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

(Name: Tilak Vignesh SRN: PES2UG19CS432)

Table of Contents

Task 1: Sniffing - Writing Packet Sniffing Program

The objective of this lab is to understand the sniffing program which uses the pcap library. Withpcap, the task of sniffers becomes invoking a simple sequence of procedures in the pcap library. Understanding sniffex. Please download the sniffex.c program from the tutorial mentioned above, compile and run it. You should provide screen dump evidence to show that your program runs successfully and produces expected results

Attacker machine:	10.0.2.8
Victim machine:	10.0.2.6
Server machine:	10.0.2.9

CODE :

The main fragments of the code are the main function and the got_packet function.

The below screenshots only show the main parts of the code required for sniffing and not the other helper functions and structure definitions.

*Untitled - Notepad

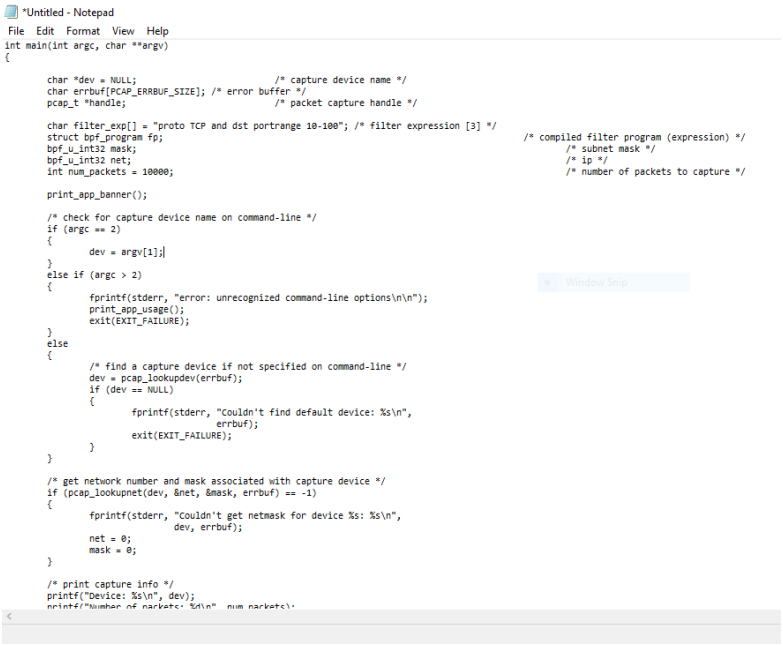
File Edit Format View Help

```
void got_packet(u_char *args, const struct pcap_pkthdr *header, const u_char *packet)
{

        static int count = 1; /* packet counter */

        /* declare pointers to packet headers */
        const struct sniff_ethernet *ethernet; /* The ethernet header [1] */
        const struct sniff_ip *ip;                      /* The IP header */
        const struct sniff_tcp *tcp;              /* The TCP header */
        const char *payload;                            /* Packet payload */

        int size_ip;
        int size_tcp;
        int size_payload;

        printf("\nPacket number %d:\n", count);
        count++;

        /* define ethernet header */
        ethernet = (struct sniff_ethernet *)(packet);

        /* define/compute ip header offset */
        ip = (struct sniff_ip *)(packet + SIZE_ETHERNET);
        size_ip = IP_HL(ip) * 4;
        if (size_ip < 20)
        {
                printf("   * Invalid IP header length: %u bytes\n", size_ip);
                return;
        }

        /* print source and destination IP addresses */
        printf("       From: %s\n", inet_ntoa(ip->ip_src));
        printf("         To: %s\n", inet_ntoa(ip->ip_dst));

        /* determine protocol */
        switch (ip->ip_p)
        {
        case IPPROTO_TCP:
                printf("   Protocol: TCP\n");
                break;
        case IPPROTO_UDP:
                printf("   Protocol: UDP\n");
                return;
        case IPPROTO_ICMP:
                printf("   Protocol: ICMP\n");
                return;
        case IPPROTO_IP:
                printf("   Protocol: IP\n");
                return;
        default:
```

*Untitled - Notepad

File Edit Format View Help

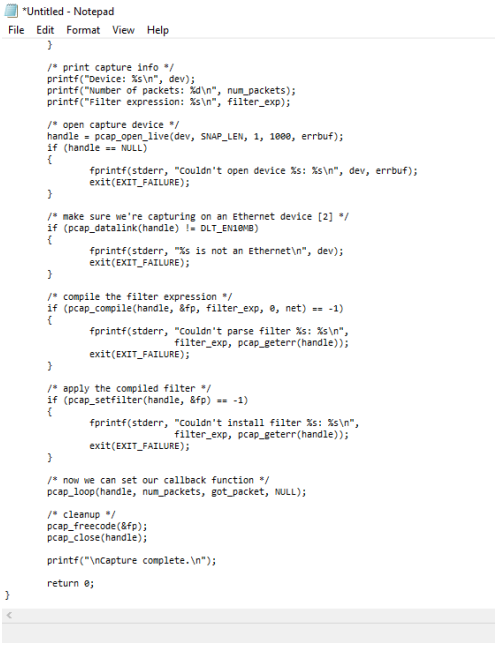
```
        case IPPROTO_IP:
                printf("   Protocol: IP\n");
                return;
        default:
                printf("   Protocol: unknown\n");
                return;
        }

        /*
         *  OK, this packet is TCP.
         */

        /* define/compute tcp header offset */
        tcp = (struct sniff_tcp *)(packet + SIZE_ETHERNET + size_ip);
        size_tcp = TH_OFF(tcp) * 4;
        if (size_tcp < 20)
        {
                printf("   * Invalid TCP header length: %u bytes\n", size_tcp);
                return;
        }

        printf("   Src port: %d\n", ntohs(tcp->th_sport));
        printf("   Dst port: %d\n", ntohs(tcp->th_dport));

        /* define/compute tcp payload (segment) offset */
        payload = (u_char *)(packet + SIZE_ETHERNET + size_ip + size_tcp);

        /* compute tcp payload (segment) size */
        size_payload = ntohs(ip->ip_len) - (size_ip + size_tcp);

        /*
         * Print payload data; it might be binary, so don't just
         * treat it as a string.
         */
        if (size_payload > 0)
        {
                printf("   Payload (%d bytes):\n", size_payload);
                print_payload(payload, size_payload);
        }

        return;
}

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 */
```



*Untitled - Notepad

File Edit Format View Help

```
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 TCP and dst portrange 10-100"; /* filter expression [3] */
        struct bpf_program fp;                                          /* compiled filter program (expression) */
        bpf_u_int32 mask;                                               /* subnet mask */
        bpf_u_int32 net;                                                /* ip */
        int num_packets = 10000;                                        /* 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");
                print_app_usage();
                exit(EXIT_FAILURE);
        }
        else
        {
                /* find a capture device if not specified on command-line */
                dev = pcap_lookupdev(errbuf);
                if (dev == NULL)
                {
                        fprintf(stderr, "Couldn't find default device: %s\n",
                                        errbuf);
                        exit(EXIT_FAILURE);
                }
        }

        /* get network number and mask associated with capture device */
        if (pcap_lookupnet(dev, &net, &mask, errbuf) == -1)
        {
                fprintf(stderr, "Couldn't get netmask for device %s: %s\n",
                                dev, errbuf);
                net = 0;
                mask = 0;
        }

        /* print capture info */
        printf("Device: %s\n", dev);
        printf("Number of packets: %d\n", num_packets);
```



*Untitled - Notepad

File Edit Format View Help

```
        }

        /* print capture info */
        printf("Device: %s\n", dev);
        printf("Number of packets: %d\n", num_packets);
        printf("Filter expression: %s\n", filter_exp);

        /* open capture device */
        handle = pcap_open_live(dev, SNAP_LEN, 1, 1000, errbuf);
        if (handle == NULL)
        {
                fprintf(stderr, "Couldn't open device %s: %s\n", dev, errbuf);
                exit(EXIT_FAILURE);
        }

        /* make sure we're capturing on an Ethernet device [2] */
        if (pcap_datalink(handle) != DLT_EN10MB)
        {
                fprintf(stderr, "%s is not an Ethernet\n", dev);
                exit(EXIT_FAILURE);
        }

        /* compile the filter expression */
        if (pcap_compile(handle, &fp, filter_exp, 0, net) == -1)
        {
                fprintf(stderr, "Couldn't parse filter %s: %s\n",
                                filter_exp, pcap_geterr(handle));
                exit(EXIT_FAILURE);
        }

        /* apply the compiled filter */
        if (pcap_setfilter(handle, &fp) == -1)
        {
                fprintf(stderr, "Couldn't install filter %s: %s\n",
                                filter_exp, pcap_geterr(handle));
                exit(EXIT_FAILURE);
        }

        /* now we can set our callback function */
        pcap_loop(handle, num_packets, got_packet, NULL);

        /* cleanup */
        pcap_freecode(&fp);
        pcap_close(handle);

        printf("\nCapture complete.\n");

        return 0;
}
```

Task 1.1: Understanding how a Sniffer Works

Here we capture any ICMP packets moving between the victim and a random ip address in this case I have chosen 74.6.136.150

