**Problem 1: Calls within sniffex**

These are the calls within the sniffex.c code

* 1. pcap\_lookupdev: Finds capture device for sniffing
  2. pcap\_lookupnet: Returns mask and network number for capture device
  3. pcap\_open\_live: Starts sniffing process on the desired capture device
  4. pcap\_datalink: Returns the type of capture device
  5. pcap\_compile: Compiles filter expression in a regular string such that the filter can be set
  6. pcap\_setfilter: Sets the filter from pcap\_compile
  7. pcap\_next: Sniffs a single packet at a time
  8. pcap\_loop: Continuous sniffing
  9. pcap\_freecode: Free memory allocated by pcap\_compile
  10. pcap\_close: Terminate the session

**Problem 2: Root privilege and sniffex**

Root privilege is required because sniffex needs to access a network device. A non-root user is unable to do this. This is the code that causes the failure:

dev = pcap\_lookupdev(errbuf);

if (dev == NULL) {

fprintf(stderr, "Couldn't find default device: %s\n",

errbuf);

exit(EXIT\_FAILURE);

}

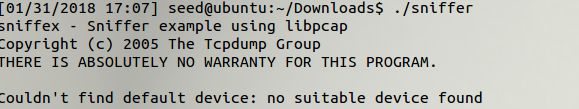


Figure : Trying to run sniffer without root privilege.

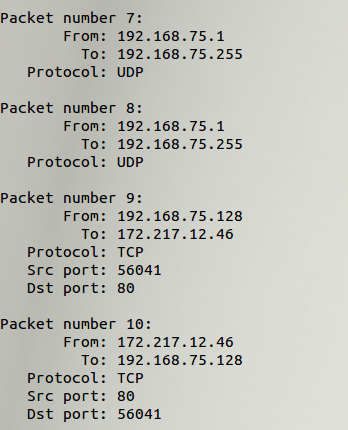


Figure : Sniffer running with root. Note promiscuous mode is enabled (as is the default). The last packets are only sniffed because of promiscuous mode

**Problem 3: Promiscuous Mode**

Promiscuous mode allows the sniffer to pass all traffic from a network controller, not just the traffic it’s supposed to receive. The third argument of the function call “pcap\_open\_live(dev, SNAP\_LEN, 1, 1000, errbuf);” enables and disables promiscuous mode. By default, it is set to 1, enabling promiscuous mode. Changing this to 0 disables it.

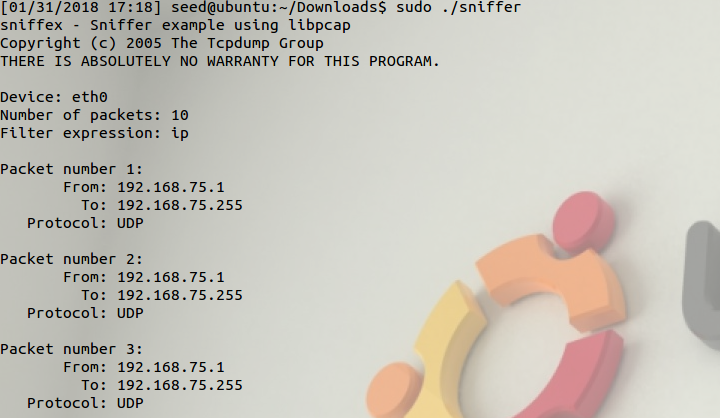


Figure : Sniffer without promiscuous mode. Note how the from/to addresses are the same and only

**Problem 4: Filter Expressions**

Our first task in this part is to write a filter such that we only get ICMP packets from two hosts. To do this, I ran another terminal and used the command “ping 8.8.8.8.”

Sniffex has a string (character array) called filter\_exp[], which can be modified to filter particular packets. In this case, I used the string “icmp and (src host 10.253.18.28 and dst host 8.8.8.8) or (src host 8.8.8.8 and dst host 10.253.18.28)” to filter ICMP packets between 8.8.8.8 and 10.253.18.28 (my virtual machine).

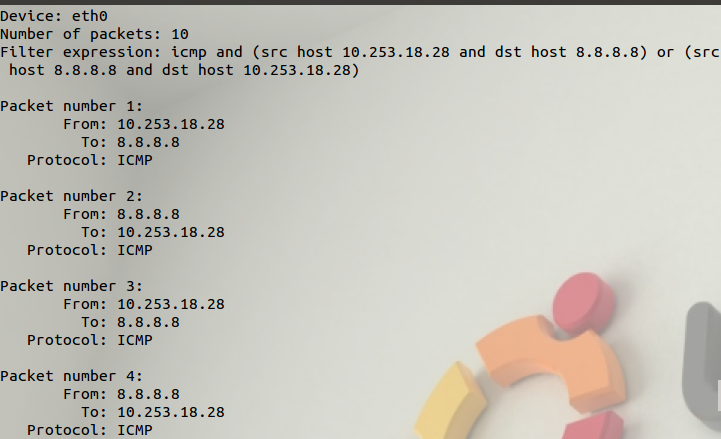


Figure : ICMP filtering between two hosts. I ran the command “ping 8.8.8.8” in order to create ICMP packets to be sniffed.



