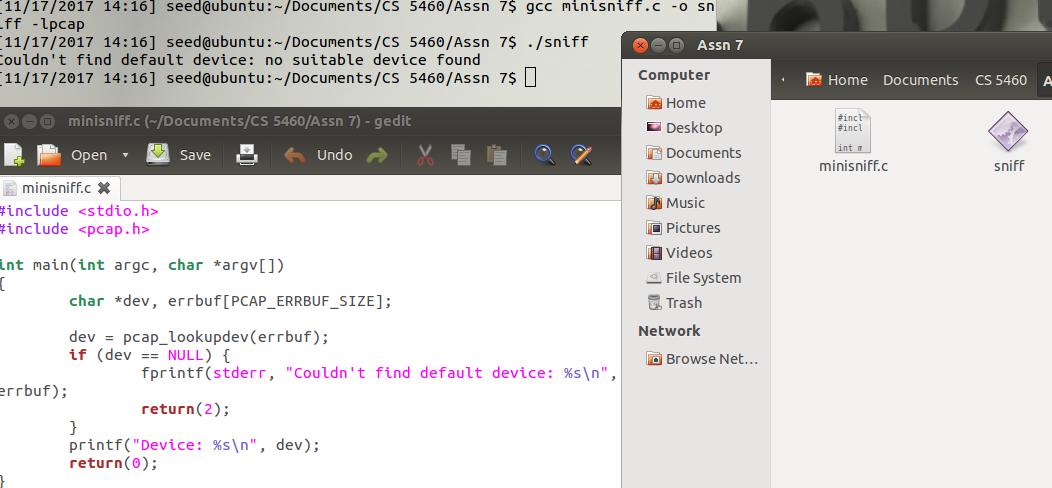
Daniel Oliveros

CS 5460 – Assn 7

**Packet Sniffing and Spoofing Lab**

**Task 1:**



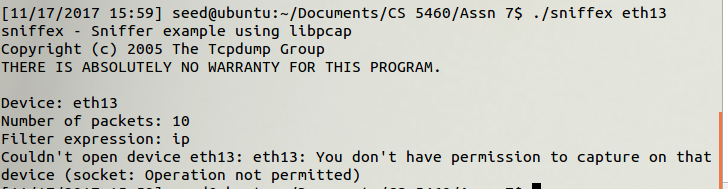


**1a.**

**Problem 1:** First, we have to find the inputted device, the library call for this is *pcap\_lookupdev(errbuf)*, this call also allows us to have an error buffer. After this, we have to open the device, this is done using the method  *pcap\_open\_live(char \*device, int snaplen, int promisc, int to\_ms,char \*ebuf)*, which uses various parameters to determine promiscuity when listening through the drive, as well as other important guidelines to follow and variables to use. Following this, we focus on filtering our sniffing to a particular port, this is done following a two step process, we start by compiling the filter using the method *pcap\_compile(pcap\_t \*p, struct bpf\_program \*fp, char \*str, int optimize, bpf\_u\_int32 netmask)*, and after this, we apply the filter. Filter application is performed using the command *pcap\_setfilter(pcap\_t \*p, struct bpf\_program \*fp)*. Now that we have determined the particular location to sniff we can start sniffing packets, a good method for this is *pcap\_loop(pcap\_t \*p, int cnt, pcap\_handler callback, u\_char \*user)*, which is based on callbacks, so it waits until the next packet comes through the network and handles it.

**Problem 2:**

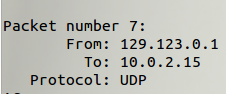
When I tried to run sniffex as a non-root user, I got this error:



This is due to the normal user not having permission to capture data from devices.

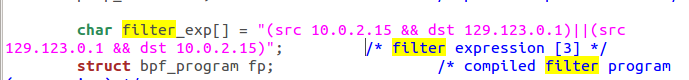
**Problem 3:**

To test this scenario, we ca first run the program in non-promiscuous mode, listening only to the eth13 port. By listening to this port for a minute and not opening any websites, we were able to measure this steady network traffic:



**1b.**

For the first filter condition, I changed my code slightly and it looked like this:

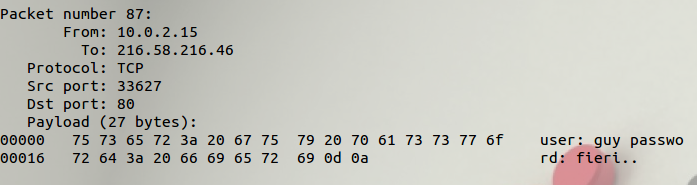


These hosts were the ones I identified as those that communicate periodically to ensure there is still a network connection. This meant that whenever I browsed a new site the packet count would increase very slightly, as opposed to the typically large increase that is seen without the filter.



For the second condition, the port range filter was very useful. It allows us to scan every port within a certain range.

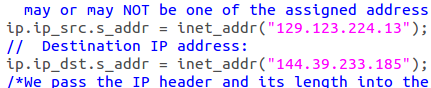
**1c.**

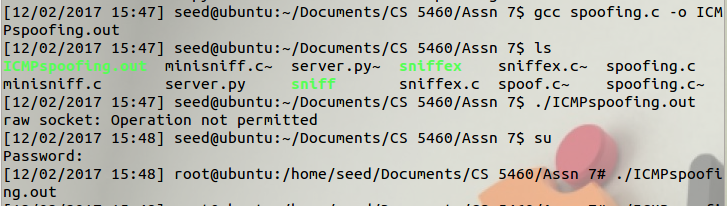


I sent this data through a telnet request to Google, due to how outdated our telnet program was it got me stuck in their server. Anyhow, the request was sent in clear text, making it easy to spot any data being sent.

**Task 2:**

**2a.** IFound a spoofing file online at <http://www.enderunix.org/docs/en/rawipspoof/>. The destination IP is Garrett’s. Using that program, I just changed the IP addresses within it to match the behavior I was trying to emulate.



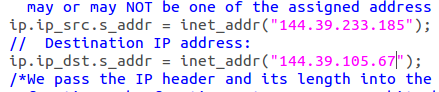


Here’s a screen capture of the wireshark log related to this request.



**2b.**

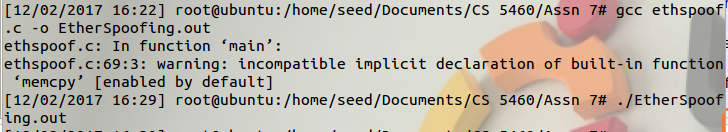
For the source, I used my laptop’s IP to send a spoofed echo request to Garret’s laptop.



Here’s a wireshark screen capture showing the request that was sent.



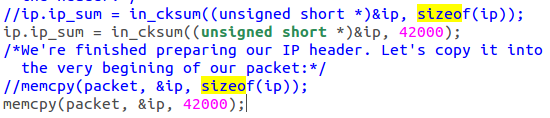
**2c.** Found some data for how to perform this request at <http://aschauf.landshut.org/fh/linux/udp_vs_raw/ch01s03.html>

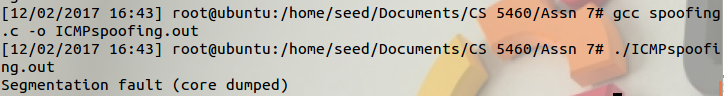


Here’s a wireshark log of the resulting spoofed request.

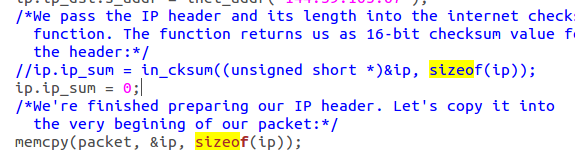


**Question 4:** Yes, if it is small enough it will work regardless of the value we give it for the IP. We set the length field to be 4200 and it still sent the request correctly. Once we set it to 42000 we got a segmentation fault.





**Question 5:** Strangely enough no. I messed with the value the checksum was being stored into by setting it to zero and the request was still sent.

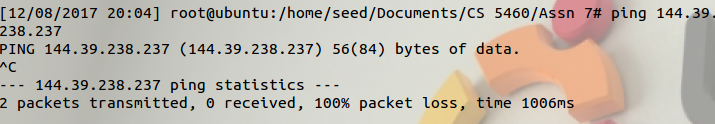




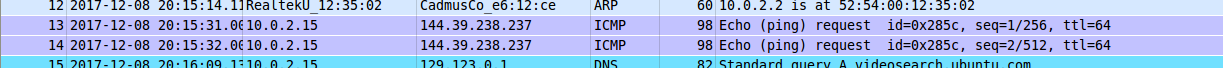
**Question 6:** Various internal requests require admin privileges. When it comes to sockets, root privileges are necessary in order for the process to open raw sockets.

**Task 3:** Due to the network settings of our virtual machine, we are unable to sniff packets traveling from a different VM. We tried before, but promiscuous mode was unable to sniff packets coming through different Virtual Machines. So, for this we will ping an ip. Sniff it using WireShark, and then spoof it using our spoofing program.

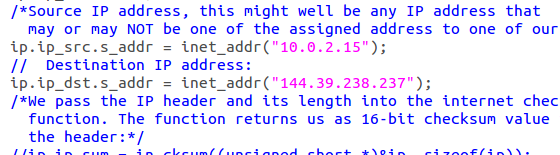
First, we simulate a ping request to another IP. In this case I used Garrett’s.



WireShark logs these requests



We now just look at them, and copy their contents into our spoofing program.



After compiling and running the program, this is the resulting request.



This request is quite similar to the one we sniffed. It is impressive how easy it is to spoof a request once you know its contents.