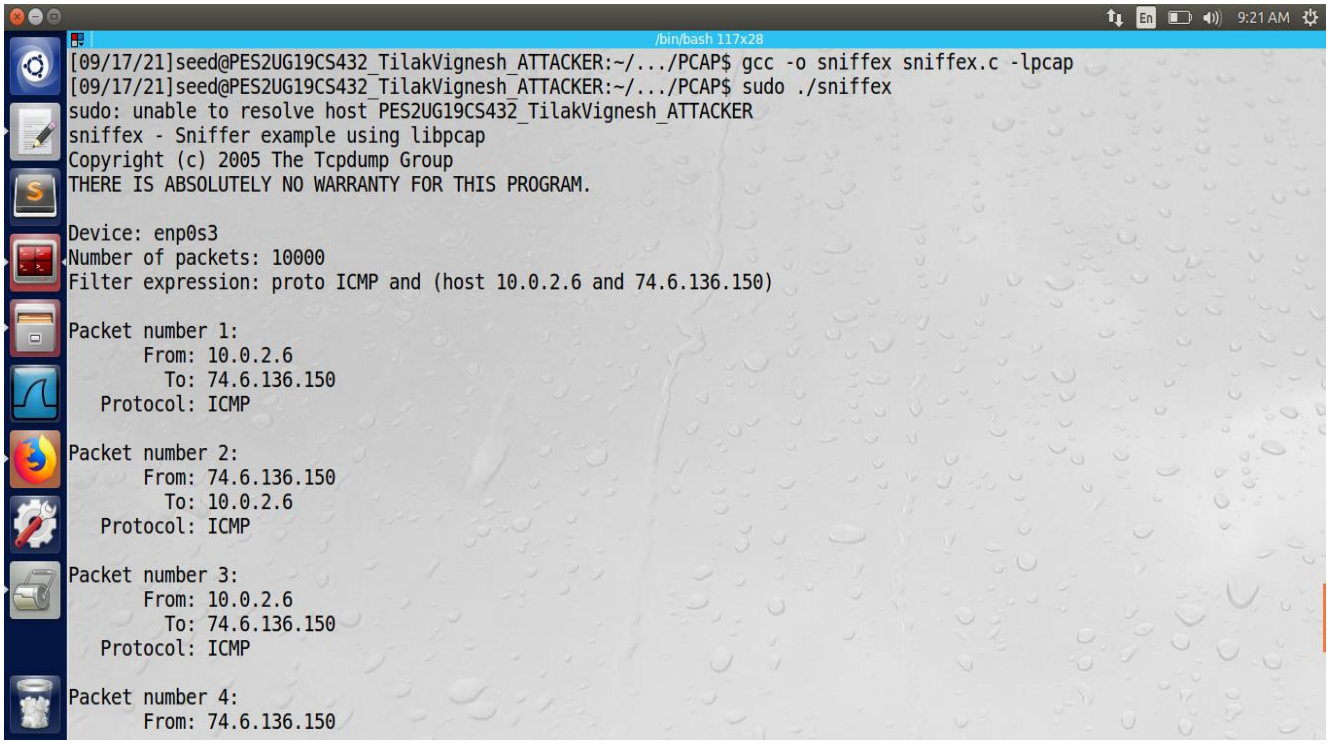
Promiscuous Mode On:

Command:

```
gcc -o sniffex sniffex.c-

lpcap
```

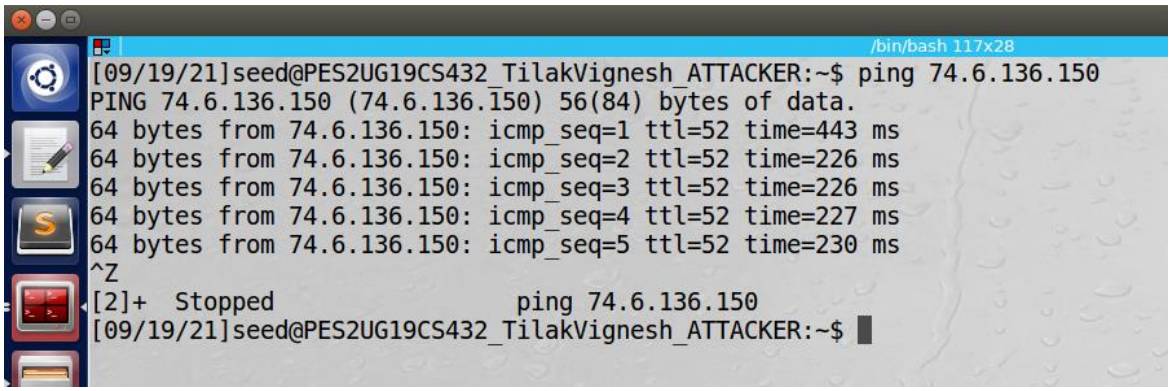
```
sudo ./ sniffex
```

ATTACKER:



The attacker captures packets with the host ip addr 10.0.2.6 and the ip addr 74.6.136.150

open another terminal in same VM and ping any ip address



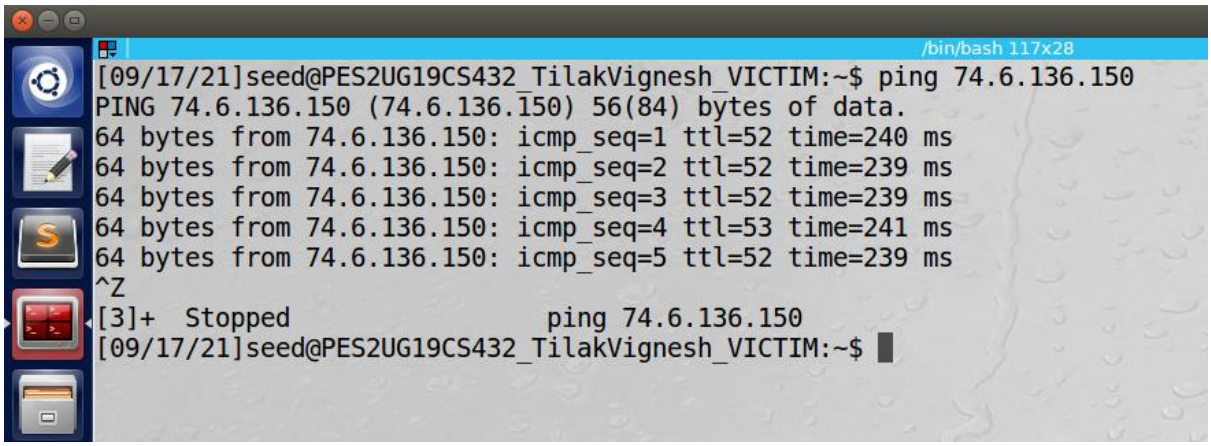
The ping works normally, nothing unusual about this is detected. These packets aren't captured by the sniffer program

Command:

`ping 1.2.3.4(any ipaddress)`

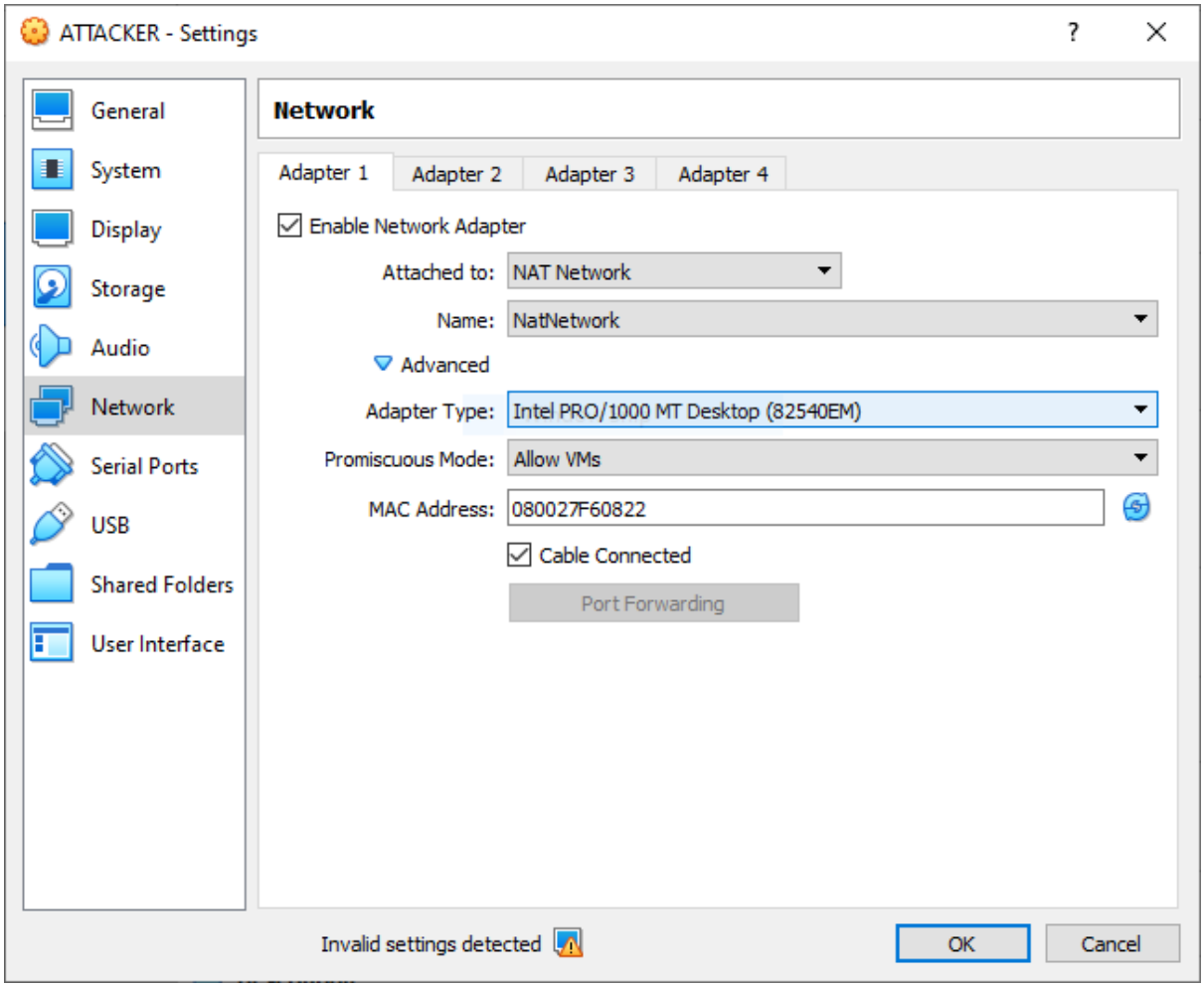
Provide a screenshot of your observations.

VICTIM:



Here the victim is pinging the ip address 74.6.136.150 which in turn is sending back responses. The attacker captures all of this since the promiscuous mode is on.

Show that when promiscuous mode is switched on the sniffer program can sniff through all the packets in the network.



As the settings show, the promiscuous mode is on hence all these packets can be sniffed or else that wouldn't have been the case.

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

For the sniffer program we first define the filter, this tells us what type of packets we need to capture. We then compile this filter into BPF using pcap_compile and then set the filter using pcap_set AFTER opening a live session on any NIC using pcap_open_live. Then the packets are then captured using the pcap_loop function and finally closes using the pcap_close function.

The pcap_loop function takes a function parameter. This function defines how to handle captured packets.

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

Root privileges are necessary to run sniffex because the program operates in promiscuous mode. Since we use a raw socket to sniff packets in the low level layers, these sockets are required to operate in root privilege as they bypass normal functioning.

