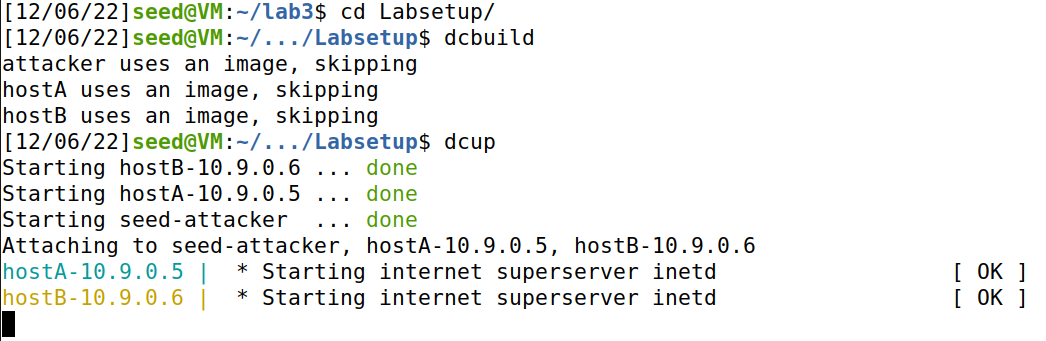
Jourdon Freeman

862006435

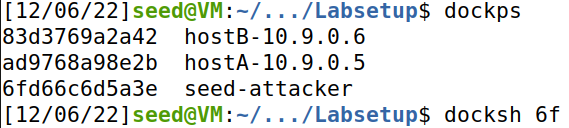
Lab 3

# Introduction & Setup

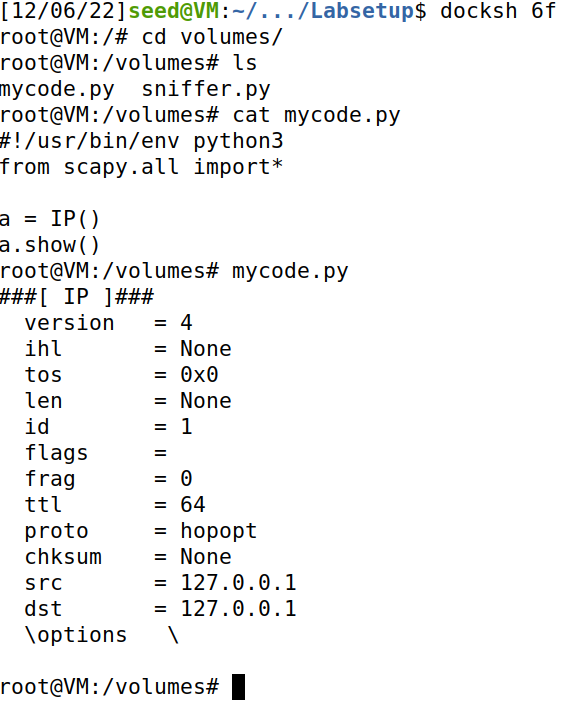
In this lab, we’re using Scapy to intercept data between two hosts—A and B.



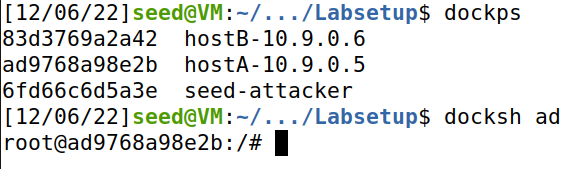
Setting up the hosts and attacker



In a new tab, logging into attacker



Running mycode.py from volumes folder in root

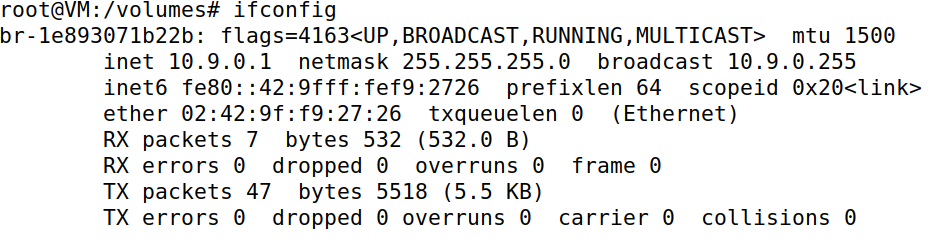


In another tab, logging into hostA

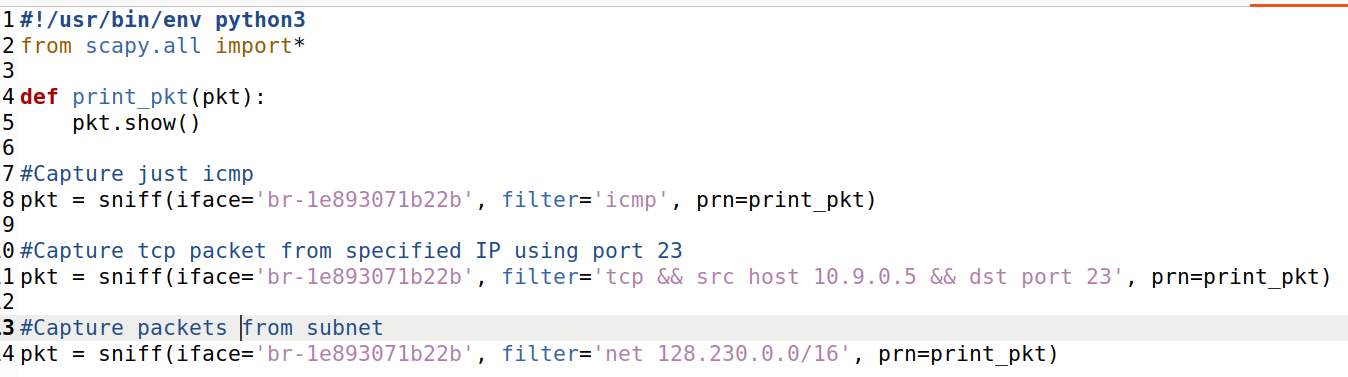
From here, we have everything needed to complete the lab.

# 1.1

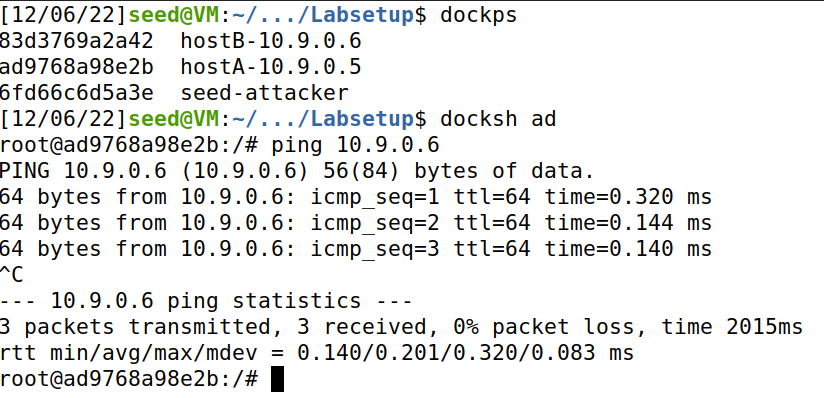
1.1 tasks us with getting 3 kinds of packets to intercept from hosts A and B. First, let’s set up sniffer.py using our own network interface and get each one at a time. (the tasks not in use will be commented out)



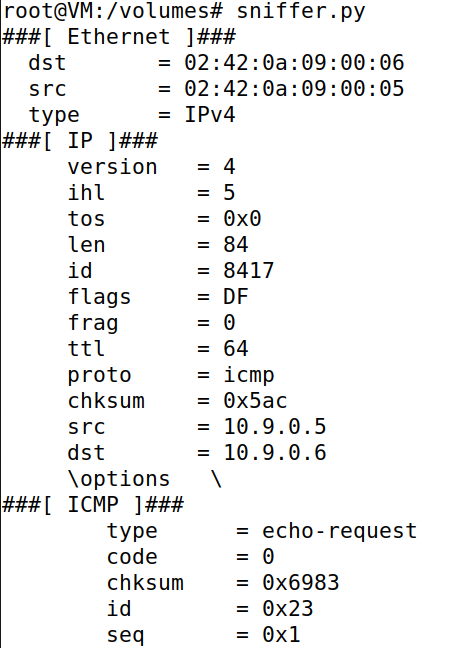
Running *ifconfig* to get network interface name



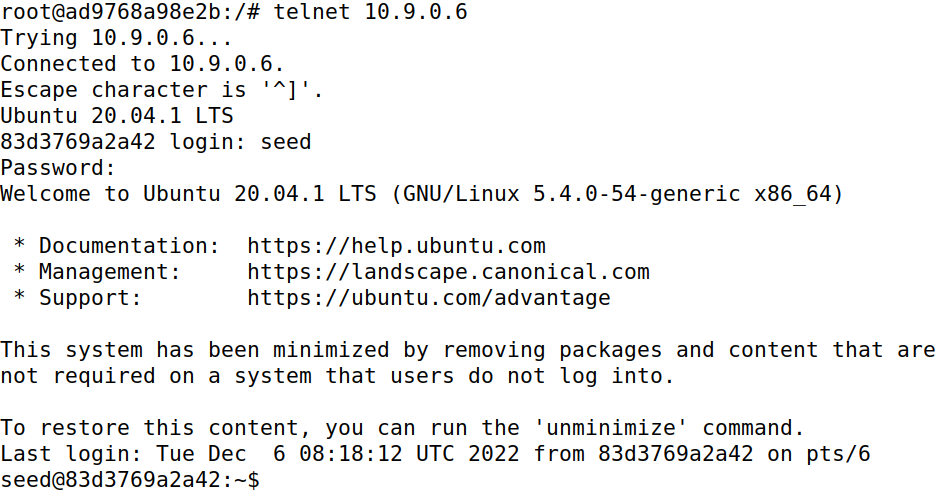
Sniffer.py is now properly setup and ready to handle the first task: Capture only the ICMP packet.



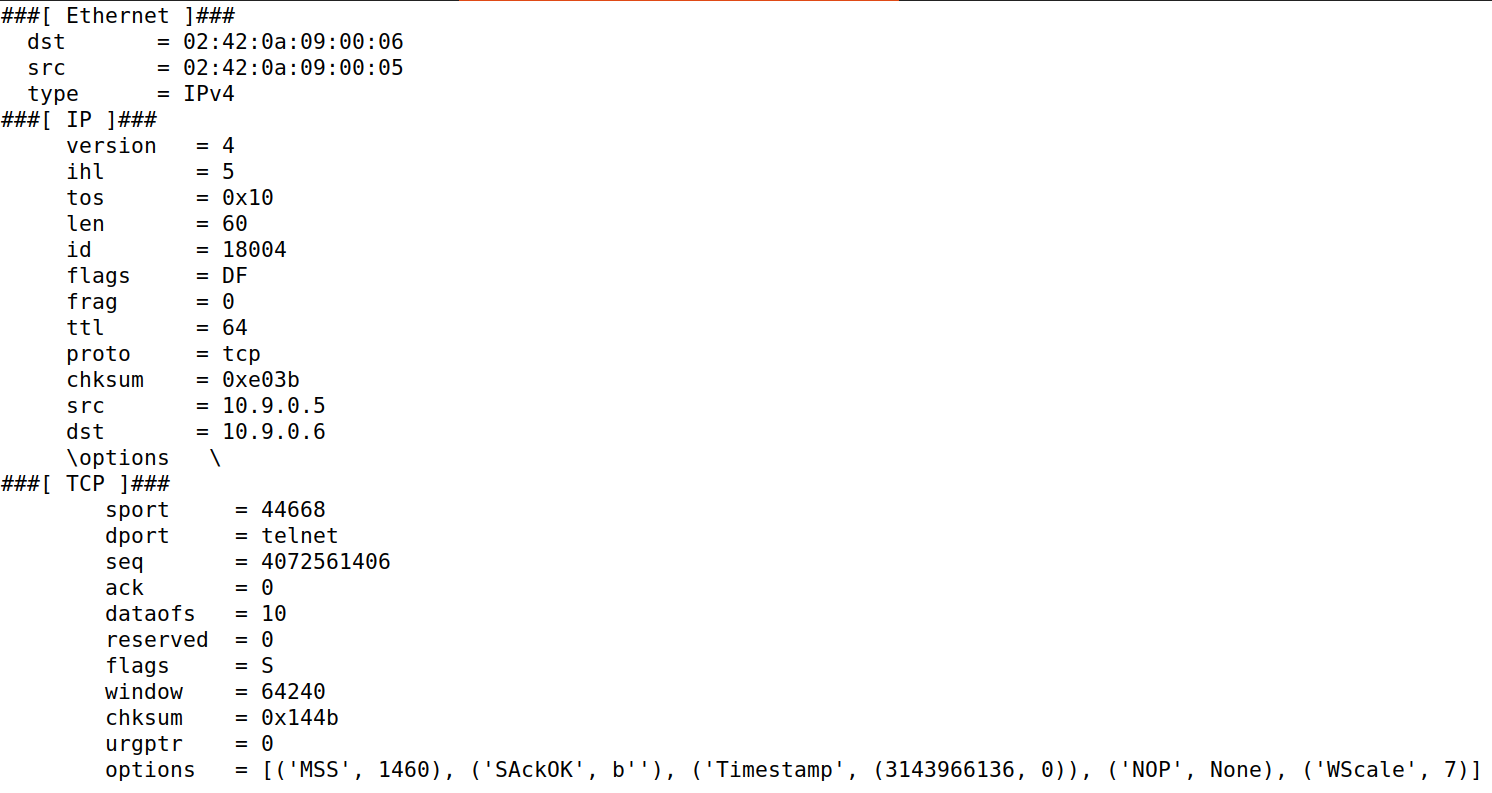
Using the ping command, we can send data from hostA to hostB.



Sniffer.py is run from the attacker and intercepts the ping containing the ICMP.

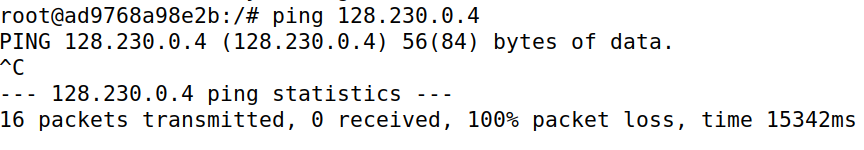


To get the TCP packet through port 23, I used telnet to log into hostB from hostA.

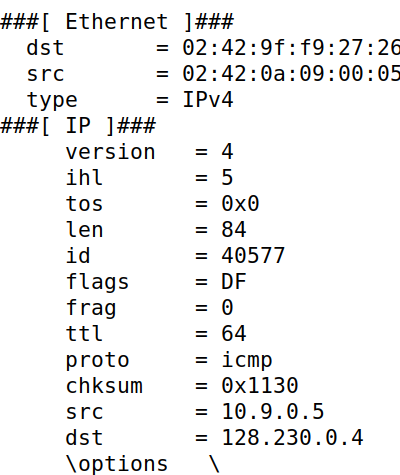


Here is a packet from using telnet. The destination port field says ‘telnet’ which is port 23.

Finally, let’s get a packet from a subnet. I’ll be using the provided subnet 128.230.0.0/16.



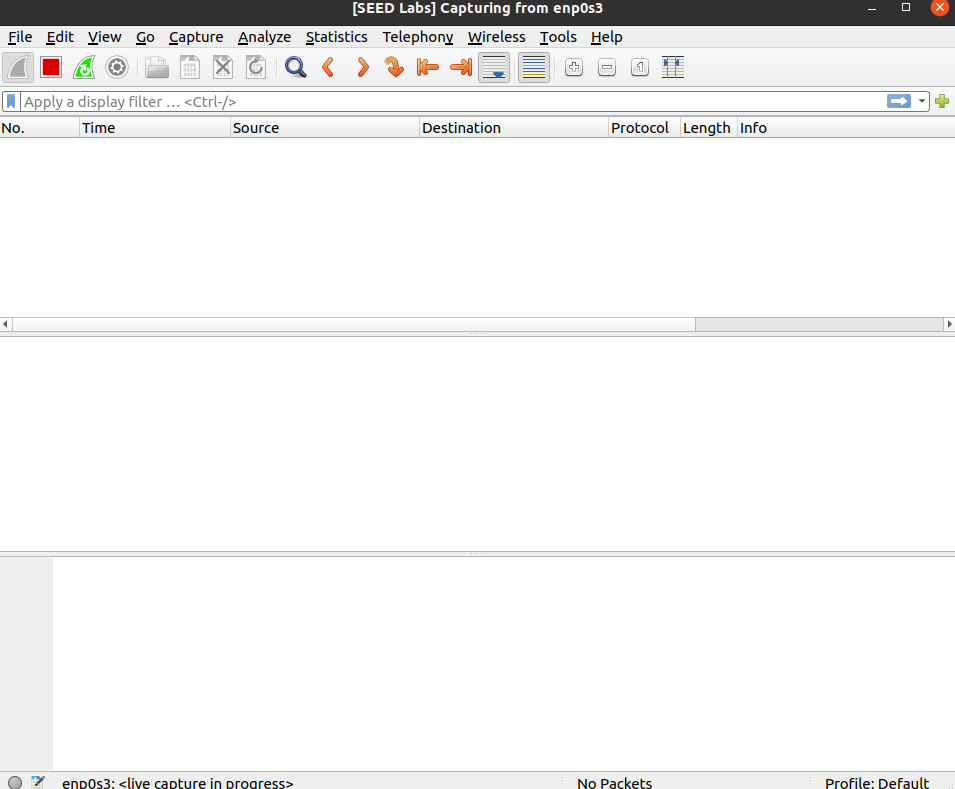
From hostA, I pinged an address within the subnet.



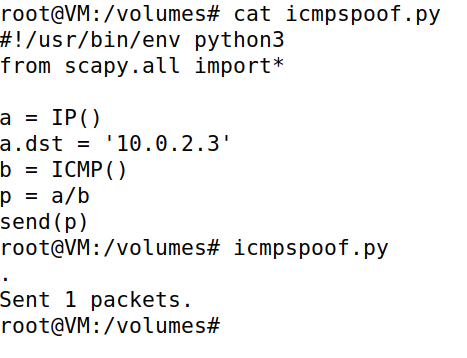
Here’s the packet from the attacker. At the bottom, you can see where the ping came from and went.

# 1.2

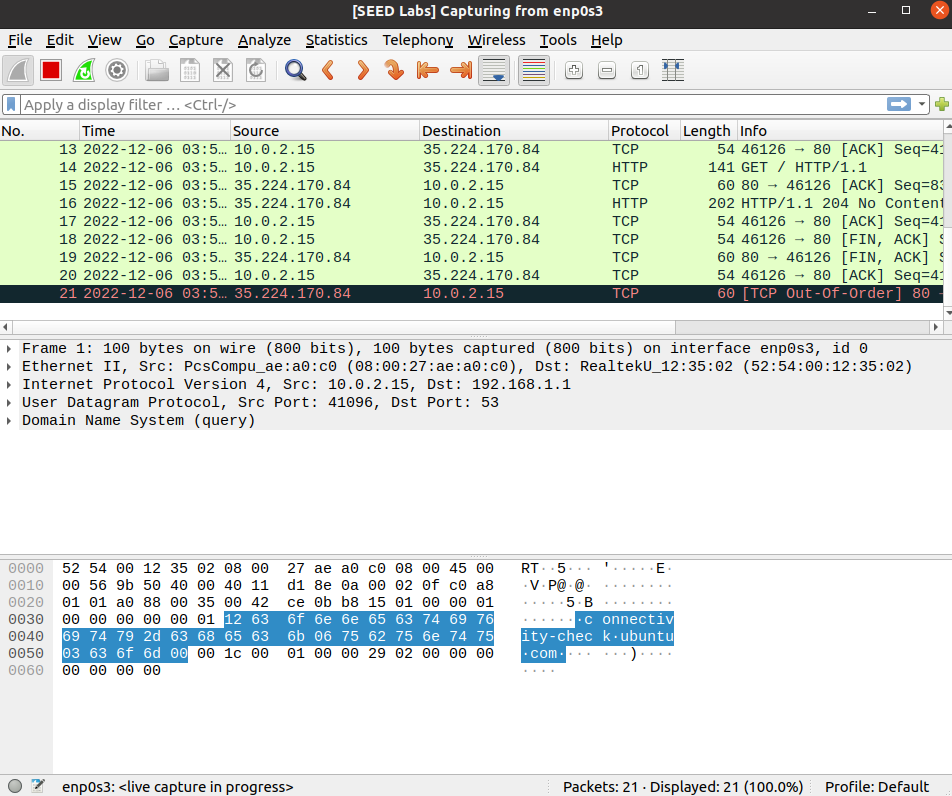
1.2 tasks us with proving that we can use Wireshark to monitor our spoofing. To begin, let’s open Wireshark and start capturing from enp0s3 which is our network.



The lab uses python directly to write the code needed to send the packet. Instead, I created another python script called icmpspoof.py which contains the same code. I kept the sample IP address as well. All that needs to be done now is to send the packet.

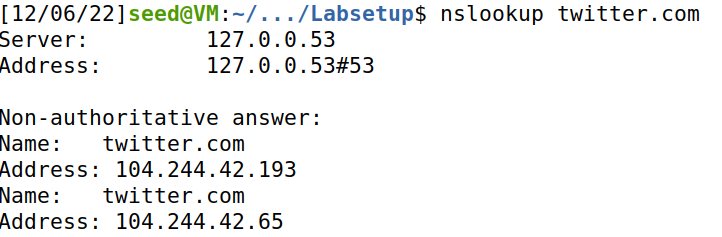


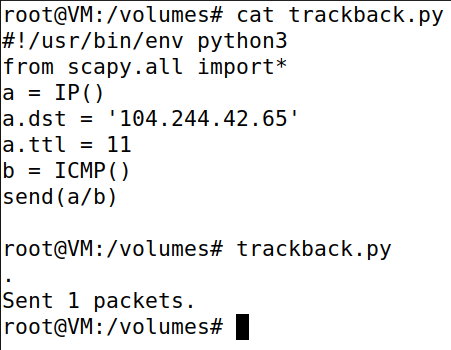
Now let’s look at Wireshark to see our captured packet.

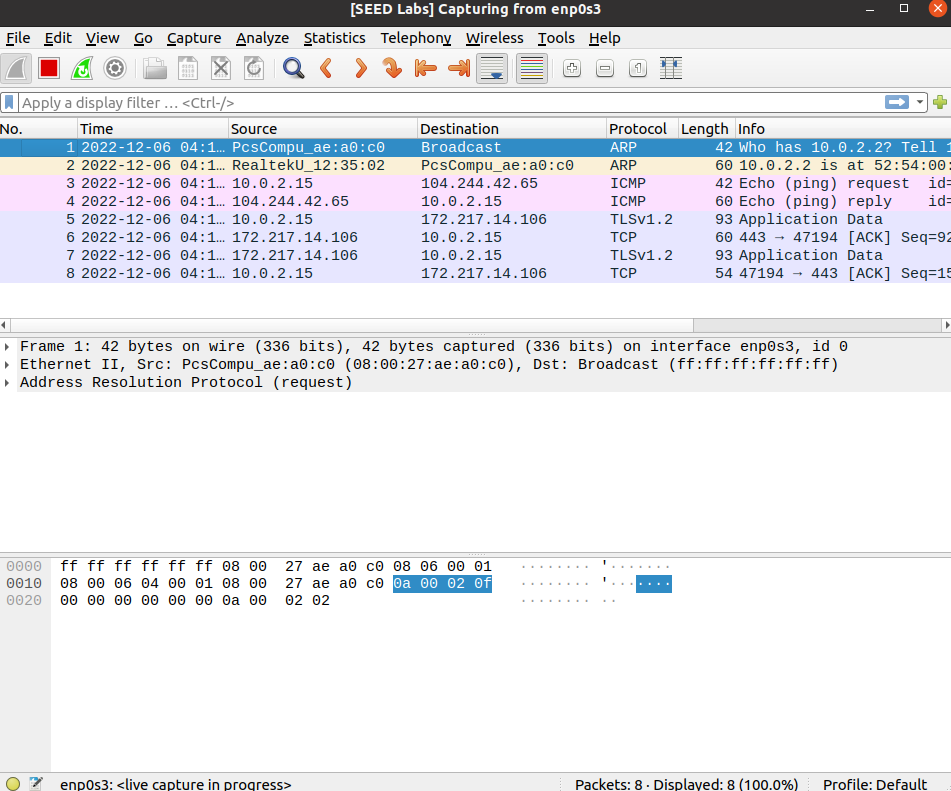


# 1.3

This task replicates the traceroot tool which estimates the distance between our network and a chosen destination in terms of routers. For this, I chose twitter.com.



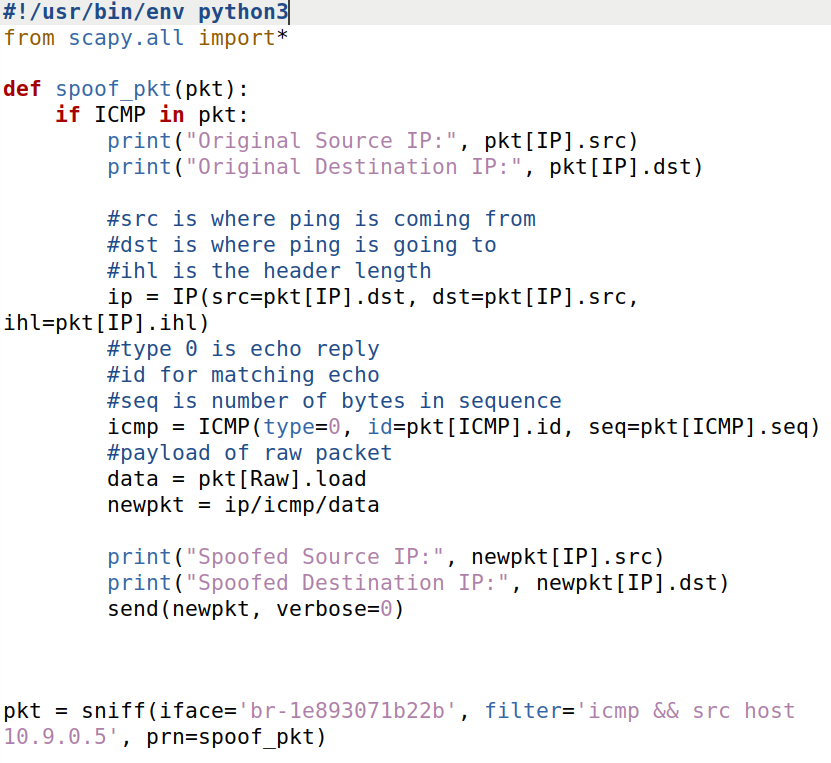




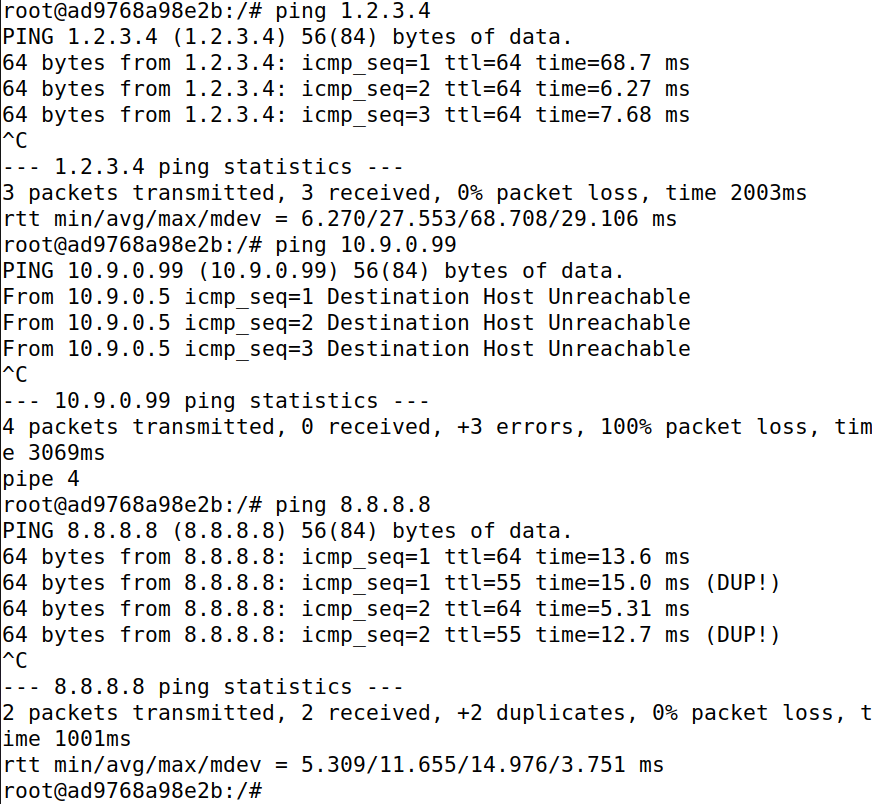
In lines 3 and 4, you can see my VM and twitter responding to each other.

# 1.4

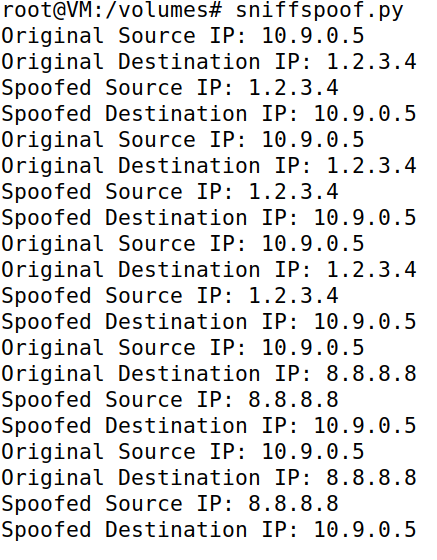
The final task is the most challenging. Using the VM and containers, ping an IP and if it’s alive, print out a response. Here’s my code for it

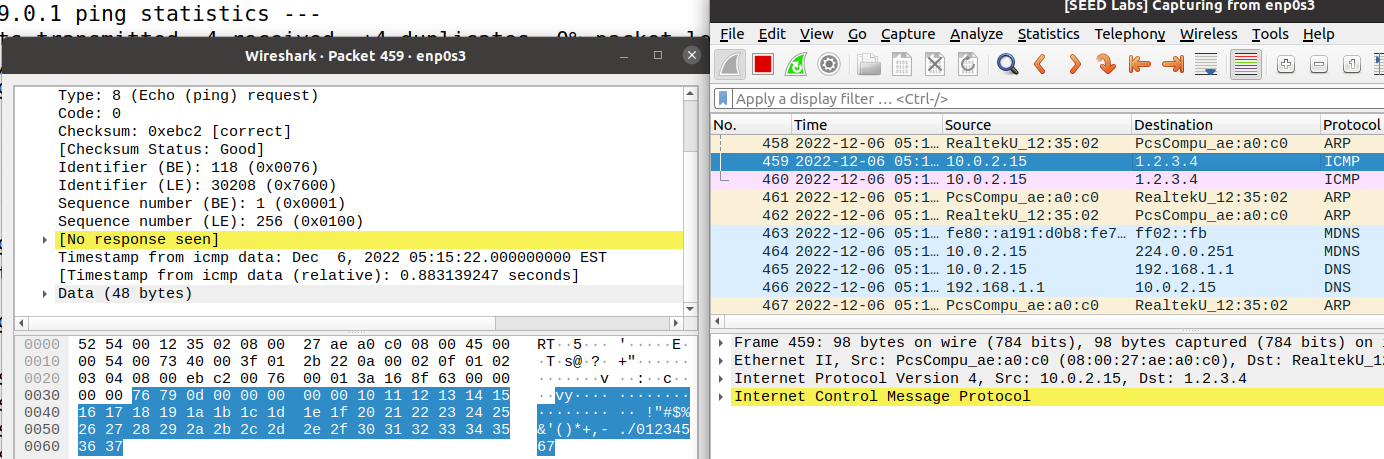


I used the sniffer.py as my starting point for writing the code. I altered the filter field so it recognizes hostA as the source IP. In spoof\_pkt, I added an if statement that only prints out the packet if it has an ICMP. If it does, I print out where it came from and create a new spoofed packet using the old one’s data and print that out too.

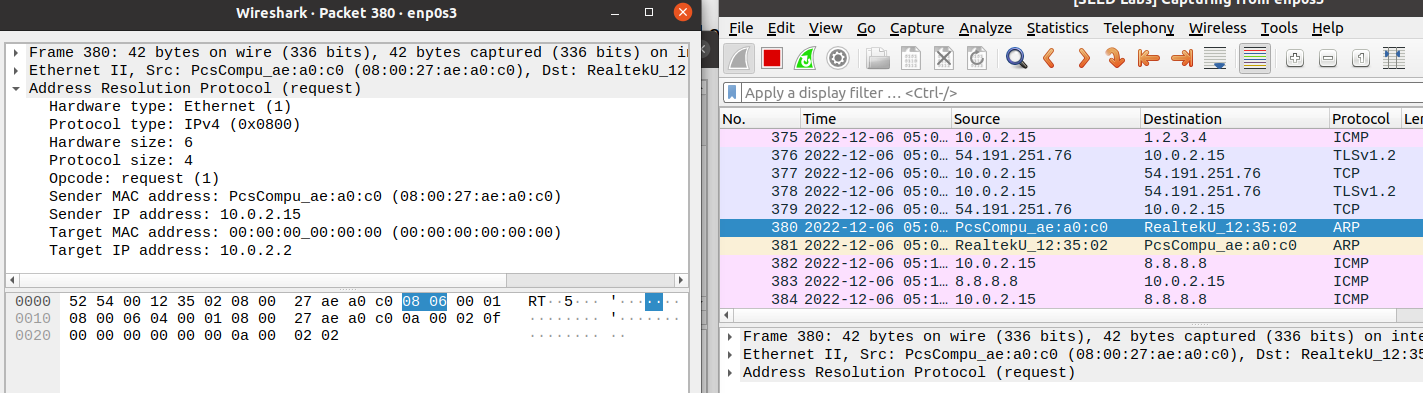


From hostA, I pinged 3 destinations: 1.2.3.4, 10.9.0.99, and 8.8.8.8. All 3 of these are pulled from the instructions. This is the output for the attacker:

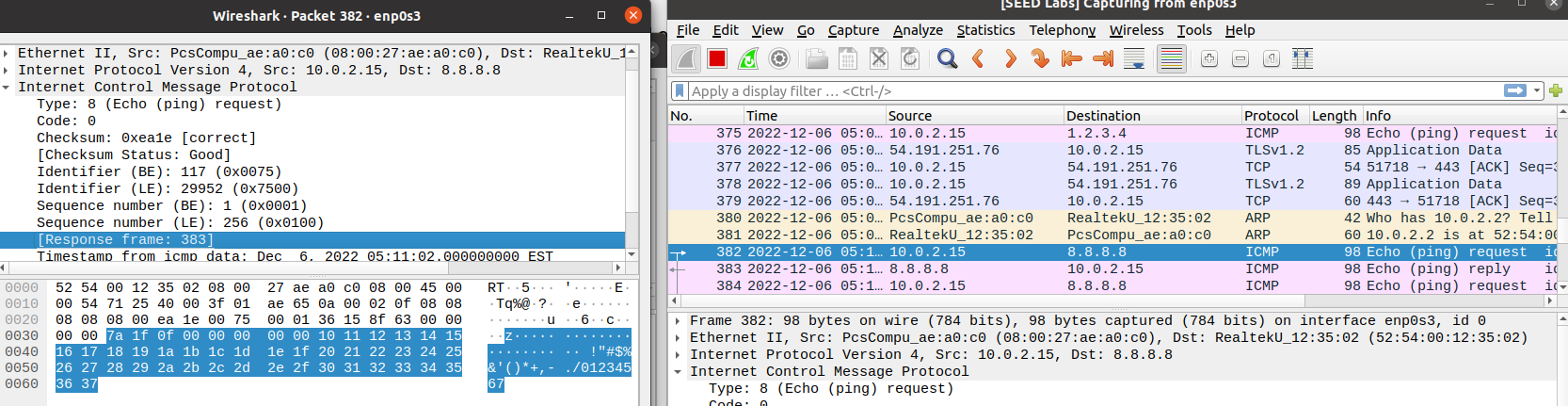




Using Wireshark, we can see the ping to 1.2.3.4 gets no response and sends that back to hostA.



10.9.0.99 gets nothing returned. This is because of ARP. ARP broadcasts a signal across the LAN asking for the given IP address. Since no one responds due to it not existing, nothing gets sent back. This is unlike ICMP with 1.2.3.4 who sent back whatever information it got before it died out.



8.8.8.8 has a slightly different response than 1.2.3.4 since it actually exists. Instead of [No response seen], we get the target’s MAC and IP addresses.