**SEED Labs – Packets Sniffing and Spoofing.**

בס"ד

**Ex6 – Networking.**

**Submitted by:**

**Alon Barak – 213487598**

**Idan Philosof – 324076066**

**Accompanied by Dr. Ran Dubin**

**SEED LABS – PACKET SNIFFING AND SPOOFING**

1. Overview:

Packet sniffing and spoofing are important concepts in the network security world.

In this Lab Alon and Idan did their best in order to implement and understand these two concepts better.

In this Lab we chose to use Containers (Docker) to best illustrate multiple users during the Lab.

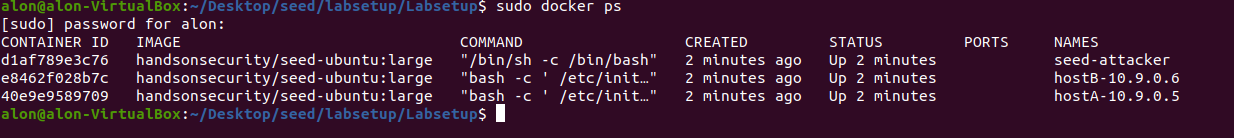
Just to make things clean, the IP addresses below are the ones we used during the Lab with different Containers:

The Attacker-host : 10.9.0.1

A-host : 10.9.00.5

B-host : 10.9.0.6

Virtual Machine-Ip: 10.0.2.15



The Lab programs and tests were performed on Ubuntu 20.04 Virtual Machine environment (Linux).

We added a GitHub Link to our Lab repository, feel free to check it and to impress.

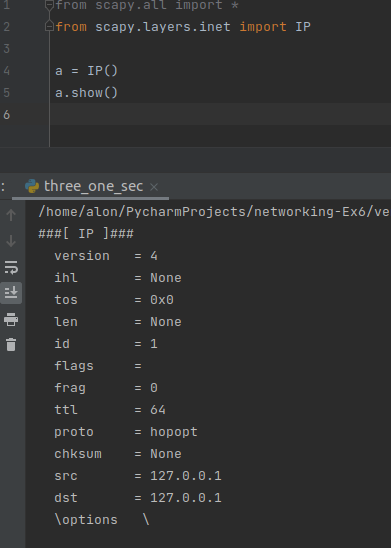
<https://github.com/AlonBarak-dev/SEED-LABS.git>

**Lab task set 1: Using Scapy to Sniff and Spoof Packets**

The package Scapy allows us the programmers to perform complicated programs such as Sniffer and Spoofing in a convenient and simple way in Python.

In the section below we will present our performance with the package according to the demands of the Lab.

1:

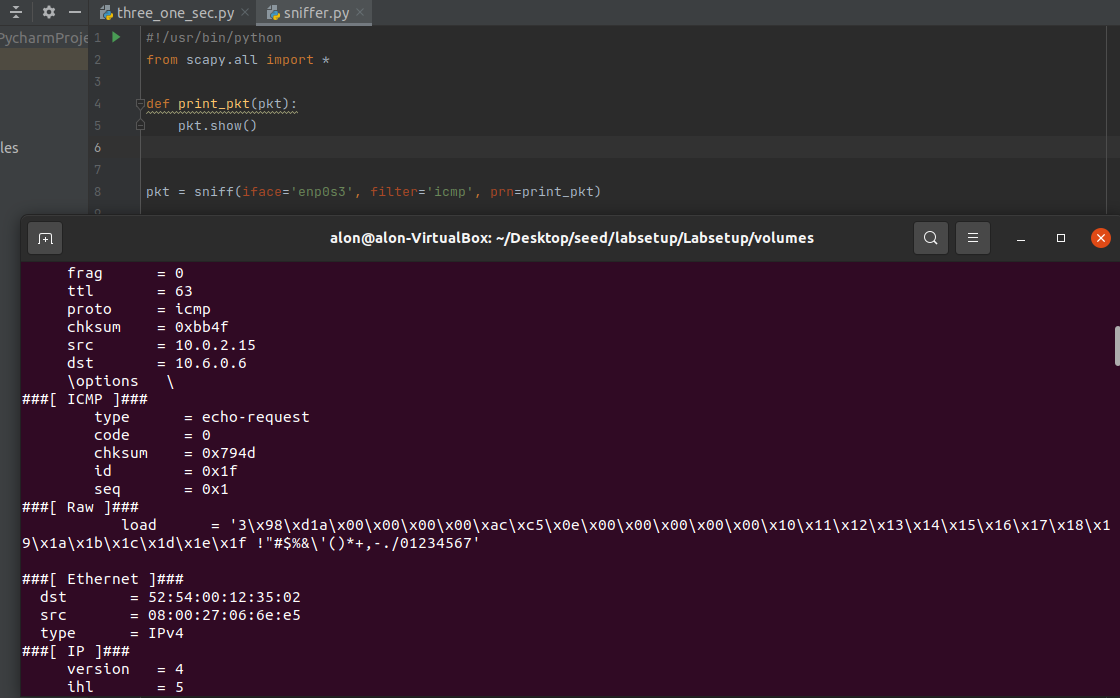


In the code above we present an example of a creation of an IP packet using Scapy package and its visual layer with the help of the method show() from Scapy.

We see an IP packet with its fields and attributes, such as: TTL, Source IP, Destination IP and more.

1.1A:

In this program we used Scapy to create a monitor who can Sniff and show the packets which transforming in its interface level.



The sniff() function receive interfaces, filters and a method as an input and create a Sniffer base on the user demands.

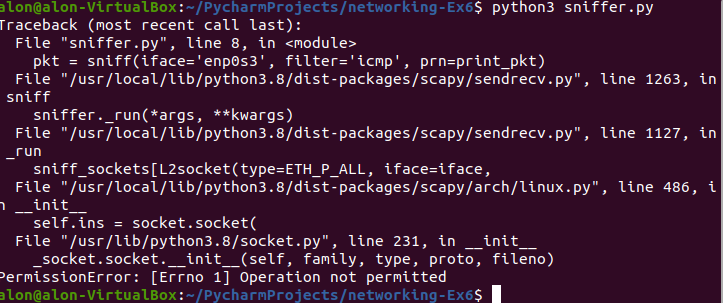
In the example above, we filtered the Sniffer to see only ICMP packets from the 'enp0s3' interface of our VM.

The output of this program (as shown in the added picture) is all the ICMP packets who transformed through the 'enp0s3' interface.

Each packet is presented with its major fields, such as: id, src, dest, type and more.

In order to run the program above from a VM environment, the User must run the program using the root privilege. The reason is simple, in order to gain access to transformed packets in the network we must have an admin privilege to do so, thus, the program will fail without a root privilege.

The error which occurs to us when we tried to access the program without a 'sudo' privilege was:

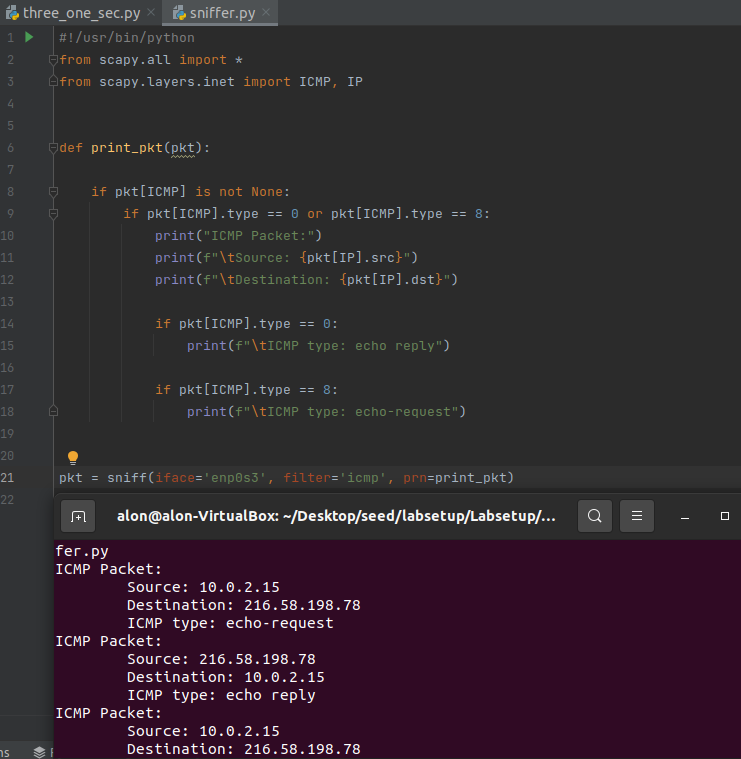


1.1B:

Usually, when we sniff packets, we are only interested in certain types of packets. We can do that by setting filters in sniffing.

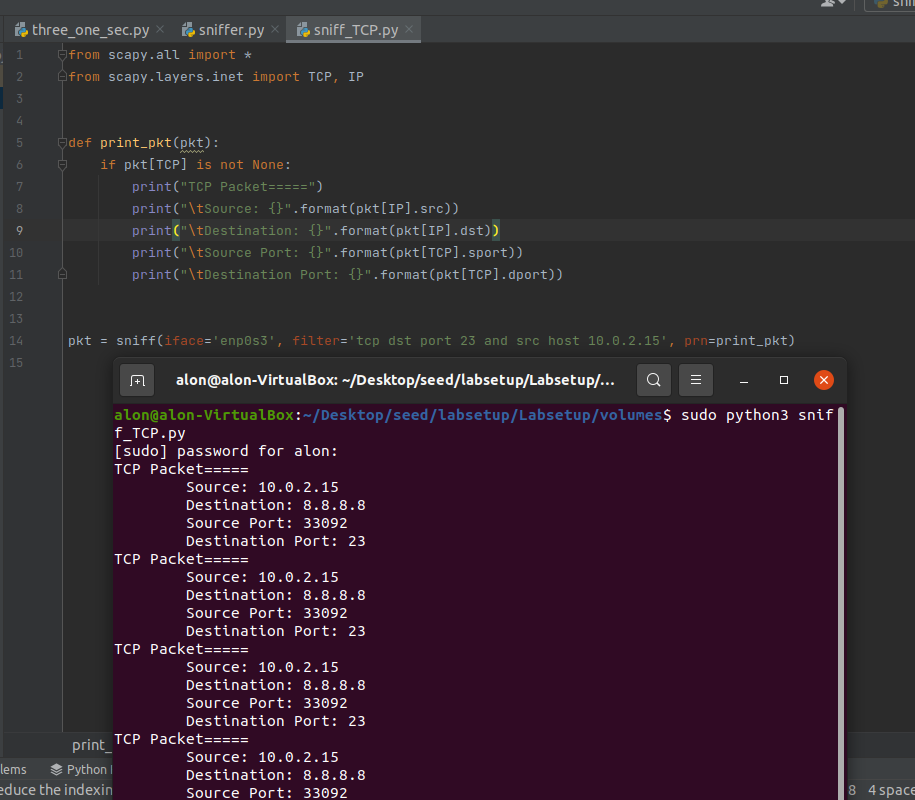
In the examples below we will present various ways of filtering in Scapy.

Capture only the ICMP packets:



In the code above we checked each packet if its protocol type is as ICMP protocol, the ones who had the ICMP protocol were the ones who got printed in the terminal as the output of the program.

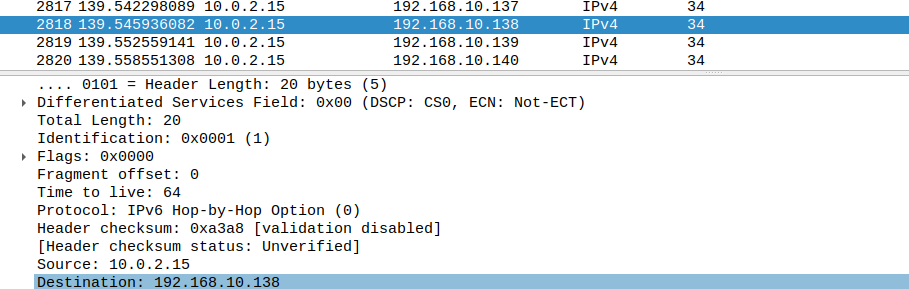
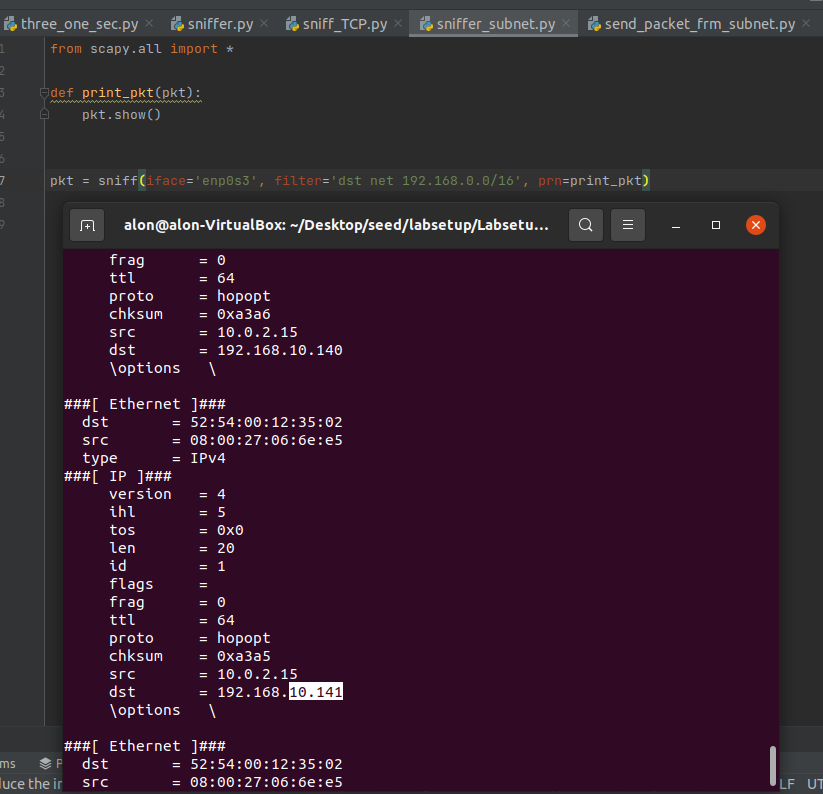
Capture any TCP packet that comes from a particular IP and with a destination port number 23:



In the code above we used the 'filter' functionality in Sniff() from Scapy. The method allow us to write down in words the kind of packets we are looking for.

For our target we typed in "tcp dst port 23 and src host 10.0.2.15", which will generate for us only the packets who fits into these rules we set.

Capture packets comes from or go to a particular subnet, pick any subnet you wish except the subnet your VM is attached to:



In this example, we were asked to show packets from the same subnet other than our VM subnet. In order to do so we wrote in the filter option of the Sniff() method the following String:

"dst net 192.168.0.0/16"

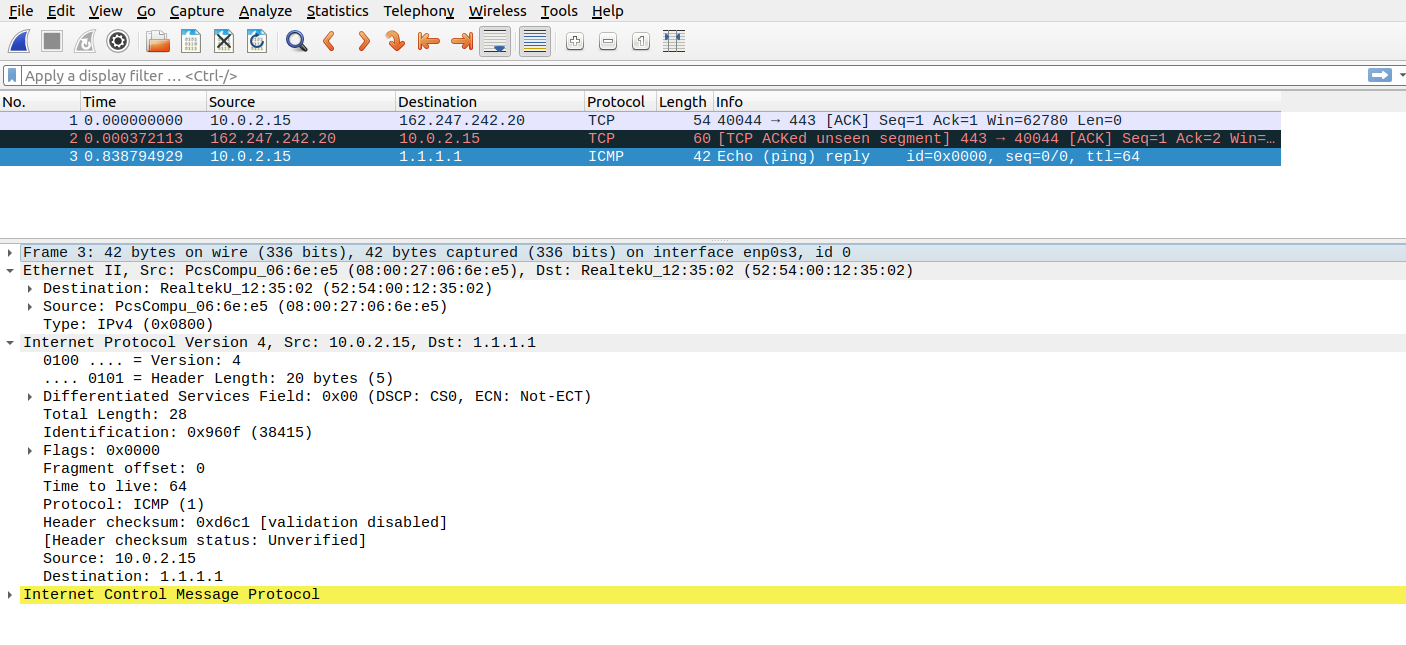
The chosen subnet were 192.168.0.0/16 and the output were packets who go to/from this subnet.

**Task 1.2: Spoofing ICMP Packets**

In this section we will discuss Packet Spoofing using the same Scapy package from previous tasks.

Spoofing is an important network tool. Its allows the programmer to set the Packet's attributes by himself and by that to create a Spoofed Packet without someone noticed. תמונה שמכילה טקסט, אלקטרוניקה

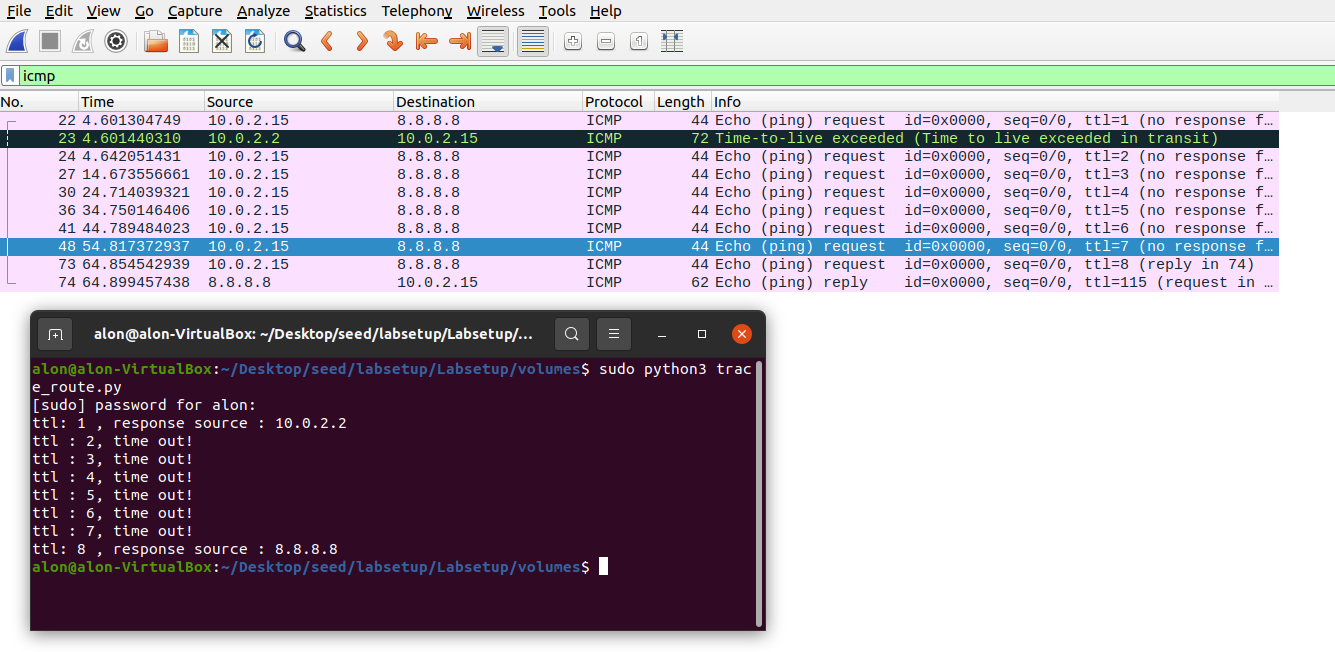
התיאור נוצר באופן אוטומטי



In the program mentioned above, we tried to modify a packet (Spoofing) and its fields. We set the Source IP as a fake IP address (1.1.1.1) and thus, Spoofed the ICMP packet.

**Task 1.3: Trace Route**

The main goal of this section is to estimate the distance (number of routers), between our VM and a selected destination (Google server, in our case).

 In the program mentioned above, we looped and incremented the TTL field of the packet we sent each time we reached a time Out and sent once again the packet to the same destination but with the new TTL field only.

Once the Packet reached fully to its destination we knew the estimated distance between our VM to Google.com server. In our case, we found the distance to be 8 routers.

And as we can see, both in wireShark and in our program, the distance is estimated around 8 routers.

Of course this result is not constant and might change according to the network conditions.

**Task 1.4: Sniffing and-then Spoofing:**

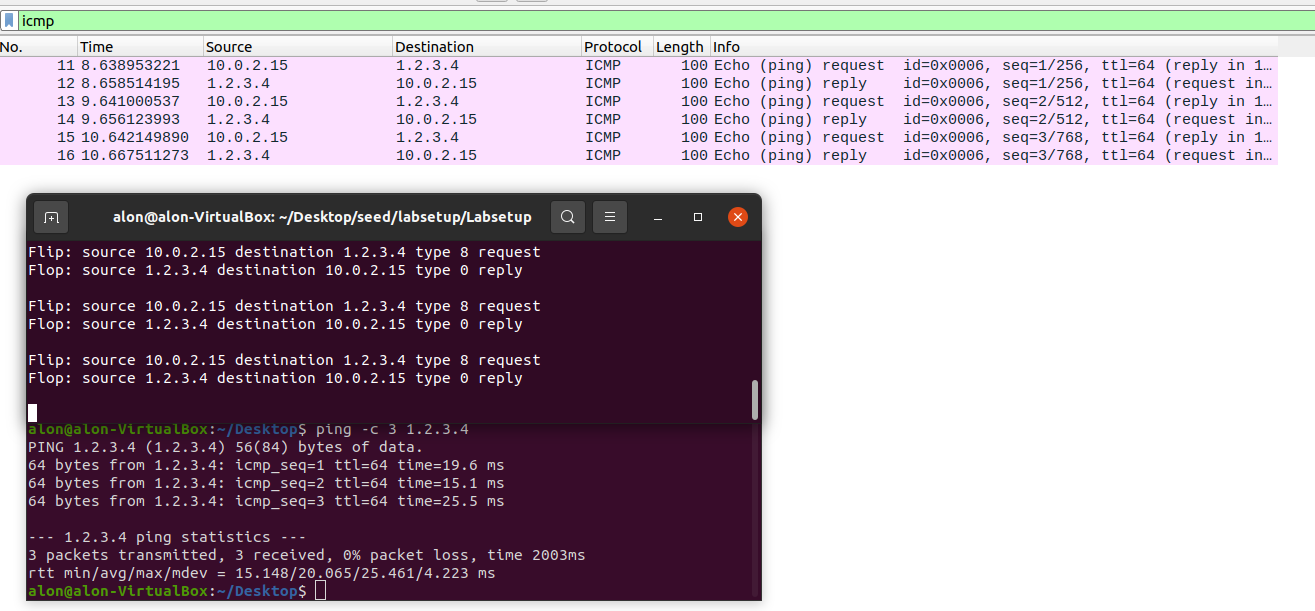
The purpose of this section is to combine both Sniffing and Spoofing for the first time, while using Scapy package in Python.

**What is an ARP protocol?**

A communication protocol used for discovering the link layer address, such as a MAC address, associated with a given internet layer address, typically an IPv4 address. These packets are packets that the IP system broadcasts to all devices on a network (or VLAN), to determine the owner of a specific IP address.

The main goal of our code is to return a reply packet back to the source, even if the request isn't available at the moment (faking). We will test our code in three different scenarios:

Once for a non-existing host on the Internet:



In this scenario, our VM sent a ping to a non-existing host on the Internet. The ARP protocol asks for the fake IP "1.2.3.4" and then the Attacker (another VM) return an answer to it so an ICMP reply packet is returned to the source address, thus we gain 0% packet loss instead of 100% loss in an normal situation. We can see the communication (except for the ARPs which we cut for some reason..) between the source VM and our Attacker.

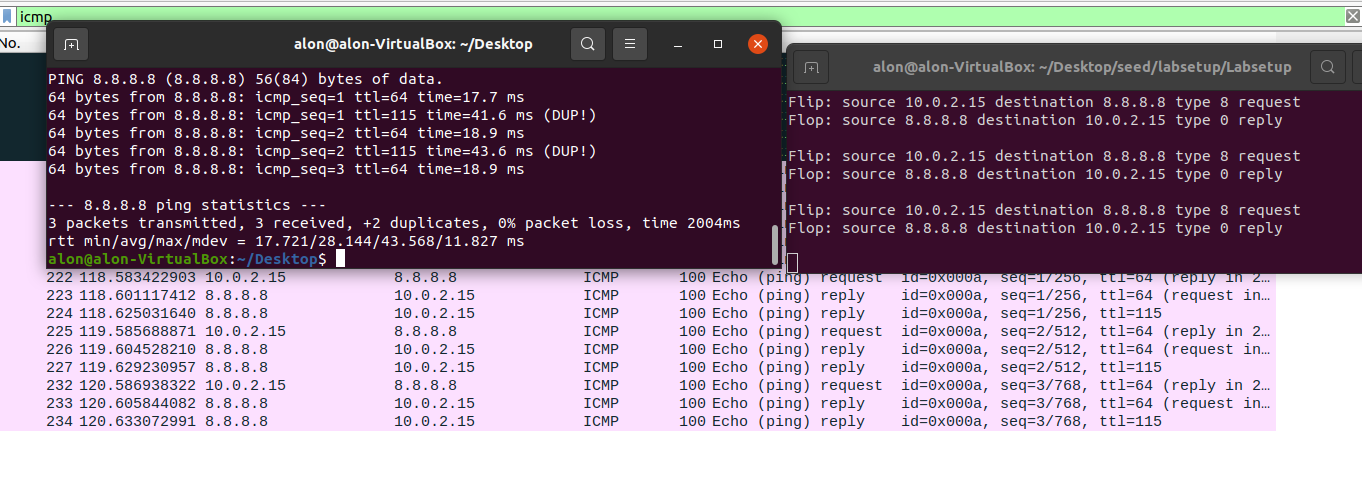
Secondly, for a non-existing host on the LAN:

\*\*\*\*

For some reason we couldn’t get response from the program.

\*\*\*\*

Thirdly, for an existing host on the Internet:



In this scenario we needs to Spoof an existing host on the Internet (Google.com / 8.8.8.8).

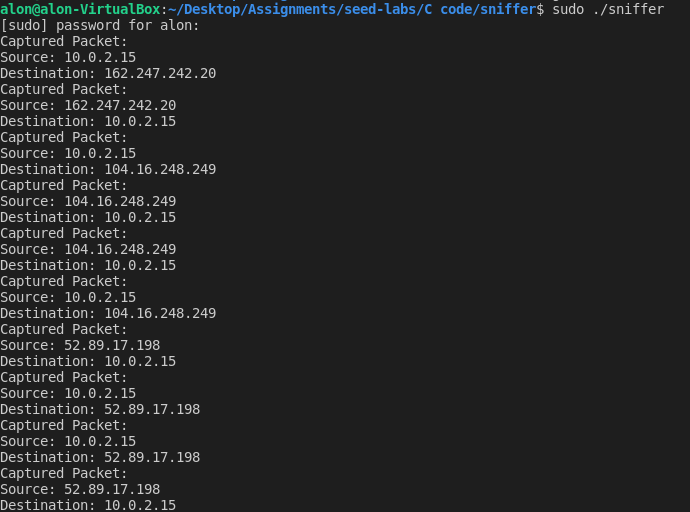
As captured above, we got Duplicates while pinging for the IP address. The reason is simple, since it is a real destination it is also response to the source together with our Program. Thus, we receive Duplicates from two destinations which looks the same for the source host.

**2 Task: Writing Programs to Sniff and Spoof packets**

In this section in the Lab we will change our program language to C. in this part of the Lab we compiled and ran the code on the Virtual Machine of Ubuntu 20.04 and used the Docker to create Containers and run the programs through the Containers.

**2.1 Task: Writing Packet Sniffing Program**

In order to write a Sniffer in C we will use the pcap library.



In the added picture above we can see the output of the code program sniffer.c (located in the C files).

In that program we created a Sniffer program with the help of the pcap library for capturing traffic in the network. The program display the source and destination of each packet in the network (only for IPv4 packets).

**2.1A Task: Understanding How a Sniffer Works**

Q&A section:

Q1: please use your own words to describe the sequence of the library calls that are essential for sniffer programs.

A1:

1. Open a live pcap session on NIC with the name of the network interface. We use pcap\_open\_live() method from the pcap library for that. This method allows us to capture the traffic in the interface's network and bind the socket as well.
2. For filtering the packets we use the pcap\_compile() and pcap\_setfilter() who responsible for compiling the filter String into an actual filter program.
3. We capture the packets using the method pcap\_loop().

Q2: why do you need the root privilege to run a sniffer program?

Where does the program fail if it is executed without the root privilege?

A2:

A root privilege is needed in order to set up the card in promiscuous mode and raw socket, without it we cannot see the entire network traffic in the interface.

Without the root privilege, the pcap\_open\_live() method would fail and cause an error in the entire program.

Q3: please turn on and turn off the promiscuous mode in your sniffer program. Can you demonstrate the difference when this mode is on and off?

Please describe hoe can you demonstrate this.

A3:

In order to turn off and on the promiscuous mode we need to changed the third value in the pcap\_open\_live() to 0, rest will result in ON.

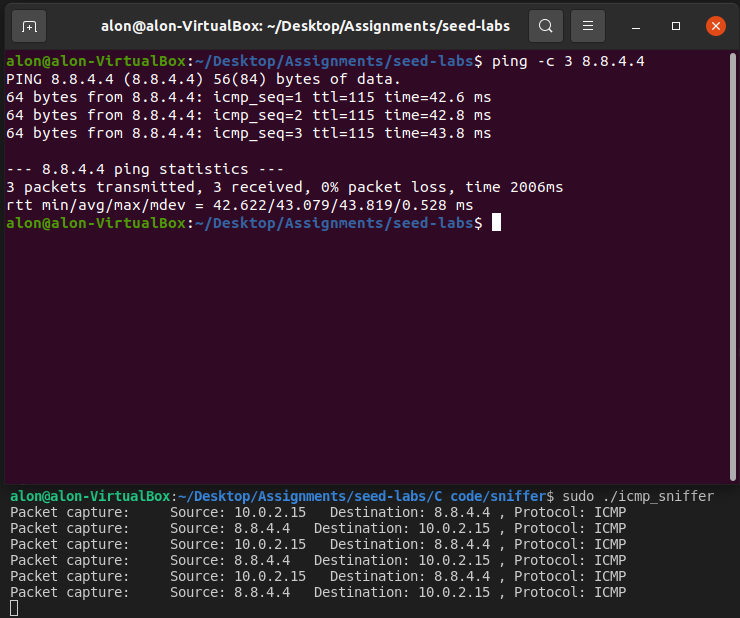
Without the promiscuous mode, a sniffer can only capture packets transformed directly to/ from/ routed through the host will be captured by the sniffer program.

With the promiscuous mode we can see the entire network traffic in the sniffer.

**21.B Task: Writing Filters**

Capture the ICMP packets between two specific hosts:

in this section we need to build a Sniffer that capture ICMP packets between two specific hosts.



In the added picture above we can see a communication between two hosts (10.0.2.15(in black) & 10.9.0.5(in purple)) with ICMP Packets as wanted in the Lab.

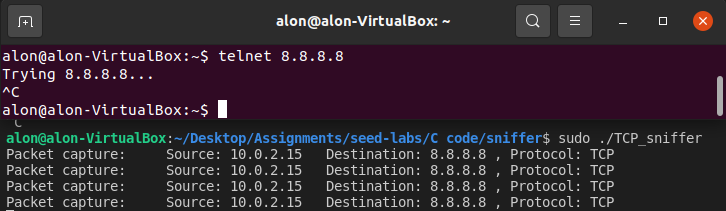
In that code, we used almost the same code as before but this time the filter packet is "ip proto icmp"

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

This time we needed to capture only TCP packets whose port range is between 10 to 100.

Since these are very low and busy ports we decided to use the telnet option.

telnet connect one host to another through port 23 (which is between 10 to 100), in addition, telnet uses the TCP protocol.



In the added example we can see the telnet functionality on the IP address of Google.com (8.8.8.8) and in the lower part of the picture we can see the sniffer sniffing only these kind of packets.

**2.1C Task: Sniffing Passwords**

In this section we wrote code but couldn’t check if it is true or not since we couldn’t use the telnet command in this part of the Lab for some reason.

We still would love to hear opinions about our program (added in the zip file).

**2.2 Task: Spoofing**

When a normal user sends out a packet, operating systems usually do not allow the user to set all the fields in the protocol headers (such as TCP, UDP, and IP headers).

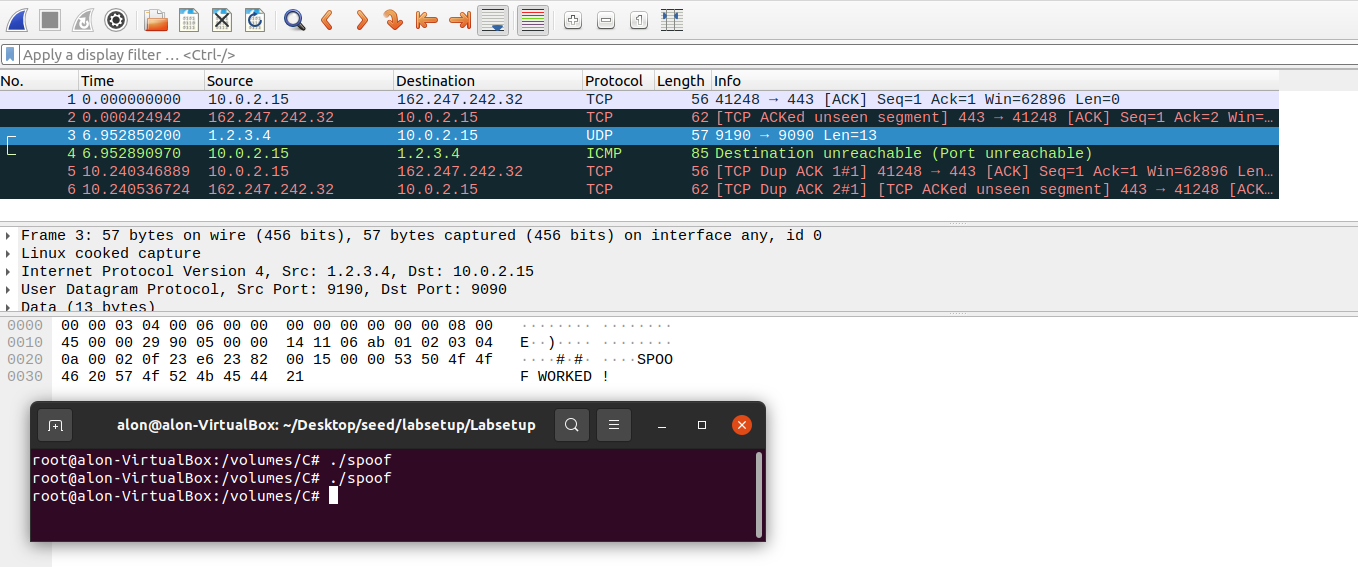
OSes will set most of the fields, while only allowing users to set a few fields, such as the destination IP address, the destination port number, etc.

However, if users have the root privilege, they can set any arbitrary field in the packet headers. This is called packet spoofing, and it can be done through raw sockets.

In the next sections we will present to you our Sniffing, Spoof an ICMP Echo response, Sniff and then Spoof programs.

**2.2A Task: Write a Spoofing program**

In this section we will present to you our C code to Spoofing a packet. (my\_spoofing.c)



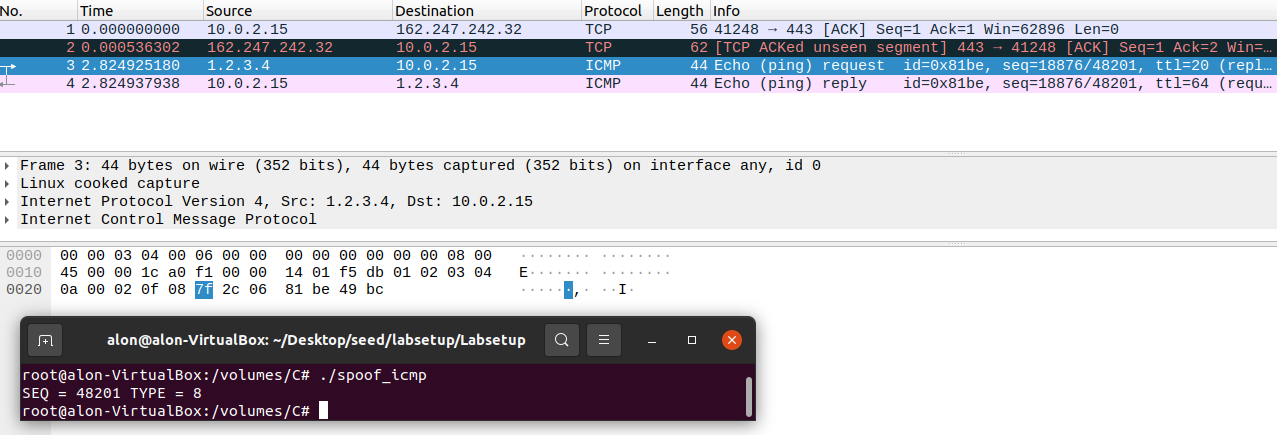
In the program mentioned above we can see that the source IP has changed to 1.2.3.4 as a result from a Packet Spoofing.

we took the example from the task info and added to it a header that contains a UDP protocol and sent it to destination 10.0.2.15 (from - 1.2.3.4) but faking it.

The program was created with a pcap library and modified the IP headers to use the source IP as 1.2.3.4 and destination as victim IP (10.0.2.15). When executed the packet was created with 1.2.3.4 and sent to the victim.

**2.2B Task: Spoof an ICMP Echo Request**

In this section we need to spoof an echo request on behalf of another machine. We used wireShark to find



In the added picture we can see that the source and the destination has been changed (since 1.2.3.4 wasn’t existing before the program).

Though the ICMP request originated from 10.9.0.5 , the attacker created the packet with a spoofed IP.

So, the remote server once received the ICMP packet, it responded back to the source IP that is present in the packet instead of sending to the attacker.

Thus, the attacker spoofed an ICMP Echo request.

Q&A section:

Q4: Can you set the IP packet length field to an arbitrary value regardless of how big the actual packet is?

A4: Yes, the IP packet length field can be any arbitrary value.

Q5: Using the raw socket programming, do you have to calculate the checksum for the IP header?

A5: No, When using the raw sockets, you can tell the kernel to calculate the checksum for the IP header. In IP header fields it’s actually the default option, ip\_check = 0 will let the kernel do it unless you change it to a different value but then you’ll have to use a checksum method, thus, we don’t need to

Q6. Why do you need the root privilege to run the programs that use raw sockets? Where does the program fail if executed without the root privilege?

A6 . Root privileges are necessary to run programs that implement raw sockets.

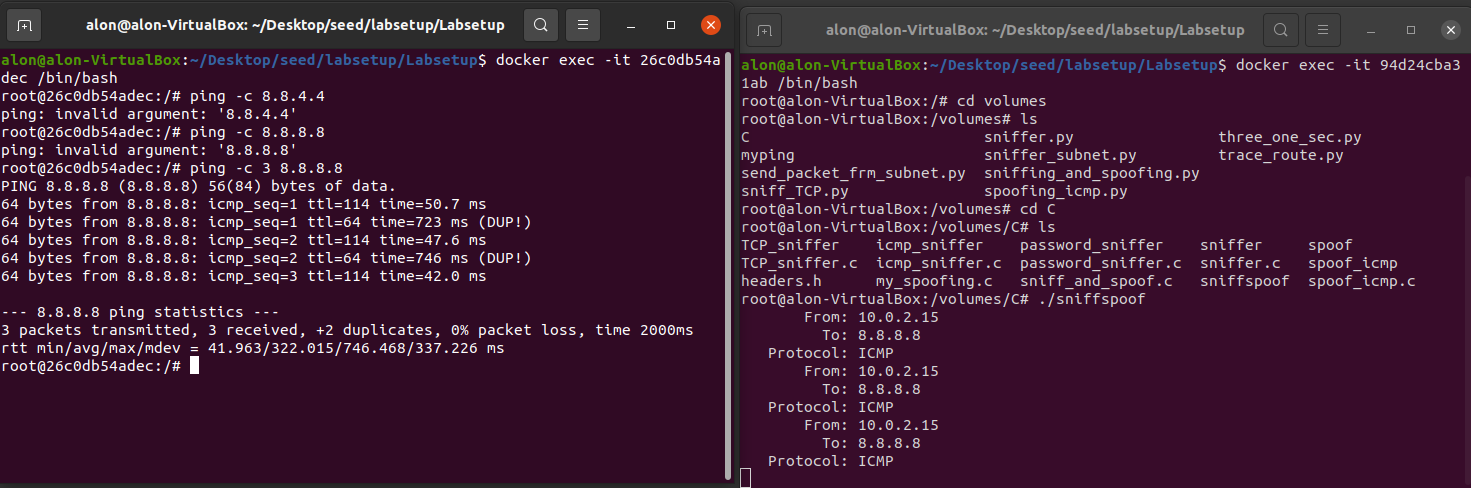
Root privileges users can set any field in the packet headers and to access the sockets and put the interface card in promiscuous mode.

If we run the program without the root privilege, it will fail at socket setup. (for the same reason we said moments ago).

**2.3 Task: Sniff and Spoof**

In this section we combine Sniffing and Spoofing in C for the first time. We used two Machines in this Lab part.

sniff-and-then-spoof program runs on the attacker machine, which monitors the LAN through packet sniffing. Whenever it sees an ICMP echo request, regardless of what the target IP address is, your program should immediately send out an echo reply using the packet spoofing technique.



תמונה שמכילה שולחן

התיאור נוצר באופן אוטומטי

In the added pictures above we can see the output of our program, basically we notice a conversation between our attacker and the Client. The attacker machine was in promiscuous mode and then when we executed our spoofing program, the NIC captured all the packets that reached and the program then processed in such a way, it modified the destination IP as source IP and source IP as destination IP.

Once the packet is created it sends the packet out and the victim has received it. Thus, we spoofed the ICMP echo request.