The program fails on opening the device (i.e. NIC). It says couldn't open device, need root privileges to do so. (It fails during the pcap_open_live part of the code)

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

The answer to this is giving through the course of the first task.

Promiscuous Mode Off:

Promiscuous mode can be switched off in the attacker machine:

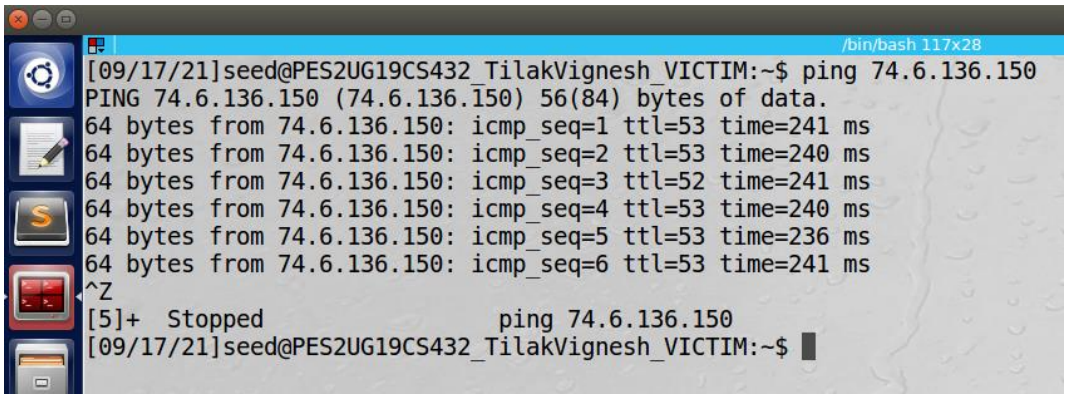
Go to Machine -> Settings -> Network -> Advanced -> Promiscuous mode -> "Deny"

Show that when promiscuous mode is switched off the sniffer program is not able to sniffthrough all the packets in the network.

Command:

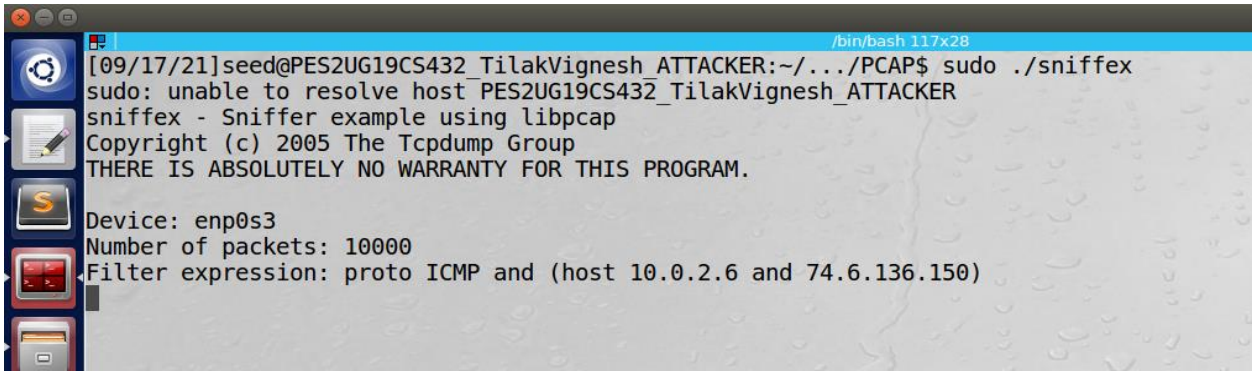
```
gcc -o sniffex sniffex.c -lpcapsudo

./sniffex
```

VICTIM:



The victim pings the ip address just in the previous task

ATTACKER:



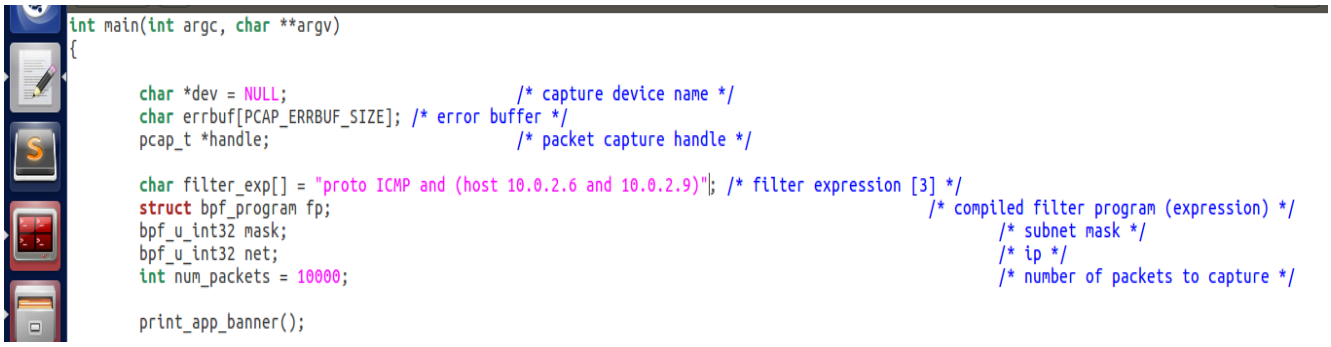
The attacker doesn't capture a single packet from the victim or the ip address, this shows that when promiscuous mode is off the packets aren't captured.

Task 1.2: Writing Filters

1) Capture the ICMP packets between two specific hosts

We capture ICMP packets between the victim and the server by modifying the filter to capture these packets.

CODE :



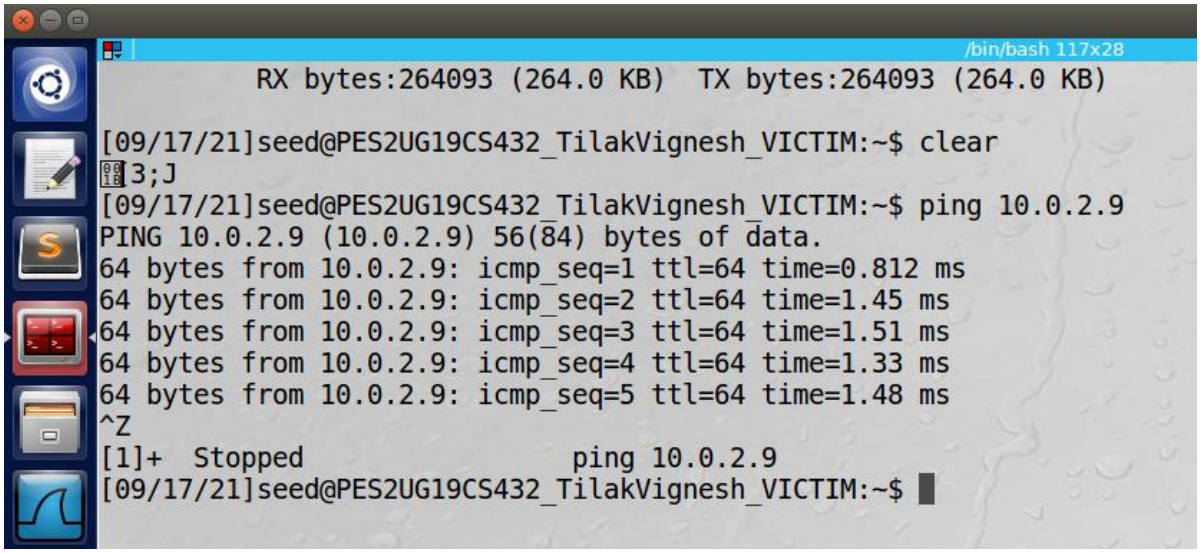
```
int main(int argc, char **argv)
{

        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 10.0.2.6 and 10.0.2.9)"; /* filter expression [3] */
        struct bpf_program fp;                                              /* compiled filter program (expression) */
        bpf_u_int32 mask;                                                   /* subnet mask */
        bpf_u_int32 net;                                                    /* ip */
        int num_packets = 10000;                                            /* number of packets to capture */

        print_app_banner();
```

In the filter_exp string we can see that packets with host ip 10.0.2.6(victim) and 10.0.2.9(server) are captured.

Command:

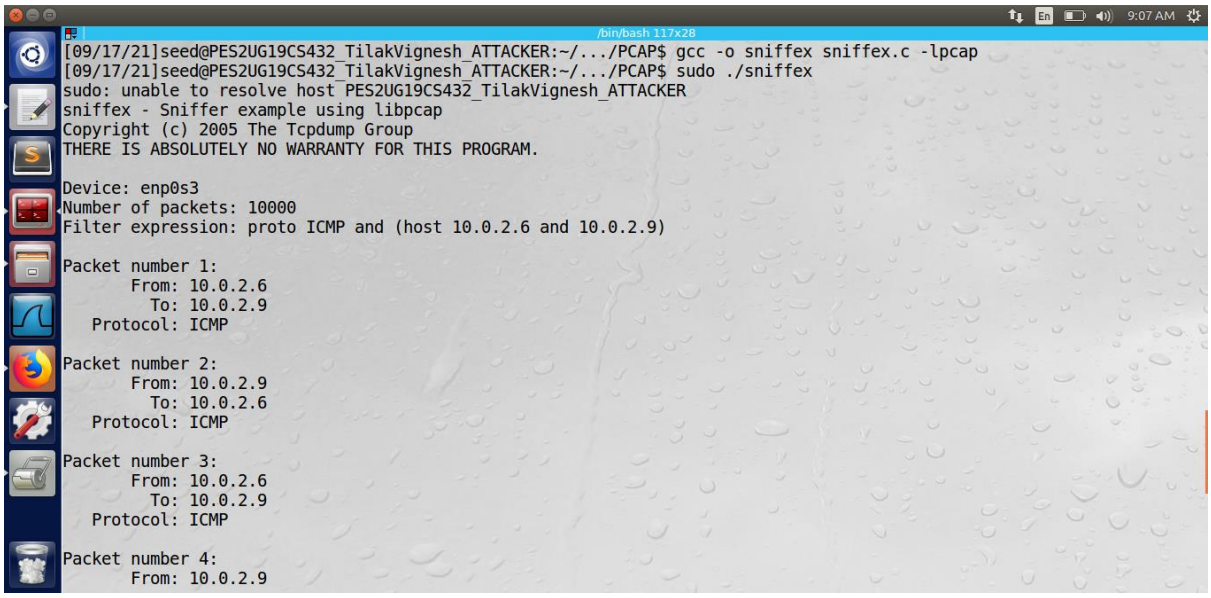
```
gcc -o sniffex sniffex.c -lpcap

sudo ./sniffex
```

VICTIM:



Here the victim is pinging the server.

