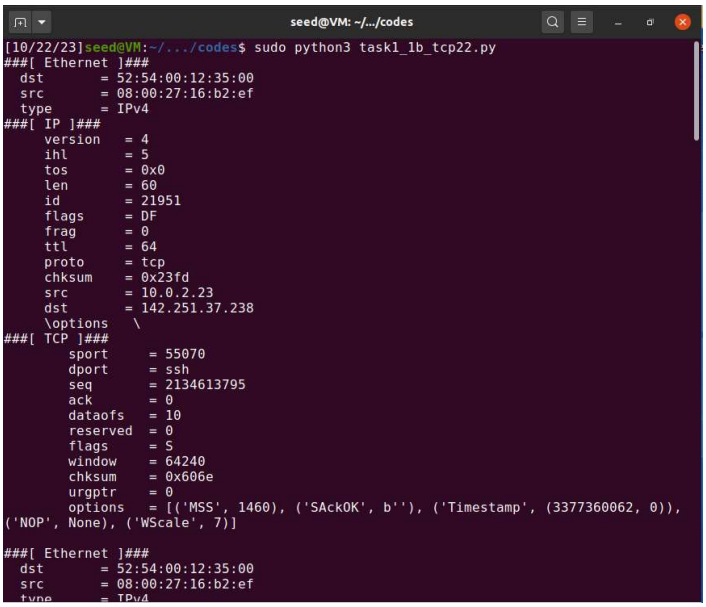
Explanation: The above result is generated using BPF filter to get only UDP packets in transit. Sniff() is used for packet capture, pkt.show() displays the TCP packets captured at destination port 23 and source 10.0.2.23.

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:



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:

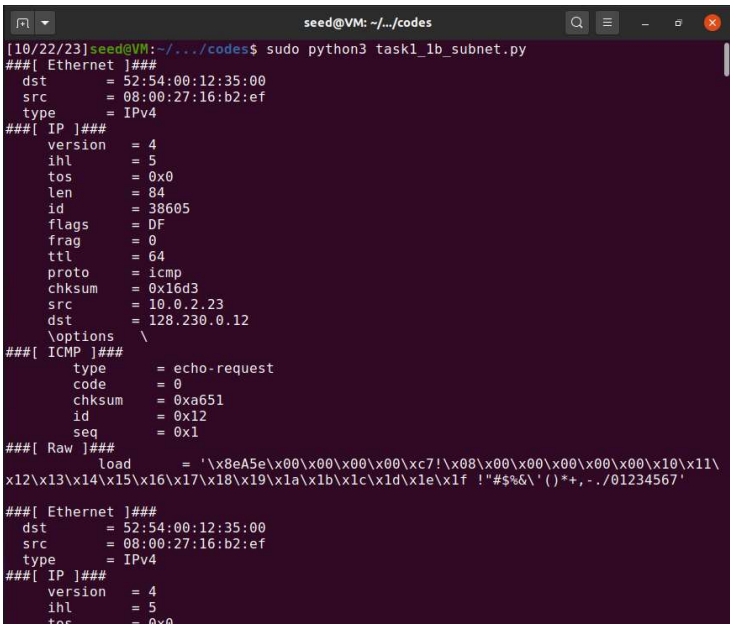
Subnit: 128.230.0.0/16; Filter=”net 128.230.0.0/16”

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:



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:

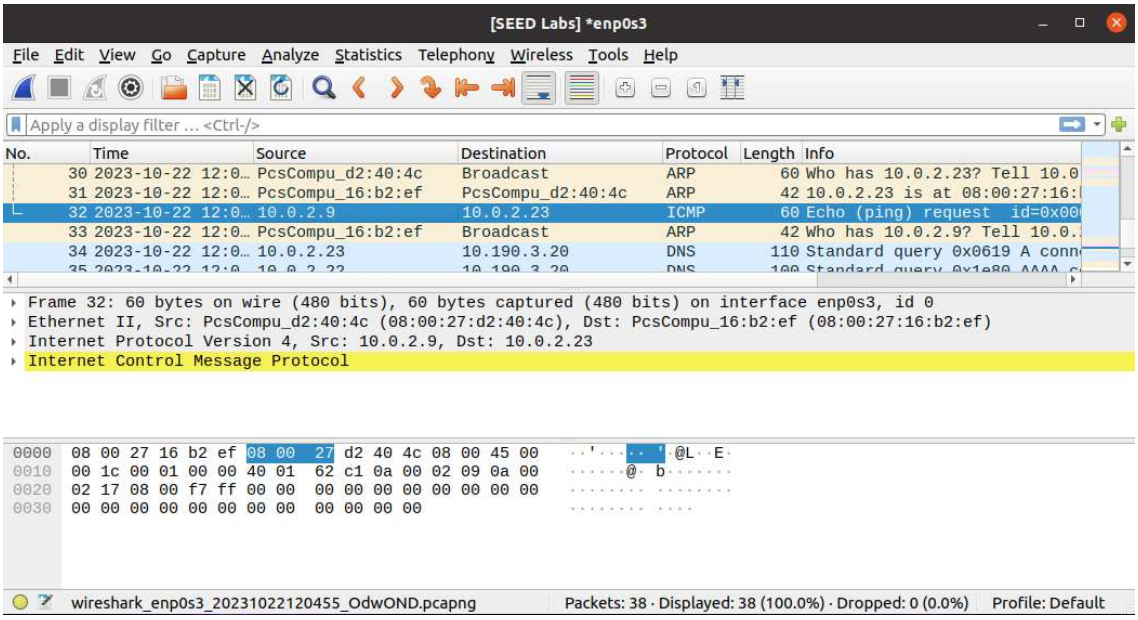
Code with explanation:

```
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)



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

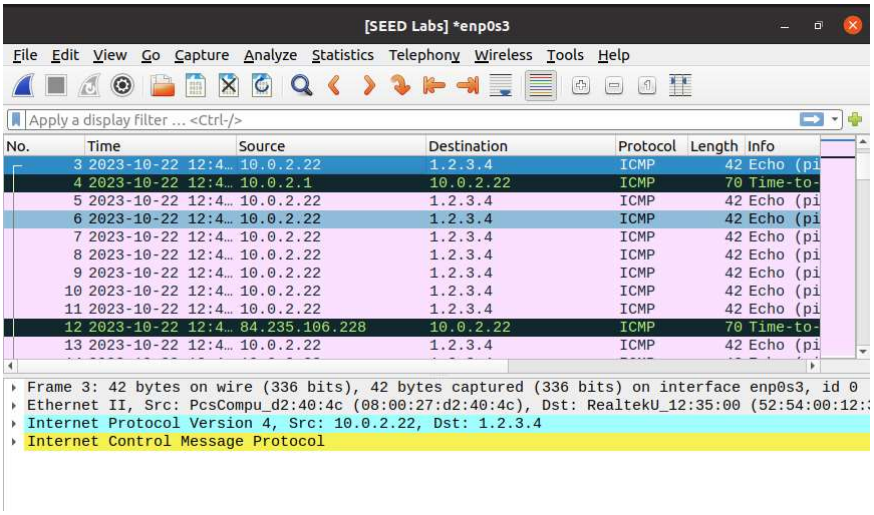
The objective of this task is to use Scapy to estimate the distance, in terms of number of routers, between your VM and a selected destination. We will repeat this procedure until our packet finally reach the destination.

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

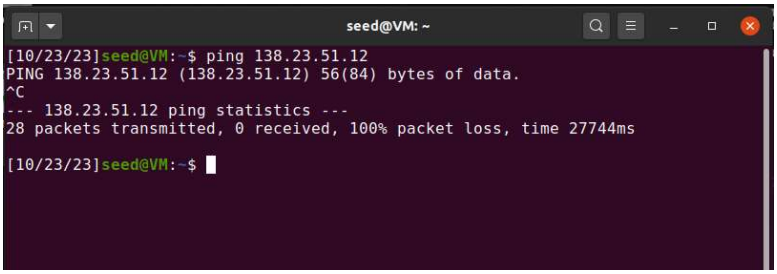
Code with explanation:

```
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)
```

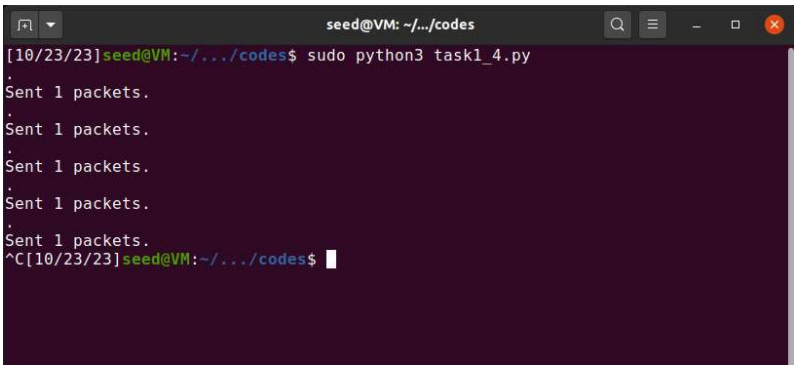
Evidence:

Initially at victim (10.0.2.23): $ ping 138.23.51.12

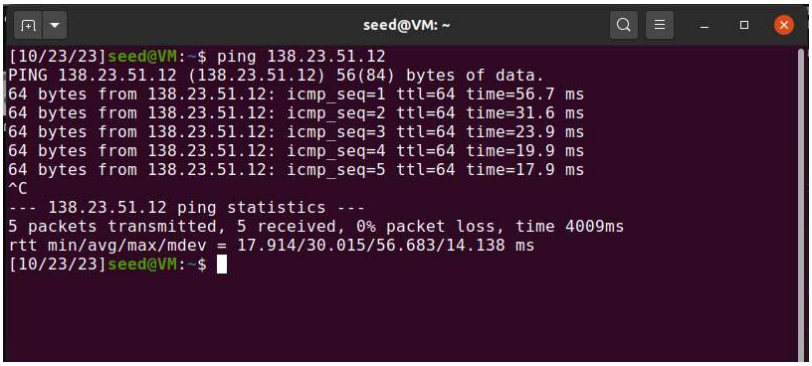
Gets no reply



Run sniff_then_spoof program at Host (10.0.2.22):
$ sudo python3 task1_4.py



10.0.2.23 receives reply for: $ ping 138.23.51.12



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.

***