# Lab1 Report

**Task 1.a: Understanding sniffex. Please download the sniffex.c program from the tutorial mentioned above, compile and run it. You should provide screendump evidence to show that your program runs successfully and produces expected results.**

Codes:



Codes explanation:

The method got\_packet(…) is a method showing that we have captured a packet successfully and tell us some details about the packet;

In the main function, the name of the NIC and the size of Buffer has been told to the handle by the following code:

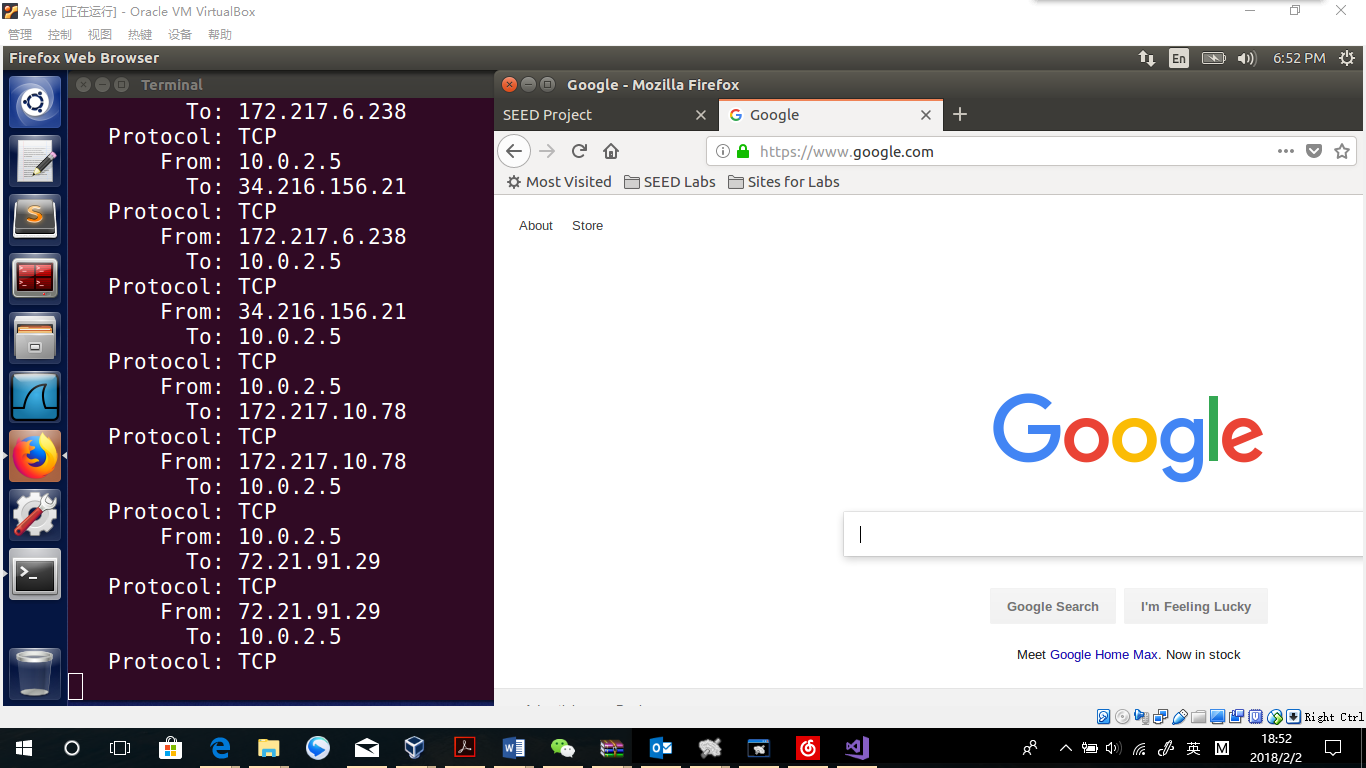
*handle = pcap\_open\_live("enp0s3", BUFSIZ, 1, 1000, errbuf);*

While compiling filter\_exp into BPF, since the filter is empty so there are no packets designed to be refused.

Complie and excute:



View google.com through explorer



**Observation:**

We complile the file successfully. While running the program, there is no output until I visit the Internet. All packets are listed in the terminal, including protocol type, source address and destination address.

**Explanation:**

Those packets are from the communications between host and websites. Since we visit the SEED Project website and Google website, TCP packets has been captured in addresses among host(10.0.2.5) and the two websites. The reason that nothing was printed before we use the explorer is that there are no other process using the Ethernet (In this case, is enp0s3). And because I have nothing in the *char filter\_exp[]* parameter, all packets should be captured and printed.

**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.**

First, a handle should be produced telling the sniffer programs the NIC name and buffer size by *pcap\_open\_live*.

Then, design of the filter should be done by *pcap\_compile* and *pcap\_setfilte* including building the BPF code and set it to the handle.

Thirdly, pcap\_loop is used as a starter of the sniffer program. Inside, running times and operations should be given.

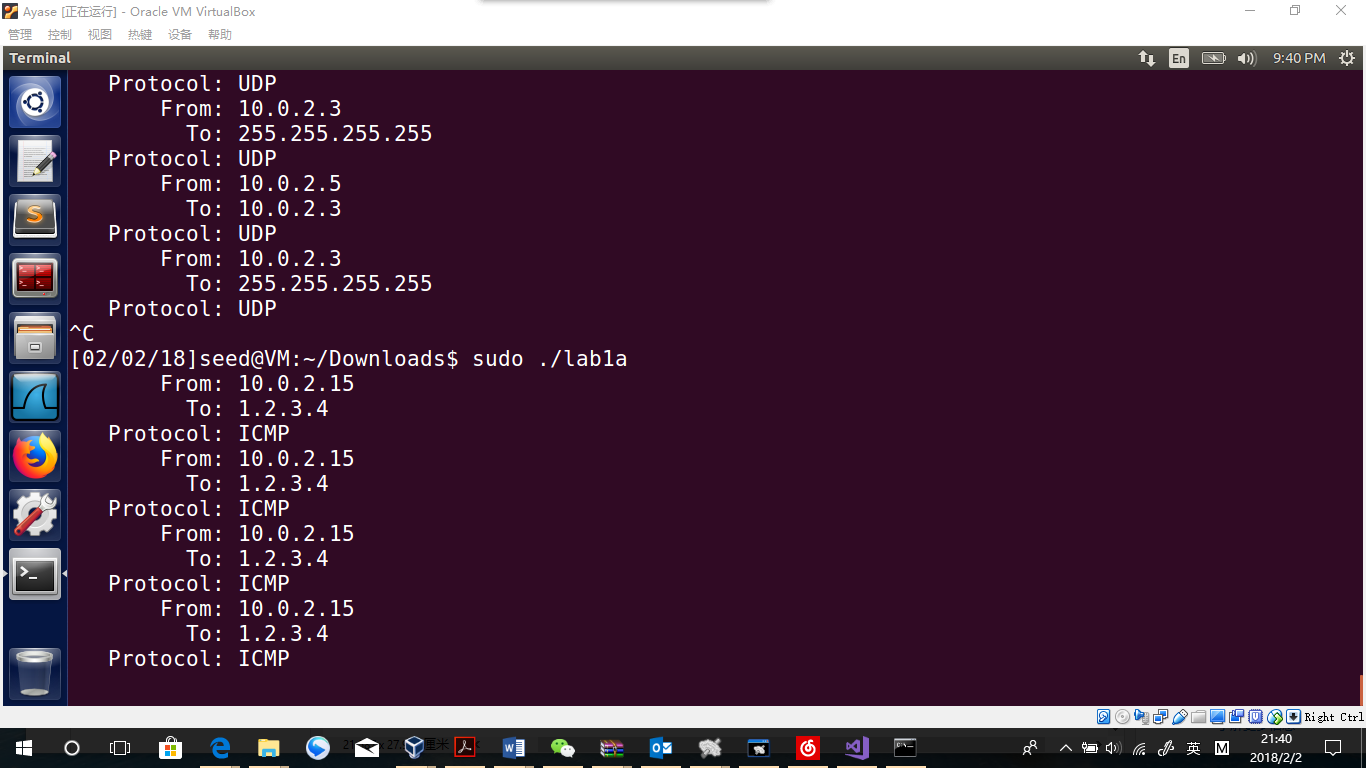
Finally, close the handle by pcap\_close.

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

Without the root privilege, the program will end since the library call “pcap\_open\_live” executes. This is because the method create a raw socket. Usually an OS does not allow users to create a raw socket themselves.

