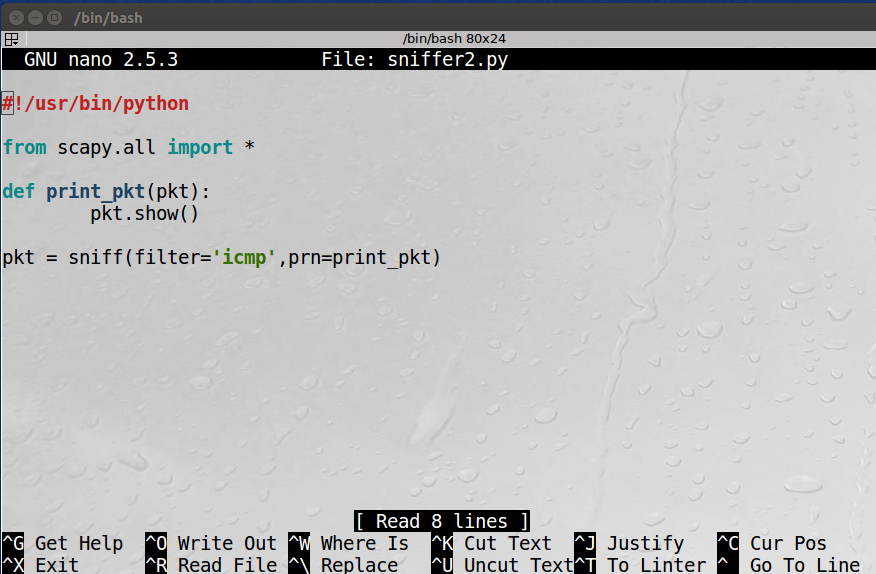
Internet Security - Lab 1 Report

# 1. Using Tools to Sniff and Spoof Packets

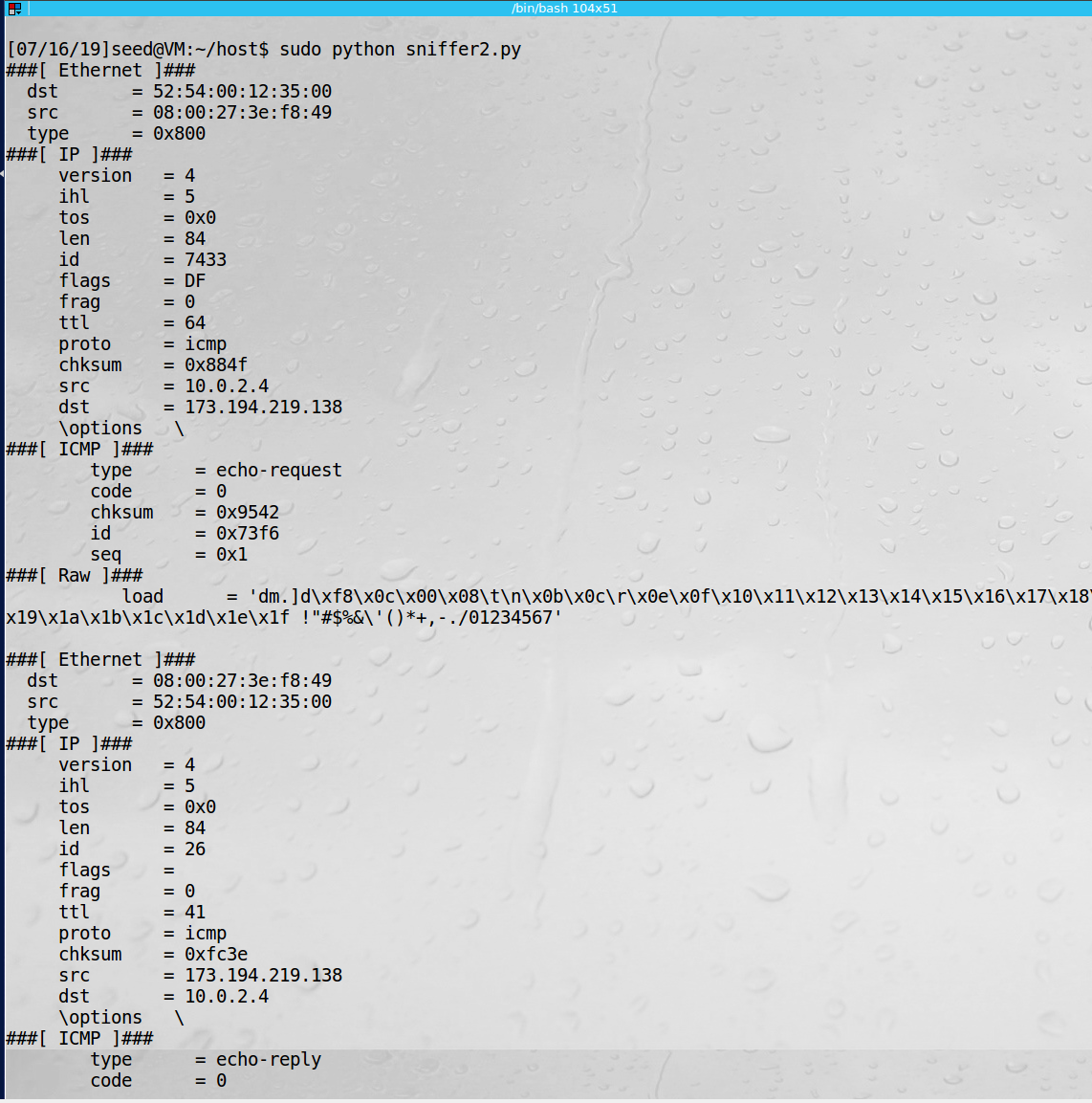
## Task 1.1 Sniffing Packets

### Task 1.1a

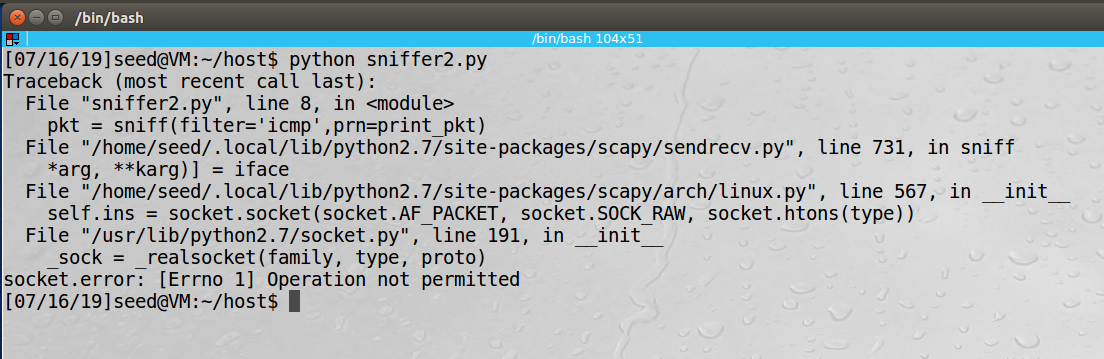
The python code for sniffing ICMP packets:



