Kevin Huang
CSS 537
Assignment 1: Packet Sniffing and Spoofing
01/24/2022

Introduction

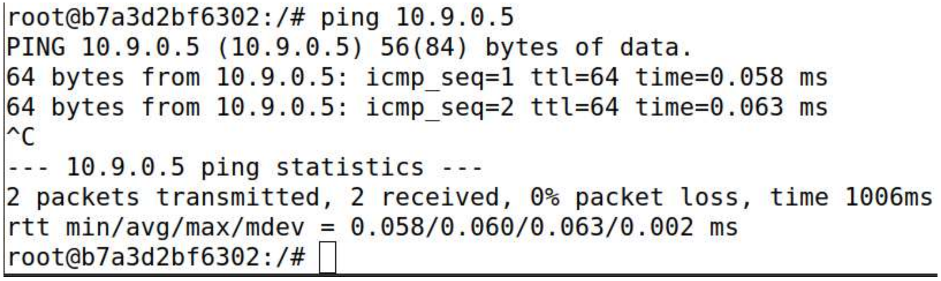
The purpose of this assignment is for us to better understand sniffing and spoofing through writing our own code to perform these tasks. It is important to not only know how to use sniffing and spoofing tools but to also know how they work. This lab walks us through sniffing and spoofing using two methods. The first method utilizes the pcap library while the second method uses raw sockets.

Task 1.1: Sniffing Packets

Task 1.1A:

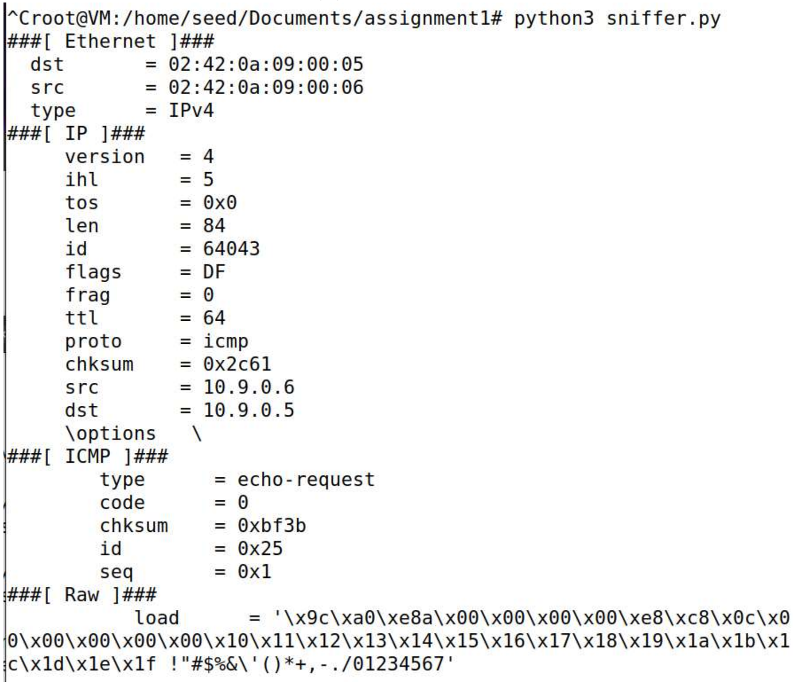
In the first screenshot, I used hostB (10.9.0.6) to ping hostA (10.9.0.5). We can see that this operation was successful and 2 packets were sent and 2 responses were received. The sniffer program, sniffer.py, was run with root privilege on the attacker machine and in the second screenshot, we can see that it is successfully capturing packets (both the request and the reply). When I try and run sniffer.py as seed, the operation was denied. This is because you need root privilege to run scapy to gain access to all the packets.

Screenshot 1.1A.1: ping 10.9.0.5 from 10.9.0.6



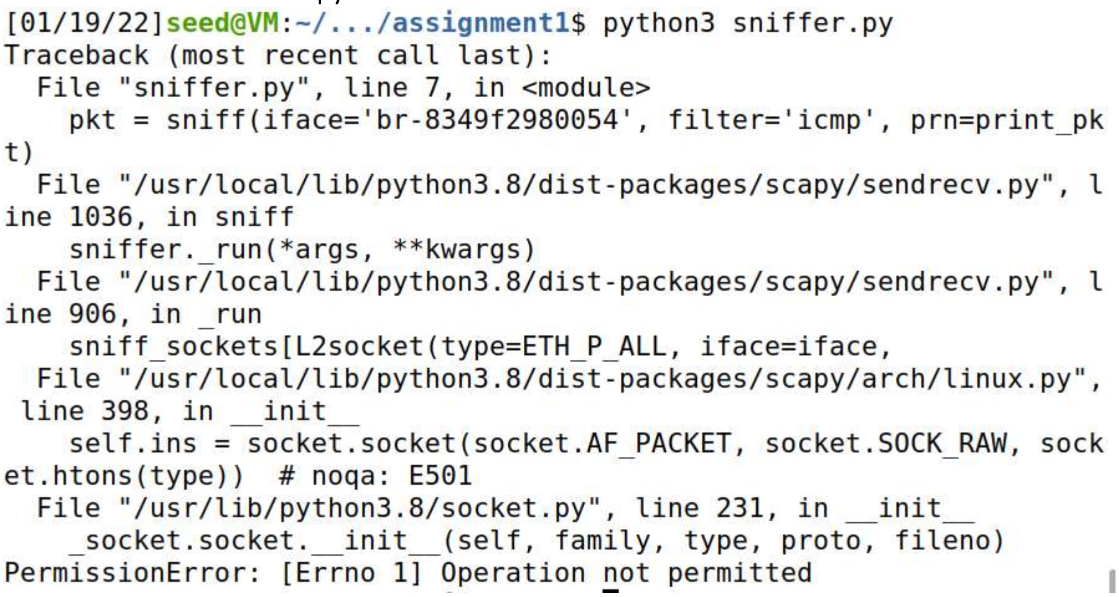
```
root@b7a3d2bf6302:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.058 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.063 ms
^C
--- 10.9.0.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1006ms
rtt min/avg/max/mdev = 0.058/0.060/0.063/0.002 ms
root@b7a3d2bf6302:/# 
```

Screenshot 1.1A.2: sniffer.py run as root



```
^Croot@VM:/home/seed/Documents/assignment1# python3 sniffer.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 64043
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x2c61
     src       = 10.9.0.6
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xbf3b
        id        = 0x25
        seq       = 0x1
###[ Raw ]###
           load      = '\x9c\xa0\xe8a\x00\x00\x00\x00\xe8\xc8\x0c\x0
0\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1
c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

Screenshot 1.1A.3: sniffer.py run not as root

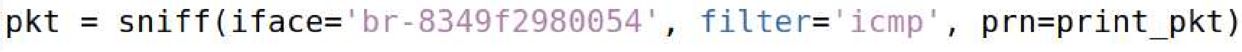


