

COMP-8677

Networking and Data Security

LAB 4 Report

Karan Vishavjit

Student Id: 110099867

1. **Packet Construction with Scapy**

In this exercise, you will practice to construct several packets using scapy. To start, run

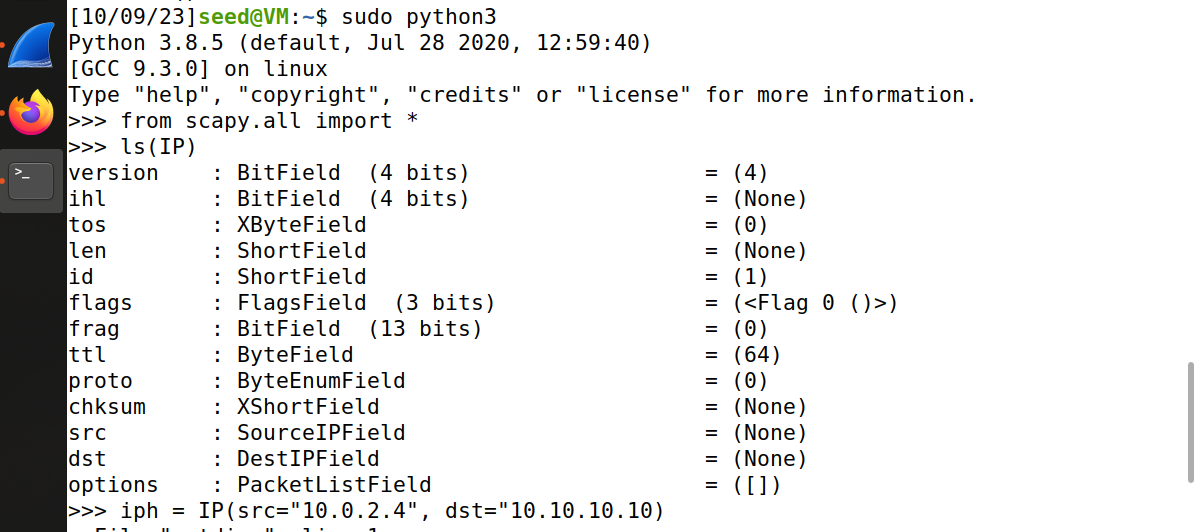
$sudo python3 (if you work on VM root, sudo is not needed) and then import scapy package:

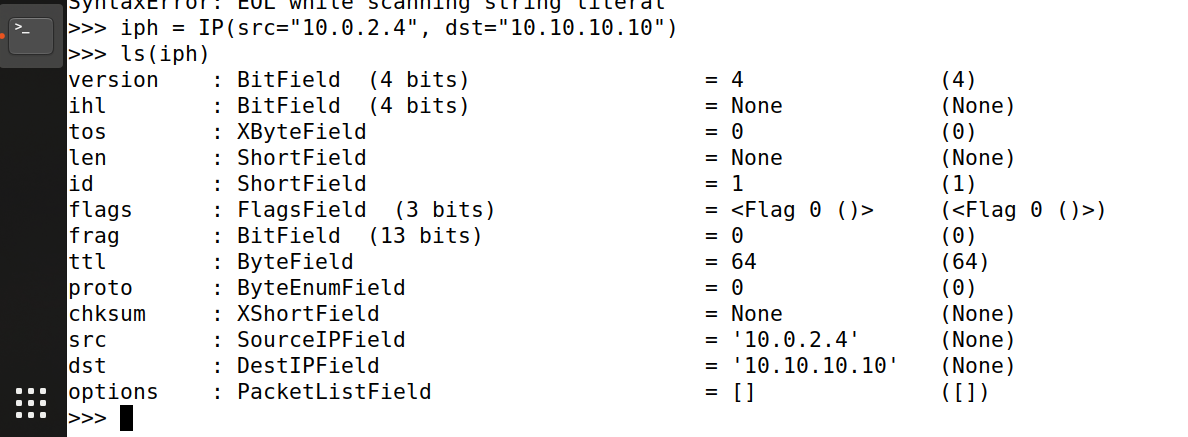
from scapy.all import \*

Then, you are ready to practice constructing various packets. In each question, show your

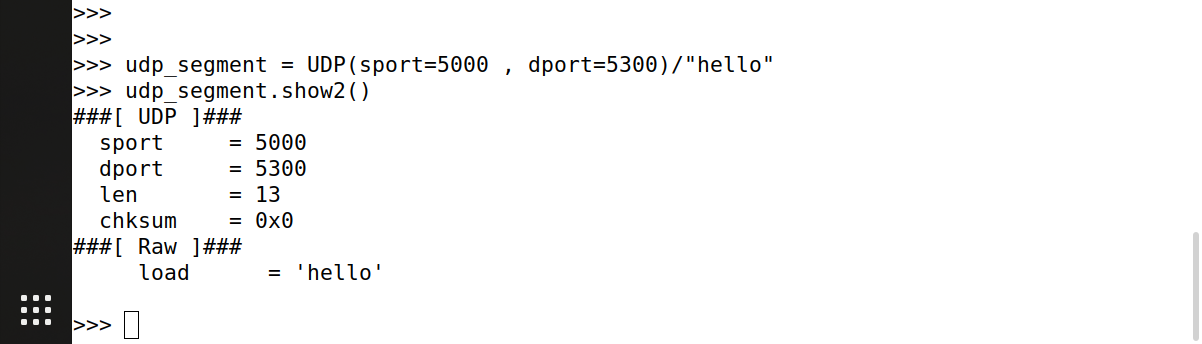
screen shots as the evidence of your work.

1. **IP() is the function to construct a default IP header. You can use ls(IP()) to view the content. The first column is the field of IP header and the third column is the example format for the value of each field. You can assign the value to create the IP header you want. Please construct an ip header iph with source 10.0.2.4 and destination 10.10.10.10. Use ls to show packet header.**

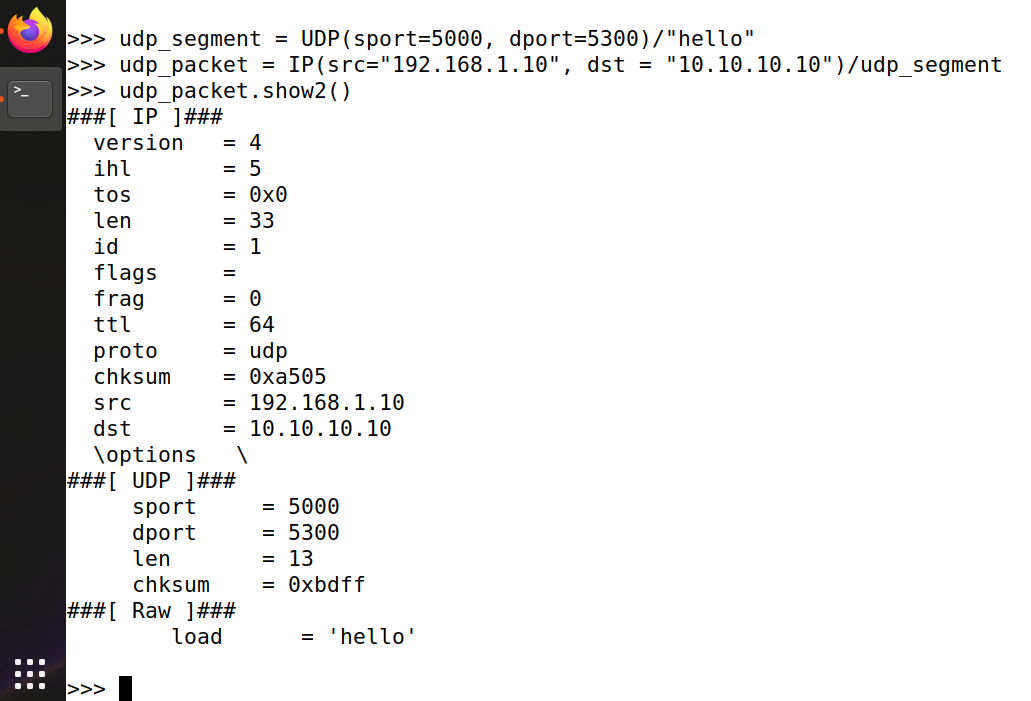
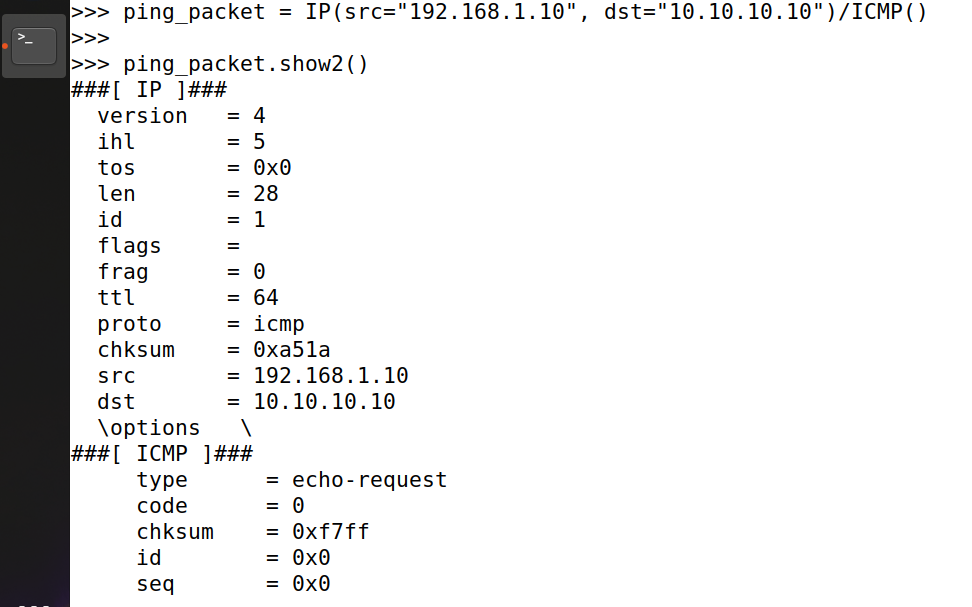




1. **Create a UDP segment with source port number 5000 and destination port number 5300 and data=”hello”. Use show2 to show your result.**



1. **You can create ping packet by stacking IP header over ICMP(). Create a ping packet with your VM as source IP and 10.10.10.10 as your destination IP. Create an ip packet with the same source and destination IP (as in the ping packet) but with UDP segment in item 2 as its data field. Use show2() function to show the packet content.**



1. **For a packet pkt, pkt[IP] is IP datagram and pkt[UDP] is the UDP segment of pkt. For ip datagram in item 3, use show2 to show the UDP segment.**



1. **Sniffing Packets**

Wireshark is the most popular sniffing tool, and it is easy to use. We will use it throughout the entire lab. The objective of the current task is to learn how to use Scapy to do packet sniffing in Python programs.

-----------------------------------------------------------

#!/usr/bin/python

from scapy.all import \*

def print\_pkt(pkt):

pkt.show2()

pkt = sniff(filter=’icmp’,prn=print\_pkt, iface=”br-xxx”) # br-xxx is the interface on VM you want to sniff

------------------------------------------------------

**Task A.**

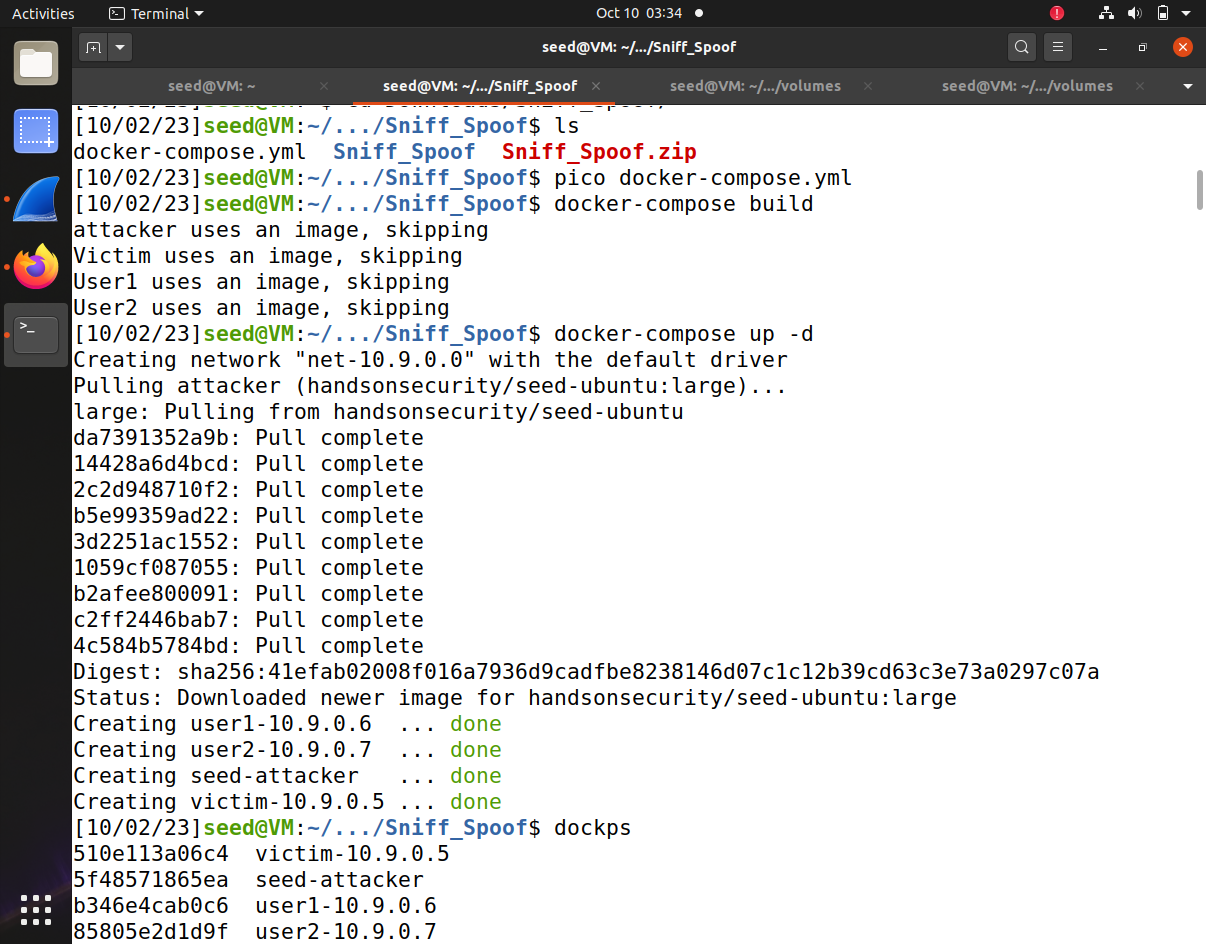
* **The above program sniffs packets. For each captured packet, the callback function print pkt() will be invoked; this function will print out some of the information about the packet.**
* **Run the program with the root privilege and demonstrate that you can indeed capture packets.**
* **After that, run the program again, but without using the root privilege; describe and explain your observations.**

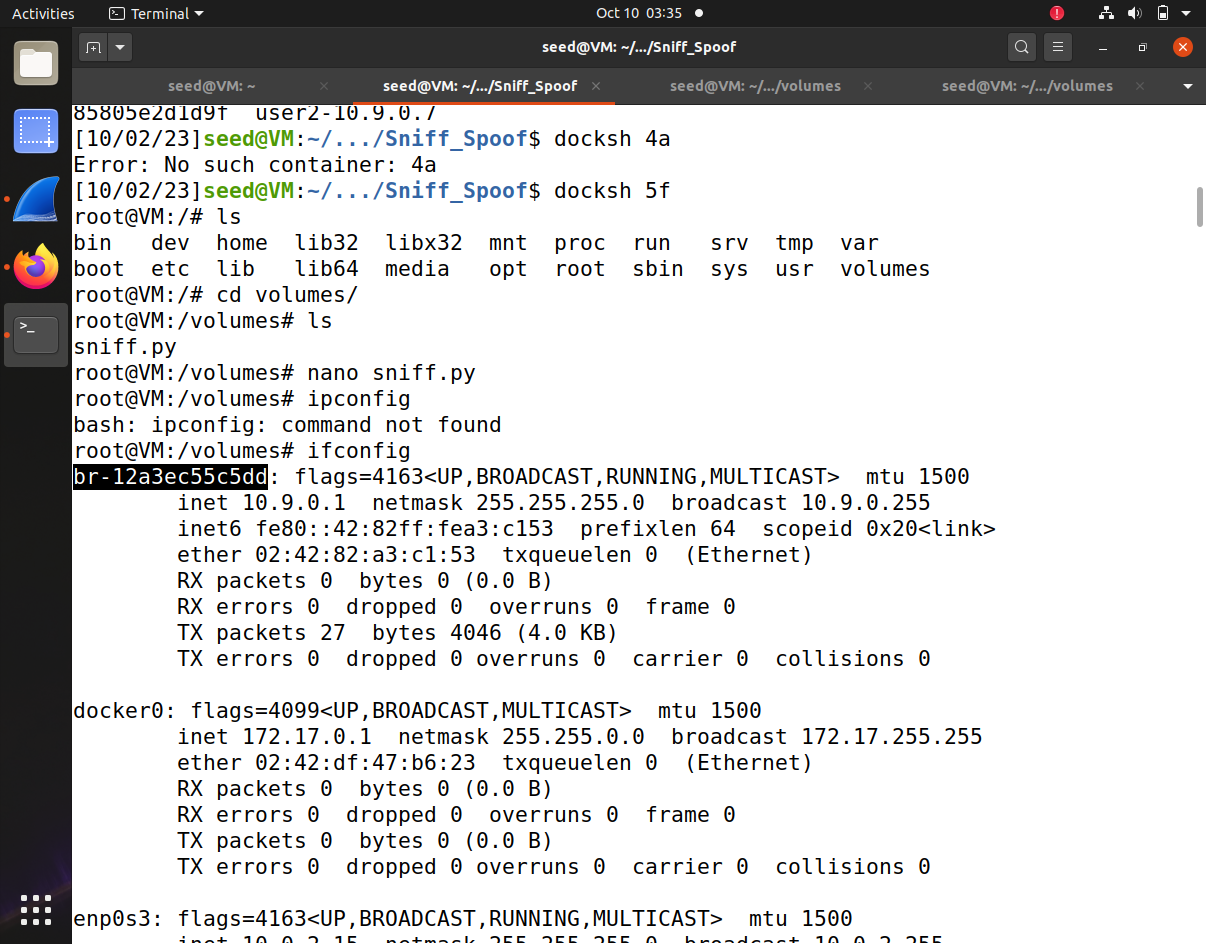
// Run the program with the root privilege

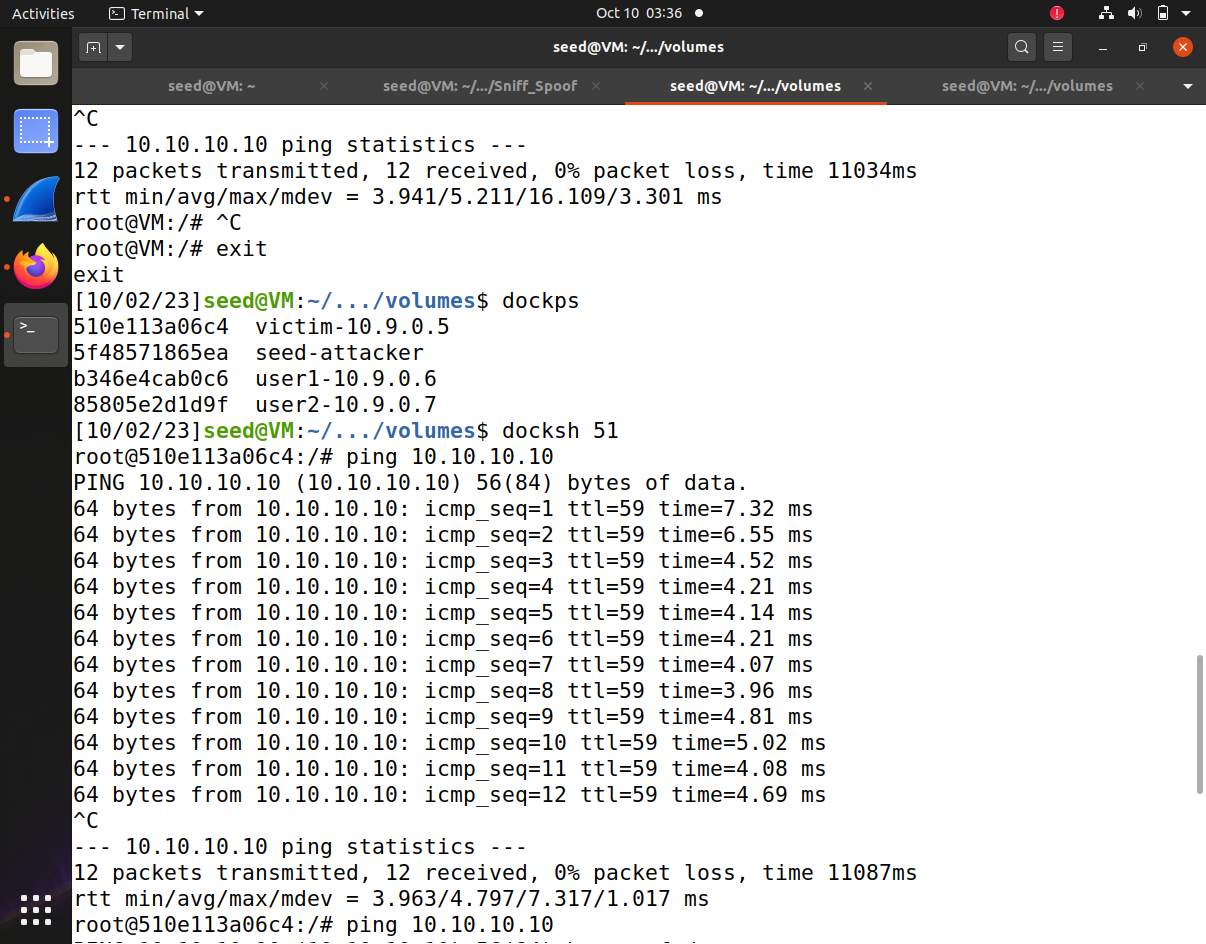
**$ sudo python sniffer.py**

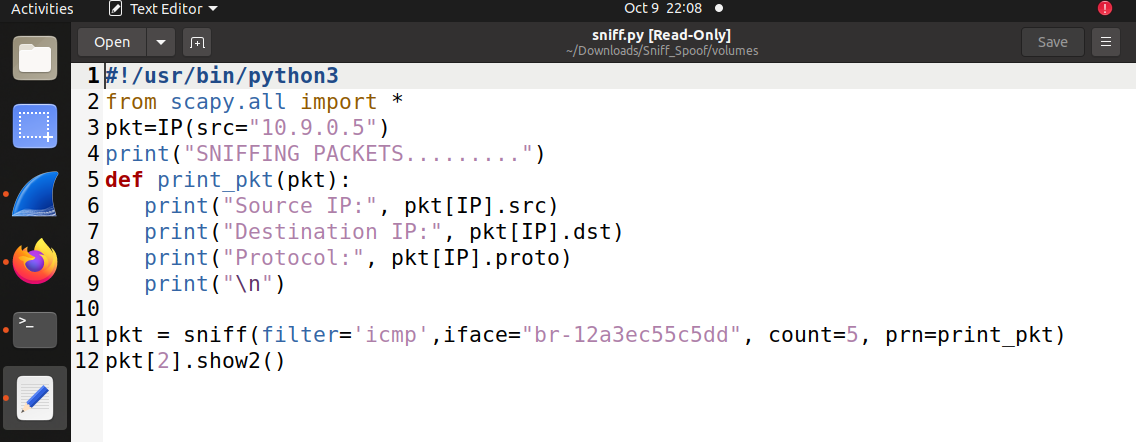
// Run the program without the root privilege

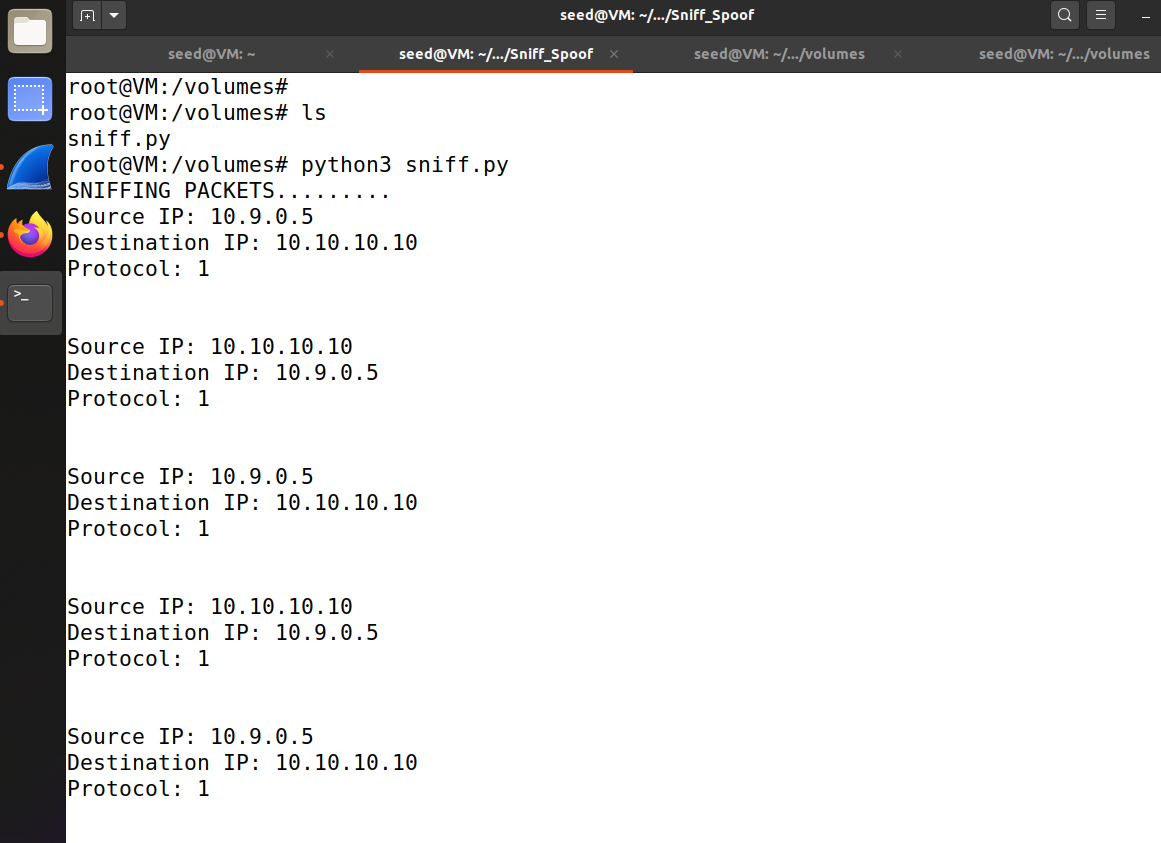
**$ python sniffer.py**

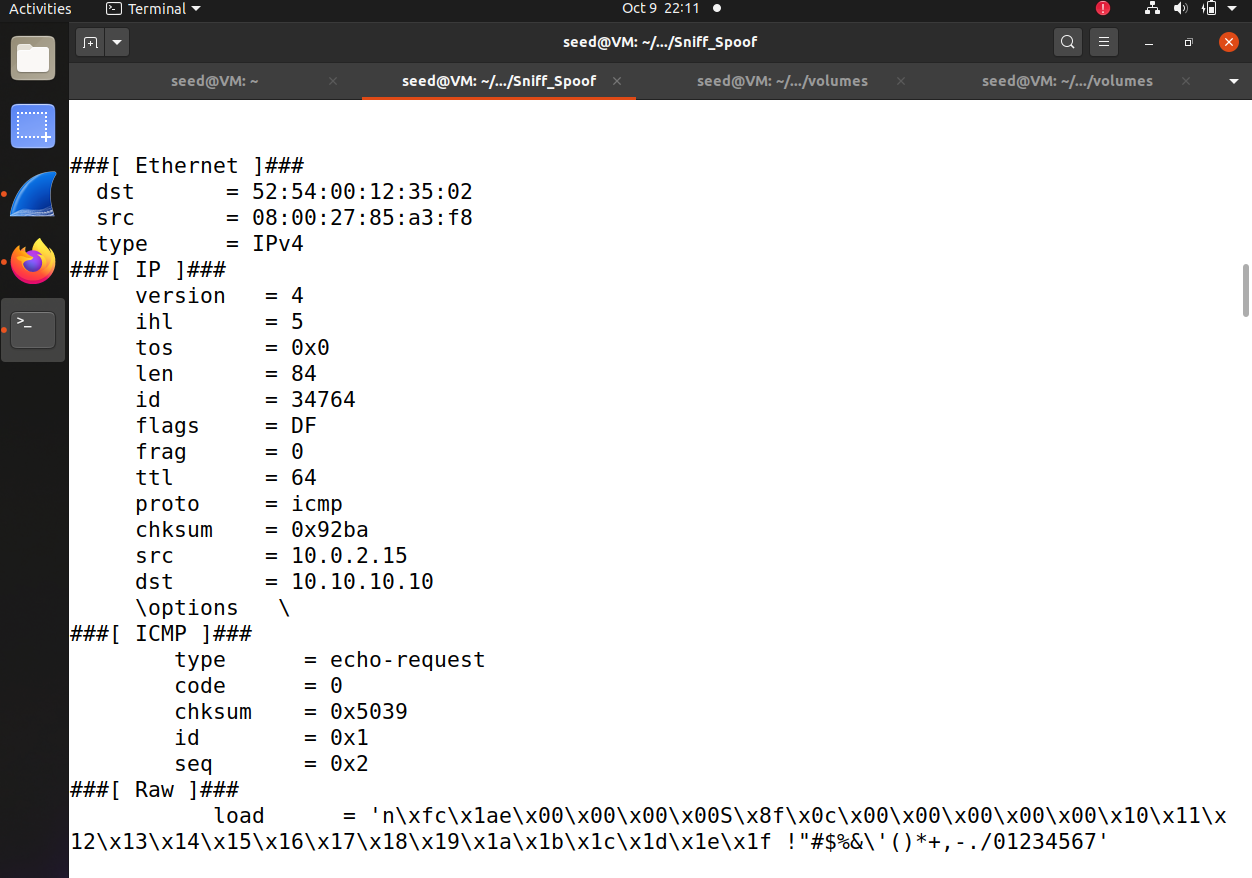












**Task B.**

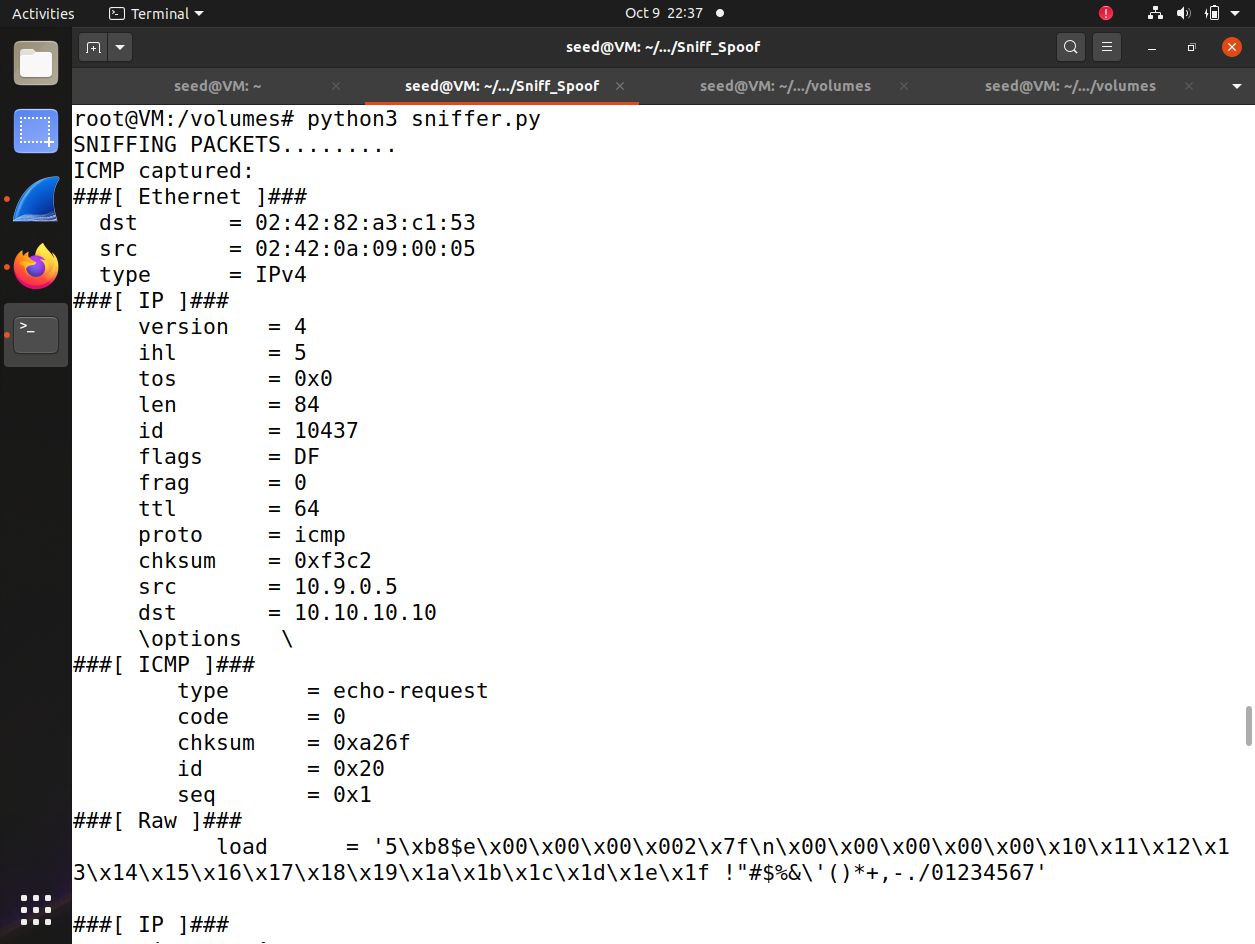
**In this task, you need to modify the program to simultaneously achieve two goals:**

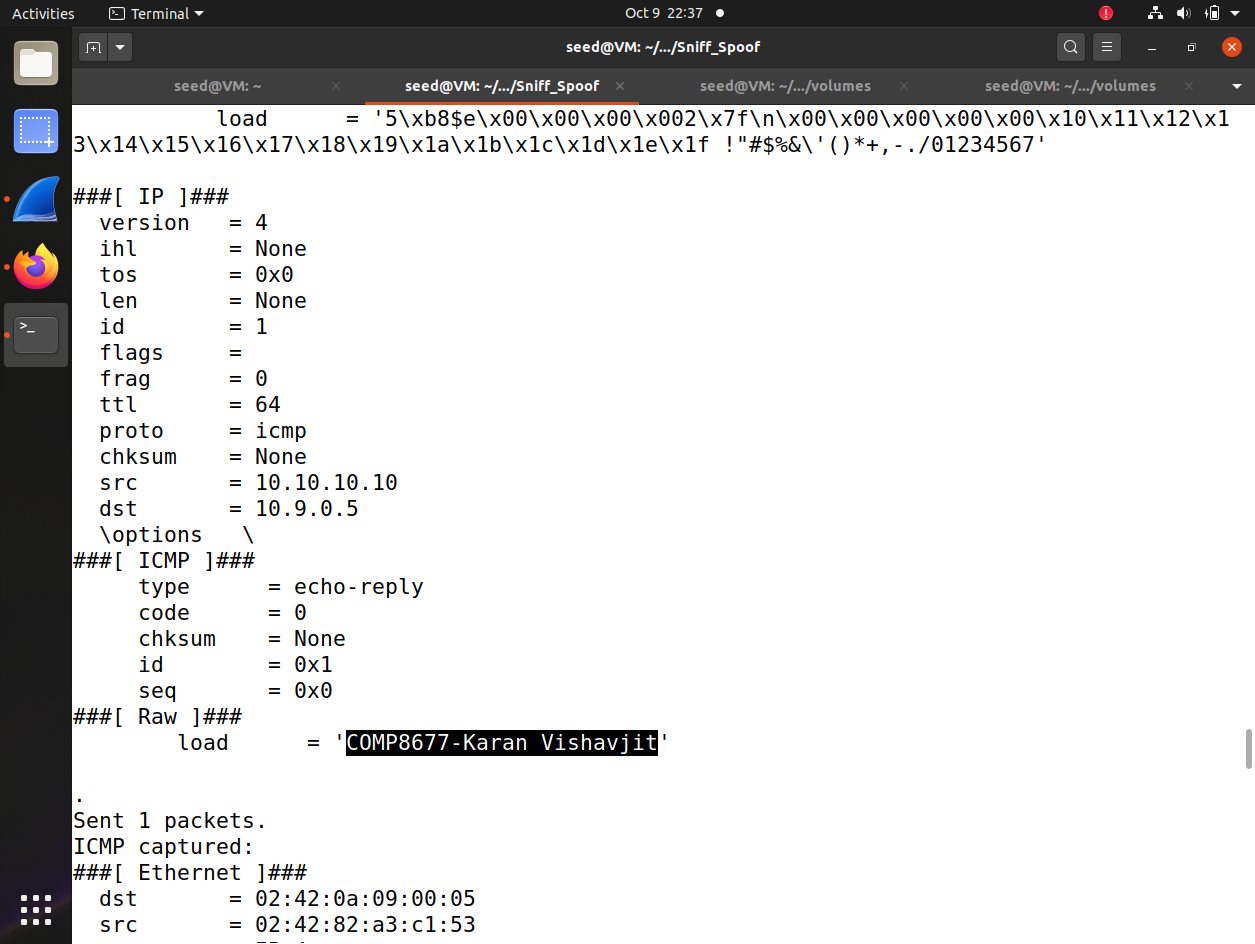
1. **When we sniff packets, we are only interested certain types of packets. Your program only sniffs the ICMP packet with source IP address 10.10.10.10.**
2. **For each captured ICMP packet, reverse the source and destination IP address and**

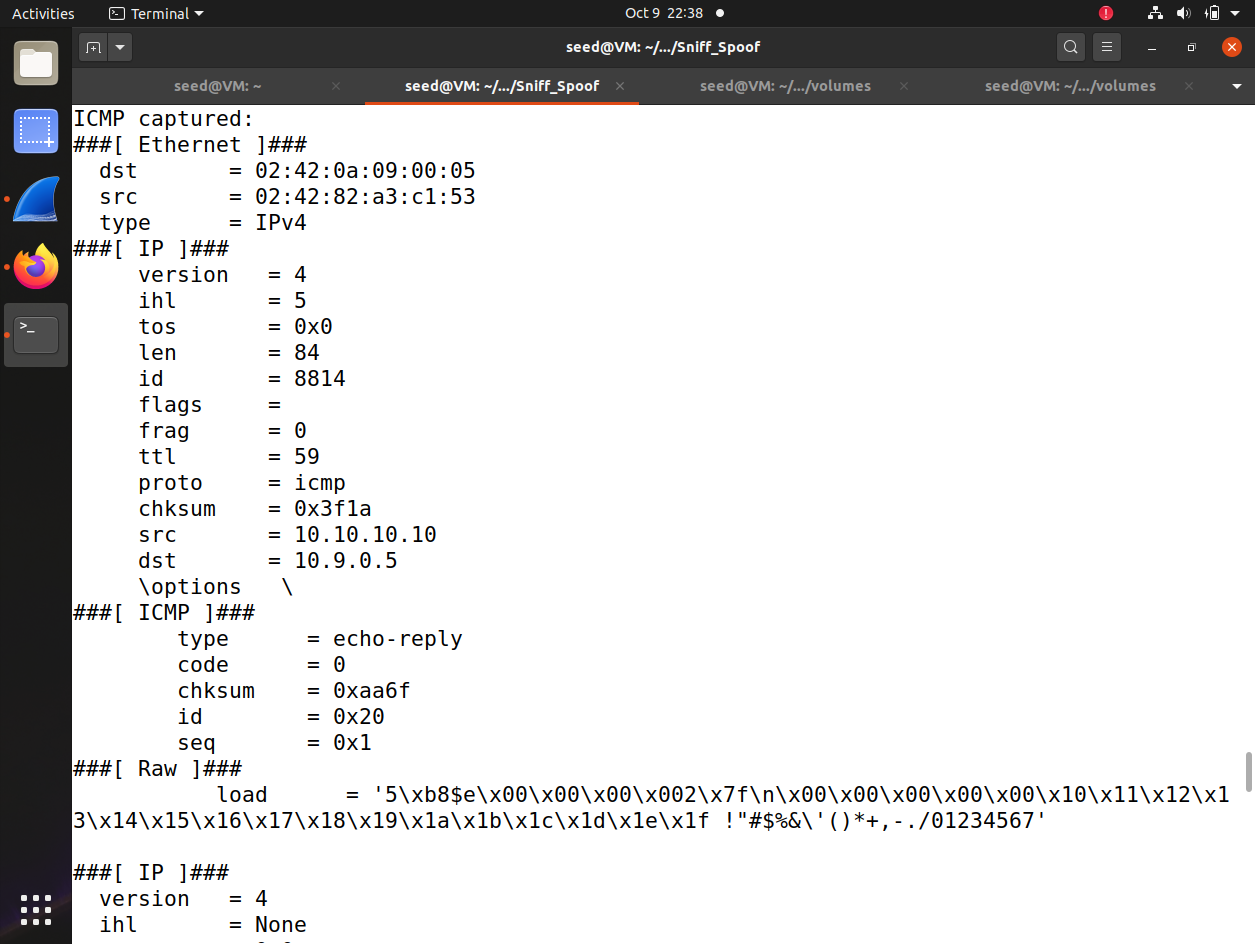
**modify the ICMP data field as “COMP8677-yourname”. Finally, send the modified packet.**

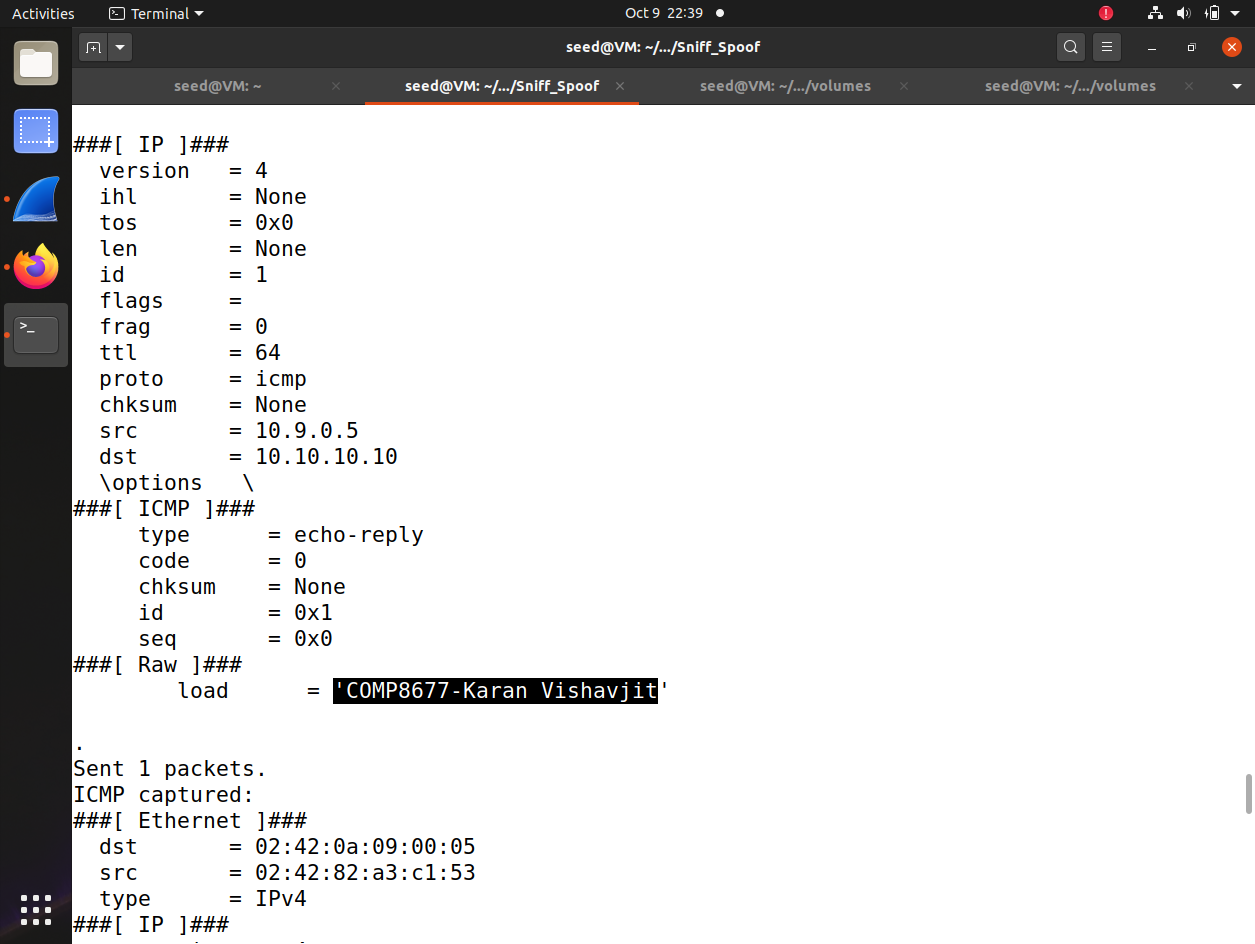
**Run your Wireshark to check if 10.10.10.10 replied to your packet sent by item 2. If yes, give a screenshot for one such packet. In this task, provide your program and the said screenshot.**

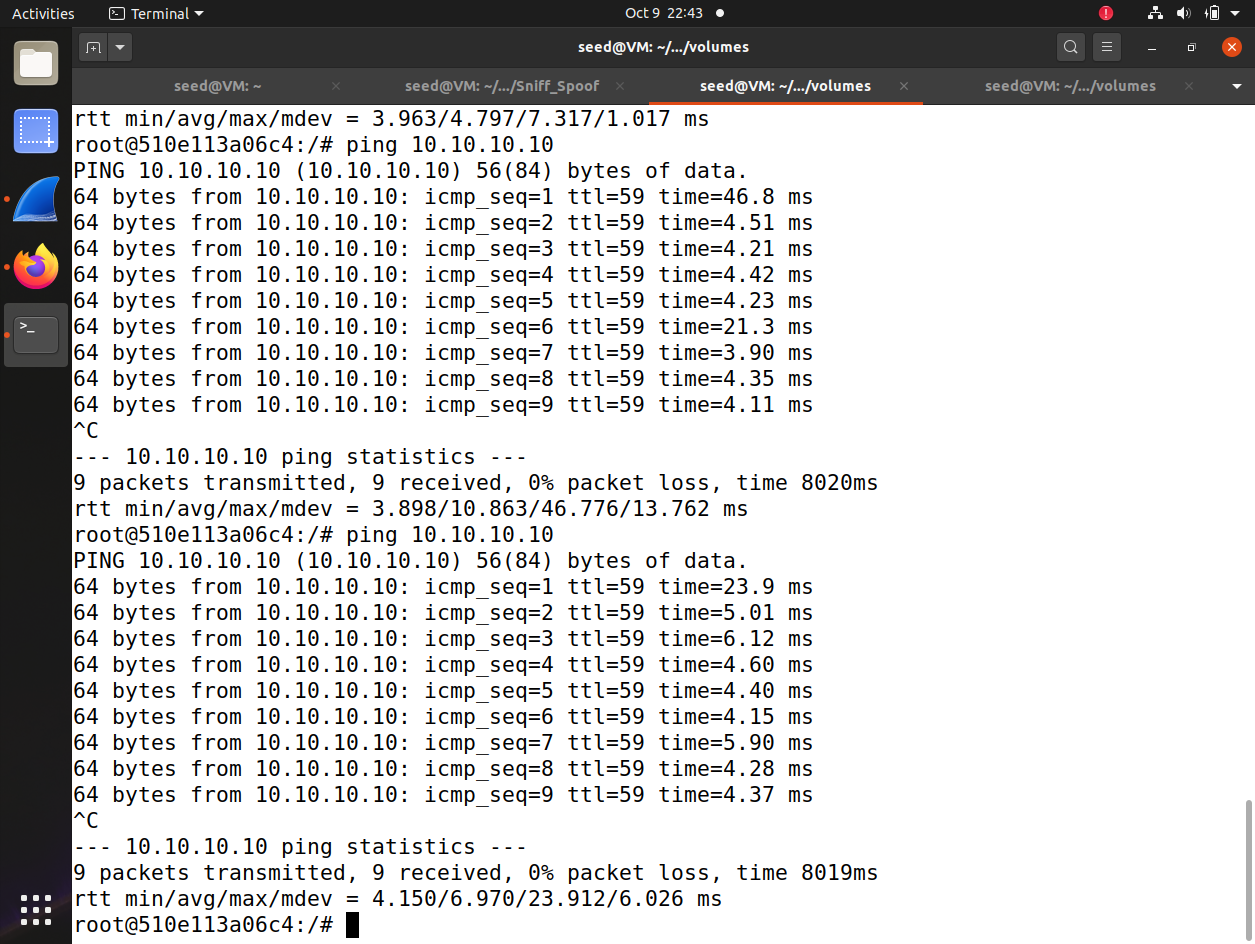


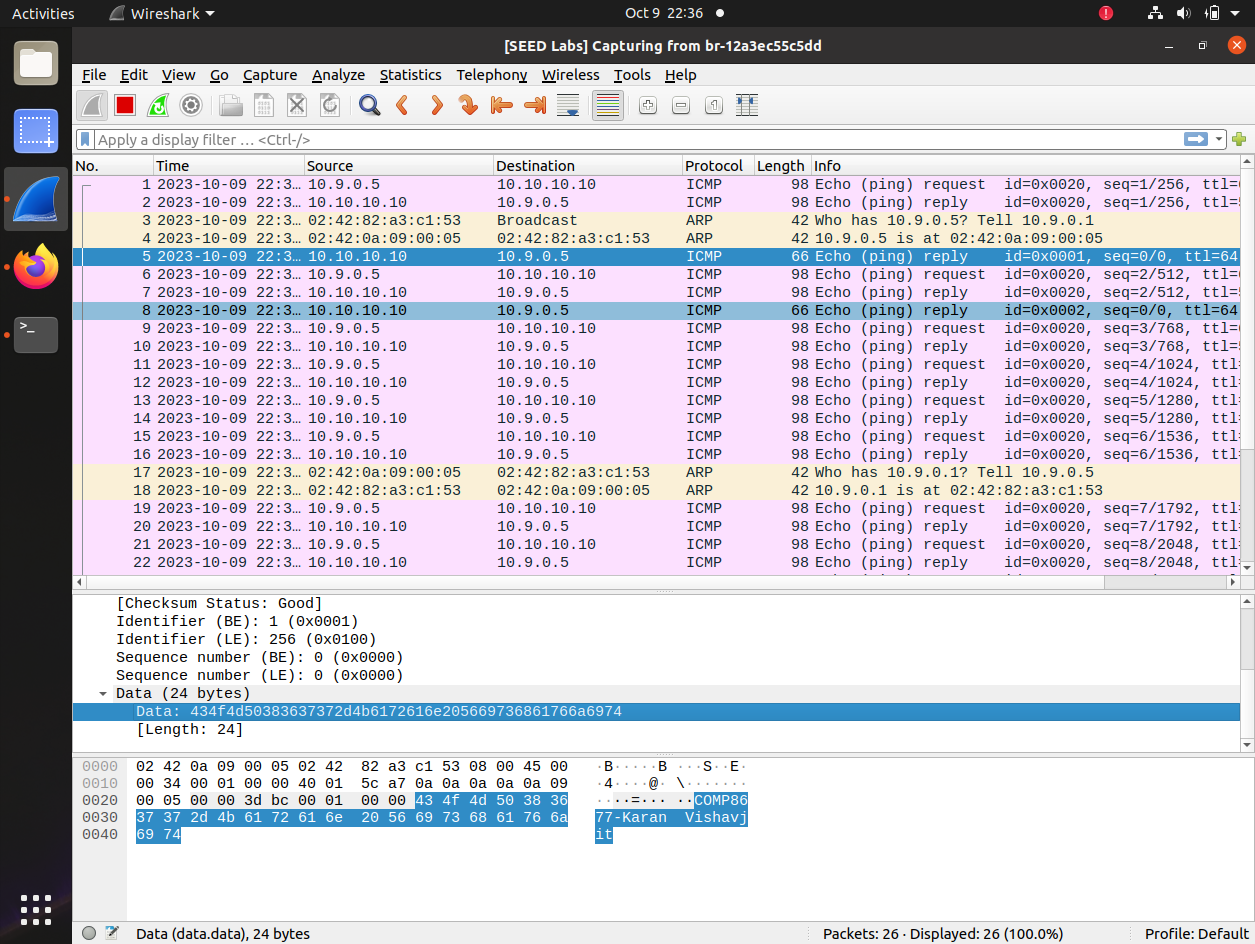










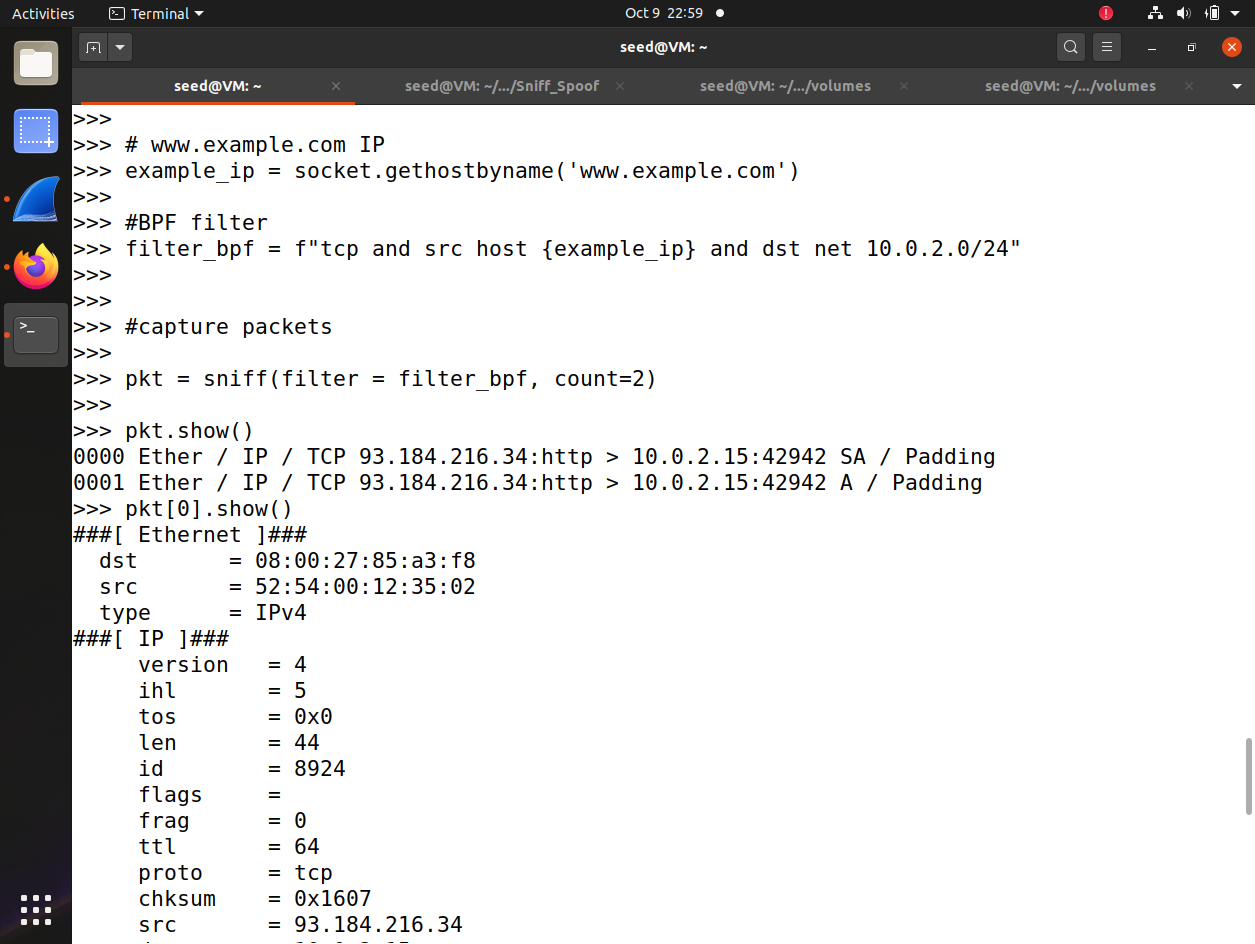


**Task C.**

**In this task, you will practice more for BPF filter. You have studied one in your Task**

**B. If necessary, check the reference file BPF.pdf. Test your solution using the sniff function on the command line of python and show one packet content. Here is my example for the test.**

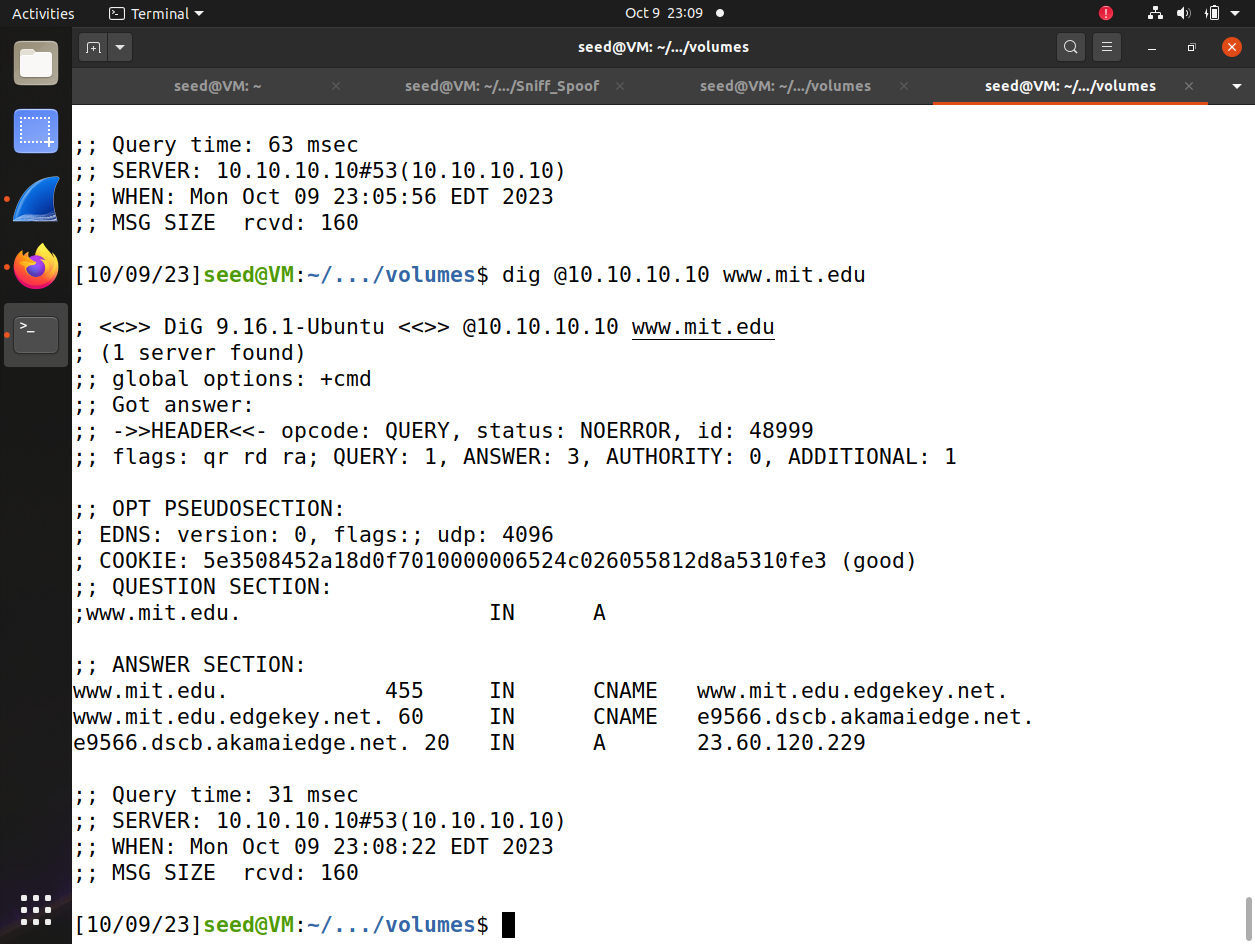
**a) Capture any TCP packet that comes from www.example.com with destination network being your VM subnet (mostly 10.0.2.0/24). We remind you that in order to capture packets from example.com, you of course need to visit the web site.**

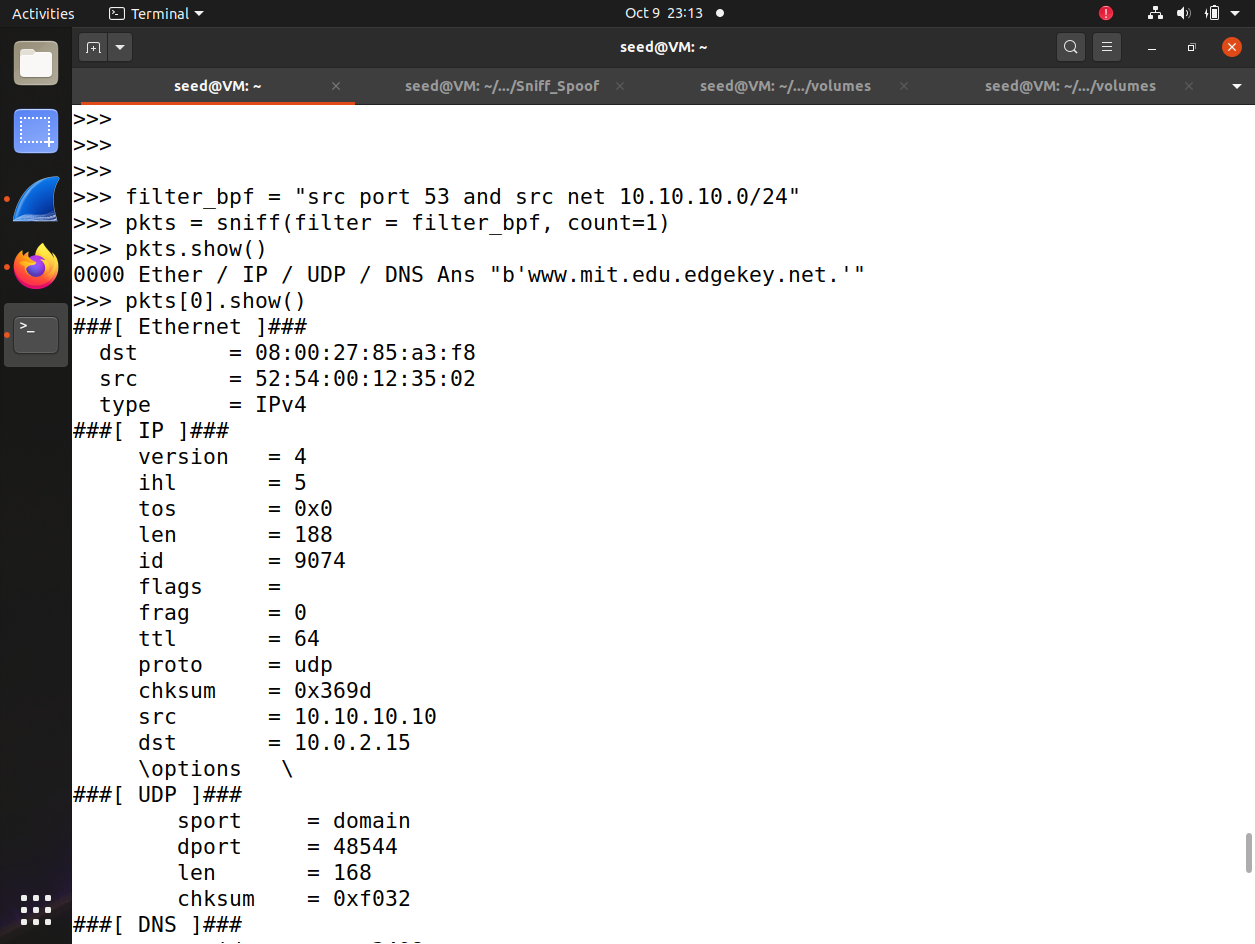


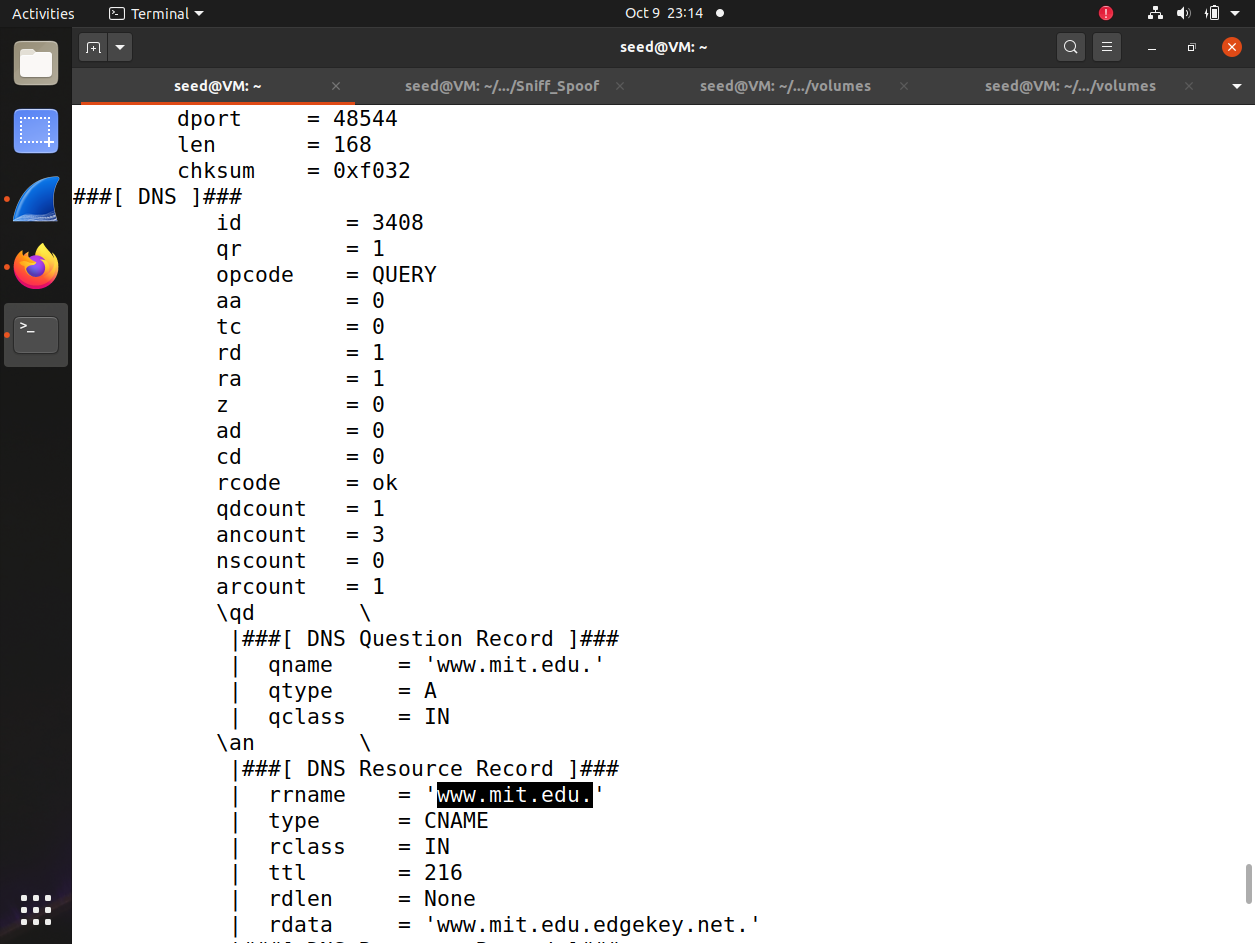


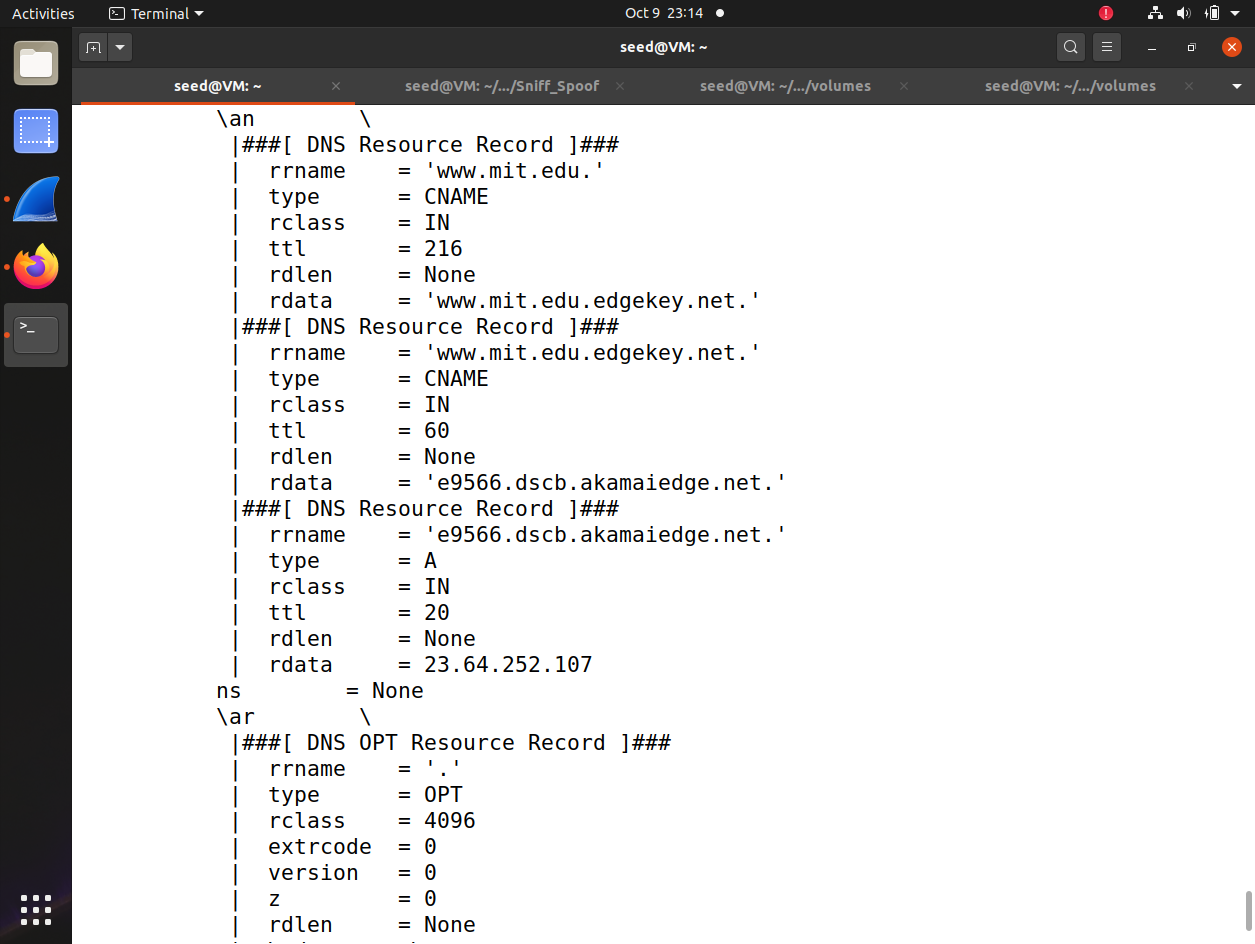
**b) Capture packets that come from source port 53 and a particular network such as 10.10.10.0/24. In the test, run $dig @10.10.10.10 www.mit.edu. (note: if you do your**

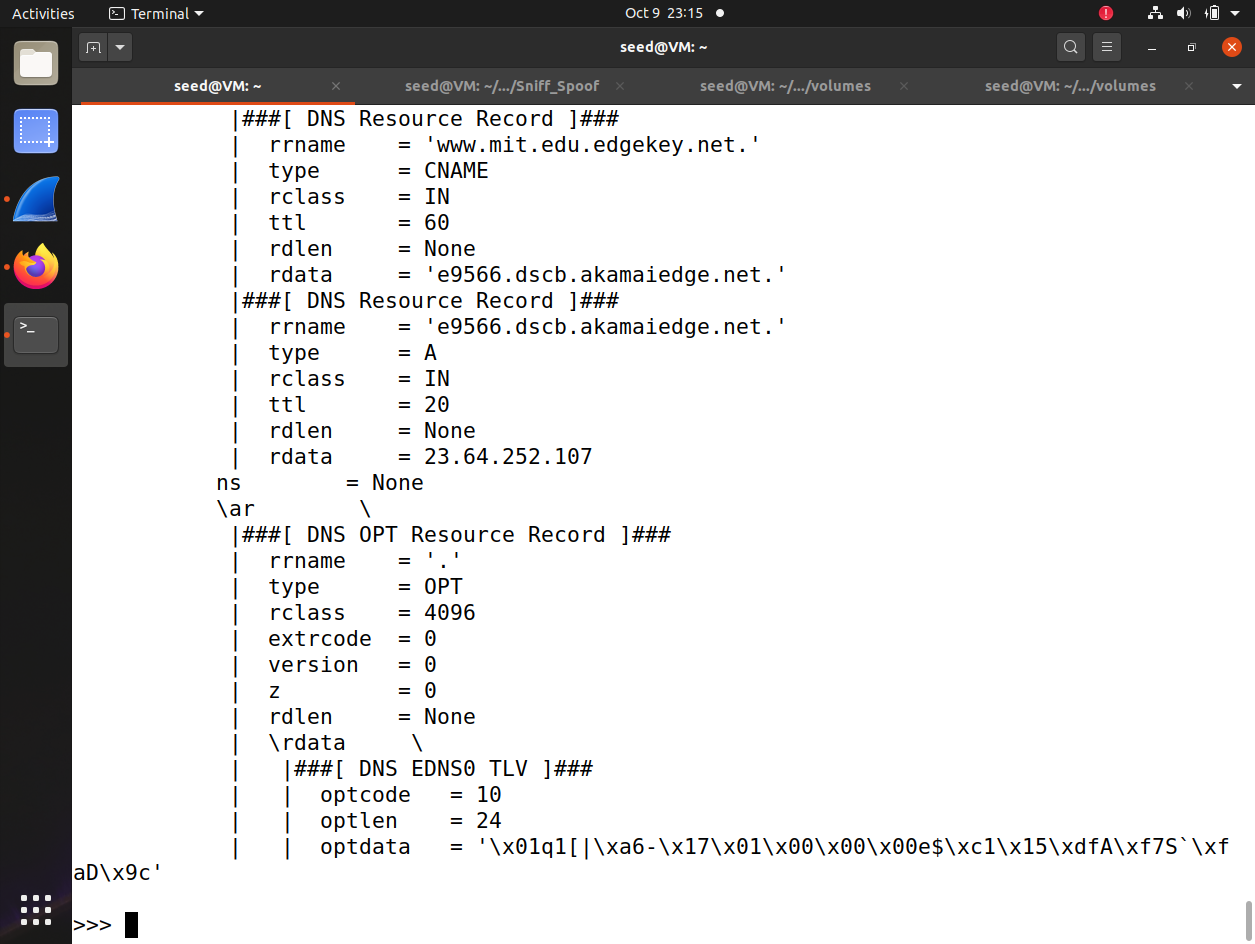
**assignment at home, you can change 10.10.10.10 to 8.8.8.8 and the subnet 8.8.0.0/16).**











1. **ARP Cache Poisoning Attack**

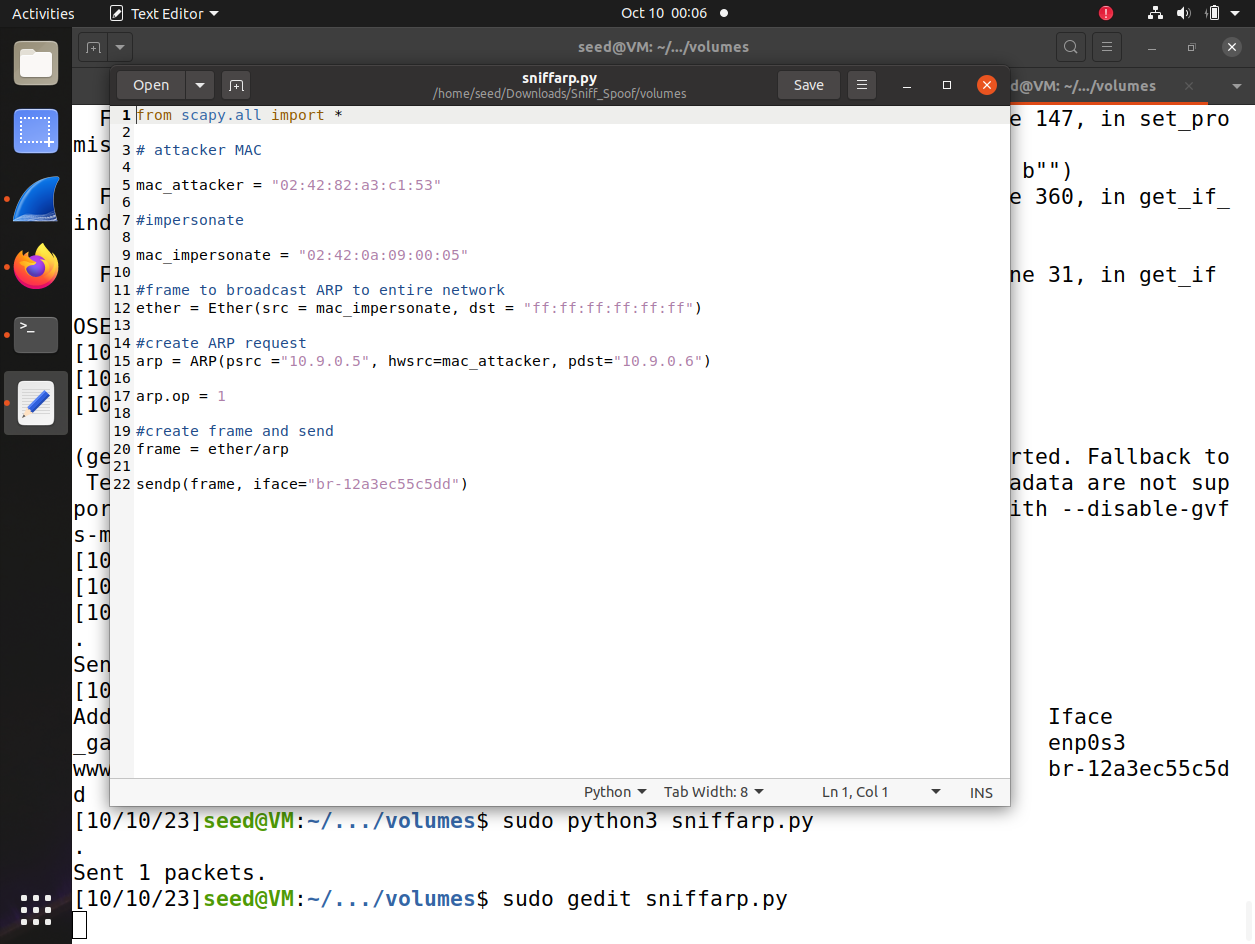
**When our computer needs to send a packet to another computer (such as the gateway router) in the same LAN, it will first run ARP protocol to find this computer’s MAC (if it does not have this in its ARP table). In this problem, you will practice the ARP cache poisoning attack. Using dockercompose.yml, you will simulate a subnet 10.9.0.0/24, containing attacker 10.9.0.1 and users 10.9.0.5, 10.9.0.6. Let mac\_d be the mac address of 10.9.0.d. Our task is to cheat 10.9.0.6 to include entry <10.9.0.5, mac\_1> into its arp table. The following is the incomplete code with question marks. Note: sendp() is similar to send() but it sends a link layer frame while send() only sends an ip packet. mac\_d should be replaced with the actual mac address of 10.9.0.d.**

**Complete the following tasks:**

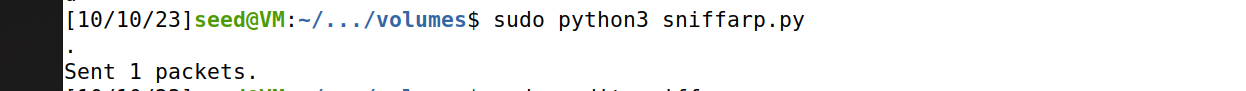
**1. complete the above program.**

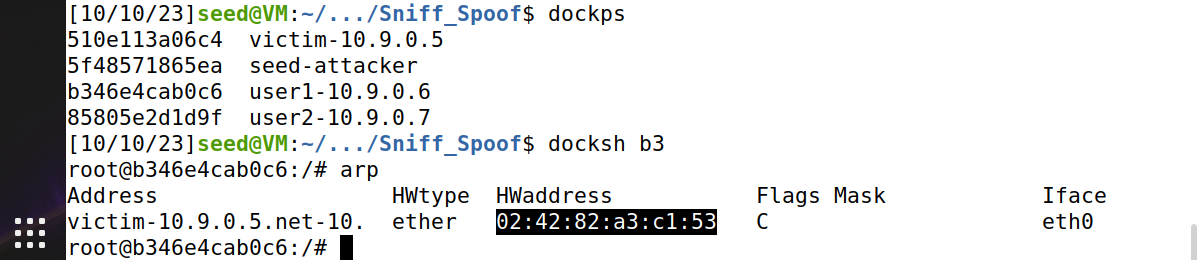
**Hint: this runs ARP protocol and so it should indicate it is a broadcast frame; attack idea: if**

**10.9.0.6 receives an ARP query, it will record (sender IP, sender MAC) of the query frame into its ARP table.**



1. **run arp on 10.9.0.6 to see if entry (10.9.0.5, MAC\_1) is included in its arp table.**





**3. Explain how the attack works.**

Attacker prepares the attack by setting up specific MAC address (mac\_attacker = "02:42:82:a3:c1:53") and another MAC address they want to impersonate (mac\_impersonate = "02:42:0a:09:00:05").

Packet is crafted with to appear as if it is coming from impersonated IP address (psrc ="10.9.0.5") but attackers MAC address (hwsrc=mac\_attacker) as the source.

Message is broadcasted to entire local network using broad cast MAC address ("ff:ff:ff:ff:ff:ff").

Devices on the network on receiving this request update their ARP tables and when legitimate device with ip (“10.9.0.6”) responds to the request, its response is ignored because ARP table is already updated with attackers MAC address for this IP.

As a result traffic intended for IP (“10.9.0.5”) from (“10.9.0.6”) is sent to attacker machine instead and attacker can intercept, modify as desired.