Results from the packet sniffer after running with root privilege and pinging google.com from the same machine:



Running the program without root privileges:

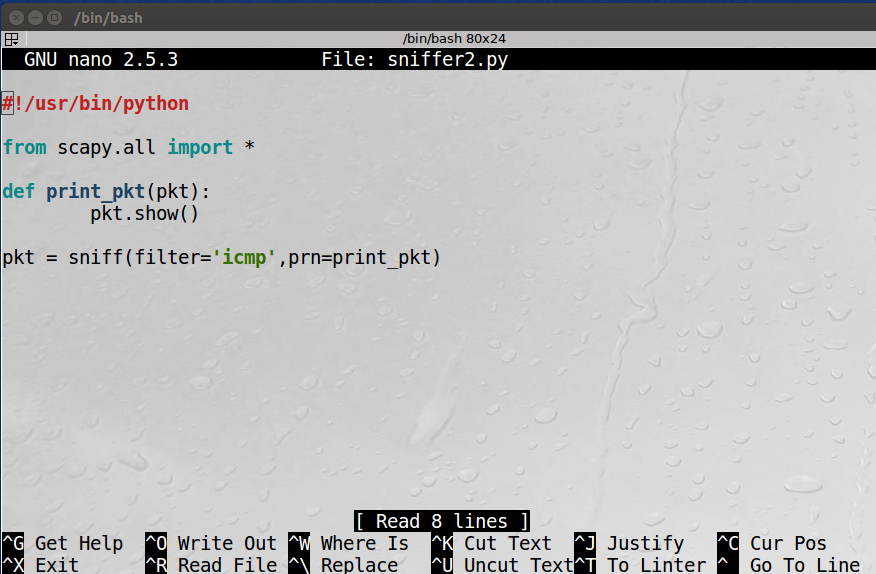


**Observation**: In the screenshots above we demonstrate how to use scapy to sniff network traffic. If we try to run without using the root privileges the program will not run.

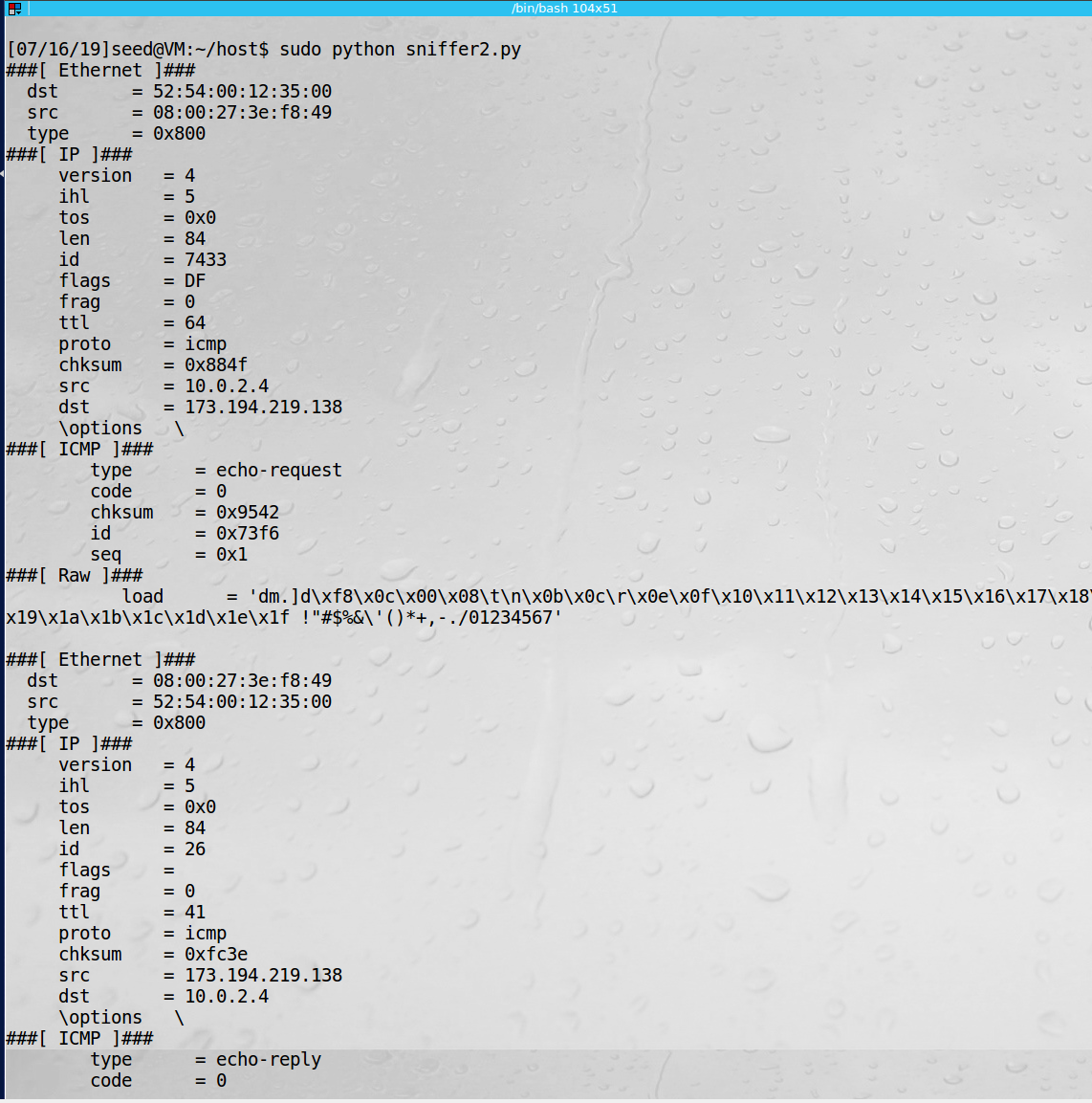
**Explanation:** Scapy is one of the tools makes it easy to sniff network traffic. It requires root privileges to run since you need to put the NIC into promiscuous mode.