ATTACKER:

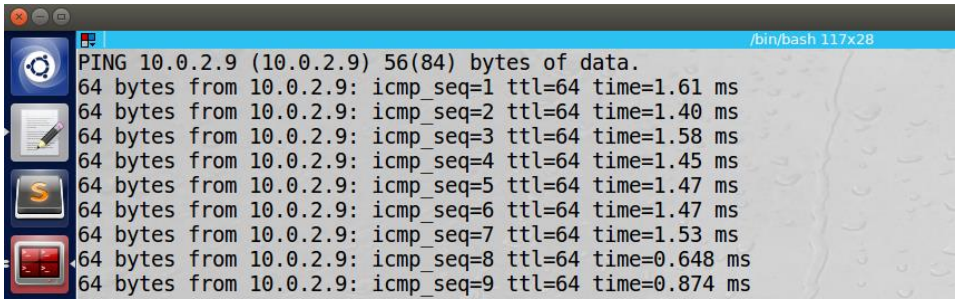


We can see that packets are captured between 2 hosts successfully.

open another terminal in the same VM and ping any ip address

ping 10.0.2.9

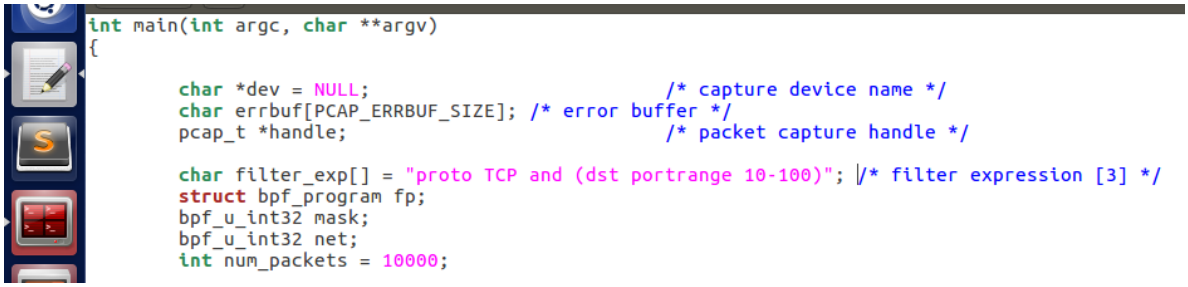
ATTACKER:



There's absolutely no difference in the packets captured by the sniffer program. It doesn't show these packets which are pinged.

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

CODE:



```
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 TCP and (dst portrange 10-100)"; /* filter expression [3] */
        struct bpf_program fp;
        bpf_u_int32 mask;
        bpf_u_int32 net;
        int num_packets = 10000;
```

You can observe the filter here. This filter captures only TCP packets with the destination ports ranging from 10-100

Command:

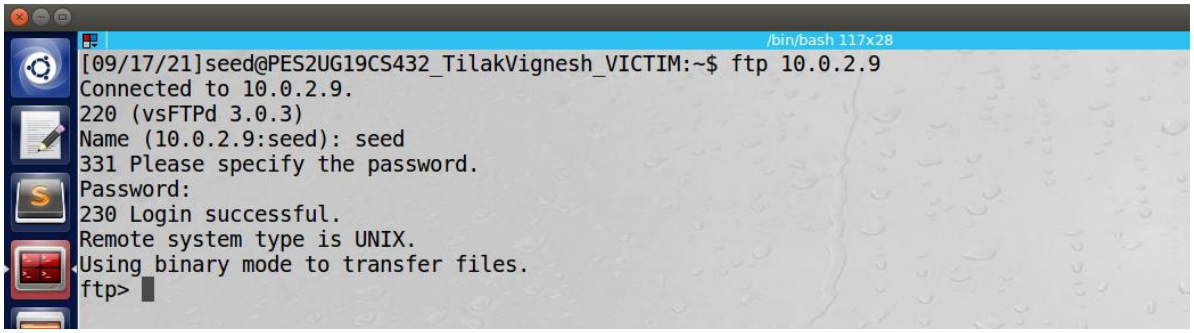
sudo ./sniffex

ftp 10.0.2.8

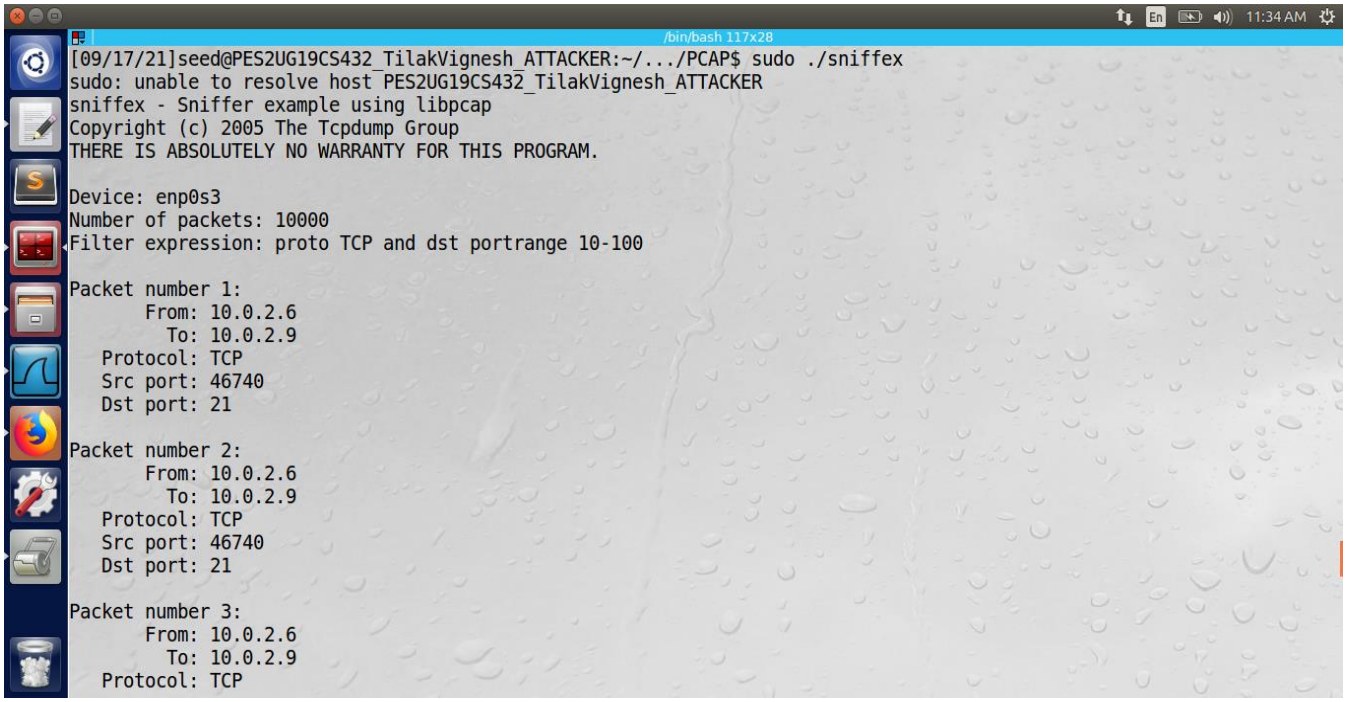
We are establishing an ftp stream between the victim and the server and the attacker is sniffing the packets.

Provide screenshots of your observations.

VICTIM:



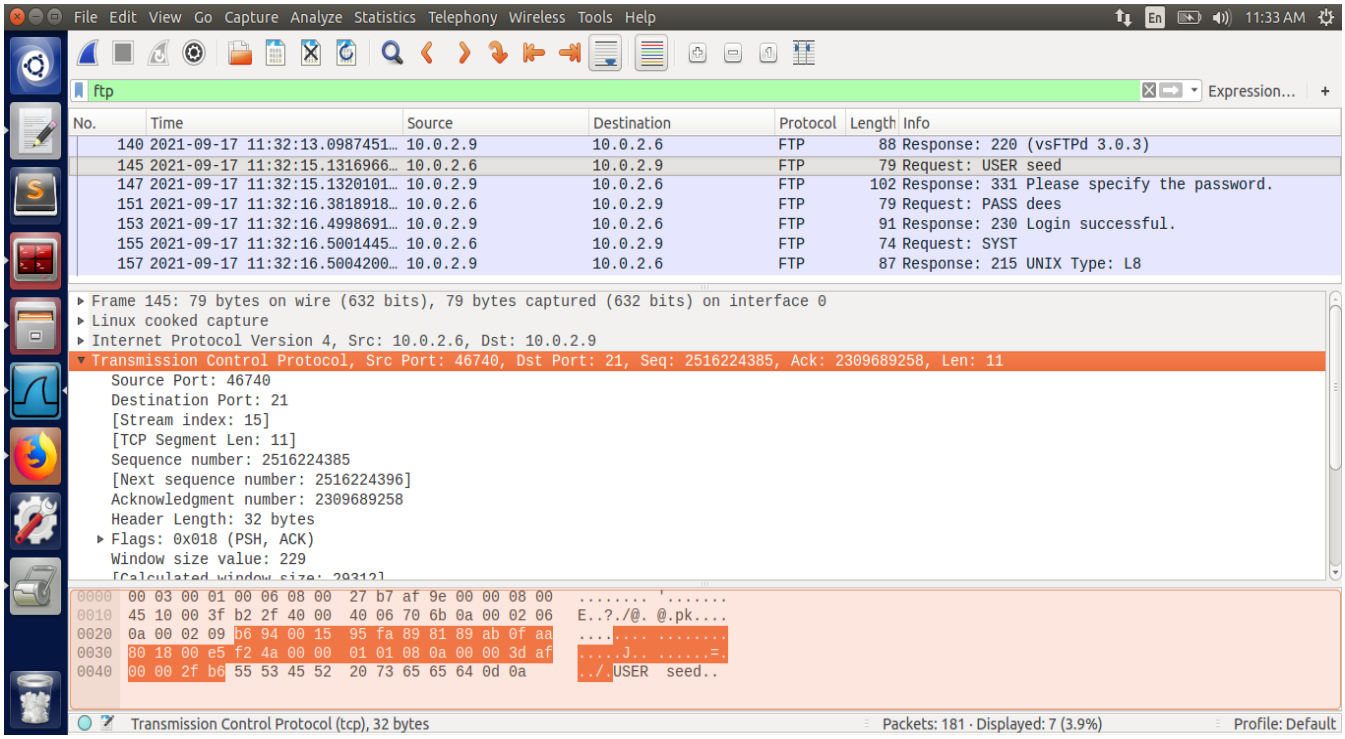
ATTACKER:



We can observe from the above screenshot that ftp packets between the victim and the attacker are captured.

