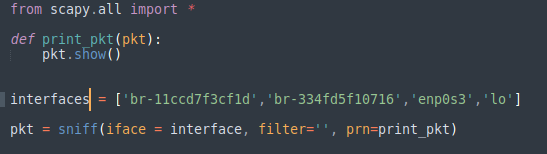
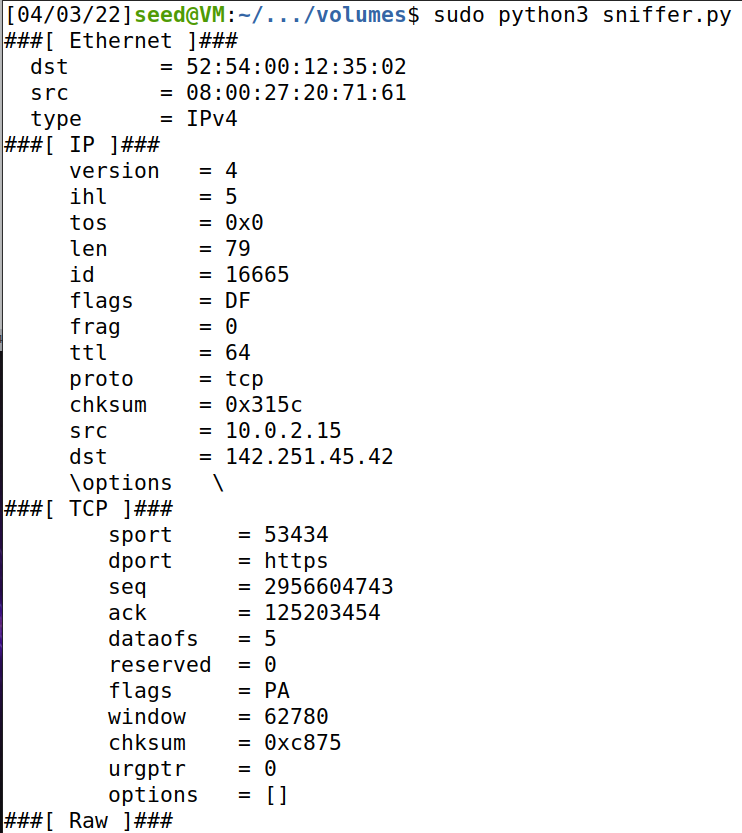
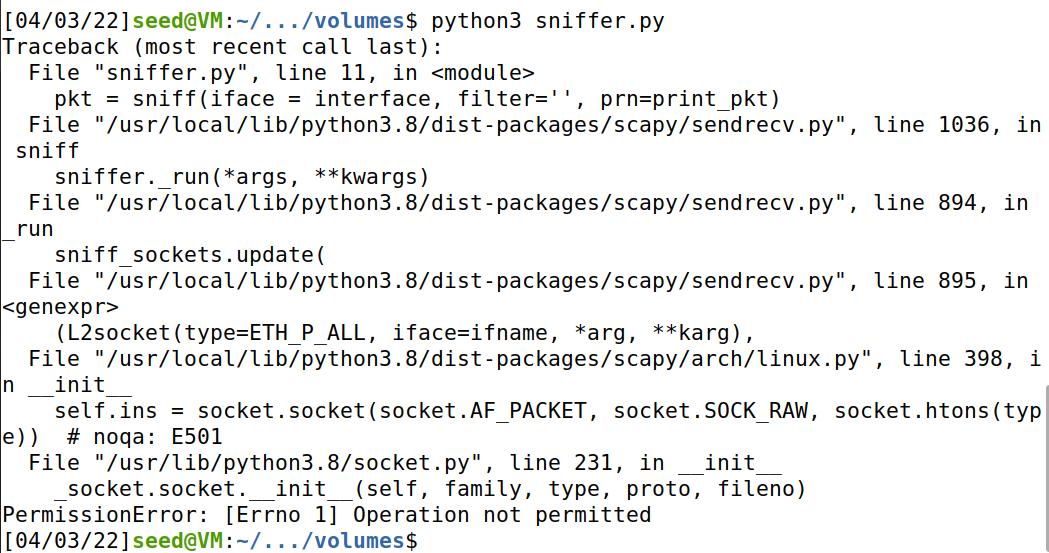
Val Robichaux

Homework 4

CSCE 465

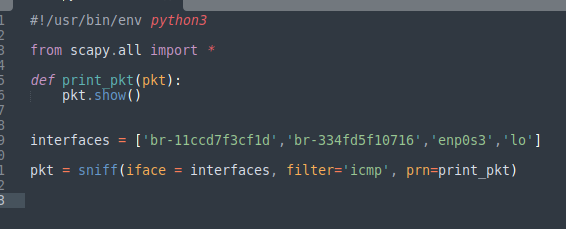
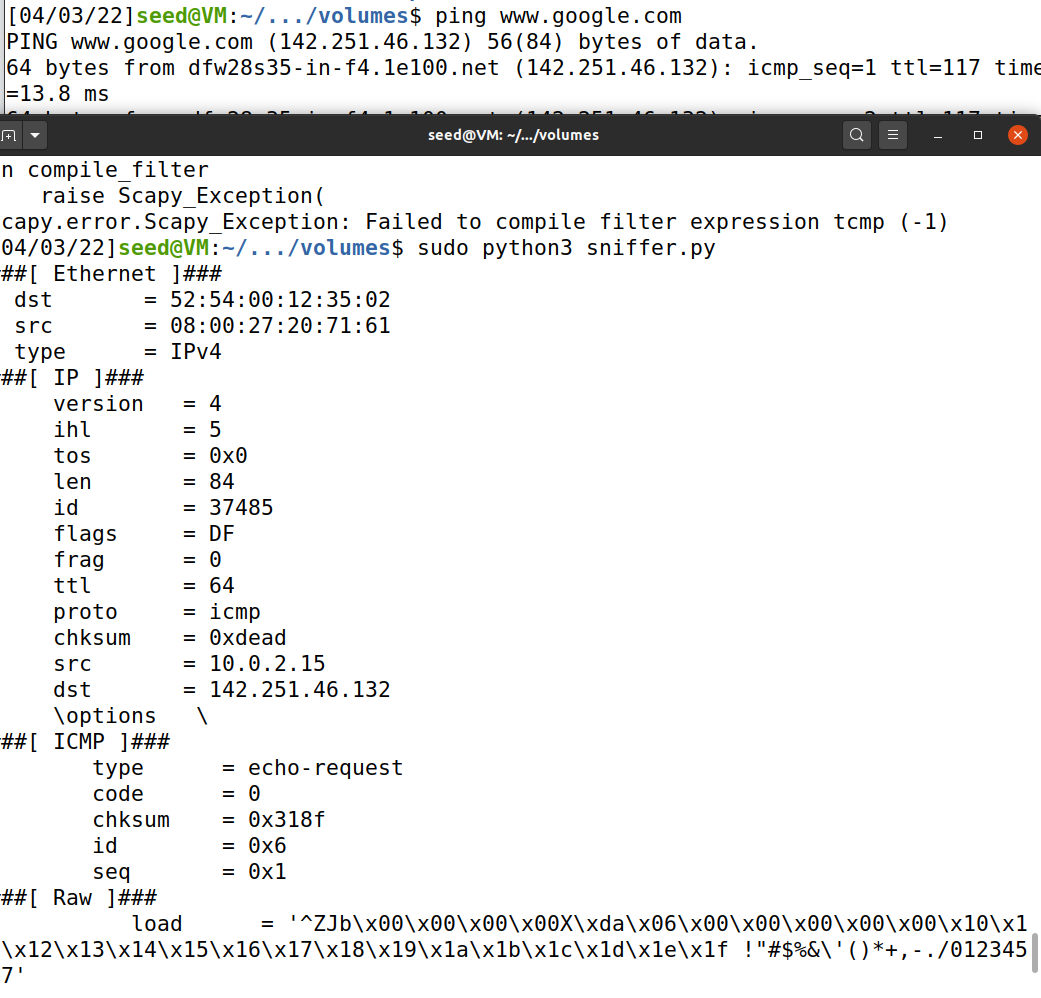
# Deliverables