### Task 1.1b

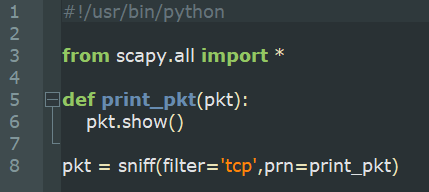
Code for capturing only ICMP traffic:



Output of capturing only ICMP traffic and sending a ping request to google.com:



Code for capturing only TCP traffic:



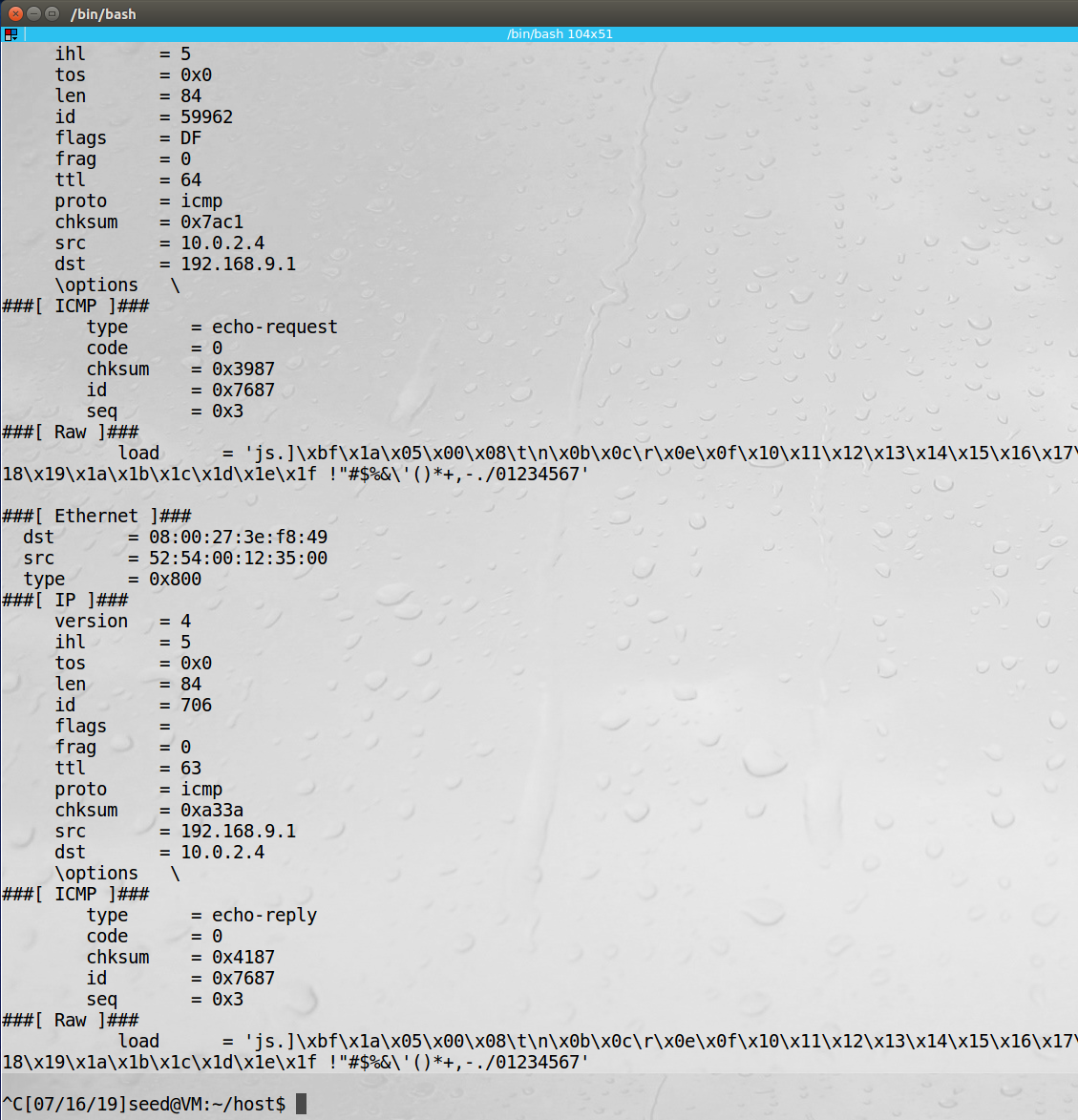
Output from sniffing only TCP packet and opening Firefox browser:



Code for capturing packets only in subnet 192.168.0.0/16:



Capture of packets only to/from 192.168.0.0/16 while pinging 192.168.9.1:

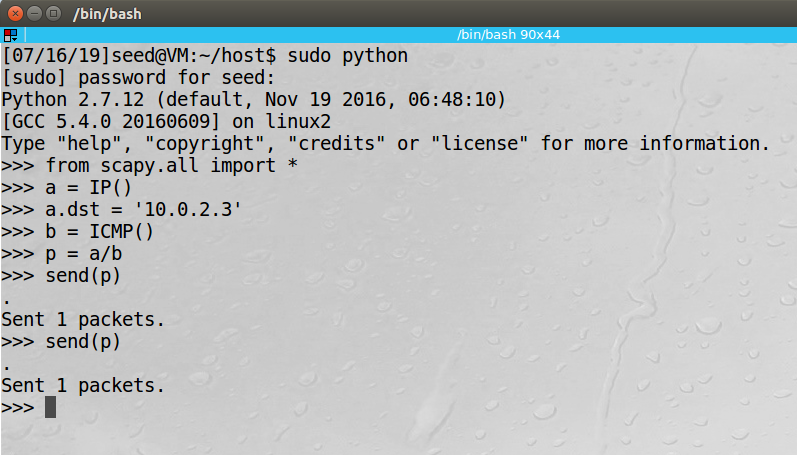


**Observation**: The screenshots above show some of the filters that can be used to capture only specific packets. We first only process ICMP packts, then TCP with a destination port of 23 and then packets only to/from subnet 192.168.0.0/16.

**Explanation:** Scapy allows for various traffic filters to capture only the packets that we are concerned with.

## Task 1.2 Spoofing ICMP Packets

Code for spoofing ICMP packet:



Wireshark output from spoofed packet:

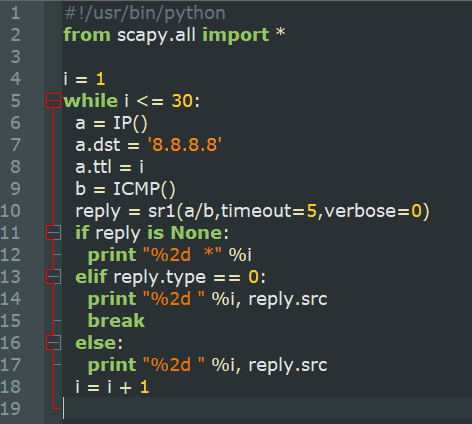


**Observation**: Above we use Scapy to spoof ICMP packets, making it look like the packet came from IP 10.0.2.3 when that was not the case.

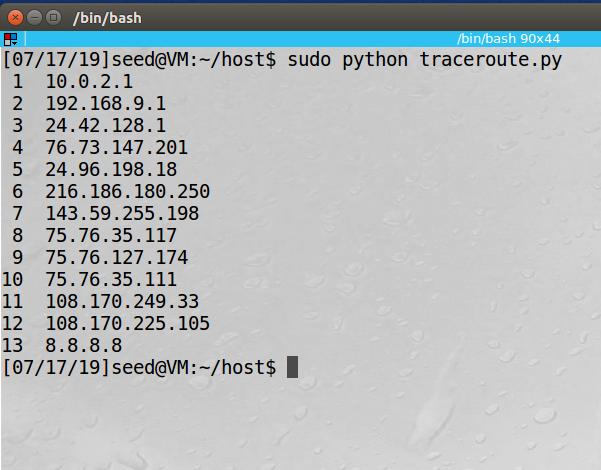
**Explanation:** Scapy make operations like spoofing an ICMP packet trivial. Packets can be spoofed and made to look like a different origin that it actually came from.

## Task 1.3 Traceroute

Code for traceroute to 8.8.8.8:



Output from executing traceroute program to 8.8.8.8:

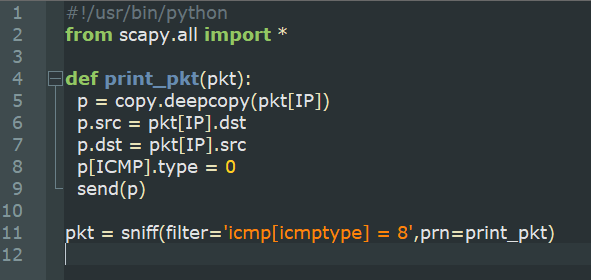


**Observation**: In the screenshot above we can see how traffic is routed from the host to the Google DNS server (8.8.8.8)

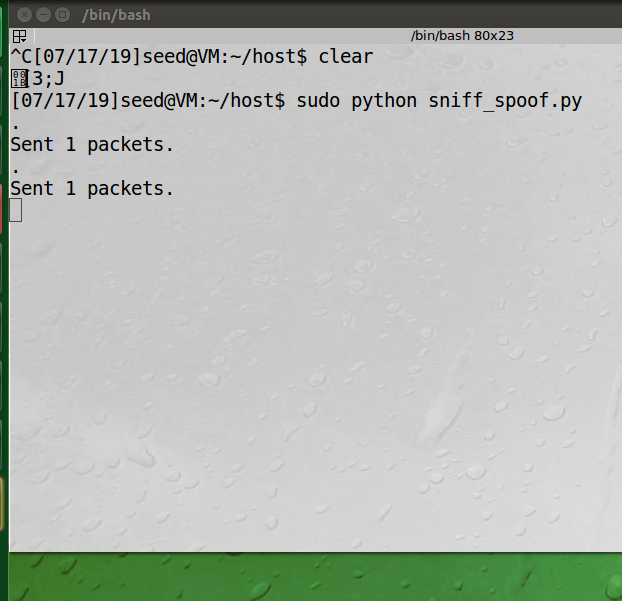
**Explanation:** Scapy can also be used for traceroute like functionality to show the route that was taken between the host and the server.

## Task 1.4 Sniffing and-then Snooping

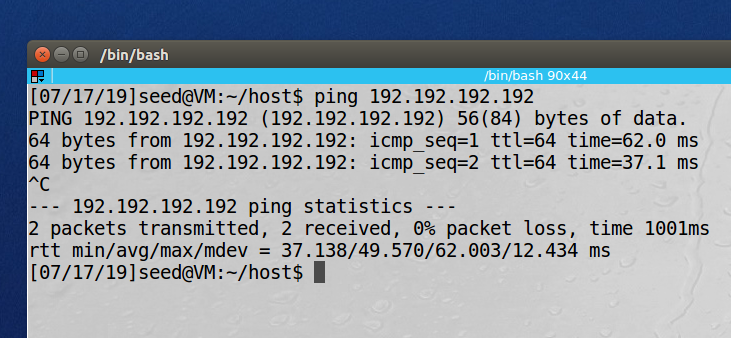
Code for Sniffing and Spoofing program:



VM B running sniff and spoof program:



VM A requesting ping from 192.192.192.192, and receiving the spoofed reply from VM B:



**Observation**:

**Explanation:**

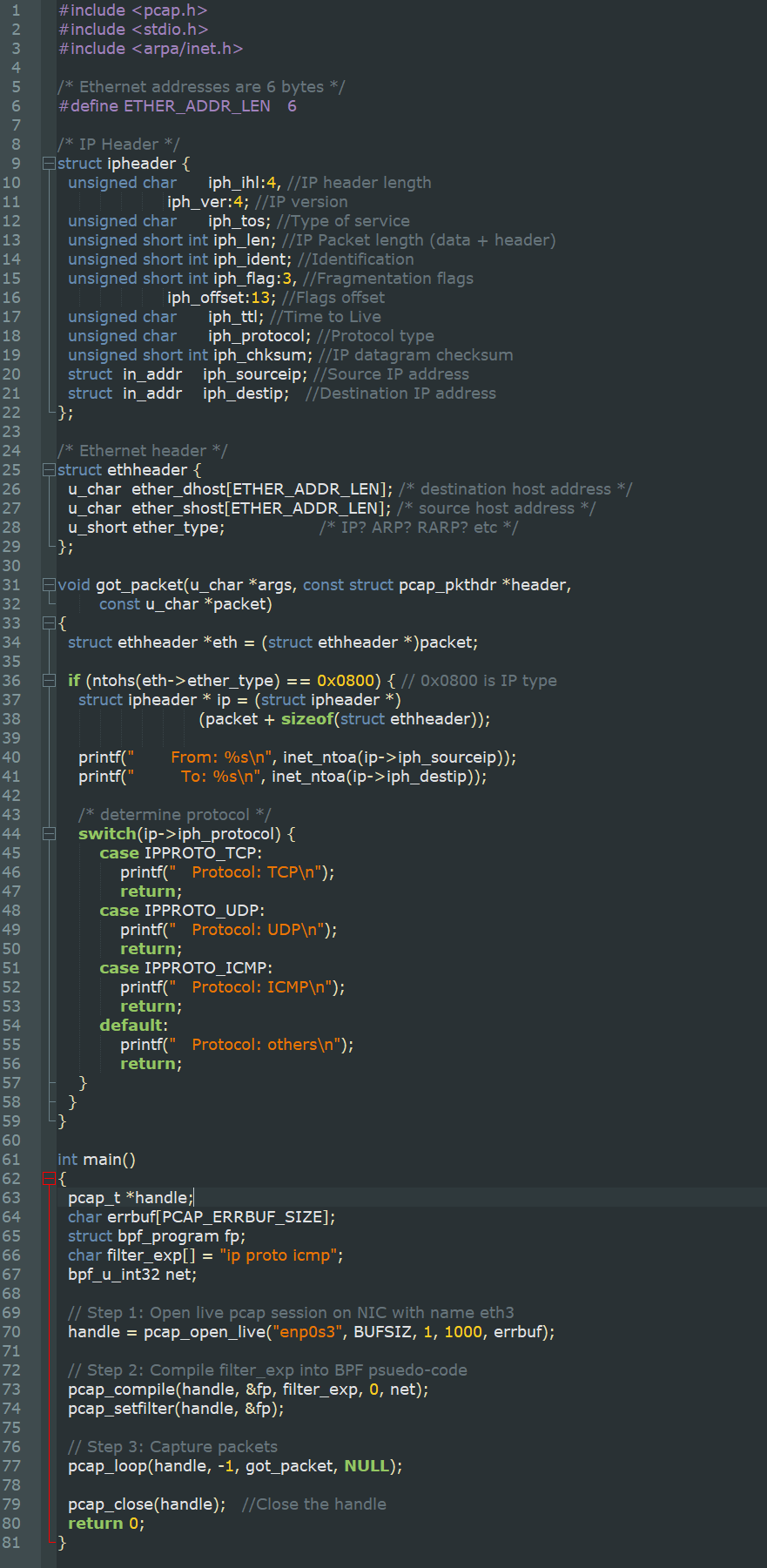
# 2. Writing Programs to Sniff and Spoof Packets

