

# **CNS LAB**

## **ASSIGNMENT -2**

### **Writing Programs to Sniff and Spoof Packets using pcap (C programs)**

---

**NAME :SIRI S**

**SEMESTER :5**

**SECTION :H**

**SRN :PES1UG19CS485**

---

### **Screenshots of Execution:**

ATTACKER: 10. 0. 2. 4

VICTIM: 10. 0. 2. 5

### **Task1: Sniffing-Writing Packet Sniffing Program**

When we run the ‘Actual Sniffer’ code from [www.tcpdump.org](http://www.tcpdump.org), we get the following

```
[09/19/21]seed@SiriS_PES1UG19CS485:~/bin/python$ sudo ./actualsniff
Jacked a packet with length of [-1216466944]
[09/19/21]seed@SiriS_PES1UG19CS485:~/bin/python$
```

When we run the sniffer.c code from the attacker vm, the following packets are captured.

```
SEEDUbuntu [Running]
Terminal [09/19/21]seed@SiriS_PES1UG19CS485:~/.../code$ sudo ./sniffex
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: enp0s3
Number of packets: 40
Filter expression: proto TCP and dst portrange 10-100

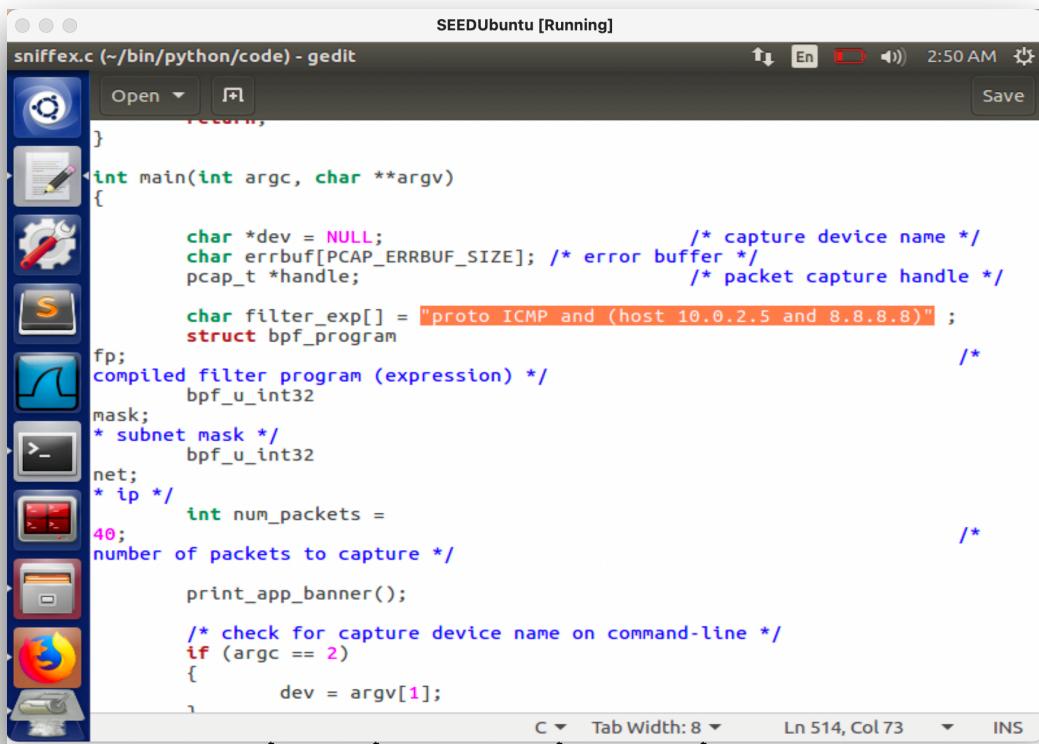
Packet number 1:
    From: 10.0.2.4
    To: 117.18.237.29
    Protocol: TCP
    Src port: 45676
    Dst port: 80

Packet number 2:
    From: 10.0.2.4
    To: 117.18.237.29
    Protocol: TCP
    Src port: 45678
```

```
SEEDUbuntu [Running]
Terminal
Packet number 4:
    From: 10.0.2.4
    To: 117.18.237.29
    Protocol: TCP
    Src port: 45678
    Dst port: 80
    Payload (437 bytes):
00000  50 4f 53 54 20 2f 20 48  54 54 50 2f 31 2e 31 0
d  POST / HTTP/1.1.
00016  0a 48 6f 73 74 3a 20 6f  63 73 70 2e 64 69 67 6
9  .Host: ocsp.digi
00032  63 65 72 74 2e 63 6f 6d  0d 0a 55 73 65 72 2d 4
1  cert.com..User-A
00048  67 65 6e 74 3a 20 4d 6f  7a 69 6c 6c 61 2f 35 2
e  gent: Mozilla/5.
00064  30 20 28 58 31 31 3b 20  55 62 75 6e 74 75 3b 2
0  0 (X11; Ubuntu;
00080  4c 69 6e 75 78 20 69 36  38 36 3b 20 72 76 3a 3
6  Linux i686; rv:6
00096  30 2e 30 29 20 47 65 63  6b 6f 2f 32 30 31 30 3
0  0.0) Gecko/20100
```

# Task 1.1: Understanding how a Sniffer Works

Upon changing the sniffex.c code that is written using the pcap API and using the victim VM (10.0.2.5) to attack google (8.8.8.8)

