Cristian Molina

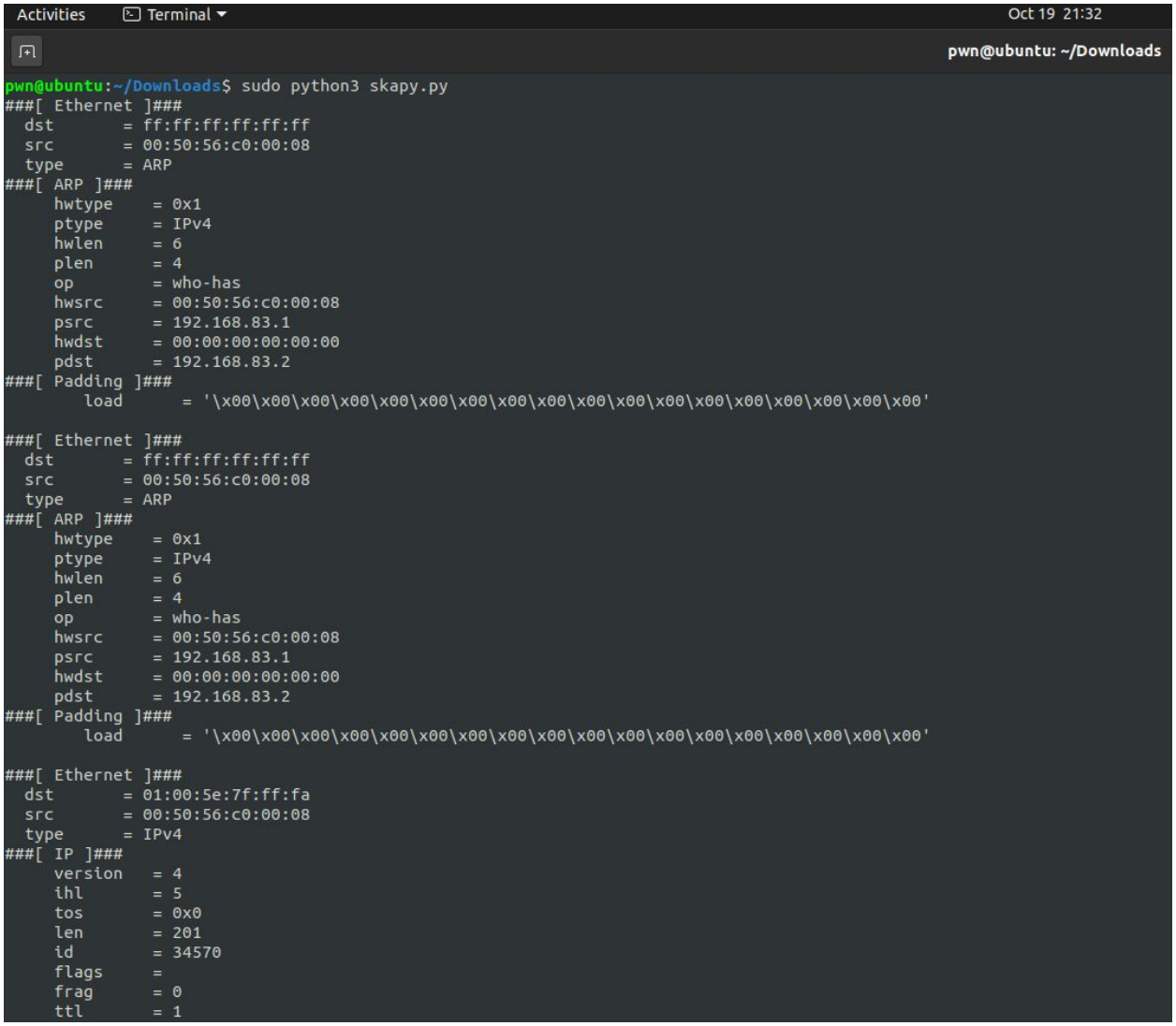
Computer Networks

Dr. Tosh

Assignment #3

Assignment 3 – Computer Networks

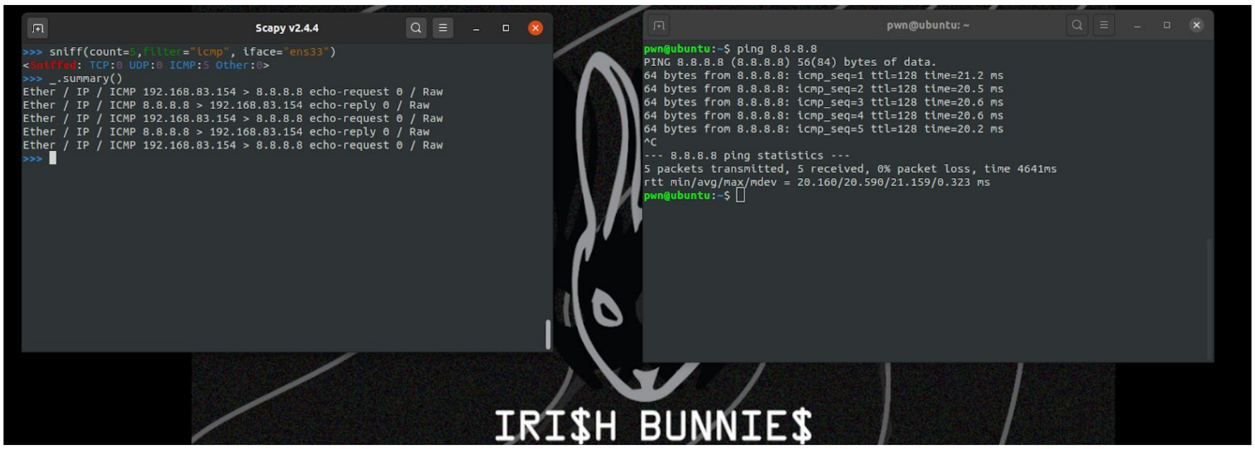
Task 1: Sniffing Packets using Scapy. **(with sudo command)**
Using the sudo command, it is possible to see the network traffic. This is shown in hex format. There is also an ARP table, a padding section and an IP section.