Note: Observe Source port and Destination port using Wireshark capture.

WIRESHARK CAPTURE:



In the wireshark capture above we see that the src port can be anything, for this particular packet it's 46748 but the dst port number always remains the same. This is done by the filter we wrote earlier.

Task 1.3: Sniffing Passwords

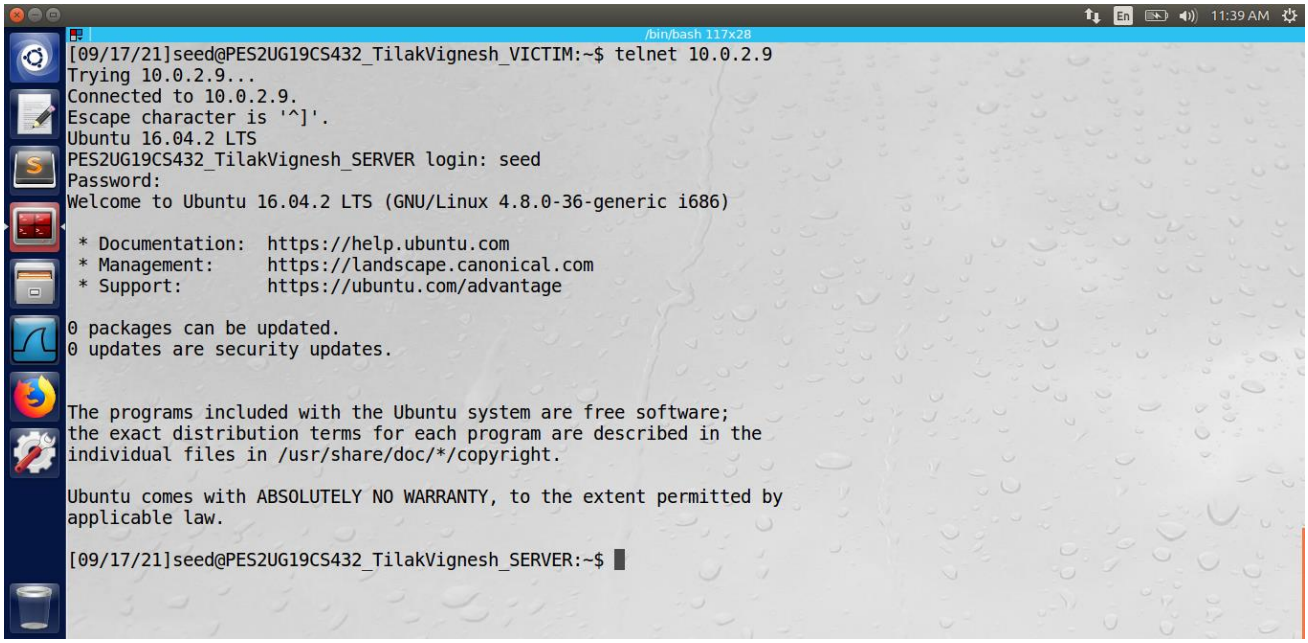
We sniff telnet packets and in turn get the password of the server.

Command:

telnet 10.0.2.9

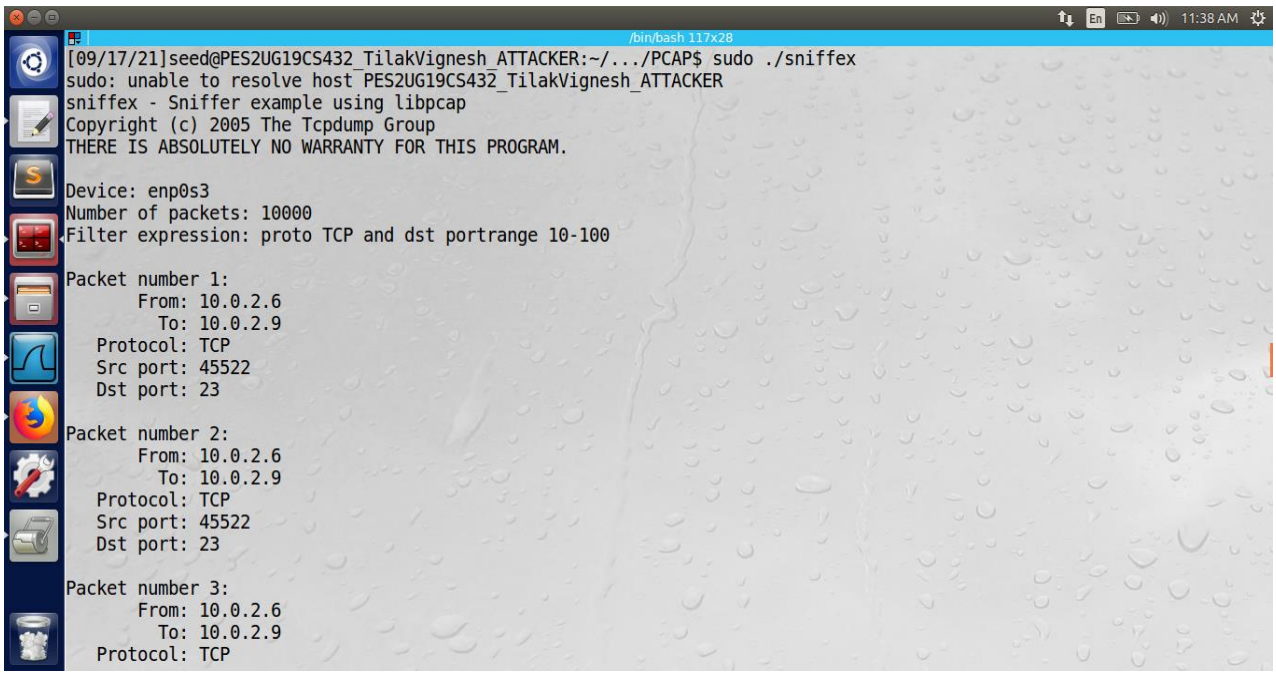
Provide screenshots of your observations

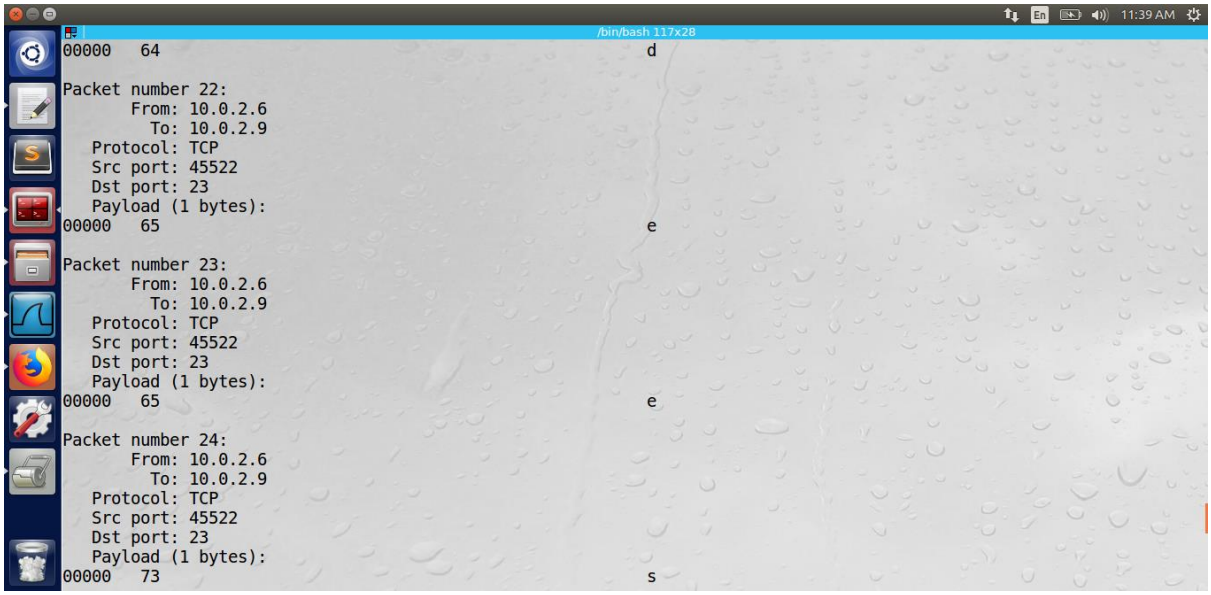
VICTIM:



The victim connects to the server in the above screenshot.