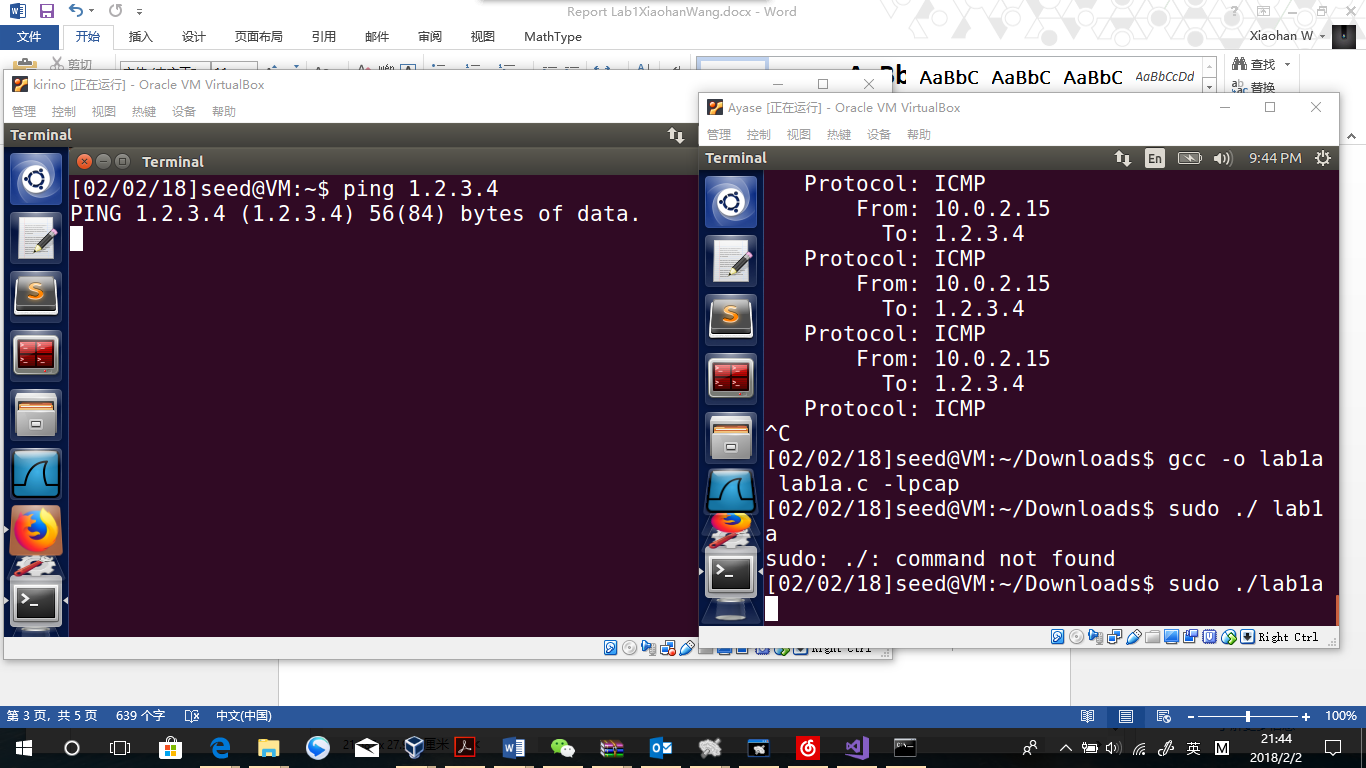
**Problem 3: Please turn on and turn off the promiscuous mode in the sniffer program. Can you demonstrate the difference when this mode is on and off? Please describe how you demonstrate this.**

Promiscuous mode open:



As you can see, in the host machine(10.0.2.5), with the sniffer program I can see my another virtual machine pinging “1.2.3.4”

handle = pcap\_open\_live("enp0s3", BUFSIZ, 0, 1000, errbuf);



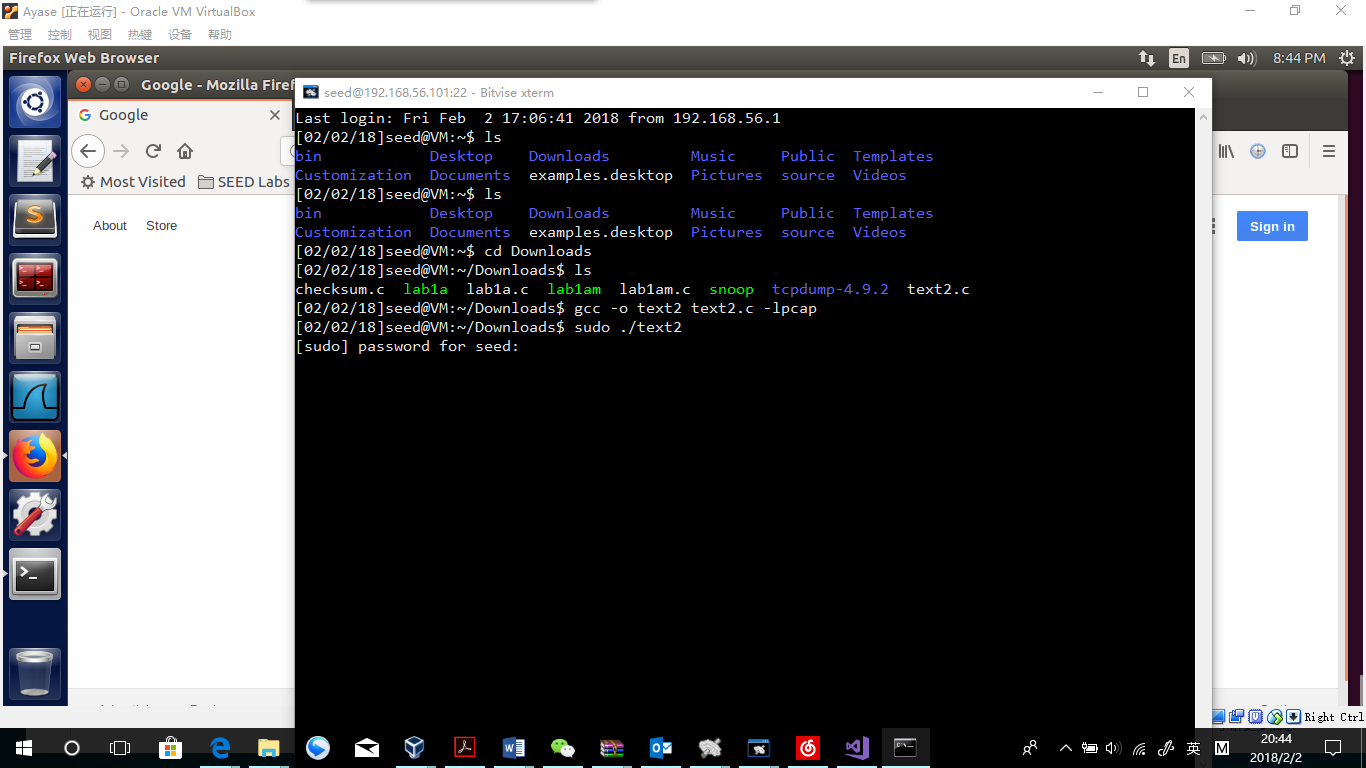
I turned off the promiscuous mode and as you can see now the it can not see another virtual machine “kirino” pinging.