```
[01/19/22]seed@VM:~/.../assignment1$ python3 sniffer.py
Traceback (most recent call last):
  File "sniffer.py", line 7, in <module>
    pkt = sniff(iface='br-8349f2980054', filter='icmp', prn=print_pk
t)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", l
ine 1036, in sniff
    sniffer._run(*args, **kwargs)
  File "/usr/local/lib/python3.8/dist-packages/scapy/sendrecv.py", l
ine 906, in _run
    sniff_sockets[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, sock
et.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
```

Task 1.1B:

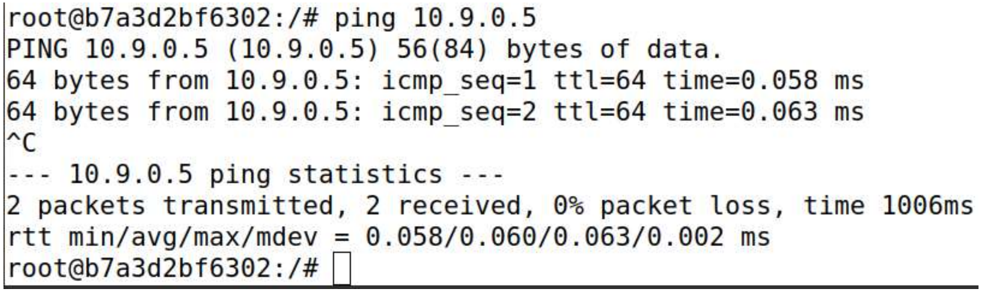
The first 3 screenshots are similar to task 1.1A where only the ICMP packets are captured. For the second bullet point for capturing TCP packets from an IP address with destination port 23, I referred to the Berkeley packet filters and used '&&' to concatenate the 3 conditions. Screenshots 4-6 show this process. Since port 23 is for Telnet, I sent a Telnet request from hostB (10.9.0.6). The TCP packet was successfully captured. For the third bullet point, I used the 'net' filter for subnet 128.230.0.0/16 and pinged 128.230.0.8 which is within the subnet (screenshots 7-9). The screenshots show success for capturing packets with each of the 3 different filters.

Screenshot 1.1B.1: code for ICMP filter



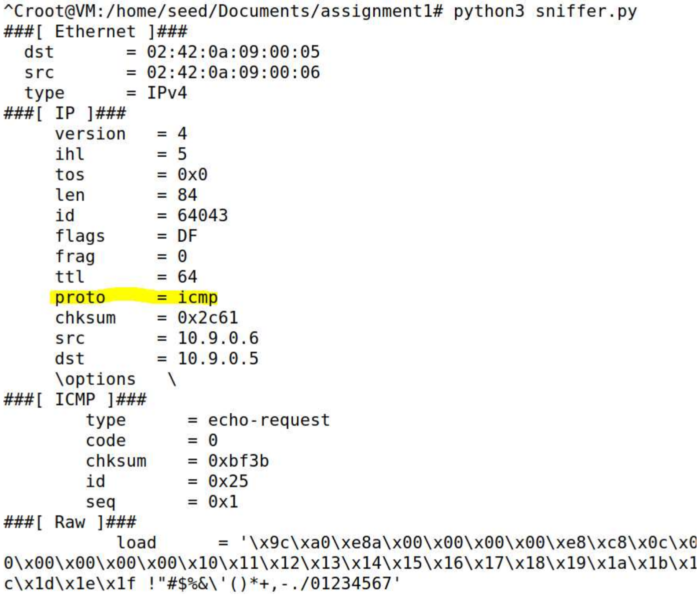
```
pkt = sniff(iface='br-8349f2980054', filter='icmp', prn=print_pkt)
```

Screenshot 1.1B.2: ping command to test ICMP filter



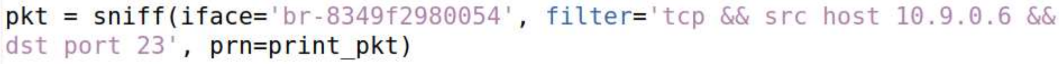
```
root@b7a3d2bf6302:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=64 time=0.058 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=64 time=0.063 ms
^C
--- 10.9.0.5 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1006ms
rtt min/avg/max/mdev = 0.058/0.060/0.063/0.002 ms
root@b7a3d2bf6302:/# 
```

Screenshot 1.1B.3: Captured ICMP packet



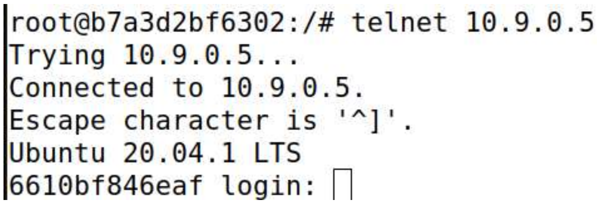
```
^Croot@VM:/home/seed/Documents/assignment1# python3 sniffer.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 64043
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0x2c61
     src       = 10.9.0.6
     dst       = 10.9.0.5
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0xbf3b
        id        = 0x25
        seq       = 0x1
###[ Raw ]###
           load      = '\x9c\xa0\xe8a\x00\x00\x00\x00\xe8\xc8\x0c\x0
0\x00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1
c\x1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

Screenshot 1.1B.4: Code for TCP filter



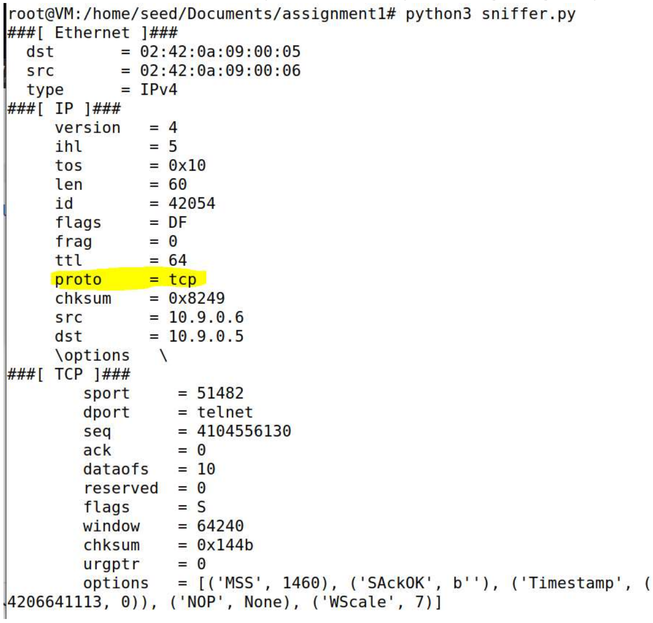
```
pkt = sniff(iface='br-8349f2980054', filter='tcp && src host 10.9.0.6 &&
dst port 23', prn=print_pkt)
```

Screenshot 1.1B.5: run telnet



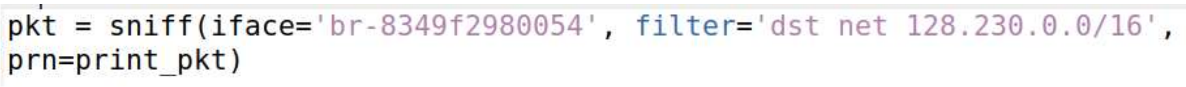
```
root@b7a3d2bf6302:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
6610bf846eaf login: 
```

Screenshot 1.1B.6: Captured TCP packet going to port 23 (telnet)



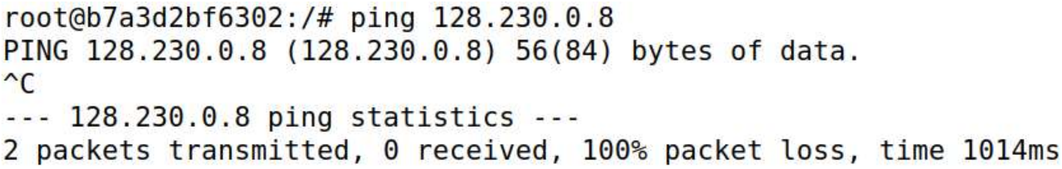
```
root@VM:/home/seed/Documents/assignment1# python3 sniffer.py
###[ Ethernet ]###
  dst       = 02:42:0a:09:00:05
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x10
     len       = 60
     id        = 42054
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = tcp
     chksum    = 0x8249
     src       = 10.9.0.6
     dst       = 10.9.0.5
     \options   \
###[ TCP ]###
        sport     = 51482
        dport     = telnet
        seq       = 4104556130
        ack       = 0
        dataofs   = 10
        reserved  = 0
        flags     = S
        window    = 64240
        chksum    = 0x144b
        urgptr    = 0
        options   = [('MSS', 1460), ('SAckOK', b''), ('Timestamp', (
4206641113, 0)), ('NOP', None), ('WScale', 7)]
```

Screenshot 1.1B.7: Code for subnet filter



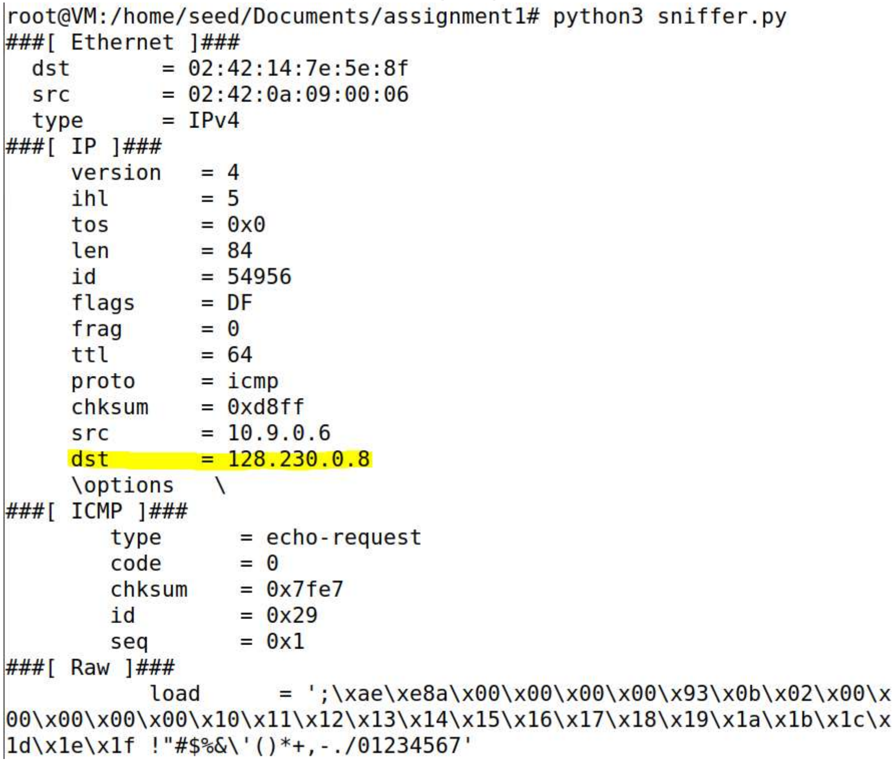
```
pkt = sniff(iface='br-8349f2980054', filter='dst net 128.230.0.0/16',
prn=print_pkt)
```

Screenshot 1.1B.8: Ping a host within subnet



```
root@b7a3d2bf6302:/# ping 128.230.0.8
PING 128.230.0.8 (128.230.0.8) 56(84) bytes of data.
^C
--- 128.230.0.8 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1014ms
```

