

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.

```
Activities Terminal Oct 19 21:32
pwn@ubuntu: ~/Downloads$ sudo python3 skapy.py
###[ Ethernet ]###
dst      = ff:ff:ff:ff:ff:ff
src      = 00:50:56:c0:00:08
type     = ARP
###[ ARP ]###
hwtype   = 0x1
ptype    = IPv4
hwlen    = 6
plen     = 4
op       = who-has
hwsrc    = 00:50:56:c0:00:08
psrc     = 192.168.83.1
hwdst    = 00:00:00:00:00:00
pdst     = 192.168.83.2
###[ Padding ]###
load     = '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
###[ Ethernet ]###
dst      = ff:ff:ff:ff:ff:ff
src      = 00:50:56:c0:00:08
type     = ARP
###[ ARP ]###
hwtype   = 0x1
ptype    = IPv4
hwlen    = 6
plen     = 4
op       = who-has
hwsrc    = 00:50:56:c0:00:08
psrc     = 192.168.83.1
hwdst    = 00:00:00:00:00:00
pdst     = 192.168.83.2
###[ Padding ]###
load     = '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
###[ Ethernet ]###
dst      = 01:00:5e:7f:ff:fa
src      = 00:50:56:c0:00:08
type     = IPv4
###[ IP ]###
version  = 4
ihl      = 5
tos      = 0x0
len      = 201
id       = 34570
flags    =
frag     = 0
ttl      = 1
```

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



The image displays two terminal windows side-by-side, overlaid on a dark background featuring a stylized white rabbit head and the text "IRISH BUNNIES\$".

The left terminal window, titled "Scapy v2.4.4", shows the following commands and output:

```
>>> sniff(count=1, filter="icmp", lface="ens33")
<ctrl-c>: TCP:5 UDP:0 ICMP:5 Other:2>
>>> ...summary()
Ether / IP / ICMP 192.168.83.154 > 8.8.8.8 echo-request 0 / Raw
Ether / IP / ICMP 8.8.8.8 > 192.168.83.154 echo-reply 0 / Raw
Ether / IP / ICMP 192.168.83.154 > 8.8.8.8 echo-request 0 / Raw
Ether / IP / ICMP 8.8.8.8 > 192.168.83.154 echo-reply 0 / Raw
Ether / IP / ICMP 192.168.83.154 > 8.8.8.8 echo-request 0 / Raw
>>>
```

The right terminal window, titled "pwn@ubuntu: ~", shows the following commands and output:

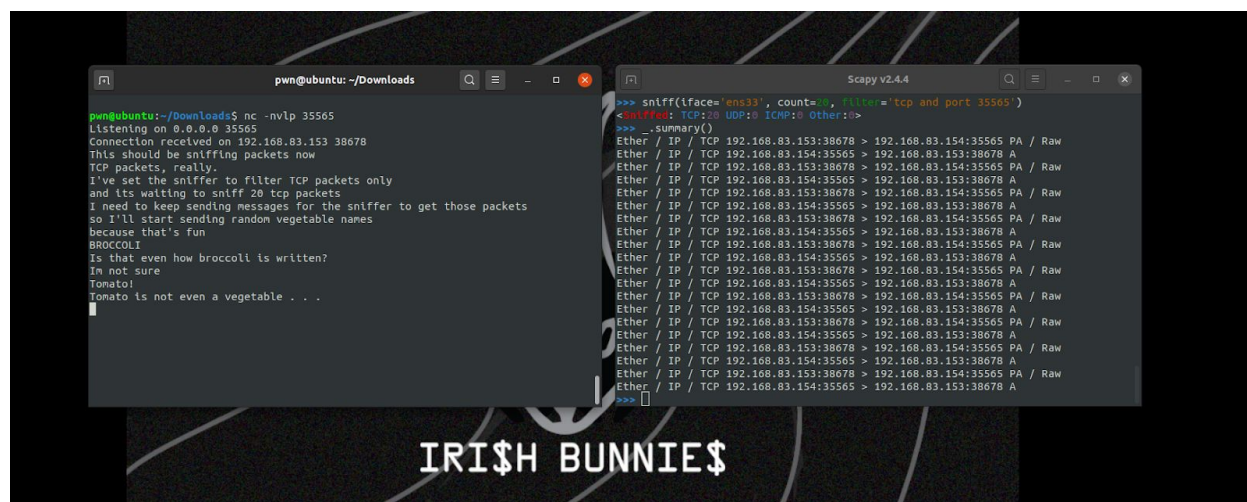
```
pwn@ubuntu:~$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data:
64 bytes from 8.8.8.8: icmp_seq=1 ttl=128 time=21.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=128 time=20.5 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=128 time=20.6 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=128 time=20.6 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=128 time=20.2 ms
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4641ms
rtt min/avg/max/mdev = 20.160/20.590/21.159/0.323 ms
pwn@ubuntu:~$
```

Using my Python Script:

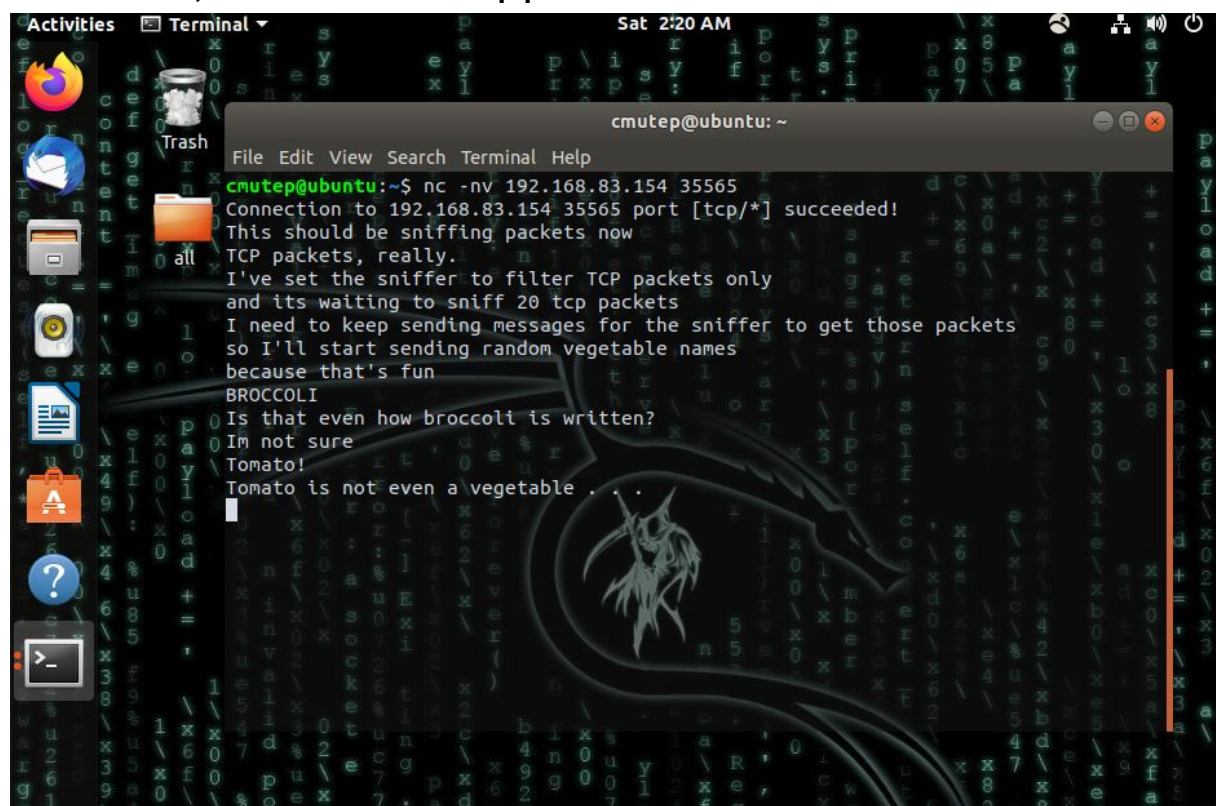
[illegible]

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.

