

2 lab tasks

2.1 Task 1: Writing Packet Sniffing Program

Task1.a:

Problem 1:

The first call is `pcap_lookupdev()`, which finds the device name that we want to sniff on.

The 2nd is `pcap_open_live`, which opens a sniffing session. We provide it with the device name that we found in the first step.

After that, we need to call `pcap_datalink()` to determine what type of ethernet header we are working with.

If we want to filter certain types of traffic, we call `pcap_compile()`

Next, we call `pcap_loop()` to capture n number of packets.

Problem 2:

We need to run `sniffex` with root privilege because we cannot enter promiscuous mode which is needed to capture all traffic on a given network.

Problem 3:

In theory, promiscuous mode should affect the packets captured. However, in this test, I could not really see a difference.

To demonstrate this, here are some screen caps:

Promiscuous Mode off:

```
Device: eth13
Number of packets: 10
Filter expression: ip

Packet number 1:
  From: 10.0.2.15
  To: 10.0.2.3
  Protocol: UDP

Packet number 2:
  From: 10.0.2.3
  To: 10.0.2.15
  Protocol: UDP

Packet number 3:
  From: 10.0.2.15
  To: 10.0.2.3
  Protocol: UDP

Packet number 4:
  From: 10.0.2.3
  To: 10.0.2.15
  Protocol: UDP

Packet number 5:
  From: 10.0.2.15
  To: 10.0.2.3
  Protocol: UDP

Packet number 6:
  From: 10.0.2.3
  To: 10.0.2.15
  Protocol: UDP

Packet number 7:
  From: 10.0.2.15
  To: 10.0.2.3
  Protocol: UDP

Packet number 8:
  From: 10.0.2.3
  To: 10.0.2.15
  Protocol: UDP

Packet number 9:
  From: 10.0.2.15
  To: 10.0.2.3
  Protocol: UDP

Packet number 10:
  From: 10.0.2.3
  To: 10.0.2.15
  Protocol: UDP

Capture complete.
[05/07/2019 13:26] seed@ubuntu:~$
```

Promiscuous mode on:

```
Device: eth13
Number of packets: 10
Filter expression: ip

Packet number 1:
  From: 10.0.2.15
  To: 224.0.0.251
  Protocol: UDP

Packet number 2:
  From: 10.0.2.15
  To: 10.0.2.3
  Protocol: UDP

Packet number 3:
  From: 10.0.2.3
  To: 10.0.2.15
  Protocol: UDP

Packet number 4:
  From: 10.0.2.15
  To: 10.0.2.3
  Protocol: UDP

Packet number 5:
  From: 10.0.2.3
  To: 10.0.2.15
  Protocol: UDP

Packet number 6:
  From: 10.0.2.15
  To: 10.0.2.3
  Protocol: UDP

Packet number 7:
  From: 10.0.2.3
  To: 10.0.2.15
  Protocol: UDP

Packet number 8:
  From: 10.0.2.15
  To: 10.0.2.3
  Protocol: UDP

Packet number 9:
  From: 10.0.2.3
  To: 10.0.2.15
  Protocol: UDP

Packet number 10:
  From: 10.0.2.15
  To: 224.0.0.251
  Protocol: UDP

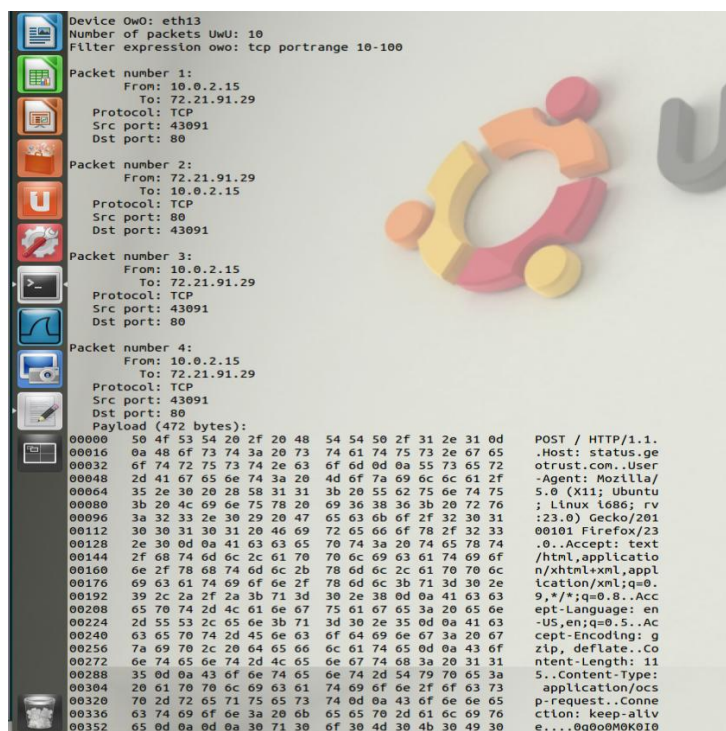
Capture complete.
[05/07/2019 13:41] seed@ubuntu:~$
```


Task 1.b

- Capture the ICMP packets between two specific hosts



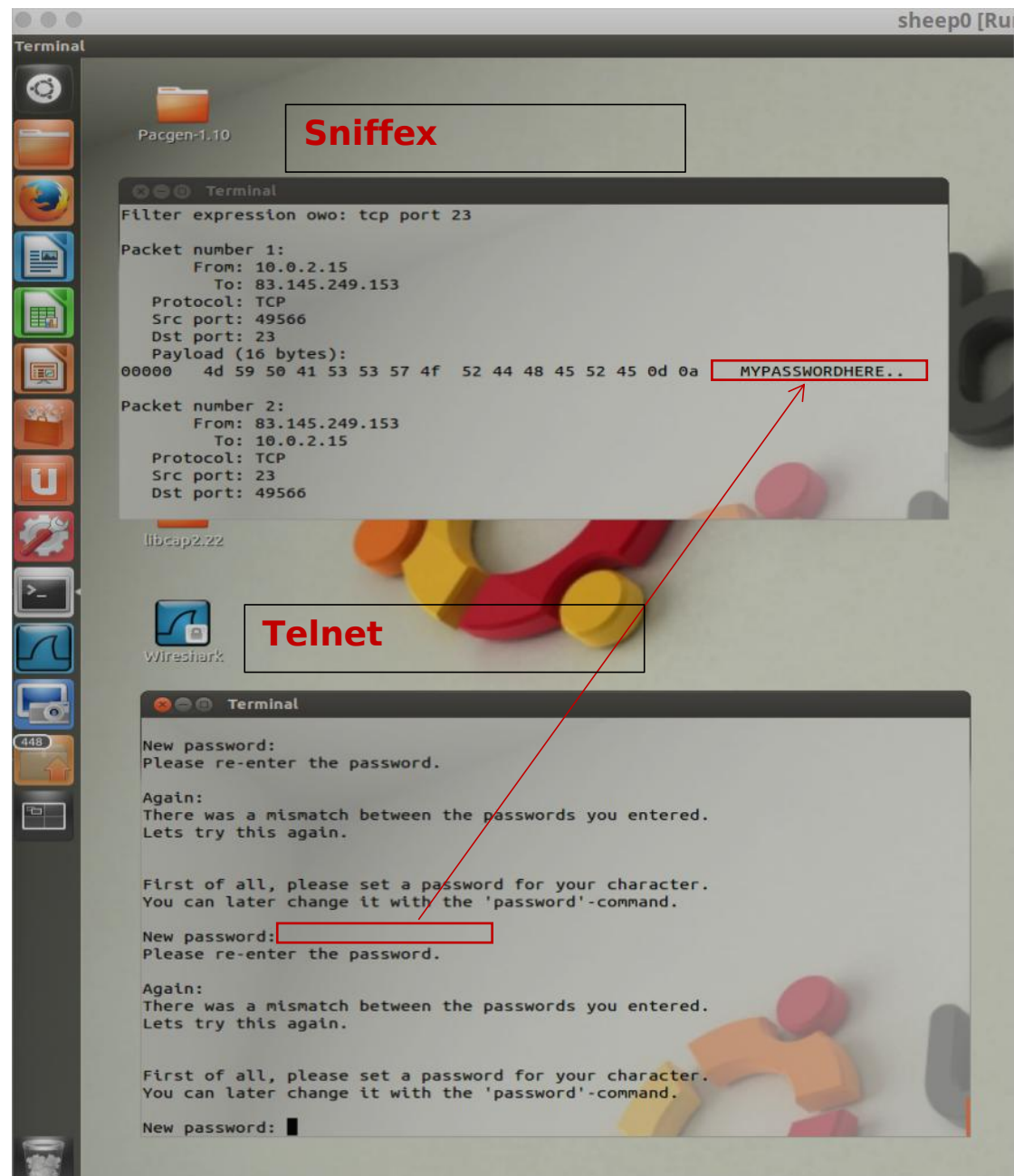
- Capture the TCP packets that have a destination port range from to port 10 - 100



Task 1.c

To sniff passwords over telnet, we need to change the filter expression to only capture TCP packets on port 23 (the default port for telnet). Then, we can sniff the packets. What we will see is a request, and then a response containing the password as plain text.

Here is a screen cap to show this:



Task 2.a

In this part, I gave a UDP packet a custom payload that says “Say Hello to my Little Packet”. The actual hex values for this string can be seen at the bottom of the 2nd screen cap, while the translated text can be seen at the bottom of the 1st.

The image displays two screenshots from a cybersecurity lab. The top screenshot is a Wireshark packet capture showing a UDP packet. The packet list shows three packets, with the third packet (No. 3) selected. The packet details pane shows the following information:

- Frame 4: 71 bytes on wire (568 bits), 71 bytes captured (568 bits)
- Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:11 (00:00:00:00:00:11)
 - Destination: 00:00:00:00:00:11 (00:00:00:00:00:11)
 - Address: 00:00:00:00:00:11 (00:00:00:00:00:11)
 - ... 0 ... = 10 bit: Globally unique address (factory default)
 - ... 0 ... = 10 bit: Individual address (unicast)
 - Source: 00:00:00:00:00:00 (00:00:00:00:00:00)
 - Address: 00:00:00:00:00:00 (00:00:00:00:00:00)
 - ... 0 ... = 10 bit: Globally unique address (factory default)
 - ... 0 ... = 10 bit: Individual address (unicast)
- Type: IPv4 (0x0800)
- Internet Protocol Version 4, Src: 37.35.3.3, Dst: 37.35.3.3
 - ... 0101 = Version: 4
 - ... 0101 = Header Length: 20 bytes (5)
 - Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
 - Total Length: 67
 - Identification: 0x1234 (4660)
 - Flags: 0x0000, Don't Fragment
 - Time to live: 255
 - Protocol: UDP (17)
 - Header checksum: 0x1234 [validation disabled]
[header checksum status: unverified]
 - Source: 37.35.3.3
 - Destination: 37.35.3.3
- User Datagram Protocol, Src Port: 1234, Dst Port: 12345
 - Source Port: 1234
 - Destination Port: 12345
 - Length: 37
 - Checksum: 0x0000 [unverified]
[Checksum Status: Unverified]
 - Stream index: 01

The bottom screenshot shows the PacketETH - ethernet packet generator interface. The Link layer section is configured with the following settings:

- Get MAC: MAC Header
 - Destination: 00:00:00:00:00:00 (Select)
 - Source: 00:00:00:00:00:00 (Select)
 - Ethertype: 0x0800 (IPv4)
- Next layer: IPv4

