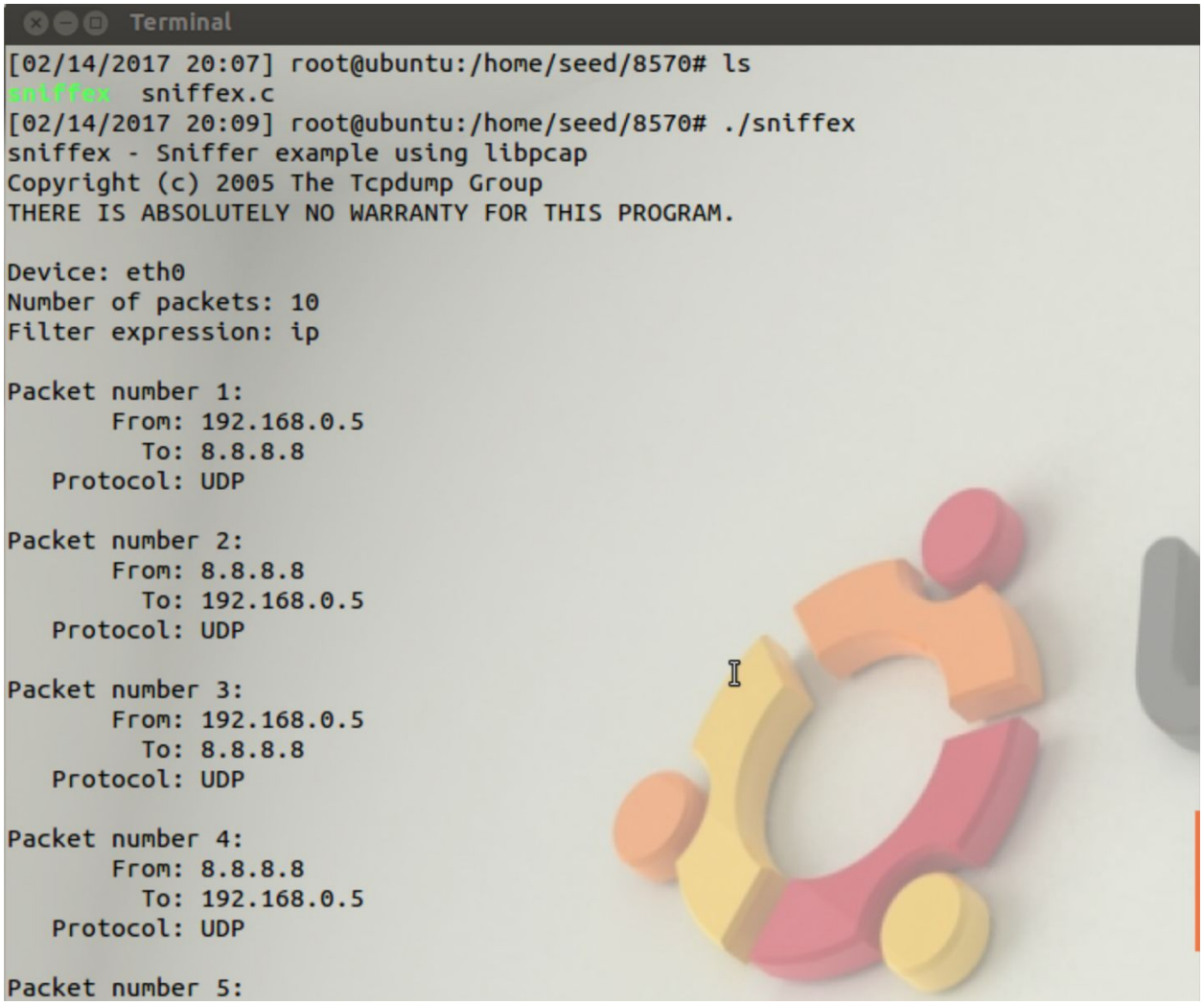
Project 1

Yang Cao
cao9@g.clemson.edu

Task 1: Writing Packet Sniffing Program

Task 1.a: Understanding sniffex.



The sniffex.c program has been downloaded and compiled. It ran successfully and produced correct results which were expected.

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 detailed explanation like the one in the tutorial.

1) Setting the device which we want to sniff on

Method1: The user defines the device and uses the string "dev" to represent the name of an interface which we are going to sniff on.

Method2: Pcap sets the device by itself. If this command fails, it will populate the string with an explanation of the error and save the error message in errbuf.

```
int main(int argc, char *argv[])
{
        char *dev, errbuf[PCAP_ERRBUF_SIZE];

        dev = pcap_lookupdev(errbuf);
        if (dev == NULL) {
                fprintf(stderr, "Couldn't find default device: %s\n", errbuf);
                return(2);
        }
        printf("Device: %s\n", dev);
        return(0);
}
```

2) Opening the device for sniffing

We use a function called pcap_open_live() to return us session handler.

```
pcap_t *pcap_open_live(char *device, int snaplen, int promisc, int to_ms,
    char *ebuf)
```

The char* device means the device which we setted in last step. Snaplen is an integer which means the maximum number of bytes to be captured by pcap; As for promisc, “true” refers to promiscuous mode and “false” refers to non-promiscuous mode. To_ms is the read timeout in milliseconds, we should use a non-zero timeout in this process. We can use ebuf to save error messages when it fails to open the device for sniffing.

3) Filtering traffic

When we finished previous steps, we use pcap_compile() to compile the program before applying our filter.

```
int pcap_compile(pcap_t *p, struct bpf_program *fp, char *str, int optimize,
    bpf_u_int32 netmask)
```

The pacp_t *p means the session handle. The struct bpf_program *fp is a reference to the position we will save the compiled filter. The string format of char* str is regular. The int optimize is an integer that determines whether the expression should be optimized or not. Bpf_u_int32 netmask determines the network mask of the network the filter applies to. This function returns -1 when it fails, in other cases it means success.

We can apply it after the expression has been compiled.

```
int pcap_setfilter(pcap_t *p, struct bpf_program *fp)
```

The pcap_t *p is our session handler, and the struct bpf_program *fp is a reference to the compiled version of the expression.

