AI Cap’n

CPSC 481 Project

Summer 2018

By Steven Kha and Danh Pham

CPSC 353-01

Introduction to Computer Security

Professor Kenytt Avery

By Anthony Le and Danh Pham

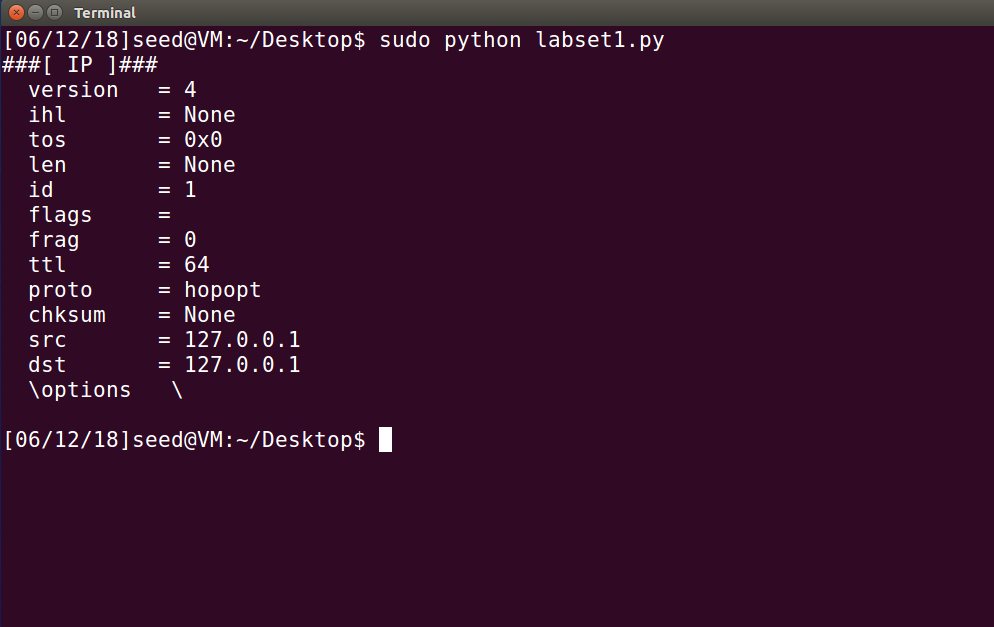
Requirements :

1. Using Tools to Sniff and Spoof Packets
2. Task 1.1: Sniffing Packets
3. Task 1.2: Spoofing ICMP Packets
4. Task 1.3: Traceroute
5. Task 1.4: Sniffing and-then Spoofing

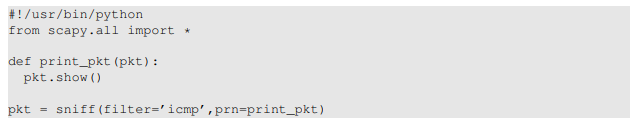
Instruction:

1. Using scapy functionalities into our example python code.
2. After typing “from scapy.all import \*” into our program example
3. Using command “sudo python labset1.py “ to run the program

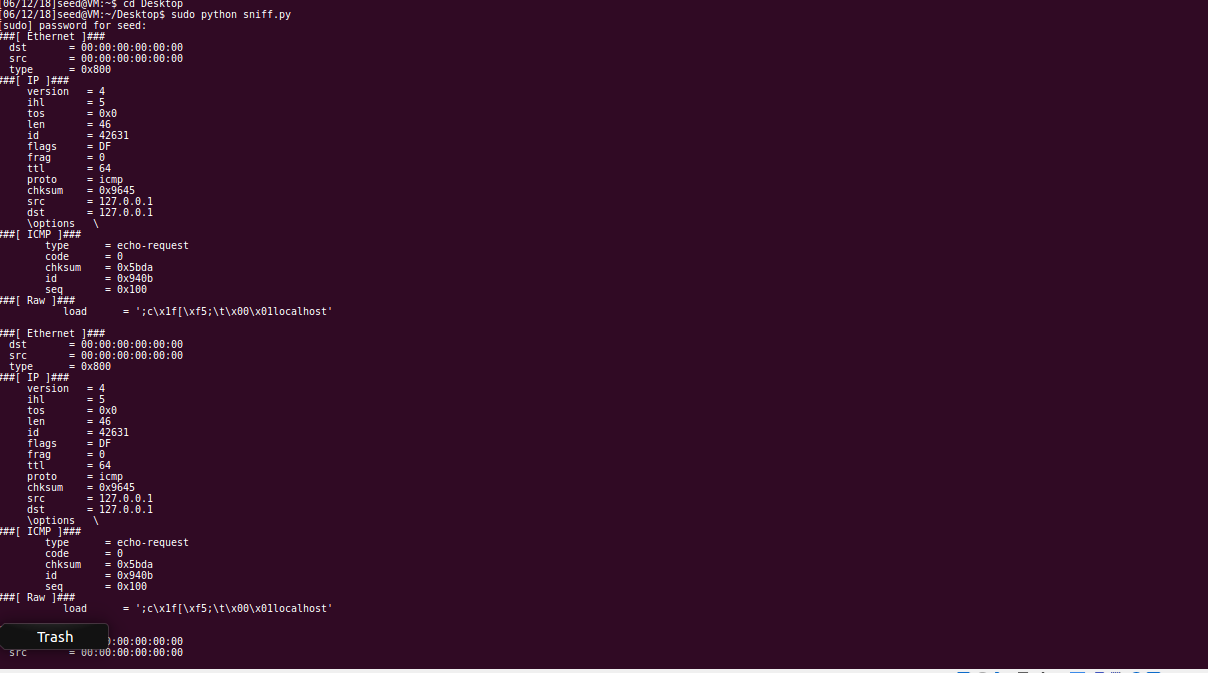
Result screen shot:



Task 1.1A we also using scapy library as the same as the labset1 but the code for snipping will be different. And the sniff python code already provide in the description.

