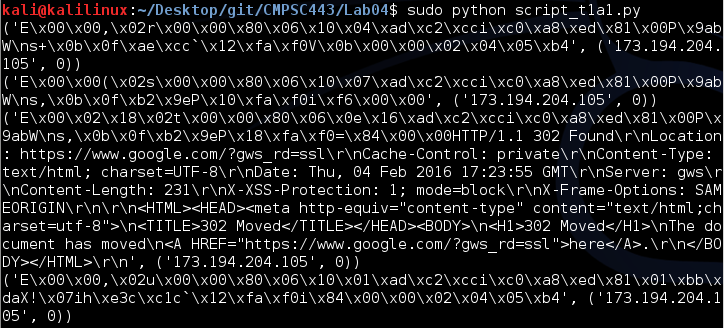
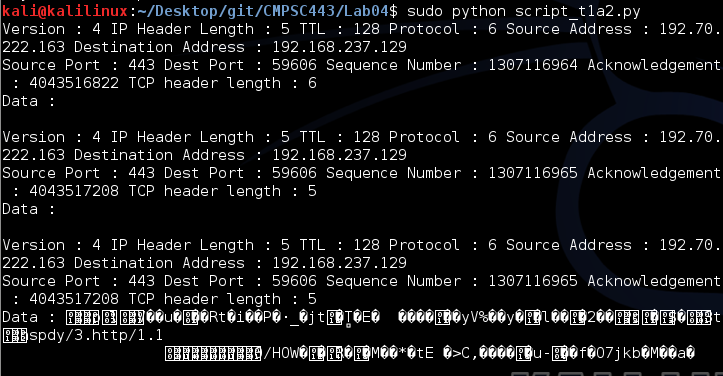
**Task 1: Writing a Packet Sniffing Program**

The first part of task one is to follow a tutorial of packet sniffing with python (<http://www.binarytides.com/python-packet-sniffer-code-linux/>). The tutorial first has us sniff for packets and outputs any data (in 16 bit chunks) received from the socket. An example of this output is below in Figure 1.



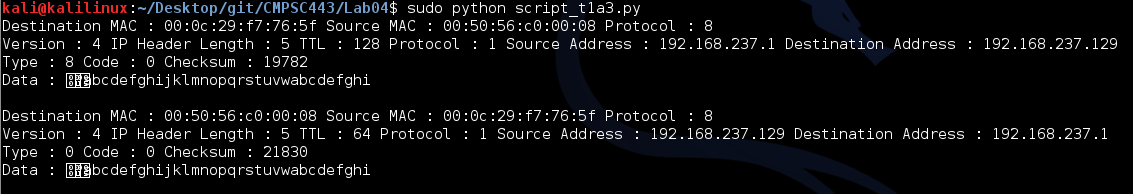
**Figure 1:** Data output from socket

Next, the tutorial has us parse the packets to make the data inside them more readable. The parsing is shown below in Figure 2.



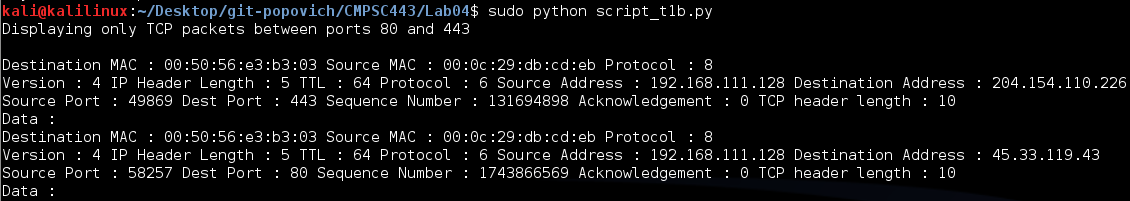
**Figure 2:** Parsed data output from socket

Unfortunately, the above code had some drawbacks. It was restricted to sniffing only incoming TCP packets and could only return IP frames. The last part of the tutorial has us extend the previous program to sniff TCP along with UDP and ICMP packets, sniff incoming and outgoing packets, and deliver Ethernet headers along with the IP frames. The output of said program is below in Figure 3.

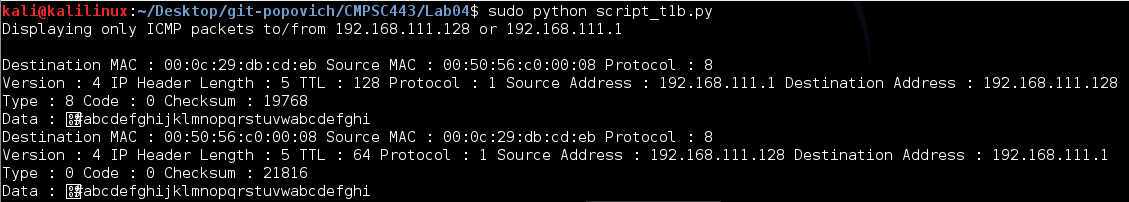


**Figure 3:** Parsing incoming and outgoing frames along with Ethernet headers

The next part of task 1 (Part b) is to write filters. Our filter program will read from a text file (filter.txt). In that text file we will specify whether we want our program to read TCP packets or ICMP packets and specify from what ports/IP addresses as well. Figure 4 displays the script’s TCP functionality while Figure 5 highlights its ICMP functionality.



**Figure 4:** Filtering TCP packets between ports 80 and 443

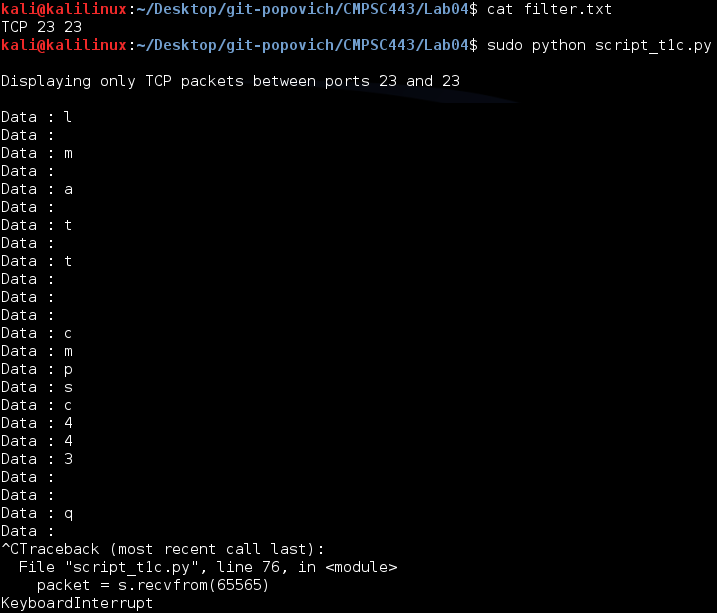


**Figure 5:** Filtering ICMP packets between IP addresses

192.168.111.128 and 192.168.111.1

The final task of task 1 is to demonstrate that if usernames and passwords aren’t encrypted or hashed they can be seen by anyone on the network between the user and the server. I registered as a new user for games.libreplanet.org with the username “matt” and password “cmpsc443”.

Figure 6 shows exactly this. The script from task 1 part b was modified to only output the data section of packets and not the headers, MAC and IP addresses, etc. You can see me type the command to login (‘l’), my username, my password, and the command to quit (‘q’), all in plaintext.

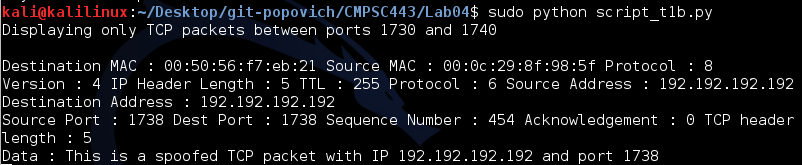


**Figure 6:** Demonstration of telnet’s unencrypted text communication

**Task 2: Spoofing**

Task 2 begins by following another tutorial, this time at “http://www.binarytides.com/raw-socketprogramming-

in-python-linux/”. The tutorial explains a python TCP packet spoofing program and gives us the code to do so. With a little modification, I was able to spoof a packet to and from the IP address 192.192.192.192 over port 1738. I confirmed this with my previous python packet sniffing program. The results can be seen in Figure 7.



**Figure 7:** Confirmation of TCP packet spoofing

The next part of task 2 was to spoof an ICMP Echo request.

[coming soon…]

The final part of task 2 was to spoof an Ethernet Frame.

[coming soon…]

**Questions:**

Yes, you can set the IP packet length field to any value. The kernel will eventually overwrite the value you put in and fill the correct total length of the packet before it is put onto the network.

No, just like with the packet length field, you do not have to calculate the checksum for the IP header. The checksum will be overwritten by the kernel.