4) The actual sniffing

We can capture a single packet at a time by using pcap_next().

```
u_char *pcap_next(pcap_t *p, struct pcap_pkthdr *h)
```

The pcap_t *p means the session handler, and the struct pcap_pkthdr *h is a pointer to a structure which contains general information of the packet. This function returns a u_char

pointer to the packet. We can also enter a loop which waits for n number of packets to be sniffed before being done through using pcap_loop().

```
int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user)
```

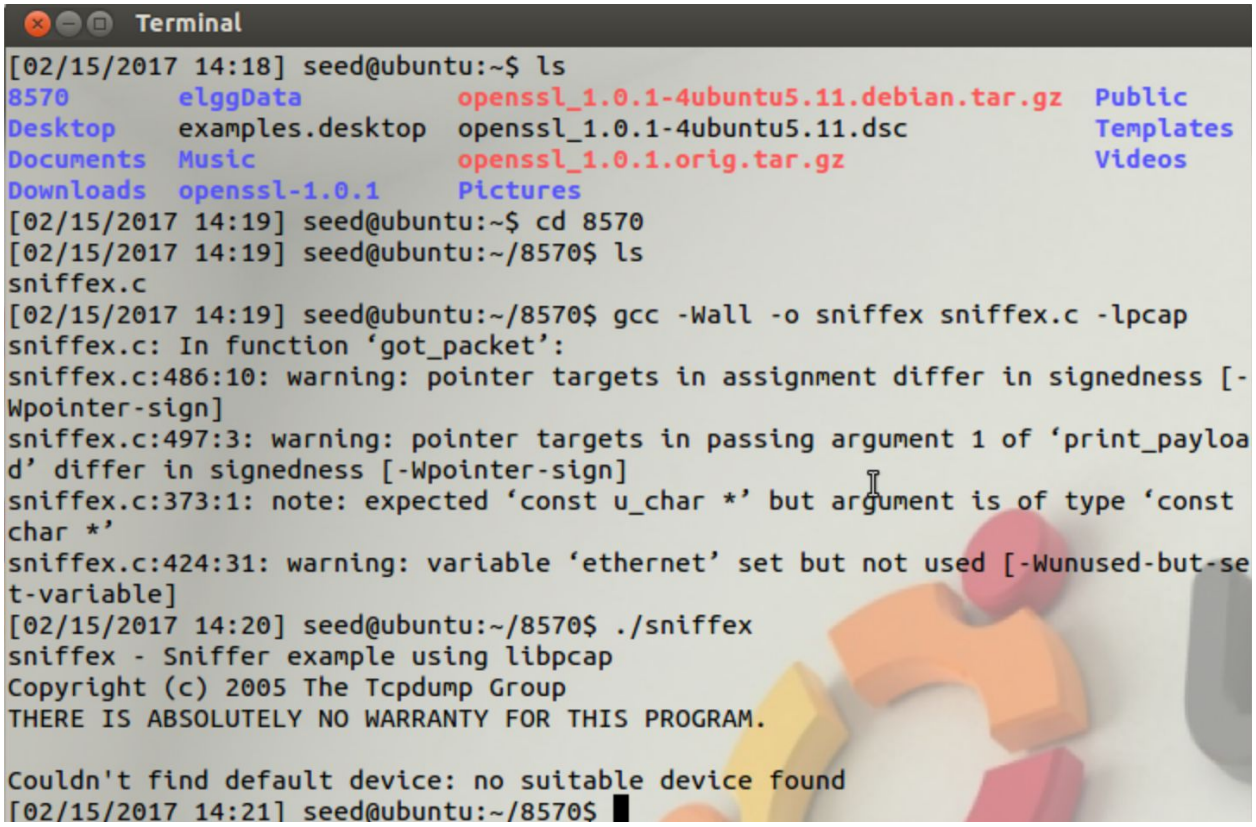
5) Closing the sniffer session

```
/* And close the session */
pcap_close(handle);
```

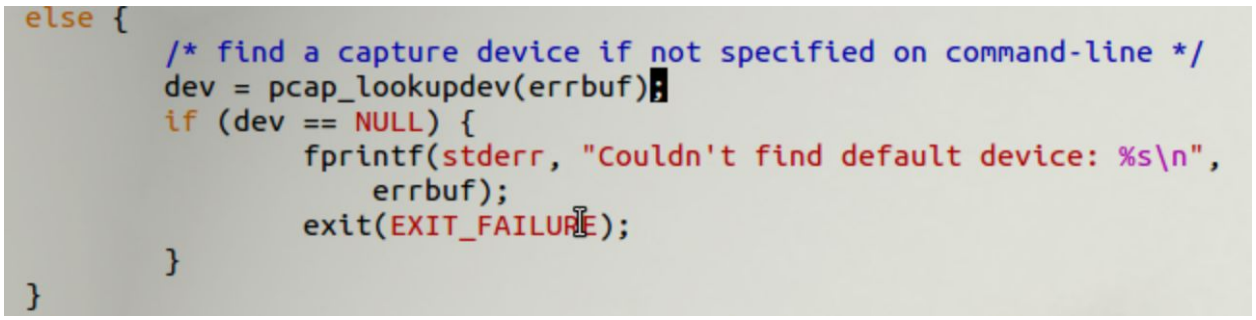
We use this function to close the sniffer session after finishing our assignments.