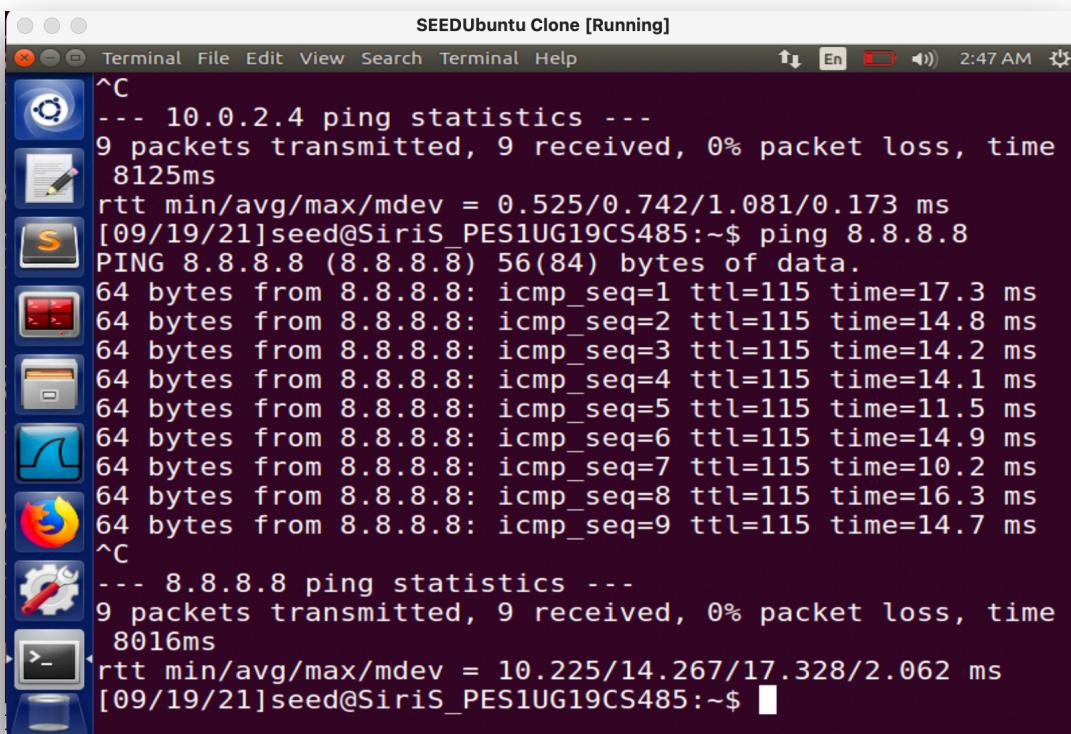


The screenshot shows a terminal window titled "SEEDUbuntu [Running]". The window contains a code editor with the file "sniffex.c" open. The code is written in C and uses the pcap API to capture ICMP packets from host 10.0.2.5 to 8.8.8.8. The code includes comments explaining the purpose of various variables and sections. The terminal window has a dark theme with icons on the left.

```
SEEDUbuntu [Running]
sniffex.c (~bin/python/code) - gedit
Open ⌄ ⌂ Save
}
int main(int argc, char **argv)
{
    char *dev = NULL; /* capture device name */
    char errbuf[PCAP_ERRBUF_SIZE]; /* error buffer */
    pcap_t *handle; /* packet capture handle */

    char filter_exp[] = "proto ICMP and (host 10.0.2.5 and 8.8.8.8)" ;
    struct bpf_program
fp; /* compiled filter program (expression) */
    bpf_u_int32
mask;
    * subnet mask */
    bpf_u_int32
net;
    * ip */
    int num_packets =
40; /* number of packets to capture */
    print_app_banner();
    /* check for capture device name on command-line */
    if (argc == 2)
    {
        dev = argv[1];
    }
    /*
Ln 514, Col 73
C Tab Width: 8 ▾ INS
```

Changing the code



The screenshot shows a terminal window titled "SEEDUbuntu Clone [Running]". The window displays the output of a ping command from the victim VM (10.0.2.4) to Google's IP address (8.8.8.8). The output shows 9 packets transmitted, 9 received, 0% packet loss, and a round-trip time (RTT) ranging from 10.2 ms to 17.3 ms. After the ping to Google, the user performs a Ctrl-C (^C) to stop the process. The terminal then shows a ping command to Google again, resulting in similar statistics. The terminal window has a dark theme with icons on the left.

```
SEEDUbuntu Clone [Running]
^C
--- 10.0.2.4 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time
8125ms
rtt min/avg/max/mdev = 0.525/0.742/1.081/0.173 ms
[09/19/21]seed@SiriS_PES1UG19CS485:~$ 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=17.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=14.8 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=14.2 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=115 time=14.1 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=115 time=11.5 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=115 time=14.9 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=115 time=10.2 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=115 time=16.3 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=115 time=14.7 ms
^C
--- 8.8.8.8 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time
8016ms
rtt min/avg/max/mdev = 10.225/14.267/17.328/2.062 ms
[09/19/21]seed@SiriS_PES1UG19CS485:~$ █
```

Pinging Google (destination address) from victim VM

The following icmp packets are captured in the attacker VM using pcap.

```
SEEDUbuntu [Running]
Terminal
^C
[09/19/21]seed@SiriS_PES1UG19CS485:~/.../code$ sudo ./sniffex
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: enp0s3
Number of packets: 40
Filter expression: proto ICMP and (host 10.0.2.5 and 8.8.8.8)

Packet number 1:
    From: 10.0.2.5
    To: 8.8.8.8
    Protocol: ICMP

Packet number 2:
    From: 8.8.8.8
    To: 10.0.2.5
    Protocol: ICMP
```

```
SEEDUbuntu [Running]
Terminal
Packet number 15:
    From: 10.0.2.5
    To: 8.8.8.8
    Protocol: ICMP

Packet number 16:
    From: 8.8.8.8
    To: 10.0.2.5
    Protocol: ICMP

Packet number 17:
    From: 10.0.2.5
    To: 8.8.8.8
    Protocol: ICMP

Packet number 18:
    From: 8.8.8.8
    To: 10.0.2.5
    Protocol: ICMP
^[[c^C
[09/19/21]seed@SiriS_PES1UG19CS485:~/.../code$ █
```

**Problem 1: Please use your own words to describe the sequence of the library calls that are essential for sniffer programs. This is meant to be a summary, not a detailed explanation like the one in the tutorial.**

**1. Setting up device**

It can be defined using a string in the code for sniffer or sniffer decides the interface itself by picking up an active interface automatically.

**2. Initialize Sniffing**

Here in this step, after setting up a device for sniffing , sniffer Initialize PCAP and tell it to sniff on a particular device to create an environment for sniffing called as a session.

**3. Traffic Filtering**

For every session of sniffing you create you have to define any desire or rule, upon which packets are to be sniffed and analyzed. For example if you want to sniff HTTP traffic on a specific Interface of your computer , you will sniff TCP port 80 traffic on that interface (since http traffic uses port80), you will write your requirement in filter string and then compile it to apply the rule. This is called filtering and it is possible to use a blank filter but in that case it will be sniffing all packets and will be analyzing all fields.

**4. Execution /actual sniffing**

This is the execution part where sniffer is finally executed. Here we see that there are actually two main techniques to capture a packet, first is a packet is sniffed and then analyzed instantly whereas other is in which we enter into a loop that waits. for n number of packets to be sniffed before we go for analyzing part. It stores the results as asked by the user that is either to display the packets immediately or to save them in a file for future record.

**5. Ending Session**

End of the Session when you are done with the requirements of sniffing or you have enough data to analyze and to make any decisions or to make any perception about the network based on the analysis from the packets.