The IPv4 data section is configured with the following settings:

- Version: 4, Header length: 20, TOS: 0, Total length: 67, Identification: 1234
- Flags: 0, Fragment offset: 0, TTL: 25, Protocol: 17 (UDP), Header cks: 0x1234
- Source IP: 37.35.3.3, Destination IP: 37.35.3.3, Options: 0x
- Next layer: UDP

The UDP data section is configured with the following settings:

- Source port: 1234, Destination port: 12345, Length: 37, Checksum: 0x0000
- Pattern: 0, Length: 4, Apply pattern, Select payload

The bottom of the PacketETH interface shows the hex values for the payload: 53 61 79 20 48 65 6c 6c 6f 20 74 6f 20 44 79 20 4c 69 74 74 6c 65 20 50 61 63 6b 65 74.

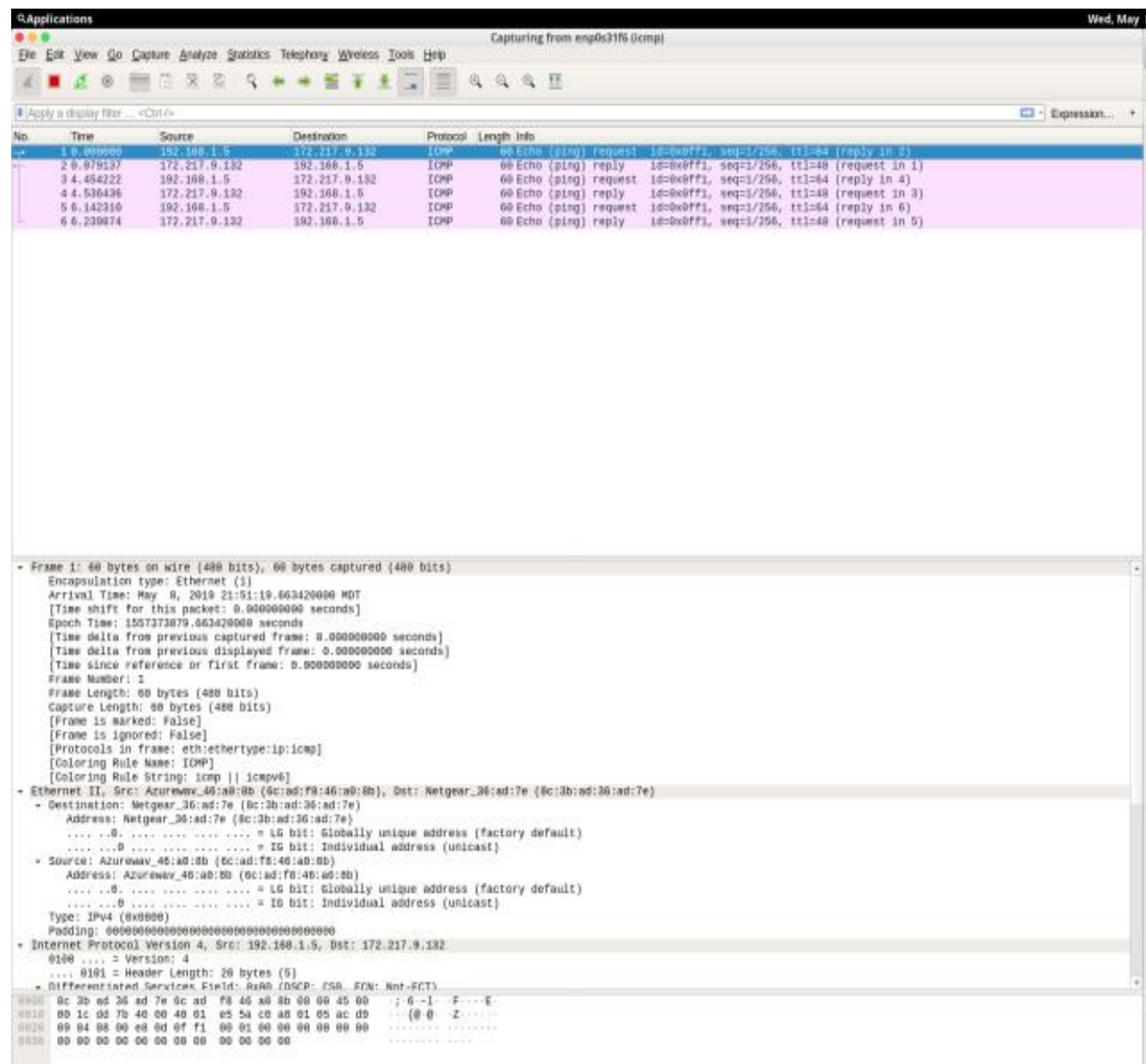
Task 2.b

I decided to spoof an ICMP request on behalf of my roommate's computer.

First, i started a packet capture with wireshark, and looked at a packet that was sent between my computer and his computer, so i could get the MAC address and IP for his computer.

Then, I started a new packet capture with ICMP as a filter, and executed the command `~$ ping www.google.com`. Again, i was able to get a suitable MAC address and IP from this.

Using my roommate's PC as a source, and google as a destination, i sent out an echo request and successfully got a response:



The screenshot shows the PackETH application interface. At the top, there's a title bar with the time 9:51 PM and the window title "PackETH - ethernet packet generator". Below the title bar is a menu bar with "File" and "Help". A toolbar contains buttons for "Builder", "Gen-b", "Gen-s", "Pcap", "Load", "Save", "Default", "Default", "Interface", "Send", and "Stop".

The main configuration area is divided into three sections:

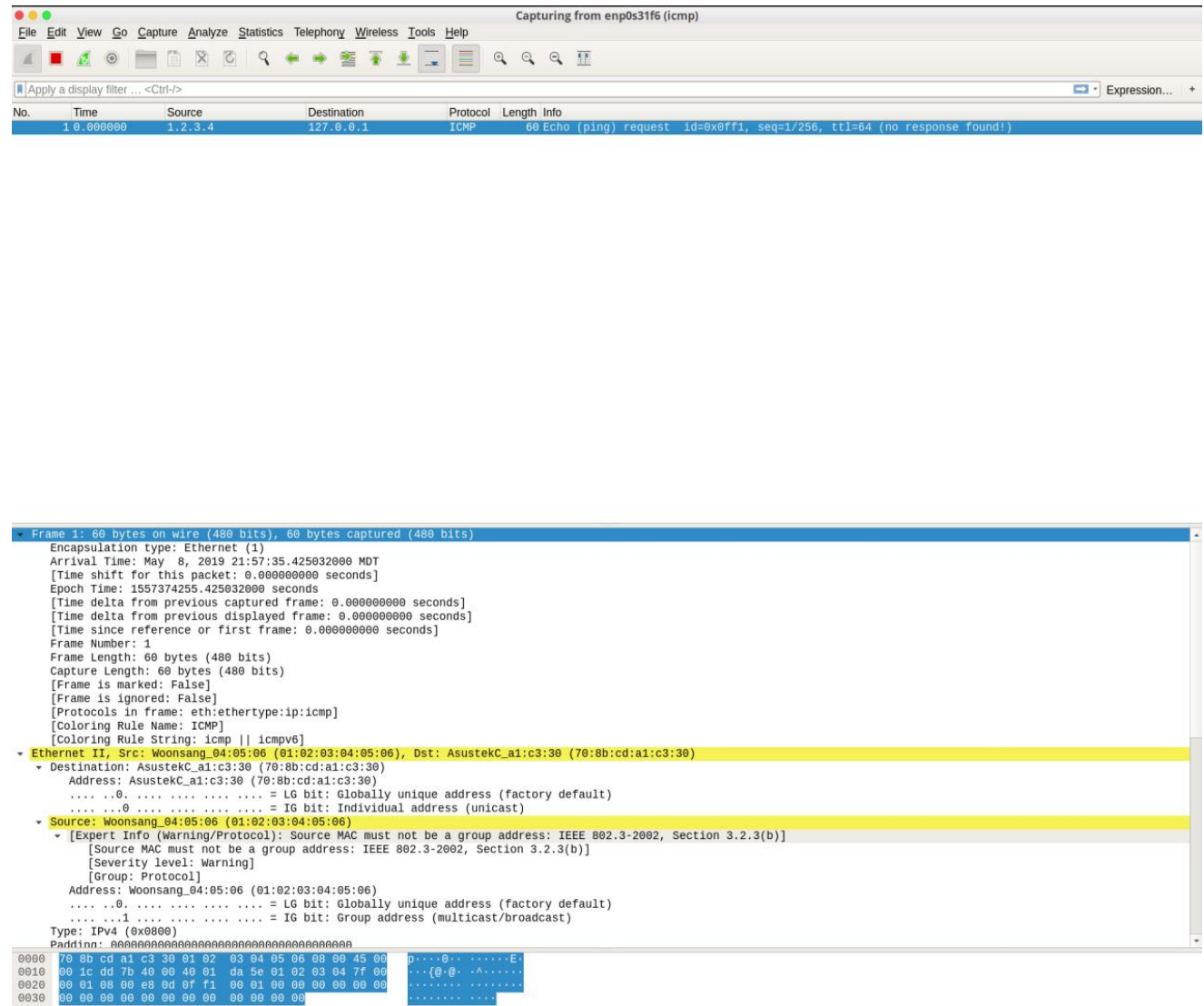
- Link layer:** Includes a "Get MAC" button, a "ver 8" radio button (selected), and radio buttons for "802.3" and "802.1q". The "MAC Header" section has "Destination" (BC:3B:AD:36:AE) and "Source" (ad:f8:46:a0:8b) fields, each with a "Select" button. The "Ethertype" is set to "0800" and "IPv4". The "802.1q VLAN fields" section includes "QinQ" (0x8100), "Tag ID" (0x 811), "Priority" (0 (Best effort)), and "CTI VLAN ID" (0x 001). The "802.3 LLC field values" section includes "Type" (LLC), "DSAP" (0x 01), "SSAP" (0x 01), "Ctrl" (0x 01), and "PID" (0x 0000).
- IPv4 data:** Includes fields for "Version" (0x 4), "Header length" (0x 5), "TOS" (0x 0), "Total length" (Auto), "Identification" (0x 00), "Flags" (Select), "Fragment offset" (0), "TTL" (64), "Protocol" (1), "Reserved" (v), "Header cks" (Auto), "Source IP" (192.168.1.5), "Destination IP" (172.217.9.132), and "Options" (0x). The "Next layer" dropdown is set to "ICMP".
- ICMP data:** Includes a "Type" dropdown (01 Echo request), "Code" (0x 00), "Checksum" (Auto), "Identifier" (0x 0000), "Seq. number" (0x 0000), and "Data" (Data pattern: , Data length: 56).

(Wow this is blurry, avert your eyes)

Note: I didn't get permission from my roommate to spoof packets on his behalf. I'm a hacker extraordinaire, I don't need permission.

Task 2.c

The source has been spoofed, and is the second entry highlighted in yellow.



Question 4:

Yes, you can set the packet length to any value, so long as it does not exceed the maximum packet size. There are ways to send packets that are bigger than this, but it is not the norm.

Question 5:

You can, but you can also set it to an arbitrary value and it will still work.

Question 6:

It cannot access the network interfaces if you are not root, which means it can't send packets.