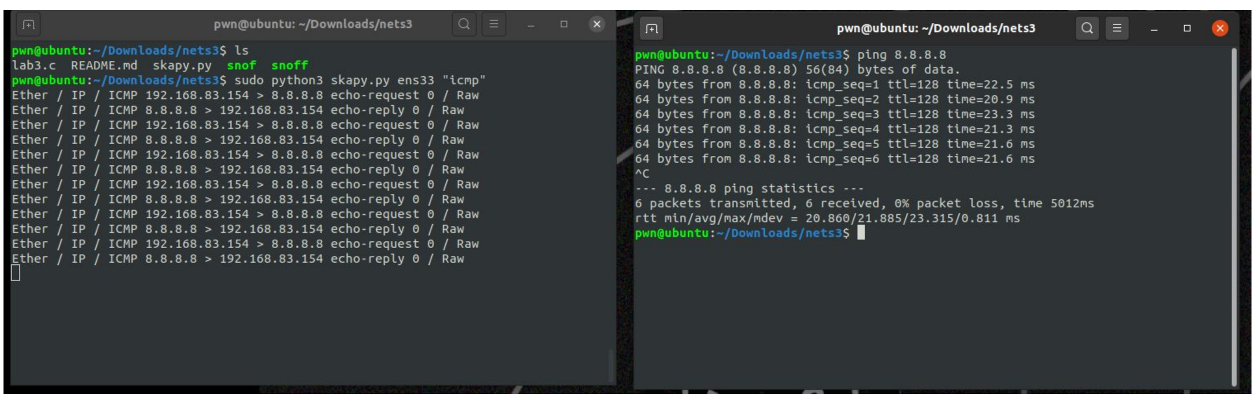
Running Scapy **(without sudo command)**
Scapy will not run without root privileges. Trying to run scapy without sudo will result in a permission error.

b) **Using filters in scapy (ICMP) - Using Terminal Only**

I pinged 8.8.8.8 and set the filter to icmp with an interface of ens33

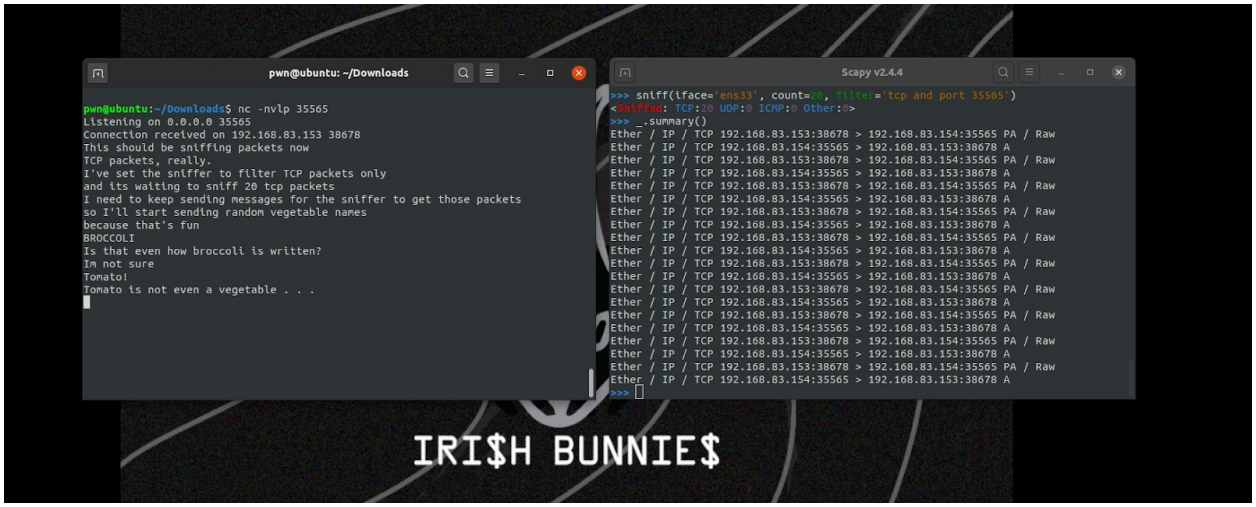


Using my Python Script:

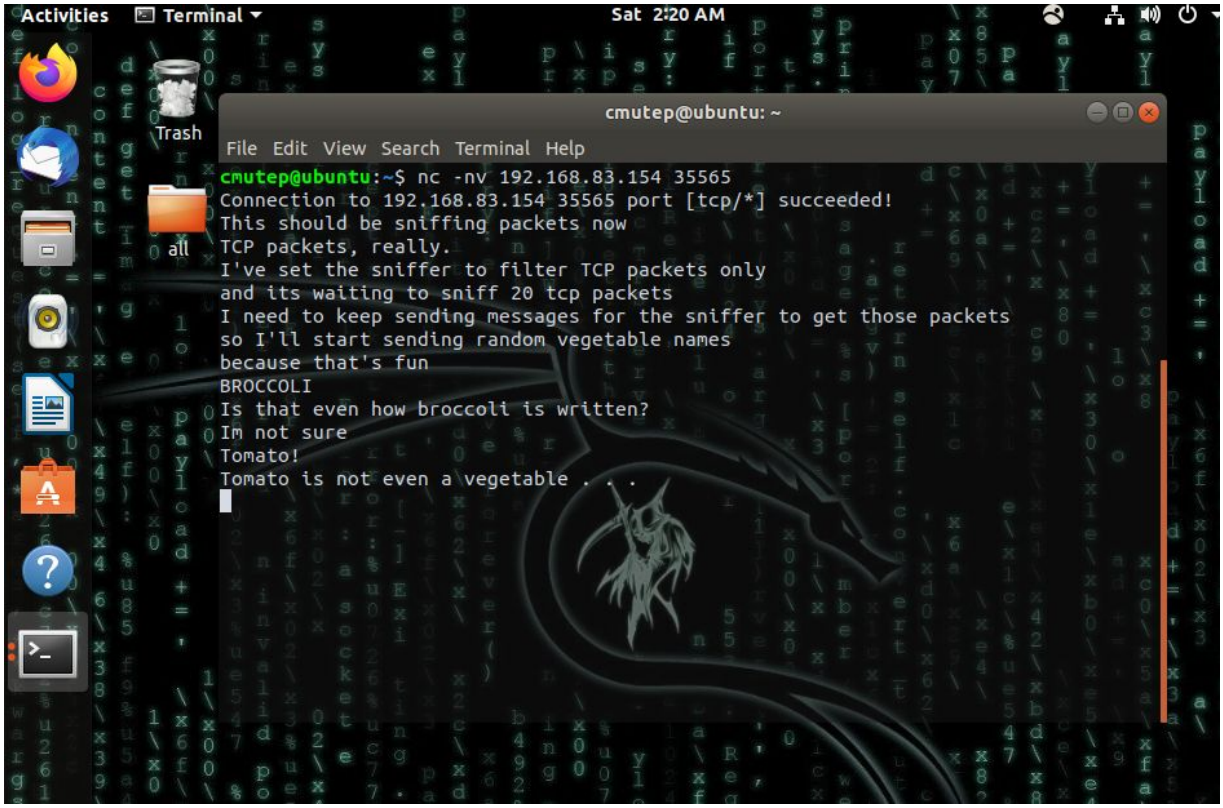


Filtering TCP packets - Using Terminal Only

This part involves two virtual machines. I tried doing everything on only one virtual machine but it would not sniff anything that way. The terminal on the left is listening on port 35565 and the terminal on the right is sniffing tcp packets.