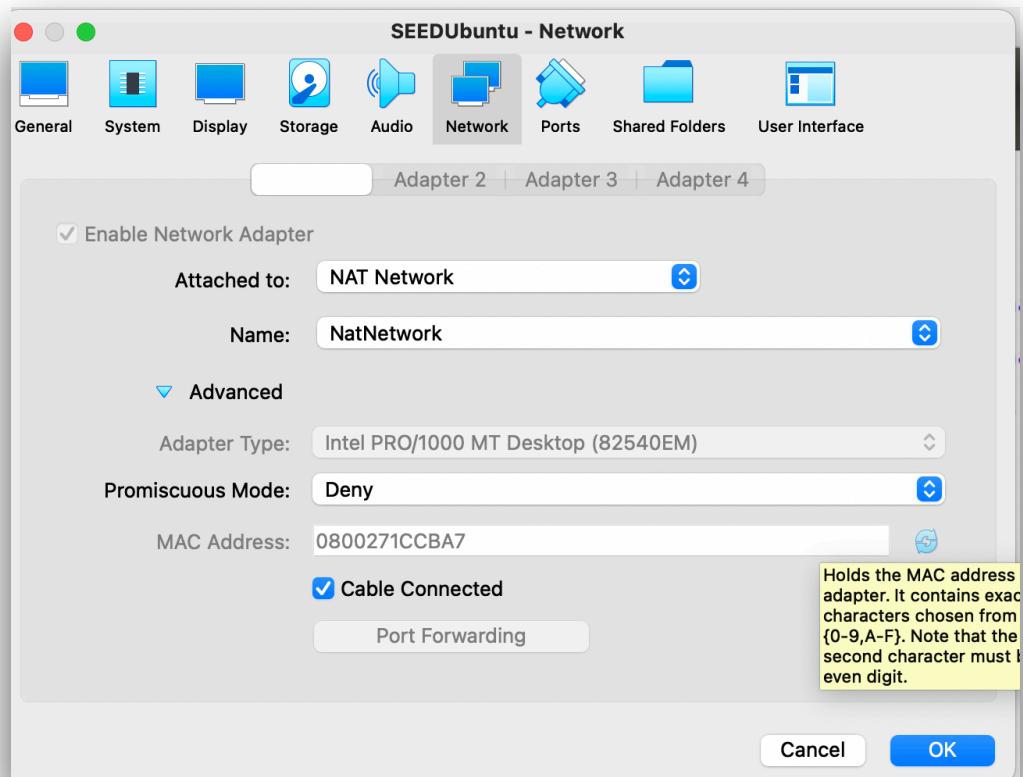
## **Problem 2: Why do you need the root privilege to run sniffex? Where does the program fail if executed without the root privilege?**

Pcap\_lookupdev() function needs root access because it wants to access network interfaces and it is impossible without root access in linux. Sniffer programs need raw sockets that allow direct sending of packets by the applications bypassing all applications in network software of operating system. And we need to be a root to create raw socket as we can't discover NIC until we are root.

## **Problem 3: Please turn on and turn off the promiscuous mode in the sniffer program. Can you demonstrate the difference when this mode is on and off? Please describe how you demonstrate this**

In really simple words promiscuous mode is one in which all the packets are sent to a computer or sniffed by sniffer and not only those which are addressed to it whereas in a non promiscuous mode only those packets are send to the computer or sniffed by sniffer which are addressed to it.

Screenshots:



Switching off the promiscuous Mode by enabling 'Deny' in network settings

SEEDUbuntu Clone [Running]

```
[09/19/21]seed@SiriS_PES1UG19CS485:~$ 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=8.19 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=8.41 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=8.37 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=115 time=9.54 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=115 time=9.22 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=115 time=14.6 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=115 time=7.89 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=115 time=15.2 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=115 time=8.05 ms
64 bytes from 8.8.8.8: icmp_seq=10 ttl=115 time=10.7 ms
64 bytes from 8.8.8.8: icmp_seq=11 ttl=115 time=7.97 ms
64 bytes from 8.8.8.8: icmp_seq=12 ttl=115 time=34.9 ms
64 bytes from 8.8.8.8: icmp_seq=13 ttl=115 time=8.25 ms
64 bytes from 8.8.8.8: icmp_seq=14 ttl=115 time=8.21 ms
64 bytes from 8.8.8.8: icmp_seq=15 ttl=115 time=16.8 ms
64 bytes from 8.8.8.8: icmp_seq=16 ttl=115 time=8.08 ms
^C
--- 8.8.8.8 ping statistics ---
16 packets transmitted, 16 received, 0% packet loss, ti
```

Pinging 8.8.8.8 from victim VM

SEEDUbuntu [Running]

```
From: 10.0.2.5
To: 8.8.8.8
Protocol: ICMP

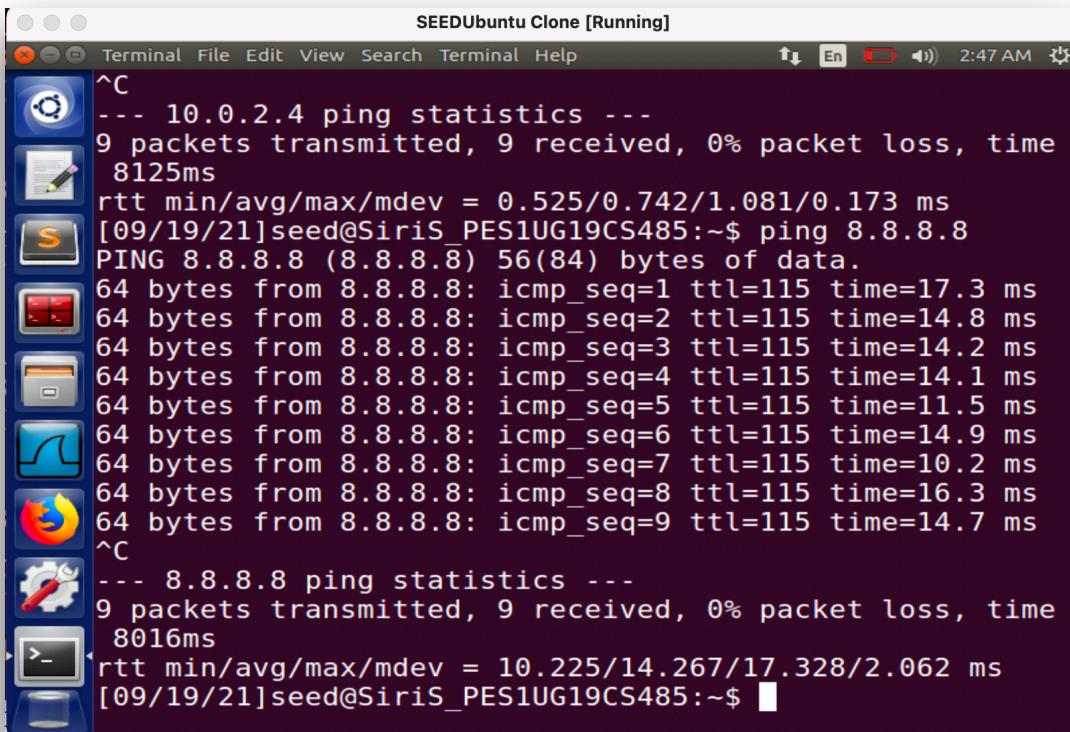
Packet number 18:
From: 8.8.8.8
To: 10.0.2.5
Protocol: ICMP
^[[c^C
[09/19/21]seed@SiriS_PES1UG19CS485:~/.../code$ gcc -o sniffex sniffex.c -lpcap
[09/19/21]seed@SiriS_PES1UG19CS485:~/.../code$ sudo ./sniffex
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: enp0s3
Number of packets: 40
Filter expression: proto ICMP and (host 10.0.2.5 and 8.8.8.8)
```

No capture of packets because promiscuous mode is off

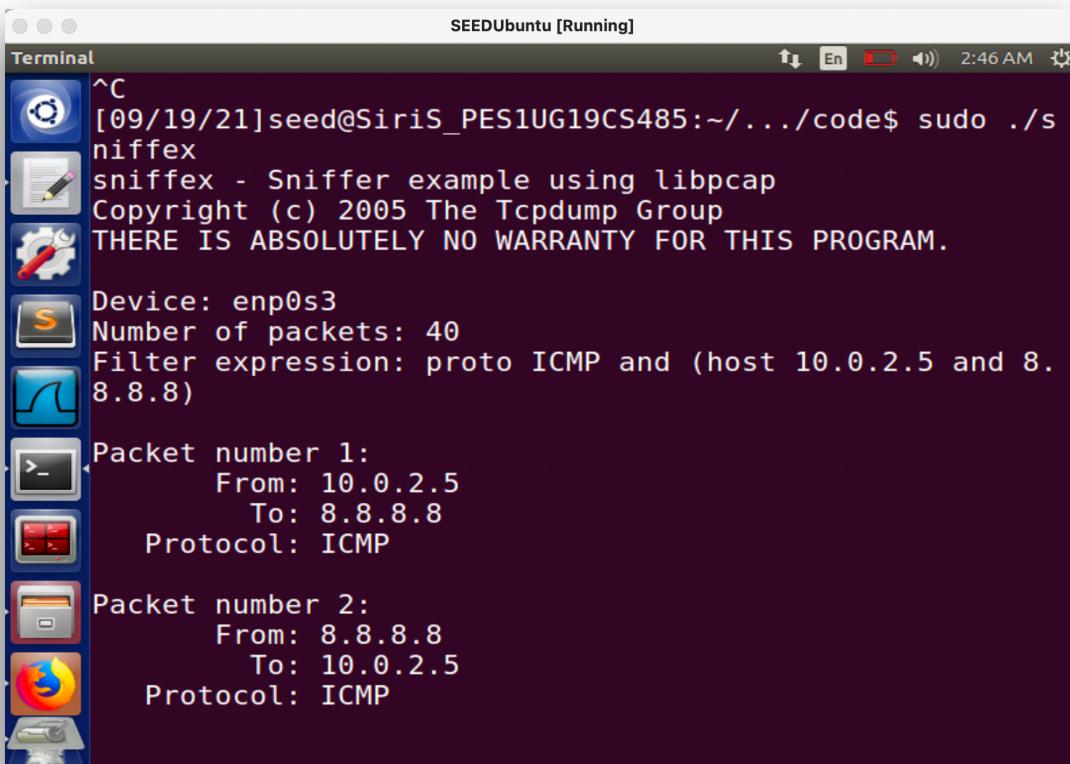
# Task 1.2: Writing Filters

## i) Capture the ICMP packets between two specific hosts



```
SEEDUbuntu Clone [Running]
Terminal File Edit View Search Terminal Help 2:47 AM
^C
--- 10.0.2.4 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time
8125ms
rtt min/avg/max/mdev = 0.525/0.742/1.081/0.173 ms
[09/19/21]seed@SiriS_PES1UG19CS485:~$ 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=17.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=14.8 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=115 time=14.2 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=115 time=14.1 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=115 time=11.5 ms
64 bytes from 8.8.8.8: icmp_seq=6 ttl=115 time=14.9 ms
64 bytes from 8.8.8.8: icmp_seq=7 ttl=115 time=10.2 ms
64 bytes from 8.8.8.8: icmp_seq=8 ttl=115 time=16.3 ms
64 bytes from 8.8.8.8: icmp_seq=9 ttl=115 time=14.7 ms
^C
--- 8.8.8.8 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time
8016ms
rtt min/avg/max/mdev = 10.225/14.267/17.328/2.062 ms
[09/19/21]seed@SiriS_PES1UG19CS485:~$
```

Pinging Google (destination address) from victim VM



```
SEEDUbuntu [Running]
Terminal 2:46 AM
^C
[09/19/21]seed@SiriS_PES1UG19CS485:~/.../code$ sudo ./sniffex
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

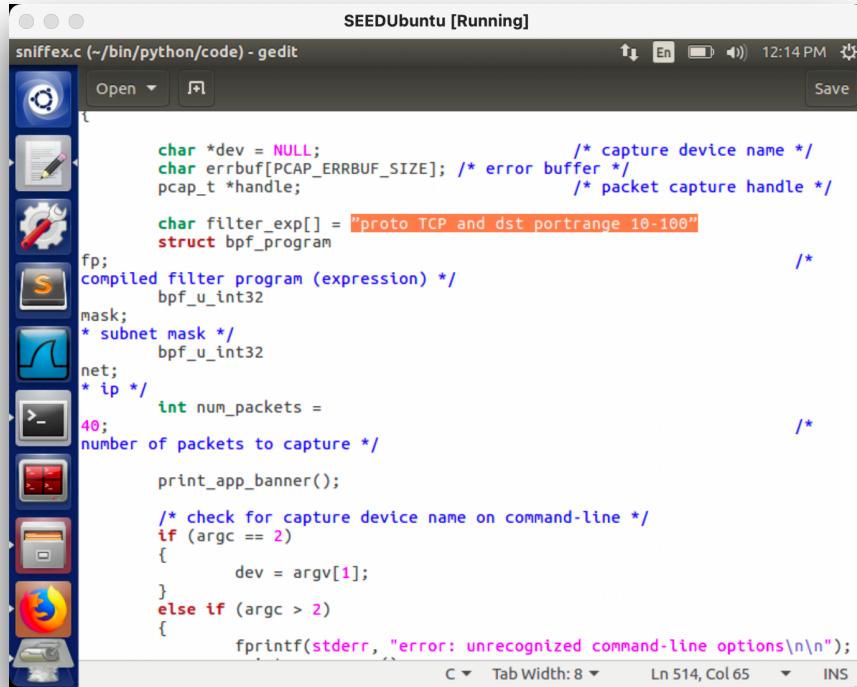
Device: enp0s3
Number of packets: 40
Filter expression: proto ICMP and (host 10.0.2.5 and 8.8.8.8)

Packet number 1:
    From: 10.0.2.5
        To: 8.8.8.8
    Protocol: ICMP

Packet number 2:
    From: 8.8.8.8
        To: 10.0.2.5
    Protocol: ICMP
```

ICMP packets captured by the attacker

**ii) Capture the TCP packets that have a destination port range from 10 - 100.**



```
SEEDUbuntu [Running]
sniffex.c (~-/bin/python/code) - gedit
Open Save
char *dev = NULL; /* capture device name */
char errbuf[PCAP_ERRBUF_SIZE]; /* error buffer */
pcap_t *handle; /* packet capture handle */

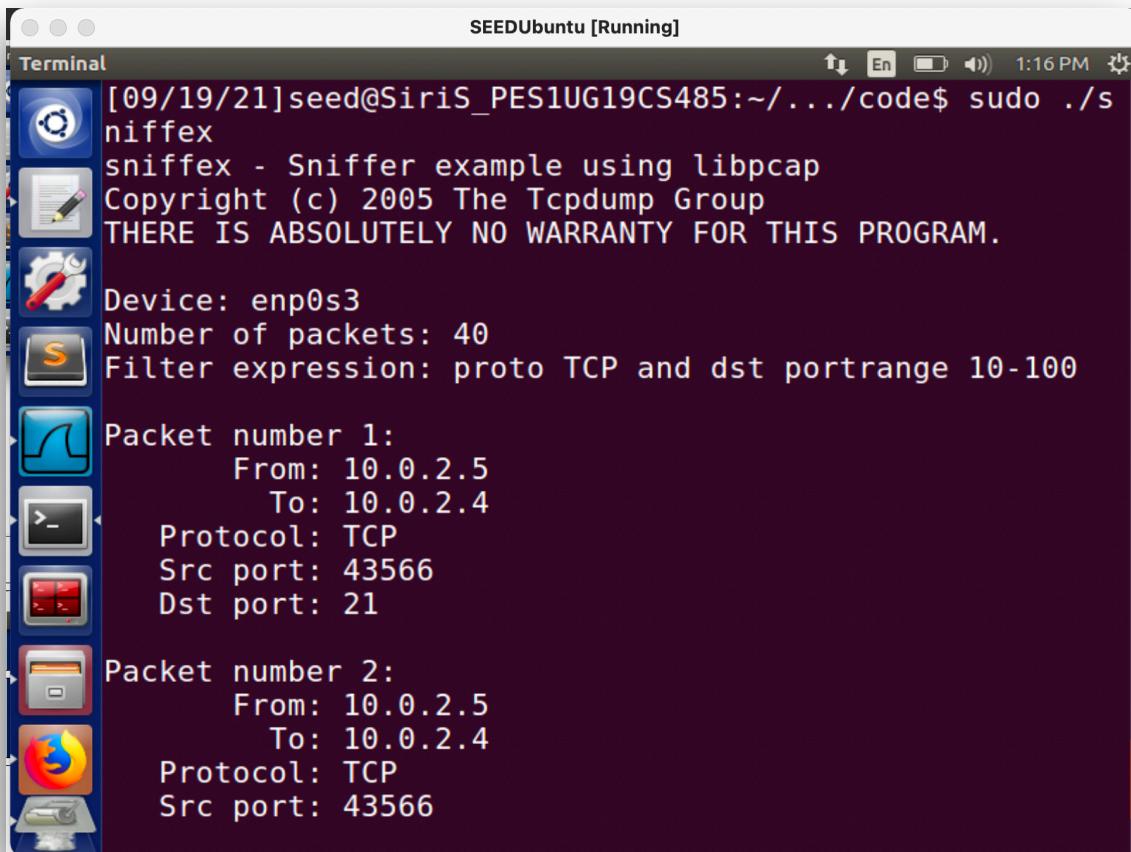
char filter_exp[] = "proto TCP and dst portrange 10-100"
struct bpf_program
fp;
compiled filter program (expression)
bpf_u_int32
mask;
* subnet mask */
bpf_u_int32
net;
* ip */
int num_packets =
40;
number of packets to capture */

print_app_banner();

/* check for capture device name on command-line */
if (argc == 2)
{
    dev = argv[1];
}
else if (argc > 2)
{
    fprintf(stderr, "error: unrecognized command-line options\n\n");
}

C Tab Width: 8 Ln 514, Col 65 INS
```

After changing the sniffed.c code



```
SEEDUbuntu [Running]
Terminal
[09/19/21]seed@SiriS_PES1UG19CS485:~/.../code$ sudo ./sniffex
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: enp0s3
Number of packets: 40
Filter expression: proto TCP and dst portrange 10-100

Packet number 1:
    From: 10.0.2.5
        To: 10.0.2.4
    Protocol: TCP
    Src port: 43566
    Dst port: 21

Packet number 2:
    From: 10.0.2.5
        To: 10.0.2.4
    Protocol: TCP
    Src port: 43566
```

Running the code to capture packets

SEEDUbuntu Clone [Running]

```
[09/19/21]seed@SiriS_PES1UG19CS485:~$ ftp 10.0.2.4
Connected to 10.0.2.4.
220 (vsFTPd 3.0.3)
Name (10.0.2.4:seed): ^C[09/19/21]seed@SiriS_PES1UG19CS
485:~$
```

Sending ftp packets to the destination machine

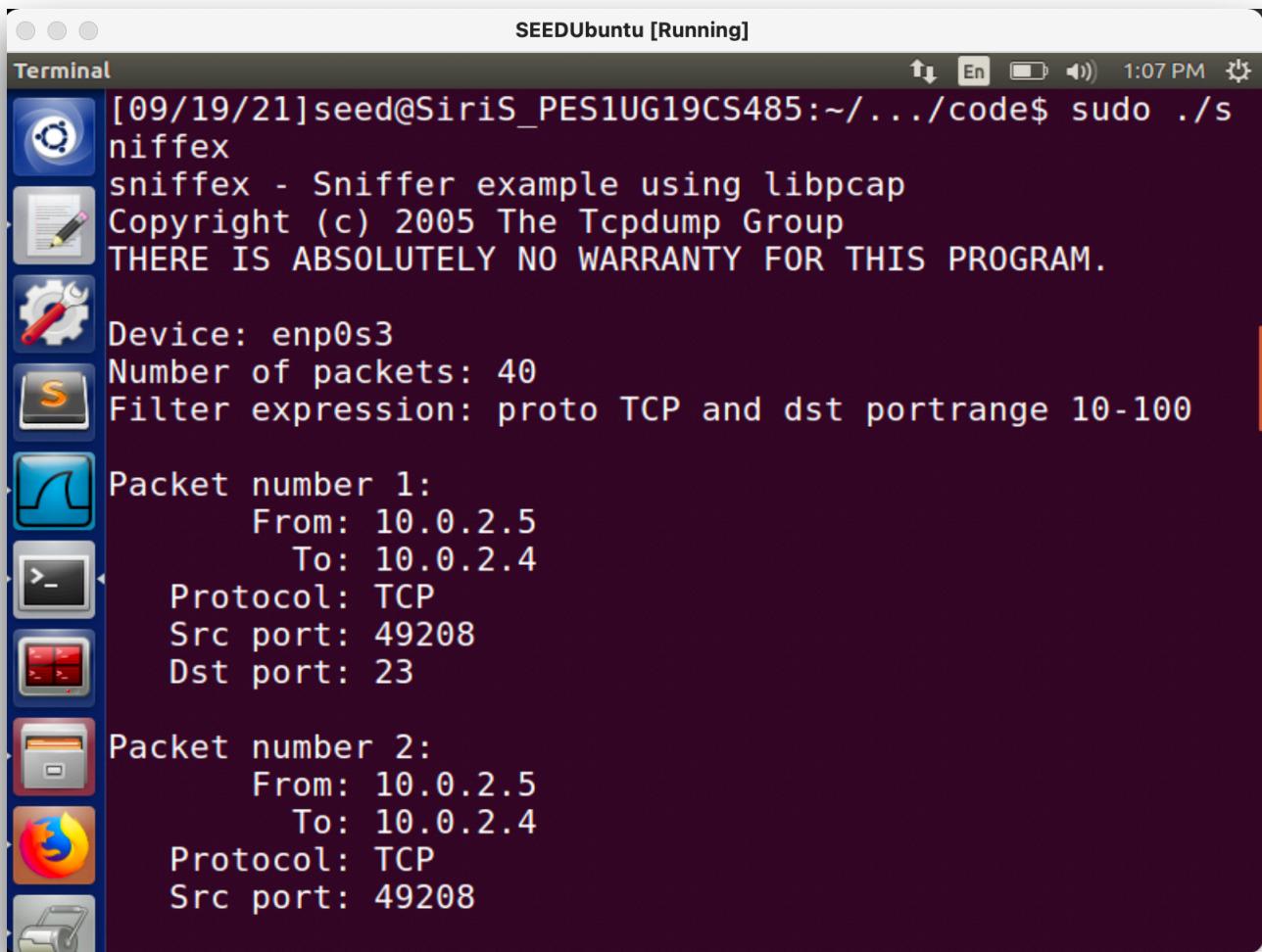
SEEDUbuntu [Running]

Source	Destination	Protocol	Length	Info
10.0.2.4	52.26.14.49	TCP	56	37966 → 443 [ACK] Seq=
52.26.14.49	10.0.2.4	TLSv1.2	123	Application Data
10.0.2.4	52.26.14.49	TCP	56	37974 → 443 [ACK] Seq=
10.0.2.5	10.0.2.4	TCP	76	43566 → 21 [SYN] Seq=
10.0.2.4	10.0.2.5	TCP	76	21 → 43566 [SYN, ACK]
10.0.2.5	10.0.2.4	TCP	68	43566 → 21 [ACK] Seq=
10.0.2.4	10.0.2.5	FTP	88	Response: 220 (vsFTPd)
10.0.2.5	10.0.2.4	TCP	68	43566 → 21 [ACK] Seq=
52.26.14.49	10.0.2.4	TLSv1.2	123	Application Data
10.0.2.4	52.26.14.49	TCP	56	37966 → 443 [ACK] Seq=
52.26.14.49	10.0.2.4	TLSv1.2	123	Application Data
10.0.2.4	52.26.14.49	TCP	56	37974 → 443 [ACK] Seq=
52.26.14.49	10.0.2.4	TLSv1.2	123	Application Data
10.0.2.4	52.26.14.49	TCP	56	37966 → 443 [ACK] Seq=
52.26.14.49	10.0.2.4	TLSv1.2	123	Application Data
10.0.2.4	52.26.14.49	TCP	56	37974 → 443 [ACK] Seq=

Frame 20: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface  
 ▶ Linux cooked capture  
 ▶ Internet Protocol Version 4, Src: 10.0.2.4, Dst: 10.0.2.5  
 ▶ Transmission Control Protocol, Src Port: 21, Dst Port: 43566, Seq: 1576090049,  
 Source Port: 21  
 Destination Port: 43566  
 [Stream index: 2]

Wireshark screenshot of the packets captured along with source (21) and destination port address

# Task 1.3: Sniffing Passwords



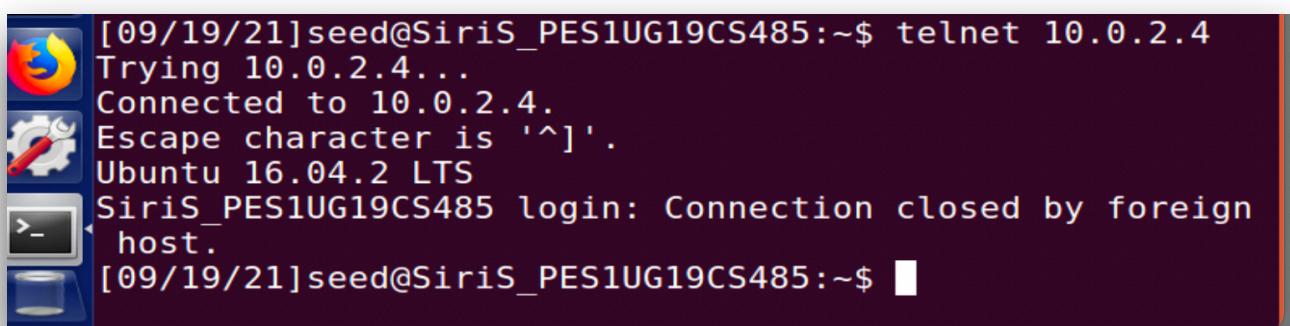
```
[09/19/21]seed@SiriS_PES1UG19CS485:~/.../code$ sudo ./sniffex
sniffex - Sniffer example using libpcap
Copyright (c) 2005 The Tcpdump Group
THERE IS ABSOLUTELY NO WARRANTY FOR THIS PROGRAM.

Device: enp0s3
Number of packets: 40
Filter expression: proto TCP and dst portrange 10-100

Packet number 1:
    From: 10.0.2.5
        To: 10.0.2.4
    Protocol: TCP
    Src port: 49208
    Dst port: 23

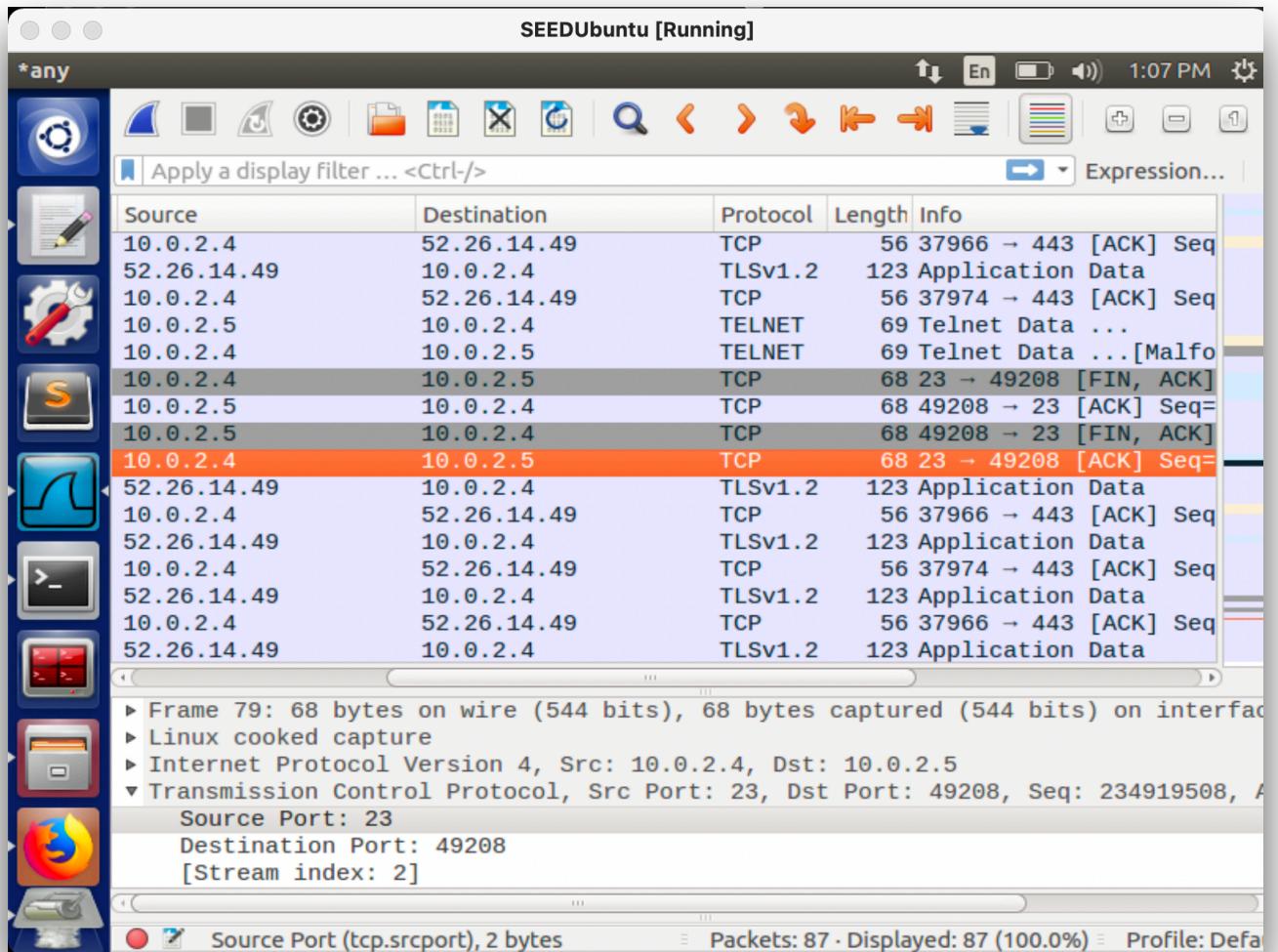
Packet number 2:
    From: 10.0.2.5
        To: 10.0.2.4
    Protocol: TCP
    Src port: 49208
```

Running the code which has a port range that includes port 23



```
[09/19/21]seed@SiriS_PES1UG19CS485:~$ telnet 10.0.2.4
Trying 10.0.2.4...
Connected to 10.0.2.4.
Escape character is '^].
Ubuntu 16.04.2 LTS
SiriS_PES1UG19CS485 login: Connection closed by foreign host.
[09/19/21]seed@SiriS_PES1UG19CS485:~$ █
```

Running telnet on the victim machine using the attacker machines ip address



Wireshark screenshot showing the TCP packets captured in port 23.  
We will connect to a telnet server (running in our VM) and get the password of the user.

# Task 2: Spoofing

## Task 2.1 - A Writing a spoofing program:

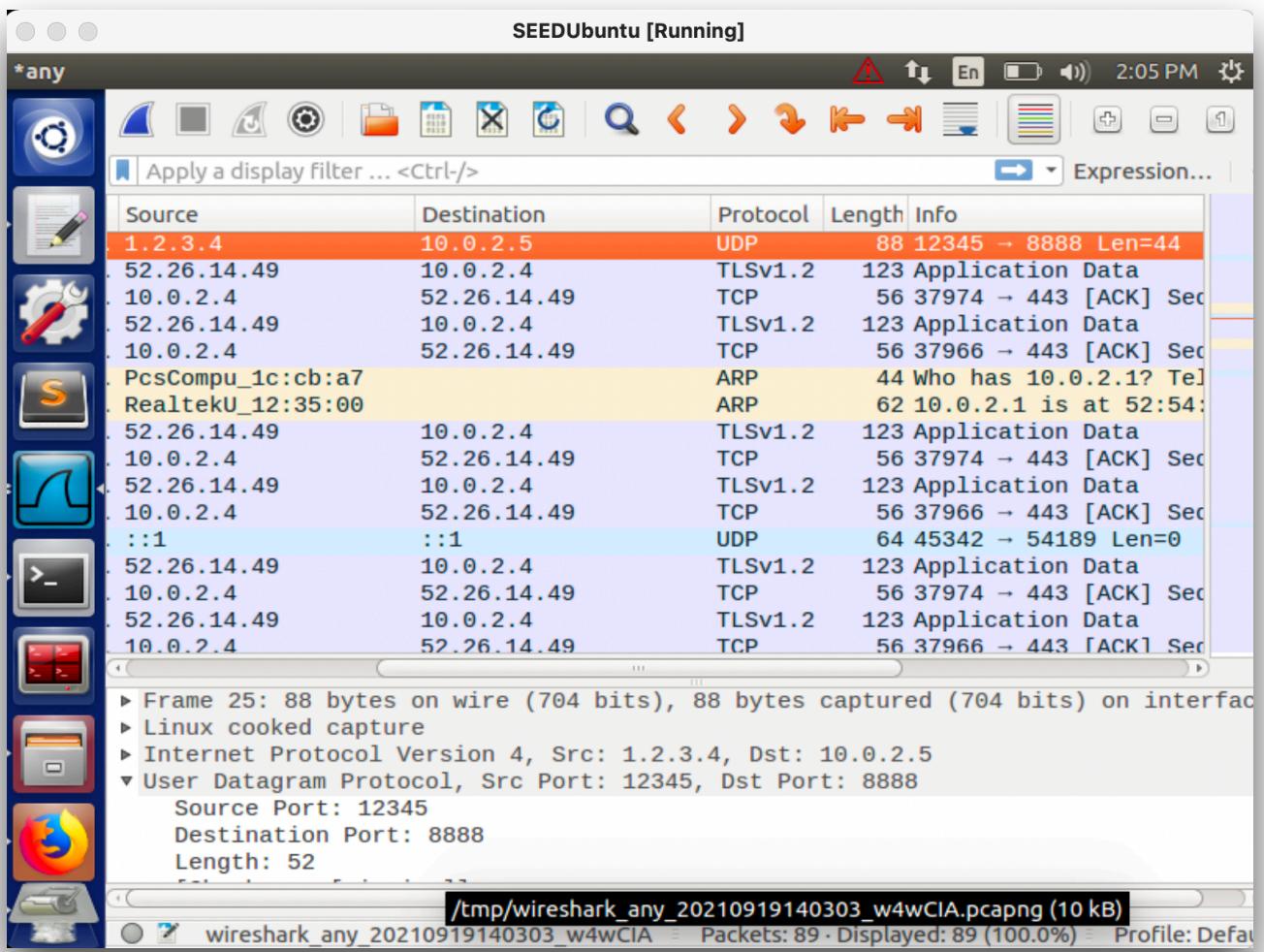
Upon running spoof\_udp.c after changing the victim ip address

```
[09/19/21]seed@SiriS_PES1UG19CS485:~/.../code$ gcc -o spoof_udp spoof_udp.c -lpcap
[09/19/21]seed@SiriS_PES1UG19CS485:~/.../code$ sudo ./spoof_udp
[09/19/21]seed@SiriS_PES1UG19CS485:~/.../code$ sudo ./spoof_udp
[09/19/21]seed@SiriS_PES1UG19CS485:~/.../code$
```

Running the code

```
[09/19/21]seed@SiriS_PES1UG19CS485:~$ nc -luv 8888
Listening on [0.0.0.0] (family 0, port 8888)
Hello Server! This is Siri, the Attacker !
```

Victim machine opens a listener at the user end to listen to the connections and we see that the user machine receives the UDP packet sent by the spoof program.

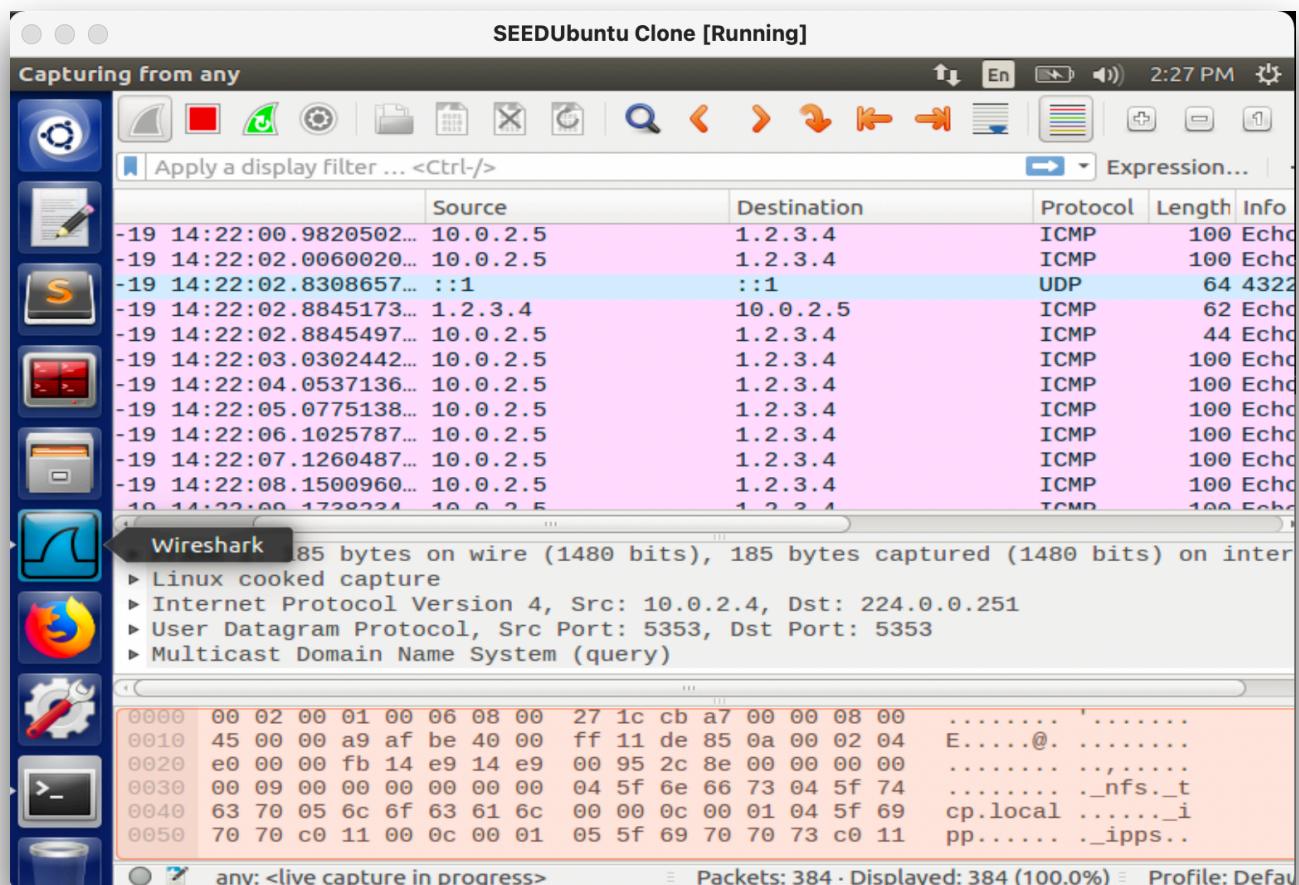


Wireshark screenshot to show the data that's captured due to spoofing

## Task 2.2 – Spoof an ICMP Echo Request

```
[09/19/21]seed@SiriS_PES1UG19CS485:~/.../code$ gcc -o s  
poof_icmp spoof_icmp.c -lpcap  
[09/19/21]seed@SiriS_PES1UG19CS485:~/.../code$ sudo ./s  
poof_icmp  
[09/19/21]seed@SiriS_PES1UG19CS485:~/.../code$ sudo ./s  
poof_icmp  
[09/19/21]seed@SiriS_PES1UG19CS485:~/.../code$ █
```

## Running the code for spoof\_icmp



## Wireshark capturing data