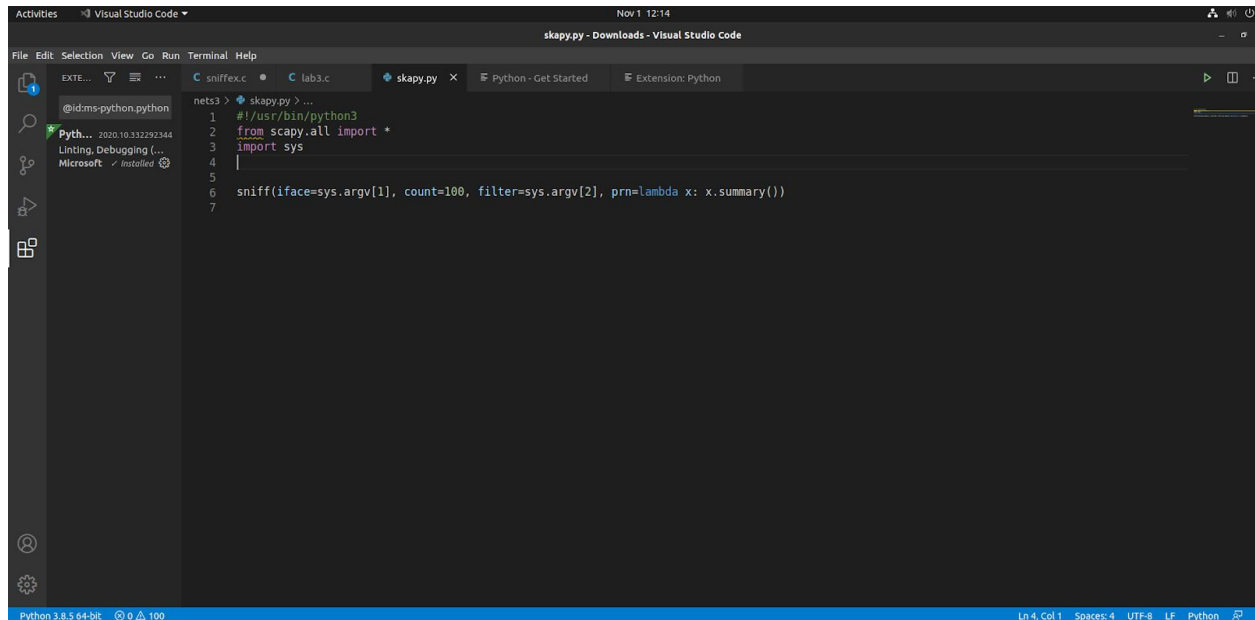


This is another VM that is using netcat to connect to the other listening machine. As they communicate, the sniffer detects tcp packets.



Filtering TCP Packets using my Python Program

Picture of the code

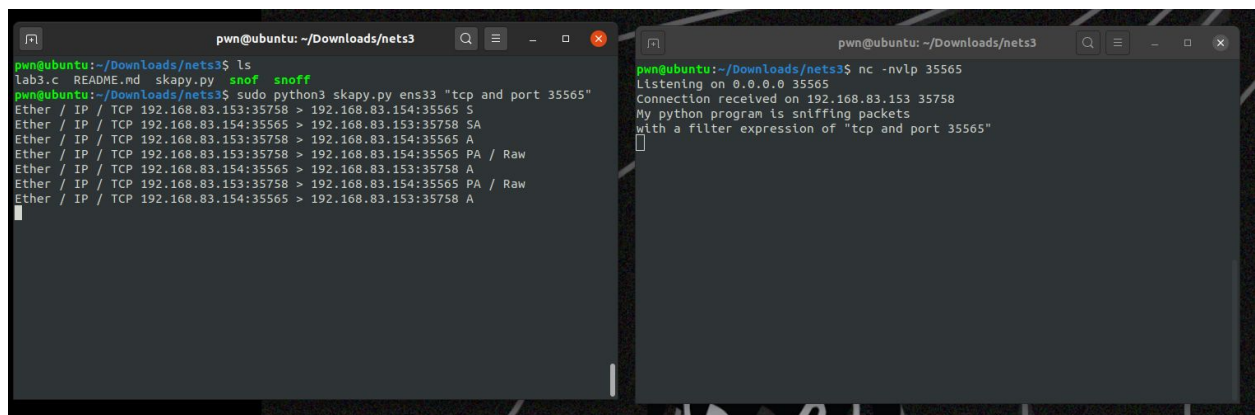


The screenshot shows the Visual Studio Code editor with the file `skapy.py` open. The code is as follows:

```
1  #!/usr/bin/python3
2  from scapy.all import *
3  import sys
4
5
6  sniff(iface=sys.argv[1], count=100, filter=sys.argv[2], prn=lambda x: x.summary())
7
```

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.



The image shows two terminal windows side-by-side. The left window shows the execution of the `skapy.py` program, and the right window shows a netcat listener.

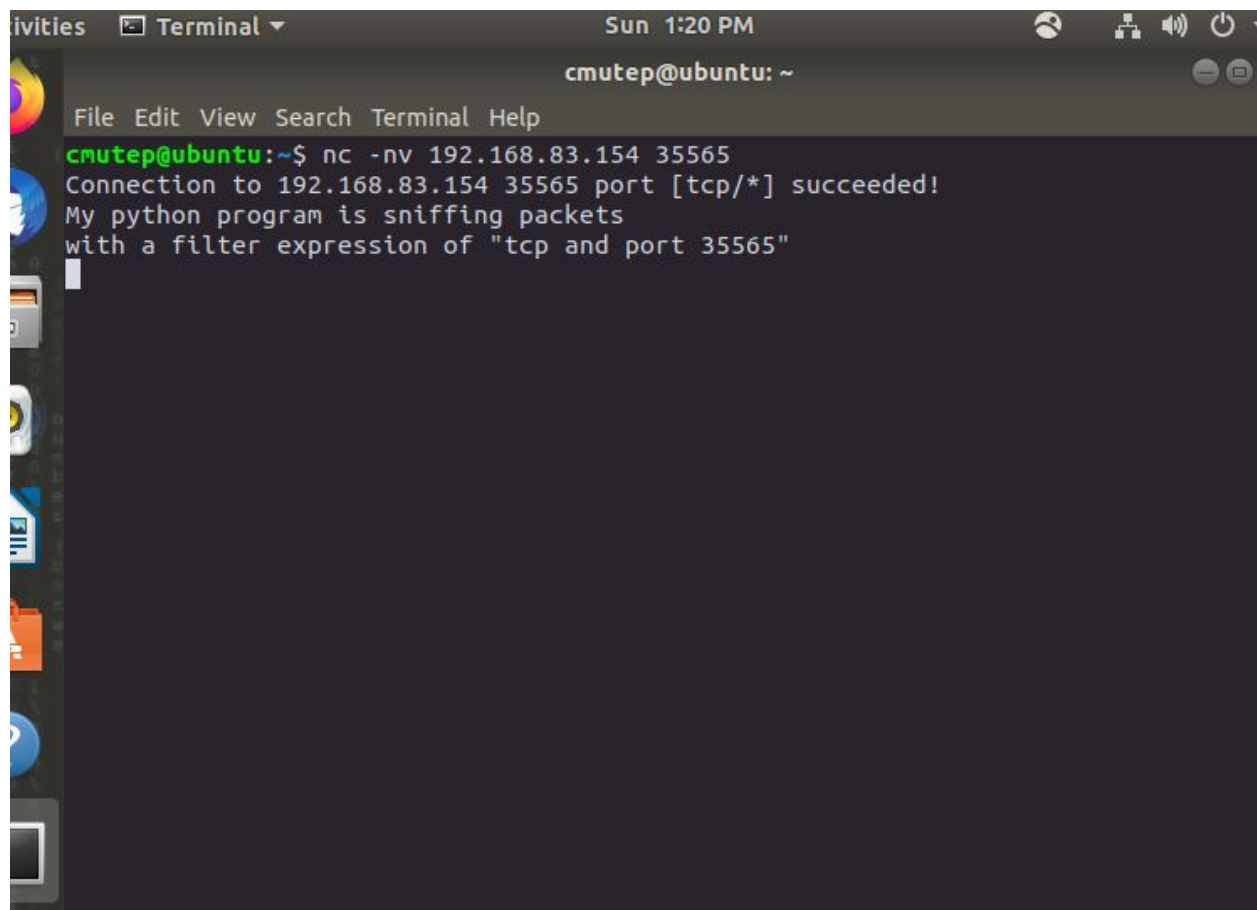
Left Terminal Window:

```
pwn@ubuntu: ~/Downloads/nets3
pwn@ubuntu:~/Downloads/nets3$ ls
lab3.c  README.md  skapy.py  snof  snoff
pwn@ubuntu:~/Downloads/nets3$ sudo python3 skapy.py ens3 "tcp and port 35565"
Ether / IP / TCP 192.168.83.153:35758 > 192.168.83.154:35565 S
Ether / IP / TCP 192.168.83.154:35565 > 192.168.83.153:35758 SA
Ether / IP / TCP 192.168.83.153:35758 > 192.168.83.154:35565 A
Ether / IP / TCP 192.168.83.153:35758 > 192.168.83.154:35565 PA / Raw
Ether / IP / TCP 192.168.83.154:35565 > 192.168.83.153:35758 A
Ether / IP / TCP 192.168.83.154:35565 > 192.168.83.153:35758 A
```

Right Terminal Window:

```
pwn@ubuntu:~/Downloads/nets3$ nc -nvlp 35565
Listening on 0.0.0.0 35565
Connection received on 192.168.83.153 35758
My python program is sniffing packets
with a filter expression of "tcp and port 35565"
```

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



```
cmutep@ubuntu: ~  
File Edit View Search Terminal Help  
cmutep@ubuntu:~$ nc -nv 192.168.83.154 35565  
Connection to 192.168.83.154 35565 port [tcp/*] succeeded!  
My python program is sniffing packets  
with a filter expression of "tcp and port 35565"
```


2) Sniffing ICMP packets in C - Code Picture

```
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_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, every 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. . .

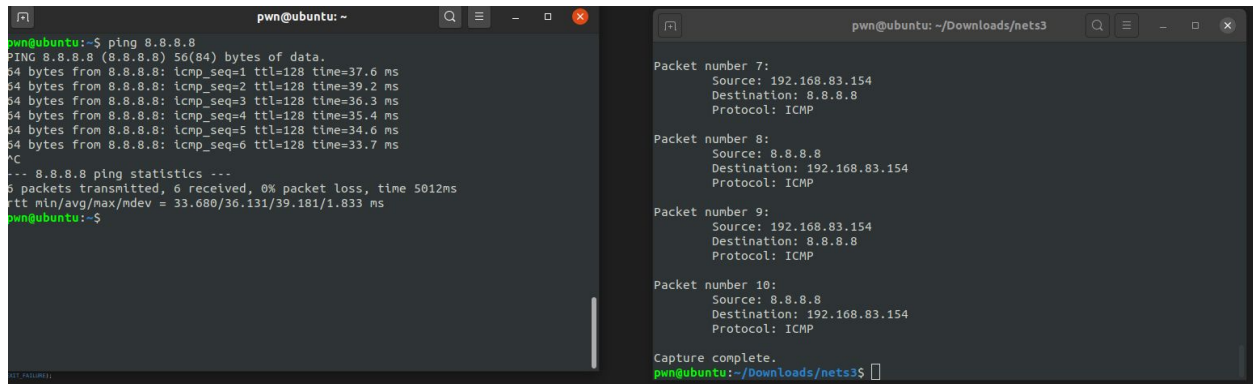
```
pwn@ubuntu:~/Downloads/nets3$ ./snof
Device: ens33
Number of packets: 100
Filter expression: tcp and port 23
Couldn't open device ens33: ens33: You don't have permission to capture on that device (socket: Operation
not permitted)
pwn@ubuntu:~/Downloads/nets3$
```

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.



The image shows two terminal windows side-by-side. The left window displays the output of a ping command to 8.8.8.8, showing 6 successful packets with varying round-trip times. The right window shows the details of the first four packets captured by a network sniffer, including packet numbers, source and destination IP addresses, and the protocol (ICMP).

