**ACS 54500 Cryptography and Network Security - Lab 2**

**Task 1.1A**

Before we do anything, we have to make sure that we set the right infertace in code. To find this, we run ‘ifconfig’ and change the interface.

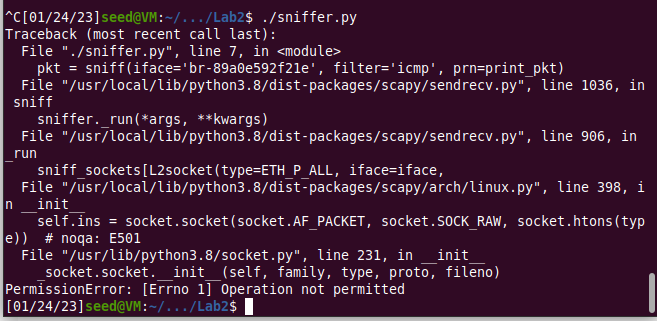
In Task 1.1A, we were supposed to send out an ICMP packet from Host A in the Seed VM, and sniff the packets sent. The below code shows how this would be done:Graphical user interface, text, application

Description automatically generated

The first part of this task is run using the command ‘sudo’ and results are shown below. Text

Description automatically generated

When the same file is run without using sudo, an error is thrown saying ‘operation not permitted’. This is because Scapy requires sudo or root priviledges to run code and without this it gives back a traceback error.



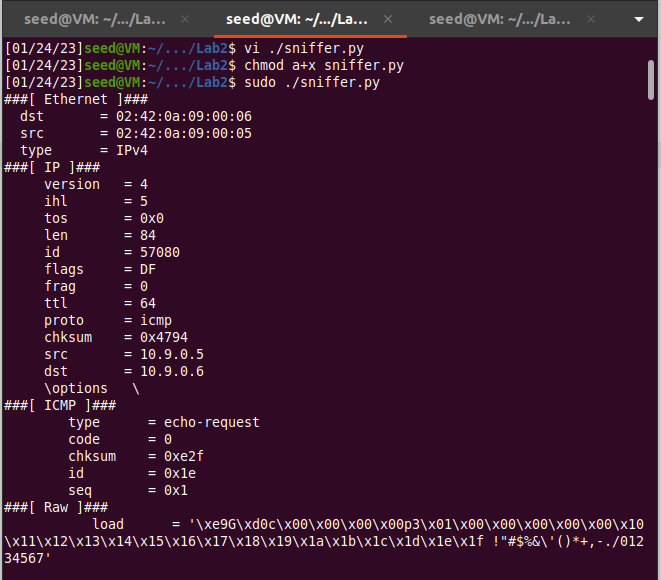
**Task 1.1B**

There are three filters that need to be applied to the sniffing program in this task.

The first one is to sniff and capture only ICMP packets. The below code shows how to do this -



When we ping host B from host A, only ICMP packets are captured, and information is displayed as follows:



To specify filters based on given situation, I referred to the BPF manual documentation as referenced in the SEED Labs Manual.

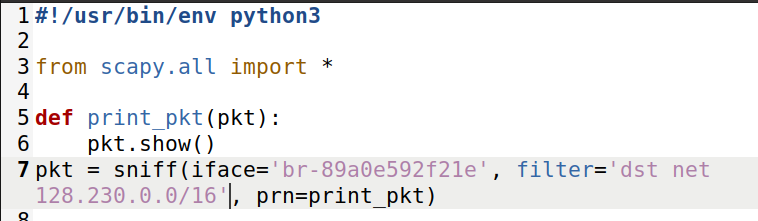
In the second filter, we have to capture any TCP packet that comes from a particular IP and with a destination port number 23. Since we ping from Host A, the filter put in includes the IP address 10.9.0.5 and the specified destination port number as 23. This is shown in the code below:Text

Description automatically generated

Output is shown as follows. As we can see, output differs from the first filter in that only TCP packets from port 23 are captured.



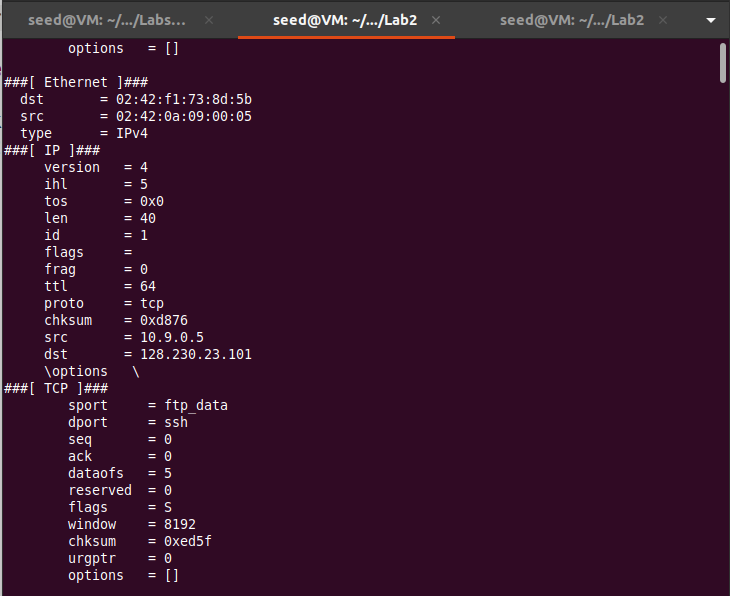
In the third filter, we have to capture packets coming from or going to a particular subnet. I chose to filter based on the destination subnet and picked 128.230.0.0/16 as this. The code is shown below:



In order to send packets and specify the destination subnet mask, I ran the following python code in Host A:

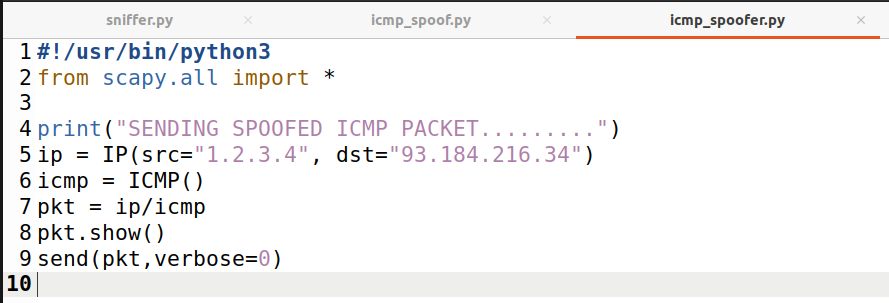


This yielded the following results. As we can see, the destination subnet matches the one we specified.



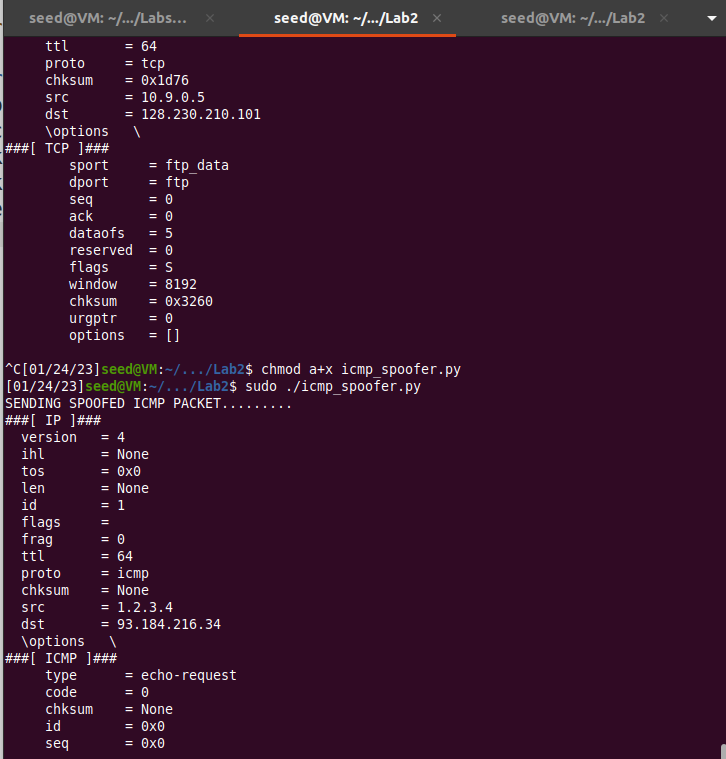
**Task 1.2**

In this task, we had to spoof ICMP echo request packets, and send them to another VM on the same network. The code to do this from an arbitrary IP to an arbitrary IP is shown below:



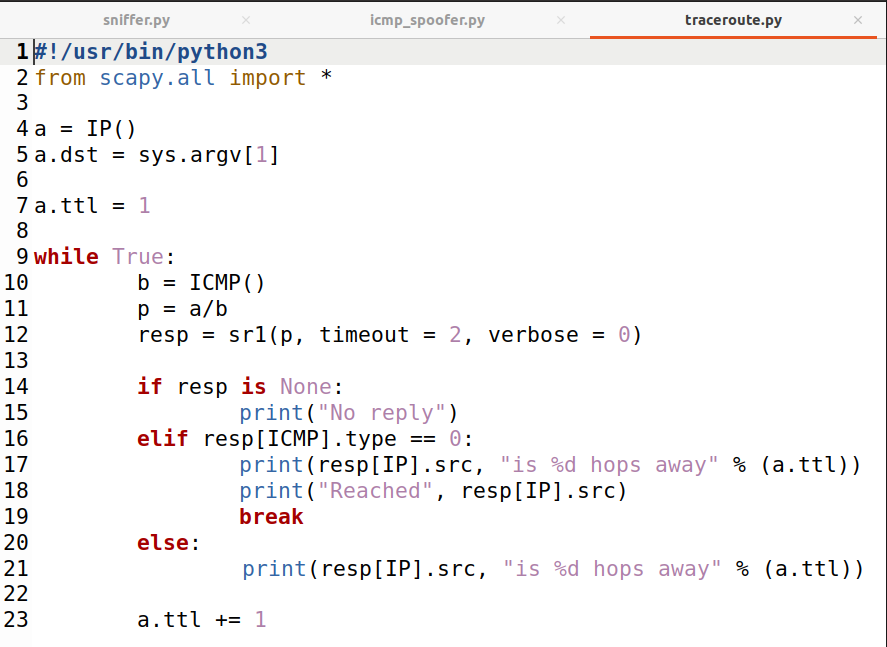
We also use Wireshark to observe whether our request will be accepted by the receiver. If it is accepted, an echo reply packet will be sent to the spoofed IP address.

When we ping Host B from Host A, the following result is observed:



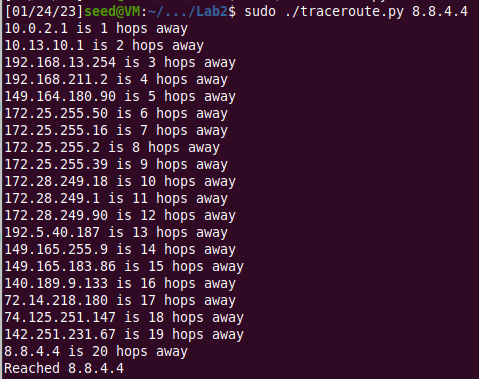
**Task 1.3**

In this task, we had to implement traceroute using Scapy. Traceroute is a tool that estimate the distance, in terms of number of routers or number of hops, between the sender’s VM and a selected destination. We send a packet of any type (in this case I used ping to send an ICMP packet) to the destination i.e., 8.8.4.4. here, while initially setting the Time-To-Live (TTL) field to 1. The first router sends an ICMP error message saying that the TTL has exceeded. From this, we get the address of the first router. In each following round, we increment the value and send another packet, and get the address of the next router, until we reach our destination. The following code shows how this can be done (a while loop is implemented):



(The in-class activity on traceroute really helped here and I modified code based on what was shown in class)

The below image shows the output obtained when the traceroute program is in with the IP 8.8.4.4:



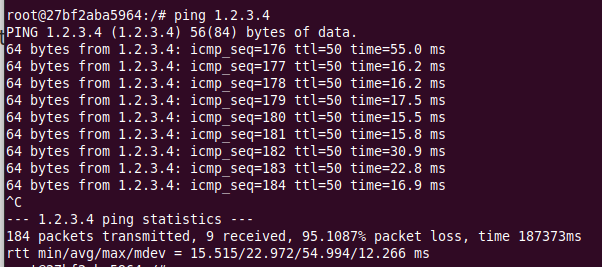
**Task 1.4**

For this task, we were supposed to implement sniffing and spoofing by using a program that combined the two. The below code shows how this is implemented. There is a function that spoofs any ICMP packet that is sniffed using the sniffer. Since we need to keep the data present in the original packet as it is in the spoofed packet, this is loaded as is. The id and sequence of the ICMP packet are used along with a packet type of 0 indicating a reply.

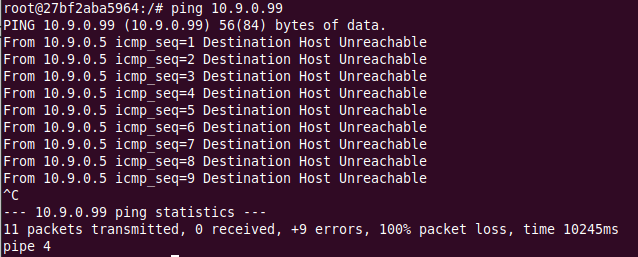
Graphical user interface, text, application

Description automatically generated

In the first ping, we send a request to 1.2.3.4, which is a non-existing host on the Internet. As we can see below, spoofed packets, indicated by ttl = 50, are sent in reply.



On the other hand, when we try to ping 10.9.0.99, which is a non-existing host on the LAN, it says that the destination host is unreachable. So why does a ping to 1.2.3.4 work but the one to 10.9.0.99 fail?



We run the command ‘ip route get’ on both destination IPs to find the router for that destination and see if there is a route that exists to allow packets to get sent. In the case of 1.2.3.4, we can see that there is a route through 10.9.0.1, but there is no route in the case of 10.9.0.99. This is because of the ARP protocol not being able to find a route when trying to send a ping. In the case of 1.2.3.4, since it is on the internet, ARP is able to find a router that routes the packet sent and able to convert the IP to a MAC address. But in case of 10.9.0.99, ARP is unable to find any routing to this destination.

Text

Description automatically generated

In the third ping, we send a request to 8.8.4.4, which is an existing host on the Internet. There are no problems here and we are able to get back the spoofed replies, shown below with ttl = 50.

