SNIFf-SPOOF

SEED LABS

Contents

[Environment Setup 2](#_Toc150547814)

[Task 1 4](#_Toc150547815)

[Task 1.1 7](#_Toc150547816)

[Task 1.1A 7](#_Toc150547817)

[Task 1.1B 12](#_Toc150547818)

[Task 1.2 20](#_Toc150547819)

[Task 1.3 23](#_Toc150547820)

[Task 1.4 25](#_Toc150547821)

[Task 2 31](#_Toc150547822)

[Task 2.1 31](#_Toc150547823)

[Task 2.1A 32](#_Toc150547824)

[Task 2.1B 34](#_Toc150547825)

[Task 2.1C 37](#_Toc150547826)

[Task 2.2 41](#_Toc150547827)

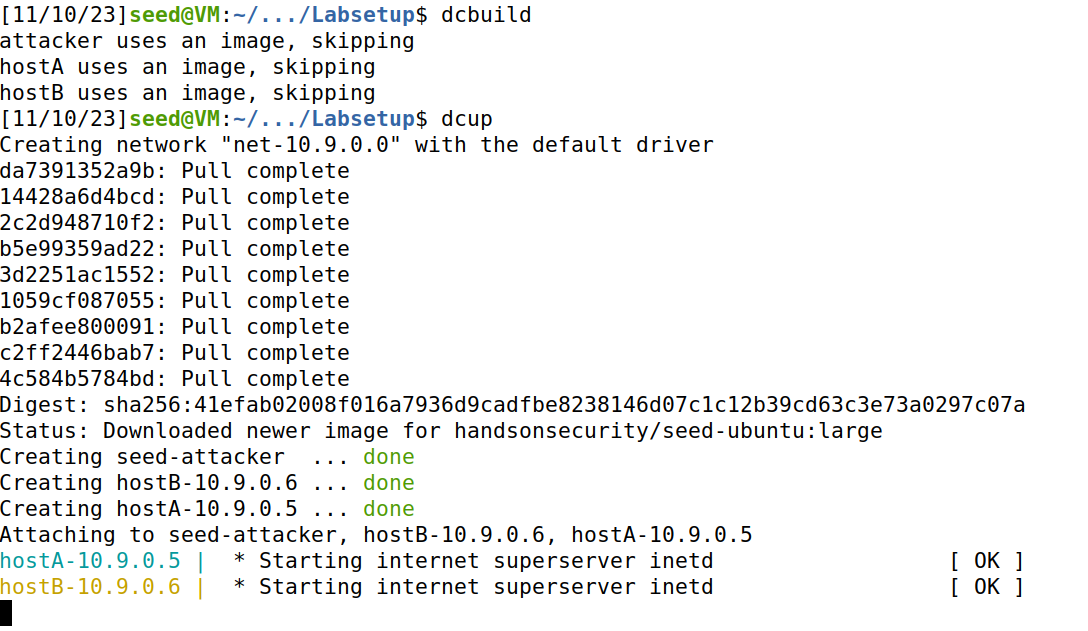
[Task 2.2A 41](#_Toc150547828)

[Task 2.2B 42](#_Toc150547829)

[Task 2.3 44](#_Toc150547830)

# Environment Setup

Setting up Dockers.

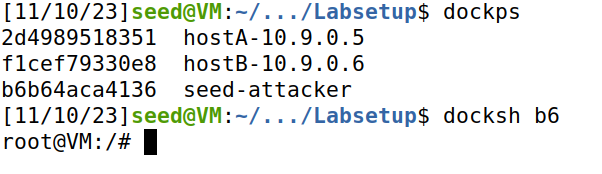


Checking Available Dockers.

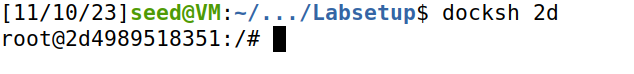
A computer screen shot of a computer error

Description automatically generated

Setting up Attacker Terminal.

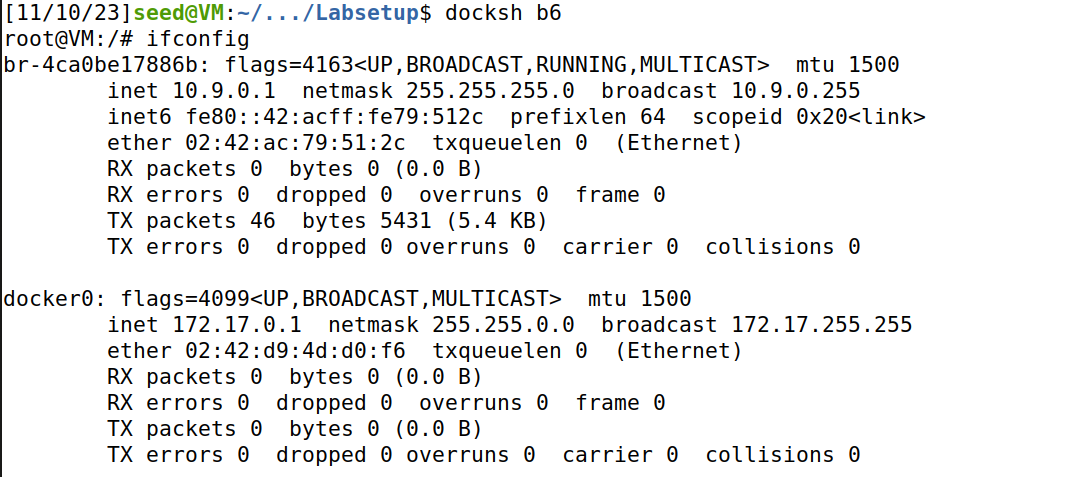


Setting up HostA Terminal.



Setting up HostB Terminal.





# Task 1

Ran the commands given in the manual and we get the IP details of the packet.

A screen shot of a computer

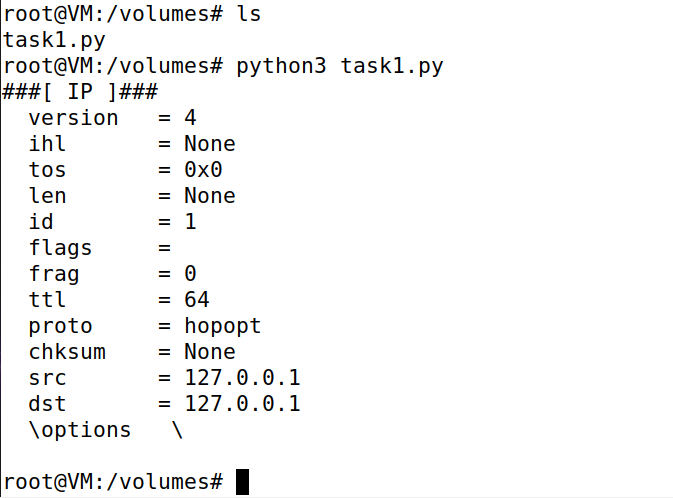
Description automatically generated

Wrote a script as the tasks were performed manually will be done automatically with the script and volumes directory.

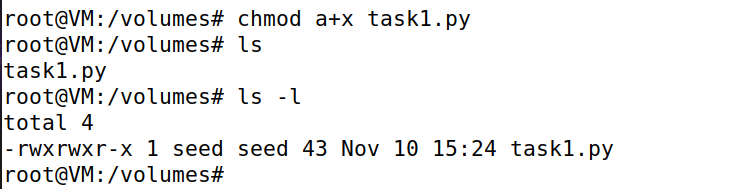
A screen shot of a computer

Description automatically generated

Now executing the script and we see that it is running the same way we did manually.



Now making the script executable and confirming which it evidently becomes as shown in the screenshot.

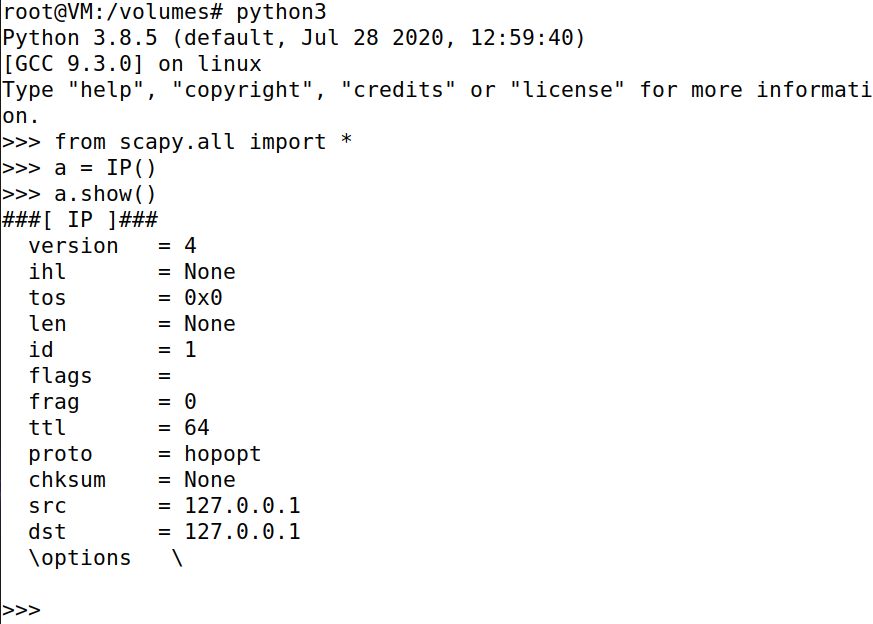


When confirming in host is also shows it is executable now.

A screenshot of a computer code

Description automatically generated

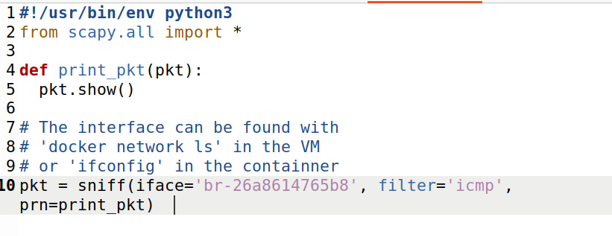
Now trying the way taught in manual if we need to change the code frequently.



## Task 1.1

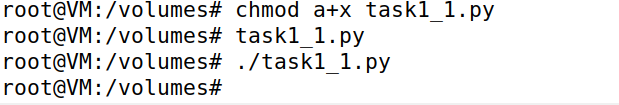
Creating python files in **volumes** via host to perform the task while setting interface to the one found above in the Environment Setup.

Now changing the file to executable and launching the code.

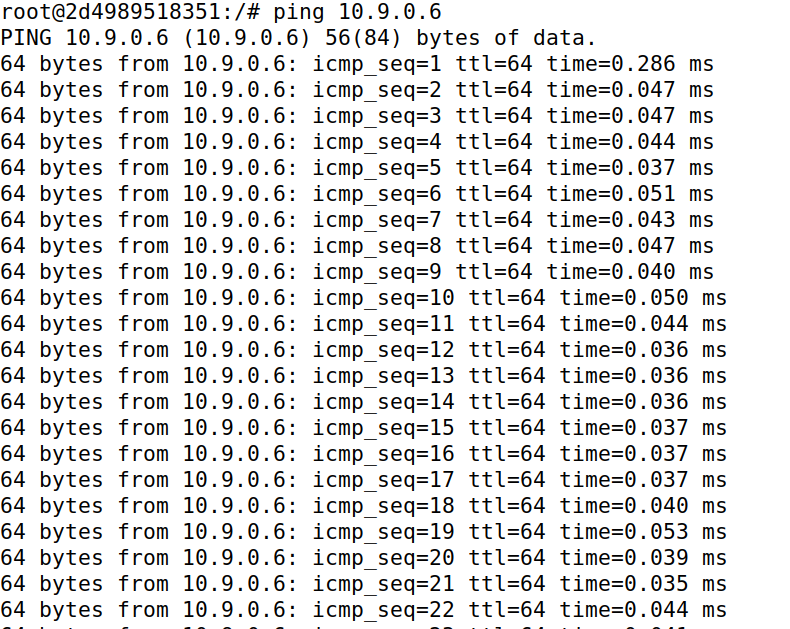


### Task 1.1A

Making the code file for the task executable and launching the code from which we can see there was an error which was fixed after I added the part in line 1 as shown in the screenshot above. After that I observed that the code is now sniffing so I moved to test if it is working.



Sending the packets from HostA to HostB.



As evident in the attacker terminal the packets have been sniffed.

A white rectangular frame with black numbers

Description automatically generated

Modified the previous code to see the number of packets received and now running as seed user.

A screenshot of a computer code

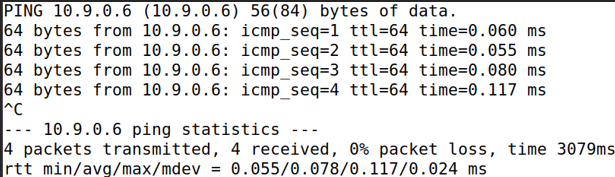
Description automatically generated

While running as normal seed user I couldn’t get permission to launch the script in attacker machine.

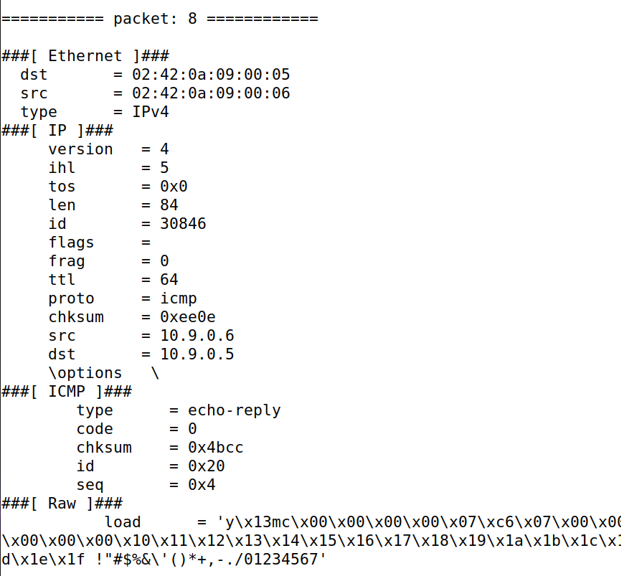
Launching the script.



Again sending the packets from HostA to HostB.



Now we see the packet numbers being displayed indicating 8 packets as 4 were transmitted and 4 received as it is visible in the above screenshot. Therefore, our code sniffed total 8 packets.

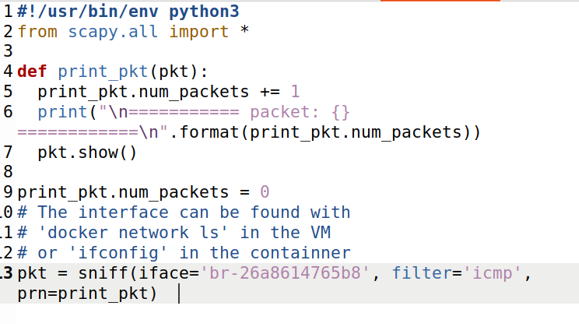


### Task 1.1B

#### Capture Only ICMP Packets

Using the previous screenshots as we caught the ICMP Packets only in that.

Modified the previous code to see the number of packets received and now running as seed user.



While running as normal seed user I couldn’t get the permission to launch the script.

Launching the script.

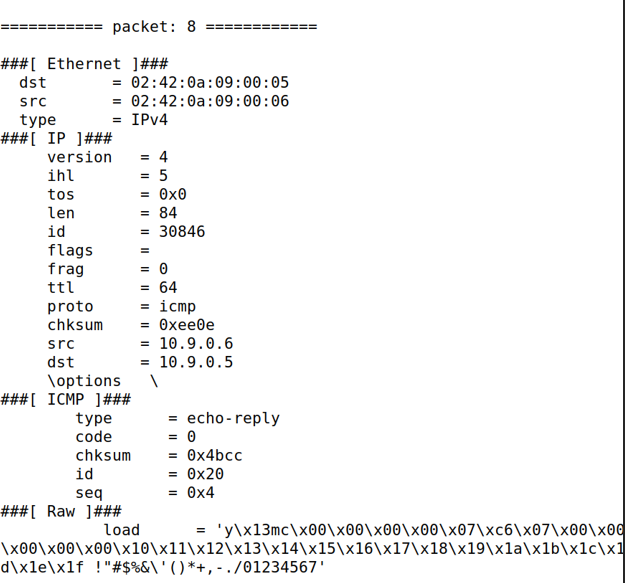


Again, sending the packets from HostA to HostB.

A screenshot of a computer code

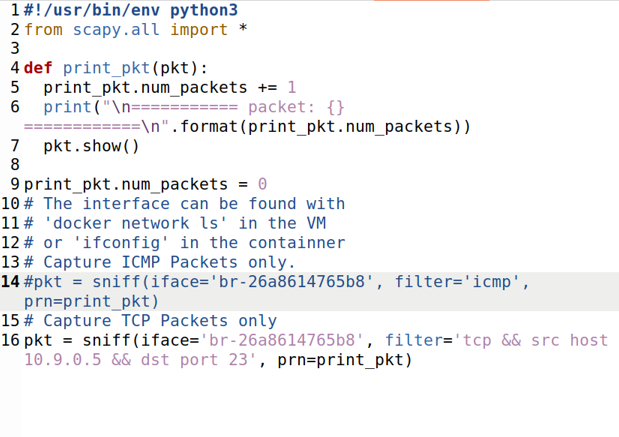
Description automatically generated

Now we see the packet numbers being displayed indicating 8 packets as 4 were transmitted and 4 received as it is visible in the above screenshot. Therefore, our code sniffed total 8 packets.

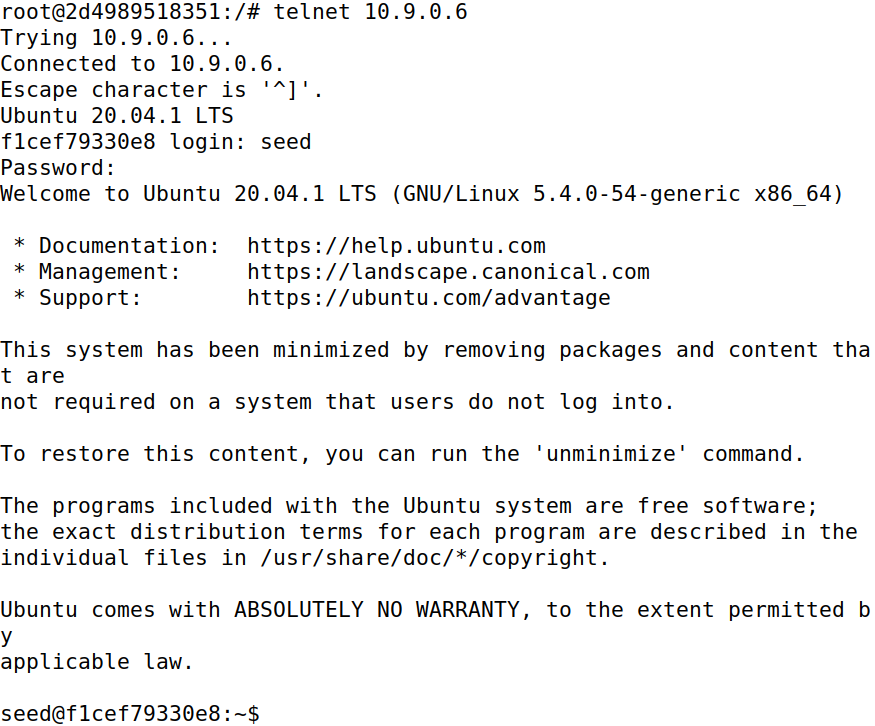


*Capture any TCP packet that comes from a particular IP and with a destination port number 23*

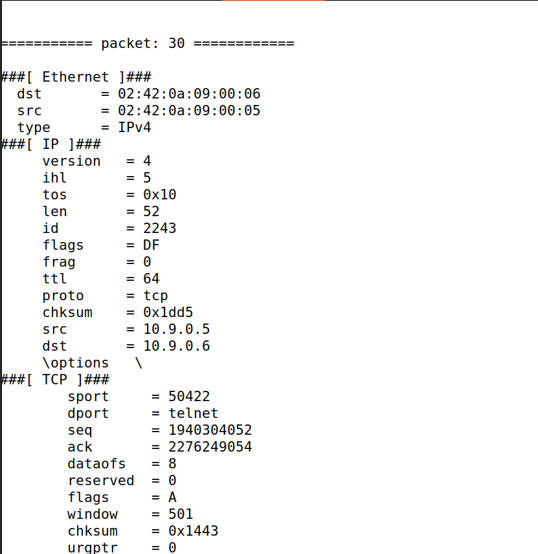
Modifying the code for it to capture only TCP packets while providing source IP and destination port.



Connecting HostA with HostB via telnet the TCP port.

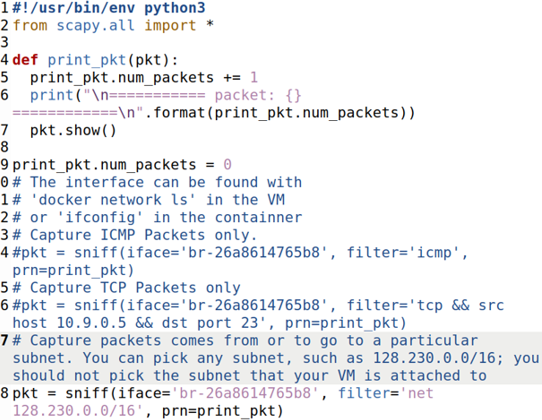


Now in the attacker terminal we can see the TCP packets being captured from HostA to HostB.



#### Capture packets comes from or to go to a particular subnet. You can pick any subnet, such as 128.230.0.0/16; you should not pick the subnet that your VM is attached to

Modifying the code to the need of the task.



Launching the Script.

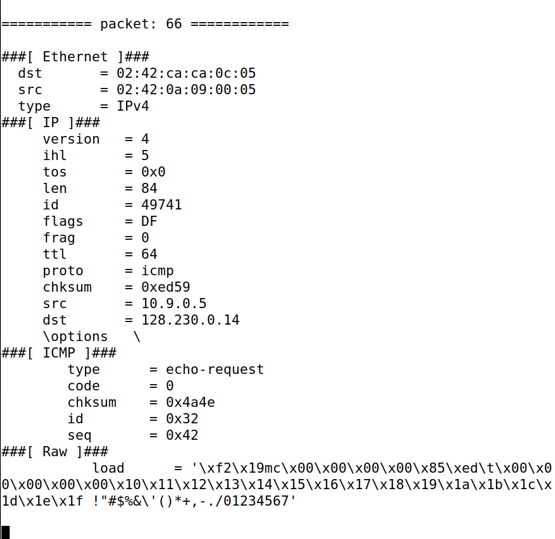


Closing the telnet connection between HostA to HostB and sending ping from HostA to an IP belonging in the respective subnet.

A black and white text

Description automatically generated

So far I caught 64 packets.



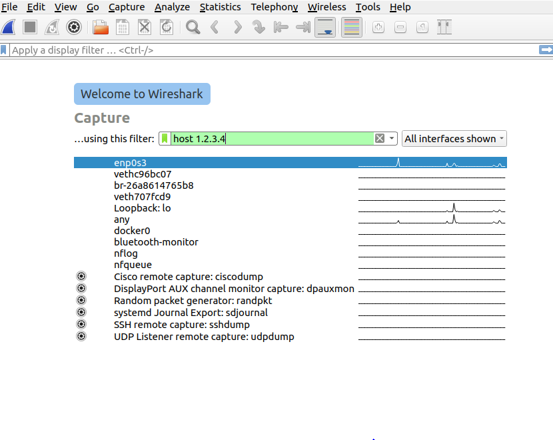
## Task 1.2

Here, I wrote a script as provided in the manual and modified a bit.

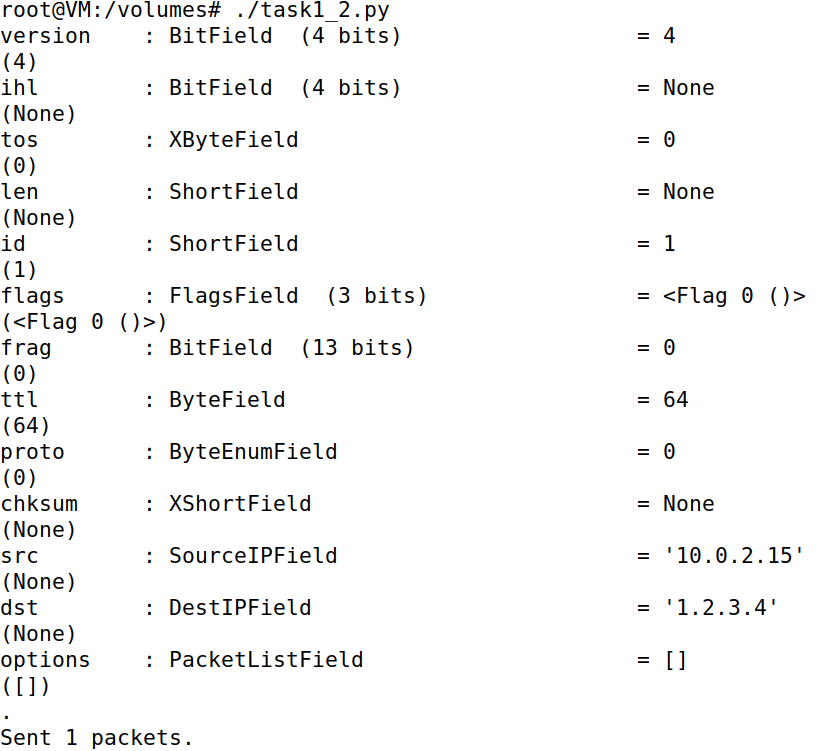
A screen shot of a computer

Description automatically generated

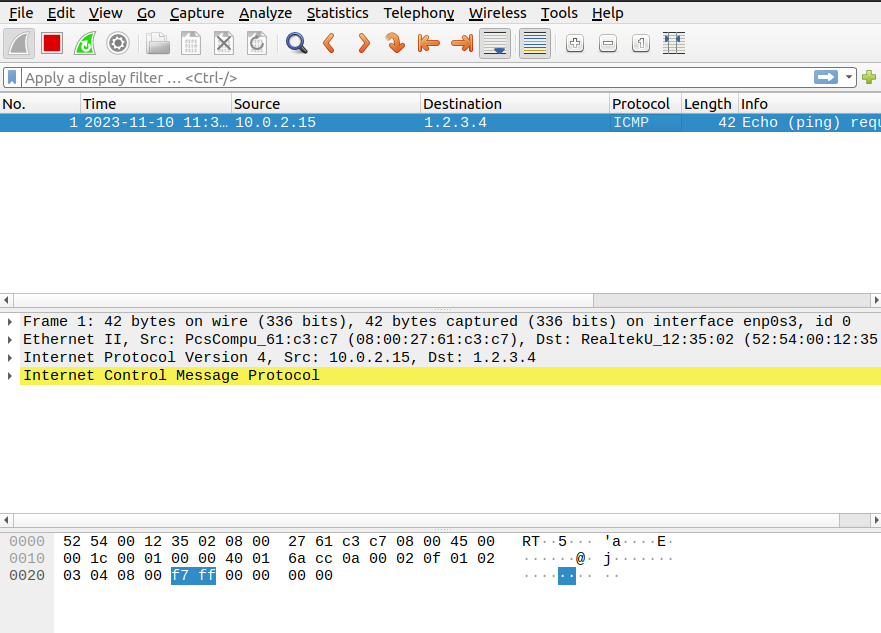
Setting up Wireshark.



Making the script file executable and launching it makes it send a packet to the destination IP from Attacker Terminal.



And we have caught our packet which is spoofed.



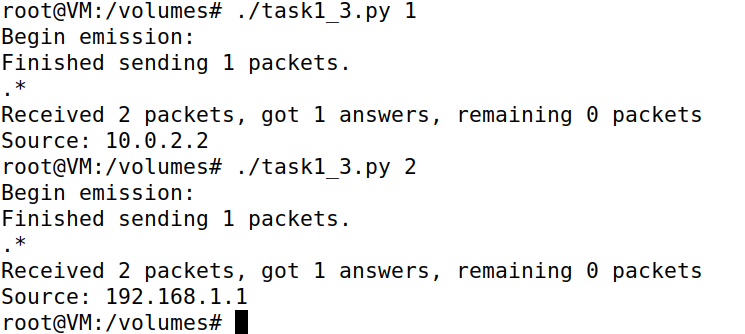
## Task 1.3

Now I wrote a script using the instructions provided in the manual for the task.

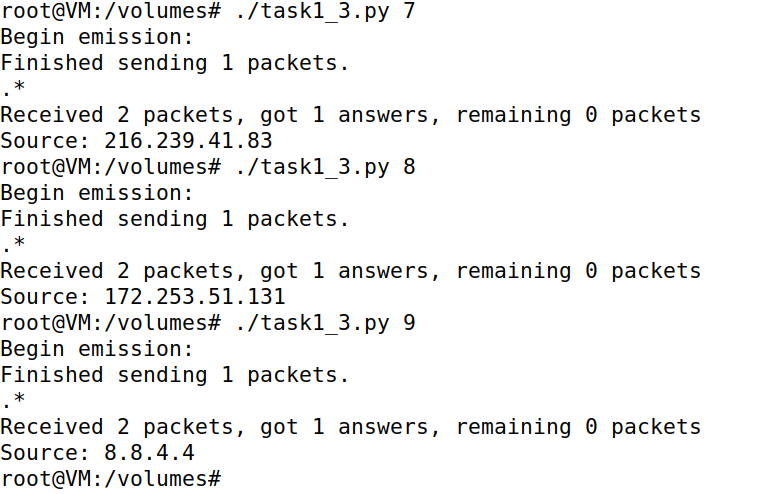
A screenshot of a computer program

Description automatically generated

Now making the code executable and launching the attack manually with packet reference numbers.



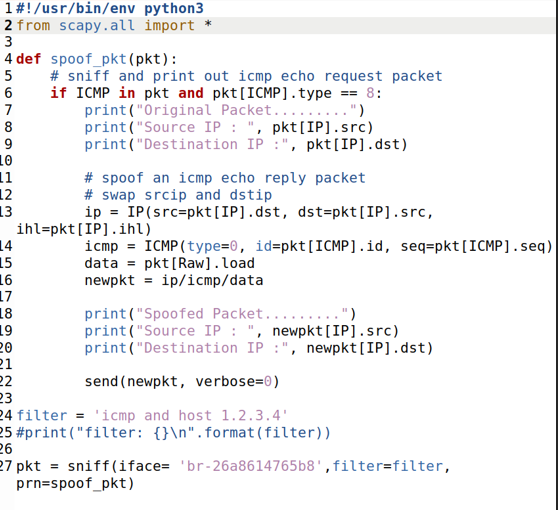
Here, after quite some packets we have reached the target IP address as source.



## Task 1.4

#### Non-existing host on the Internet

Here, I wrote a script for the purpose of sniffing and spoofing to target a non-existing host on the internet.



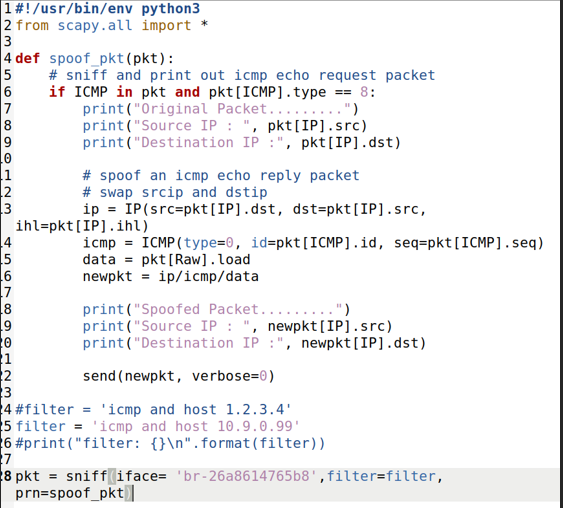
Initiating pinging process on the target from HostA.

When I launched the script from the attacker machine after making it executable, I started receiving a response.



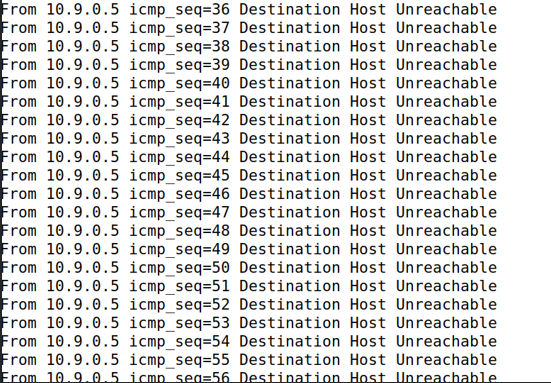
#### Non-existing host on the LAN

Modifying the code to perform the task on a non-existing LAN host.



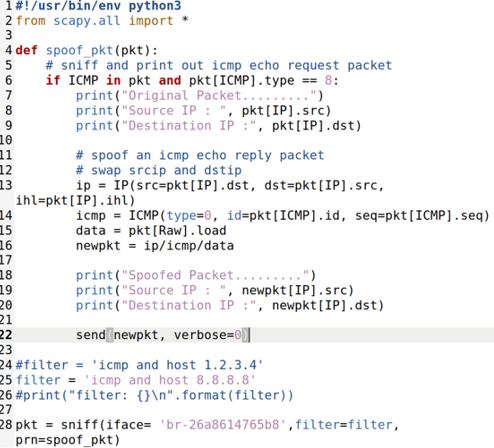
Launching the script in attacker’s terminal.

Initiating pinging process to the target but it is unreachable, and no packet can be traced.



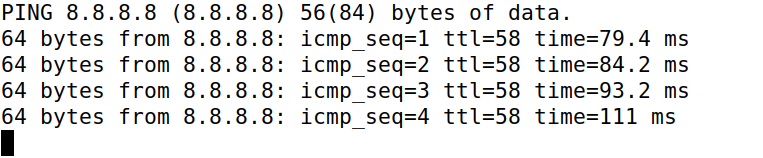
#### Existing host on the Internet

Modifying the script to target an existing Host on the internet.

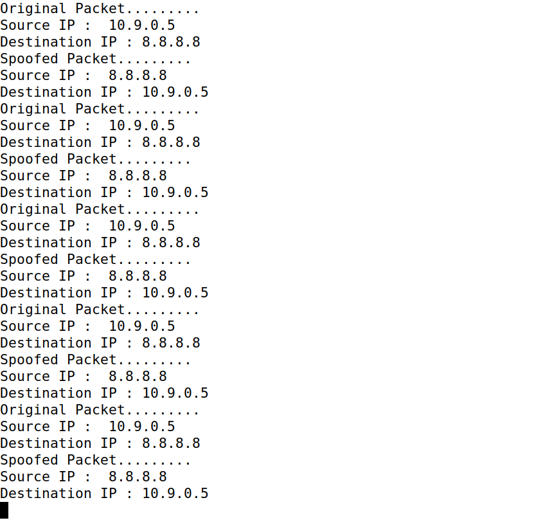


Launching the script in attacker’s terminal

Initiating pinging the target.



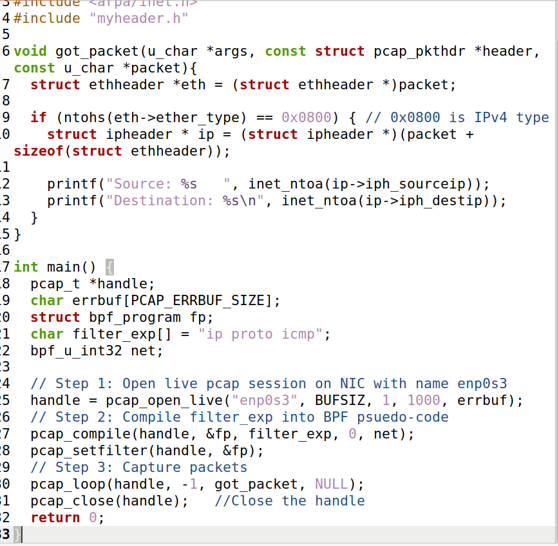
And we start seeing the packets sniffed and spoofed details.



# Task 2

## Task 2.1

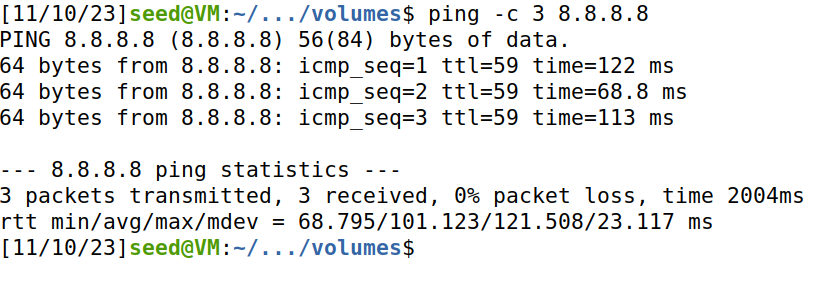
Wrote the code as per manual instructions for the sniffer.



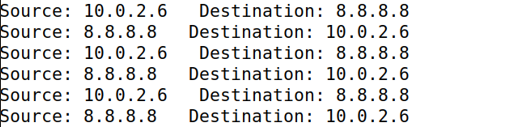
Using new terminals for the task. Moreover, Compiling and executing the sniffer.



Sending 3 packets ping to 8.8.8.8



Checking back on the sniffer we get the results of those packets.



### Task 2.1A

#### Question 1

Initially, I initiated a live pcap session on the NIC named enp0s3, achieved through the pcap\_open\_live function from the pcap library. This function facilitates the observation of the entire network traffic on the specified interface while also binding the socket. Subsequently, I implemented the setting of a filter using the pcap\_compile() method, which compiles the filter string (str) into a filter program. The pcap\_setfilter() method is then employed to specify this filter program.

Finally, I captured packets in a loop and processed them utilizing the pcap\_loop function. The use of -1 in the loop signifies an infinite loop, allowing continuous capture and processing of packets. This comprehensive process enables the monitoring and analysis of network traffic on the specified interface.

#### Question 2

#### Executing the program without root user privileges leads to a failure in the 'pcap\_open\_live' function's attempt to access the device, resulting in an error for the entire program. Consequently, establishing the card in promiscuous mode and utilizing raw sockets necessitates root privileges. This enables comprehensive visibility into the entire network traffic on the interface, underscoring the importance of root access for these specific functionalities.

#### Question 3

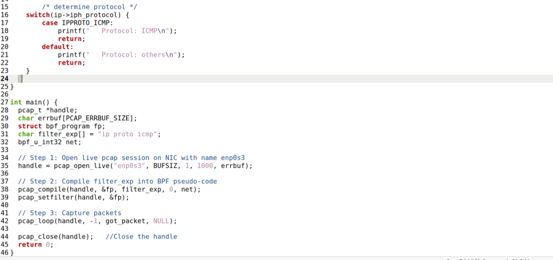
The promiscuous mode is an inherent feature of the chip in the Network Interface Card (NIC) within the computer and is activated through the 'pcap\_open\_live' function. By adjusting the third parameter of the 'pcap\_open\_live' function, setting it to 0 turns promiscuous mode OFF, while any value other than 0 activates it (ON). When promiscuous mode is turned OFF, the host only captures traffic directly related to it. Conversely, when turned ON, the host captures all traffic on the network, receiving all packets the device detects, regardless of whether they were intended for that specific device or not.

### Task 2.1B

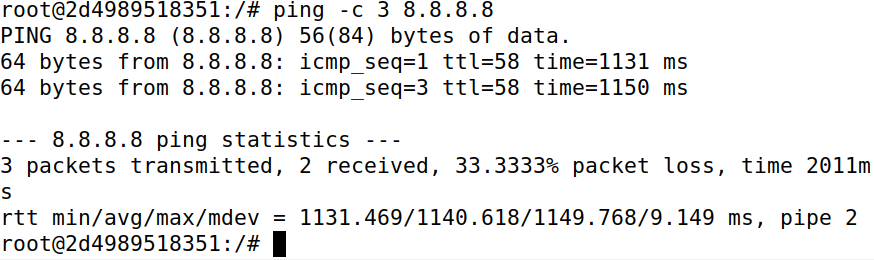
*Capture the ICMP packets between two specific hosts*

Writing the code to capture the ICMP packets between two specific hosts.

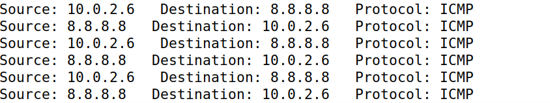




Compiling the program and executing with root privilege then I sent 3 packets from HostA terminal to IP 8.8.8.8



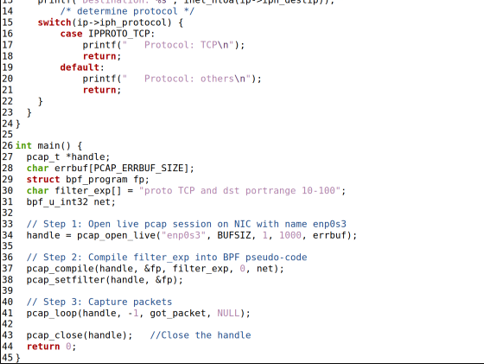
Moving back to the host terminal it is visible that the packets were caught.



#### Capture the TCP packets with a destination port number in the range from 10 to 100

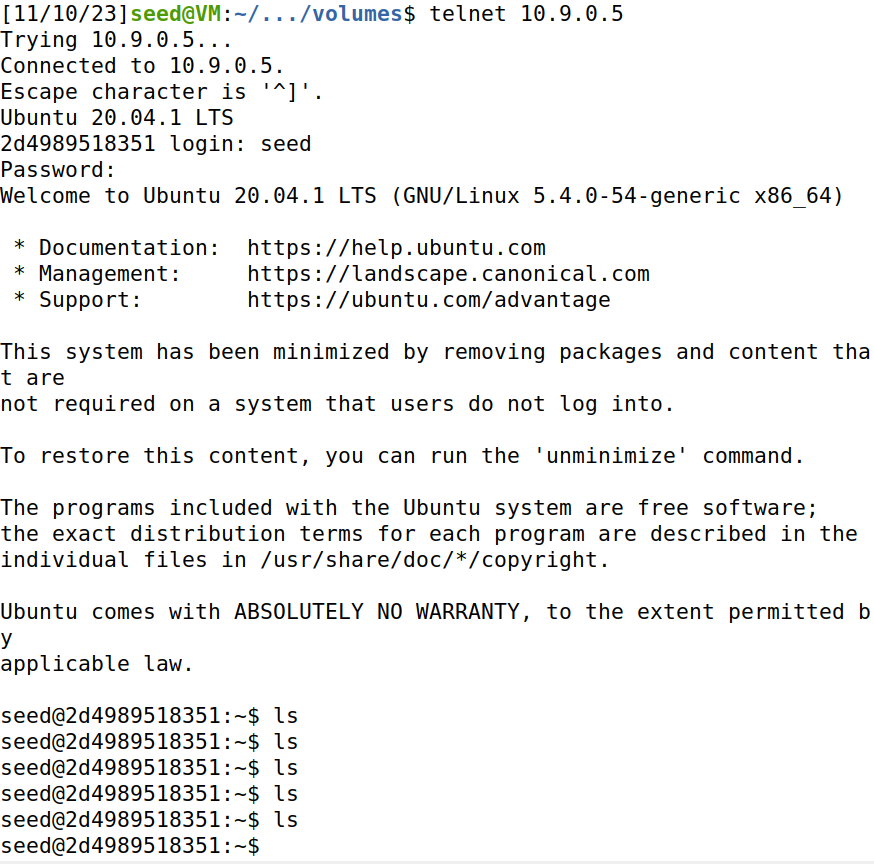
Wrote the code to capture the TCP packets with a destination port number in the range from 10 to 100.





I compiled the code and executed it with root privileges.

Connected the Host with HostA.



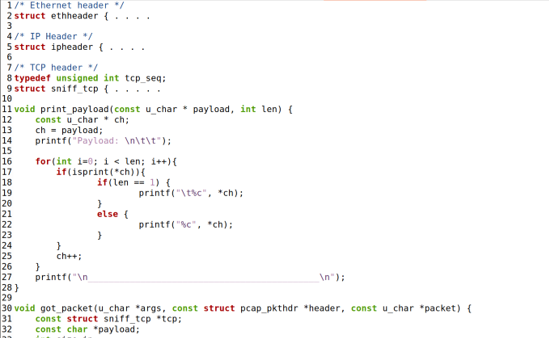
I got these packets as a result after connection and some activity.

A number of numbers on a white background

Description automatically generated

### Task 2.1C

Wrote the code for sniffing passwords.



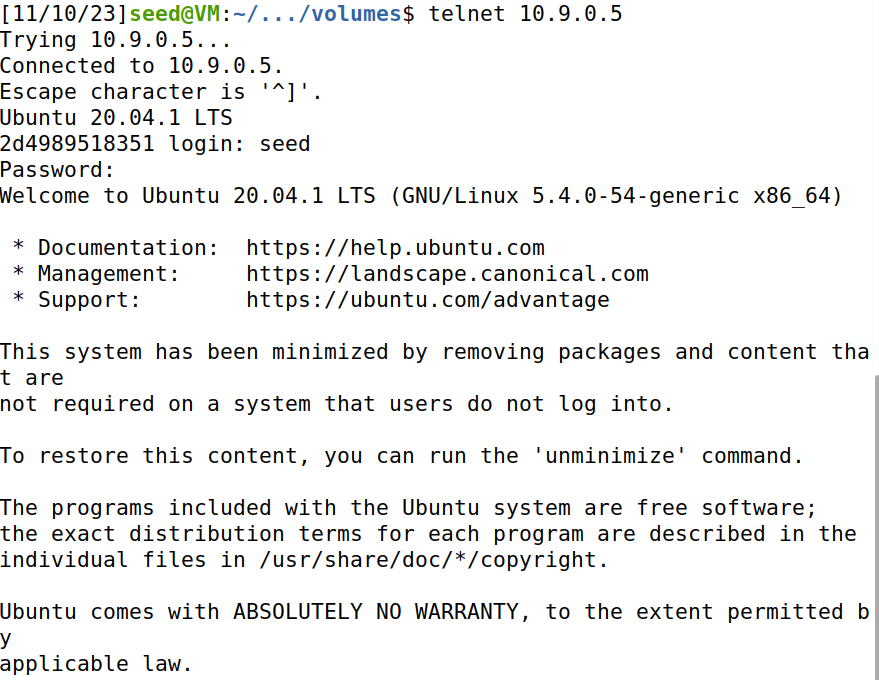
A screen shot of a computer code

Description automatically generated

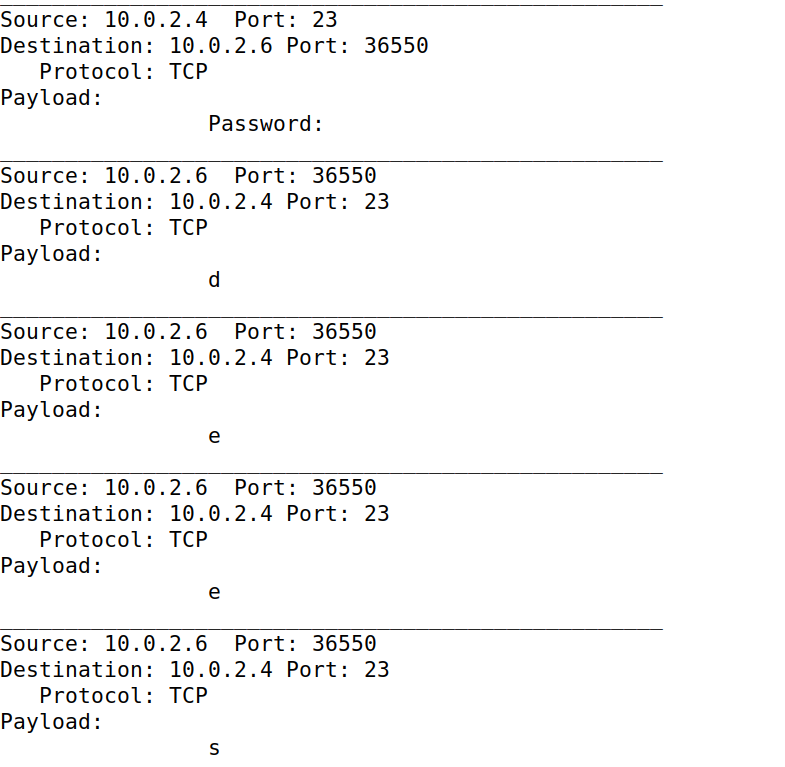
A computer screen shot of a computer code

Description automatically generated

Compiled and executed the code with root privileges and establishing telnet connection with Host from another VM.



And finally found the password in Host terminal.



## Task 2.2

### Task 2.2A

I wrote the code for spoofing program.





Further when I compiled and executed the program, I caught a packet from the source to the destination IP set to 1.2.3.4 and after a few tries I found the UDP packet coming to Host.

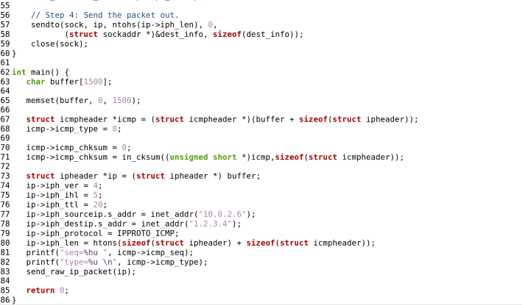
### Task 2.2B

Wrote the code for spoofing an ICMP Echo Request.



A computer screen shot of a computer code

Description automatically generated



Upon execution I found Echo with 42 length from 10.0.2.6 to 1.2.3.4 while in the terminal the sequence and type were provided as 8.

#### Question 4

Yes! The IP packet length field has the flexibility to be set to any arbitrary value. However, it's important to note that when the packet is sent, the total length of the packet is overwritten to its original size. This means that, despite setting an arbitrary value in the IP packet length field, the actual transmitted packet will conform to its original size.

#### Question 5

#### When employing raw sockets, it is possible to instruct the kernel to compute the checksum for the IP header. By default, the option for calculating the checksum in the IP header fields is set to 0 (`ip\_check = 0`), allowing the kernel to handle the computation. Unless explicitly changed to a different value, the kernel will automatically perform the checksum calculation. However, if you choose to alter this value, you will then need to employ a checksum method to handle the calculation yourself.

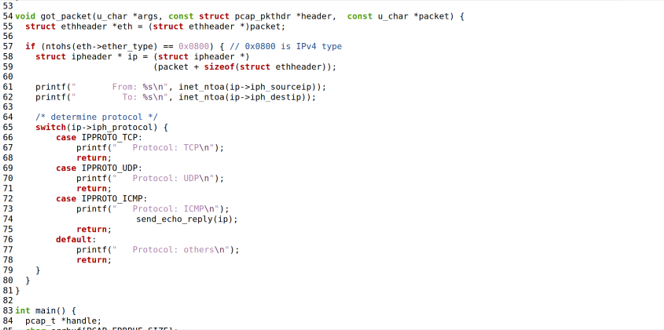
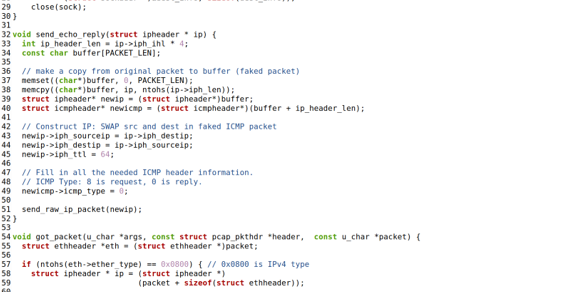
#### Question 6

Root privileges are essential for executing programs that utilize raw sockets. Regular users do not possess the necessary permissions to modify all fields within protocol headers. With root privileges, users gain the capability to manipulate any field in packet headers and access sockets, including placing the interface card in promiscuous mode. Running the program without the required root privileges will lead to a failure in the socket setup process.

## Task 2.3

Wrote the program for Sniffing and then spoofing.

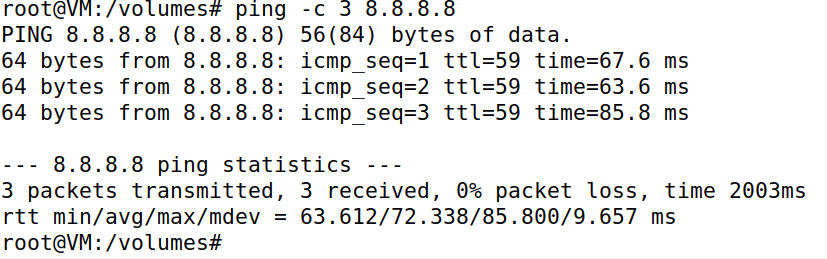




A screenshot of a computer code

Description automatically generated

Compiling and executing the code with root privileges. Sending Packets from HostB to IP 8.8.8.8



And we get the result.