```
pwn@ubuntu: ~  
pwn@ubuntu:~$ 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=128 time=37.6 ms  
64 bytes from 8.8.8.8: icmp_seq=2 ttl=128 time=39.2 ms  
64 bytes from 8.8.8.8: icmp_seq=3 ttl=128 time=36.3 ms  
64 bytes from 8.8.8.8: icmp_seq=4 ttl=128 time=35.4 ms  
64 bytes from 8.8.8.8: icmp_seq=5 ttl=128 time=34.6 ms  
64 bytes from 8.8.8.8: icmp_seq=6 ttl=128 time=33.7 ms  
^C  
--- 8.8.8.8 ping statistics ---  
6 packets transmitted, 6 received, 0% packet loss, time 5012ms  
rtt min/avg/max/mdev = 33.680/36.131/39.181/1.833 ms  
pwn@ubuntu:~$  
  
pwn@ubuntu: ~/Downloads/nets3  
Packet number 7:  
Source: 192.168.83.154  
Destination: 8.8.8.8  
Protocol: ICMP  
  
Packet number 8:  
Source: 8.8.8.8  
Destination: 192.168.83.154  
Protocol: ICMP  
  
Packet number 9:  
Source: 192.168.83.154  
Destination: 8.8.8.8  
Protocol: ICMP  
  
Packet number 10:  
Source: 8.8.8.8  
Destination: 192.168.83.154  
Protocol: ICMP  
  
Capture complete.  
pwn@ubuntu:~/Downloads/nets3$
```


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

First VM Listening on port 35565

```
pwn@ubuntu: ~  
pwn@ubuntu:~$ nc -nvlp 35565  
Listening on 0.0.0.0 35565  
  
pwn@ubuntu:~/Downloads/nets3$ gcc -o snof lab3.c -lpcap  
lab3.c: In function 'main':  
lab3.c:203:3: warning: 'pcap_lookupdev' is deprecated: use 'pcap_findalldevs' and use the first device [-Wdeprecated-declarations]  
    203 |     dev = pcap_lookupdev(errbuf);  
        |           ^  
In file included from /usr/include/pcap.h:43,  
                 from lab3.c:7:  
/usr/include/pcap/pcap.h:328:16: note: declared here  
    328 |     PCAP_API char *pcap_lookupdev(char *)  
        |                   ^  
pwn@ubuntu:~/Downloads/nets3$ sudo ./snof  
Device: ens33  
Number of packets: 10  
Filter expression: tcp and port 35565
```

Second VM Connecting to port 35565

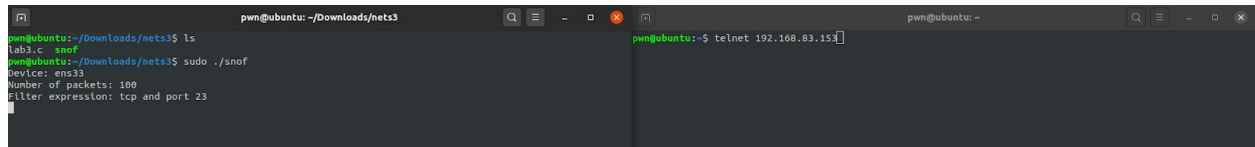
```
cmutept@ubuntu: ~  
File Edit View Search Terminal Help  
cmutept@ubuntu:~$ nc -nv 192.168.83.154 35565  
Connection to 192.168.83.154 35565 port [tcp/*] succeeded!  
i like potatoes  
they are good a nice for the body  
Oh, also, I think that my lab works :  
:~)  
yaay
```

Final result - TCP Packets are sniffed.

```
pwn@ubuntu: ~  
pwn@ubuntu:~$ nc -nvlp 35565  
Listening on 0.0.0.0 35565  
Connection received on 192.168.83.153 42348  
i like potatoes  
they are good a nice for the body  
Oh, also, I think that my lab works :  
:~)  
yaay  
  
pwn@ubuntu:~/Downloads/nets3$  
Packet number 7:  
Source: 192.168.83.154  
Destination: 192.168.83.153  
Protocol: TCP  
  
Packet number 8:  
Source: 192.168.83.153  
Destination: 192.168.83.154  
Protocol: TCP  
  
Packet number 9:  
Source: 192.168.83.154  
Destination: 192.168.83.153  
Protocol: TCP  
  
Packet number 10:  
Source: 192.168.83.153  
Destination: 192.168.83.154  
Protocol: TCP  
  
Capture complete.  
pwn@ubuntu:~/Downloads/nets3$
```

Telnet - Sniffing Passwords

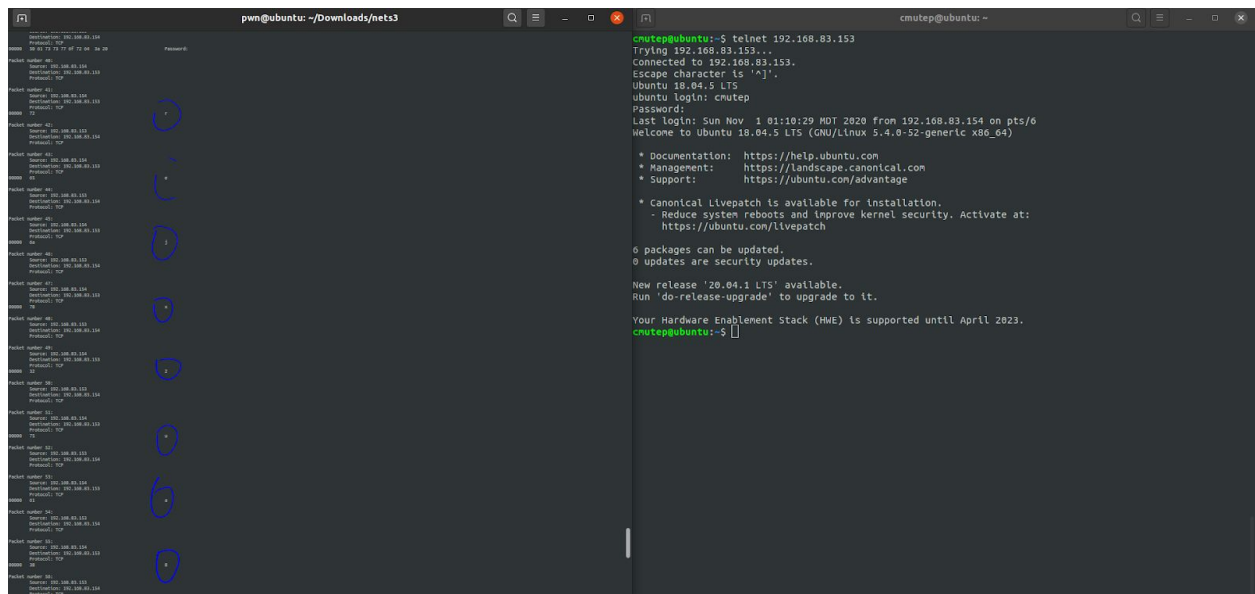
My sniffer is running on the left with different parameters this time. Now it is accepting 100 packets and the filter expression is set to tcp and port 23. The terminal on the right is connecting to the IP address of a second VM. I had to add a couple of functions for this part to work, so now the code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



```
pwn@ubuntu: ~/Downloads/nets3
pwn@ubuntu:~/Downloads/nets3$ ls
lab3.c  snort
pwn@ubuntu:~/Downloads/nets3$ sudo ./snort
Device: ens33
Number of packets: 100
Filter expression: tcp and port 23

pwn@ubuntu:~$ telnet 192.168.83.153
```

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



```
pwn@ubuntu: ~/Downloads/nets3
cmtetp@ubuntu:~$ telnet 192.168.83.153
Trying 192.168.83.153...
Connected to 192.168.83.153.
Escape character is '^J'.
Ubuntu 18.04.5 LTS
ubuntu login: cmtetp
Password:
Last login: Sun Nov  1 01:18:29 MDT 2020 from 192.168.83.154 on pts/6
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-52-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

6 packages can be updated.
0 updates are security updates.

New release '20.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Your Hardware Enablement Stack (HWE) is supported until April 2023.
cmtetp@ubuntu:~$
```

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

```
97 void
98 print_hex_ascii_line(const u_char *payload, int len, int offset)
99 {
100     int i;
101     int gap;
102     const u_char *ch;
103
104     /* offset */
105     printf("%05d  ", offset);
106
107     /* hex */
108     ch = payload;
109     for(i = 0; i < len; i++) {
110         printf("%02x ", *ch);
111         ch++;
112         /* print extra space after 8th byte for visual aid */
113         if (i == 7)
114             printf(" ");
115     }
116     /* print space to handle line less than 8 bytes */
117     if (len < 8)
118         printf(" ");
119
120     /* fill hex gap with spaces if not full line */
121     if (len < 16) {
122         gap = 16 - len;
123         for (i = 0; i < gap; i++) {
124             printf(" ");
125         }
126     }
127     printf(" ");
128
129     /* ascii (if printable) */
130     ch = payload;
131     for(i = 0; i < len; i++) {
132         if (isprint(*ch))
133             printf("%c", *ch);
134         else
135             printf(".");
136         ch++;
137     }
138
139     printf("\n");
140
141     return;
142 }
143
144 /*
145  * print packet payload data (avoid printing binary data)
146  */
147 void
148 print_payload(const u_char *payload, int len)
149 {
150     int len_rem = len;
151     int line_width = 16;          /* number of bytes per line */
152     int line_len;
153     int offset = 0;              /* zero-based offset counter */
154     const u_char *ch = payload;
155
156     if (len <= 0)
157         return;
158
159     /* data fits on one line */
160     if (len <= line_width) {
161         print_hex_ascii_line(ch, len, offset);
162         return;
163     }
164
165     /* data spans multiple lines */
166     for ( ;; ) {
167         /* compute current line length */
168         line_len = line_width % len_rem;
169         /* print line */
170         print_hex_ascii_line(ch, line_len, offset);
171         /* compute total remaining */
172         len_rem = len_rem - line_len;
173         /* shift pointer to remaining bytes to print */
174         ch = ch + line_len;
175         /* add offset */
176         offset = offset + line_width;
177         /* check if we have line width chars or less */
178         if (len_rem <= line_width) {
179             /* print last line and get out */
180             print_hex_ascii_line(ch, len_rem, offset);
181             break;
182         }
183     }
184 }
```

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`

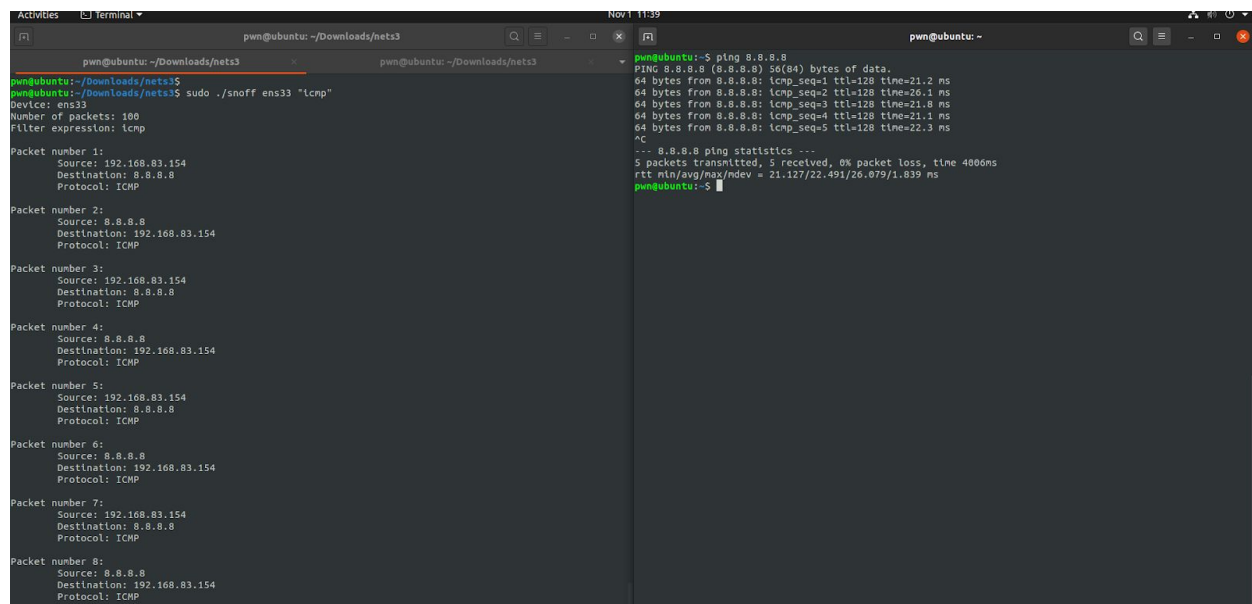
Run it in shell --> `sudo ./snoff <device_name> <filter_expression>`

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



The screenshot shows a terminal window with two panes. The left pane displays the output of the `snoff` program, which has been run with the command `sudo ./snoff ens33 "icmp"`. It shows 100 packets captured on the `ens33` interface, with a filter expression of `icmp`. The output lists 8 packets with their source, destination, and protocol details. The right pane shows the output of a `ping` command, displaying the results of a ping to `8.8.8.8`, including the number of bytes of data, the sequence number, the time to live (TTL), and the time taken for the packet to reach the destination.