ATTACKER:





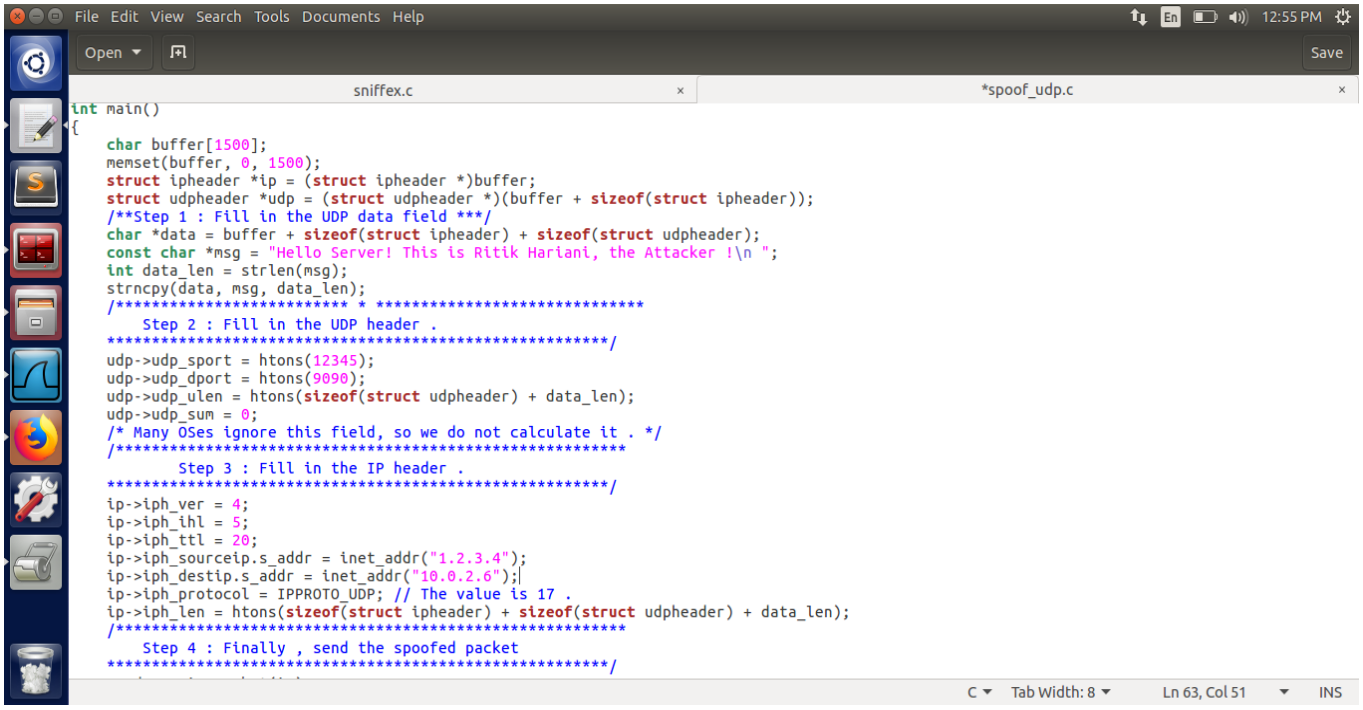
While the victim is connecting to the server the sniffer program runs in the attacker machine. In the above screenshot the password is captured letter by letter. Hence we deduct that the password of the server is "dees".

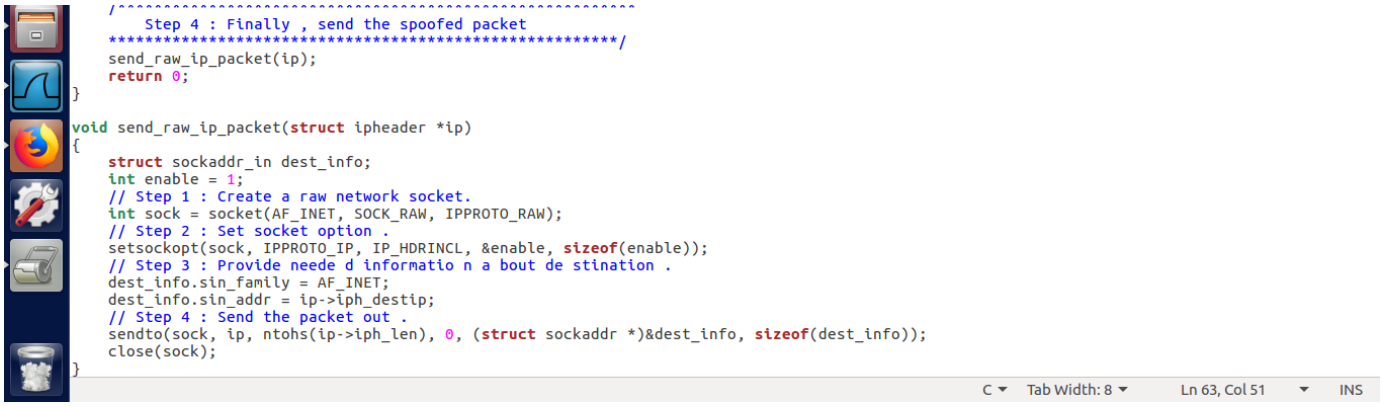
Task 2: Spoofing

The objectives of this task is to create raw sockets and send spoof packets to the user/victim machine raw sockets give programmers the absolute control over the packet construction.

Task 2.1 - A Writing a spoofing program:

CODE :





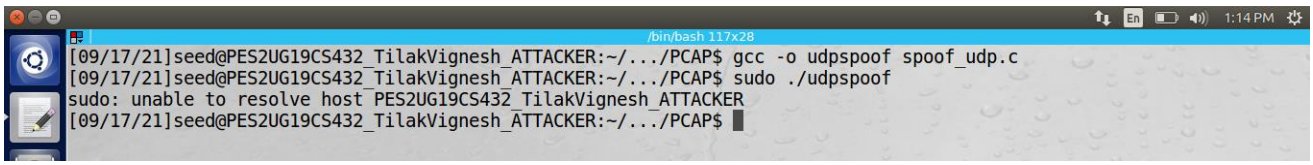
The above code shows all the necessary steps needed to fabricate and send a udp packet to the victim.

Commands in VM1:

```
sudo ./udpspoof
```

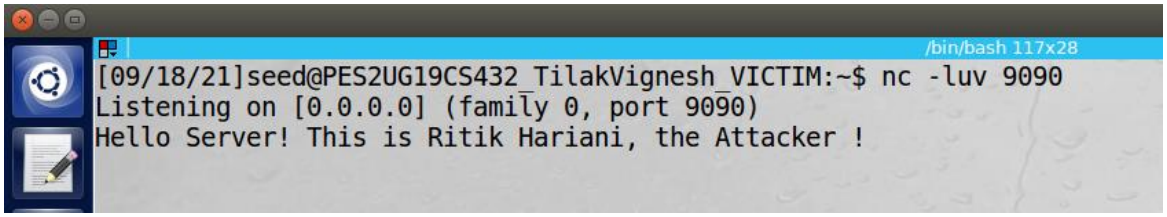
Please provide a screenshot of your observations.

ATTACKER:



We can see that the attacker has sent a spoofed UDP packet to the victim from the ip address 1.2.3.4

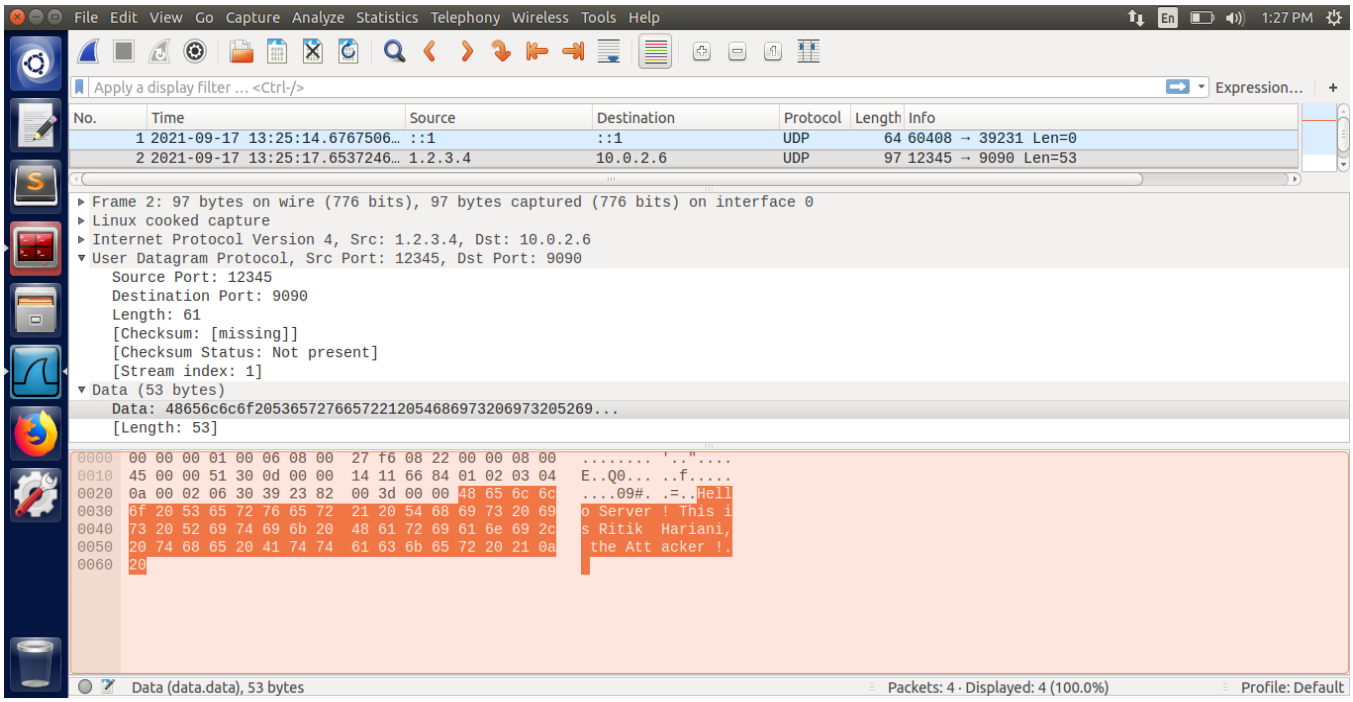
VICTIM:



The victim receives a UDP packet on port 9090 sent by the attacker. This is shown in the above screenshot.

NOTE: While doing this task I found that the message is not always displayed on the terminal as shown in the above screenshot. Most of the times the message doesn't even show up on the terminal but the packet is captured by wireshark.

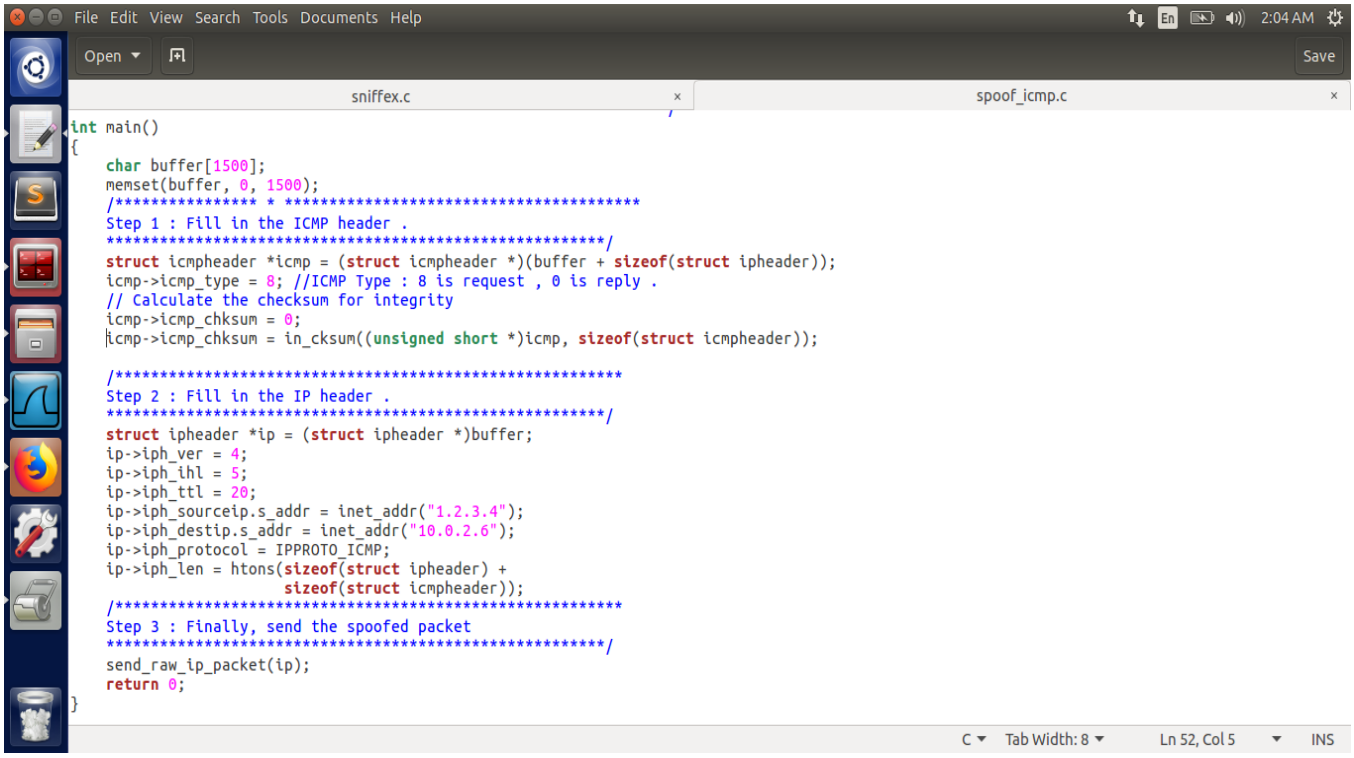
WIRESHARK CAPTURE:



The wireshark capture shows the spoofed udp packet captured and the payload associated with it.

Task 2.2 – Spoof an ICMP Echo Request

CODE:



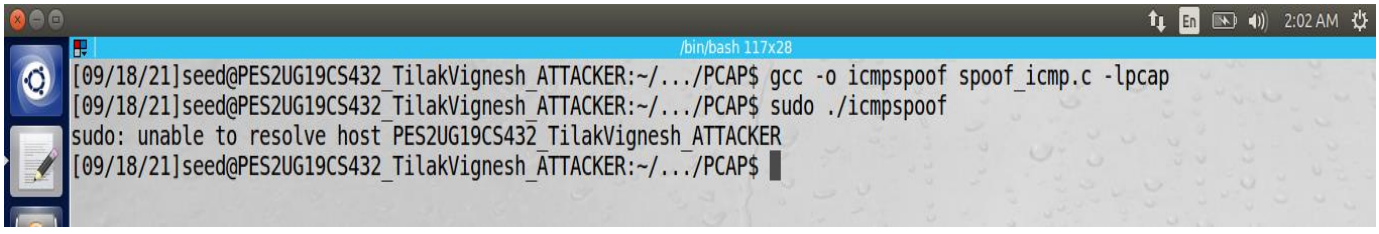
The screenshot above shows only the main fragment of the code required to spoof an ICMP echo request.

Commands :

```
sudo ./icmpspoof
```

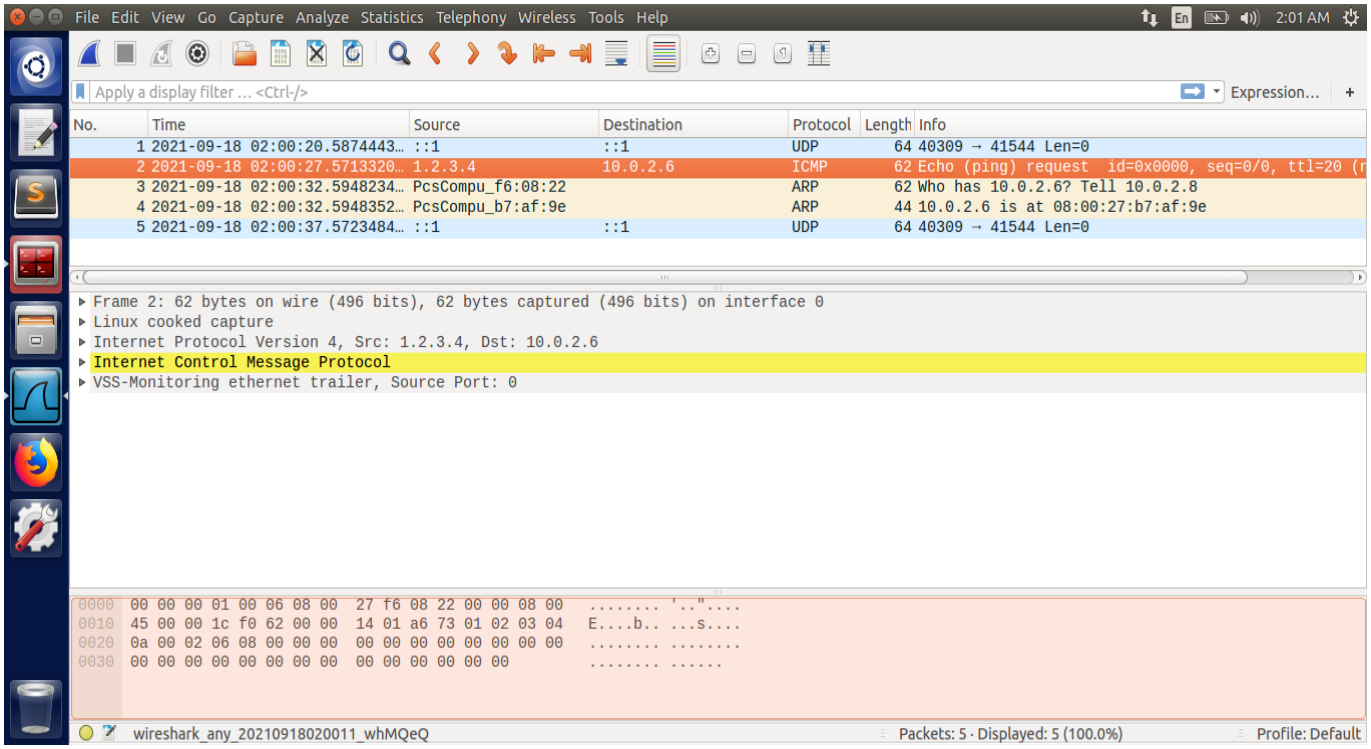
please provide the screenshot of your observation

ATTACKER:



The attacker sends a spoofed ICMP packet to the victim.

VICTIM WIRESHARK CAPTURE:

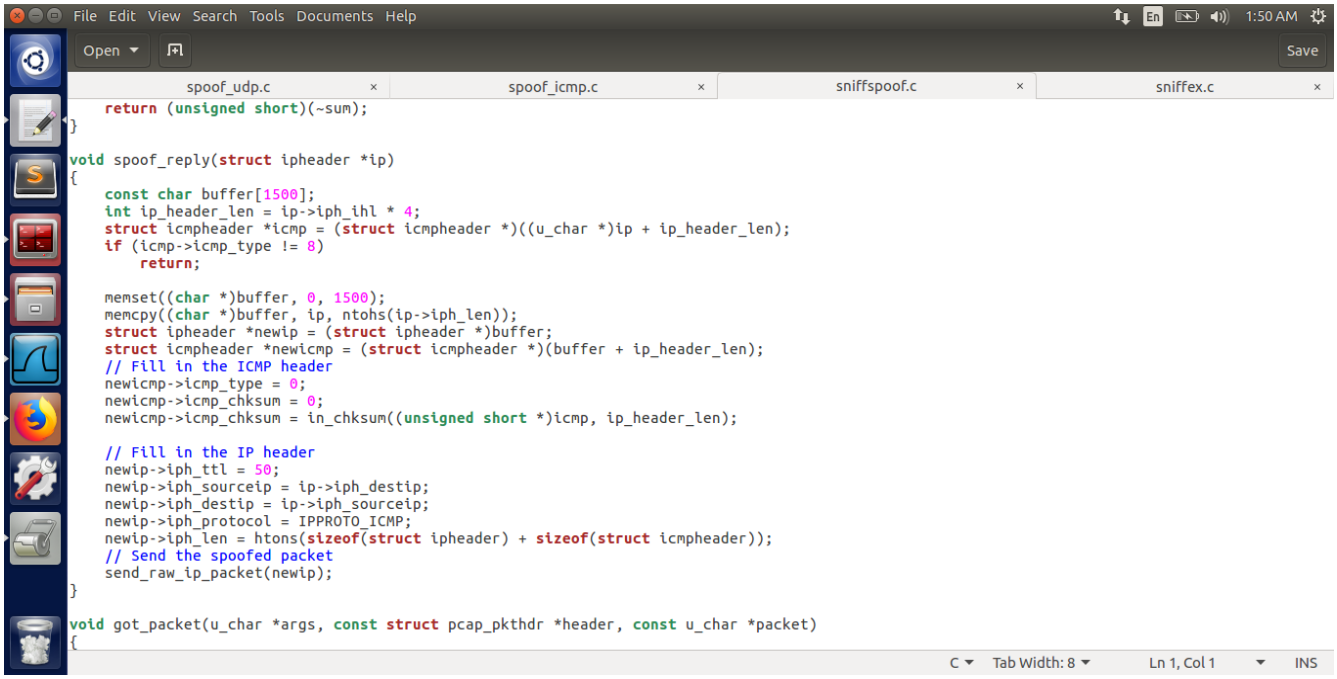


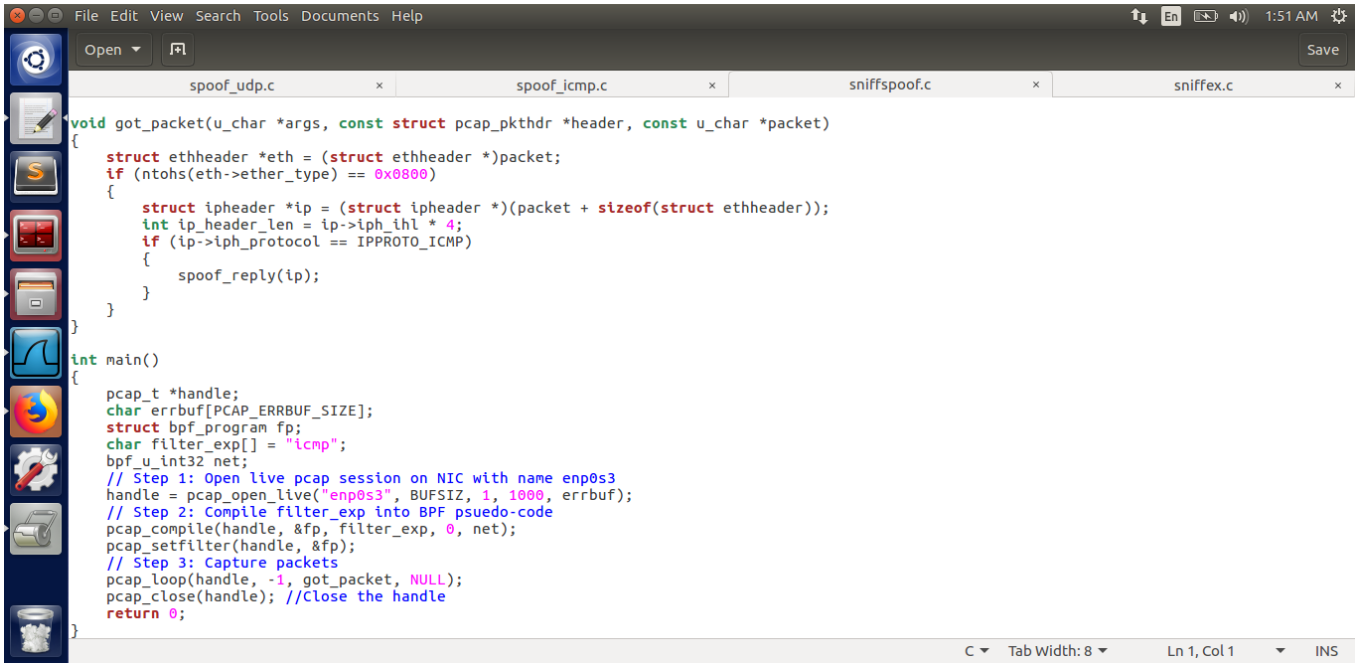
We can see that an echo ping request is sent from an ip address 1.2.3.4 to the victim. This is the spoofed icmp echo request sent by the attacker.

Task 2.3 – Sniff and then Spoof

VM A (Victim) : 10.0.2.6
Ping X : 1.2.3.4
VM B (attacker with sniffer-spoofer running): 10.0.2.8

CODE :





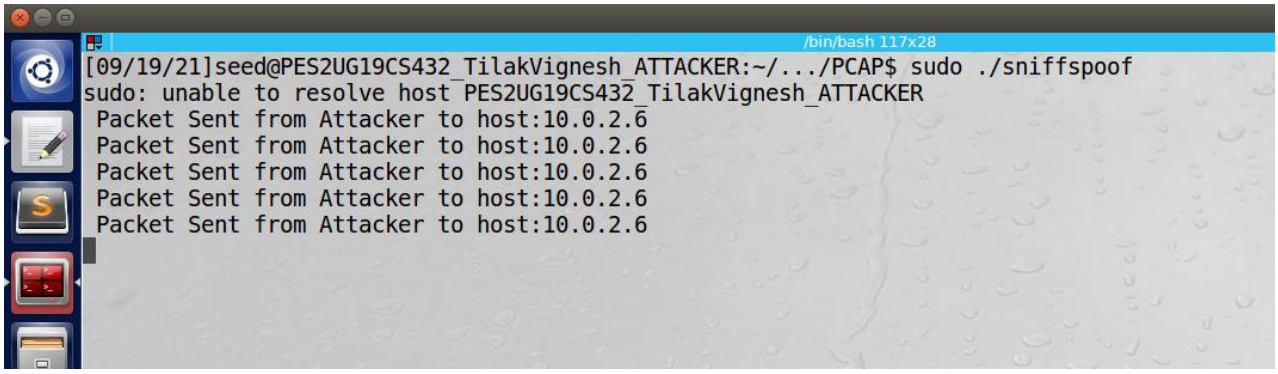
The above screenshot the main fragments of code in the sniffing followed by spoofing task.

Command:

gcc -o sniffspoof sniffspoof.c -lpcap

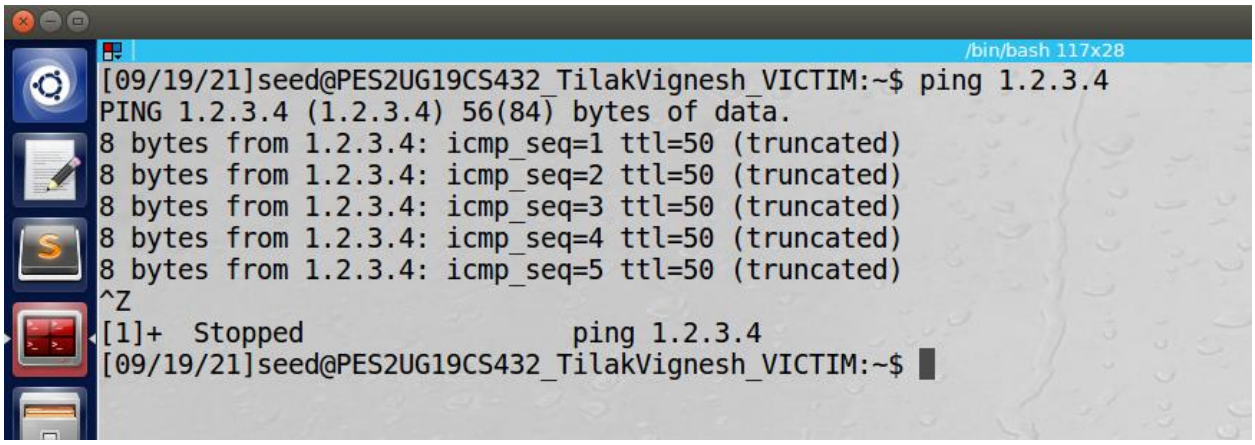
sudo ./sniffspoof

ATTACKER:



The attacker first sniffs the packets coming from the victim and then sends back a spoofed reply to the victim.

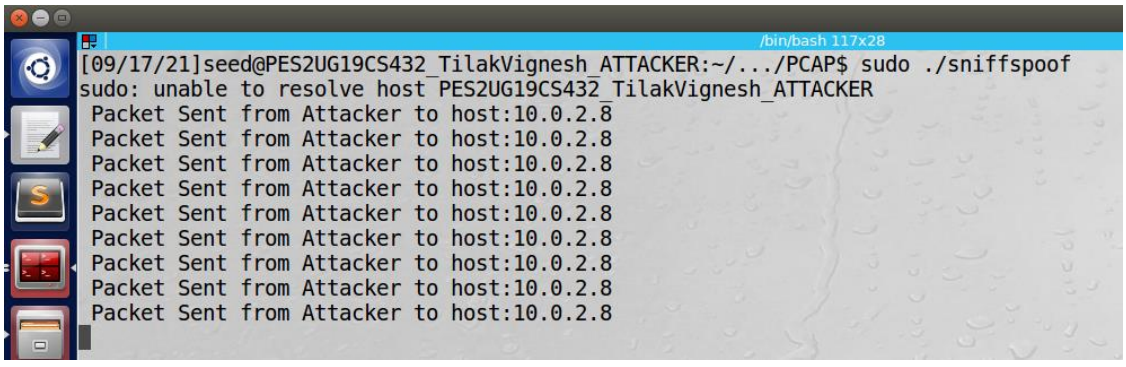
VICTIM:



Without running the sniffspoof program the ping request does not give a response. As soon as the program runs on the attacker ICMP responses as received by the victim.

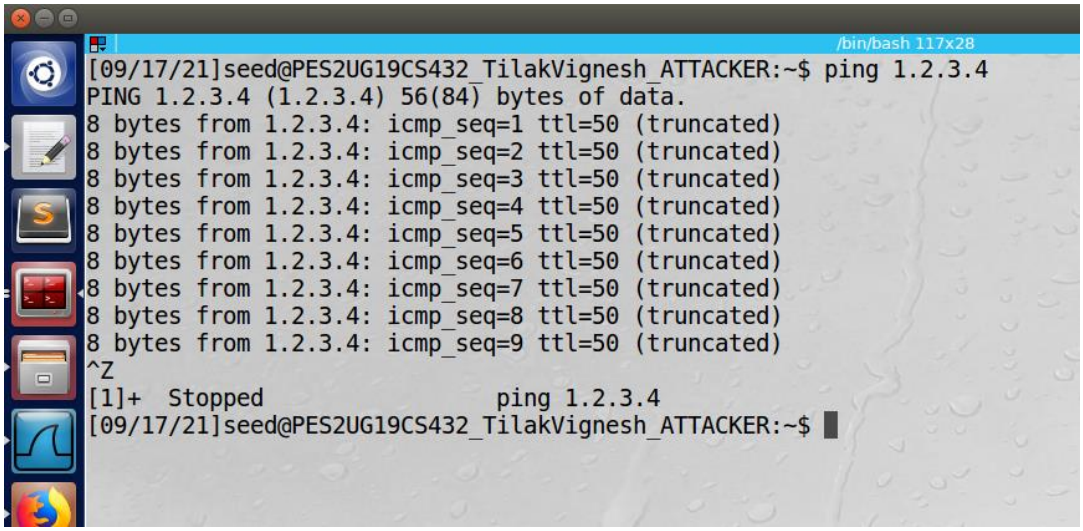
Open one more terminal on the same VM and ping 1.2.3.4.

TERMINAL 1:



The first terminal essentially acts as the attacker sending spoofed packets after sniffing.

TERMINAL 2:



Terminal 2 acts as the victim. When we start pinging 1.2.3.4 there's no response until the sniffspoof program is run.

This shows that the sniffspoof program will work for any device as long as the device(victim) is in the same LAN.

NAME: TILAK VIGNESH MEKALA

SRN: PES2UG19CS432

SEM: 5

SECTION: G

CSE

PESU-ECC