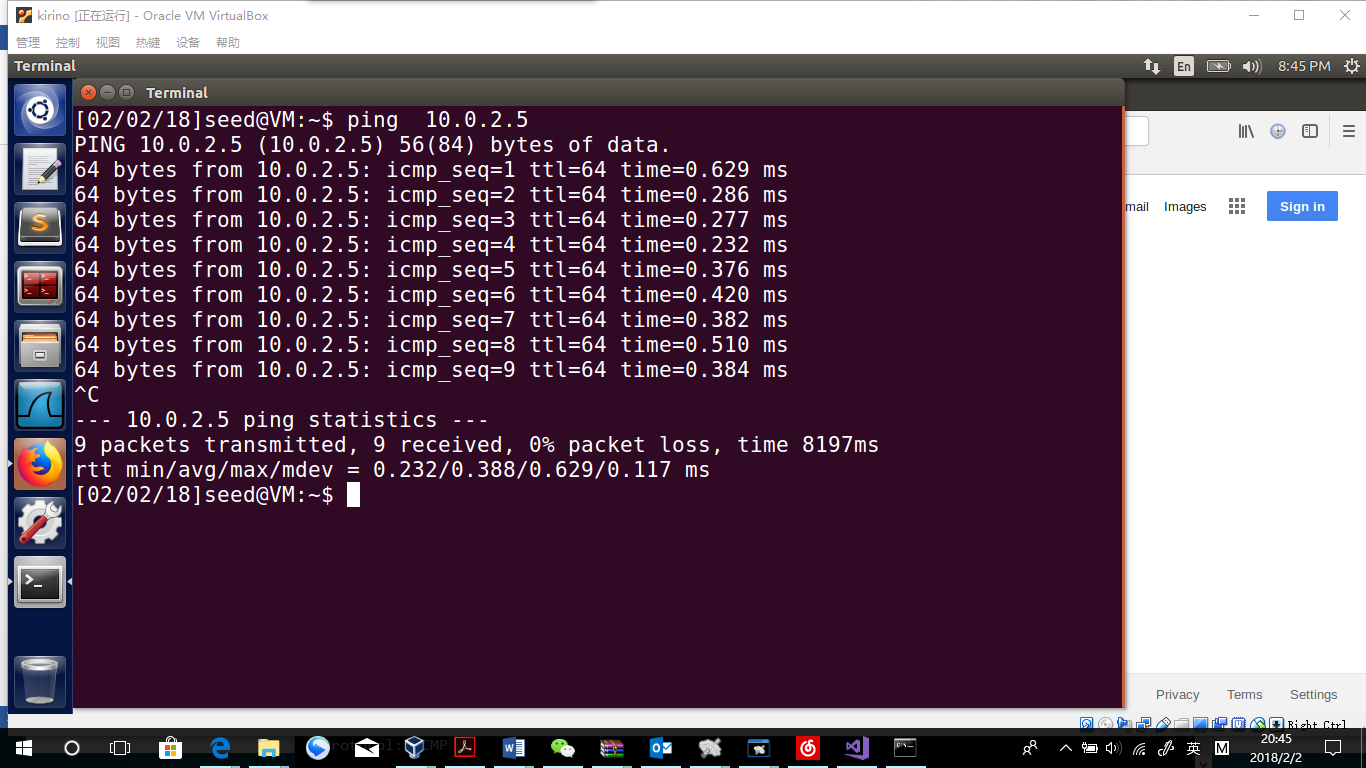
In short, with promiscuous mode closed, the sniffer program will not capture packets if the host is neither the source nor the destination. Which means, if we want to sniffer other internet activity, we should always open the promiscuous mode.

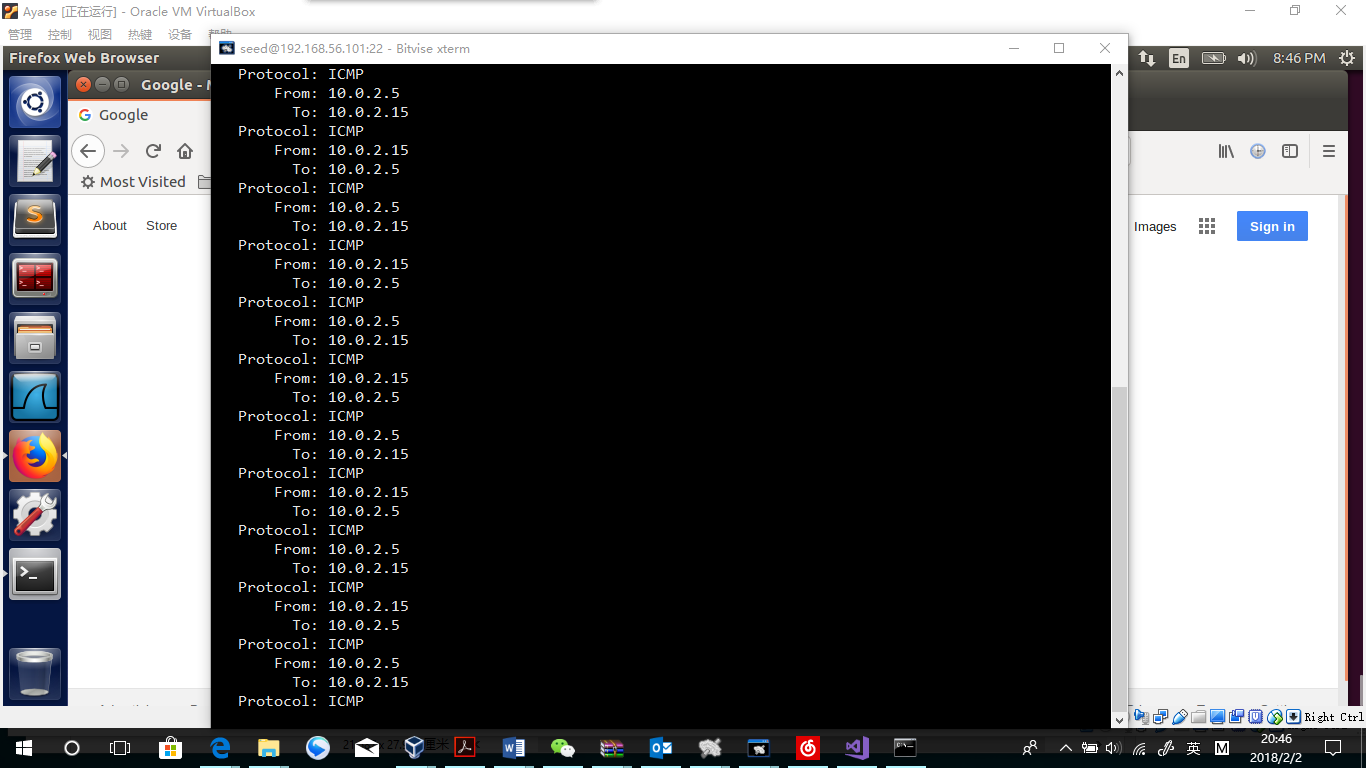
**Task 1.b: Writing Filters. Please write filter expressions for your sniffer program to capture each of the followings. In your lab reports, you need to include screendumps to show the results of applying each of these filters.**

**\_ Capture the ICMP packets between two specific hosts.**

*char filter\_exp[] = "icmp && (host 10.0.2.5 && host 10.0.2.15)";*







**Observation:**

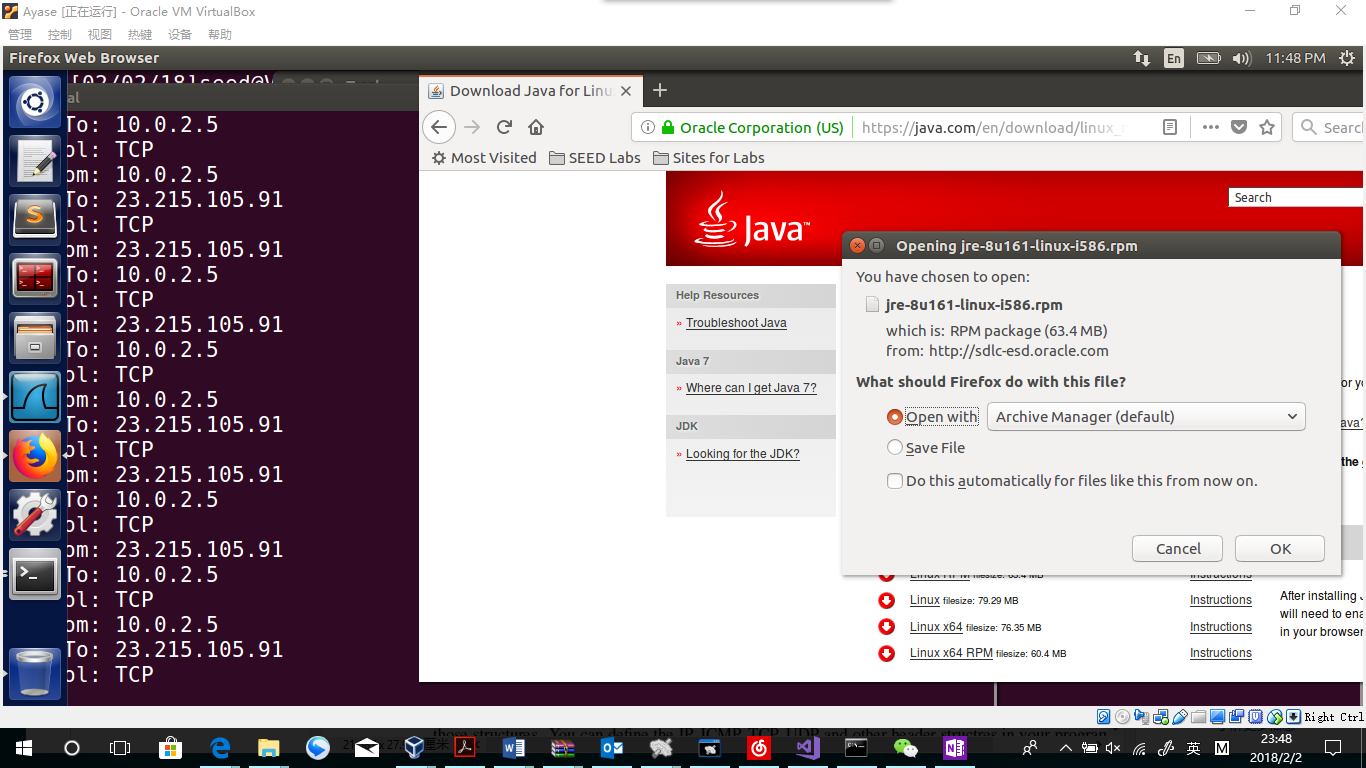
After change the information of filter, there will be no packets captured while visiting websites or getting pings from any computers but 10.0.2.15.

**Explanation:**

The filter works!!

**\_ Capture the TCP packets that have a destination port range from to port 10 - 100.**

Tcp && dst port 10-100

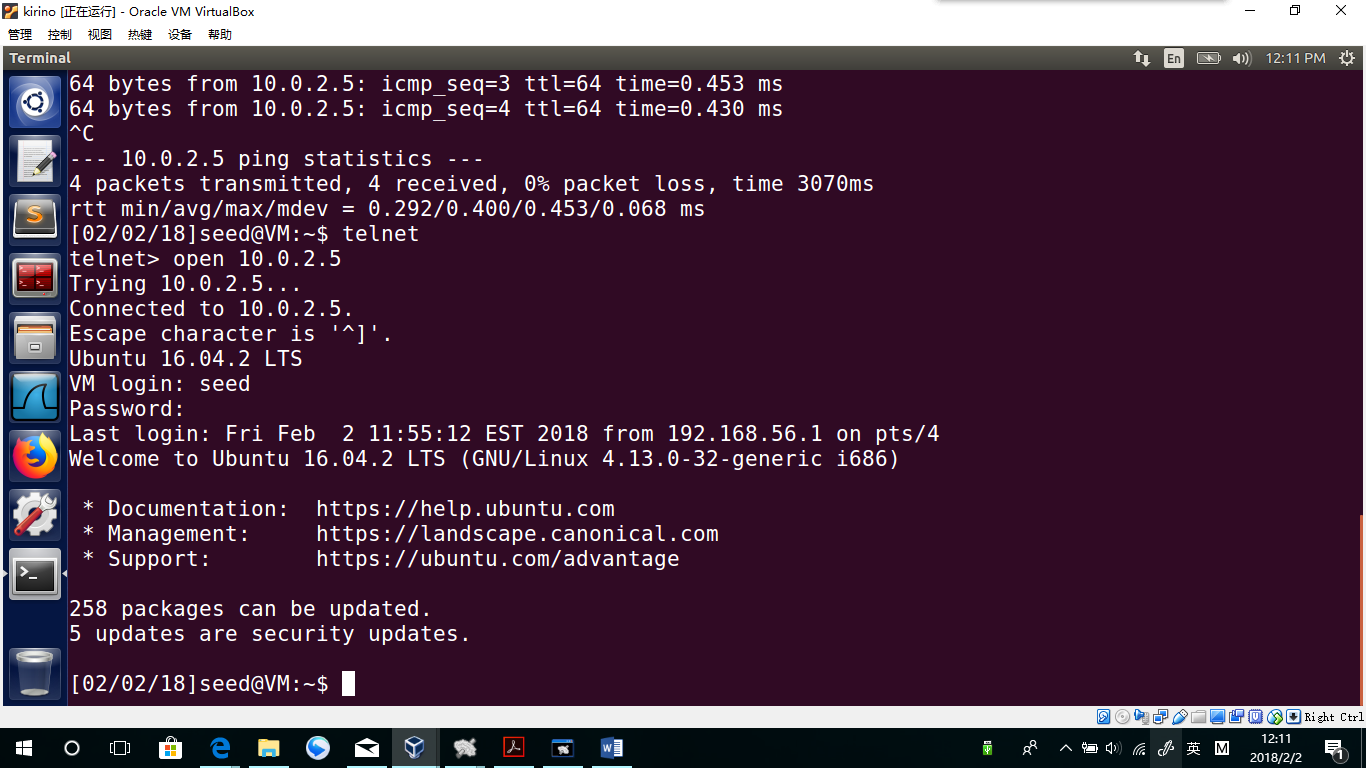


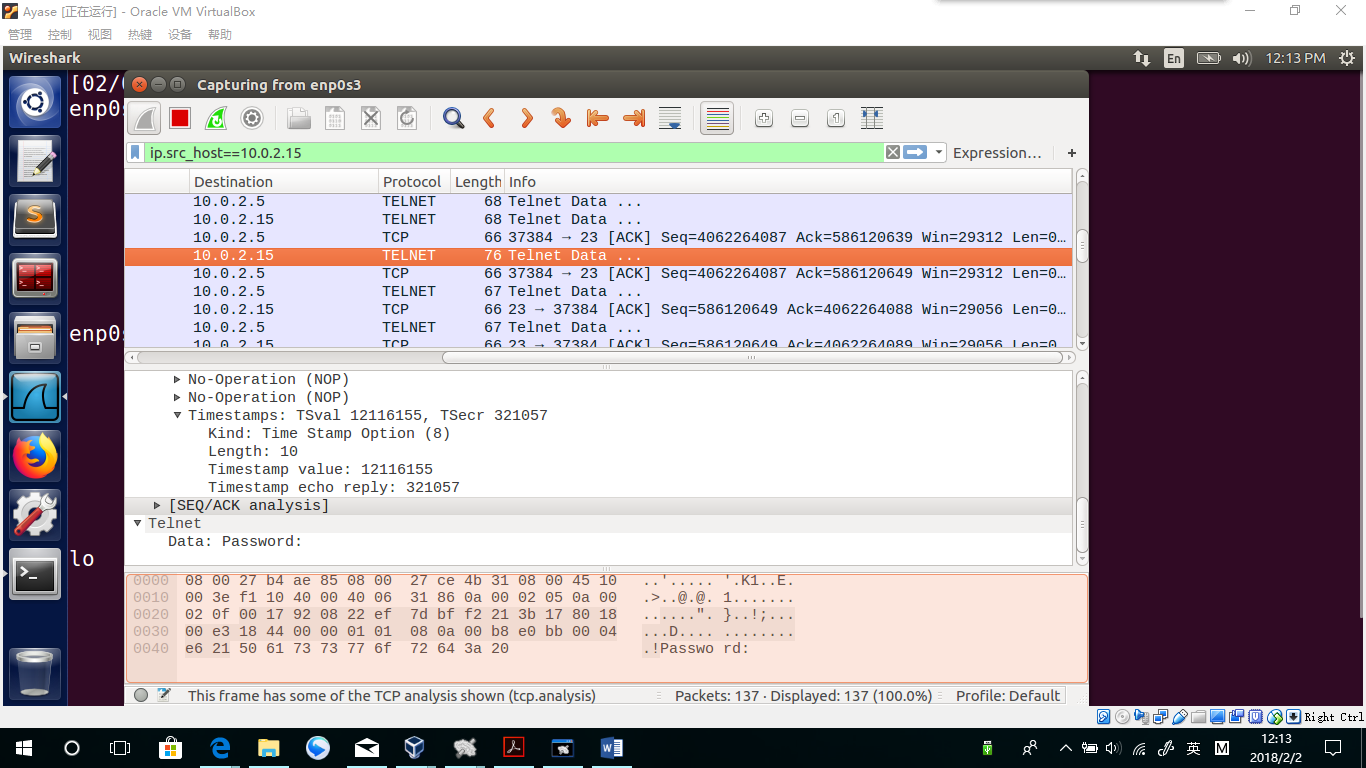
Explanation:

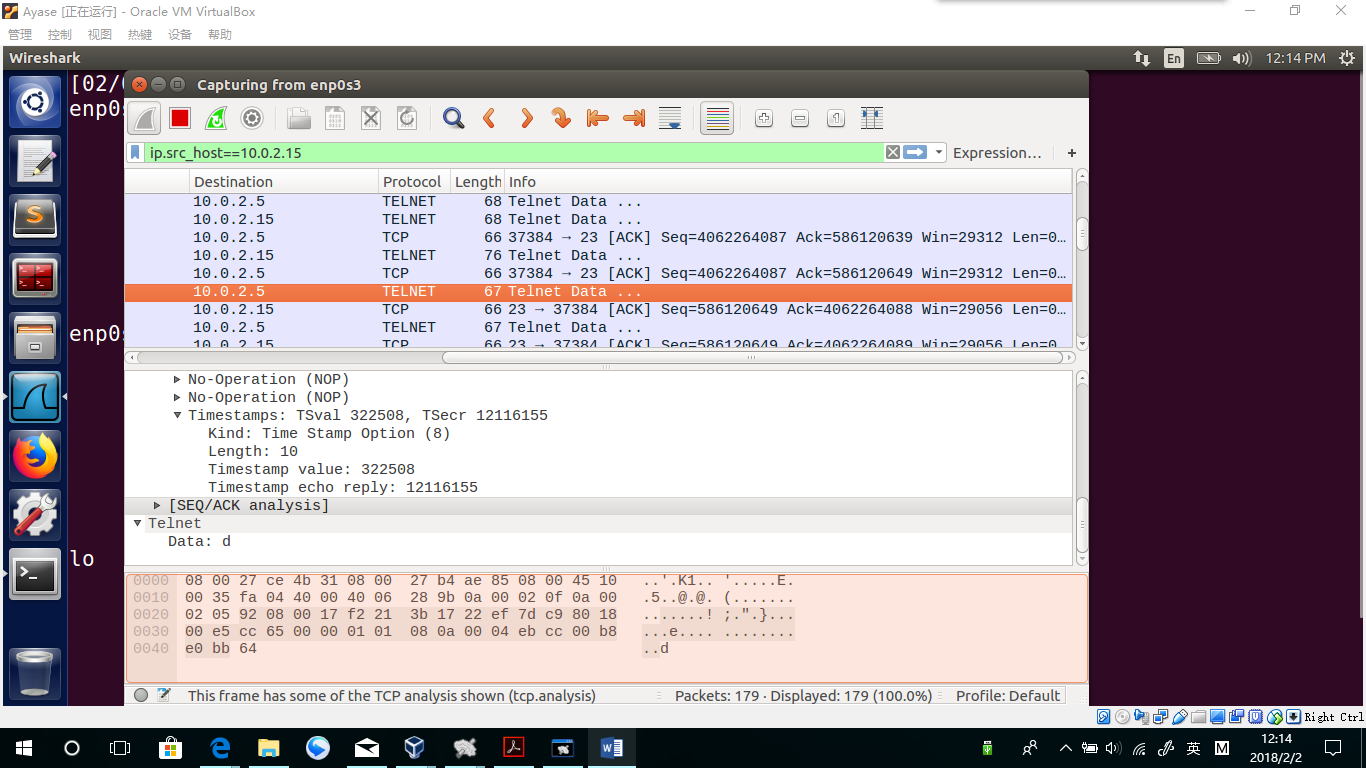
Only TCP protocol remains. While downloading small files via firefox, the packet information can not be recorded. That is because file transportation is via TCP port 115.

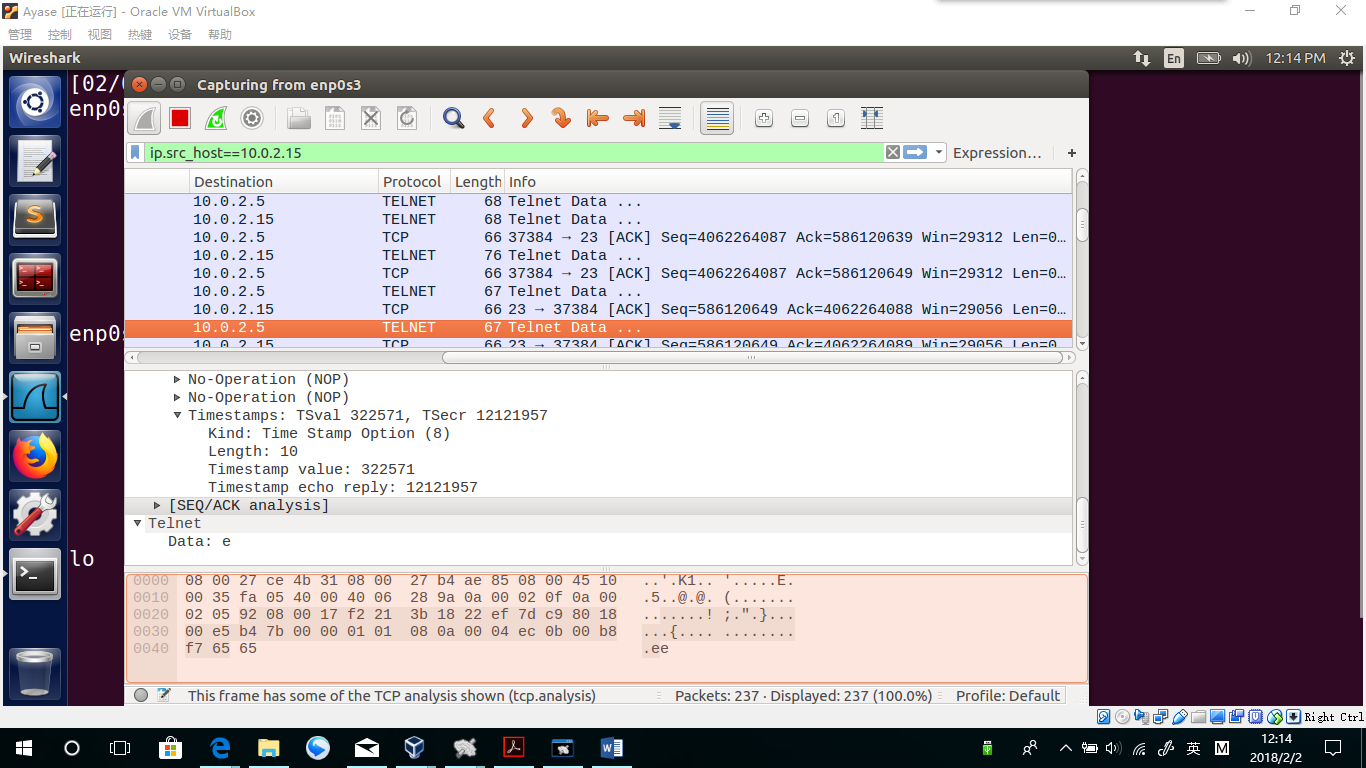
**Task 1.c: Sniffing Passwords. Please show how you can use sniffex to capture the password when somebody is using telnet on the network that you are monitoring. You may need to modify the sniffex.c a little bit if needed. You also need to start the telnetd server on your VM. If you are using our pre-built VM, the telnetd server is already installed; just type the following command to start it.**

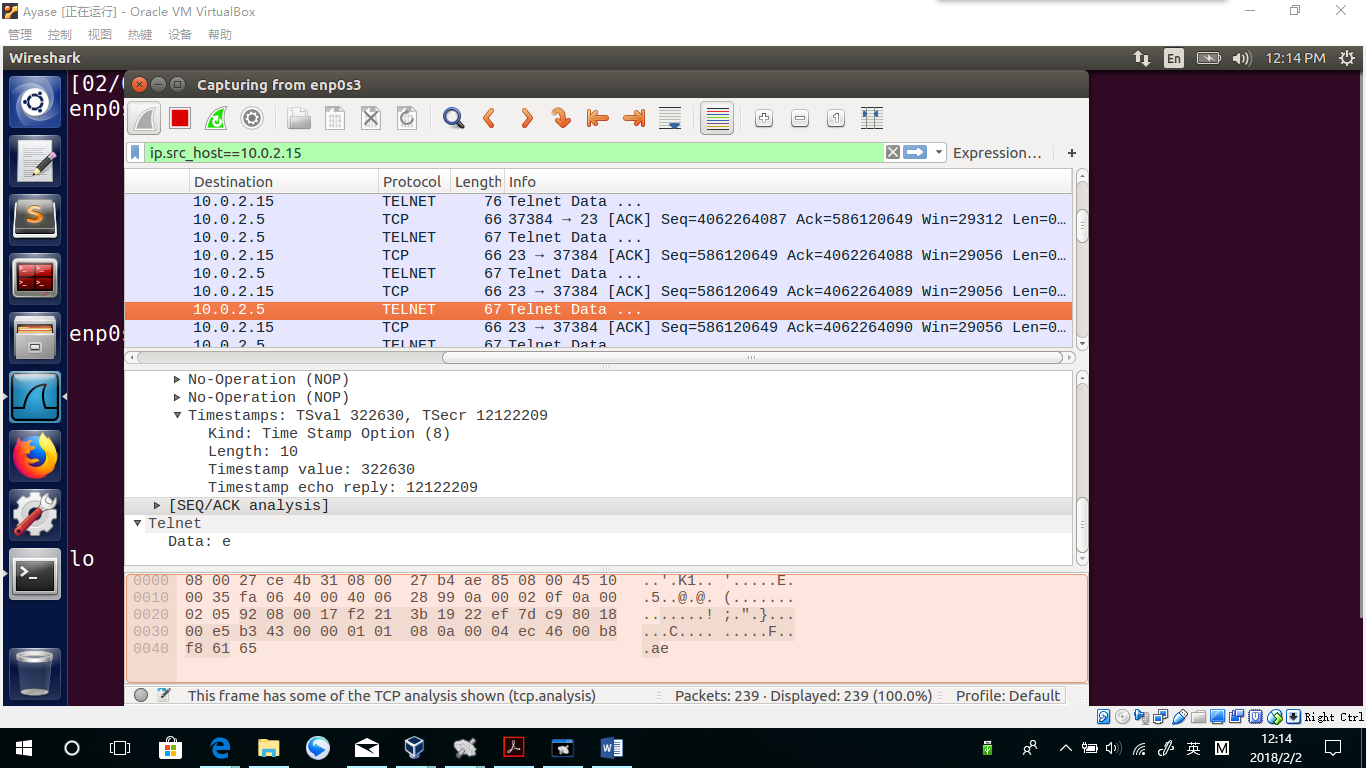
Telnet

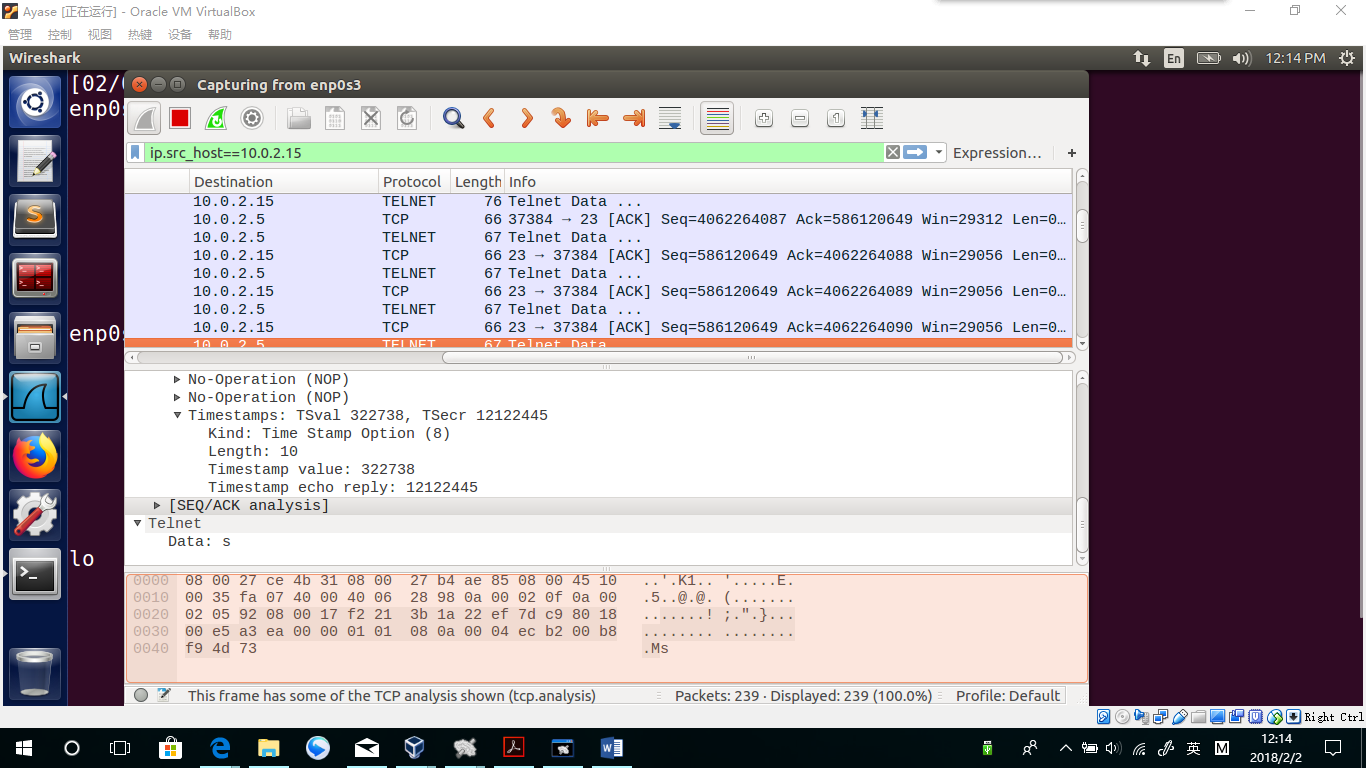












**Observation:**

Using the wireshark software, I see data in packets with a protocol of TELNET. Data is as follows

Password:

D

E

E

S

**Explanation:**

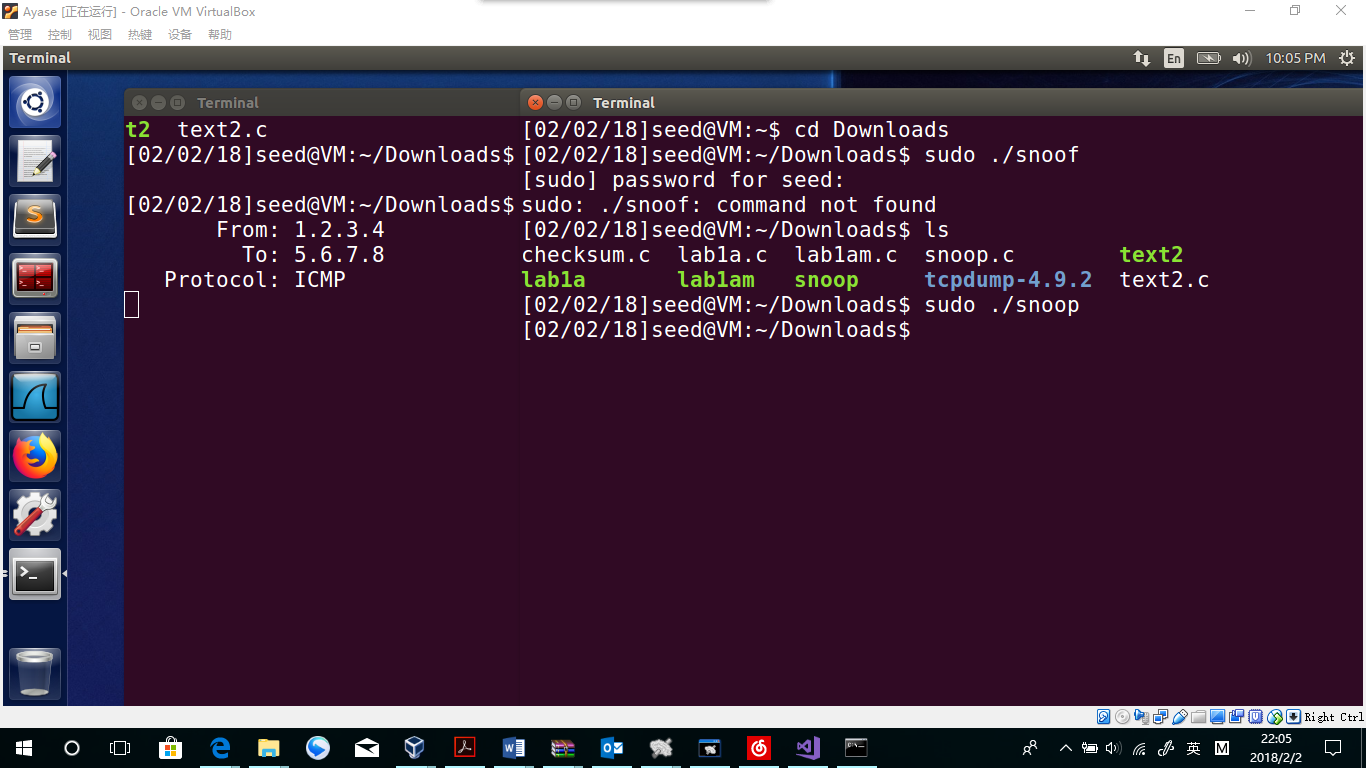
As you can see, I read the password directly, mainly because the data is transported without encryption. After I use TELNET to launch the VM from another VM, the password I entered is sent character by character without encryption.

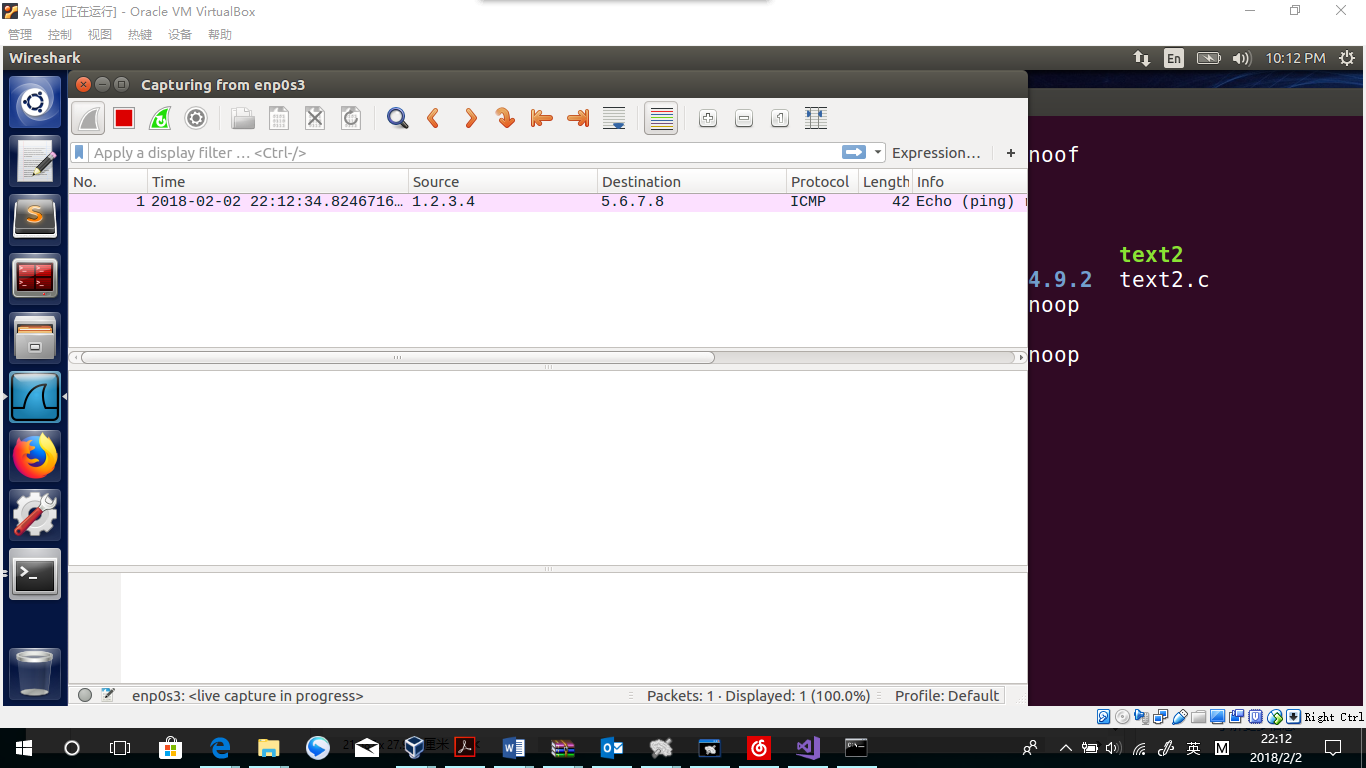
**Task 2.a: Write a spoofing program. You can write your own packet program or download one. You need to provide evidences (e.g., Wireshark packet trace) to show us that your program successfully sends out spoofed IP packets.**



**Explanation for codes:**

I build a raw socket and set all the information in ip header myself. As a matter of fact, I can set my source address and target address as two fake address.





**Observation:**

The sniffer program get a packet from 1.2.3.4 to 5.6.7.8.

**Explanation:**

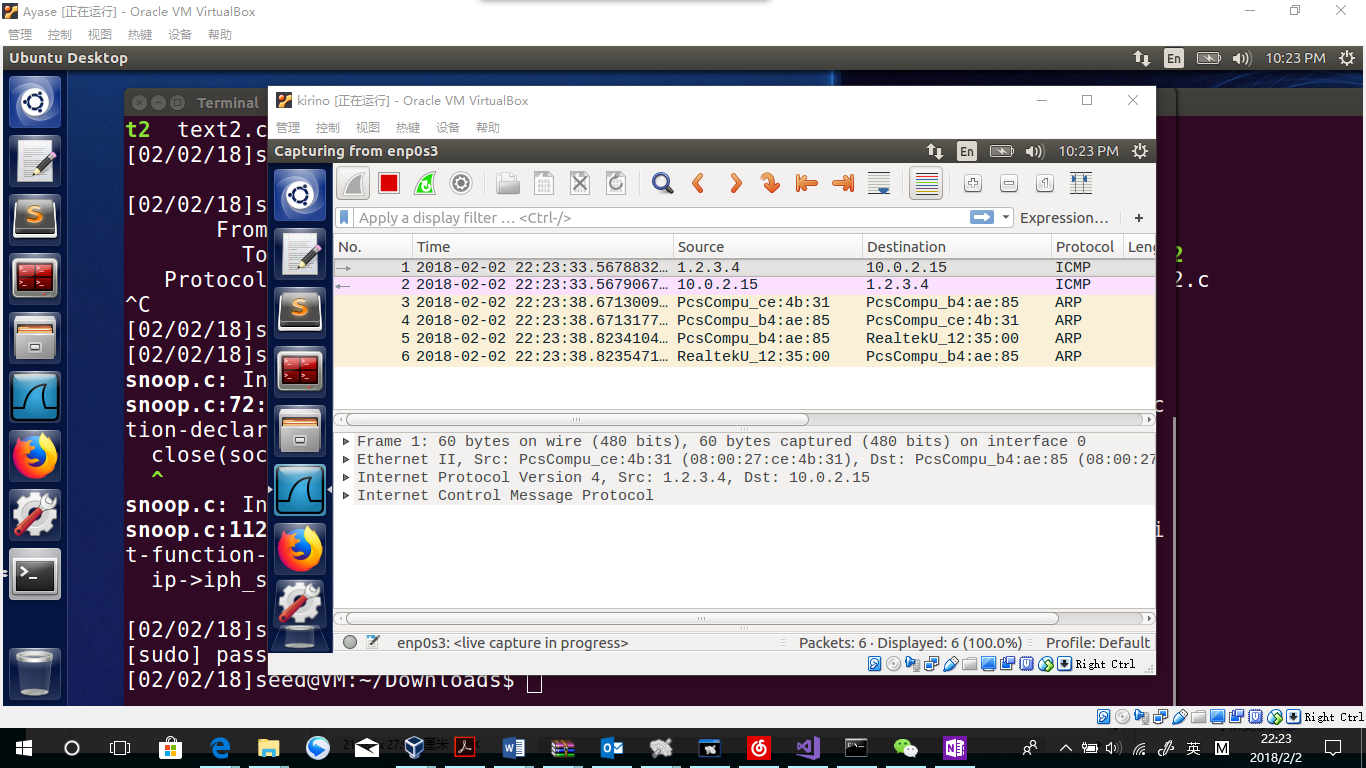
Obviously, because neither the two address above is real address, the spoof program works successfully.

**Task 2.b: Spoof an ICMP Echo Request. Spoof an ICMP echo request packet on behalf of another machine (i.e., using another machine’s IP address as its source IP address). This packet should be sent to a remote machine on the Internet (the machine must be alieve). You should turn on yourWireshark, so if your spoofing is successful, you can see the echo reply coming back from the remote machine.**

****

**Code explanation:**

The same code as 2.a, only changing the destination address to the second VM’s address



**Observation:**

In the “kirino” VM’s WireShark, an ICMP request is received. “kirino” VM send back packets to answer the ICMP.

**Explanation**:

The packet sent is “ayase” machine’s spoof packet. Changing the source address as 1.2.3.4, the spoofing program runs successfully. While “kirino” get the packet, it will think there would be a ping request form “1.2.3.4”, and that is how we expected.

**Task 3: Sniff and then Spoof In this task, you will combine the sniffing and spoofing techniques to implement the following sniff-andthen-spoof program. You need two VMs on the same LAN. From VM A, you ping an IP X. This will generate an ICMP echo request packet. If X is alive, the ping program will receive an echo reply, and**

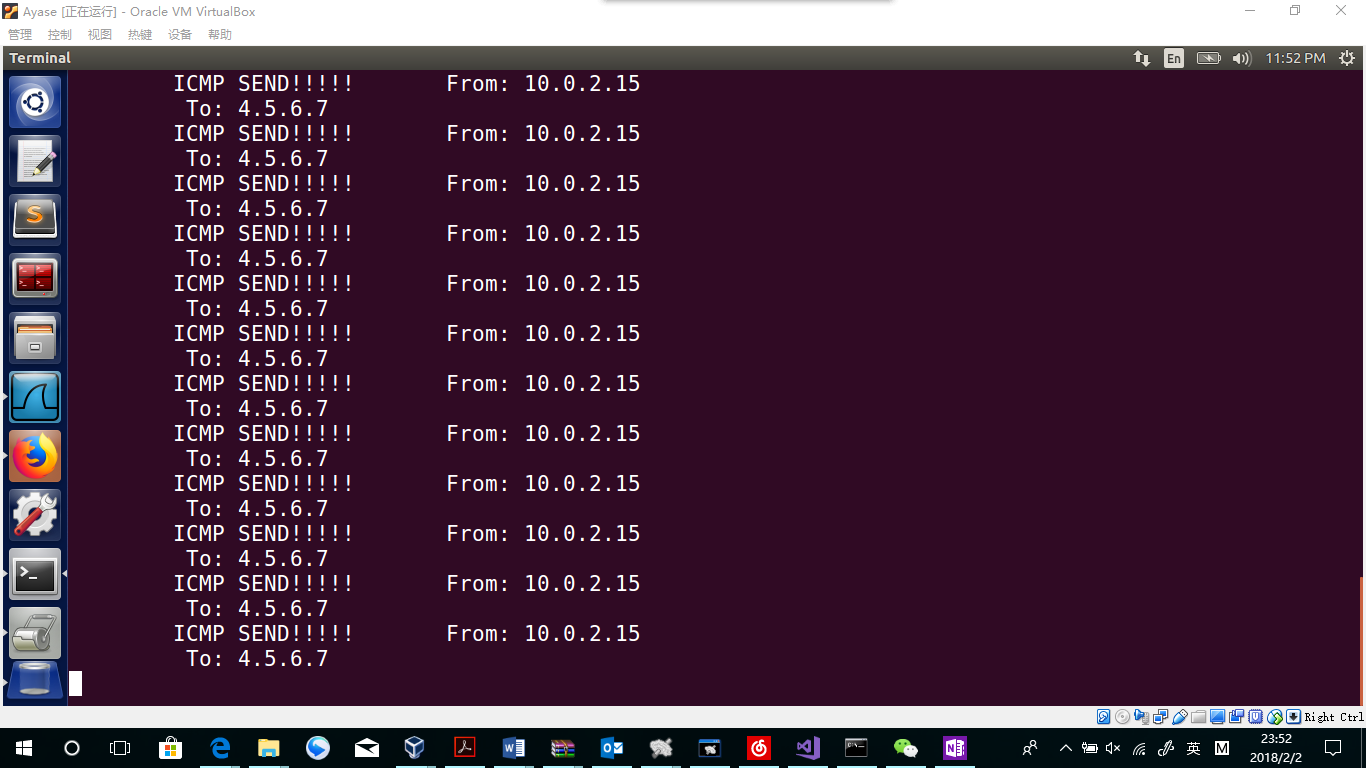
**print out the response. Your sniff-and-then-spoof program runs on VM B, which monitors the LAN through packet sniffing. Whenever it sees an ICMP echo request, regardless of what the target IP address is, your program should immediately send out an echo reply using the packet spoofing technique. Therefore, regardless**

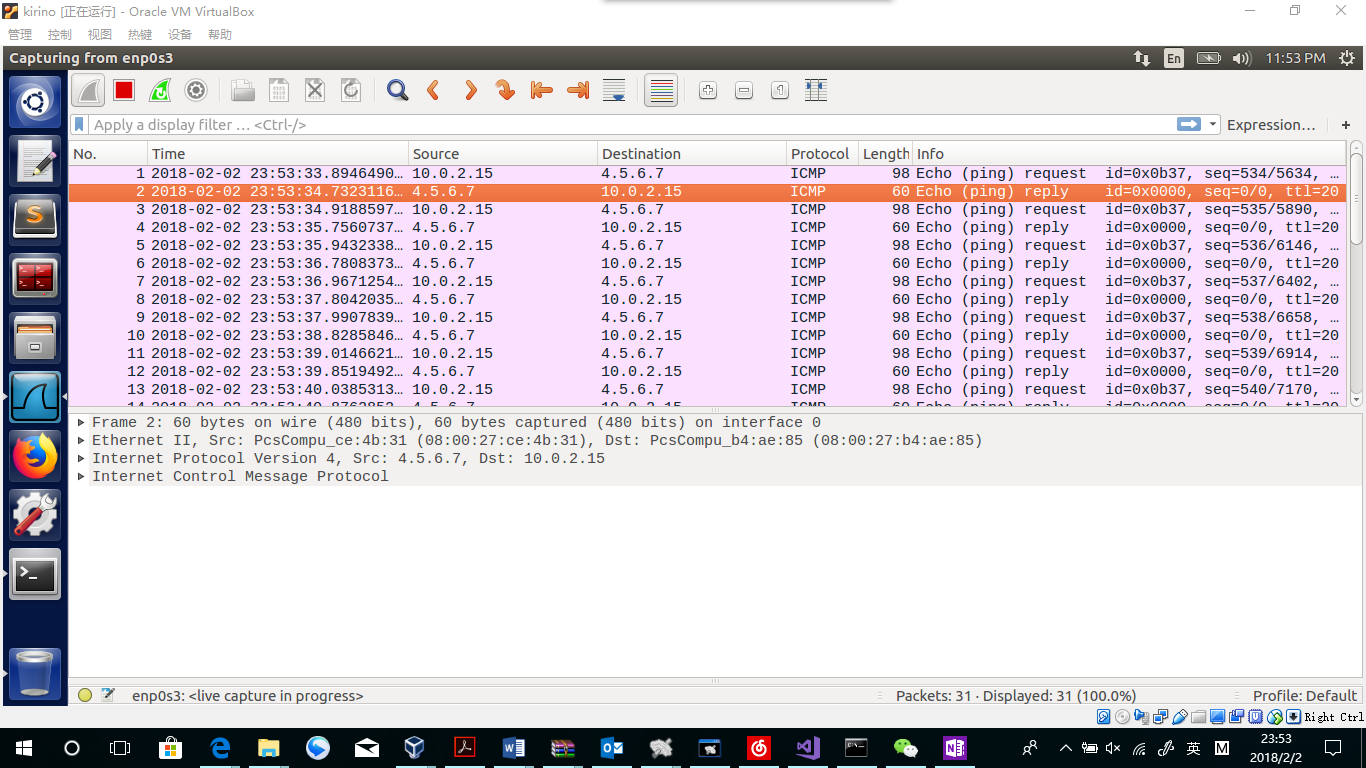
**of whether machine X is alive or not, the ping program will always receive a reply, indicating that X is alive. You need to write such a program, and include screendumps in your report to show that your program works. Please also attach the code (with adequate amount of comments) in your report.**

****



Ping in “kirino”,and see no feedback.

run in “Ayase” seems packet has been transported to “kirino”



In “kirino”’s wireshark, icmp received.

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

Yes. If the value is smaller than the size, some information may be lost. If the value is larger than the size, space area will be filled with”0”s.

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

No I do not have to. Using the raw socket programming, the calculation of the checksum for the IP header would be done automatically by the system.

**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?**

The program will fail until the new raw packet is created.

*int sock = socket(AF\_INET, SOCK\_RAW, IPPROTO\_RAW);*

This is because builing headers is only a series of usage of structure, but while constructing headers and data to a socket, the operating system will deny this operation for the security reasons. That is why we need the root privilege.