Screenshot 1.1B.9: Captured packet going to host within subnet

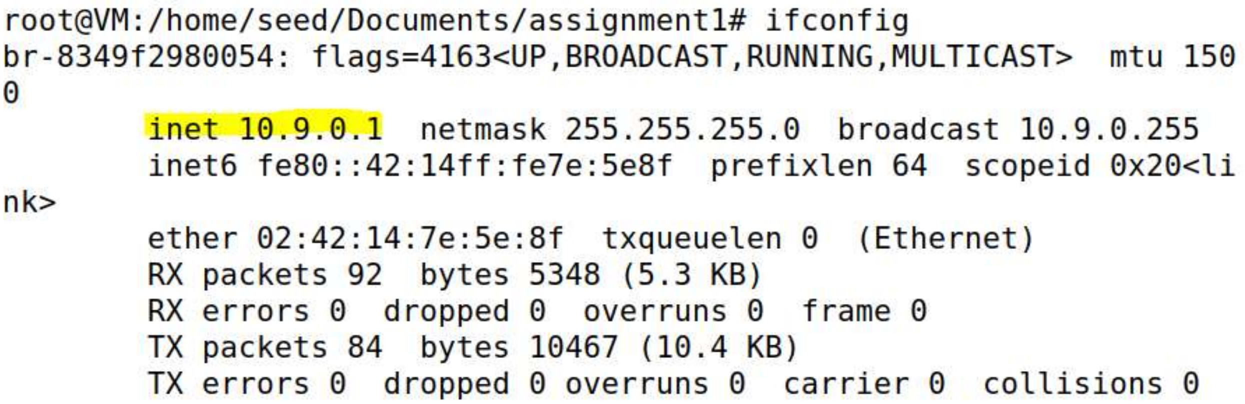


```
root@VM:/home/seed/Documents/assignment1# python3 sniffer.py
###[ Ethernet ]###
  dst       = 02:42:14:7e:5e:8f
  src       = 02:42:0a:09:00:06
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     tos       = 0x0
     len       = 84
     id        = 54956
     flags     = DF
     frag      = 0
     ttl       = 64
     proto     = icmp
     chksum    = 0xd8ff
     src       = 10.9.0.6
     dst       = 128.230.0.8
     \options   \
###[ ICMP ]###
        type      = echo-request
        code      = 0
        chksum    = 0x7fe7
        id        = 0x29
        seq       = 0x1
###[ Raw ]###
           load      = ';\xae\xe8a\x00\x00\x00\x00\x93\x0b\x02\x00\x
00\x00\x00\x00\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x
1d\x1e\x1f !"#$%&\'()*+,-./01234567'
```

Task 1.2: Spoofing ICMP Packets

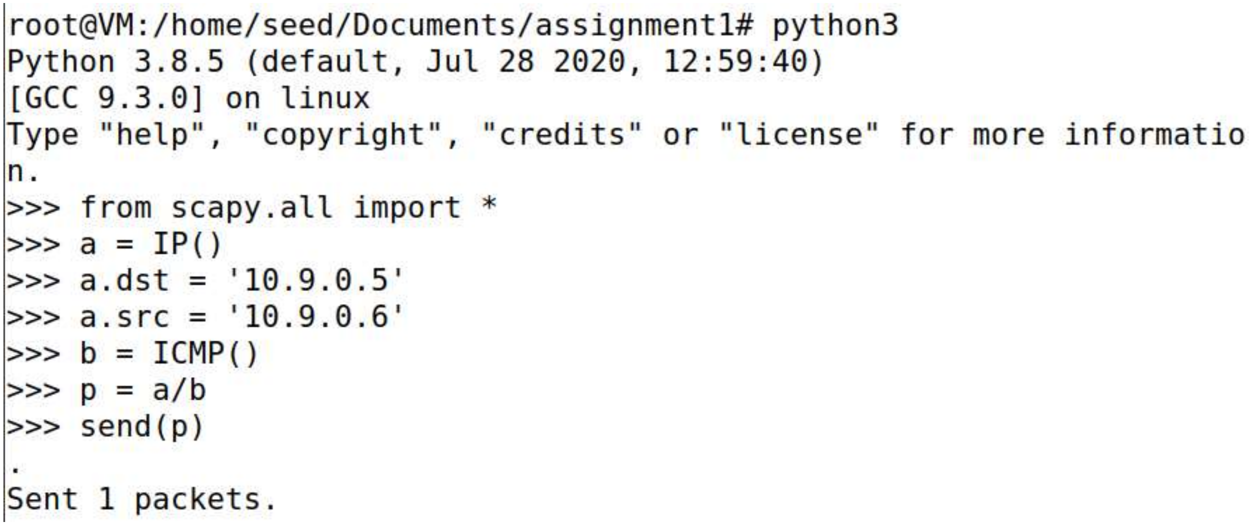
I first checked the IP address of the machine that I am using to spoof, which was 10.9.0.1. I then followed instructions for creating an ICMP packet. I changed an additional field, ‘a.src’ to a different IP address. I sent the packet and used Wireshark to observe the traffic. I found that there was indeed an ICMP reply that was sent to the spoofed source IP address.

Screenshot 1.2.1: checked attacker IP address



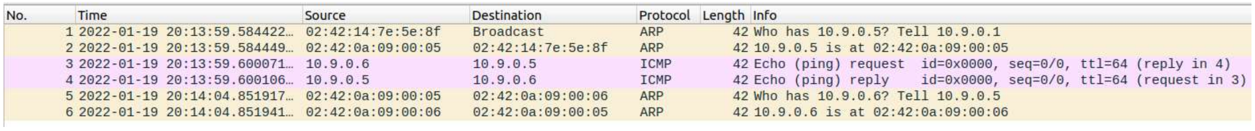
```
root@VM:/home/seed/Documents/assignment1# ifconfig
br-8349f2980054: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 150
0
        inet 10.9.0.1  netmask 255.255.255.0  broadcast 10.9.0.255
        inet6 fe80::42:14ff:fe7e:5e8f  prefixlen 64  scopeid 0x20<li
nk>
        ether 02:42:14:7e:5e:8f  txqueuelen 0  (Ethernet)
        RX packets 92  bytes 5348 (5.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 84  bytes 10467 (10.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Screenshot 1.2.2: Code for sending spoofed packet



```
root@VM:/home/seed/Documents/assignment1# python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more informatio
n.
>>> from scapy.all import *
>>> a = IP()
>>> a.dst = '10.9.0.5'
>>> a.src = '10.9.0.6'
>>> b = ICMP()
>>> p = a/b
>>> send(p)
.
Sent 1 packets.
```

Screenshot 1.2.3: Wireshark shows spoofed packet

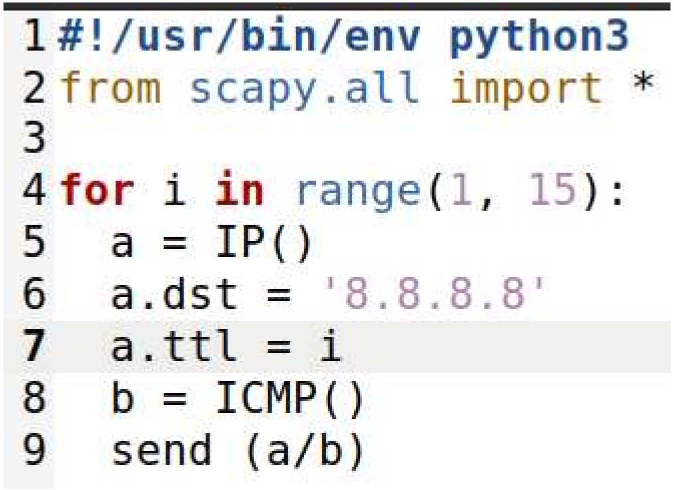


No.	Time	Source	Destination	Protocol	Length	Info
1	2022-01-19 20:13:59.584422…	02:42:14:7e:5e:8f	Broadcast	ARP	42	Who has 10.9.0.5? Tell 10.9.0.1
2	2022-01-19 20:13:59.584449…	02:42:0a:09:00:05	02:42:14:7e:5e:8f	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05
3	2022-01-19 20:13:59.600071…	10.9.0.6	10.9.0.5	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=64 (reply in 4)
4	2022-01-19 20:13:59.600106…	10.9.0.5	10.9.0.6	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 3)
5	2022-01-19 20:14:04.851917…	02:42:0a:09:00:05	02:42:0a:09:00:06	ARP	42	Who has 10.9.0.6? Tell 10.9.0.5
6	2022-01-19 20:14:04.851941…	02:42:0a:09:00:06	02:42:0a:09:00:05	ARP	42	10.9.0.6 is at 02:42:0a:09:00:06

Task 1.3: Traceroute

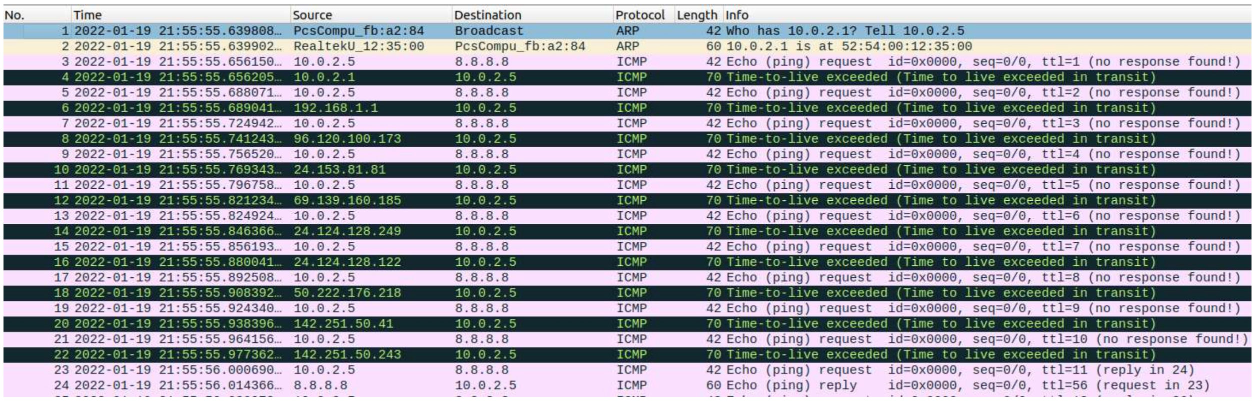
For this task, in my tracer.py code, I used a for loop in my python program to increment the TTL from 1 to 15. From Wireshark, I found that it takes estimated 11 hops for me to reach google.com. We can see in the screenshot that ICMP error messages were received for TTL's 1 through 10 and I received a reply when the TTL was set to 11.

Screenshot 1.3.1: Code for incrementing TTL