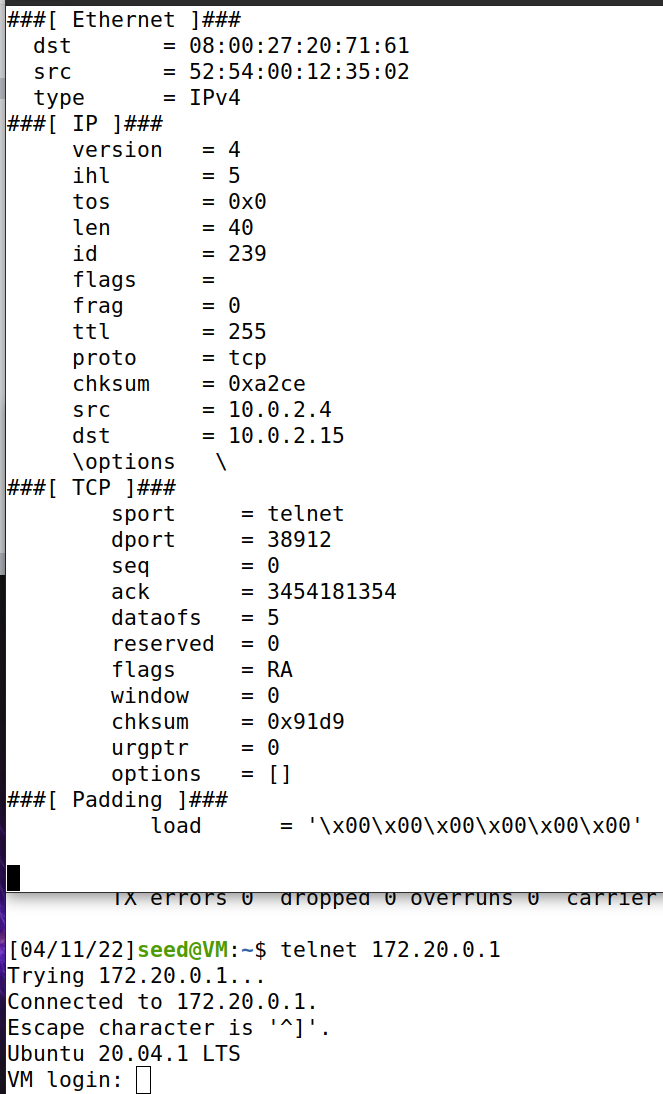
1. Use Scapy to Sniff and Spoof Packets
   1. Sniffing Packets

* Correctly perform task 1.1A
  + 
  + 

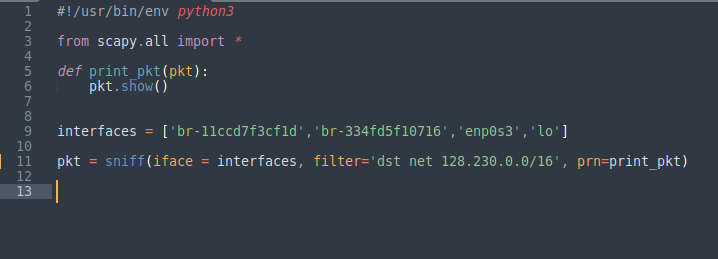
**Explanation:**

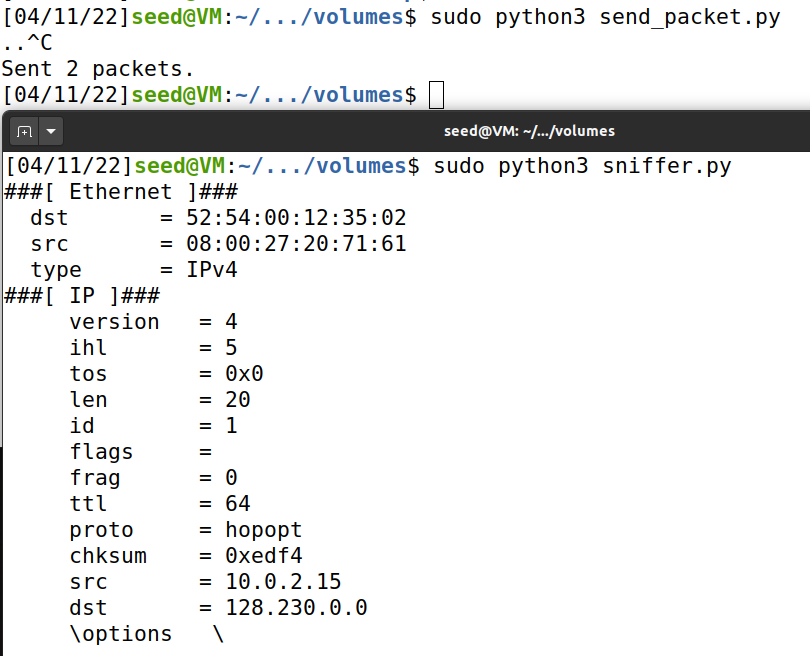
1. Screenshot one shows my snippet of code that I am using to sniff on the different interfaces that I found using the ‘ifconfig’ command.
2. I ran the program with root privileges in the first screenshot and you can see that I can see all of the traffic that is being communicated on all of our interfaces.
3. As you can see in screenshot 2 I ran the same command without root privileges and it is evident that I have to have root privileges in order to see all the traffic and capture packets.

* task 1.1 B
  + 
  + 
    - In this example you can see that I send a ping to google.com and that will generate an ICMP echo request packet. If the server that google is on is live it should receive an echo reply, and as you can see here printed out a response.
  + 

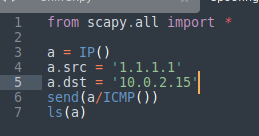


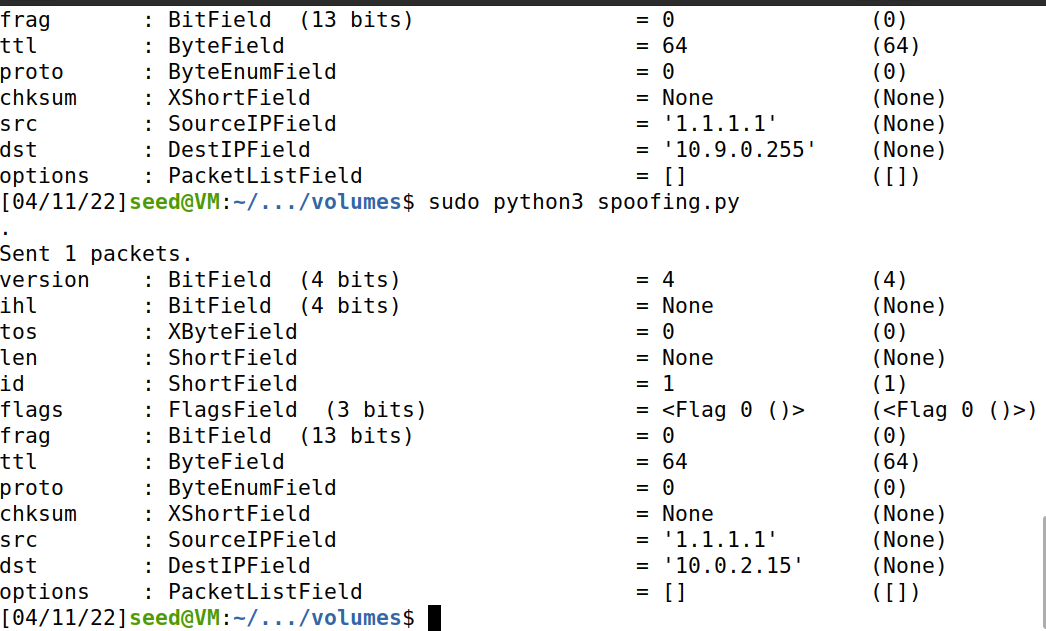
* For this example I used the filter to another VM that I am using as well as targeting specifically port 23. Using telnet I should be able to establish a connection to TCP port number 23 as well as connect to my other VM via the specified IP address. As we can see I am getting the correct TCP connections with my sniffer.



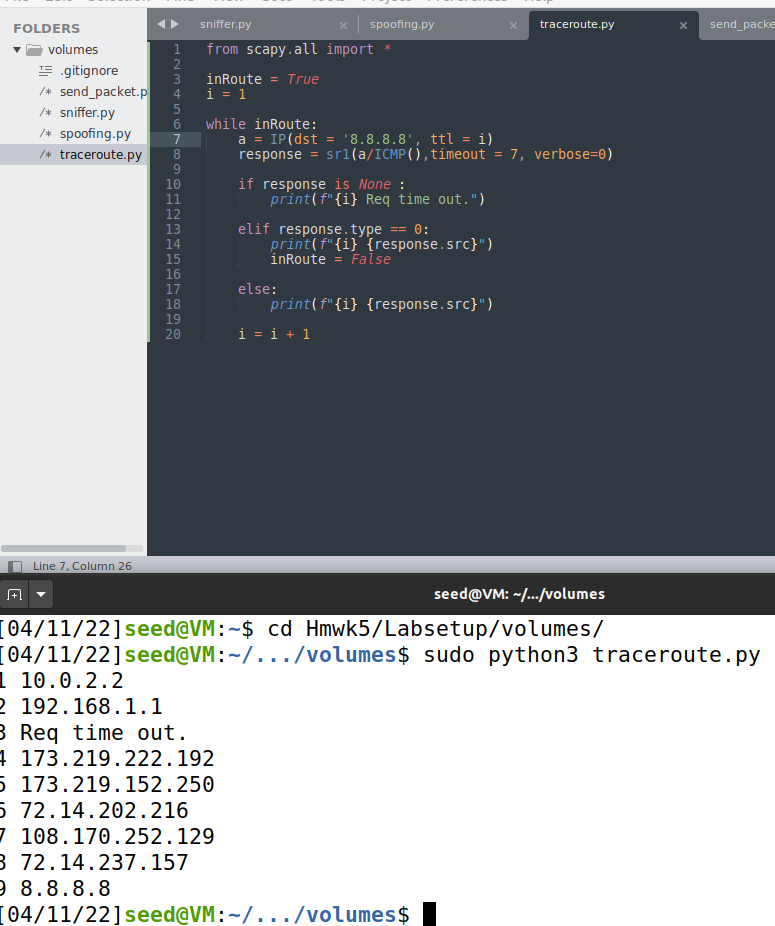


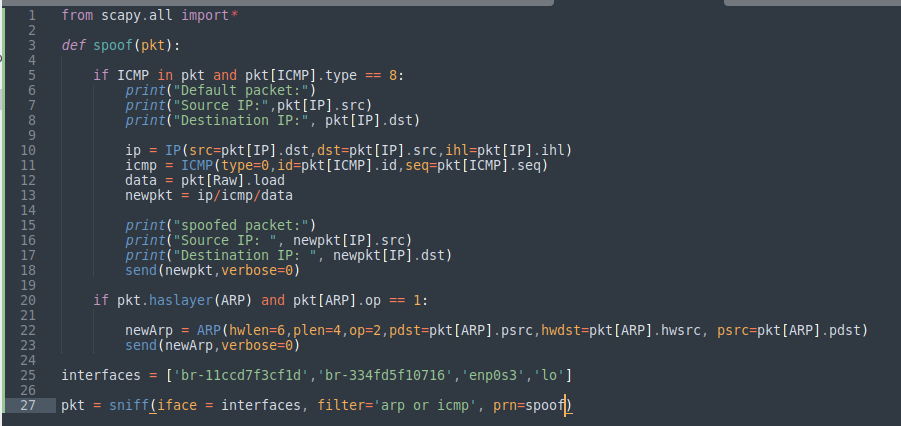
* I used a script called send\_packet.py to send a subnet packet.
* I sent the packet to a specific subnet and the program only sniffed packets that were being sent from the source machine to the destination IP of the other subnet that I used.
* As you can see here the dst matches in the second screenshot and you can see that there are packets being sent across that subnet
  1. Spoofing ICMP Packets

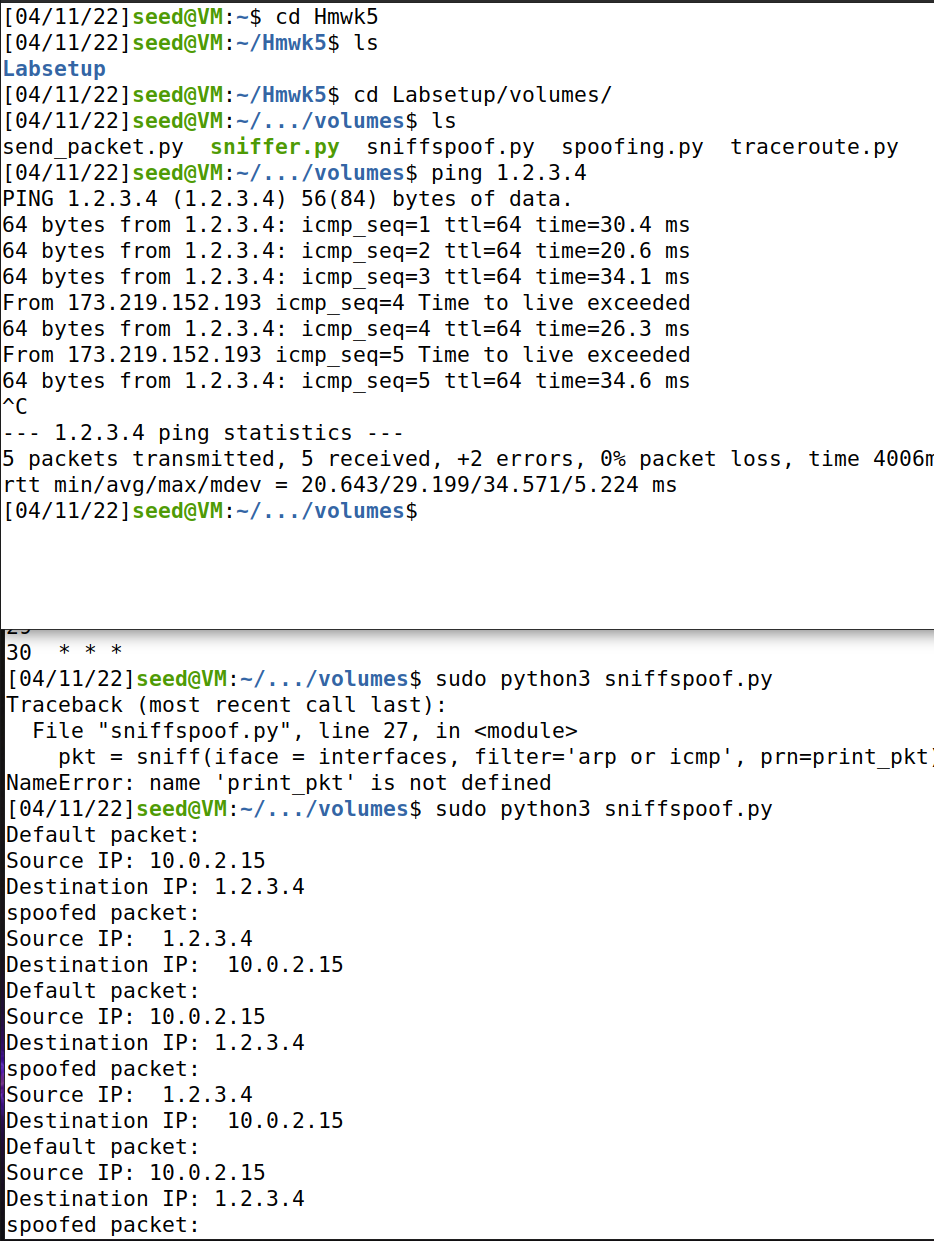


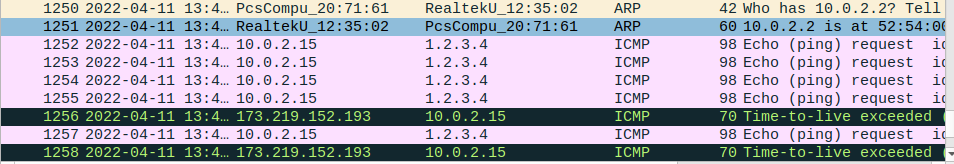




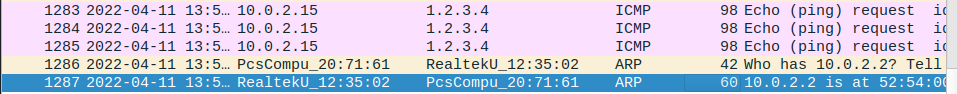
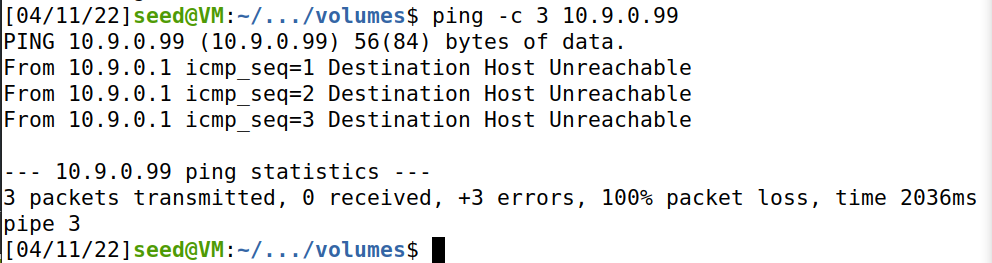
* Here I spoofed an echo request packet and sent it to another VM that was on my subnet
* I used scapy to overwrite our source IP with our own IP and sent it to my other VM.
* If you look in the sent packet field you can see that you can view the spoofed packet that I sent to my other VM that I was observing
  1. Traceroute
* 
  + The script I wrote figures out how many hops it will take to send out a packet from the source IP to the IP address destination
  + Each Line that is displayed is a different router and how many hops it takes for my packet to reach the destination IP.
  + In my case it took us 8 hops to reach the destination
  1. Sniffing and-then Spoofing



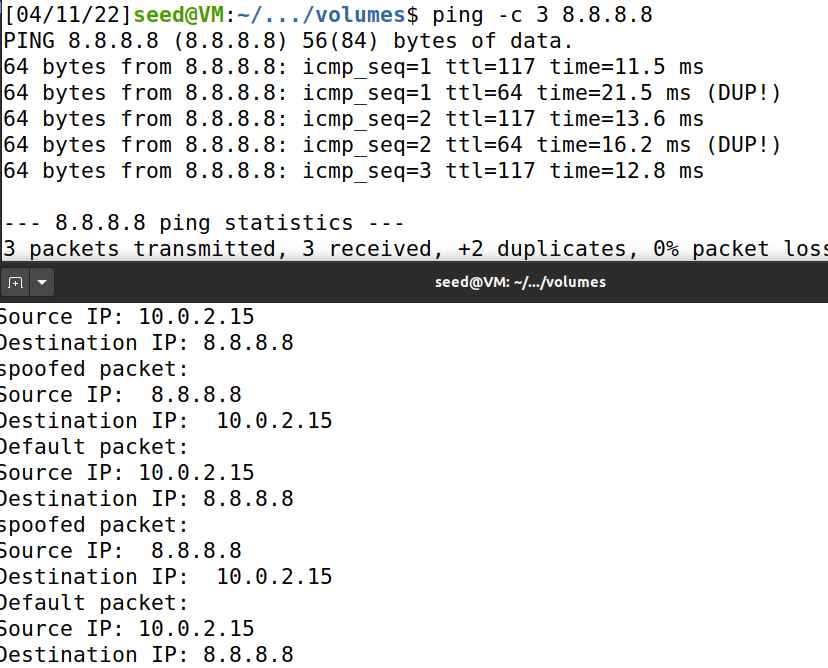


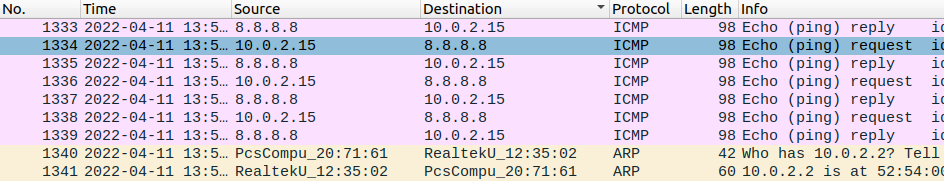


* For the first case
  + VM2 sends a ping to 1.2.3.4 and without the program we should essentially get 100% packet loss because we will never reach the destination
  + The wireshark with the ARP protocol is asking for 1.2.3.4 and asking on the network who has the IP with 1.2.3.4?
  + So with our program we return that with an answer and the ICMP packet reply goes back to VM2



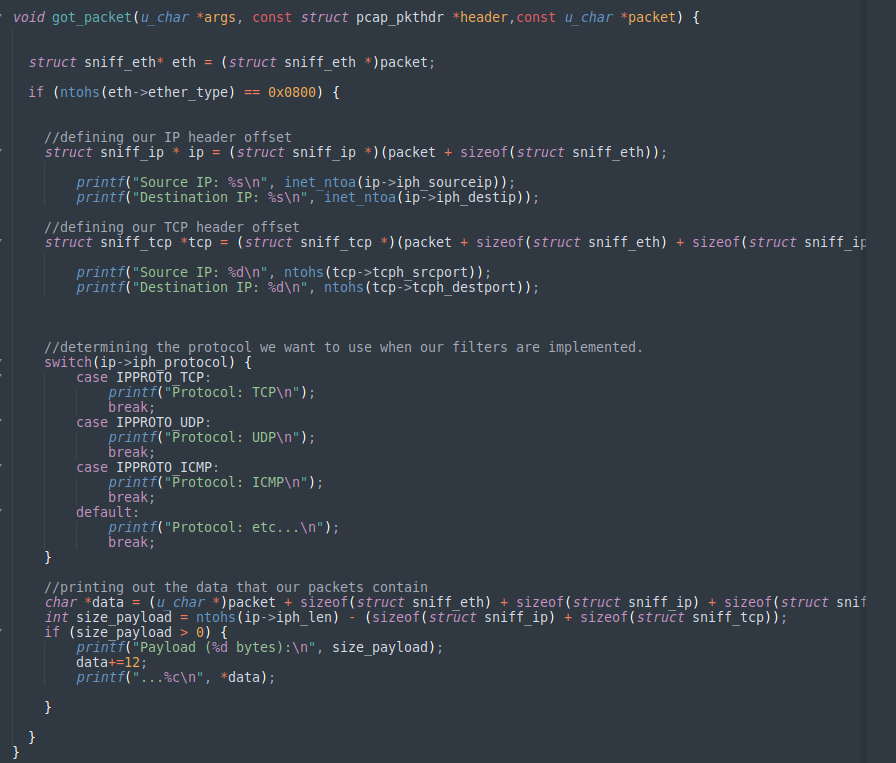
* For the second case
  + It is very similar to the first case and even though this host does not exist, the attacker VM will answer the request and send back a ICMP response packet to VM2
  + The wireshark shows that there are ICMP packets being sent over the LAN and they are being received by the host at 10

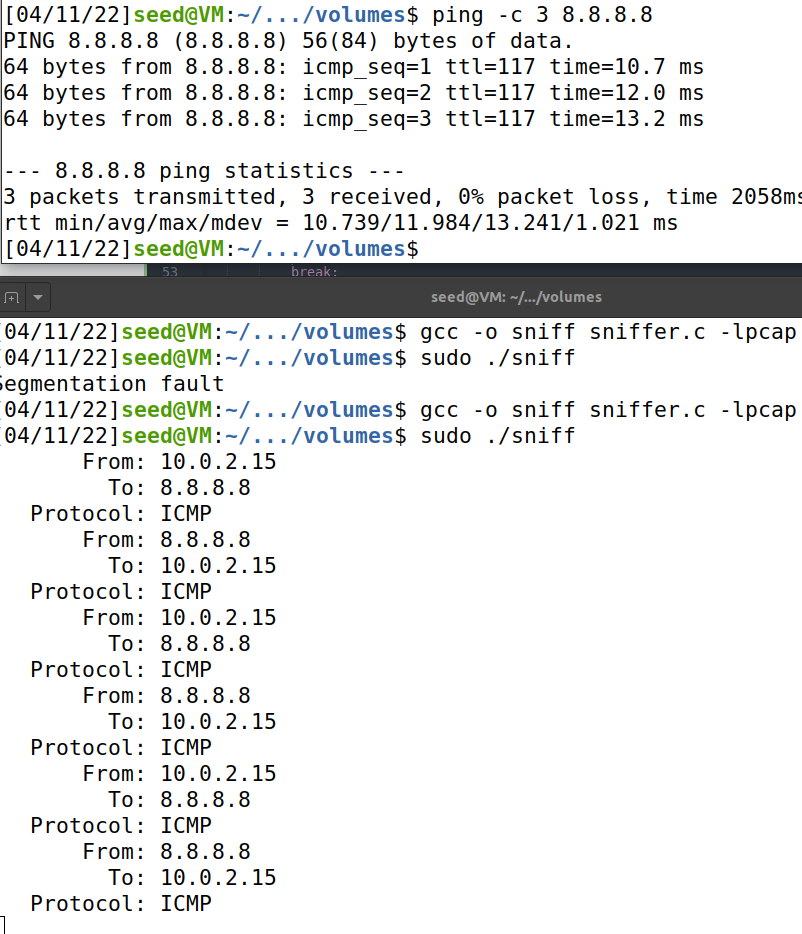


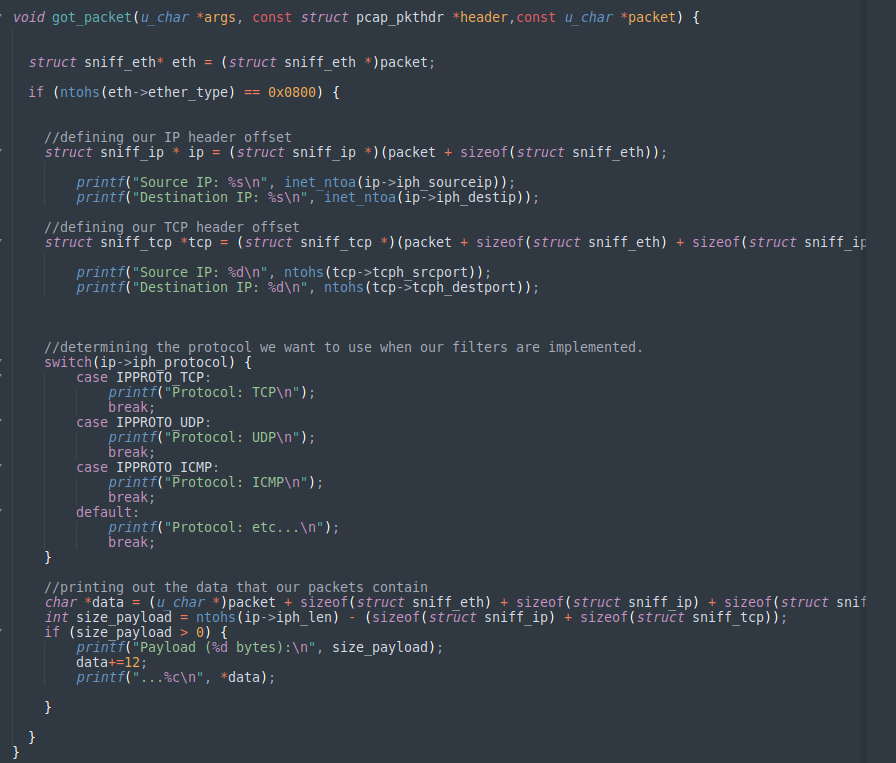
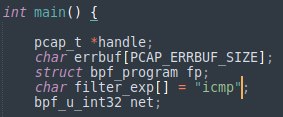
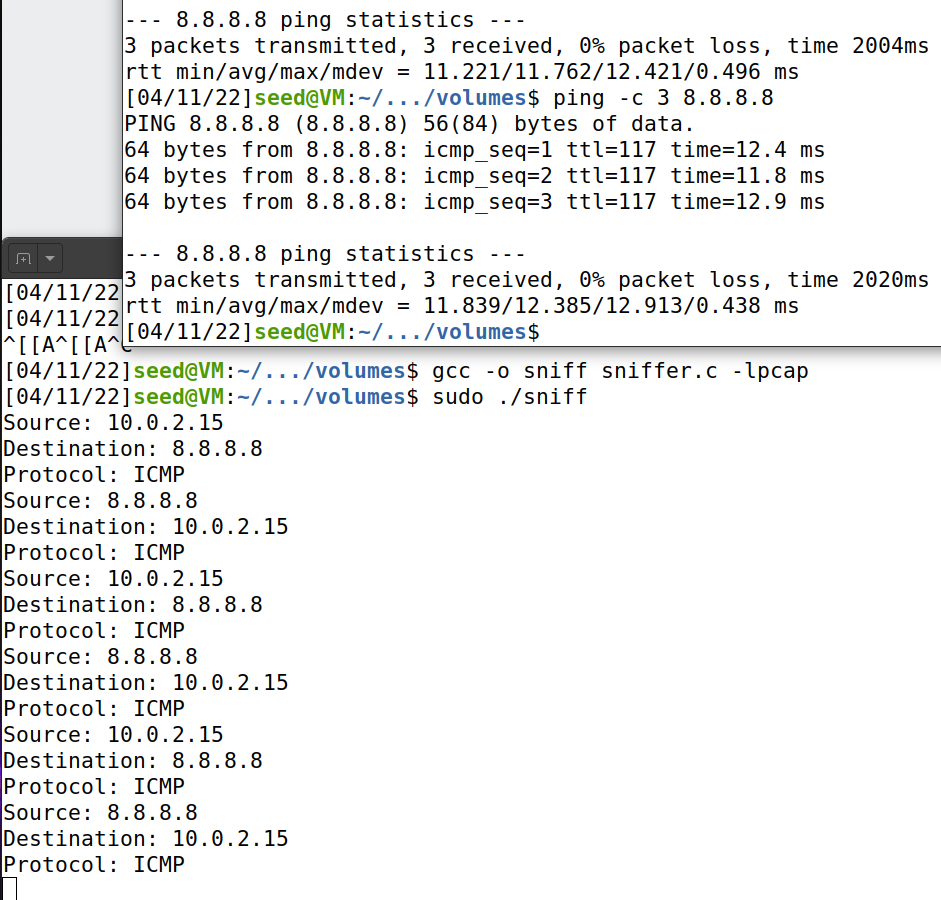


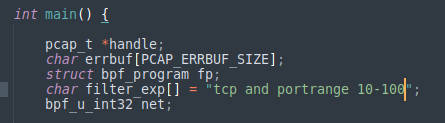
* For the third case
  + VM2 sends a ping to 8.8.8.8 which is an existing google domain. In this case we are going to get duplicate responses because the host already exists on the internet at this IP address. The real destination and the program are both responding to the source.
  + This is why we are getting the (DUP!) message since we are responding to the echo reply and request twice.

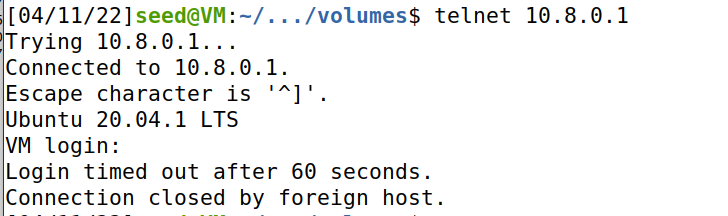
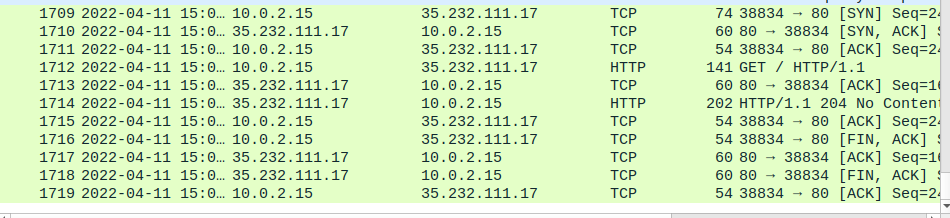
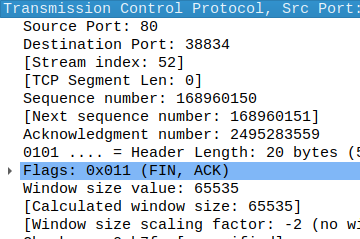
1. Writing Programs to Sniff and Spoof Packets
   1. Writing Packet Sniffing Program

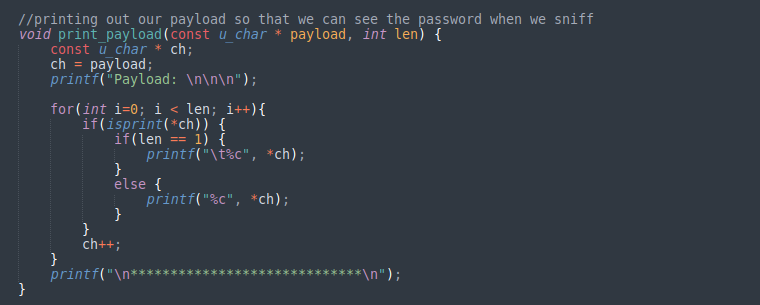


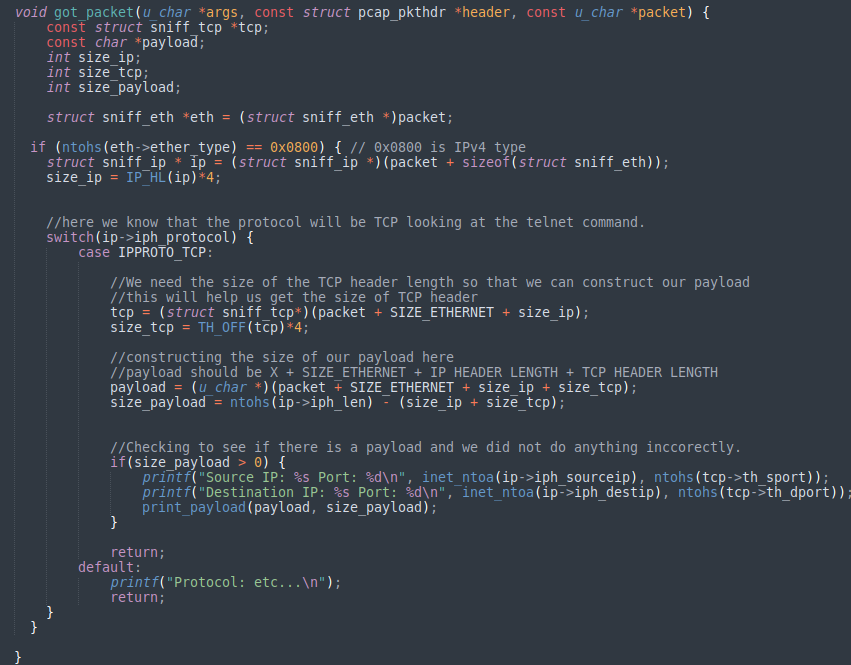


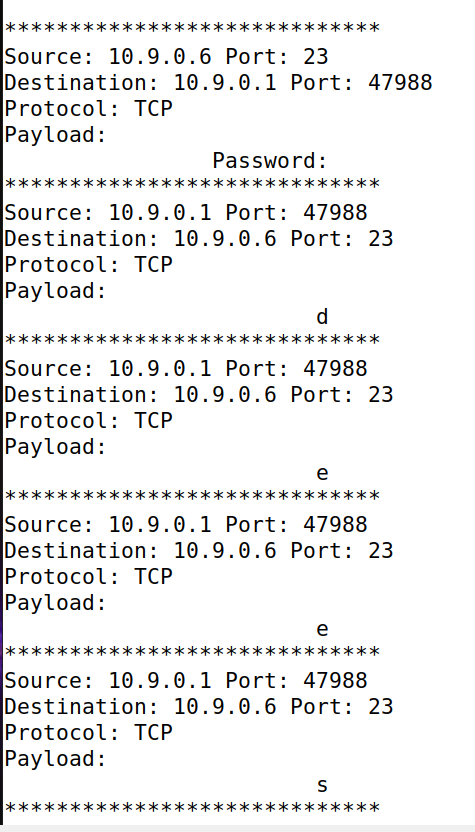
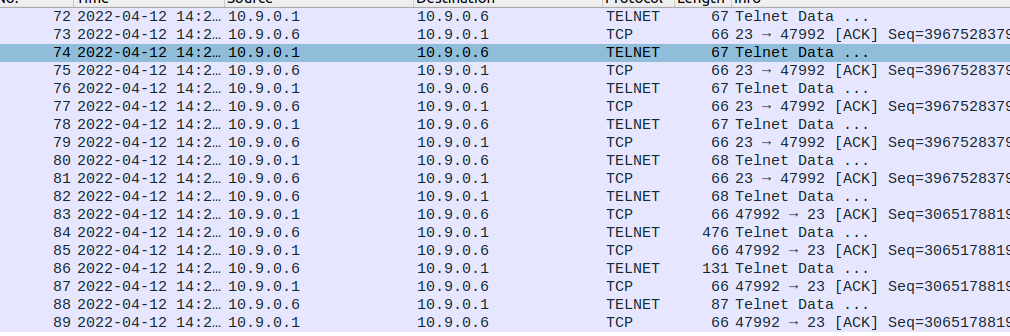
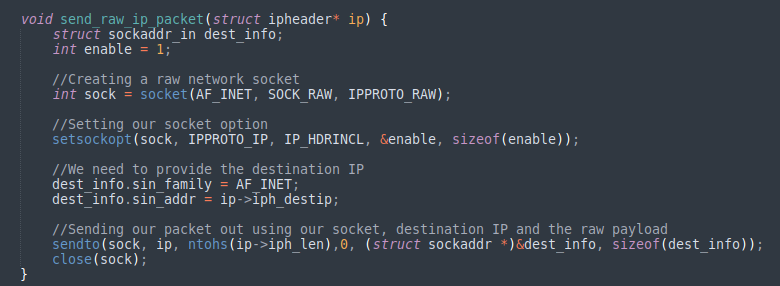
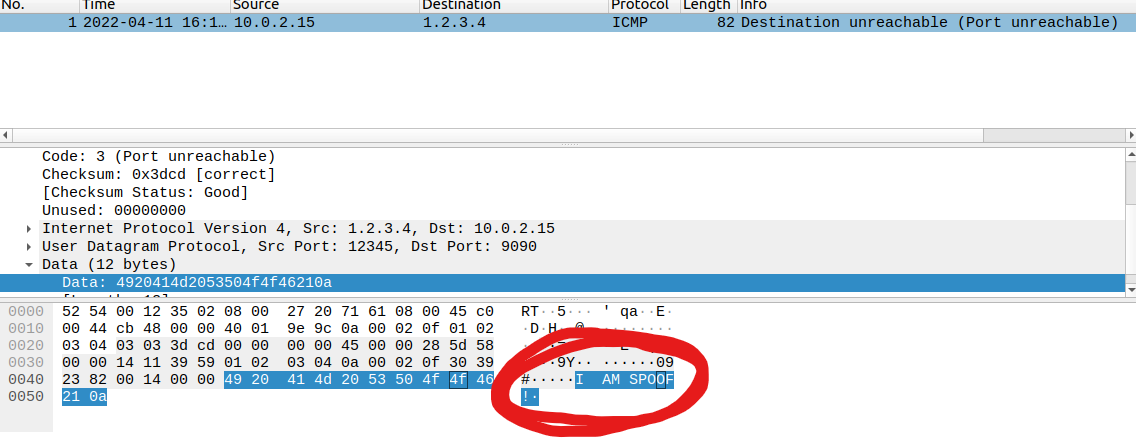
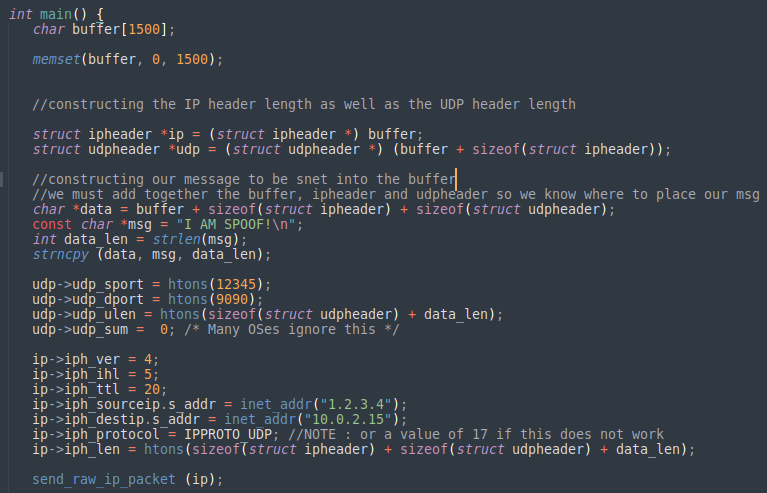
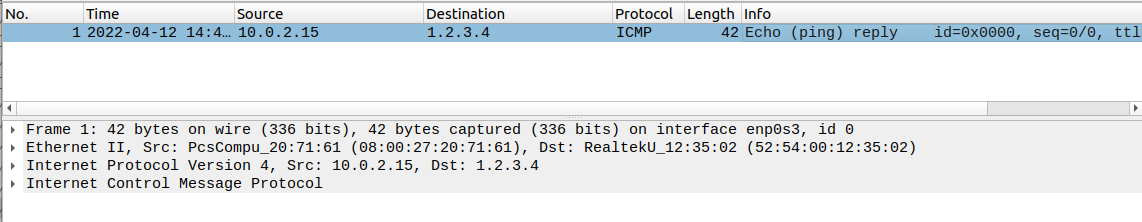
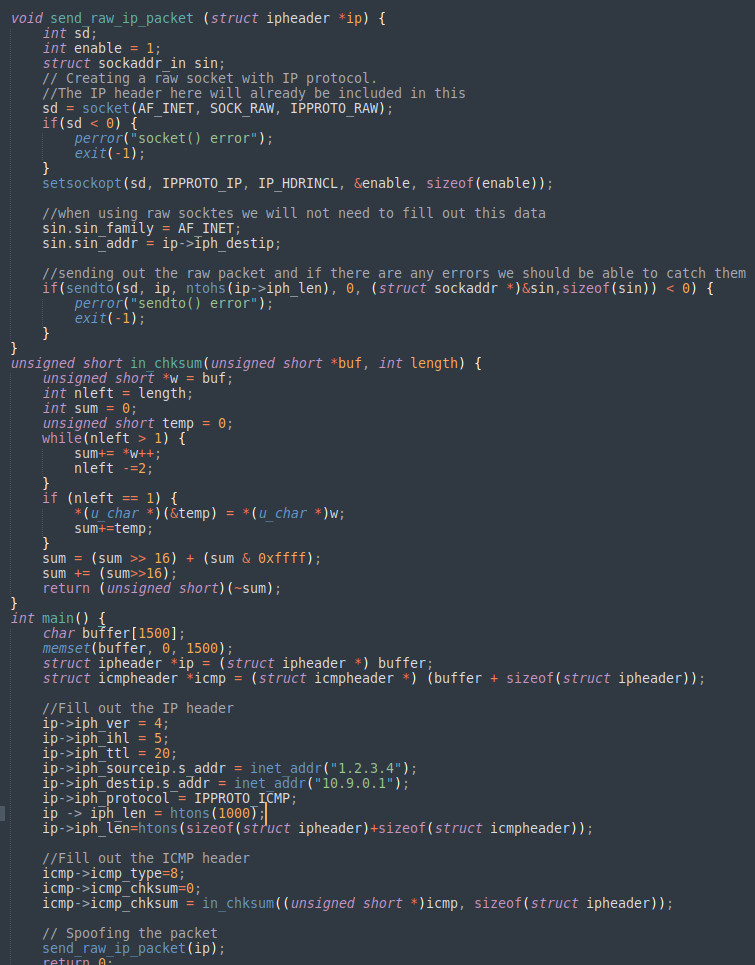
* My comments I wrote in the code explain the steps that I took when making the program that helped me achieve the correct outcome
* It involves setting ip headers and TCP and IP headers so that we can track the two endpoints of our program
* Then I am determining the type of protocol that I will need to use in my program
* After that we print out all of the information that the packet is holding in the last block of code.
* In a nutshell, the program will capture the packet, check if the header is of IPv4 type and then if it's confirmed, it will print the source for that packet.
* Q1
  + First we have to open up a live pcap session on my VM with the name enp0s3, this is done with our ‘pcap\_open\_live’ This function lets us see all the network traffic in the interface.
  + Secondly, we are setting the filter by using pcap\_compile() and pcap\_setfilter() to set our specifications
  + Lastly, we capture the packets in a loop and process them using pcap\_loop with the parameter -1 so that it runs infinitely.
* Q2
  + A root privilege is required to set up the car in promiscuous mode and the raw socket, this way we can see all the network traffic in the interface
  + If we run the program without being the root user, the pcap\_open\_live function will fail to access the device and the program will exit.
* Q3
  + The promiscuous mode is activated using the ‘pcap\_open\_live’ function
  + If you change the third parameter of the function to 0 and anything other than 0 will be ON
  + If turned off, a host is sniffing only traffic that is directly correlated with it. So only traffic that is going to, from, or routed through the host will be picked up by our sniffing software
  + If promiscuous mode is ON, it will sniff all traffic on the wire and you will get every packet that your device sees, whether or not they are directly intended for the HOST.
* Task 2.1 B
* 
  + 
    - As you can see in this portion I am capturing only the ICMP packets that are connected between my host machine and 8.8.8.8
    - This required me to use the BPF syntax and change the filter from my original sniffing software.
    - The

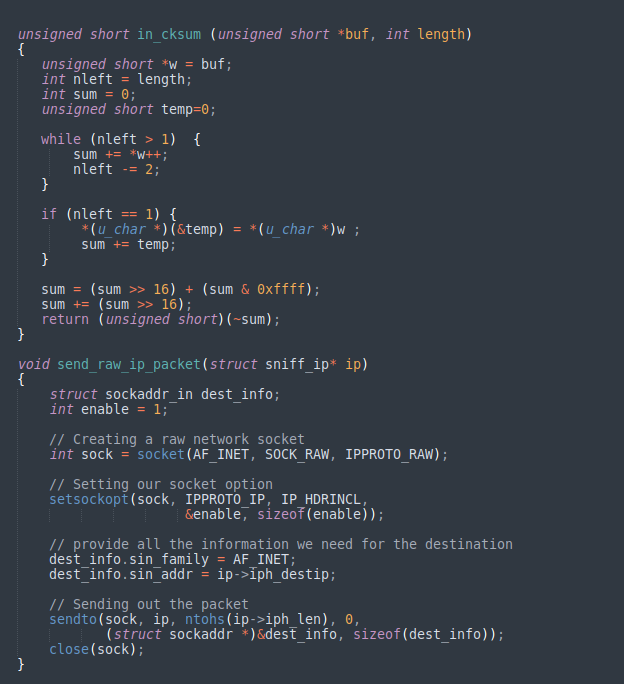


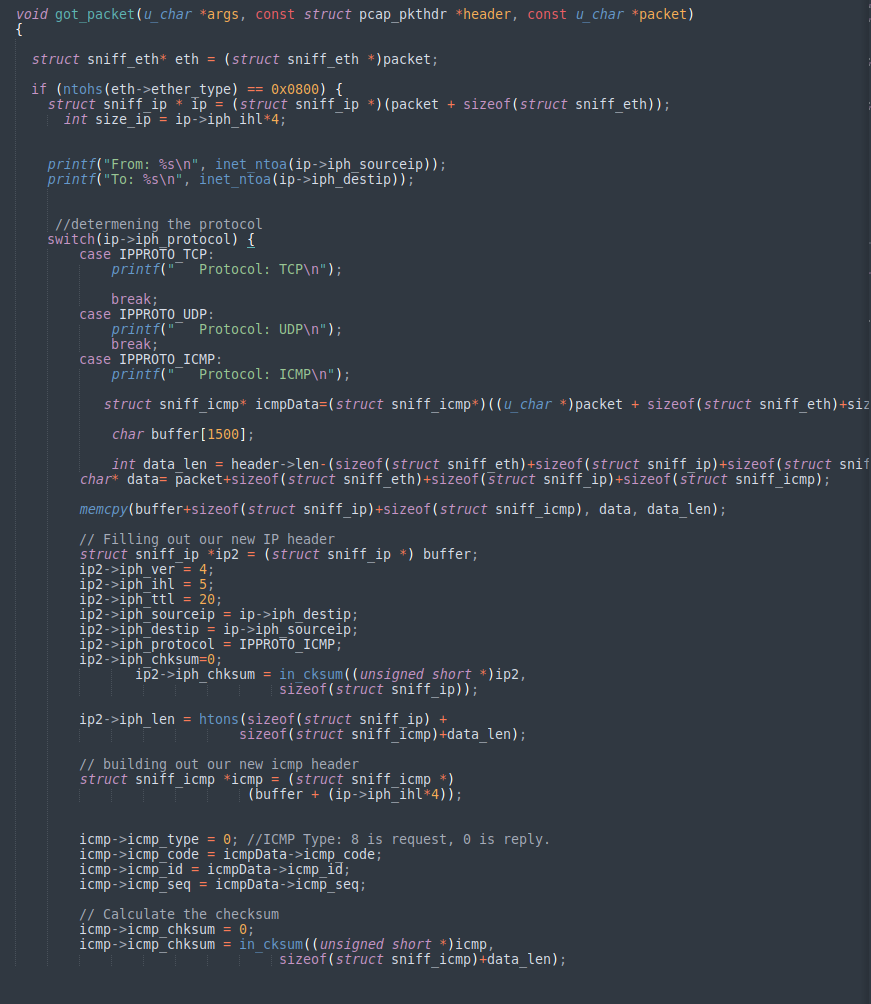
* + 
  + 
  + 
    - I was having some trouble getting this one to print out to my terminal but wire shark seemed to have the correct information
    - I used telnet to capture the TCP packet and sent it to another live VM.
    - This required me to use the BPF syntax and change the filter from my original sniffing software.
    - The wire shark pictures depict the information correctly in this case
* Correctly perform Task 2.1 C

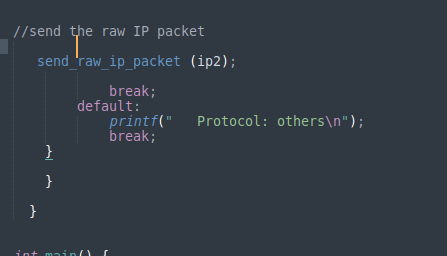


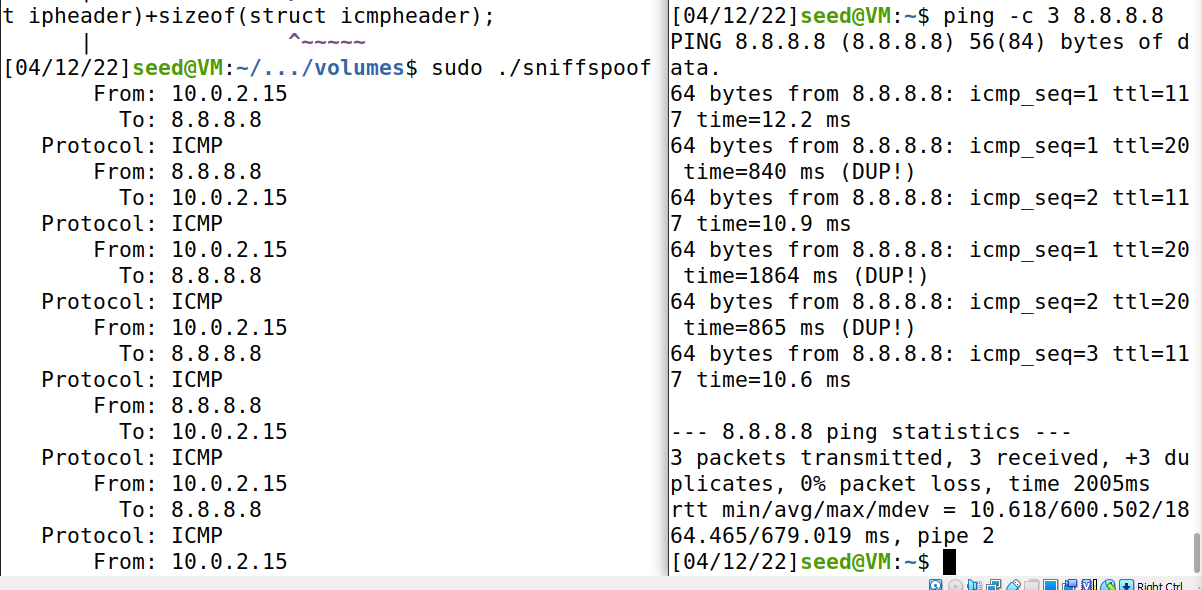


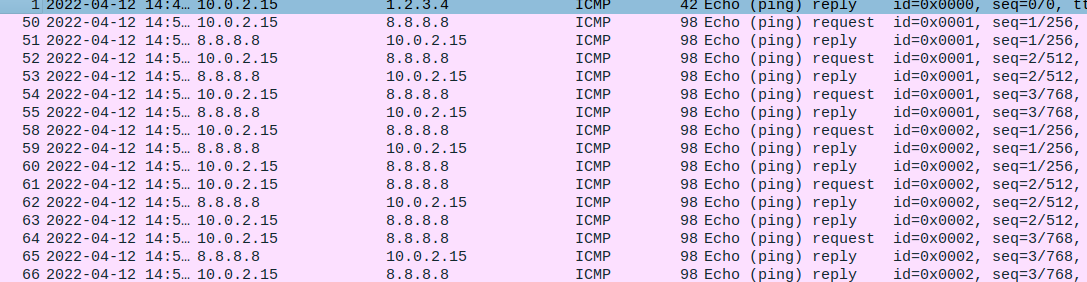
* + 
  + 
    - I capture the telnet packages and display the information accordingly
    - The payload is the most important part and I had to follow the instructions on the lab manual in order to correctly get the size to monitor and capture the TCP packets.
    - I connected to the other VM using telnet and I can monitor exactly what packets are being sent over allowing me to see the password
    - The password is on the right side of the screen and we can clearly see that it is capturing the keystrokes of the other VM and sending them over as a packet and we can now get into the victim’s VM
    - I have the wireshark screenshots showing that the TCP protocol is being called.
  1. Spoofing
* Correctly perform Task 2.2A
* 
  + 
  + I used two VM’s for this and created a spoof program using the pcap library and when I executed the spoofing machine sent a packet to the victim with the fake IP address of 1.2.3.4
  + We can see here what I circled in red was the attacking message that I sent over to the other IP
  + I also had to construct a raw socket so that I could fill in my own data and send a raw packet via that socket that contained the data I wanted it to contain
  + We had to construct the data and make sure we placed it correctly by getting the size of the buffer, udp header and the ip header so that we know where to correctly place our payload
* Correctly perform Task 2.2B
* I created a spoof ICMP request from my attacker’s machine to the source victim (10.0.2.15) and sent it to the remote server of 1.2.3.4
* The remote server responded to my ICMP request and sent it to the victim as the Echo ping reply.
* I followed my same methodology when sending the raw packets and using the raw websockets for this task
* We have to use the checksum function in order to calculate the ICMP header and the data into 16 bit words. We get the sum of all of these words and then the ones complement of the sum after we do that we can then get the correct ICMP header information.
* In my main() function I have to follow through with the same methodology as the last bullet, where I am filling out the ICMP header after I have calculated the correct information
* The IP header just needed to be filled in to match the corresponding IP’s of the attacker and the victim and I did just that.
* Q4
  + Yes, the IP packet length can be arbitrary.
  + The packet’s total length is overwritten to its original size when sent
* Q5
  + When using raw sockets, you can tell the kernel to calculate the checksum for the IP header.
* Q6
  + Root privileges are necessary to run programs that implement raw sockets. Non-privileges users do not have the permissions to change all the fields in the protocol headers.











* I followed my same methodology as the above tasks when sending the raw packets and using the raw websockets for this task
* We have to use the checksum function in order to calculate the ICMP header and the data into 16 bit words. We get the sum of all of these words and then the one's complement of the sum after we do that we can then get the correct ICMP header information.
* In my main() function I have to follow through with the same methodology as the last bullet, where I am filling out the ICMP header after I have calculated the correct information
* The IP header just needed to be filled in to match the corresponding IP’s of the attacker and the victim and I did just that.
* I combined my two scripts in order to create a sniffer and a spoofer.
* In a nutshell, the attacking machine captures all the packets that reached the program then we took those packets and processed them on our machine. Once the packet was created it was sent out to the victim, and as we can see through our spoofing, our sniffer is now picking up all of the traffic that we are sending over to the victim’s machine