```
pwn@ubuntu: ~/Downloads/nets3
pwn@ubuntu:~/Downloads/nets3$ sudo ./snoff ens33 "icmp"
Device: ens33
Number of packets: 100
Filter expression: icmp

Packet number 1:
  Source: 192.168.83.154
  Destination: 8.8.8.8
  Protocol: ICMP

Packet number 2:
  Source: 8.8.8.8
  Destination: 192.168.83.154
  Protocol: ICMP

Packet number 3:
  Source: 192.168.83.154
  Destination: 8.8.8.8
  Protocol: ICMP

Packet number 4:
  Source: 8.8.8.8
  Destination: 192.168.83.154
  Protocol: ICMP

Packet number 5:
  Source: 192.168.83.154
  Destination: 8.8.8.8
  Protocol: ICMP

Packet number 6:
  Source: 8.8.8.8
  Destination: 192.168.83.154
  Protocol: ICMP

Packet number 7:
  Source: 192.168.83.154
  Destination: 8.8.8.8
  Protocol: ICMP

Packet number 8:
  Source: 8.8.8.8
  Destination: 192.168.83.154
  Protocol: ICMP

pwn@ubuntu:~/Downloads/nets3$ 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=128 time=21.2 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=128 time=20.1 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=128 time=21.8 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=128 time=21.1 ms
64 bytes from 8.8.8.8: icmp_seq=5 ttl=128 time=22.3 ms
^C
--- 8.8.8.8 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 400ms
rtt min/avg/max/mdev = 21.127/22.491/26.079/1.839 ms
pwn@ubuntu:~/Downloads/nets3$
```


TCP Sniffing with finished code

```
pwn@ubuntu: ~/Downloads/nets3
Packet number 4:
  Source: 192.168.83.153
  Destination: 192.168.83.154
  Protocol: TCP
00000 49 20 6c 69 6b 65 20 70 6f 74 61 74 6f 65 73 21  I like potatoes!
00016 20 3a 29 0a                                     :).

Packet number 5:
  Source: 192.168.83.154
  Destination: 192.168.83.153
  Protocol: TCP

Packet number 6:
  Source: 192.168.83.153
  Destination: 192.168.83.154
  Protocol: TCP
00000 4c 6f 6f 6b 21 20 54 68 69 73 20 69 73 20 6e 6f  Look! This is no
00016 74 20 65 76 65 6e 20 65 6e 63 72 79 70 74 65 64  t even encrypted
00032 21 20 0a                                         ! .

Packet number 7:
  Source: 192.168.83.154
  Destination: 192.168.83.153
  Protocol: TCP

Packet number 8:
  Source: 192.168.83.153
  Destination: 192.168.83.154
  Protocol: TCP
00000 49 73 6e 27 74 20 74 68 61 74 20 63 6f 6c 6c 3f  Isn't that cool?
00016 0a                                               .

Packet number 9:
  Source: 192.168.83.154
  Destination: 192.168.83.153
  Protocol: TCP

Packet number 10:
  Source: 192.168.83.153
  Destination: 192.168.83.154
  Protocol: TCP
00000 49 20 6c 69 6b 65 20 69 74 0a                 I like it.

Packet number 11:
  Source: 192.168.83.154
  Destination: 192.168.83.153
  Protocol: TCP

pwn@ubuntu:~$ nc -nvlp 35565
Listening on 0.0.0.0 35565
Connection received on 192.168.83.153 35740
I like potatoes! :)
Look! This is not even encrypted!
Isn't that cool?
I like it
```

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

```
pwn@ubuntu: ~/Downloads/nets3
00000 50 61 73 73 77 6f 72 64 3a 20                Password:

Packet number 48:
  Source: 192.168.83.154
  Destination: 192.168.83.153
  Protocol: TCP

Packet number 49:
  Source: 192.168.83.154
  Destination: 192.168.83.153
  Protocol: TCP
00000 72                                               r

Packet number 50:
  Source: 192.168.83.153
  Destination: 192.168.83.154
  Protocol: TCP

Packet number 51:
  Source: 192.168.83.154
  Destination: 192.168.83.153
  Protocol: TCP
00000 65                                               e

Packet number 52:
  Source: 192.168.83.153
  Destination: 192.168.83.154
  Protocol: TCP

Packet number 53:
  Source: 192.168.83.154
  Destination: 192.168.83.153
  Protocol: TCP
00000 6a                                               j

Packet number 54:
  Source: 192.168.83.153
  Destination: 192.168.83.154
  Protocol: TCP

Packet number 55:
  Source: 192.168.83.154
  Destination: 192.168.83.153
  Protocol: TCP
00000 78                                               x

Packet number 56:
  Source: 192.168.83.153
  Destination: 192.168.83.154
  Protocol: TCP

cmute@ubuntu:~$ telnet 192.168.83.154
Trying 192.168.83.154...
telnet: Unable to connect to remote host: Connection refused
pwn@ubuntu:~$ telnet 192.168.83.153
Trying 192.168.83.153...
Connected to 192.168.83.153.
Escape character is '^['.
Ubuntu 18.04.5 LTS
Ubuntu login: cmute
Password:
Last login: Sun Nov 1 01:02:58 MST 2020 from ubuntu on pts/7
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-52-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

6 packages can be updated.
0 updates are security updates.

New release '20.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Your Hardware Enablement Stack (HWE) is supported until April 2023.
cmute@ubuntu:~$
```

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