```python
#!/usr/bin/env python3
from scapy.all import *

for i in range(1, 15):
    a = IP()
    a.dst = '8.8.8.8'
    a.ttl = i
    b = ICMP()
    send (a/b)
```

Screenshot 1.3.2: Wireshark results

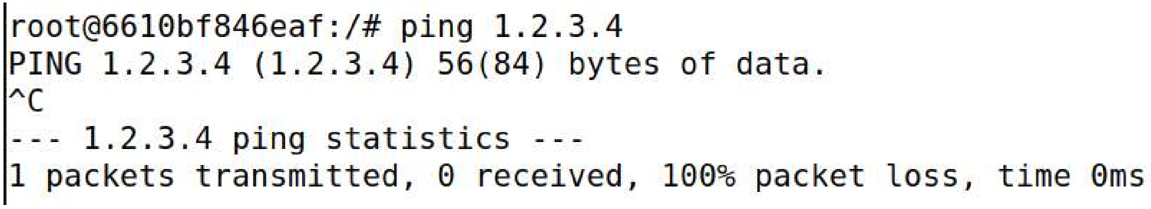


No.	Time	Source	Destination	Protocol	Length	Info
1	2022-01-19 21:55:55.639808…	PcsCompu_fb:a2:84	Broadcast	ARP	42	Who has 10.0.2.1? Tell 10.0.2.5
2	2022-01-19 21:55:55.639902…	RealtekU_12:35:00	PcsCompu_fb:a2:84	ARP	60	10.0.2.1 is at 52:54:00:12:35:00
3	2022-01-19 21:55:55.656150…	10.0.2.5	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=1 (no response found!)
4	2022-01-19 21:55:55.656205…	10.0.2.1	10.0.2.5	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
5	2022-01-19 21:55:55.688071…	10.0.2.5	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=2 (no response found!)
6	2022-01-19 21:55:55.689041…	192.168.1.1	10.0.2.5	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
7	2022-01-19 21:55:55.724942…	10.0.2.5	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=3 (no response found!)
8	2022-01-19 21:55:55.741243…	96.120.100.173	10.0.2.5	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
9	2022-01-19 21:55:55.756520…	10.0.2.5	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=4 (no response found!)
10	2022-01-19 21:55:55.769343…	24.153.81.81	10.0.2.5	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
11	2022-01-19 21:55:55.796758…	10.0.2.5	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=5 (no response found!)
12	2022-01-19 21:55:55.821234…	69.139.160.185	10.0.2.5	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
13	2022-01-19 21:55:55.824924…	10.0.2.5	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=6 (no response found!)
14	2022-01-19 21:55:55.846366…	24.124.128.249	10.0.2.5	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
15	2022-01-19 21:55:55.856193…	10.0.2.5	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=7 (no response found!)
16	2022-01-19 21:55:55.880041…	24.124.128.122	10.0.2.5	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
17	2022-01-19 21:55:55.892508…	10.0.2.5	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=8 (no response found!)
18	2022-01-19 21:55:55.908392…	50.222.176.218	10.0.2.5	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
19	2022-01-19 21:55:55.924340…	10.0.2.5	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=9 (no response found!)
20	2022-01-19 21:55:55.938396…	142.251.50.41	10.0.2.5	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
21	2022-01-19 21:55:55.964156…	10.0.2.5	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=10 (no response found!)
22	2022-01-19 21:55:55.977362…	142.251.50.243	10.0.2.5	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
23	2022-01-19 21:55:56.000690…	10.0.2.5	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=11 (reply in 24)
24	2022-01-19 21:55:56.014366…	8.8.8.8	10.0.2.5	ICMP	60	Echo (ping) reply id=0x0000, seq=0/0, ttl=56 (request in 23)

Task 1.4: Sniffing and-then Spoofing

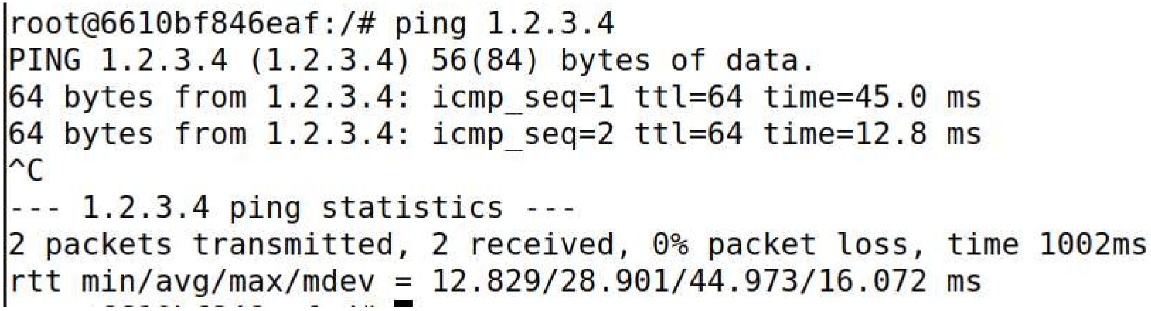
In this task, I modified the sniffer.py program from the previous tasks into the sniffspoof.py program. Instead of printing out the spoofed packet, the function now creates an ICMP reply and send it back to the requestor. For scenario 1 with IP address 1.2.3.4, in the first screenshot without the sniffspoof.py program, no reply was received. Once the sniffspoof program was running, it received replies back. For scenario 2, IP address 10.9.0.99 which is a non-existing host on the LAN, with the sniffspoof program running, I get a destination host unreachable error. This is because since the host is within the subnet, even though the IP address is spoofed, the ARP was not spoofed. There is no record for the MAC address in the ARP table we there's an error in the routing. For scenario 3, IP address 8.8.8.8, I get duplicate replies because I get a spoofed one and also another reply from the actual existing host.

Screenshot 1.4.1: ping 1.2.3.4, sniffspoof.py not running



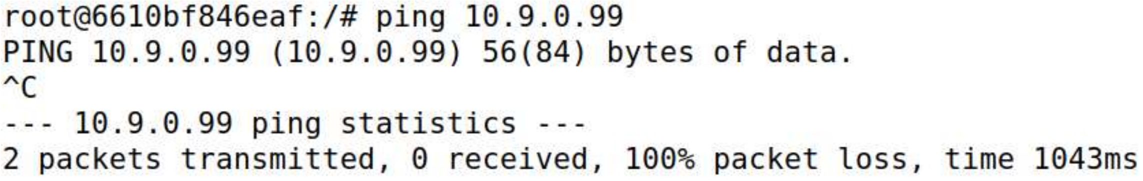
```
root@6610bf846eaf:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
^C
--- 1.2.3.4 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

Screenshot 1.4.2: ping 1.2.3.4, sniffspoof.py is running



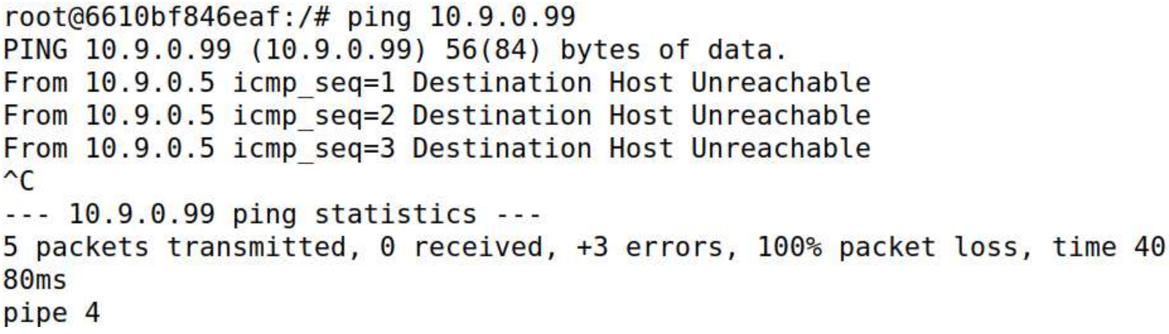
```
root@6610bf846eaf:/# ping 1.2.3.4
PING 1.2.3.4 (1.2.3.4) 56(84) bytes of data.
64 bytes from 1.2.3.4: icmp_seq=1 ttl=64 time=45.0 ms
64 bytes from 1.2.3.4: icmp_seq=2 ttl=64 time=12.8 ms
^C
--- 1.2.3.4 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 12.829/28.901/44.973/16.072 ms
```

Screenshot 1.4.3: ping 10.9.0.99, sniffspoof.py not running



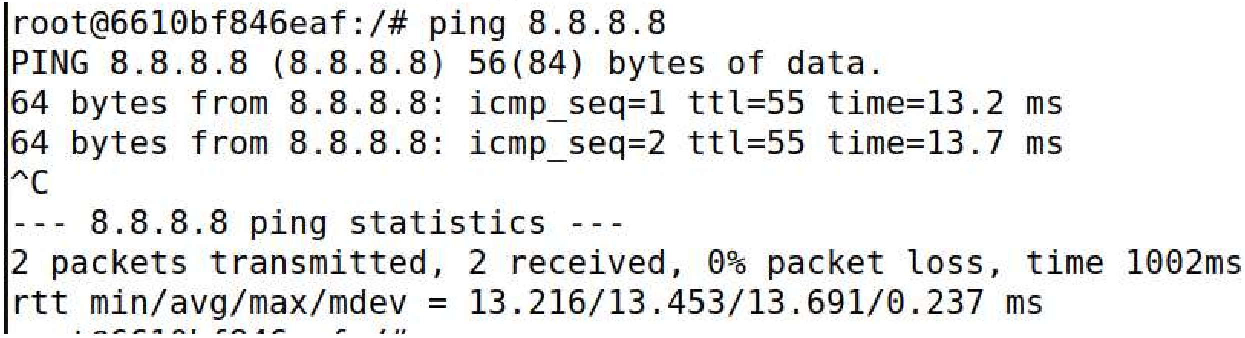
```
root@6610bf846eaf:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
^C
--- 10.9.0.99 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1043ms
```

Screenshot 1.4.4: ping 10.9.0.99, sniffspoof.py running



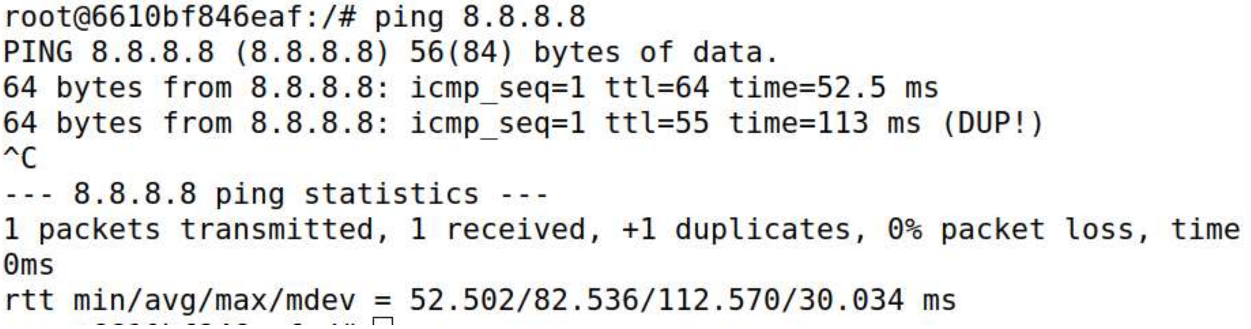
```
root@6610bf846eaf:/# ping 10.9.0.99
PING 10.9.0.99 (10.9.0.99) 56(84) bytes of data.
From 10.9.0.5 icmp_seq=1 Destination Host Unreachable
From 10.9.0.5 icmp_seq=2 Destination Host Unreachable
From 10.9.0.5 icmp_seq=3 Destination Host Unreachable
^C
--- 10.9.0.99 ping statistics ---
5 packets transmitted, 0 received, +3 errors, 100% packet loss, time 40
80ms
pipe 4
```

Screenshot 1.4.5: ping 8.8.8.8, sniffspoof.py not running



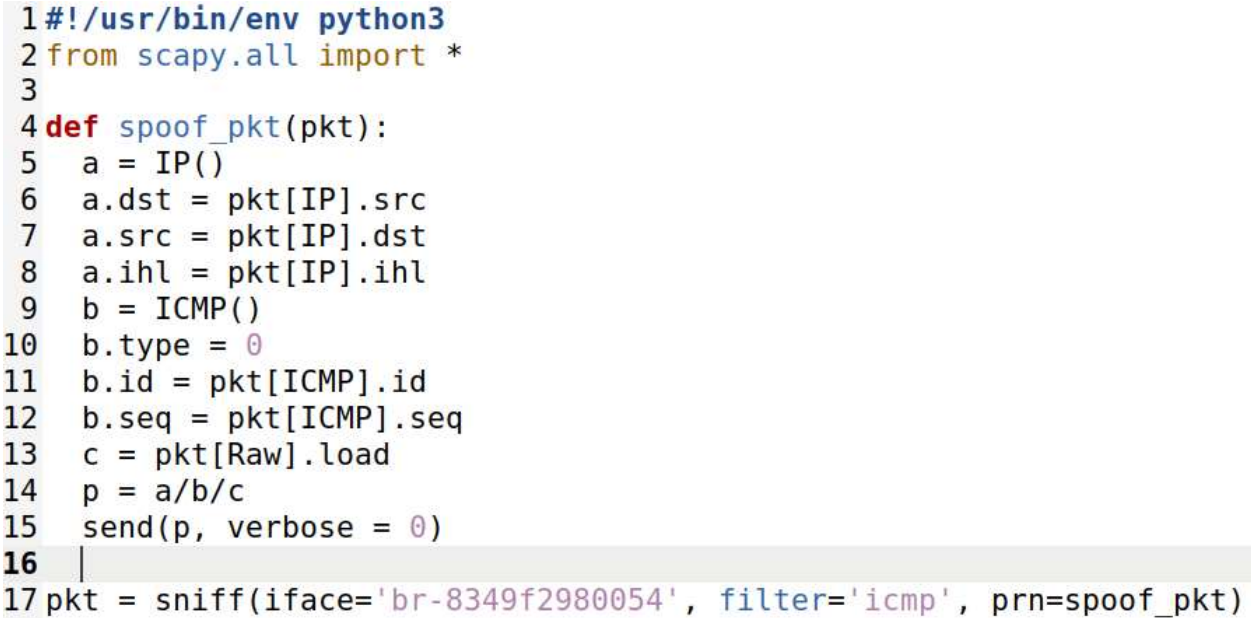
```
root@6610bf846eaf:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=55 time=13.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=55 time=13.7 ms
^C
--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 13.216/13.453/13.691/0.237 ms
```

Screenshot 1.4.6: ping 8.8.8.8, sniffspoofy.py is running



```
root@6610bf846eaf:/# 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=52.5 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=55 time=113 ms (DUP!)
^C
--- 8.8.8.8 ping statistics ---
1 packets transmitted, 1 received, +1 duplicates, 0% packet loss, time
0ms
rtt min/avg/max/mdev = 52.502/82.536/112.570/30.034 ms
```

Screenshot 1.4.7: sniffspoof.py code



```python
#!/usr/bin/env python3
from scapy.all import *

def spoof_pkt(pkt):
  a = IP()
  a.dst = pkt[IP].src
  a.src = pkt[IP].dst
  a.ihl = pkt[IP].ihl
  b = ICMP()
  b.type = 0
  b.id = pkt[ICMP].id
  b.seq = pkt[ICMP].seq
  c = pkt[Raw].load
  p = a/b/c
  send(p, verbose = 0)

pkt = sniff(iface='br-8349f2980054', filter='icmp', prn=spoof_pkt)
```

Task 2.1: Writing Packet Sniffing Program

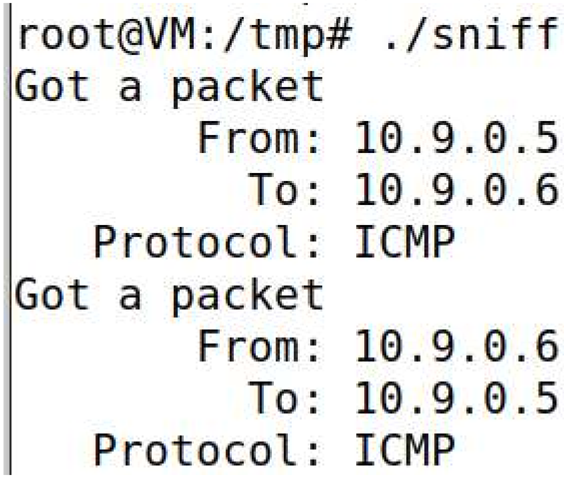
Task 2.1A: Understanding How a Sniffer Works

Question 1: The first important library call is pcap_open_live because it opens a specified device for capturing. The second library call is pcap_compile which helps compile a filter expression. In our case, the filter expression is “ICMP”. The pcap_loop call is where we actually capture and then process each packet. The pcap_close call is used to close the capture device once the program is closed.

Question 2: You need root privilege to run a sniffer program because you need to be able to see all packets. You need root privilege to put the network adapter into promiscuous mode. Without root privilege, the program gets a segmentation fault and fails at pcap_compile.

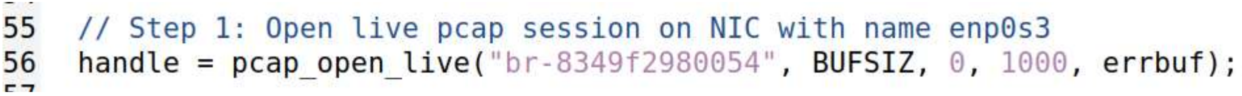
Question 3: When I changed the value of the 3rd parameter of pcap_open_live, I see that promiscuity is 0 when sniff.c is running and I also see that no packets are being sniffed. We can check and see for packets to be successfully sniffed, promiscuity is 1 when I check ip -d link show dev br-8349f2980054.

Screenshot 2.1A.1: Successful packet capture



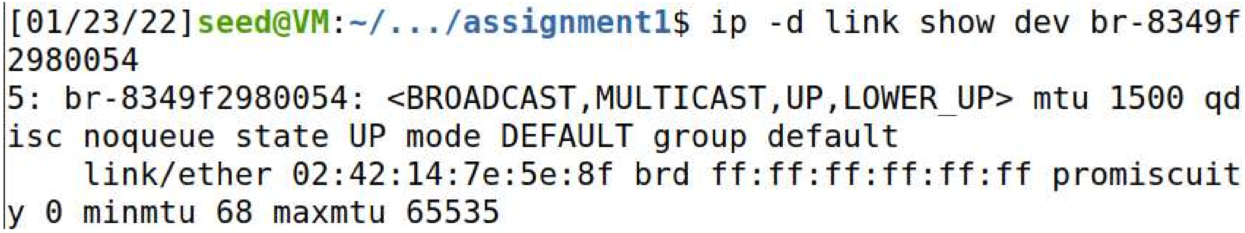
```
root@VM:/tmp# ./sniff
Got a packet
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: ICMP
Got a packet
       From: 10.9.0.6
         To: 10.9.0.5
   Protocol: ICMP
```

Screenshot 2.1A.2: Value of 3rd parameter changed to 0



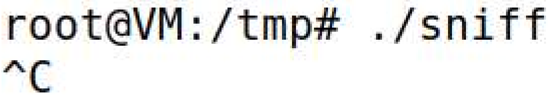
```
  // Step 1: Open live pcap session on NIC with name enp0s3
  handle = pcap_open_live("br-8349f2980054", BUFSIZ, 0, 1000, errbuf);
```

Screenshot 2.1A.3: Promiscuity is 0 (off)



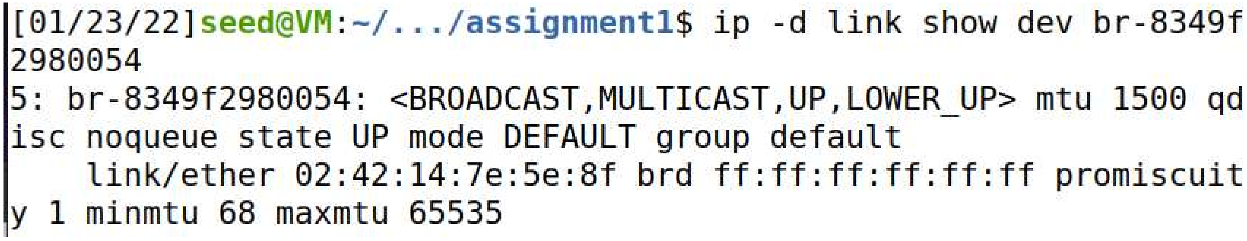
```
[01/23/22]seed@VM:~/.../assignment1$ ip -d link show dev br-8349f
2980054
5: br-8349f2980054: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qd
isc noqueue state UP mode DEFAULT group default
    link/ether 02:42:14:7e:5e:8f brd ff:ff:ff:ff:ff:ff promiscuit
y 0 minmtu 68 maxmtu 65535
```

Screenshot 2.1A.4: Unable to sniff packets



```
root@VM:/tmp# ./sniff
^C
```

Screenshot 2.1A.5: Promiscuity 1 (on) while sniff is running

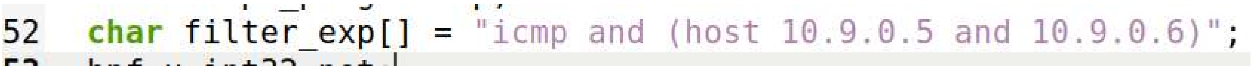


```
[01/23/22]seed@VM:~/.../assignment1$ ip -d link show dev br-8349f
2980054
5: br-8349f2980054: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qd
isc noqueue state UP mode DEFAULT group default
    link/ether 02:42:14:7e:5e:8f brd ff:ff:ff:ff:ff:ff promiscuit
y 1 minmtu 68 maxmtu 65535
```

Task 2.1B: Writing Filters

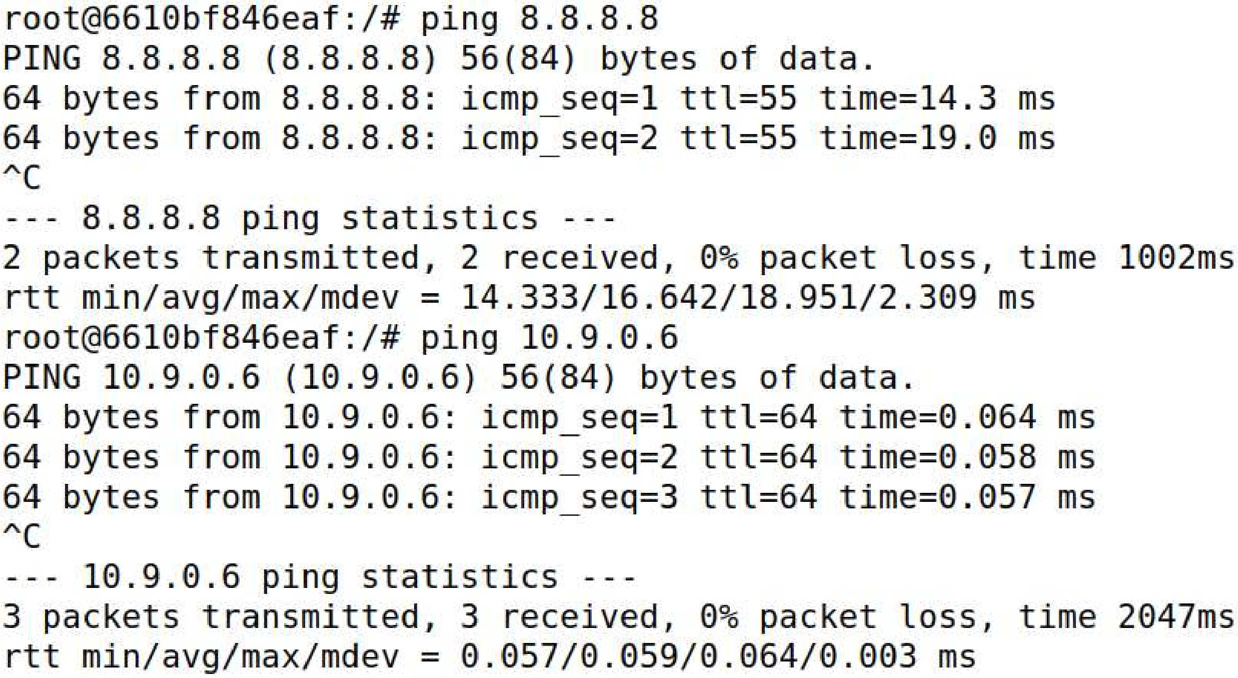
I updated the filter to “icmp and (host 10.9.0.5 and 10.9.0.6)”. Then I used 10.9.0.5 to first ping 8.8.8.8. Nothing was sniffed by the attacker because 8.8.8.8 did not match the filter. I then pinged 10.9.0.6 and then I was able to sniff packets.

Screenshot 2.1B.1: Filter for ICMP between hosts 10.9.0.5 and 10.9.0.6



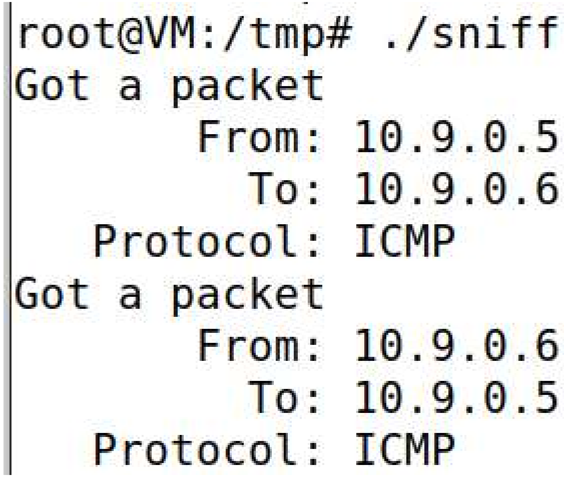
```
char filter_exp[] = "icmp and (host 10.9.0.5 and 10.9.0.6)";
```

Screenshot 2.1B.2: Host 10.9.0.5 first pinged 8.8.8.8 and then pinged 10.9.0.6



```
root@6610bf846eaf:/# 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=55 time=14.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=55 time=19.0 ms
^C
--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 14.333/16.642/18.951/2.309 ms
root@6610bf846eaf:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.064 ms
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.058 ms
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.057 ms
^C
--- 10.9.0.6 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2047ms
rtt min/avg/max/mdev = 0.057/0.059/0.064/0.003 ms
```

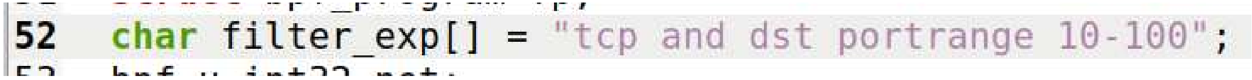
Screenshot 2.1B.3: Attacker only sniffed packets between 10.9.0.5 and 10.9.0.6



```
root@VM:/tmp# ./sniff
Got a packet
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: ICMP
Got a packet
       From: 10.9.0.6
         To: 10.9.0.5
   Protocol: ICMP
```

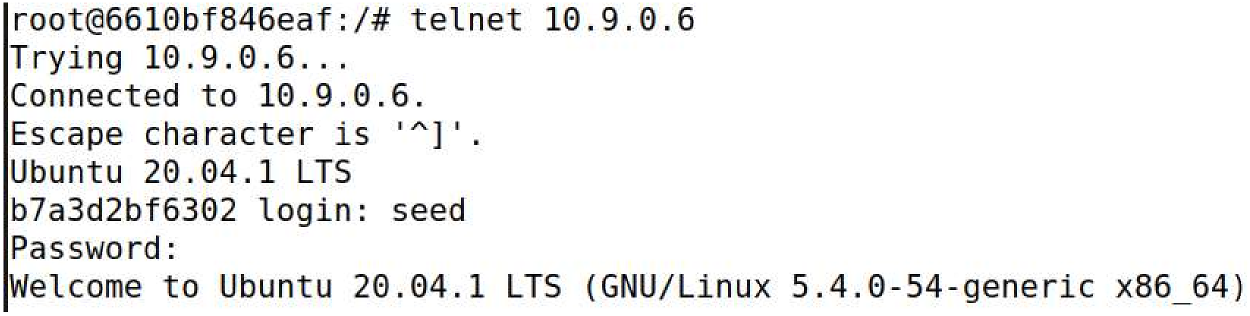
For the second part on capturing TCP packets with destination port number in range 10-100, I updated the filter expression to “tcp and dst portrange 10-100”. Since telnet is within the port range, I sent a telnet request from 10.9.0.5 to 10.9.0.6 and the attacker machine was able to sniff these packets.

Screenshot 2.1B.4: TCP packets with destination port range 10-100



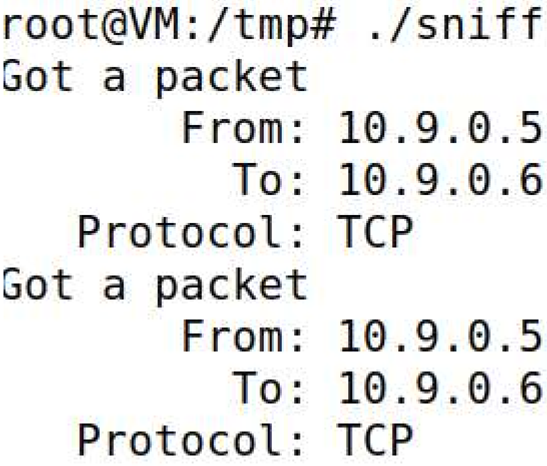
```
char filter_exp[] = "tcp and dst portrange 10-100";
```

Screenshot 2.1B.5: open telnet connection from 10.9.0.5 to 10.9.0.6



```
root@6610bf846eaf:/# telnet 10.9.0.6
Trying 10.9.0.6...
Connected to 10.9.0.6.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
b7a3d2bf6302 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
```

Screenshot 2.1B.6: Attacker machine successful sniffs telnet packets

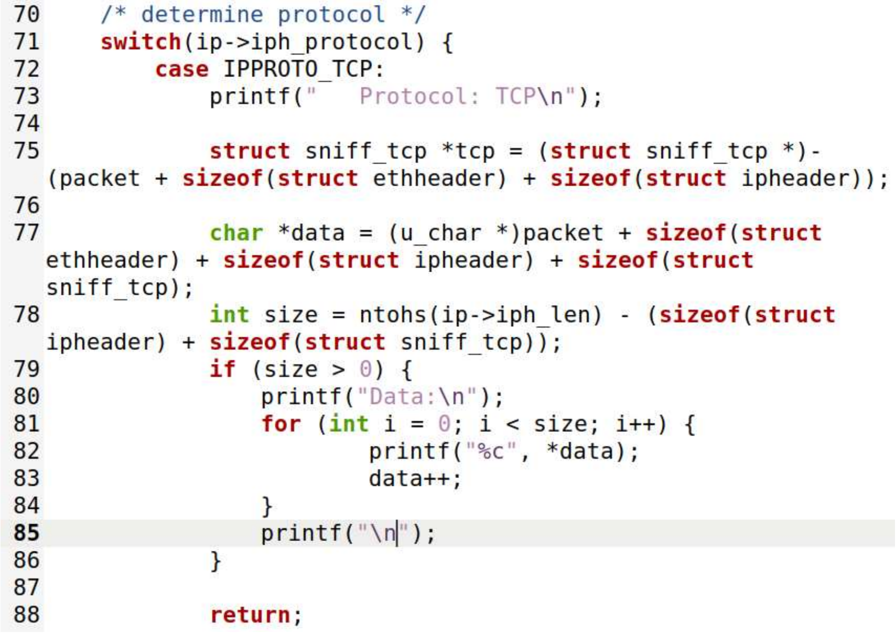


```
root@VM:/tmp# ./sniff
Got a packet
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: TCP
Got a packet
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: TCP
```

Task 2.1C: Sniffing Passwords

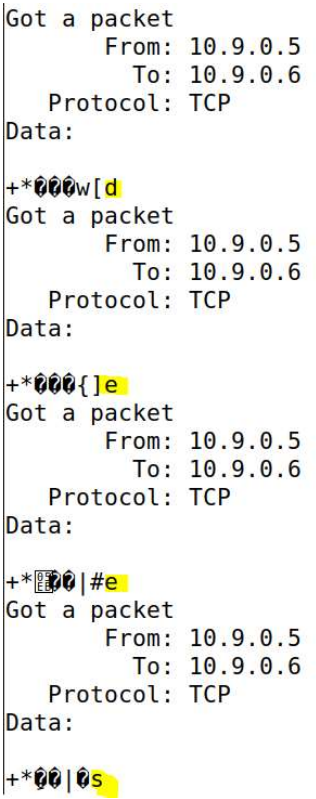
I updated my sniff.c program to also display the data for TCP packets. To extract the data, we need to go past the headers and put a pointer to the start of the data to print it out. After I typed the password “dees” in, I can see that these packets were sniffed.

Screenshot 2.1C.1: Added code to print out data



```
    /* determine protocol */
    switch(ip->iph_protocol) {
        case IPPROTO_TCP:
            printf("   Protocol: TCP\n");

            struct sniff_tcp *tcp = (struct sniff_tcp *)-
(packet + sizeof(struct ethheader) + sizeof(struct ipheader));

            char *data = (u_char *)packet + sizeof(struct
ethheader) + sizeof(struct ipheader) + sizeof(struct
sniff_tcp);
            int size = ntohs(ip->iph_len) - (sizeof(struct
ipheader) + sizeof(struct sniff_tcp));
            if (size > 0) {
                printf("Data:\n");
                for (int i = 0; i < size; i++) {
                        printf("%c", *data);
                        data++;
                }
                printf("\n");
            }

            return;
```

Screenshot 2.1C.2: Captured packets with password “dees”



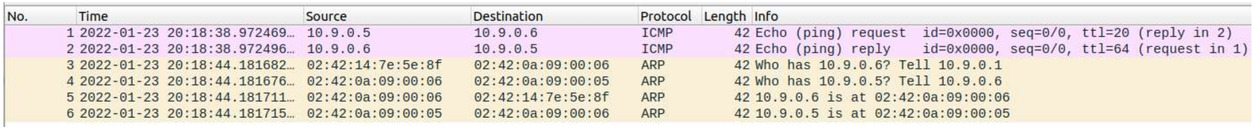
```
Got a packet
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: TCP
Data:

+*���w[d
Got a packet
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: TCP
Data:

+*���{]e
Got a packet
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: TCP
Data:

+*���|#e
Got a packet
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: TCP
Data:

+*��|�s
```

Task 2.2: Spoofing

Task 2.2A: Write a spoofing program

Using the reference code provided, I created spoof.c which spoofs a ICMP request from 10.9.0.5 that gets sent to 10.9.0.6. Through raw socket programming, we are able to build an ICMP request from scratch and put in our desired IP addresses. Using Wireshark, I saw that this was successfully sent to 10.9.0.6 and we got a reply as well.