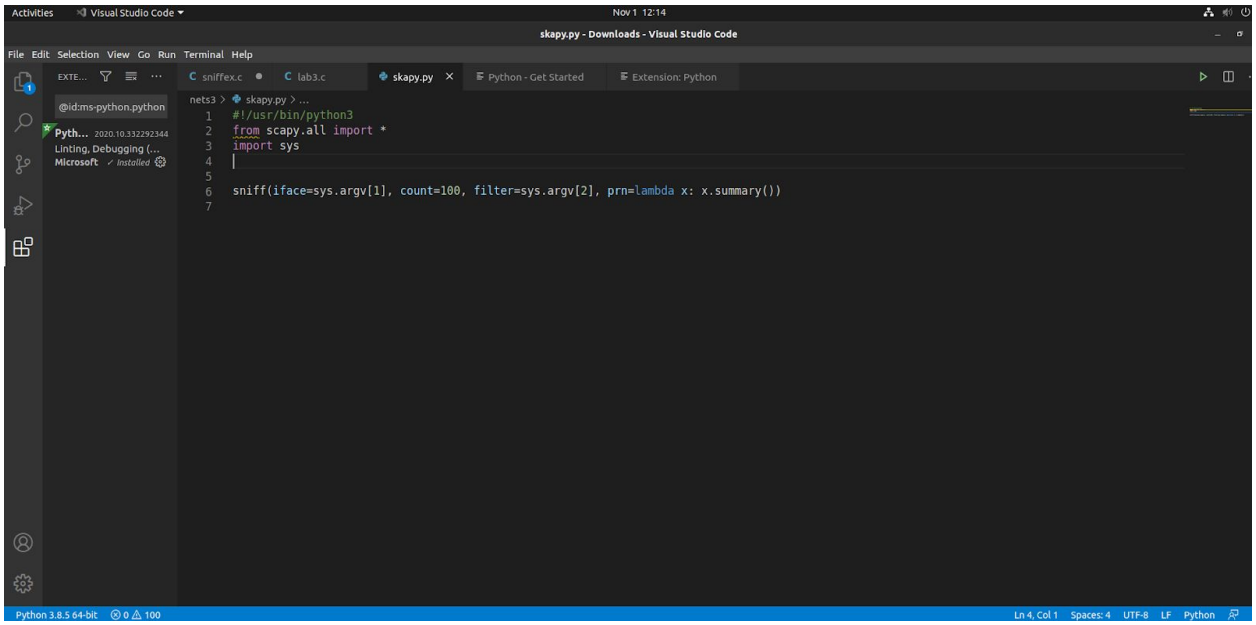


This is another VM that is using netcat to connect to the other listening machine. As they communicate, the sniffer detects tcp packets.



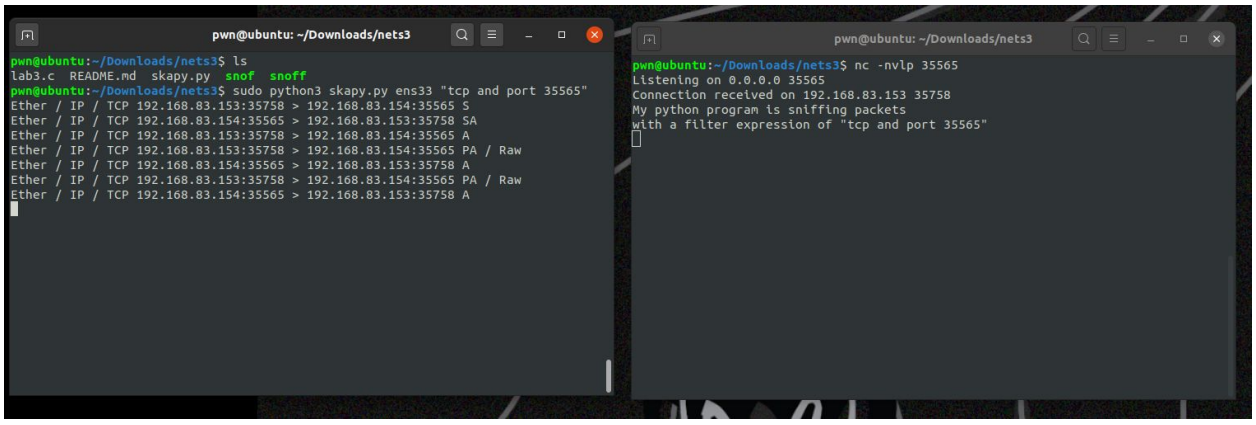
Filtering TCP Packets using my Python Program

Picture of the code

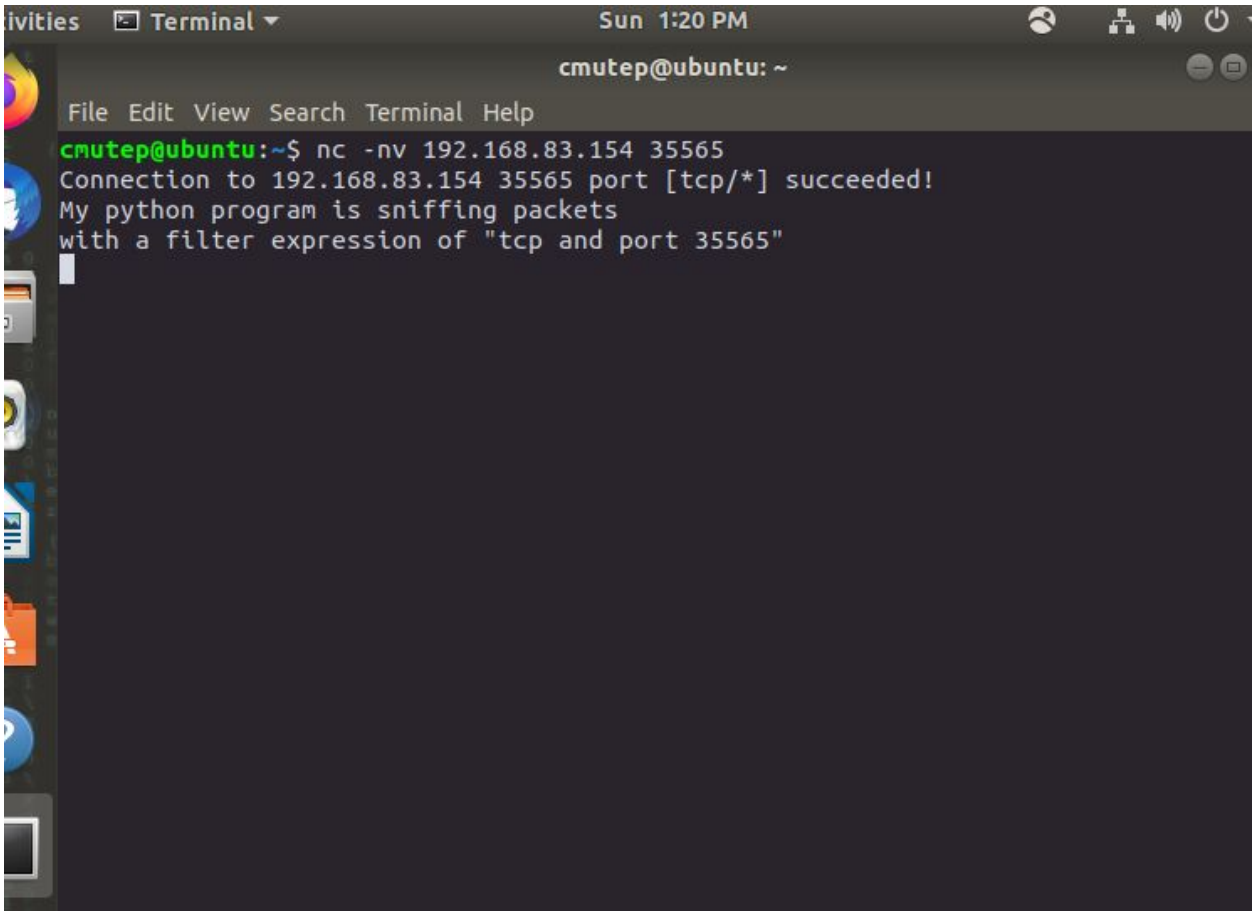


Running it

Here you can see my program running on the left, sniffing TCP packets and a netcat listener on the right on port 35565. As you can see, my program works exactly as the terminal version above.



Again, this is another VM and communicates with the VM above through port 35565.



2) Sniffing ICMP packets in C - Code Picture

```c
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[] = "icmp";     /* 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 = 10;           /* number of packets to capture */

    /* 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);
    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;
}
```

2.1 In your own words, describe the sequence of the library calls that are essential for sniffer programs. (Just summary only)

> Pcap_open_live -> opens capture device

> pcap_compile -> compiles the filter expression

> pcap_setfilter -> applies the compiled filter

>pcap_loop -> sets the callback function

>got_packet -> the callback function (dissects/prints packet)

2.2 Why do you need the root privilege to run a sniffer program?

Because to put the network adapter into promiscuous mode requires root privileges. If it didn't, evey user could fully control the network adapter, then every user could see the full traffic on that controller, including all the traffic of other users.

Where does the program fail if it is executed without the root privilege?

Running sniffer without root privileges. . .

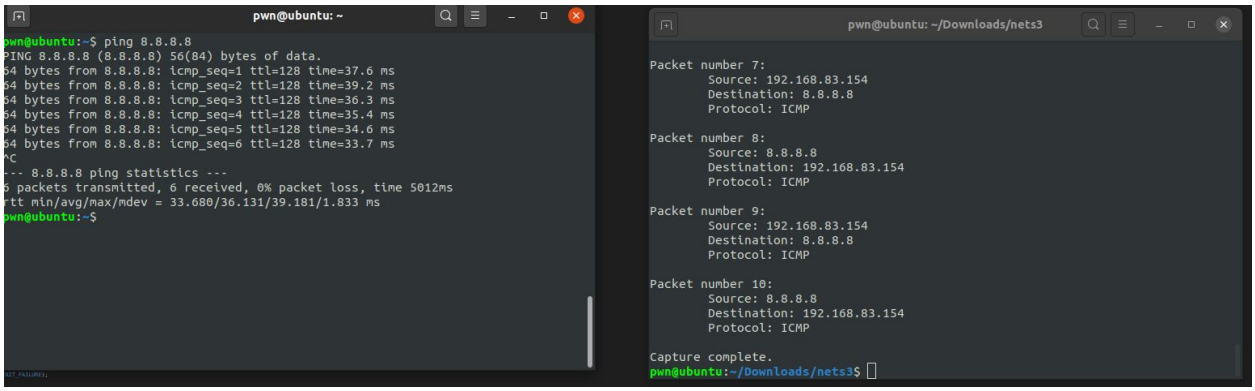
```
pwn@ubuntu:~/Downloads/nets3$ ./snof
Device: ens33
Number of packets: 100
Filter expression: tcp and port 23
Couldn't open device ens33: ens33: You don't have permission to capture on that device (socket: Operation
 not permitted)
pwn@ubuntu:~/Downloads/nets3$
```

Taking a look at the code, it fails in the pcap_open_live function.

```
/* 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);
}
```

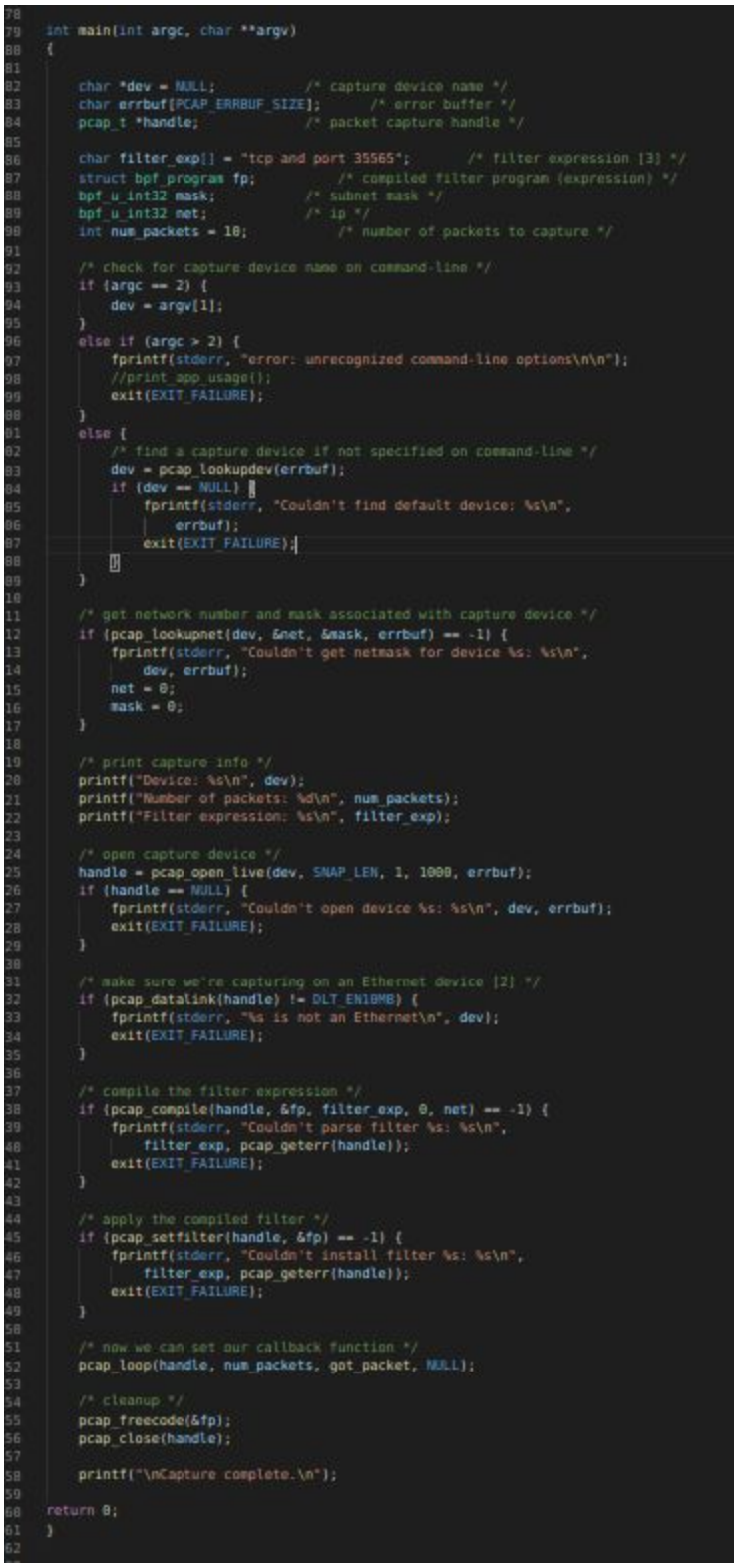
Running the code with ICMP filter

The code captures 10 packets, showing the source and destination address of those packets.



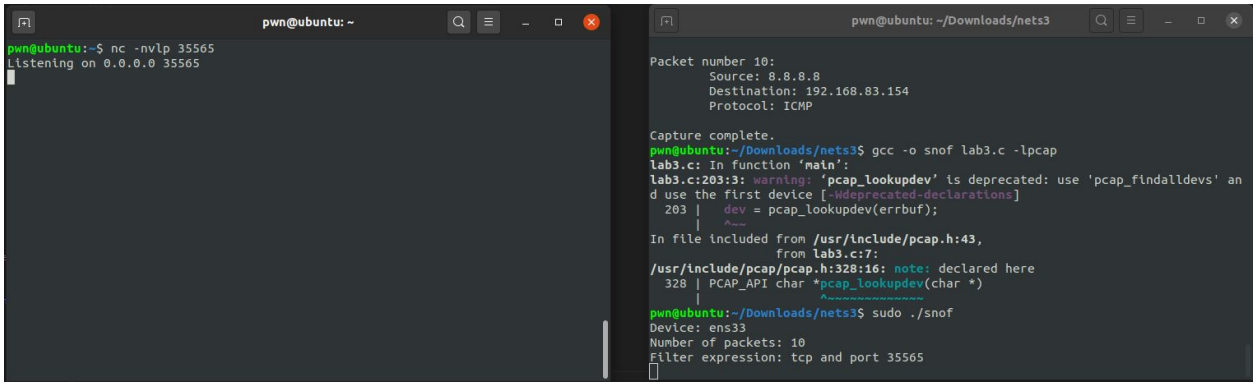
Sniffing TCP Packets in C

If you can see, on line 86 I changed the filter_exp[] to “tcp and port 35565” for this example.

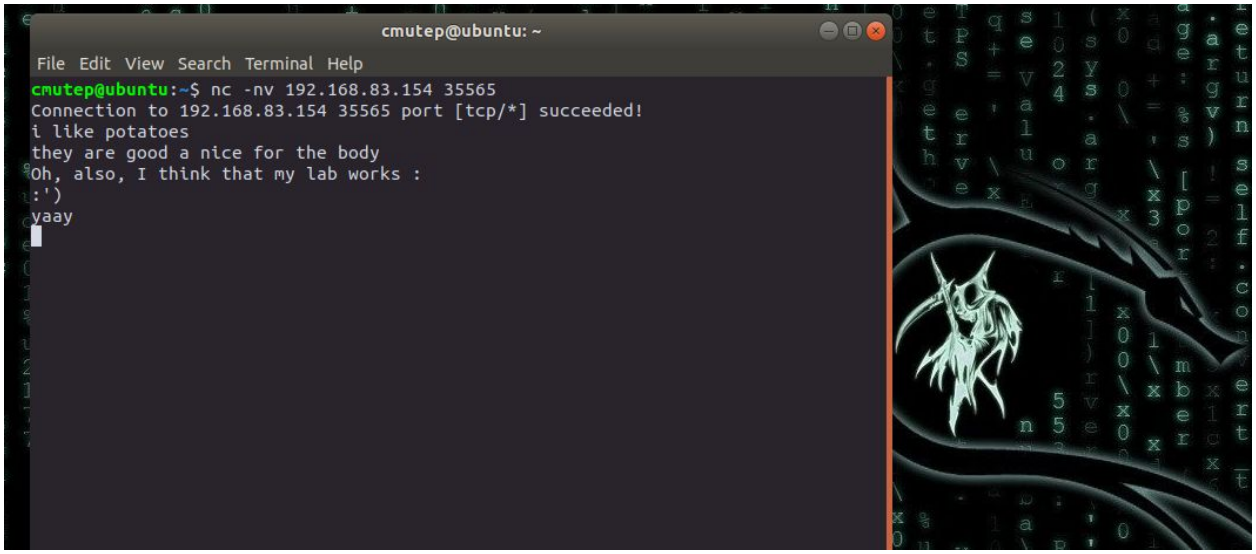


```
78
79  int main(int argc, char **argv)
80  {
81
82      char *dev = NULL;           /* capture device name */
83      char errbuf[PCAP_ERRBUF_SIZE];      /* error buffer */
84      pcap_t *handle;             /* packet capture handle */
85
86      char filter_exp[] = "tcp and port 35565";       /* filter expression [3] */
87      struct bpf_program fp;          /* compiled filter program (expression) */
88      bpf_u_int32 mask;           /* subnet mask */
89      bpf_u_int32 net;            /* ip */
90      int num_packets = 10;           /* number of packets to capture */
91
92      /* check for capture device name on command-line */
93      if (argc == 2) {
94          dev = argv[1];
95      }
96      else if (argc > 2) {
97          fprintf(stderr, "error: unrecognized command-line options\n\n");
98          //print_app_usage();
99          exit(EXIT_FAILURE);
100     }
101     else {
102         /* find a capture device if not specified on command-line */
103         dev = pcap_lookupdev(errbuf);
104         if (dev == NULL) {
105             fprintf(stderr, "Couldn't find default device: %s\n",
106                 errbuf);
107             exit(EXIT_FAILURE);
108         }
109     }
110
111     /* get network number and mask associated with capture device */
112     if (pcap_lookupnet(dev, &net, &mask, errbuf) == -1) {
113         fprintf(stderr, "Couldn't get netmask for device %s: %s\n",
114             dev, errbuf);
115         net = 0;
116         mask = 0;
117     }
118
119     /* print capture info */
120     printf("Device: %s\n", dev);
121     printf("Number of packets: %d\n", num_packets);
122     printf("Filter expression: %s\n", filter_exp);
123
124     /* open capture device */
125     handle = pcap_open_live(dev, SNAP_LEN, 1, 1000, errbuf);
126     if (handle == NULL) {
127         fprintf(stderr, "Couldn't open device %s: %s\n", dev, errbuf);
128         exit(EXIT_FAILURE);
129     }
130
131     /* make sure we're capturing on an Ethernet device [2] */
132     if (pcap_datalink(handle) != DLT_EN10MB) {
133         fprintf(stderr, "%s is not an Ethernet\n", dev);
134         exit(EXIT_FAILURE);
135     }
136
137     /* compile the filter expression */
138     if (pcap_compile(handle, &fp, filter_exp, 0, net) == -1) {
139         fprintf(stderr, "Couldn't parse filter %s: %s\n",
140             filter_exp, pcap_geterr(handle));
141         exit(EXIT_FAILURE);
142     }
143
144     /* apply the compiled filter */
145     if (pcap_setfilter(handle, &fp) == -1) {
146         fprintf(stderr, "Couldn't install filter %s: %s\n",
147             filter_exp, pcap_geterr(handle));
148         exit(EXIT_FAILURE);
149     }
150
151     /* now we can set our callback function */
152     pcap_loop(handle, num_packets, got_packet, NULL);
153
154     /* cleanup */
155     pcap_freecode(&fp);
156     pcap_close(handle);
157
158     printf("\nCapture complete.\n");
159
160 return 0;
161 }
162
```

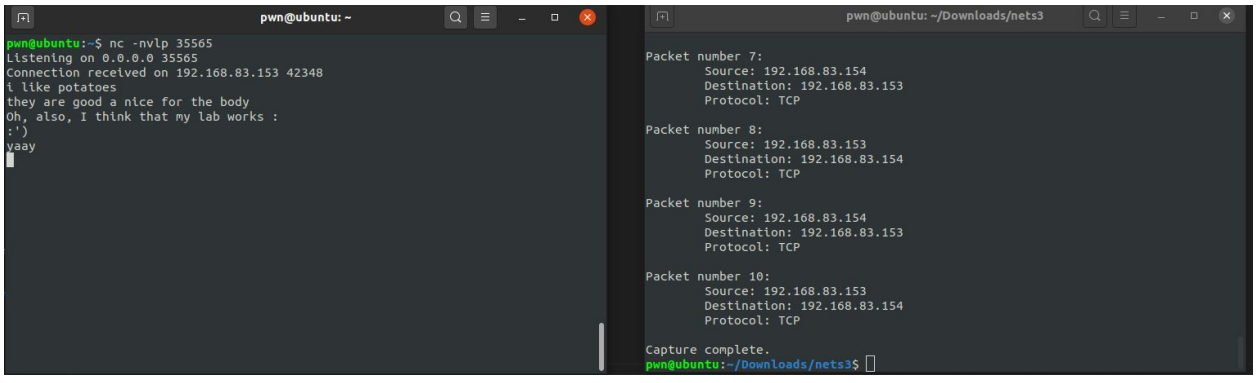
First VM Listening on port 35565



Second VM Connecting to port 35565

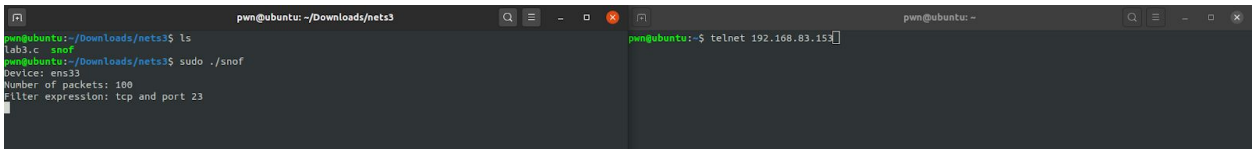


Final result - TCP Packets are sniffed.

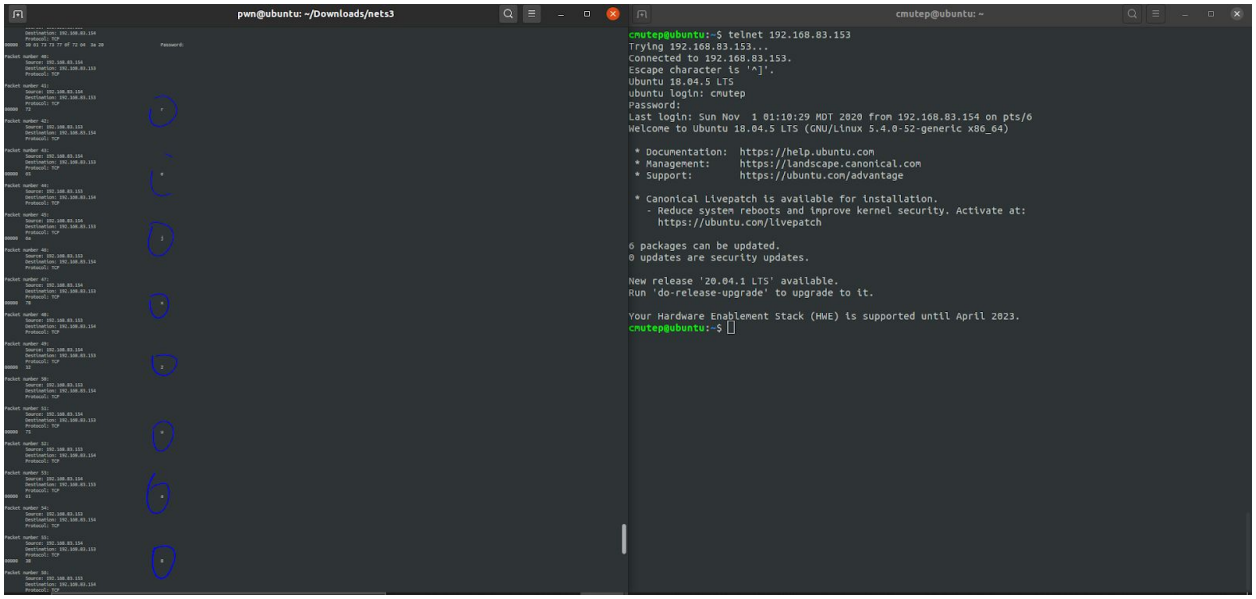


Telnet - Sniffing Passwords

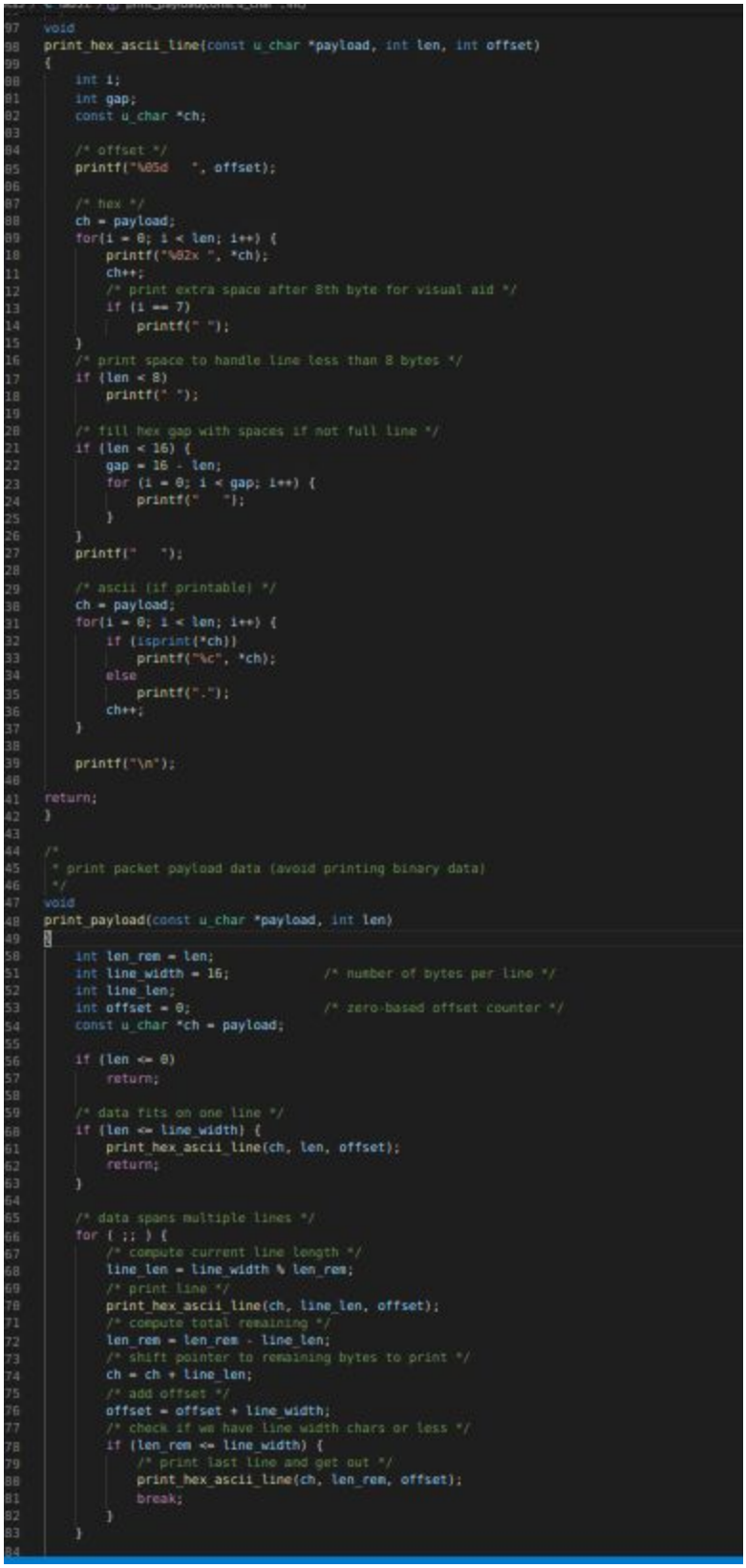
My sniffer is running on the left with different parameters this time. Now it is accepting 100 packets and the filter expression is set to tcp and port 23. The terminal on the right is connecting to the IP address of a second VM. I had to add a couple of functions for this part to work, so now the code is different.



Connection is established using telnet and my program sniffs the password. The blue circles indicate the characters from the password.



These were the two new functions I added to the code to show the passwords:



```
void
print_hex_ascii_line(const u_char *payload, int len, int offset)
{
    int i;
    int gap;
    const u_char *ch;

    /* offset */
    printf("%05d   ", offset);

    /* hex */
    ch = payload;
    for(i = 0; i < len; i++) {
        printf("%02x ", *ch);
        ch++;
        /* print extra space after 8th byte for visual aid */
        if (i == 7)
            printf(" ");
    }
    /* print space to handle line less than 8 bytes */
    if (len < 8)
        printf(" ");

    /* fill hex gap with spaces if not full line */
    if (len < 16) {
        gap = 16 - len;
        for (i = 0; i < gap; i++) {
            printf("   ");
        }
    }
    printf("   ");

    /* ascii (if printable) */
    ch = payload;
    for(i = 0; i < len; i++) {
        if (isprint(*ch))
            printf("%c", *ch);
        else
            printf(".");
        ch++;
    }

    printf("\n");

return;
}

/*
 * print packet payload data (avoid printing binary data)
 */
void
print_payload(const u_char *payload, int len)
{
    int len_rem = len;
    int line_width = 16;            /* number of bytes per line */
    int line_len;
    int offset = 0;                 /* zero-based offset counter */
    const u_char *ch = payload;

    if (len <= 0)
        return;

    /* data fits on one line */
    if (len <= line_width) {
        print_hex_ascii_line(ch, len, offset);
        return;
    }

    /* data spans multiple lines */
    for ( ;; ) {
        /* compute current line length */
        line_len = line_width % len_rem;
        /* print line */
        print_hex_ascii_line(ch, line_len, offset);
        /* compute total remaining */
        len_rem = len_rem - line_len;
        /* shift pointer to remaining bytes to print */
        ch = ch + line_len;
        /* add offset */
        offset = offset + line_width;
        /* check if we have line width chars or less */
        if (len_rem <= line_width) {
            /* print last line and get out */
            print_hex_ascii_line(ch, len_rem, offset);
            break;
        }
    }
```

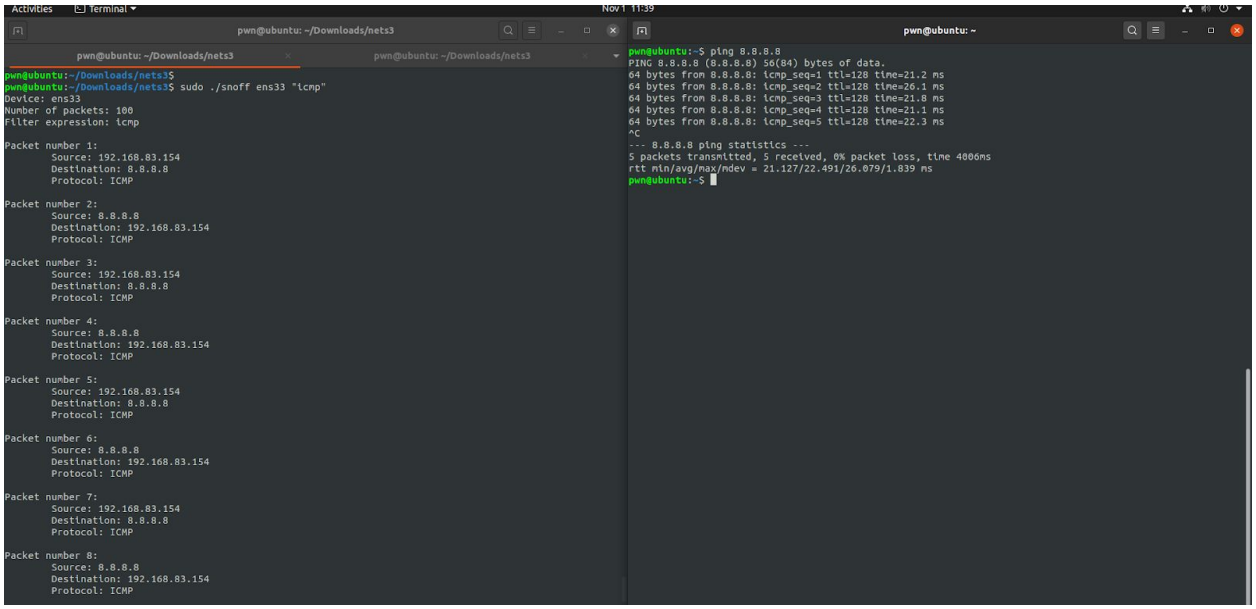
Okay, its a new day and I've made some changes to the code. The new way to run it is the following:

Compile the program --> gcc -o snoff lab3.c -lpcap

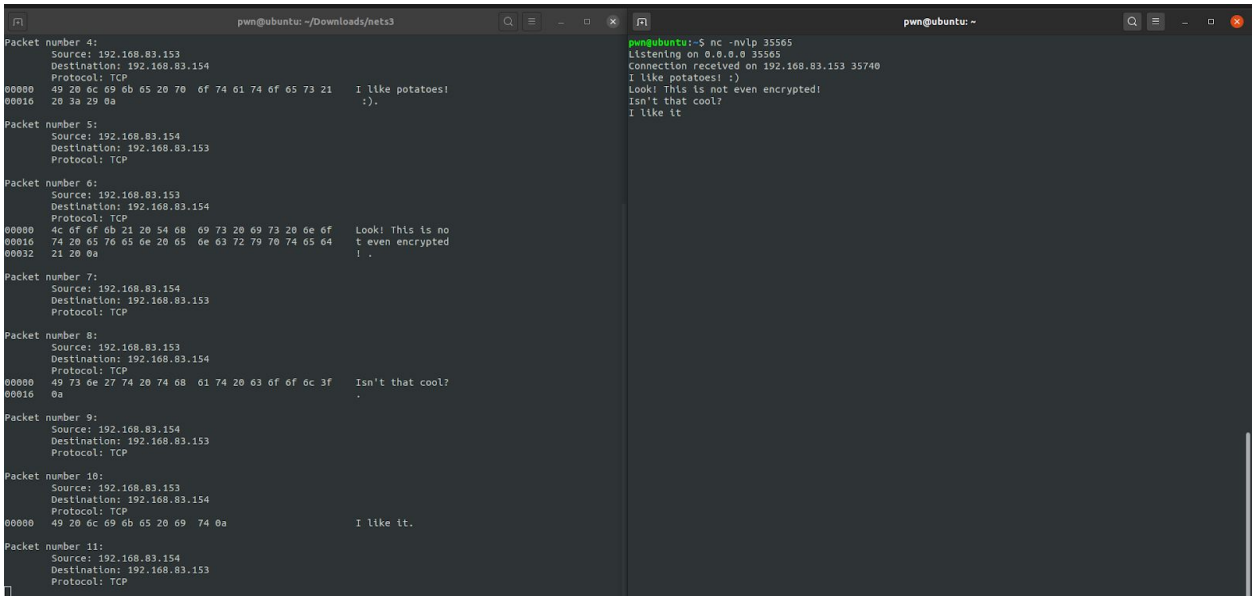
Run it in shell --> sudo ./snoff <device_name> <filter_expression>

Example run --> sudo ./snoff ens33 "icmp"
Example run --> sudo./snoff ens33 "tcp and port 35565"
Example run → sudo ./snoff ens33 "tcp and port 23"

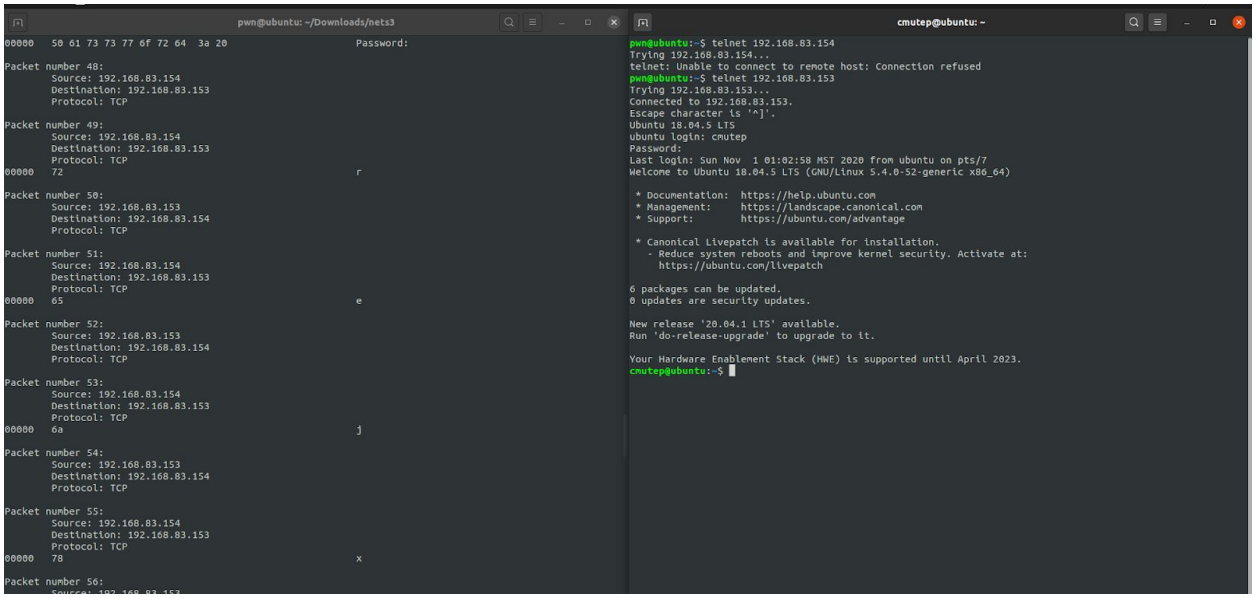
ICMP sniffing with finished code



TCP Sniffing with finished code



Telnet sniffing with finished code (pretty much the same as before)



References: https://www.tcpdump.org/pcap.html