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

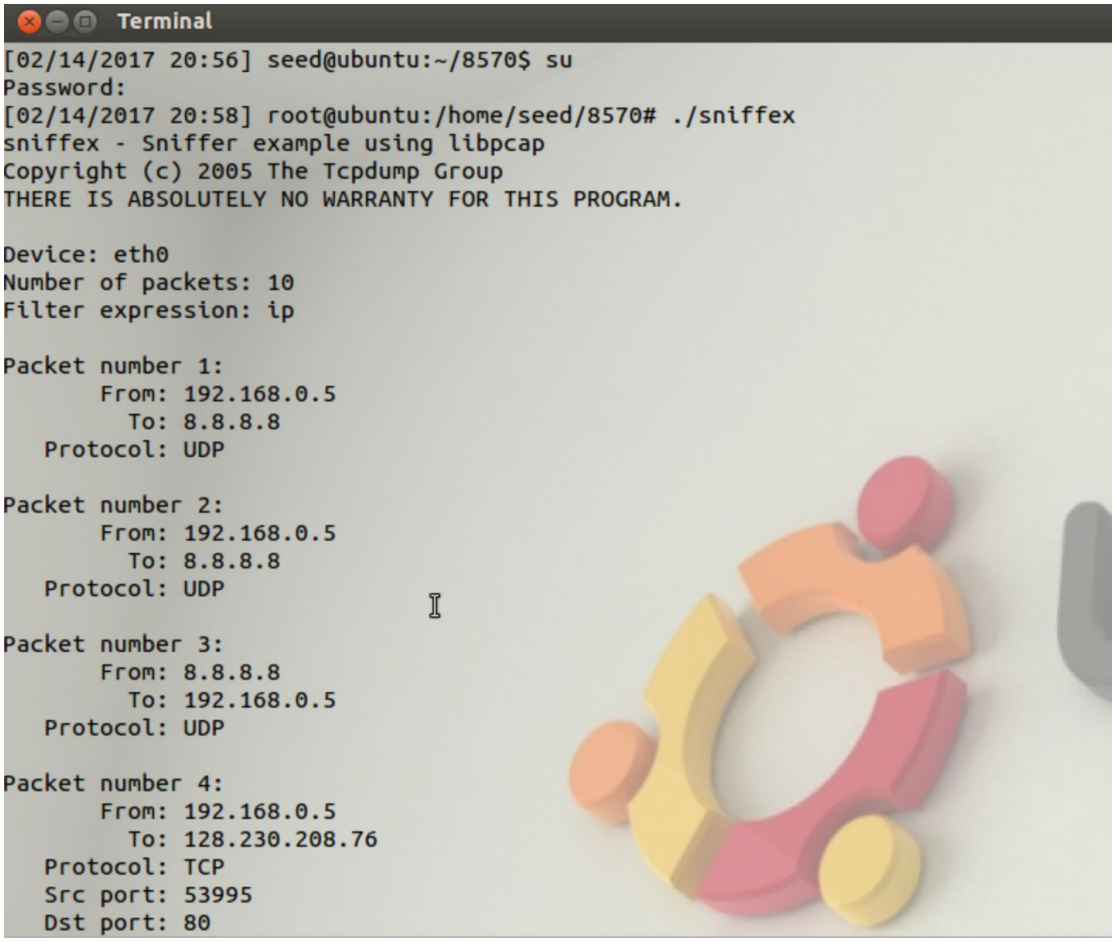
We can compile the sniffex.c program, but it can not run successfully without the root privilege. An error message displays that no suitable device is found.



It fails in the following code because pcap_lookupdev() returns null without the root privilege.

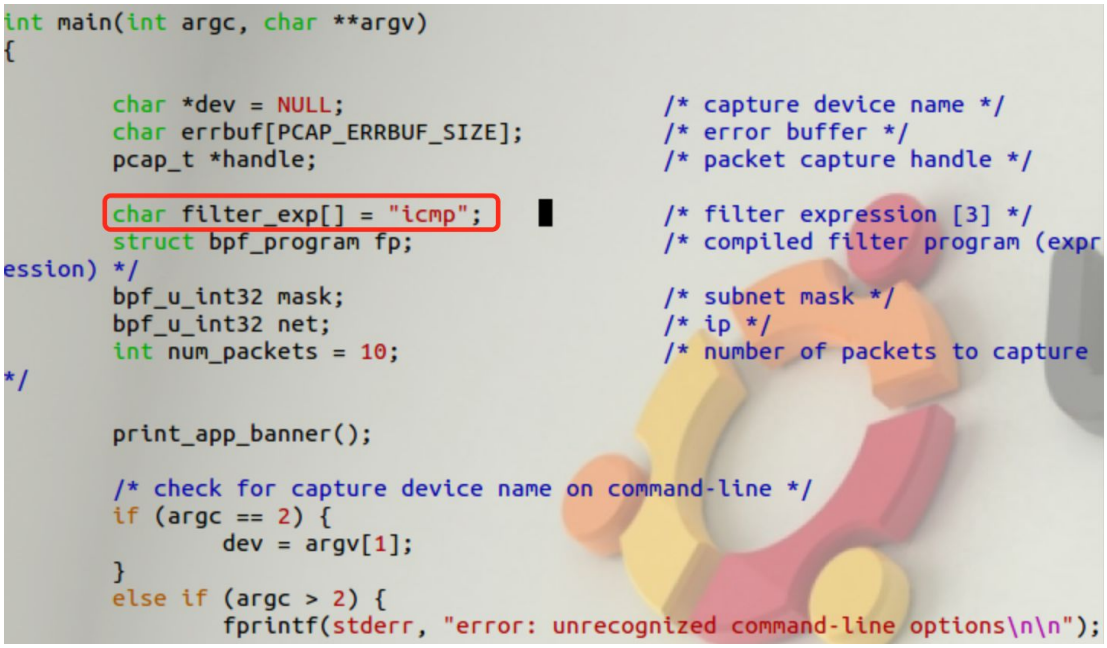


It can run successfully if we gain the root privilege.

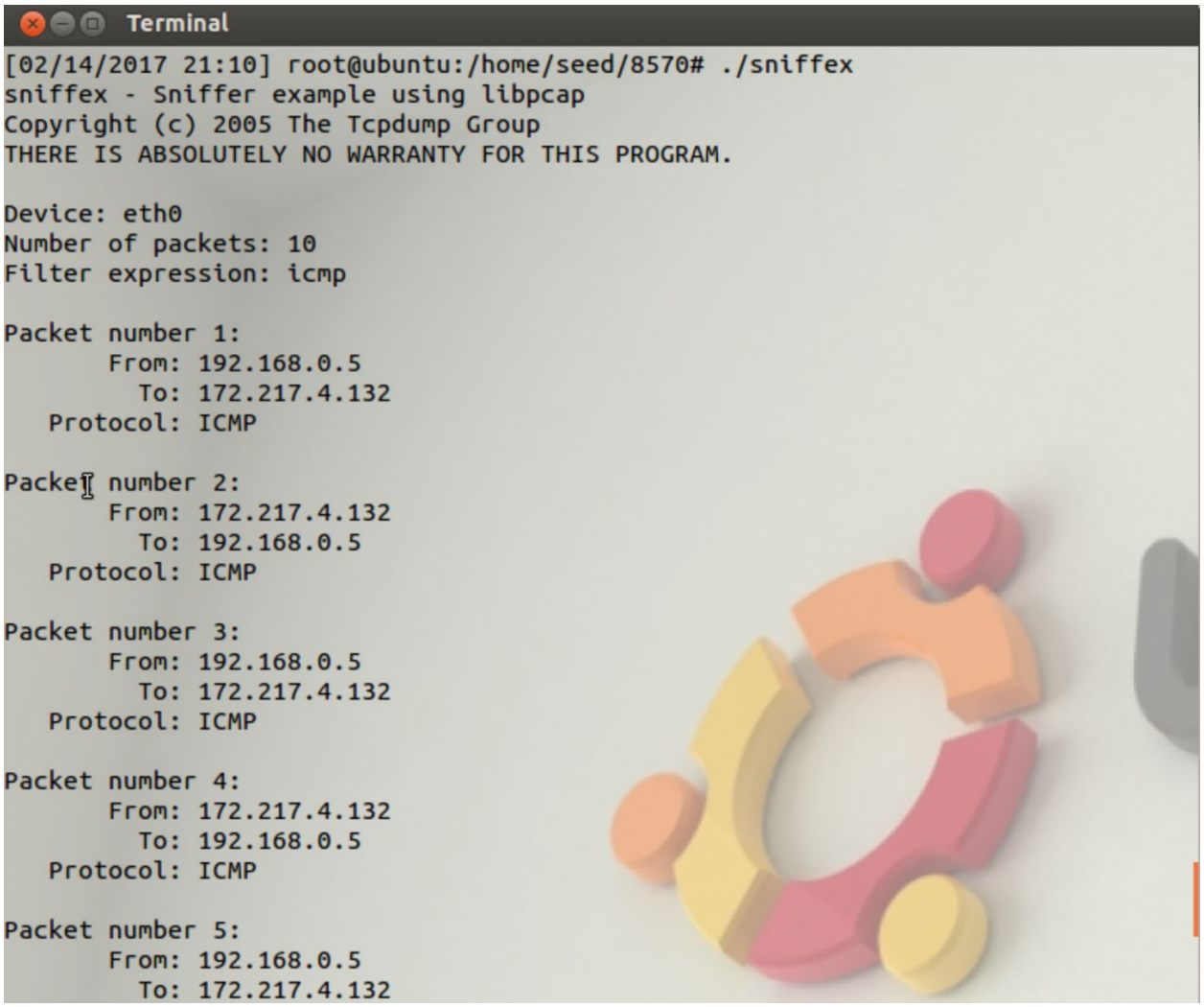


Task 1.b: Writing Filters.

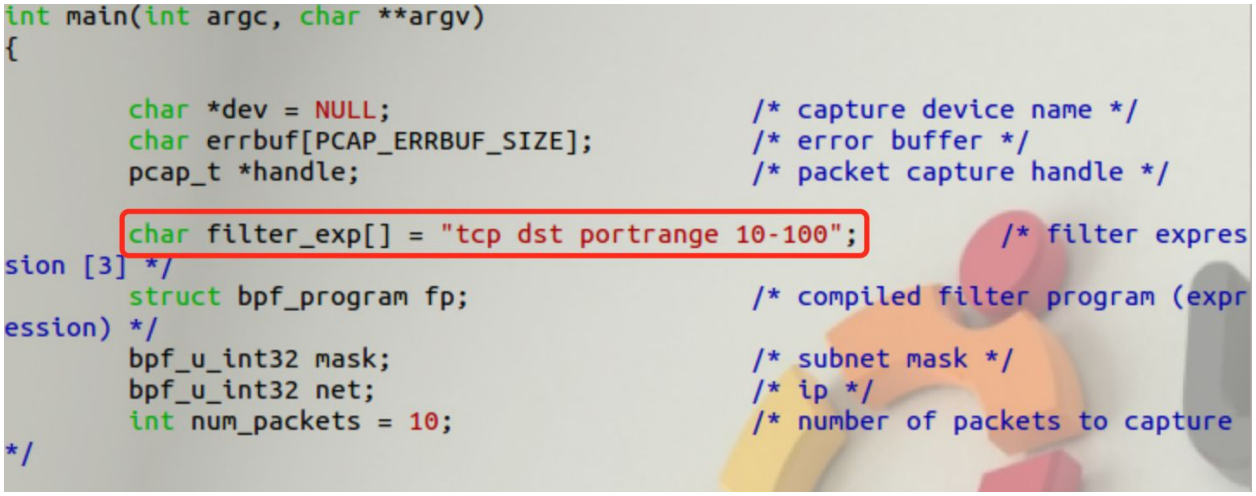
• Capturing the ICMP packets between two specific hosts by setting the filter_exp[ ] into “icmp”.



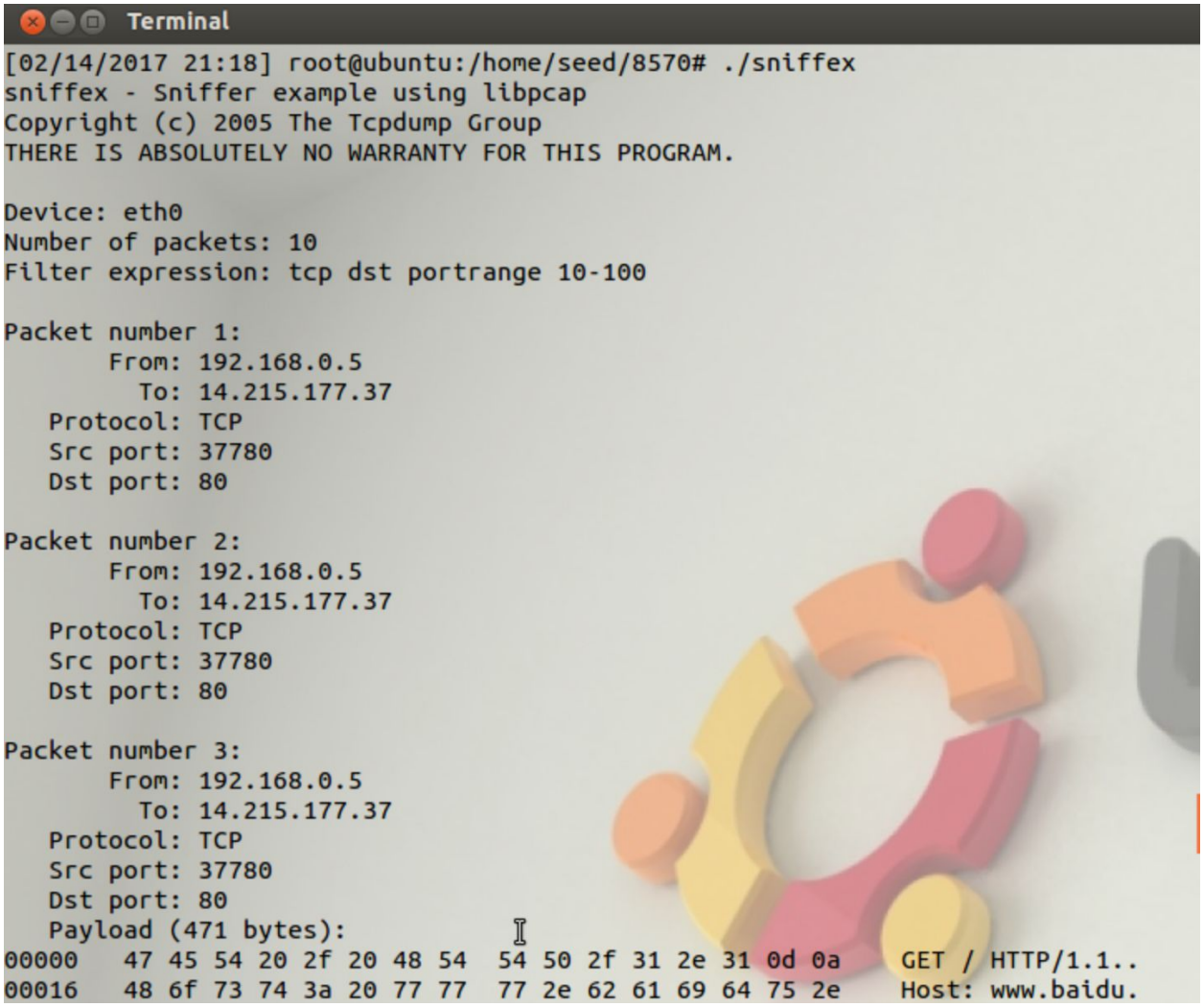
Then this program is compiled, run successfully and displayed expected results.