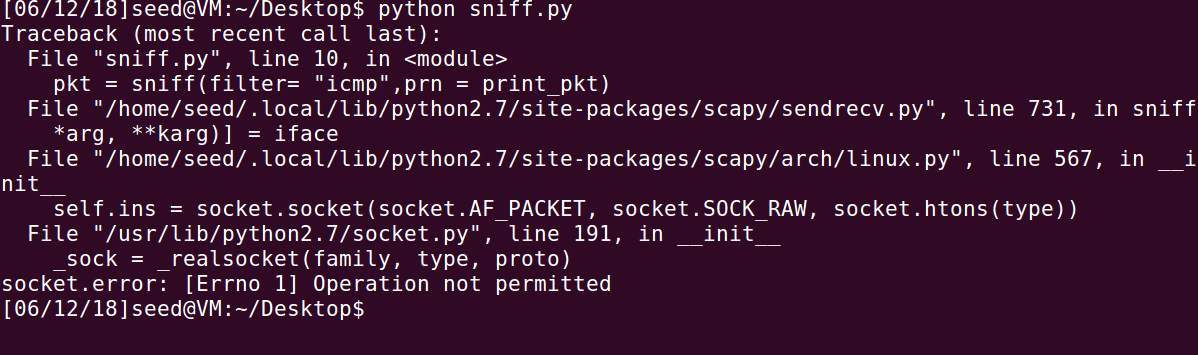


1. Task 1.1 A, we need to use the code to test with wireshark (setting VM network should be NAT only for this task).
2. Run command line in terminal “sudo python sniff.py “ .
3. Result should print out some data in your terminal which it filler with ICMP protocol .



D. Run the command “ python sniff.py” to see the result

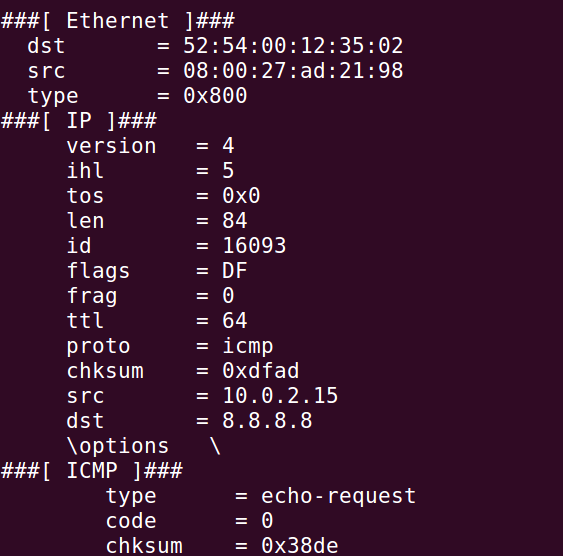
Result: as the screenshot below we can see clearly that if we run the sniff.py without sudo (root privileges ) the problem will not execute in our terminal.

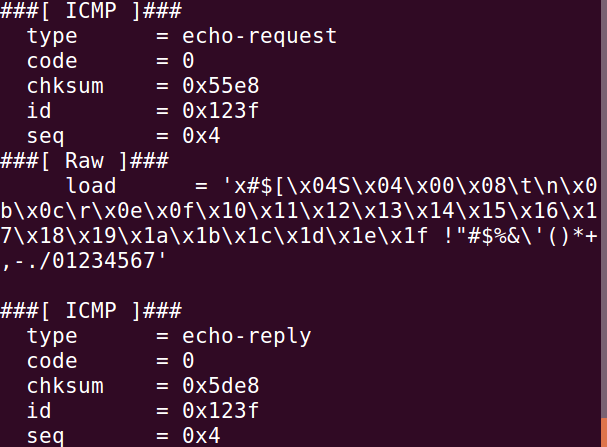


Task 1B Using Scapy’s filter (BPF syntax) complete the following:

* Capture only the ICMP packet
  + Accompanying file: sniffer.py

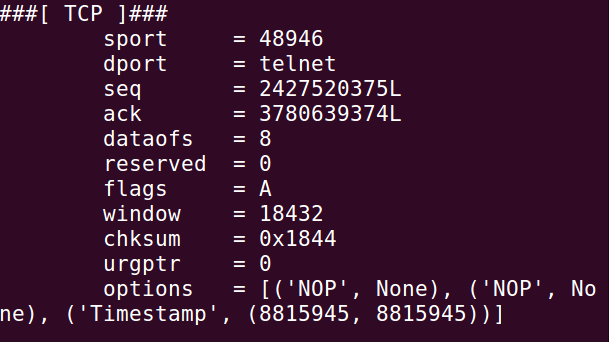
For this objective we simply ran the code provided by the instructions. There was no need to change the filter as it was already set to filter=’icmp’, which exactly what we wanted.





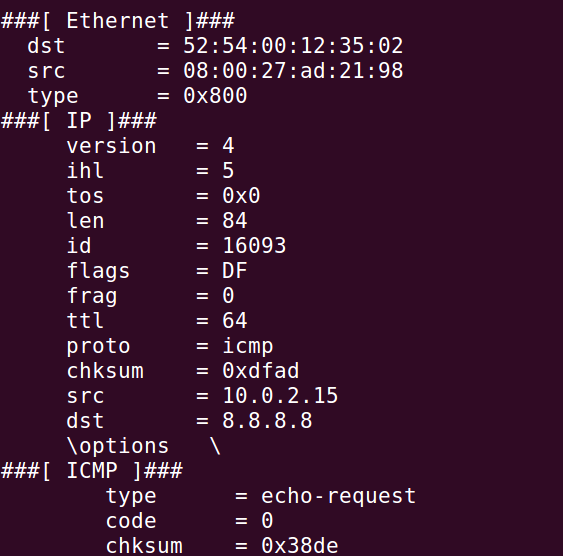
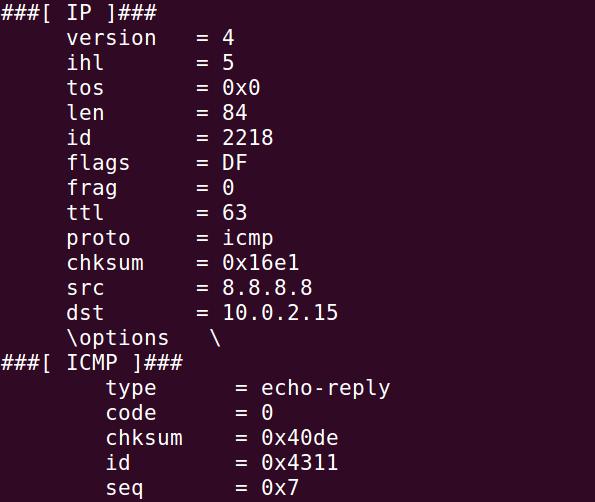
* Capture any TCP packet that comes from a particular IP and with a destination port number 23.
  + Accompanying file: snifferB.py

For this objective the filter had to be changed to the target host and to only tcp with port 23 so that when we use telnet to tell it which specific net to use, the default port shall be 23. That means that where “dport” is displayed, even though it says “telnet” it is actually port 23, which is the port we told it to use.



* 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
  + Accompanying file: snifferC.py.

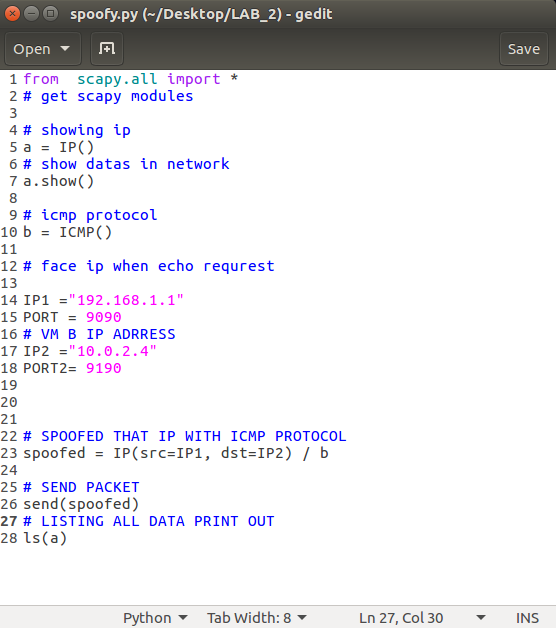
For this objective, in order to filter packets that come from a particular subnet we needed to use “net” and then specify a network such as “8.8.8.0/24” which will only show packets to and from that subnet.

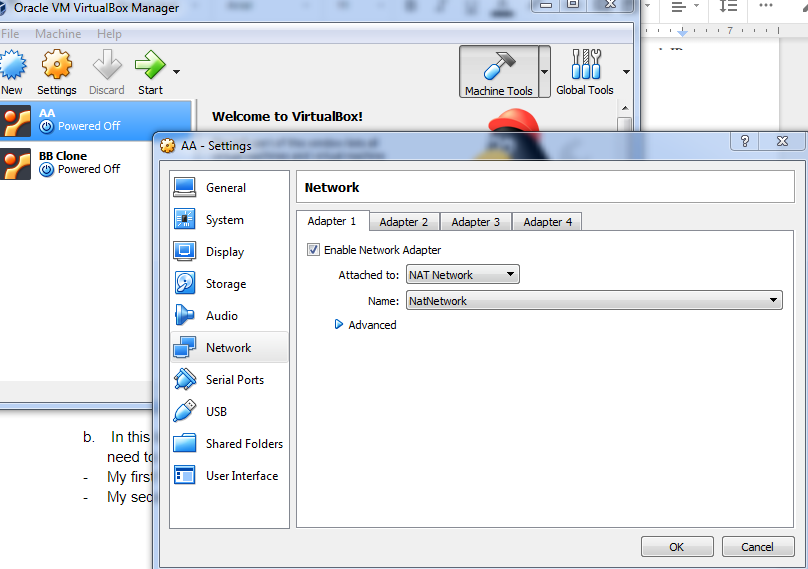


2. Spoofing ICMP packets , by editing the given code below from description

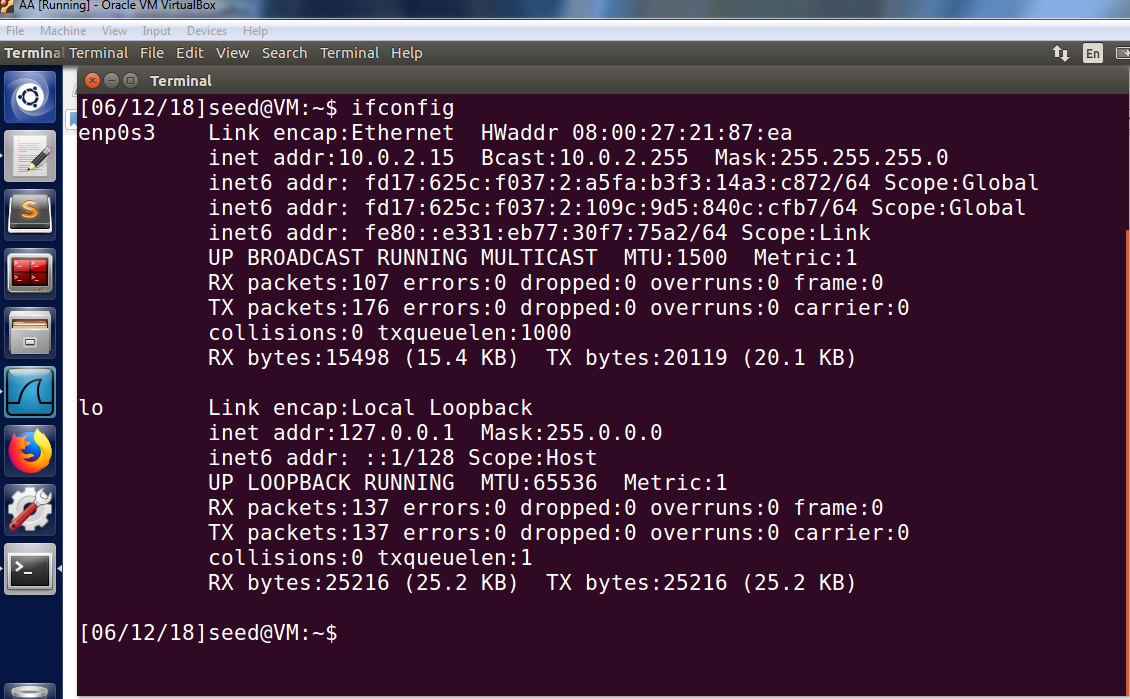


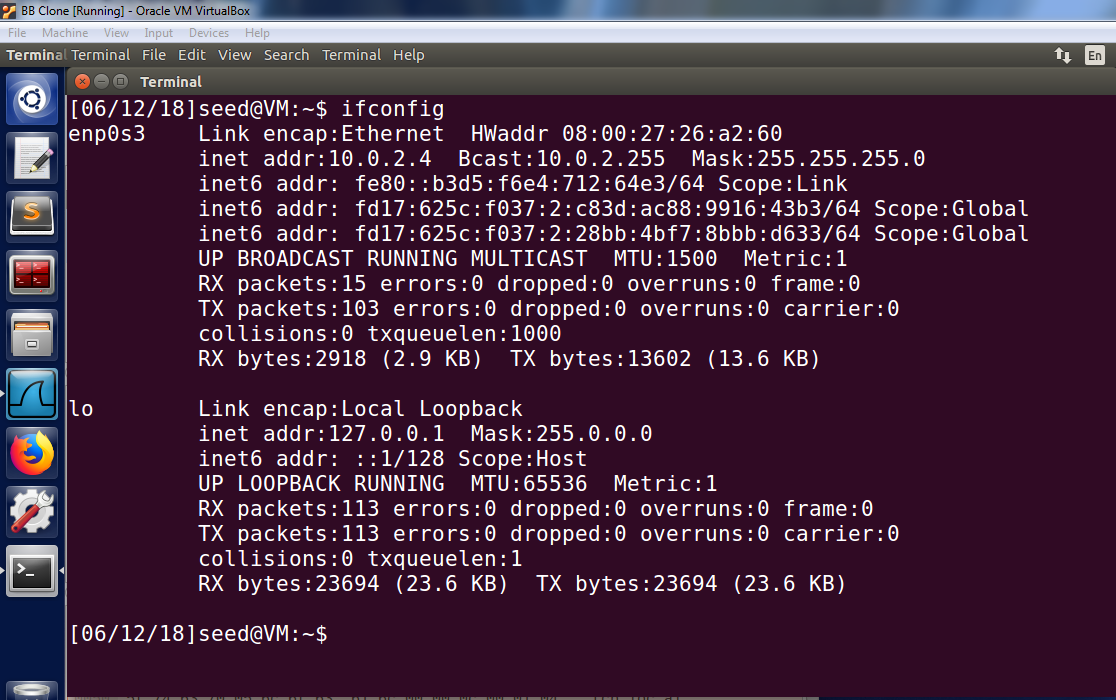
1. Editing python code

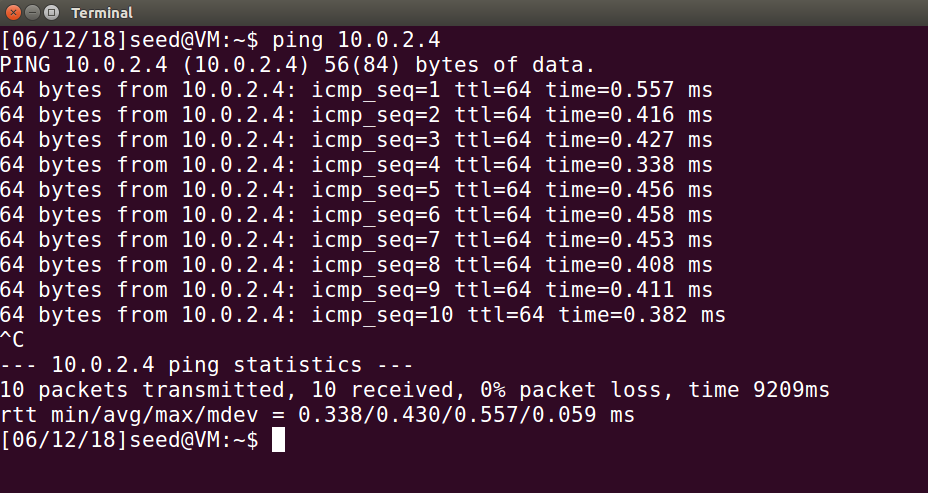


1. In this task we need to setting two VM devices. ( also in the setting virtual machine we need to change NAT under the network to “NAT Network”)

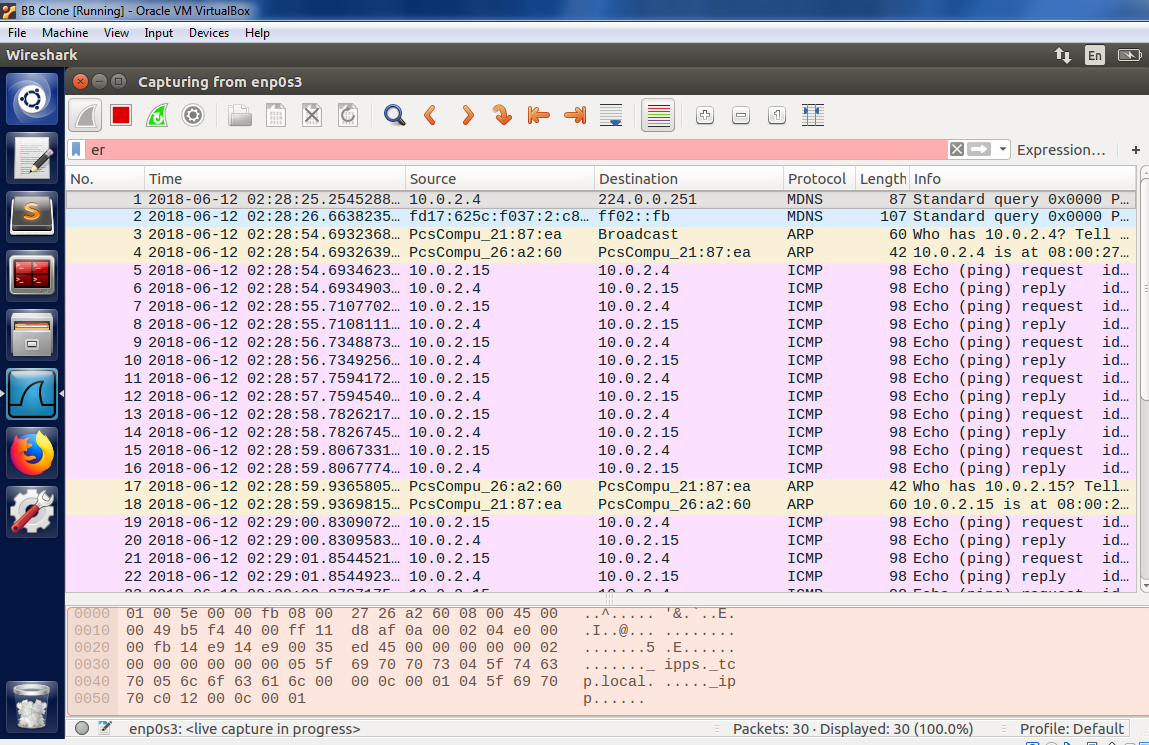
* My first VM device with IP: 10.0.2.15



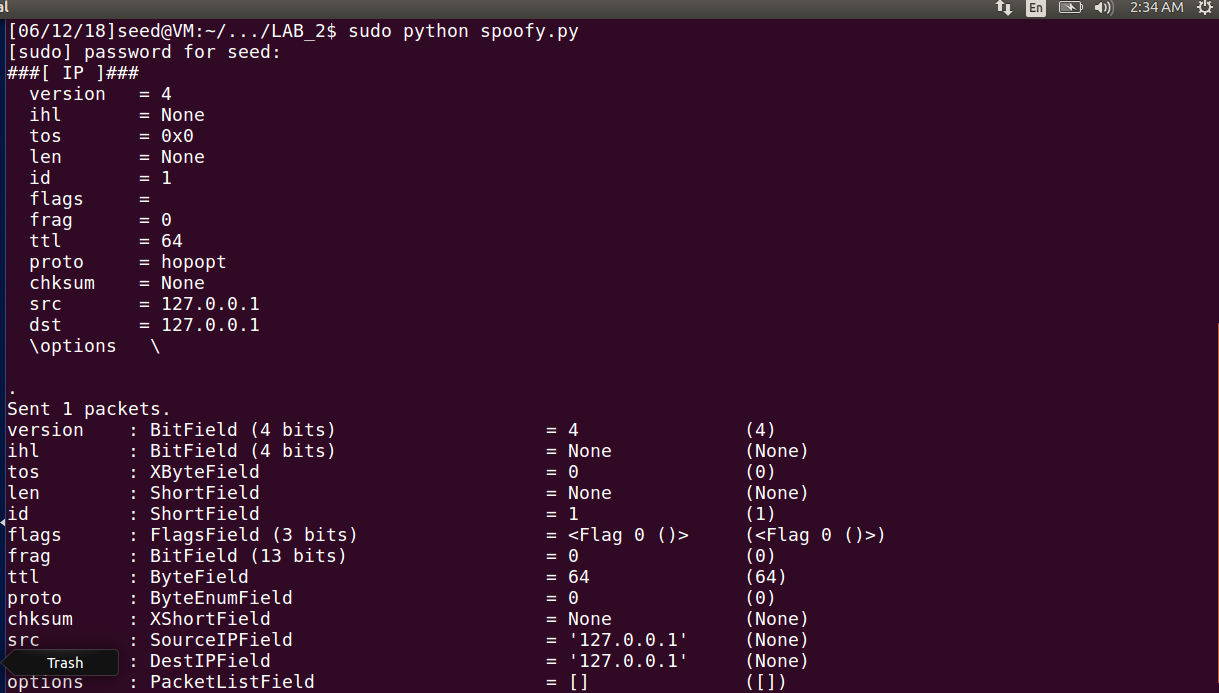
* My second VM device with IP: 10.0.2.4

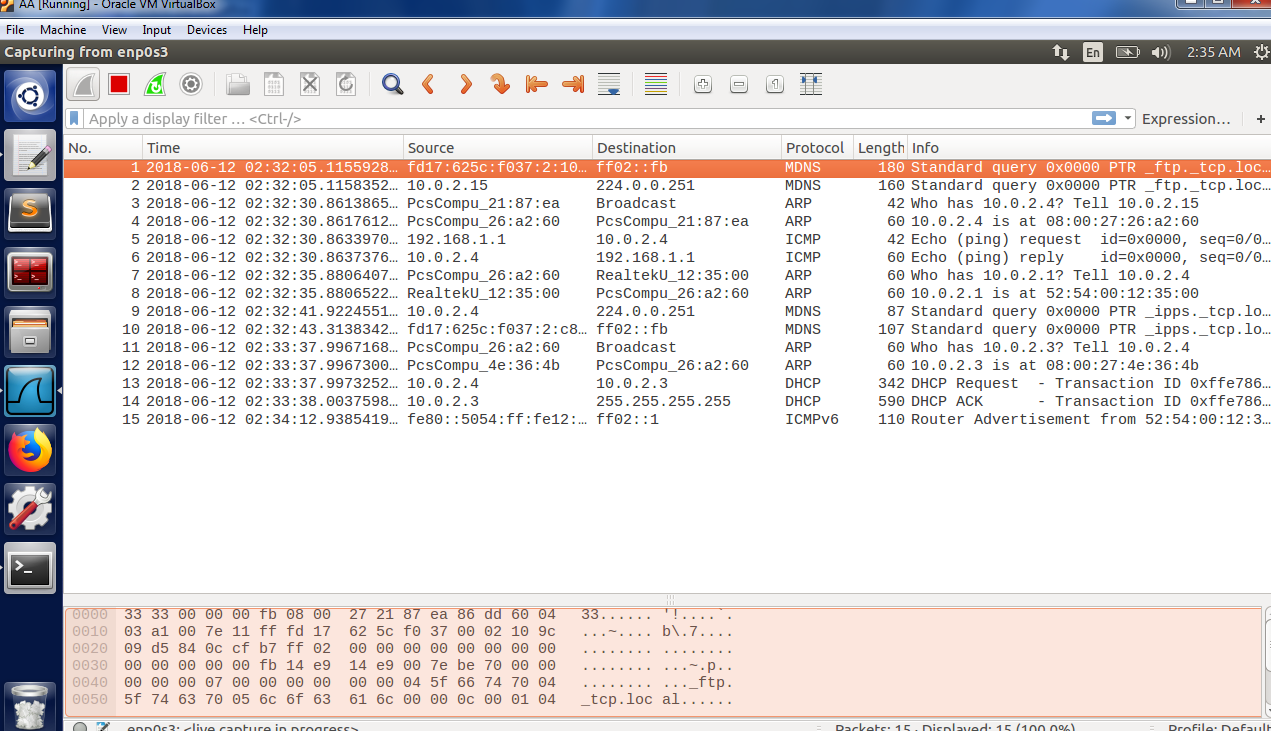
C. In terminal VM A we use “ ping 10.0.2.4 “ which it is VM B for verifying that VM B is alive (by capturing using Wireshark): note open wireshark capture when we try to do this step.

* Result from wireshark VM B after ping command : we can see clearly that echo icmp from my vm A- ip 10.0.2.15 ping to 10.0.2.4 VM B.

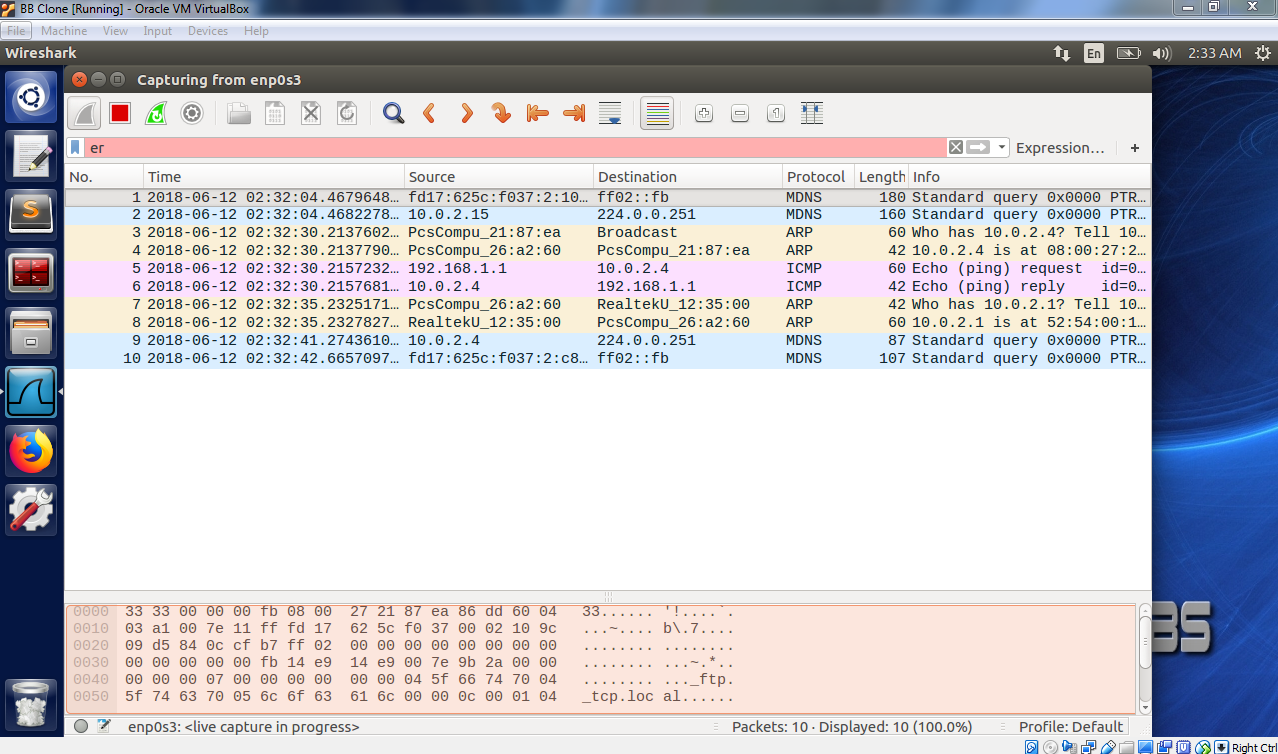


D. now we can the command “ sudo python spoofy.py “ in terminal VM A .

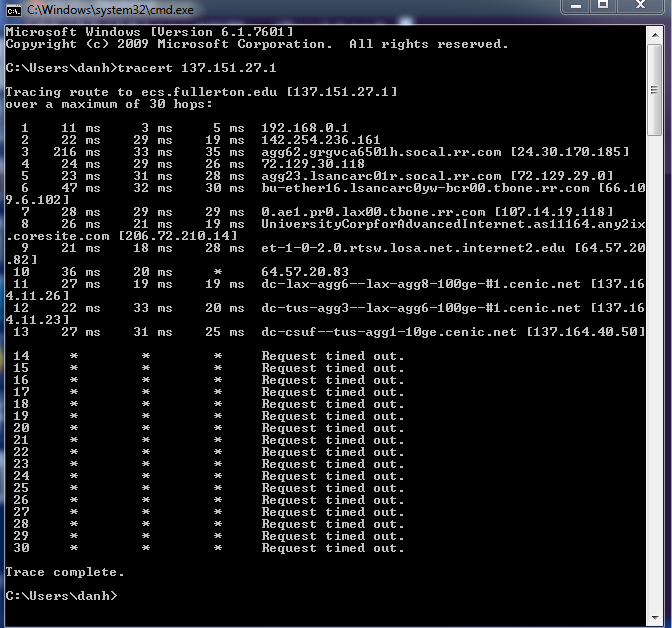




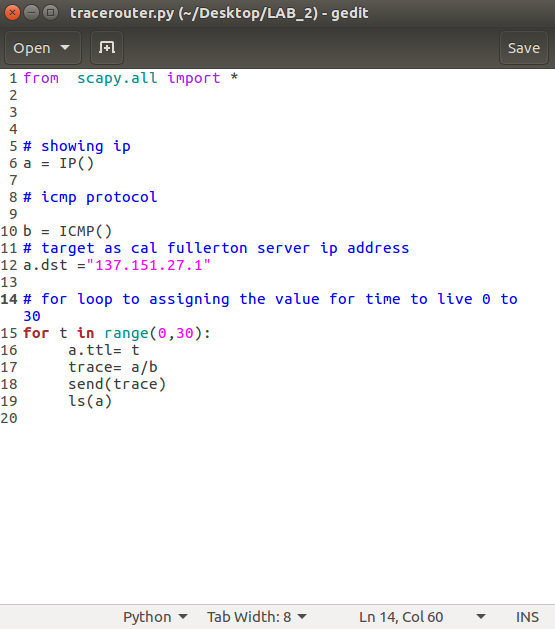
E. Go back to wireshark VM B to see the result : we can see clearly that the fake ip 192.168.1.1 replaced with ip 10.0.2.15 as the original IP from VM A.



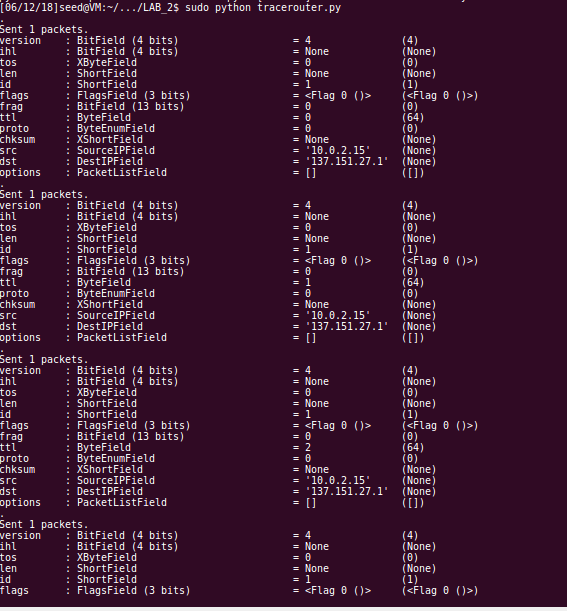
3.Traceroute

1. As a testing with window cmd, we try to use “ tracert 137.151.27.1 “ This Ip address belongs to Cal State Fullerton. 
2. In this task, we need to change our setting VM back to NAT only
3. After that we will open wireshark in VM A then try to capture the echo request from our ip to the fullerton server.
4. Run the command “sudo python tracerouter.py ”

Note: In my python code I am using a for loop to changing the ttl one at the time.

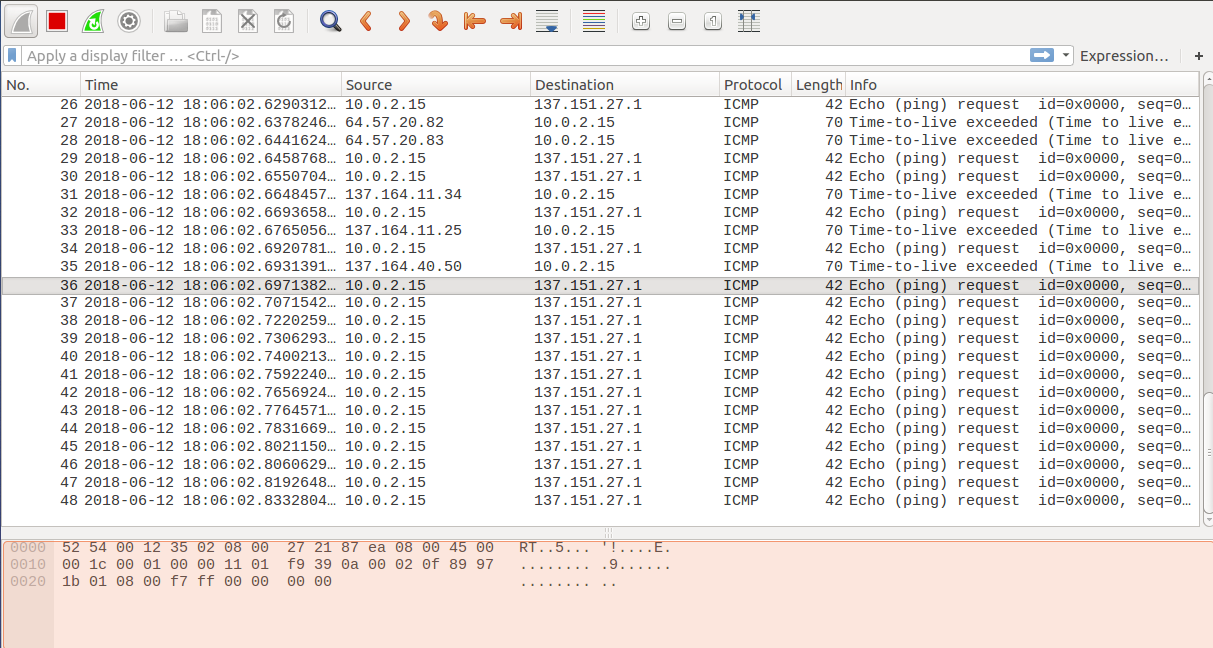


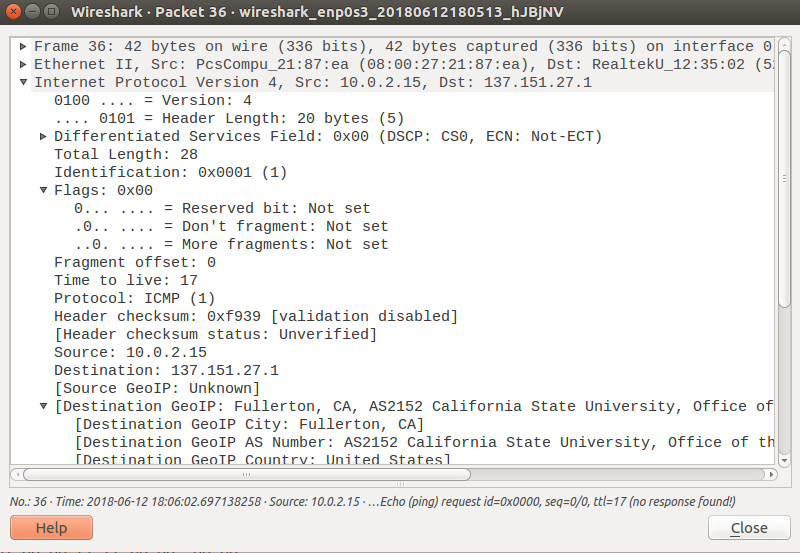
Result after run the command: we can see that the ttl changed 0 to 30 when sent 1 packets at the time.



1. Now we check wireshark to see how long does it takes to get the destination.

Conclusion: As the wireshark interface we can see that the packet # 36 is the last one get ping( so we assume that is the last destination), with a time to live = 17.





4. Sniffing then Spoofing

Instructions:

Combine the sniffing and spoofing techniques to implement the following sniff-and then-spoof program.

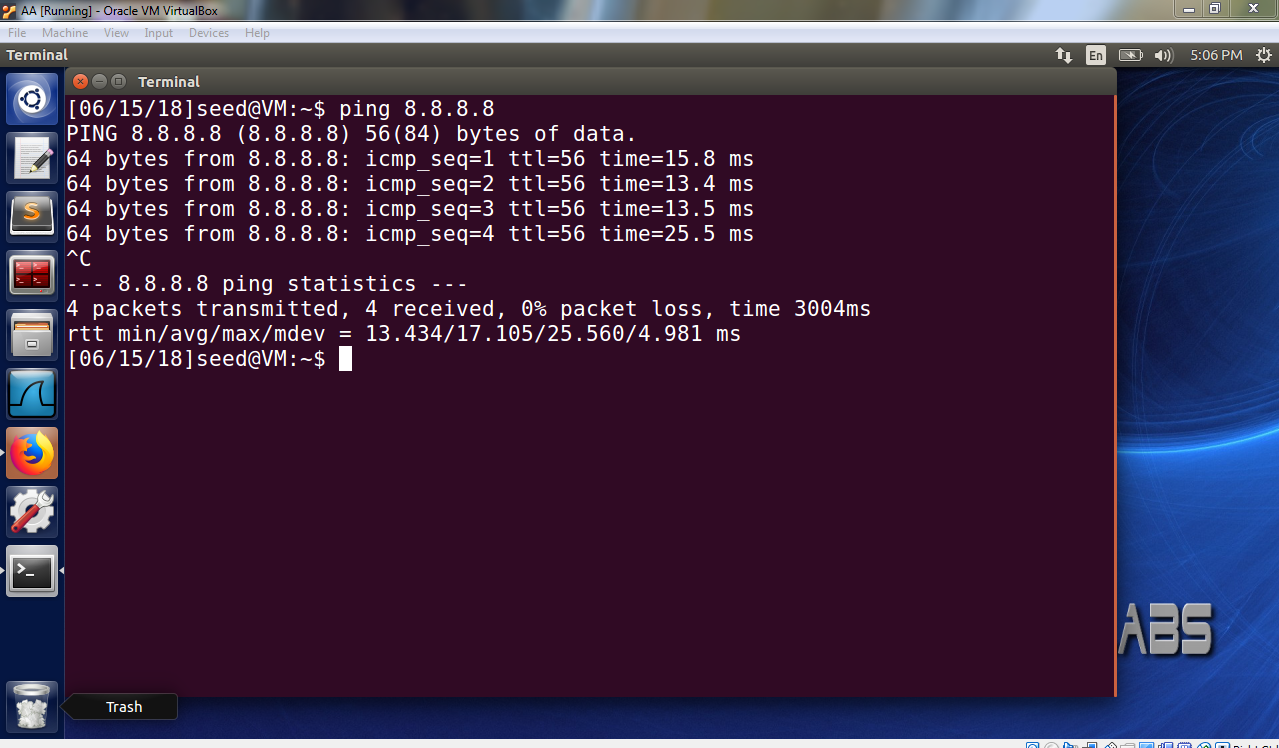
* From VM A, you ping an IP X, it should receive an echo-reply.
* From VM B, you run the sniff and spoof which monitors the LAN through packet sniffing.
  + Regardless of what the target IP address is, should send out an echo reply indicating that X is alive.

This portion of the lab uses the “sniffNspoof.py” file.

Steps:

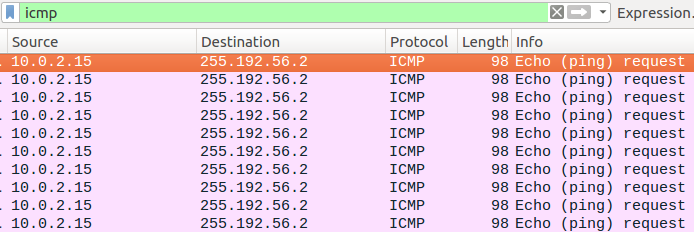
1. Clone your seed ubuntu
2. Changing setting network to “NAT NETWORK”
3. Try to do the ifconfig to see inet address( VM A should be different from VM B)
4. See Appendices A and B of [Run SEED VM on VirtualBox](http://www.cis.syr.edu/~wedu/seed/Labs_16.04/Documents/SEEDVM_VirtualBoxManual.pdf) for configuration details

a . Ping to verify “ ping 8.8.8.8” on VM A .



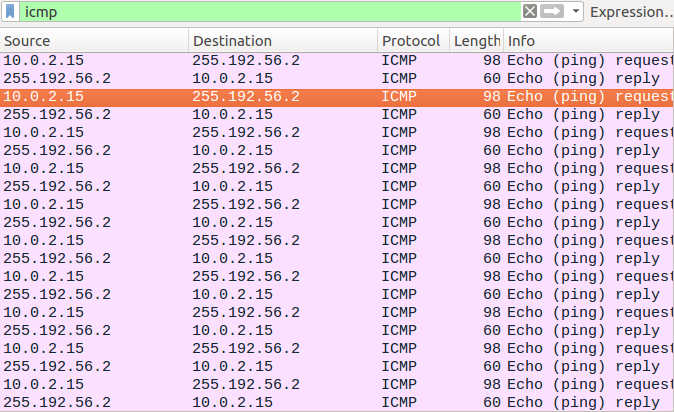
B. The purpose of this task is to fool VM A into thinking that its ping’s to any random IP are being received and replied to. Therefore to verify we use VM A’s Wireshark to check the correspondence VM A thinks it’s having.

For our purposes, we will randomly generate an IP: 255.192.56.2, here is what it looks like from VM A’s perspective without the program intercepting the pings and replying to them. The IP of VM A is 10.0.2.15.



Note that there are only echo-requests from VM A to 255.192.56.2, this is because that IP does not exist and/or is otherwise unavailable and therefore cannot respond.

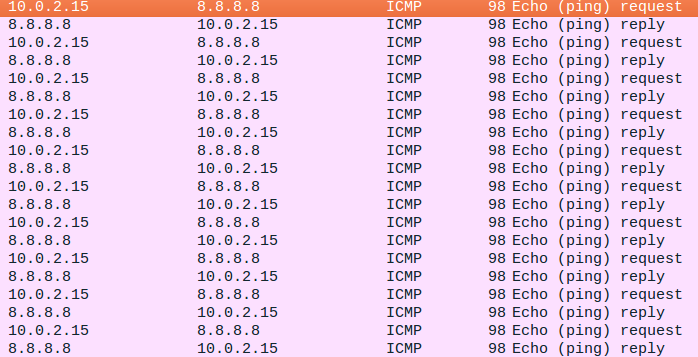
Now let us see how this looks after our program sniffs the packets sent and intercepts them.



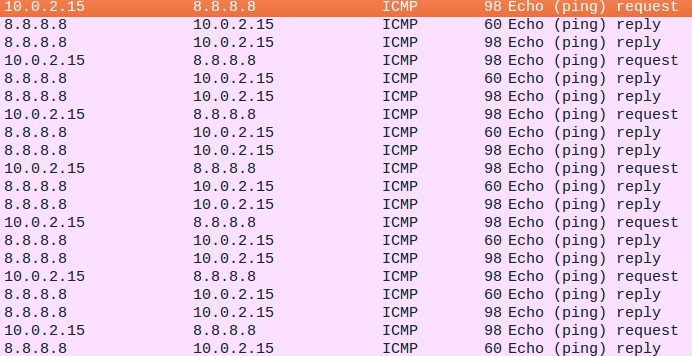
Now we can see VM A send an echo-request to 255.192.56.2 and see that IP reply back to VM A, indicating that 255.192.56.2 is alive and that it is ok to begin correspondence. This is achieved by having VM B on the same network as VM A and having that network be in promiscuous mode for both VMs, allowing VM B to easily monitor VM A packet transmissions. Allowing us to utilize Scapy’s sniff() command, in the filter we narrowed it down to be of “ICMP” type packets and since we know the network of the target we also filtered for “src net 10.0.2.0/24” which should pick up all ICMP packets from the 10.0.2.0 network.

After successfully sniffing VM A’s packets it was a simple matter of rearranging the source and destinations and assigning these to a new packet with an ICMP(type=0), indicating that it should be a reply. Once, this packet is constructed we just send it back to whomever sent it using Scapy’s .send() command, and so VM A receives an echo-reply for it’s echo-requests and believes that IP 255.192.56.2 is alive.

To verify, here’s what it looks like when VM A ping’s a legitimate IP that is alive without use of our program.



Here is what it would look like if VM A pinged 8.8.8.8 with the use of our program.



Note that when the program sends a reply the length of the packet is only 60 bytes, when the legitimate IP sends a reply the length is 98 bytes. Our program sends a reply before the legitimate IP does, and so 8.8.8.8 receives 2 echo-replies.