## 2.1 Writing a Packet Sniffing Program

### 2.1A Understanding How a Sniffer Works

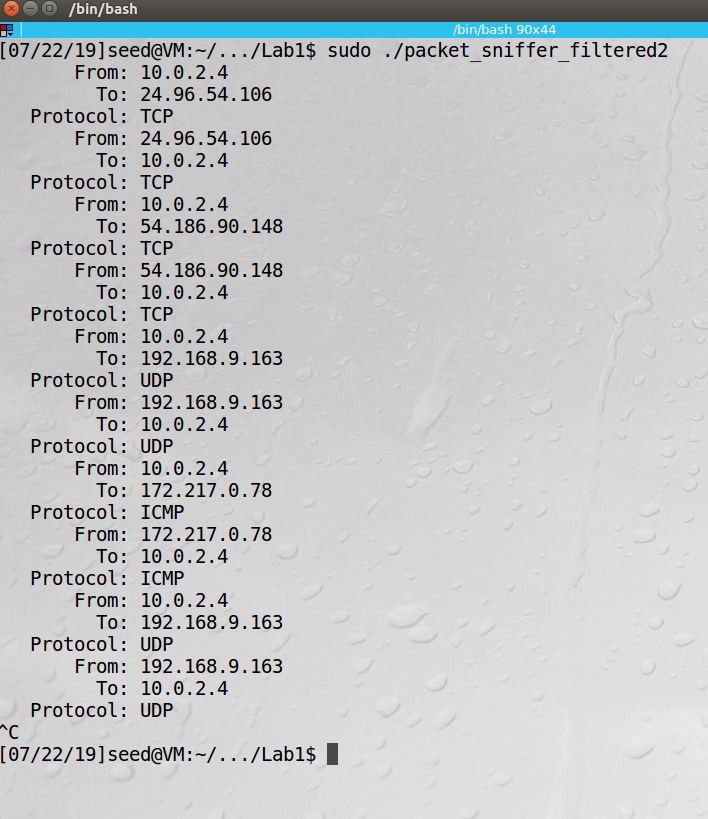
This task demonstrations using the PCAP library to sniff packets.

Code for Sniffer:



A ‘struct’ object is used to more easily populate the headers. When a packet is received the information is printed to the screen for the protocol, source IP address and destination IP address.

Output for running sniffing program, and pinging google/opening Firefox browser:



**Observation**: The image capture above shows the traffic being sniffed using the PCAP library. The program was set to capture all packets and, in the screenshot above, you can see ICMP, UDP and TCP packets being captured.

**Explanation**: The PCAP library can be used to sniff network traffic. There is a filter that can be used if there is only a certain type of packet is desired, for example, only TCP traffic, or only traffic to a specific destination address.

**Question** **1**. The first thing needed when capturing packets is for the NIC to be placed in promiscuous mode, so that it will ingest all the traffic that it sees and not just the traffic destined for that machine. By using the PCAP API we can filter down the packets to those that we care about capturing and then trigger an action after the capture, for example, print the information to the screen, like above.

**Question** **2**. Root privileges are required to access promiscuous mode and set up raw sockets, two operations that are critical for sniffing packets, without these privileges the application would fail.

**Question** **3**. When promiscuous mode is tuned off the NIC will only ingest the packets destined for that particular machine. Therefore, you would only be able to sniff the network traffic coming in and out of that machine. When promiscuous mode is re-enabled you would again be able to see traffic that comes from and goes to different machines on that network.

### 2.1B Writing Filters

Various filters can be used to capture only specific types of traffic, below are examples some examples of those filters :

Code for filter for capturing only ICMP packets between two specific hosts (10.0.2.4 and 8.8.8.8):



Filter code for capturing only TCP packets in port range 10-100:

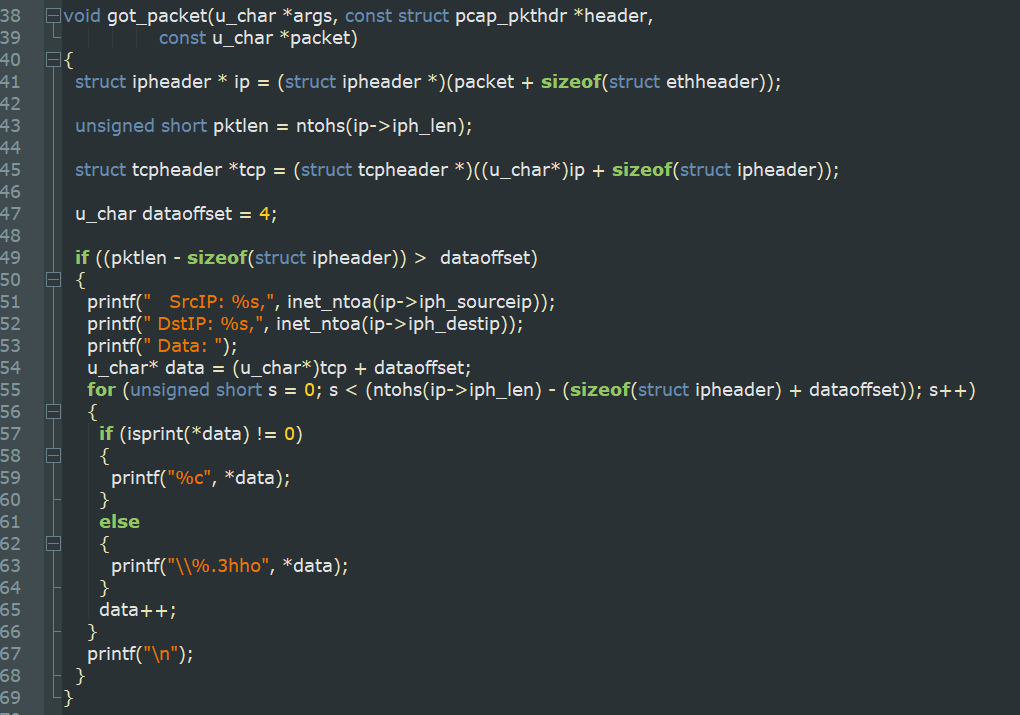


### 2.1C Sniffing Passwords

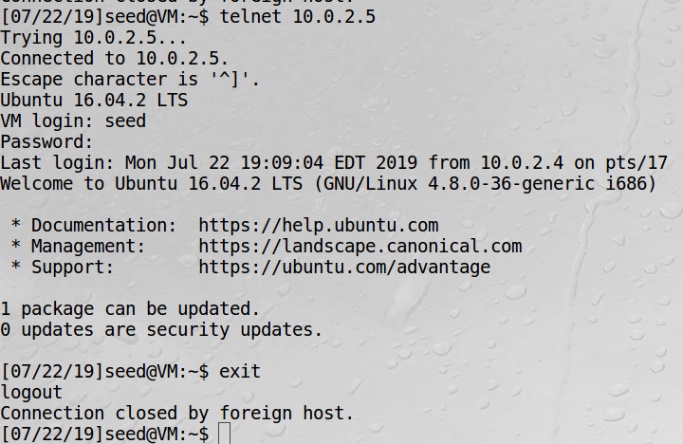
Packet sniffing can be used to exploit passwords in certain cases. Below is an example of sniffing telnet traffic to find out the user’s password:

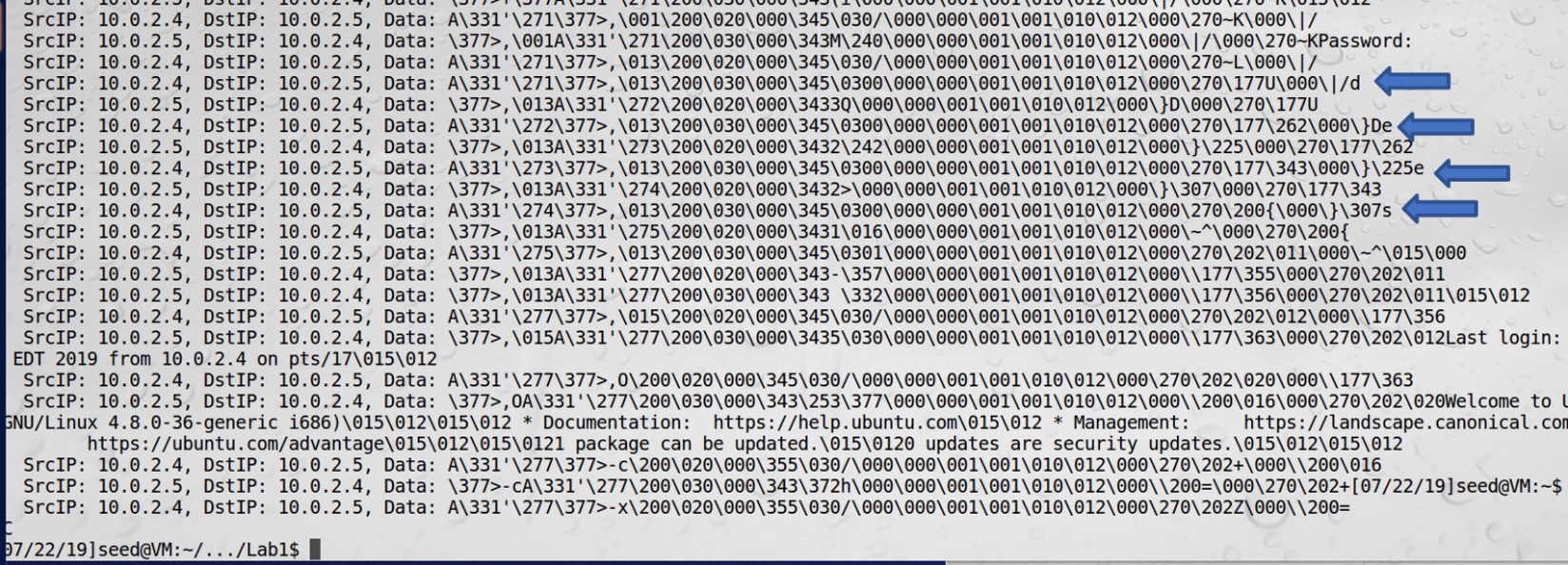
Code for sniffing passwords:





Example of Sniffed Password (‘dees’):





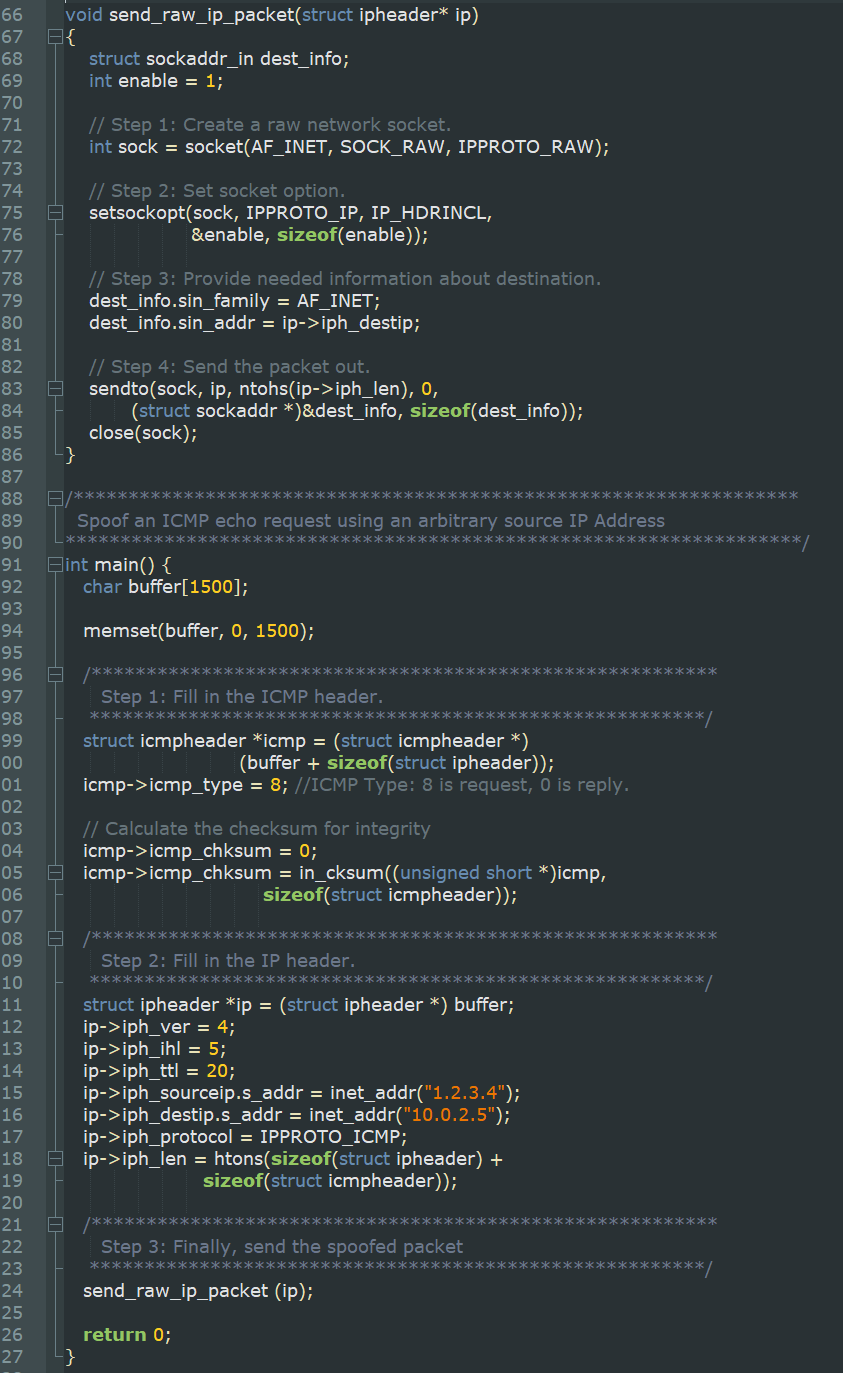
**Observation**: In the screenshots above you can see the telnet traffic being sniffed and the password ‘dees’ being captured (highlighted by the blue arrows). The capture filter was used to collection only telnet traffic and we used out program to print the data out to the screen. The password was sent from 10.0.2.4 to 10.0.2.5 and intercepted using the packet sniffing.

**Explanation**: Using the PCAP library and filtering for telnet traffic we were able to capture the packets. We then processed them and printed the info to the screen and discovered that the user’s password was ‘dees’. This task showed a practical (and nefarious) use of a sniffing program.

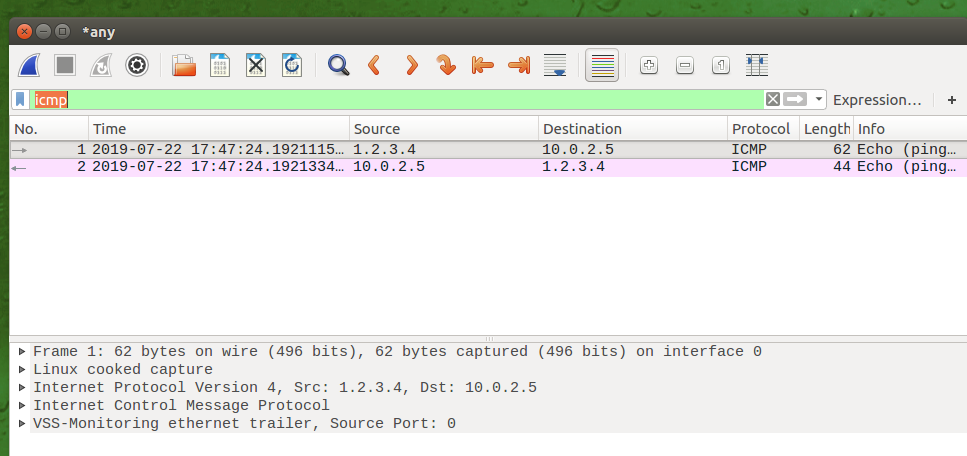
## 2.2 Spoofing

### 2.2A Write a Spoofing Program

This task shows the ability to send out an ICMP packet from spoofed IP of 1.2.3.4 to IP 10.0.2.5 by using raw sockets:



On machine 10.0.2.5 you can see the spoofed packet that it received:



**Observation**: The screenshot shows that a packet was received from source IP 1.2.3.4. This was not the actual origination of the packet; it instead came from machine 10.0.2.4 who was sending the spoofed packet.

**Explanation:** By using a raw socket, we were able to make a packet look like it originated in from a different machine than it actually did. In this task I used an ICMP packet, but other protocols can be used as well.

### 2.2B Spoof an ICMP Echo Request

See 2.2A (previous) for ICMP Spoofing since ICMP was used for both tasks.

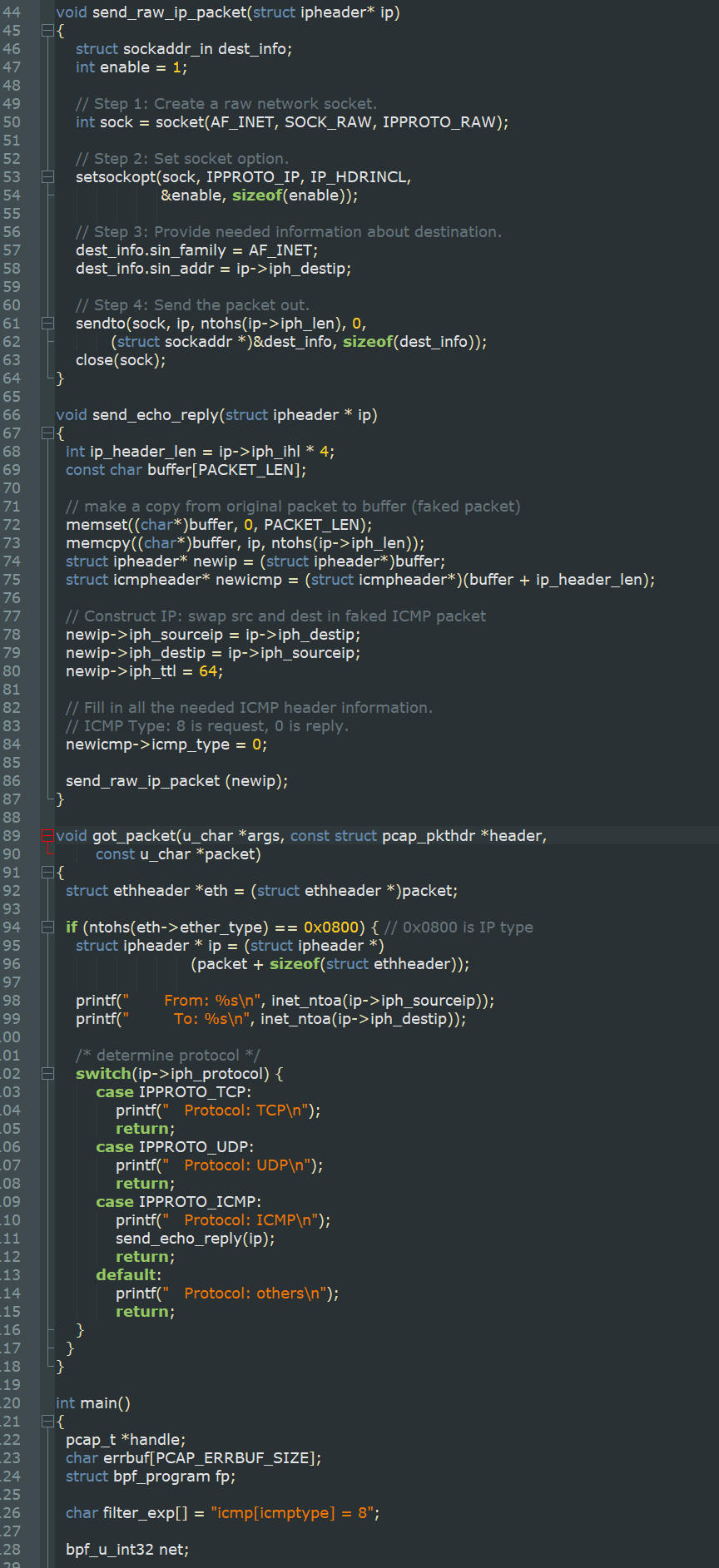
**Question 4**: Yes, you can use an arbitrary value for the IP packet length field and the spoofed packet will still be sent. This is not built in size verification for the packet length.

**Question 5**: I don’t believe it is required to calculate the checksum of the packet, however, in the code above the checksum is calculated for the ICMP packet.

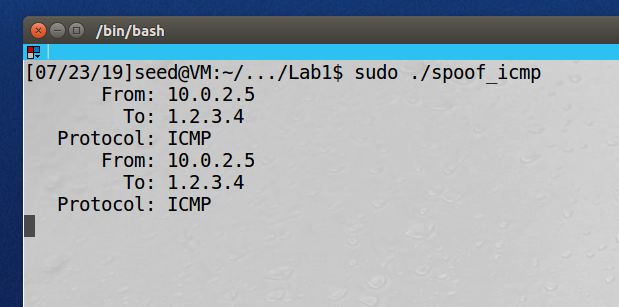
**Question 6**: Two operations that require the use of root privileges are enabling promiscuous mode on the NIC and a setting up the raw socket. Without the privileges the program would fail on the pcap\_open\_live and the send\_raw\_ip\_packet functions.

## 2.3 Sniff and then Spoof

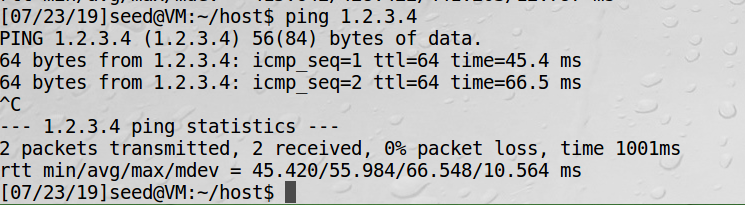
Code for sniffing an ICMP request and spoofing an ICMP reply:

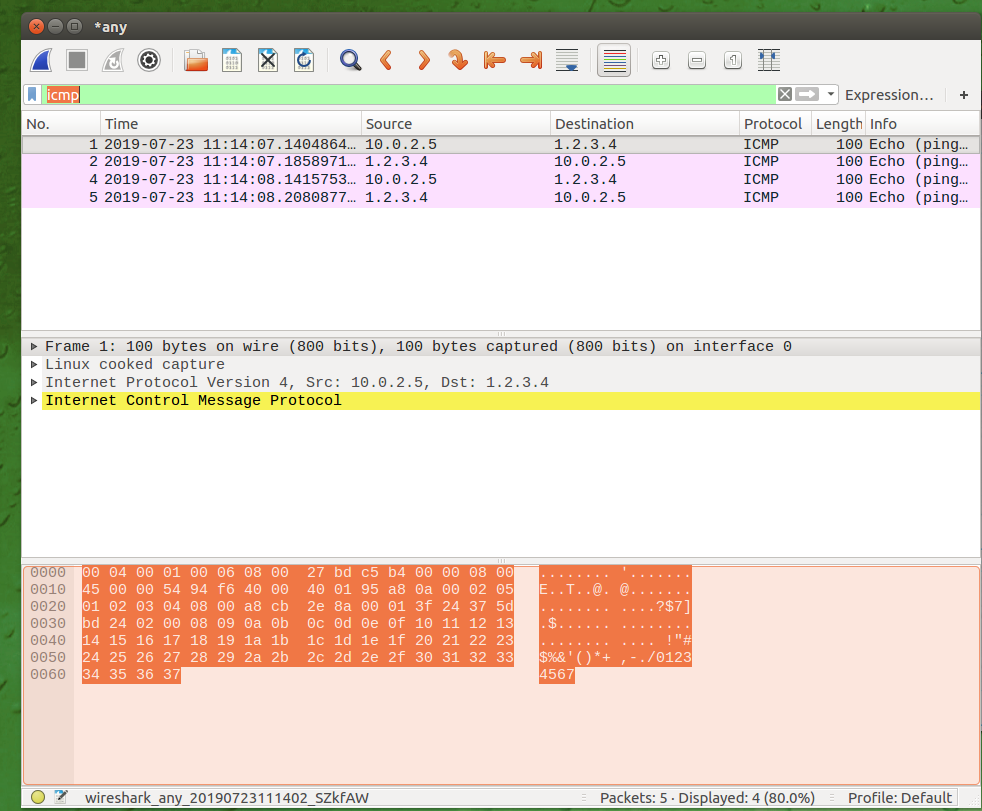


Computer A running sniffing/spoofing:



Computer B after pinging 1.2.3.4:





**Observation:** We first run the sniffing and spoofing program on computer A and then from computer B we are going to ping an IP address that is not normally reachable 1.2.3.4. If the sniff and spoof program wasn’t running on computer A then computer B’s ping to 1.2.3.4 would have timed out, but since the program was running, computer B did get a reply. We used Wireshark to show the spoofed packet above.

**Explanation:** By using promiscuous mode, PCAP and raw sockets we were able to build a program that will hijack and respond to all ICMP requests. The packet was captured, and a raw socket was sent as a response to the ICMP request. This could be used for other types of traffic as well for various network-based attacks.