Figure : TCP filtering port range 10-100. I connected to a website using HTTP in order to generate TCP packets (on port 80).

**Problem 5: Telnet and Password Sniffing**

We can use the filter expression “tcp port 23” and then telnet into the VM from another UNIX machine and sniff the password.

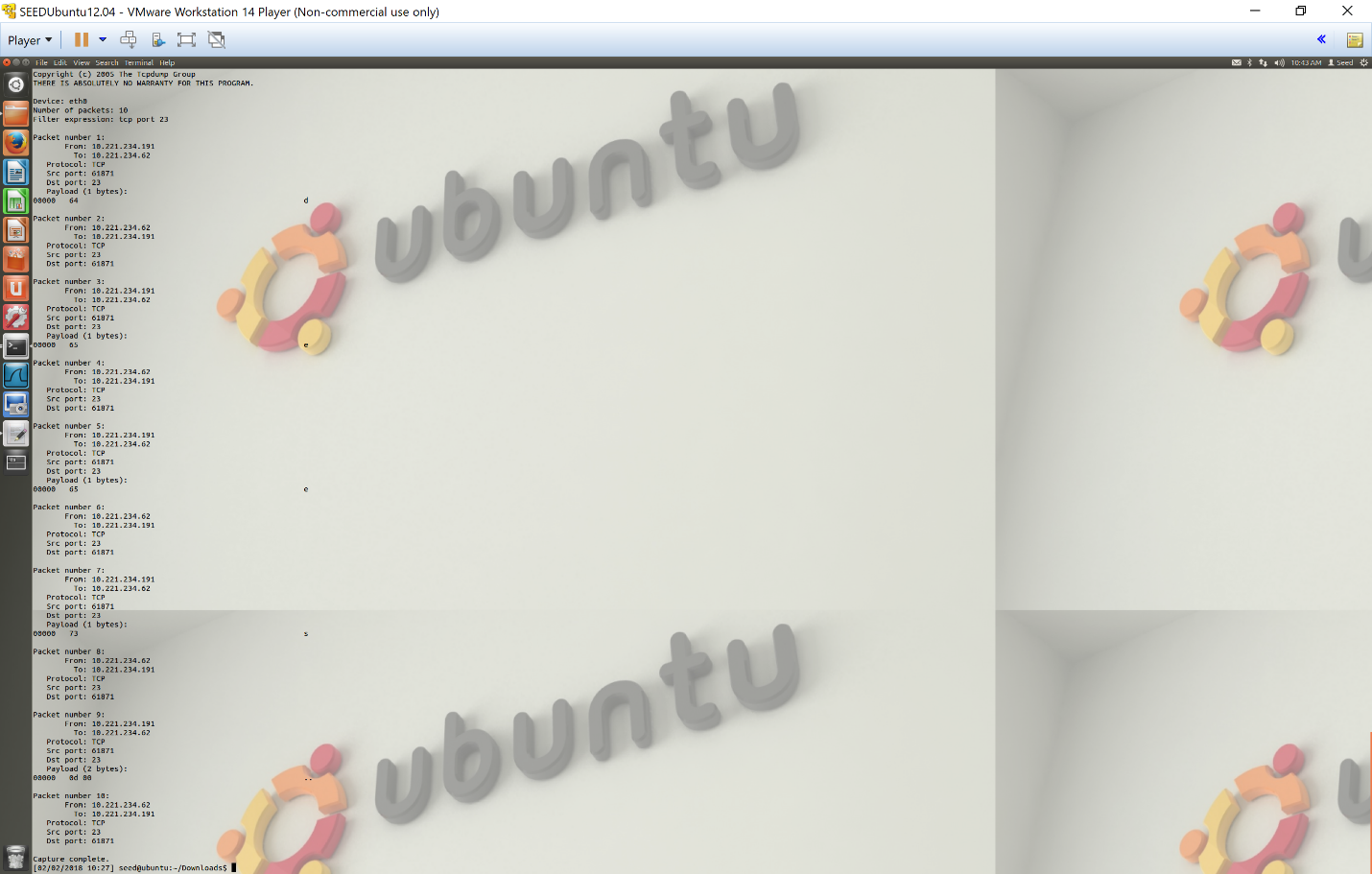


Figure : By running telnet from another UNIX machine and telling sniffex to only look at TCP packets on port 23, we can sniff the password used to log in to the VM.

Furthermore, we can also read input from an offline trace file by defining the handle with the function pcap\_open\_offline(file\_path, errbuff) in lieu of the function pcap\_open\_live.

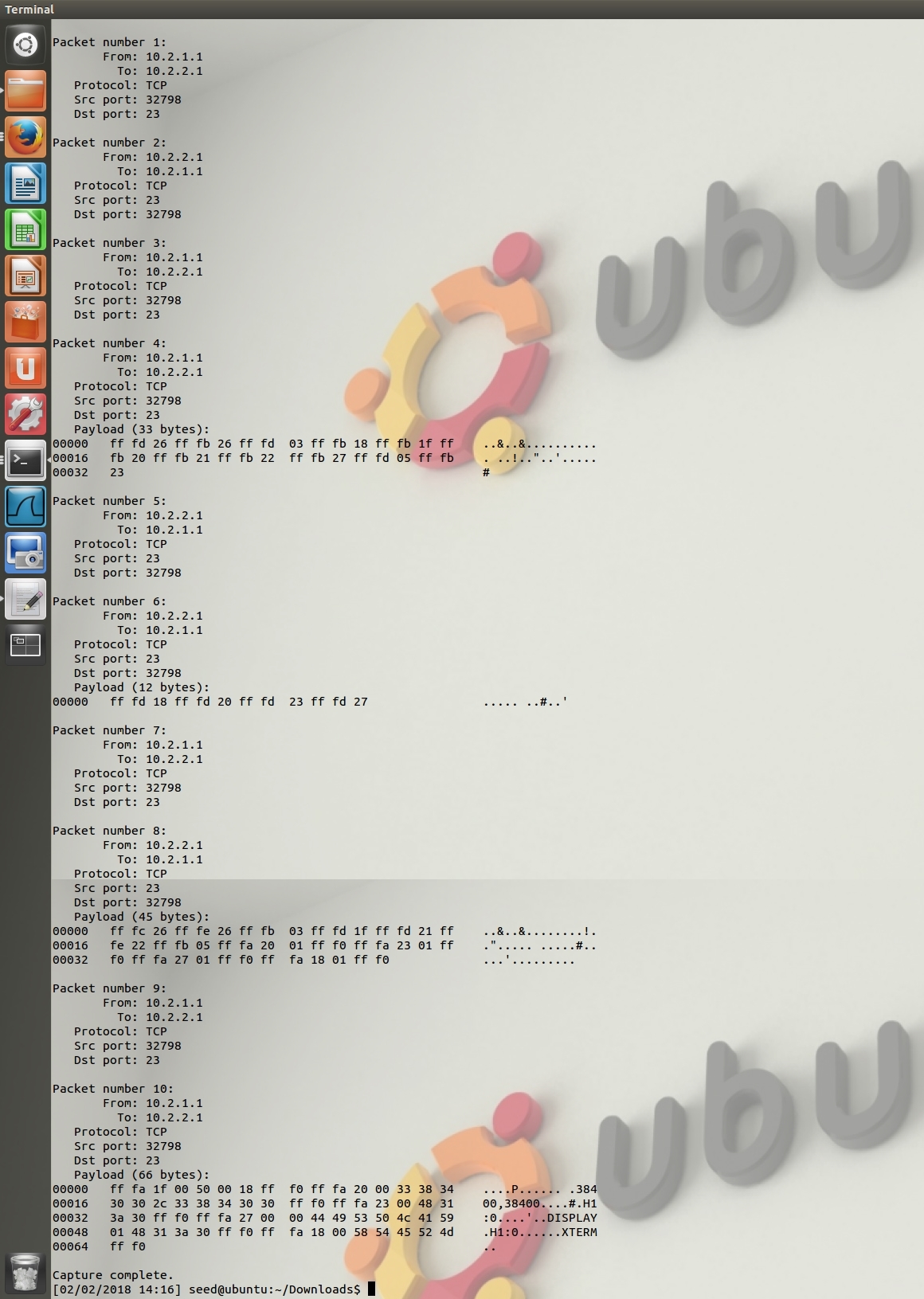


Figure : Telnet from an offline file trace. The file is stored in the same directory as the program, and then through pcap\_open\_offline, we can instruct sniffex to look at the trace file instead of a network device.

**Problem 6: Calls for Packet Spoofing**

Spoofing requires four library calls.

1. Creating a raw socket that help the program inject packets into the network.

Raw sockets allow the direct sending of packets. This is the most important step in spoofing. This is also why you need root privilege to do spoofing.

1. Set the socket option (IP, ICMP, TCP, UDP)

In order to inject our own packets, we have to know the protocols and structures to include.

1. Construct the packet
2. Send the packet through the raw socket made in the previous library calls.

**Problem 7: Root privilege for Spoofing**

We need root privilege in order to create sockets. Otherwise, the OS will set the fields for our protocol headers (IP, ICMP, TCP, UDP). The program will fail to create the sockets necessary for spoofing.

**Problem 8: Sniff and Spoof**

By combining the sniffex.c and a spoofer program, we can create a sniff and spoof program.

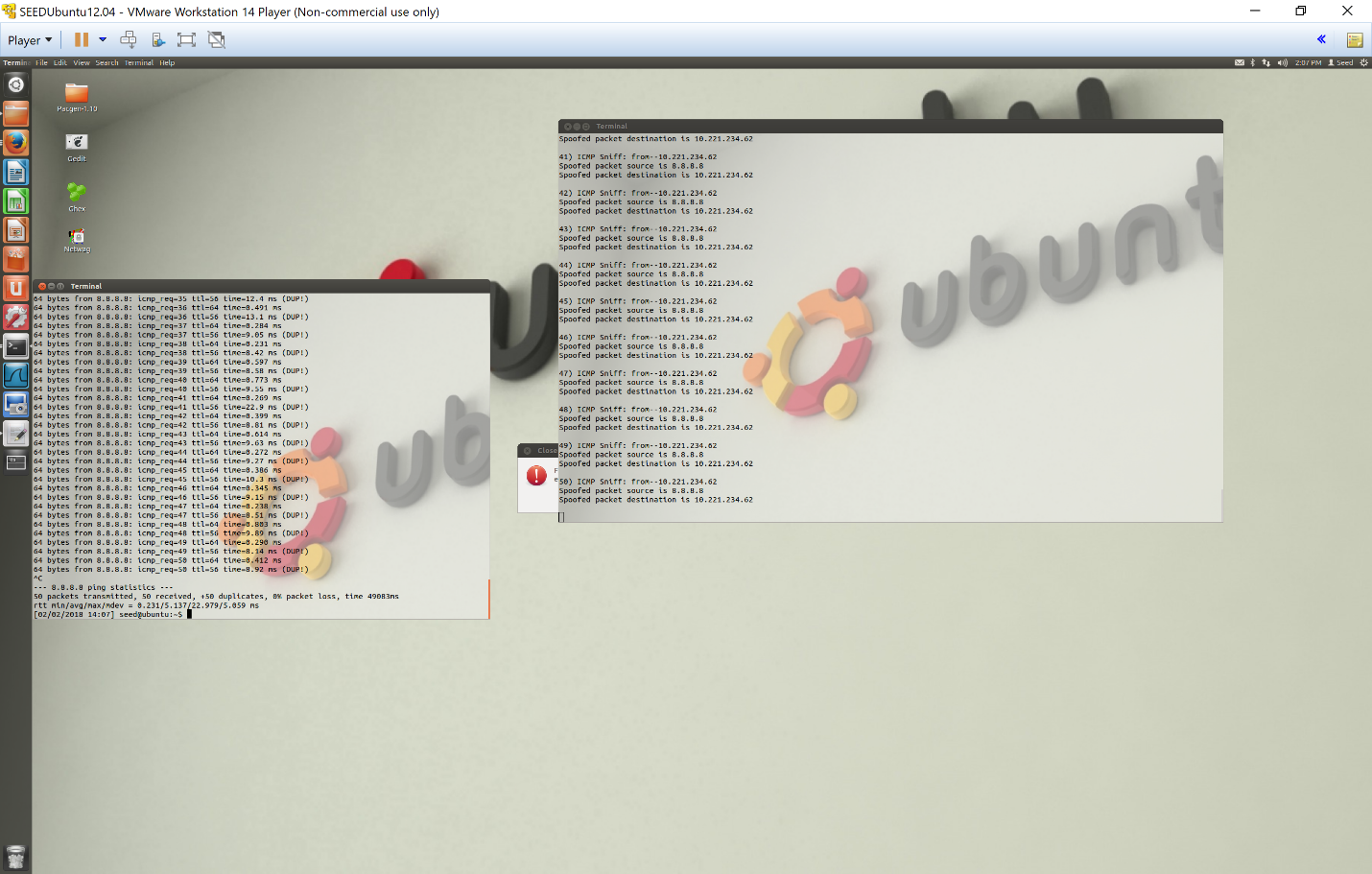


Figure : Sniffing and spoofing in action. Note how there are duplicated packets (DUP!) when running “ping 8.8.8.8” due to spoofing the ICMP packets.