Screenshot 2.2.1: Wireshark results of spoofed ICMP request from socket programming code

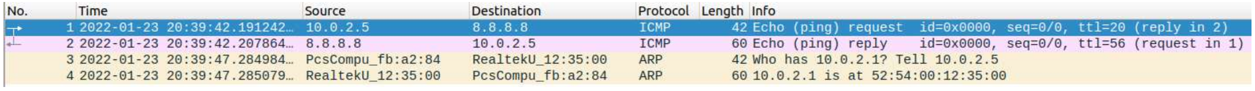


No.	Time	Source	Destination	Protocol	Length	Info
1	2022-01-23 20:18:38.972469…	10.9.0.5	10.9.0.6	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=20 (reply in 2)
2	2022-01-23 20:18:38.972496…	10.9.0.6	10.9.0.5	ICMP	42	Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (request in 1)
3	2022-01-23 20:18:44.181682…	02:42:14:7e:5e:8f	02:42:0a:09:00:06	ARP	42	Who has 10.9.0.6? Tell 10.9.0.1
4	2022-01-23 20:18:44.181676…	02:42:0a:09:00:06	02:42:0a:09:00:05	ARP	42	Who has 10.9.0.5? Tell 10.9.0.6
5	2022-01-23 20:18:44.181711…	02:42:0a:09:00:06	02:42:14:7e:5e:8f	ARP	42	10.9.0.6 is at 02:42:0a:09:00:06
6	2022-01-23 20:18:44.181715…	02:42:0a:09:00:05	02:42:0a:09:00:06	ARP	42	10.9.0.5 is at 02:42:0a:09:00:05

Task 2.2B: Spoof an ICMP Echo Request

I updated the IP addresses to spoof an ICMP request from machine 10.9.0.5 to 8.8.8.8 (an alive machine on the Internet). Wireshark was able to show that I received a reply back.

Screenshot 2.2B.1: Wireshark results of spoofing an ICMP request to a machine on the Internet



No.	Time	Source	Destination	Protocol	Length	Info
1	2022-01-23 20:39:42.191242…	10.0.2.5	8.8.8.8	ICMP	42	Echo (ping) request id=0x0000, seq=0/0, ttl=20 (reply in 2)
2	2022-01-23 20:39:42.207864…	8.8.8.8	10.0.2.5	ICMP	60	Echo (ping) reply id=0x0000, seq=0/0, ttl=56 (request in 1)
3	2022-01-23 20:39:47.284984…	PcsCompu_fb:a2:84	RealtekU_12:35:00	ARP	42	Who has 10.0.2.1? Tell 10.0.2.5
4	2022-01-23 20:39:47.285079…	RealtekU_12:35:00	PcsCompu_fb:a2:84	ARP	60	10.0.2.1 is at 52:54:00:12:35:00

Question 4: No, the IP packet length field cannot be an arbitrary value. If you look at the send_raw_ip_packet function, the sendto function requires the size of the packet that is being sent. If this does not match then there will be errors.

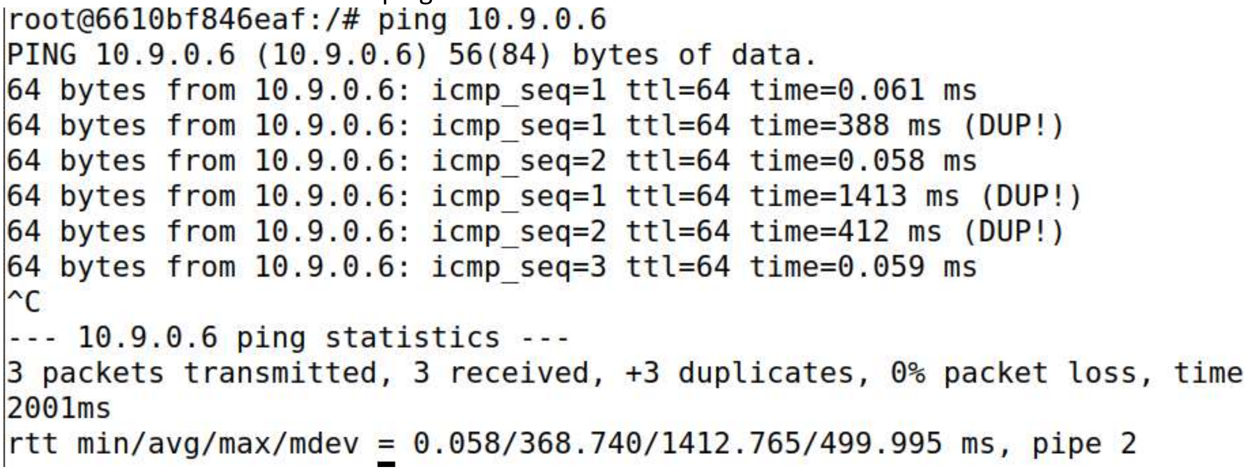
Question 5: No, it doesn't look like we need to calculate the checksum for the IP header. The only checksum we are doing is for the ICMP header.

Question 6: Raw socket programming allows you to get any packets, regardless of ownership. This is why you need root privilege. Also, without root privilege, you are not able to bind a port that is lower than 1024.

Task 2.3: Sniff and then Spoof

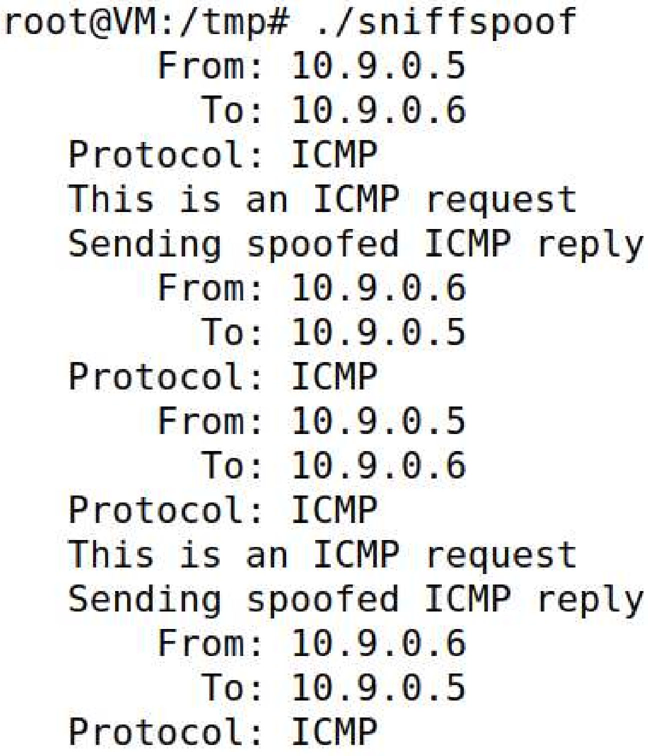
For this task, I used host 10.9.0.5 and 10.9.0.6 and an attacker machine which is all on the same LAN. The attacker machine is running sniffspoof.c. The attacker first sniffs and identify the packets that are ICMP requests. Then, it created a spoofed ICMP reply packet and sends it back to the requestor. Host 10.9.0.5 sends out a ping to 10.9.0.6 and receives both the actual ICMP reply and also the spoofed ICMP reply from the attacker machine. This is because the destination host is still alive and will also reply.

Screenshot 2.3.1: Host 10.9.0.5 pings 10.9.0.6



```
root@6610bf846eaf:/# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.061 ms
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=388 ms (DUP!)
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=0.058 ms
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=1413 ms (DUP!)
64 bytes from 10.9.0.6: icmp_seq=2 ttl=64 time=412 ms (DUP!)
64 bytes from 10.9.0.6: icmp_seq=3 ttl=64 time=0.059 ms
^C
--- 10.9.0.6 ping statistics ---
3 packets transmitted, 3 received, +3 duplicates, 0% packet loss, time
2001ms
rtt min/avg/max/mdev = 0.058/368.740/1412.765/499.995 ms, pipe 2
```

Screenshot 2.3.2: Attacker machine sniffs ICMP packets and spoofs ICMP reply



```
root@VM:/tmp# ./sniffspoof
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: ICMP
   This is an ICMP request
   Sending spoofed ICMP reply
       From: 10.9.0.6
         To: 10.9.0.5
   Protocol: ICMP
       From: 10.9.0.5
         To: 10.9.0.6
   Protocol: ICMP
   This is an ICMP request
   Sending spoofed ICMP reply
       From: 10.9.0.6
         To: 10.9.0.5
   Protocol: ICMP
```