• Capturing the TCP packets which have a destination port range from to port 10 - 100 through setting the filter_exp[ ] as “tcp dst portrange 10-100”.

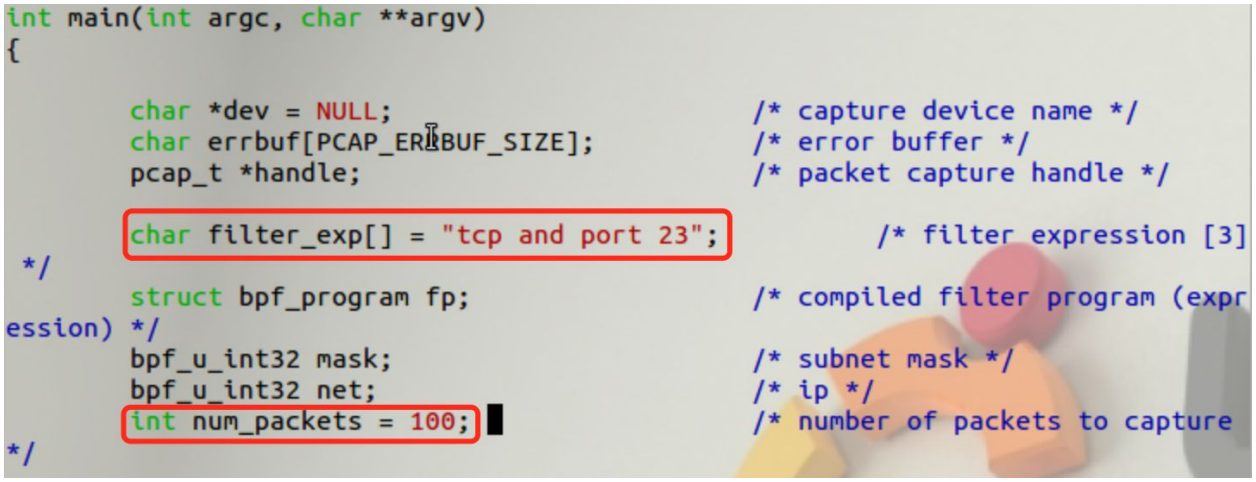


Then the program is compiled, run successfully and displayed expected results.



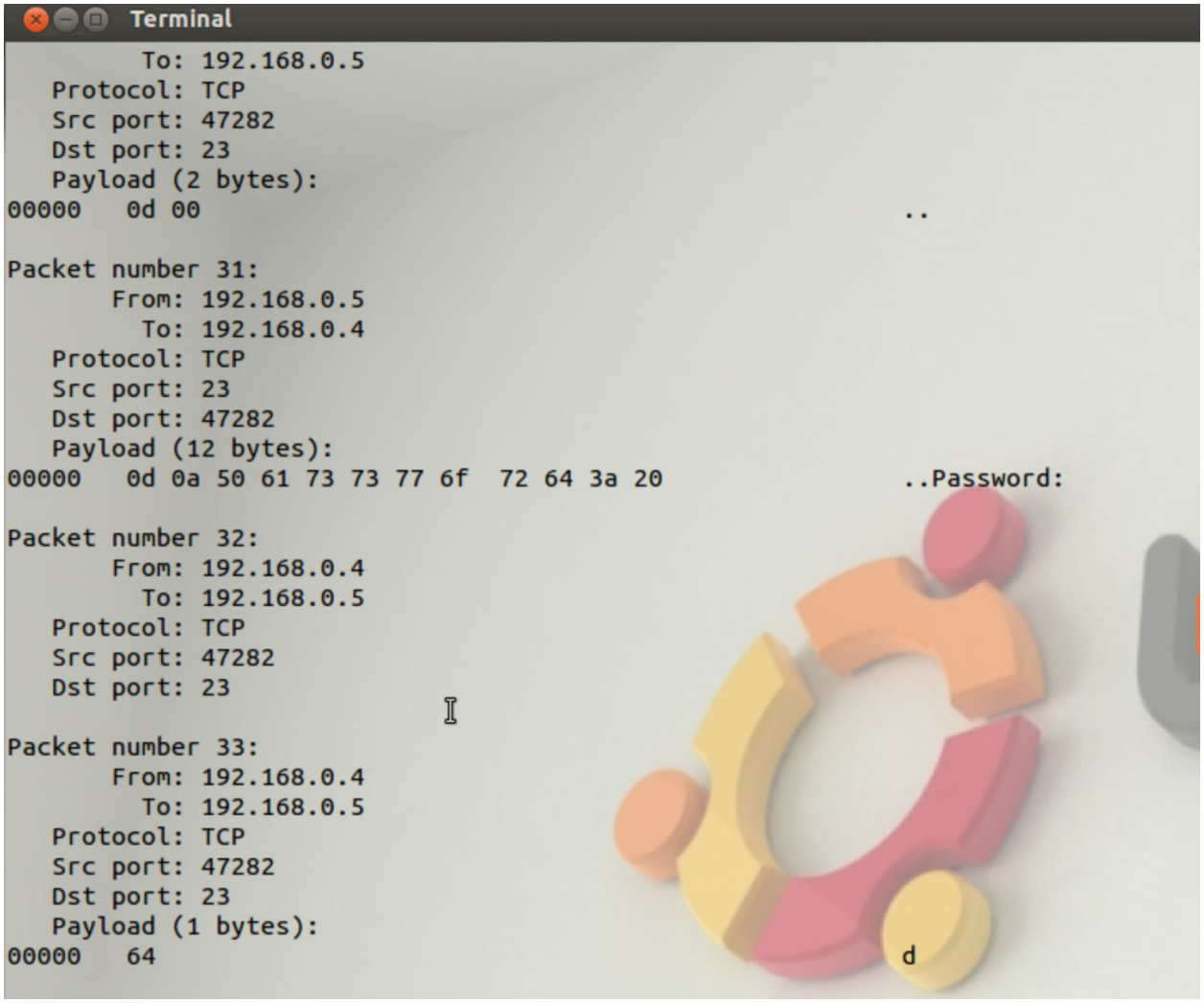
Task 1.c: Sniffing Passwords.

In this assignment, I changed the filter expression into “tcp and port 23” and set num_packets to a bigger number, such as 100.



I use vm1 to execute this program. Meanwhile, I use vm2 to login.

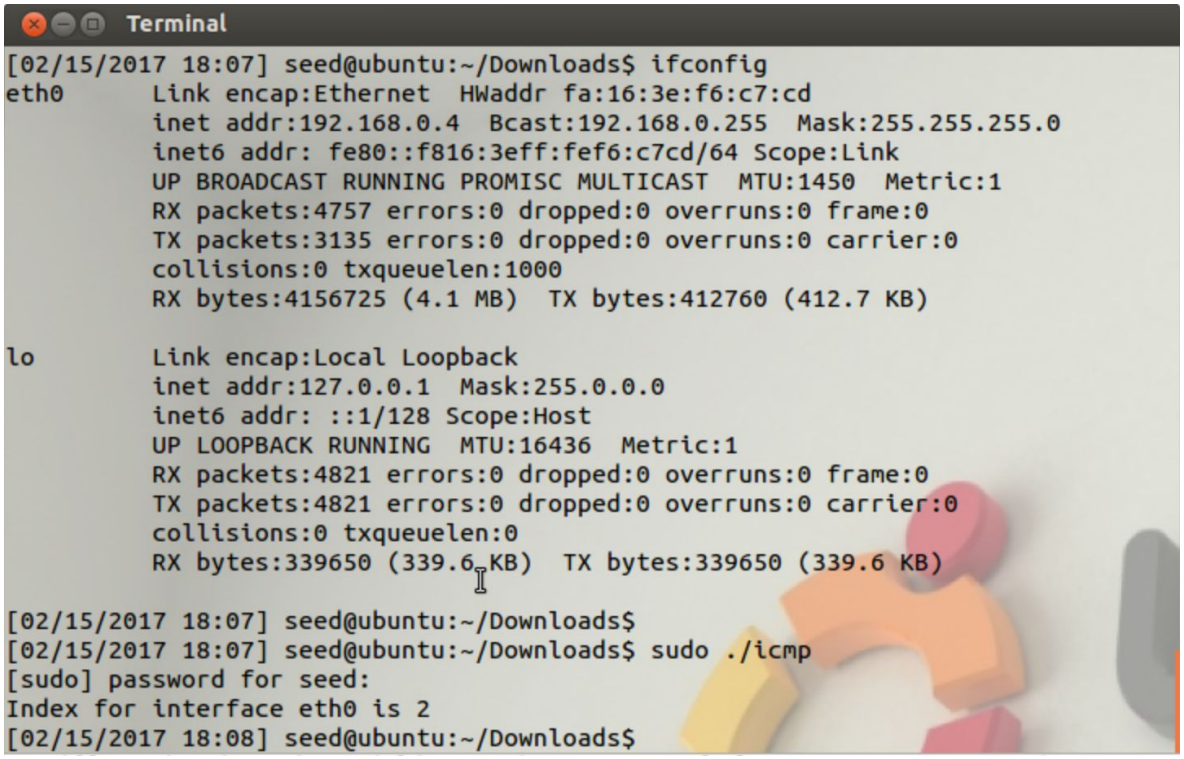
Running the sniffing program in VM1 can get the password successfully.

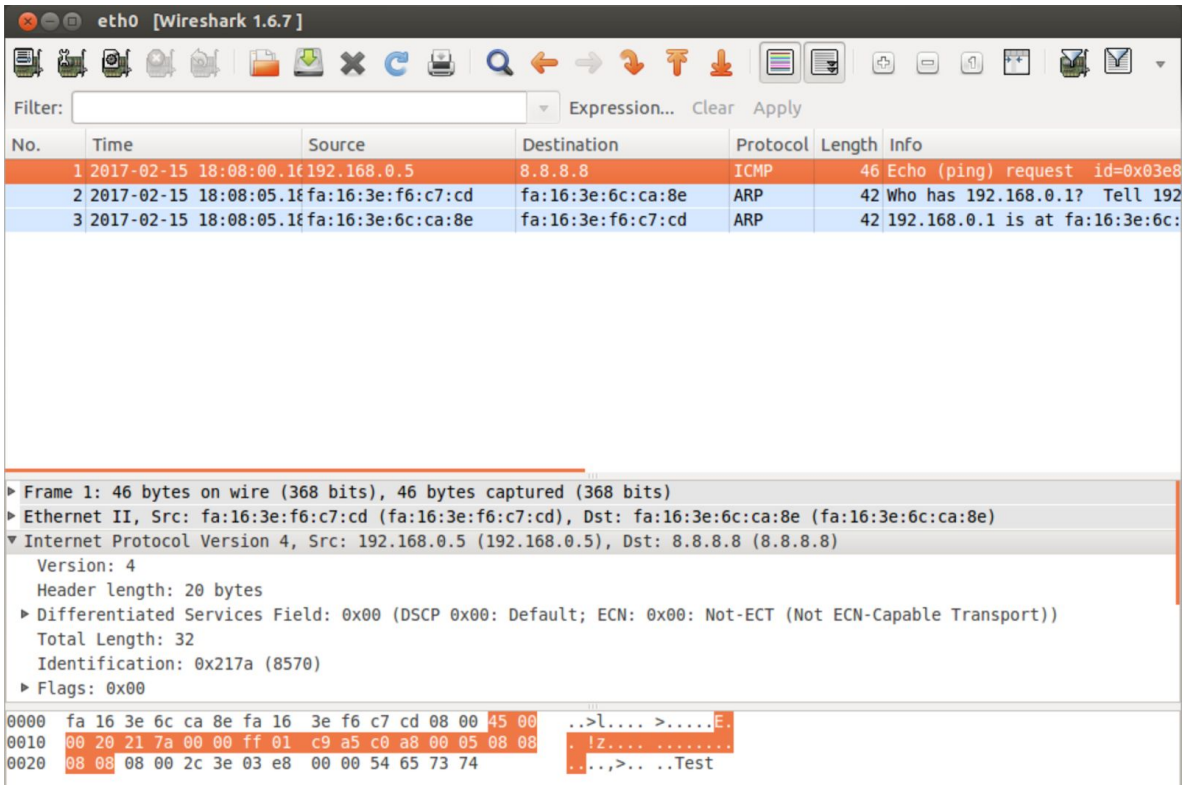


Task 2: Spoofing

Task 2.a: Write a spoofing program.

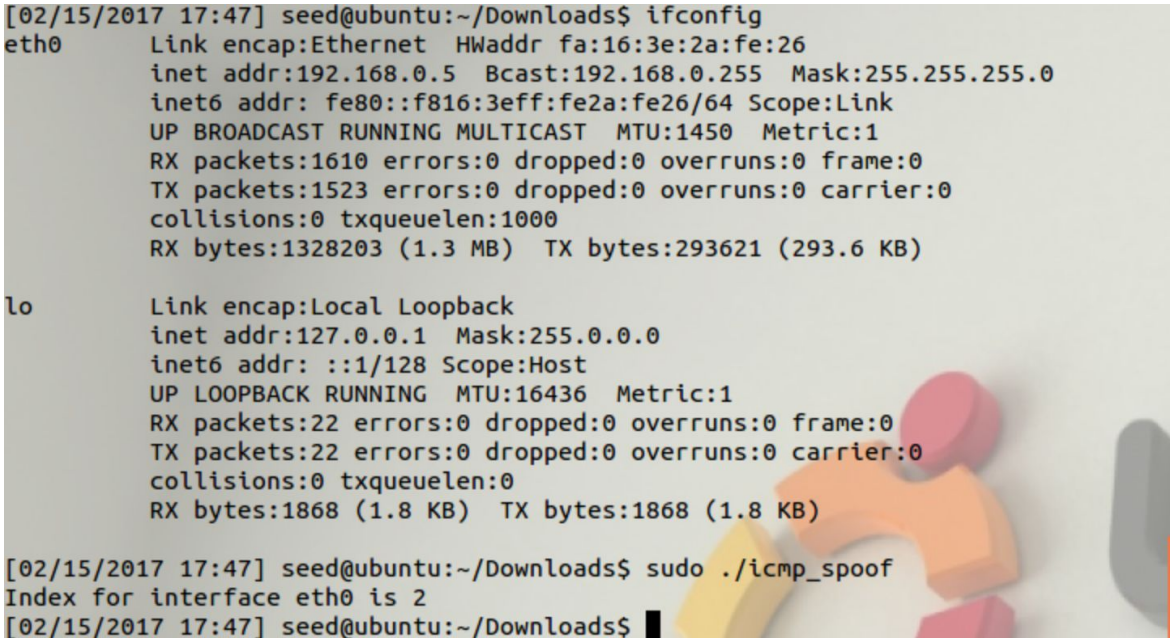
I spoofed an IP packet and sent it to 8.8.8.8, my IP address is 192.168.0.4, but I changed the source IP address into 192.168.0.5. Then the wireshark captured this packet.

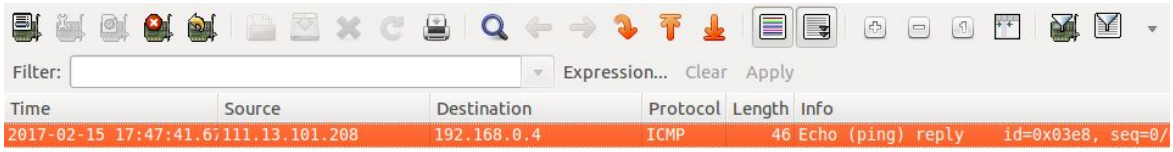


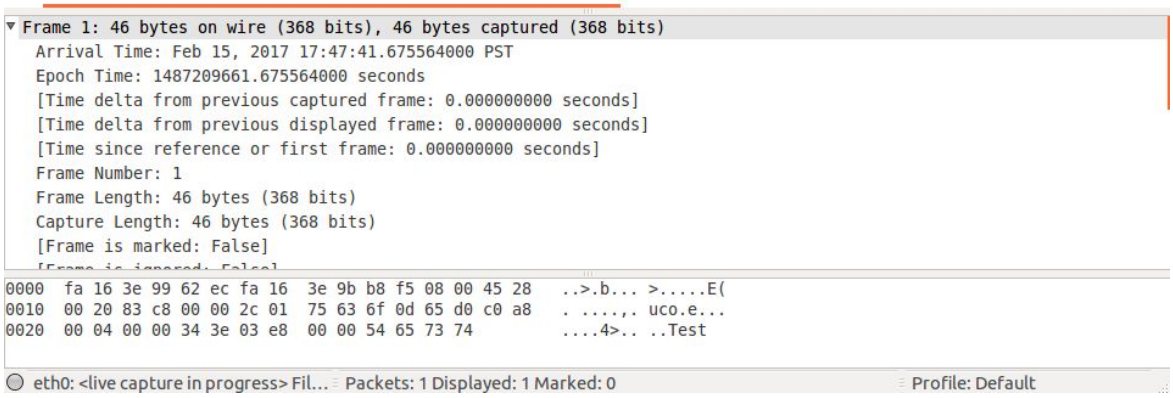


Task 2.b: Spoof an ICMP Echo Request.

In this assignment, my IP address is 192.168.0.5. I spoofed an ICMP Echo Request with the source IP address which is 192.168.0.4 to baidu.com whose IP is 111.13.101.208, and the wireshark captured the packet reply.

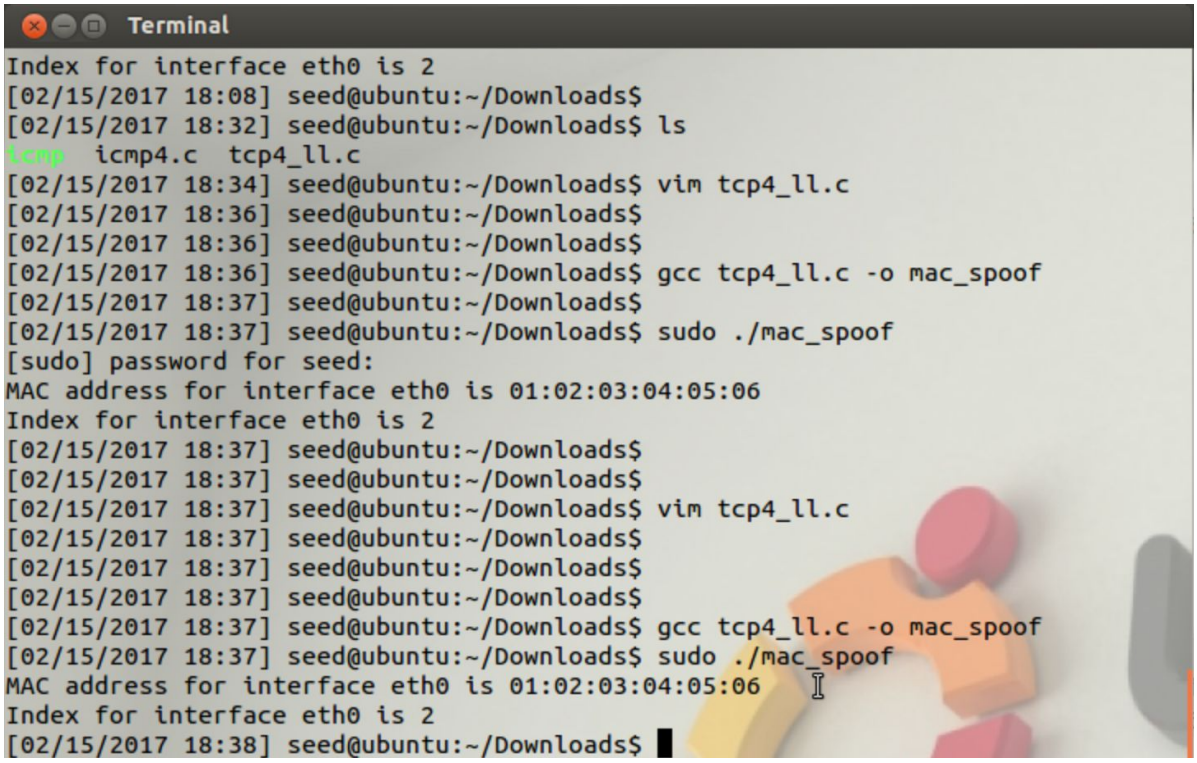


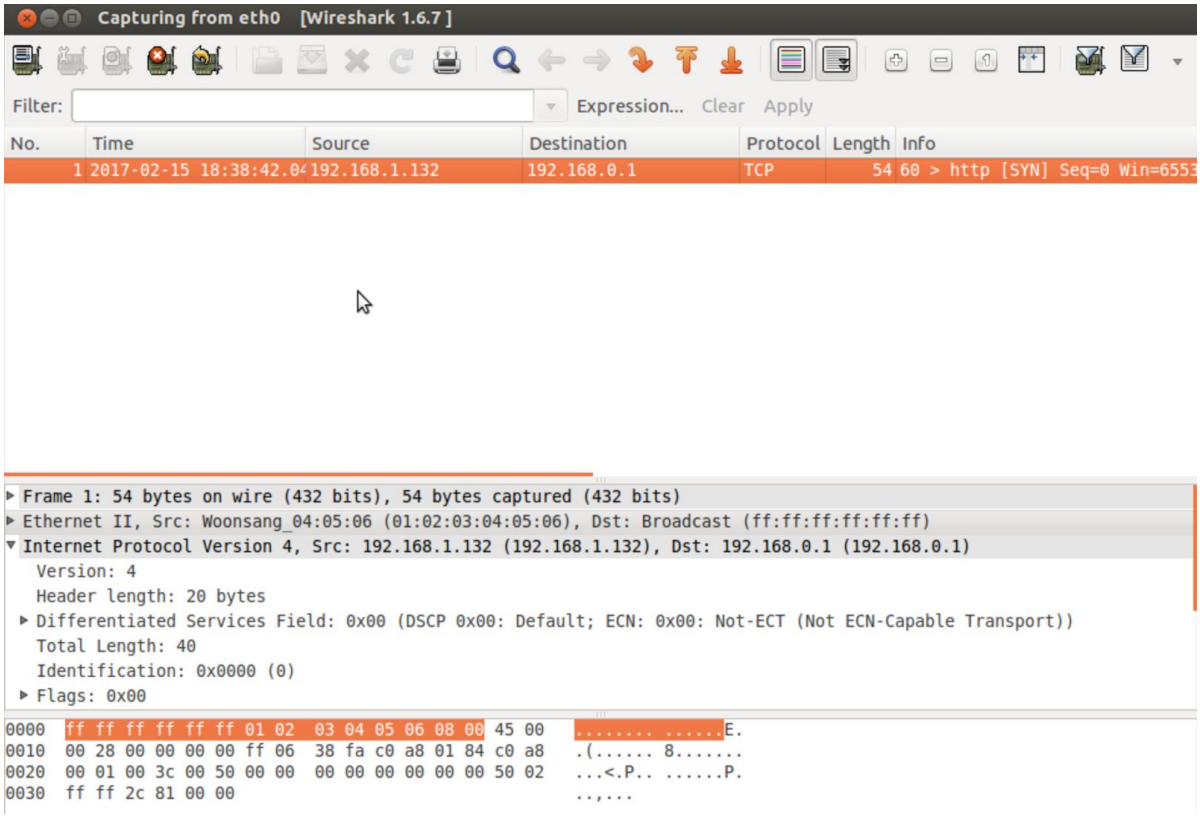




Task 2.c: Spoof an Ethernet Frame.

I setted 01:02:03:04:05:06 as the source MAC address in order to spoof an ethernet frame. Then wireshark captured this packet.





Question 4: Can you set the IP packet length field to an arbitary value, regardless of how big the actual packet is?

No, we can not do that. If we want to compile it successfully, the IP packet length field should be the same as the exact length of a whole IP packet.

Question 5: Using the raw socket programming, do you have to calculate the checksum for the IP header?

No, we don't have to calculate the checksum in case that we use the raw socket programming. A major reason is the system can be able to fill it.

Question 6: Why do you need the root privilege to run the programs that use raw sockets? Where does the program fail if executed without the root privilege?

We need the root privilege to run the programs that use raw sockets, since the socket() function needs the root privilege to get access to devices. Otherwise, we will get the error "socket() error: Operation not permitted".

It fails in the following code: