**COSC 734-101**

**Assignment: Packet Snifﬁng, Spooﬁng Lab and ARP Cache Poisoning**

**Due on April 6, 2017**

*Copyright c 2006 - 2010 Wenliang Du, Syracuse University. The development of this document is funded by the National Science Foundation’s Course, Curriculum, and Laboratory Improvement (CCLI) program under Award No. 0618680 and 0231122. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation. A copy of the license can be found at http://www.gnu.org/licenses/fdl.html.*

**1. Overview**

Packet snifﬁng and spooﬁng are the two important concepts in network security; they are two major threats in network communication. Being able to understand these two threats is essential for understanding security measures in networking. There are many packet snifﬁng and spooﬁng tools, such as Wireshark, Tcpdump, Netwox, etc. Some of these tools are widely used by security experts, as well as by attackers. Being able to use these tools is important for students, but what is more important for students in a network security course is to understand how these tools work, i.e., how packet snifﬁng and spooﬁng are implemented in software.

The objective of this lab is for students to master the technologies underlying most of the snifﬁng and spooﬁng tools. Students will play with some simple sniffer and spooﬁng programs, read their source code, modify them, and eventually gain an in-depth understanding on the technical aspects of these programs. At the end of this lab, students should be able to write their own snifﬁng and spooﬁng programs.

**2 Tasks**

**2.1 Task 1: Writing Packet Snifﬁng Program**

Sniffer programs can be easily written using the pcap library. With pcap, the task of sniffers becomes invoking a simple sequence of procedures in the pcap library. At the end of the sequence, packets will be put in buffer for further processing as soon as they are captured. All the details of packet capturing are handled by the pcap library. Tim Carstens has written a tutorial on how to use pcap library to write a sniffer program. The tutorial is available at http://www.tcpdump.org/pcap.htm. In this task, you need to read the tutorial, play with the program sniffex included in the tutorial, read the source code sniffex.c, and solve the following problems:

*NOTE: When compile the sniffex.c code, you have to add “–lpcap” in the end of the compile command.*

**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.

Sniffer programs will need to utilize pcap library calls. The ones used in “sniffex.c” are the following:

Pcap\_lookupnet - determines IPv4 network number and associated mask

Pcap\_open\_live - used to obtain capture device

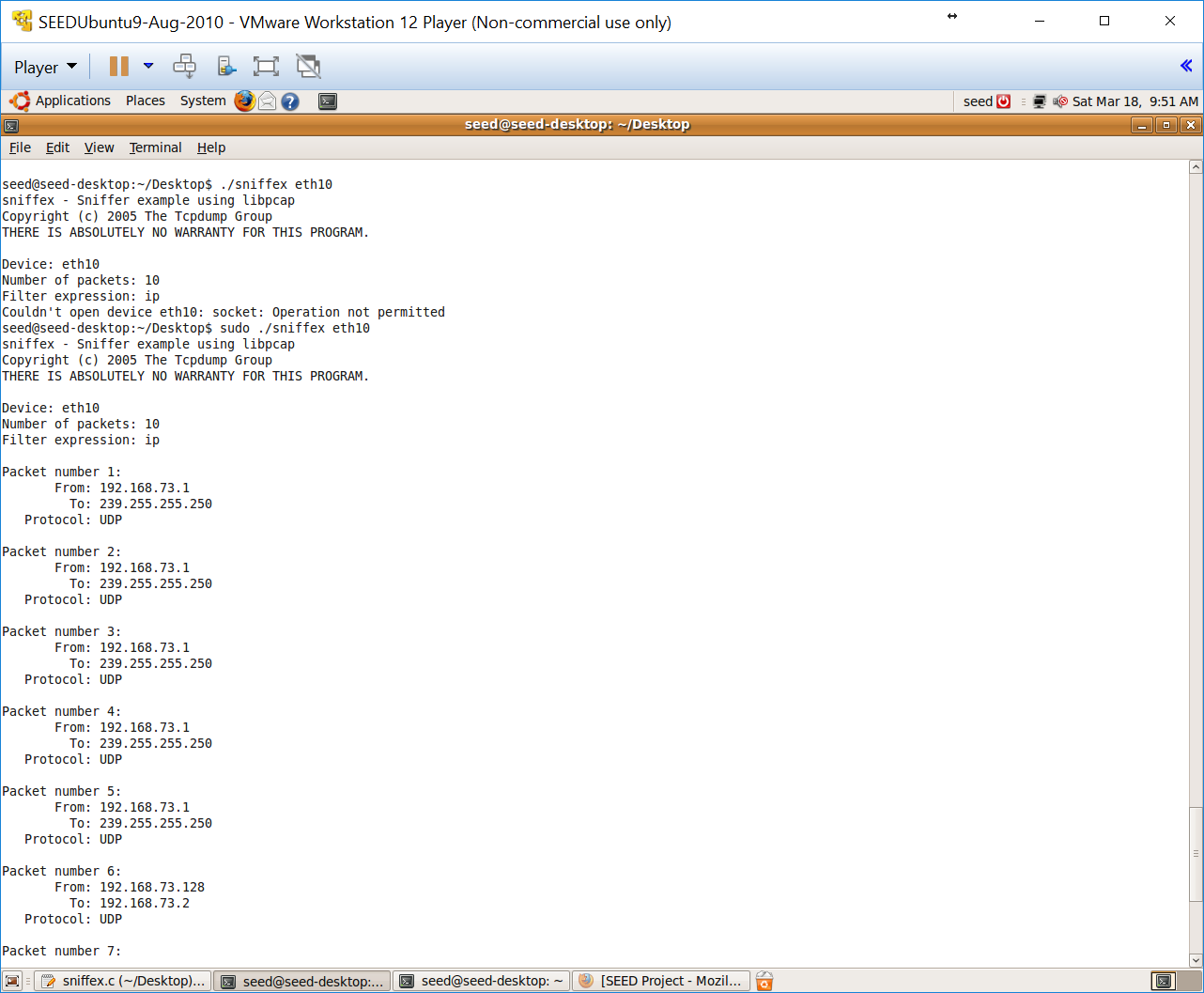
Pcap\_datalink - returns link layer type – makes sure it’s Ethernet

Pcap\_setfilter - applies a filter to a bpf\_program struct

Pcap\_loop - processes packets from capture until a certain number of packets are processed

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

You need root privilege in order to open the socket.



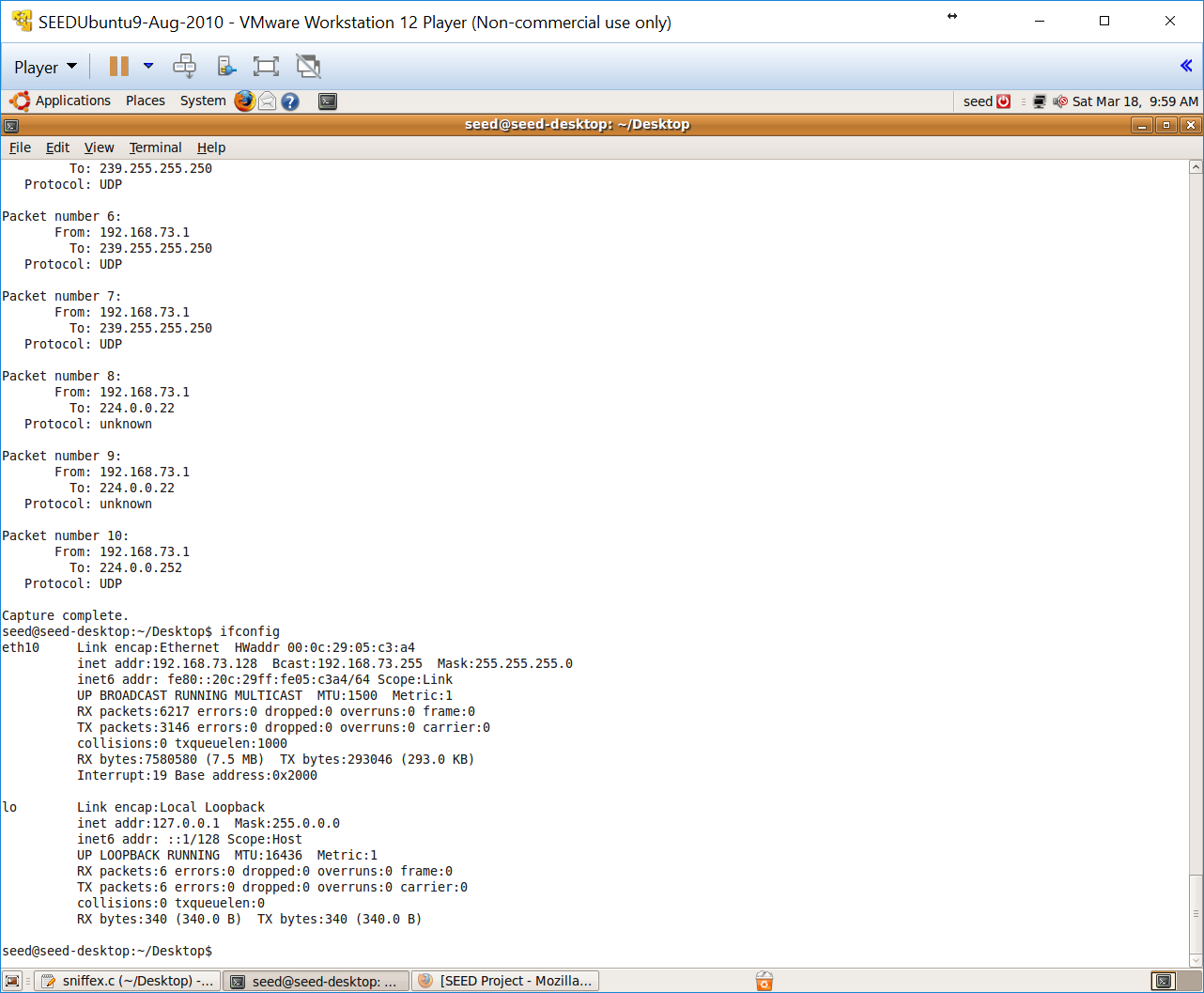
The point of failure is after the printing of the capture info.

The library call that returns an error code is pcap\_open\_live.

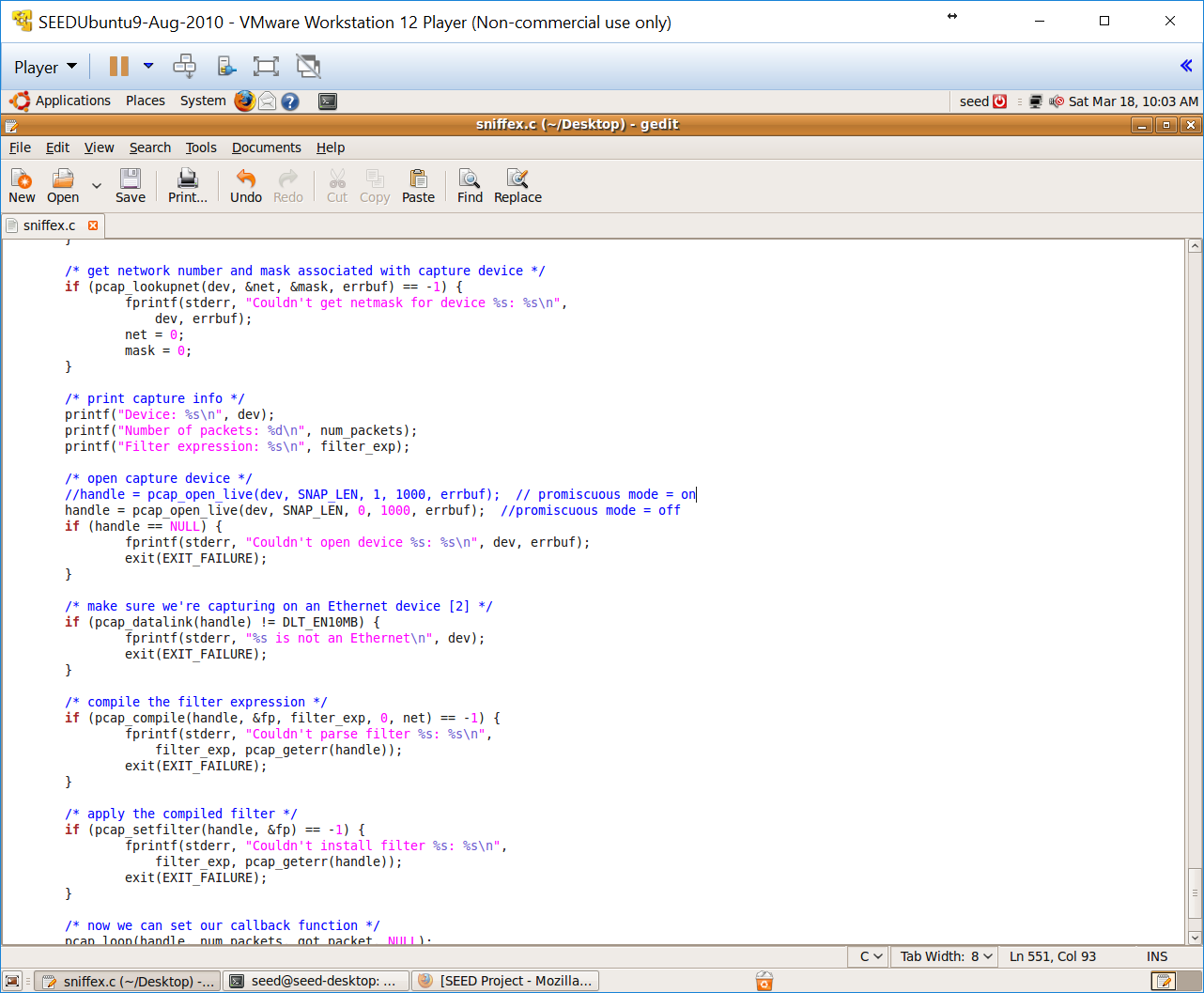
**Problem 3:** Please turn on and turn off the promiscuous mode in the sniffer program. Can you demonstrate the difference when this mode is on and off? Please describe how you demonstrate this.

Promiscuous mode set to True:

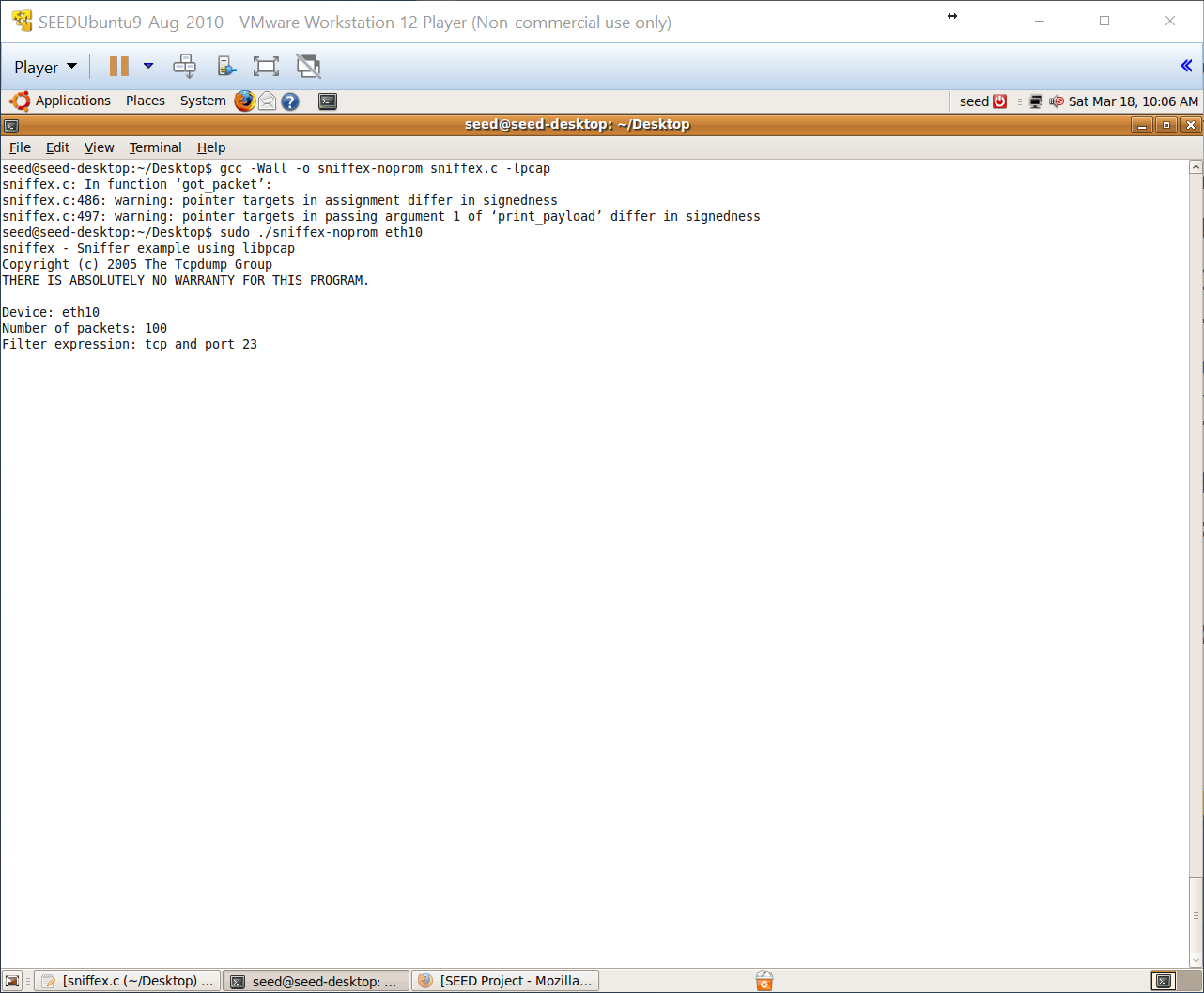
Notice that the VM has an IP address of 192.168.73.128. It has recorded packets from 192.168.73.1 to another destination.



Promiscuous mode set to False/Off in sniffex.c:



Packets collected:

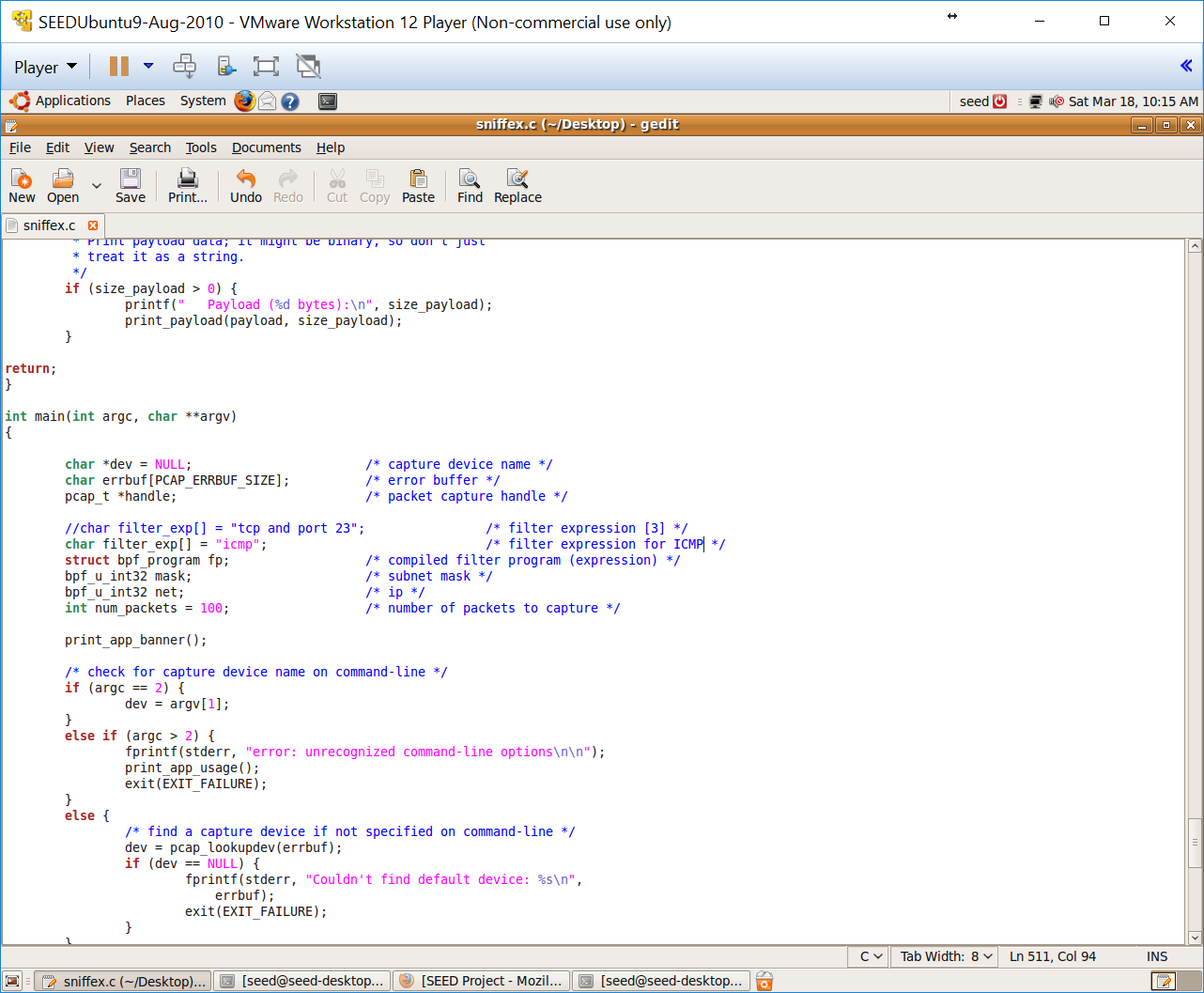


There have been no packets collected in non-promiscuous mode.

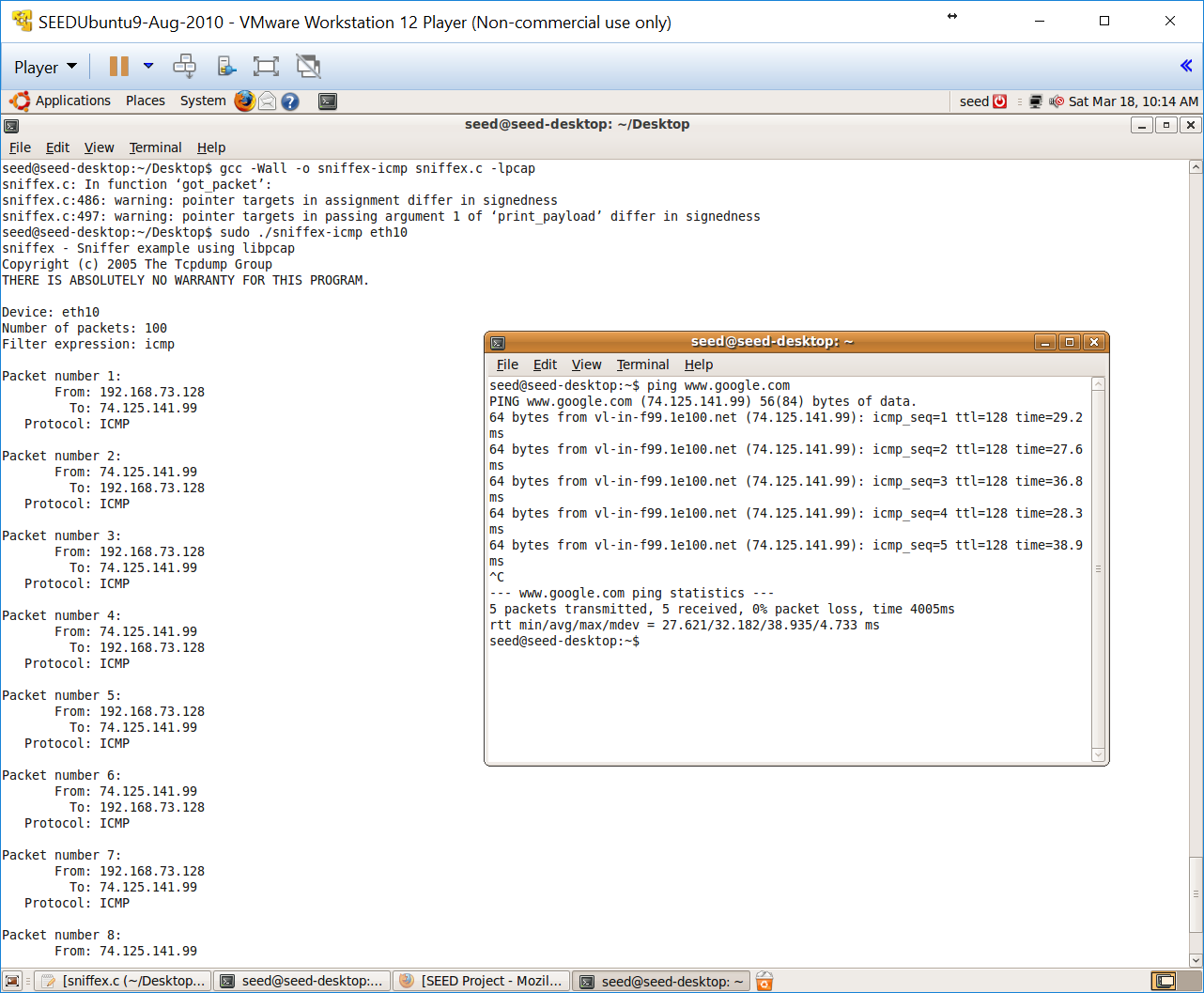
**Problem 4**: Please write ﬁlter expressions to capture each of the followings. In your lab reports, you need to include screen dumps to show the results of applying each of these ﬁlters.

* Capture the ICMP packets between two specific hosts.

Modified filter expression from the default “tcp and port 23” to “icmp”. See source code:



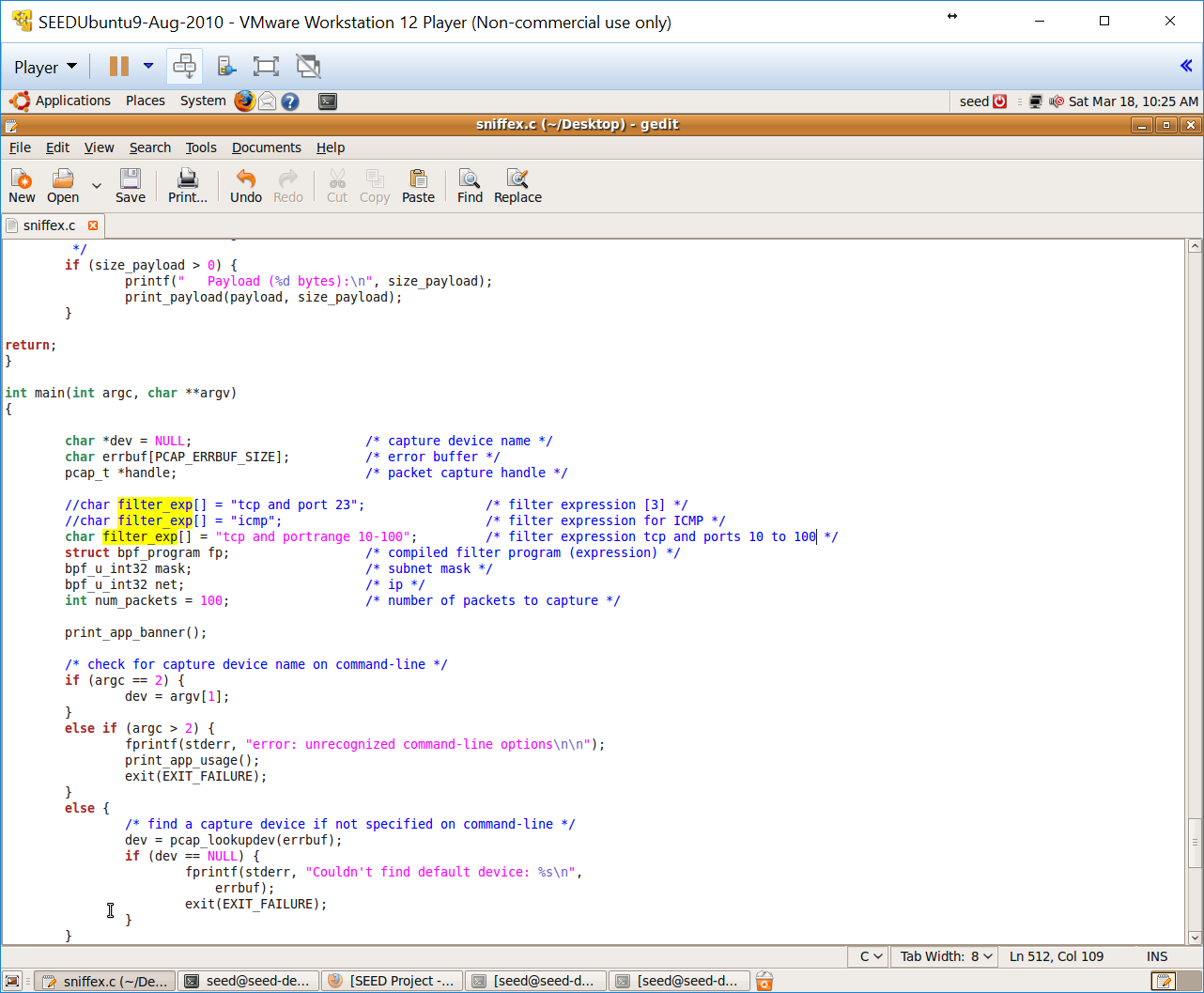
I pinged for “www.google.com” and waited to see the ICMP responses. The destination IP addresses was “www.google.com”'s public facing IP address.



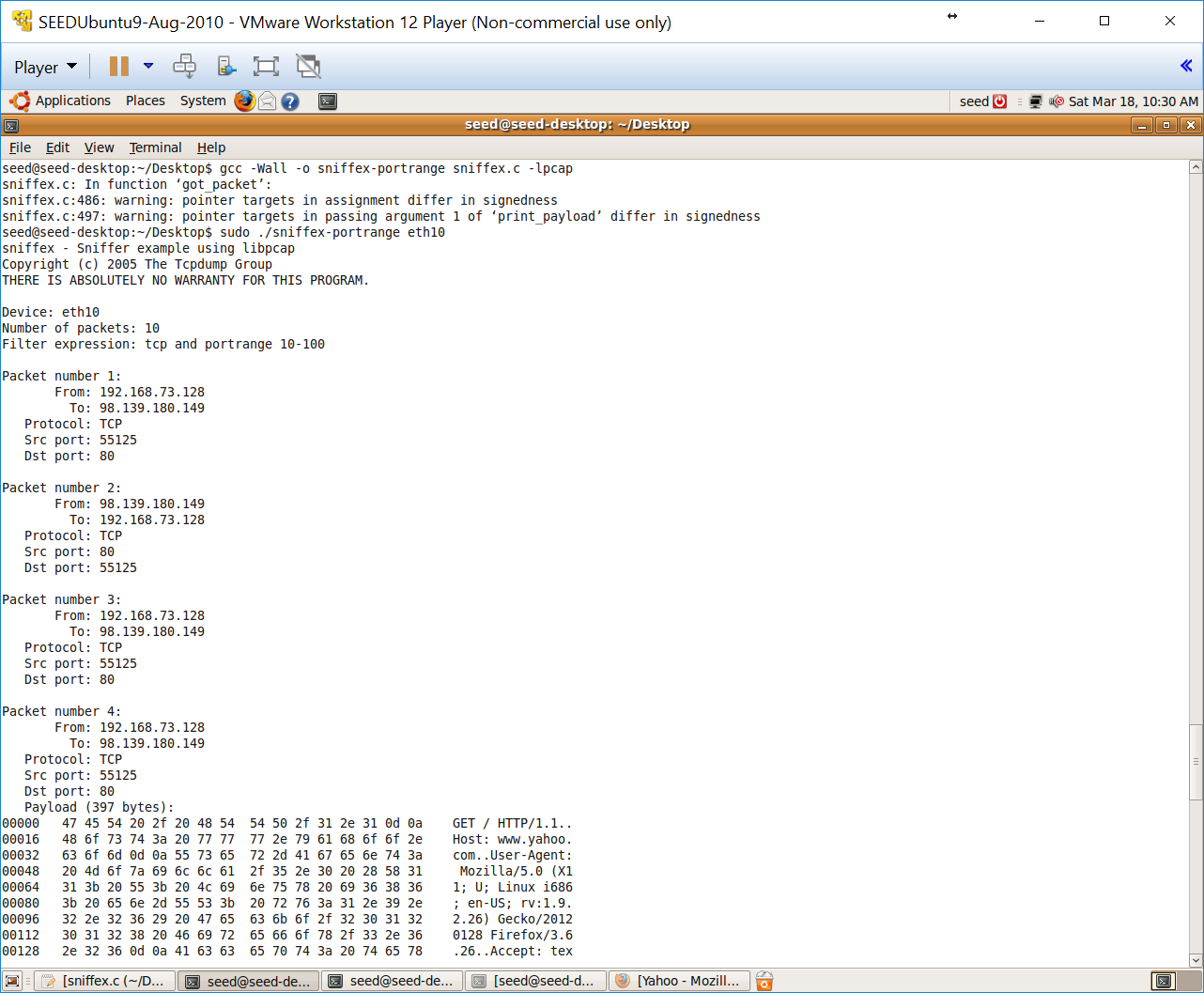
* Capture the TCP packets that have a destination port range from to port 10 - 100.

This one was slightly trickier to write. One needs to understand the syntax as your filter expression will need to be parsed. I first ‘man’ “pcap\_compile” which tells you to see ‘pcap-filter’ for the syntax of the string. ‘pcap-filter’ provides an example which indicates that a part of the expression to be used should be “portrange 10-100”.

See source code:



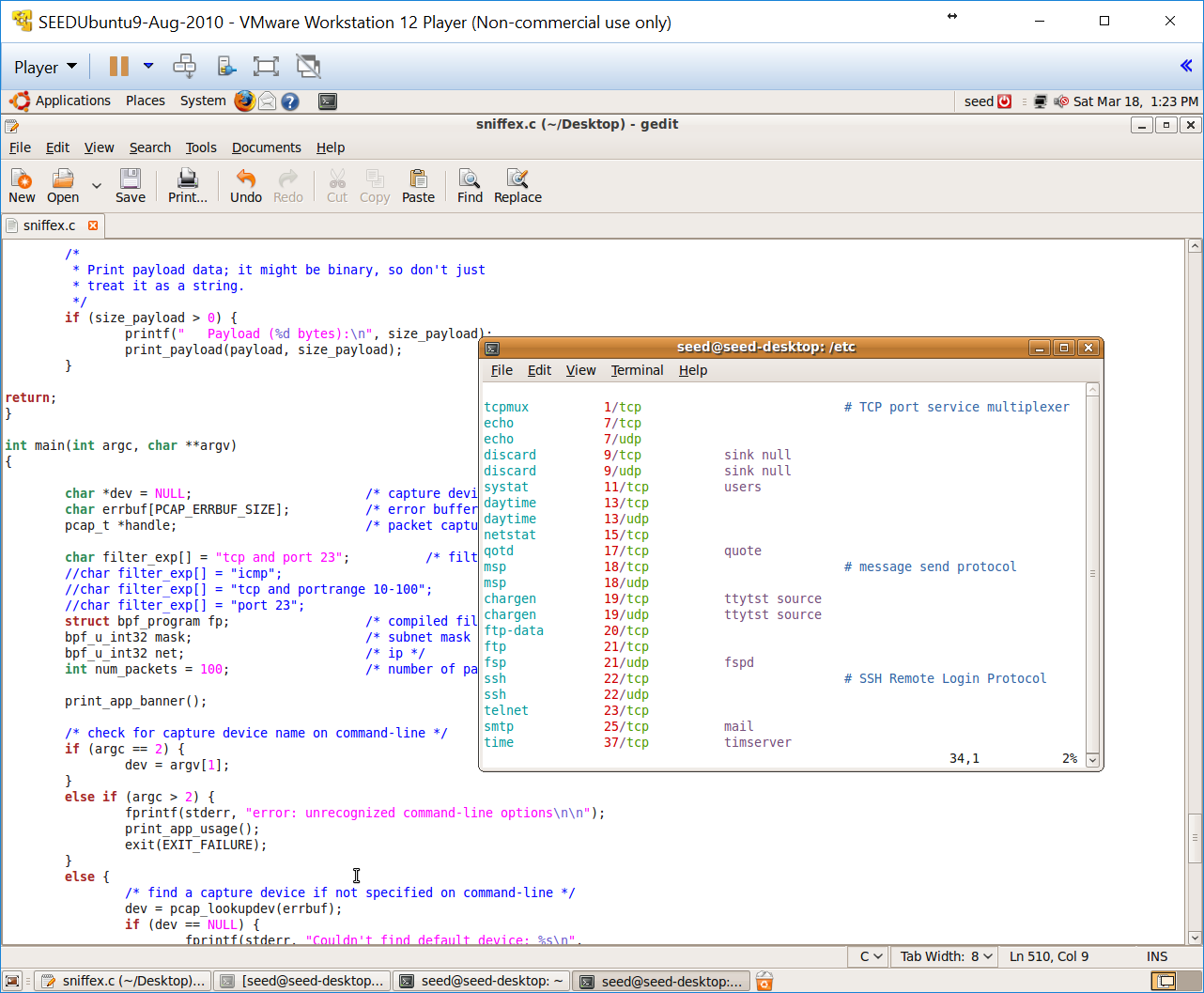
I visited “www.yahoo.com” to cause some traffic to occur on Port 80. Here’s the results:



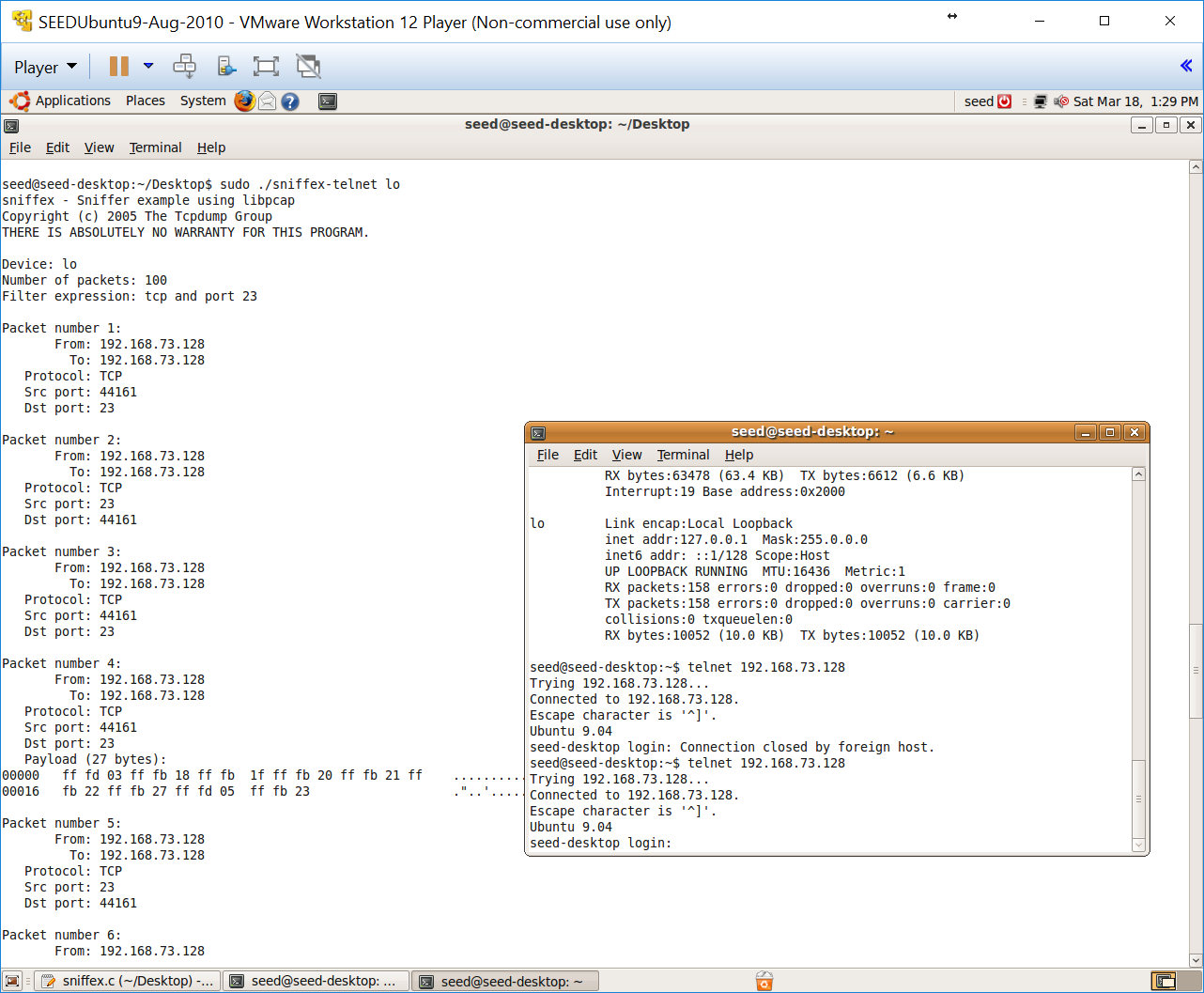
**Problem 5:** Please show how you can use sniffex to capture the password when somebody is using telnet on the network that you are monitoring. You may need to modify the sniffex.c a little bit if needed. You also need to start the telnetd server on your VM. If you are using our pre-built VM, the telnetd server is already installed; just type the following command to start it.

% sudo service openbsd-inetd start

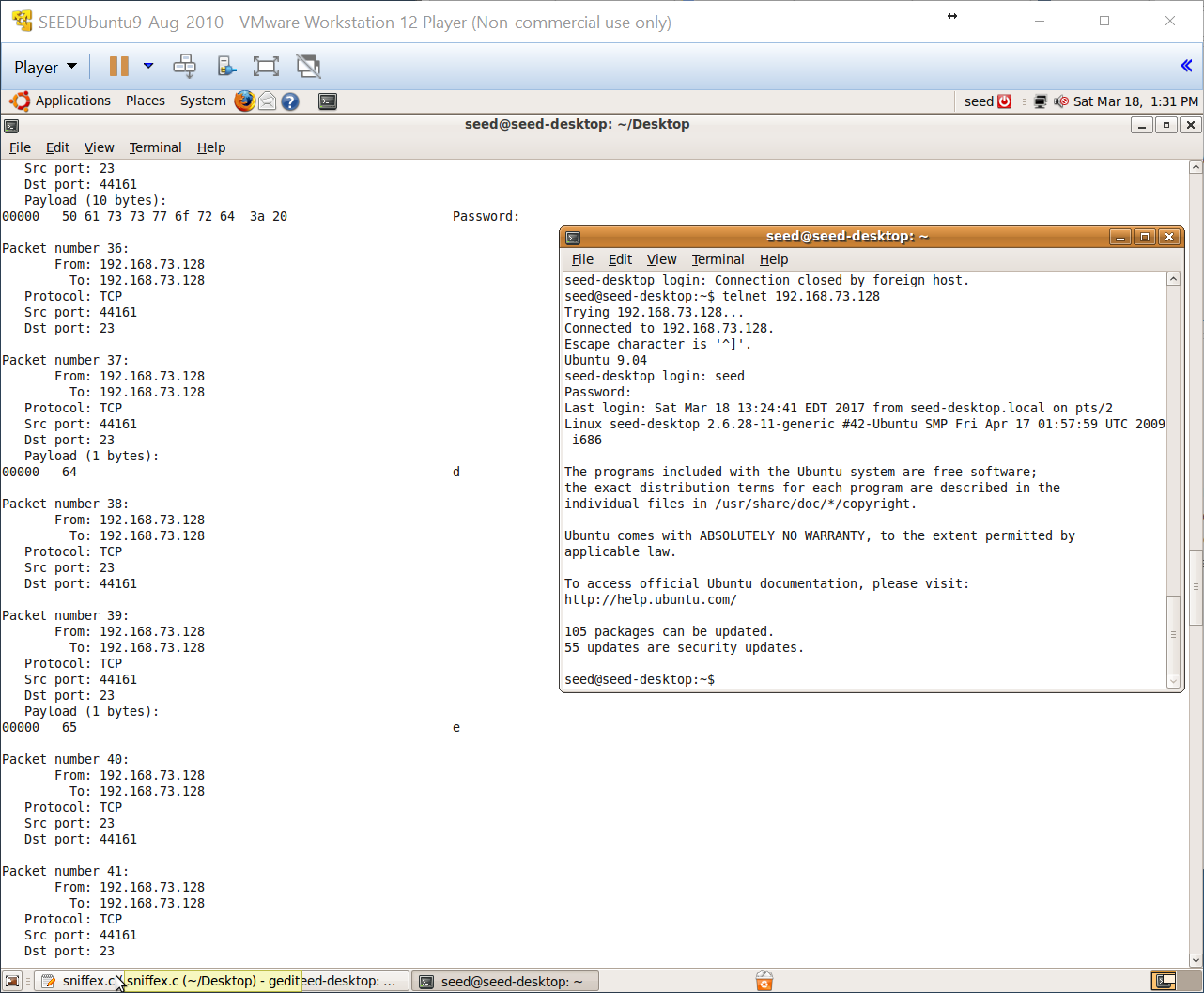
Telnet usually uses TCP port 23 based on the /etc/services file so we will leave the default “sniffex.c” filter.



During the course of this, I realized I had to listen to the “loop back” interface – lo – instead of eth10 in order to capture packets for telnet since the telnet server is being hosted locally.



Upon logging into telnet, you will see that the password is sent via clear text to the telnet server. However, each character of the password is an individual packet.



The password used here is the default VM password: dees

**2.2Task 2: ARP cache poisoning**

The ARP cache is an important part of the ARP protocol. Once a mapping between a MAC address and an IP address is resolved as the result of executing the ARP protocol, the mapping will be cached. Therefore, there is no need to repeat the ARP protocol if the mapping is already in the cache. However, because the ARP protocol is stateless, the cache can be easily poisoned by maliciously crafted ARP messages. Such an attack is called the ARP cache poisoning attack.

In such an attack, attackers use spoofed ARP messages to trick the victim to accept an invalid MAC-to-IP mapping, and store the mapping in its cache. There can be various types of consequences depending on the motives of the attackers. For example, attackers can launch a DoS attack against a victim by associating a nonexistent MAC address to the IP address of the victim’s default gateway; attackers can also redirect the trafﬁc to and from the victim to another machine, etc.

In this task, you need to demonstrate how the ARP cache poisoning attack work. Several commands can be useful in this task. In Linux we can use command arp to check the current mapping between IP address and MAC.

Once the mapping between a MAC address and an IP address is resolved as the result of executing the ARP protocol, the mapping will be cached. The cache can be poisoned by malicious crafted ARP messages.

**Problem 5**: In this task, you need to do the following:

1. Show the ARP table before attack
2. Show the ARP table after attack
3. Try the DoS attack if interesting (not required)

**Hints:**

* To conduct this lab, you need to have at least 2 machines: victim and the attacker.
* The ARP cache poisoning is only successful when the IP entry already exists in victim’s ARP catch. Otherwise the attack will not going to succeed.
* Attack tool to use: netwag 33
* If you want to conduct the DoS attack, you can use this tool: netwag 72

**3. Submission**

You need to submit a detailed lab report to describe what you have done and what you have observed; you also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits.