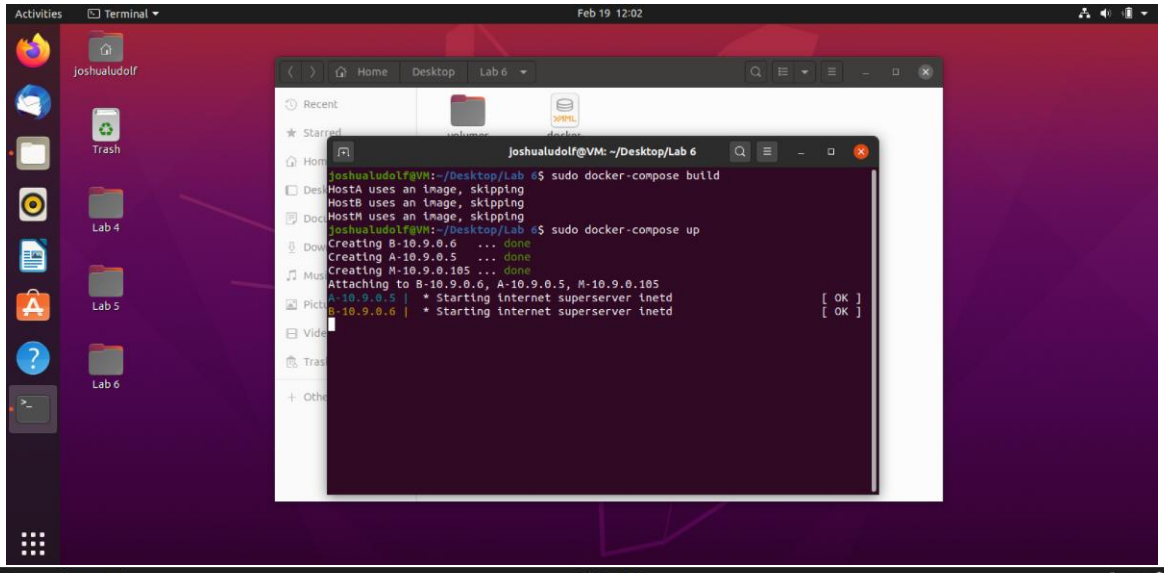
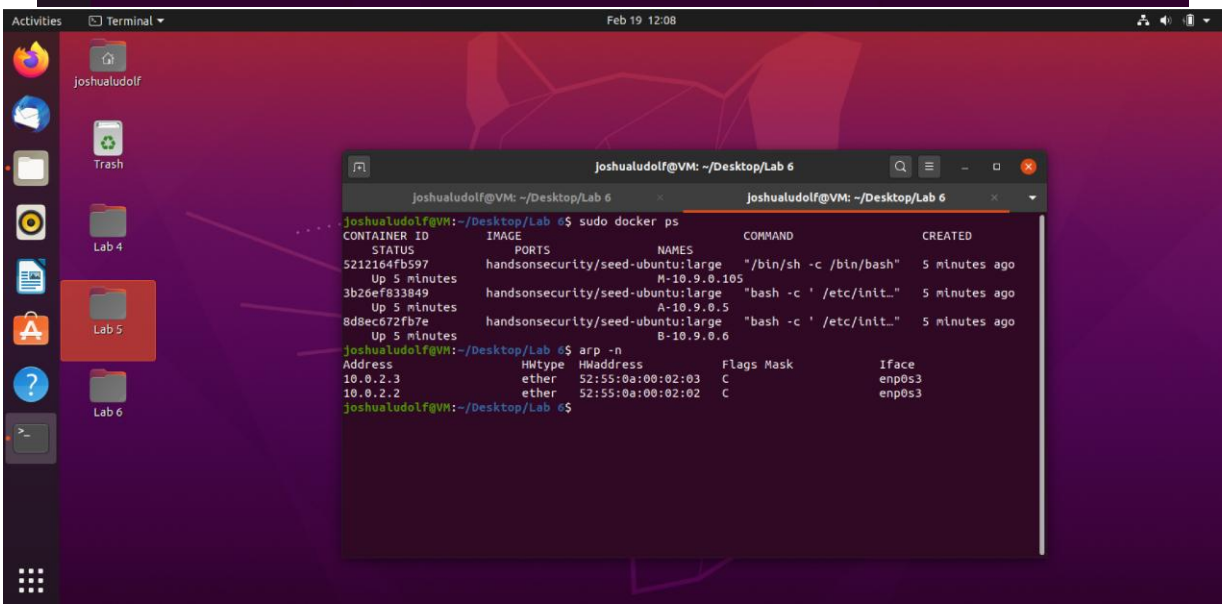
Lab 6: ARP Cache Poisoning Attack Lab

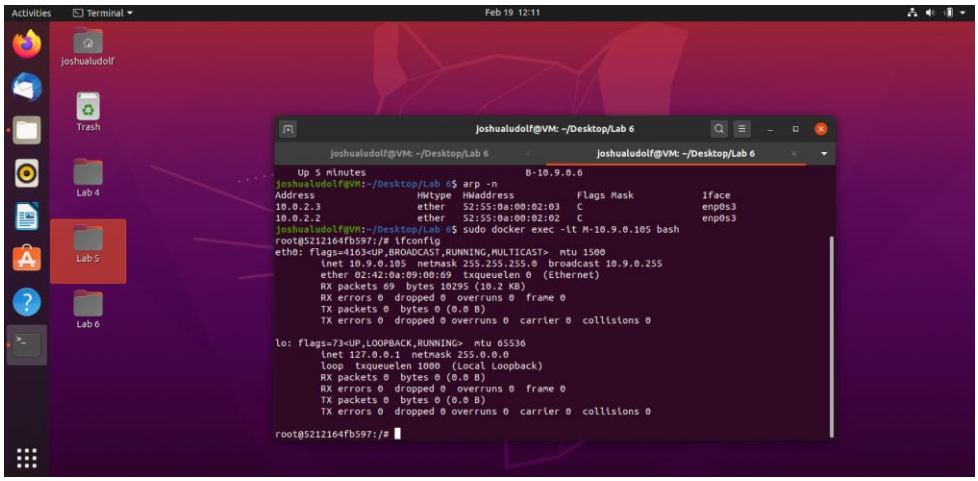
Joshua Ludolf
CSCI 4321
Computer Security

- **To start this lab, I needed to rebuild the docker container and get the host's name(s)/ip addresses:**

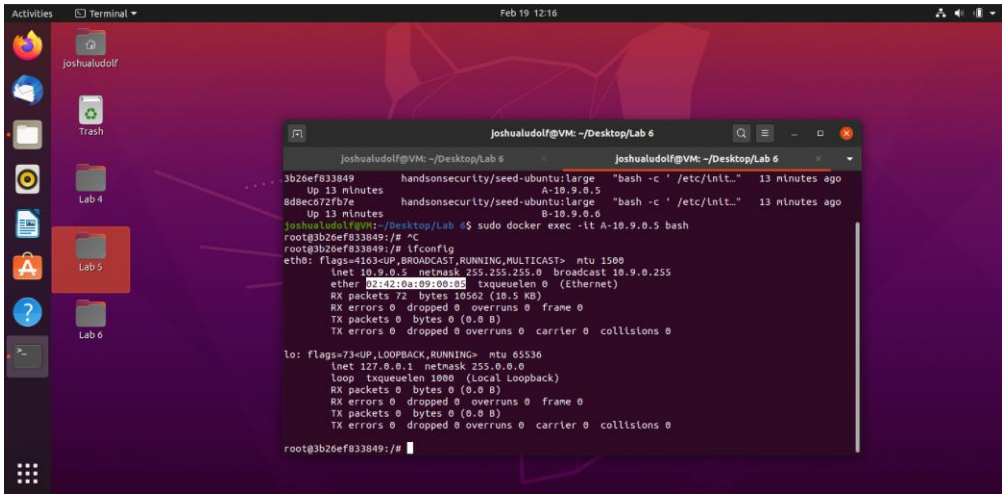




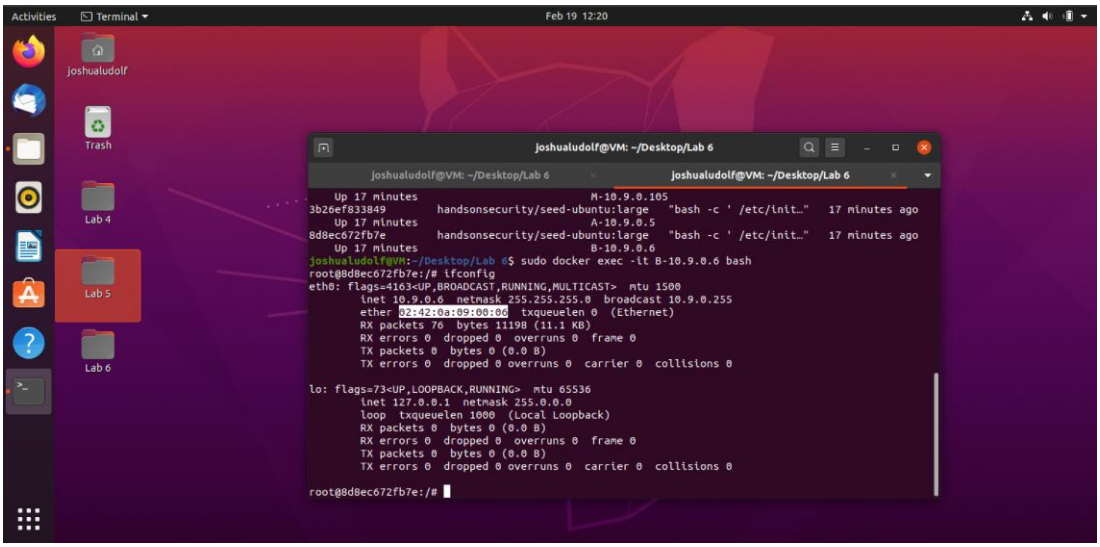
- **From there, I logged into each of them and get their IP and MAC addresses (ether:…):**
  - **For M-10.9.0.105 its mac address was 02:42:0a:09:00:69 (command to login – sudo docker exec -it M-10.9.0.255 bash and to get IP and MAC addresses – ifconfig):**



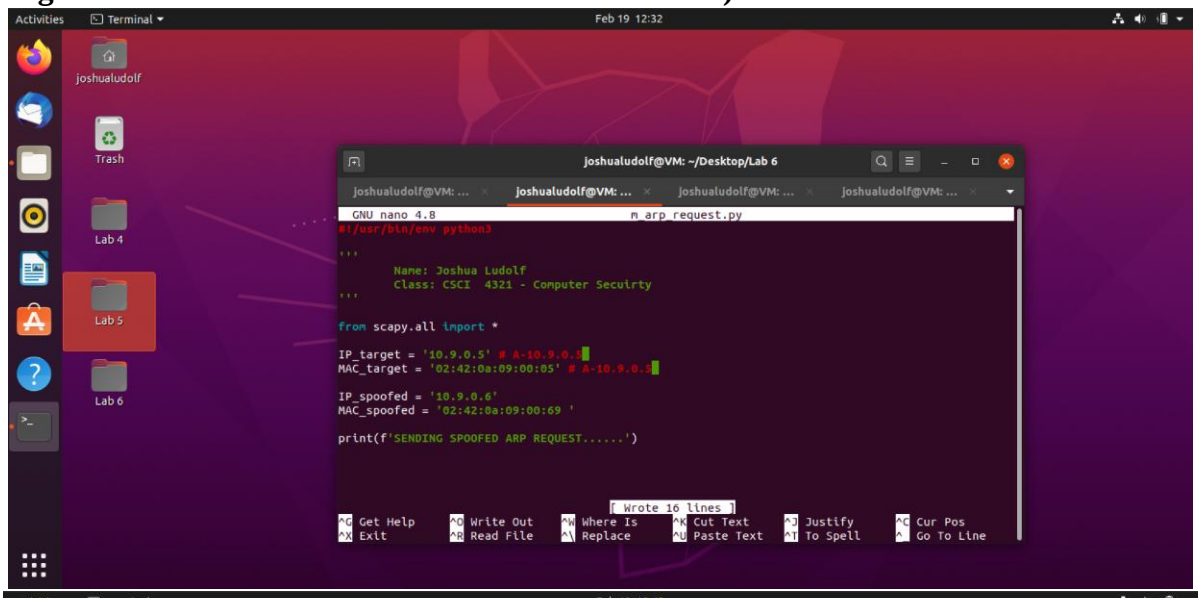
- **For A-10.9.0.5 its mac address was 02:42:0a:09:00:05:**

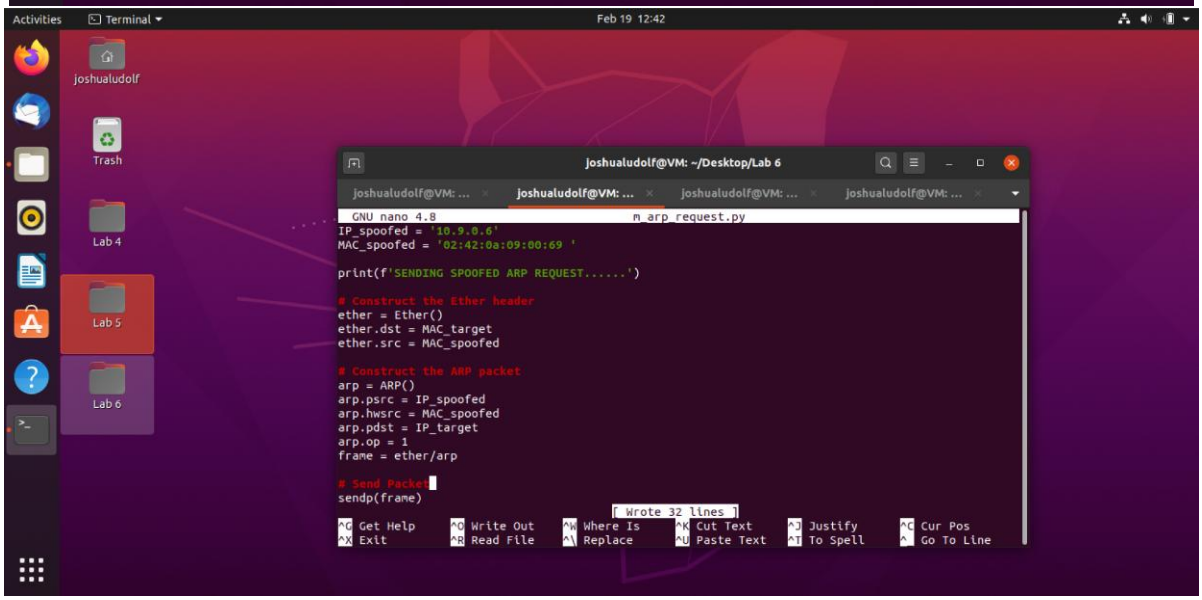


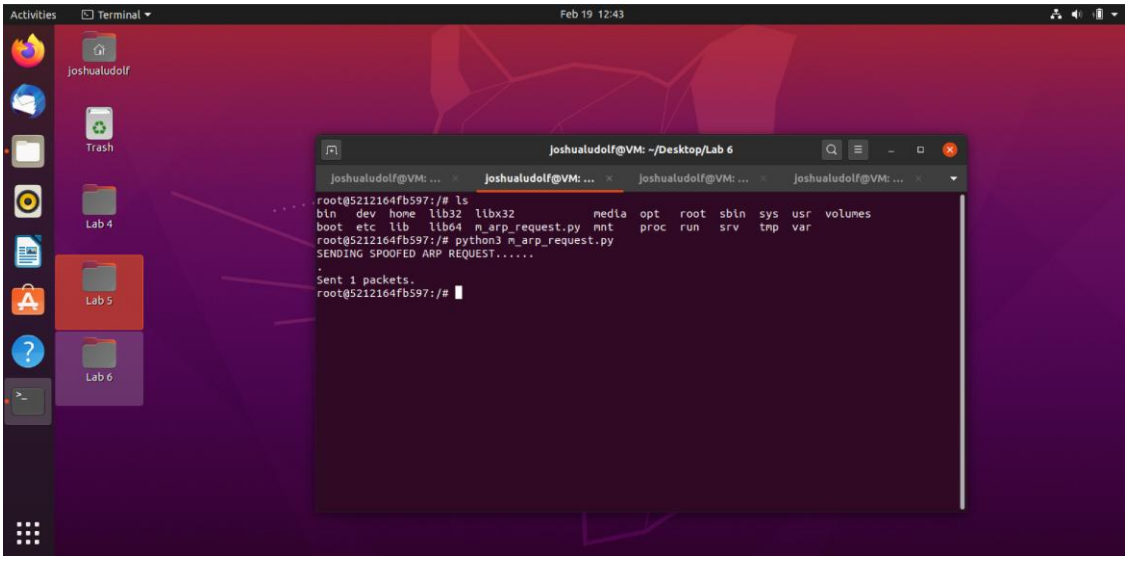
- **For B-10.9.0.6 its MAC address was 02:42:0a:09:00:06 :**



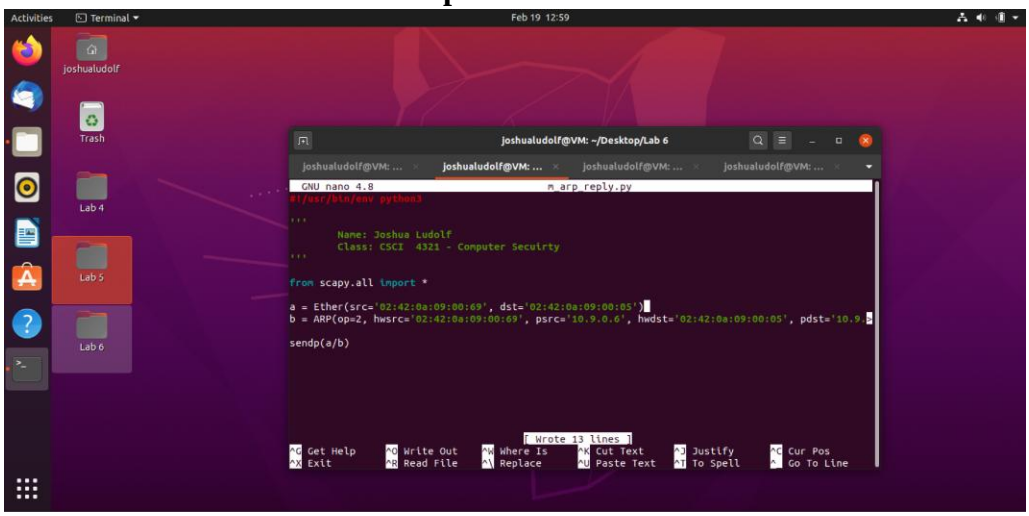
- **From there I reopened all of them as task 1.a was to construct an ARP request packet to map B's IP address to M's MAC address. Send the packet to A and check whether the attack is successful or not – which it was it sent the wrong ARP record (also same result as doing in the docker container or in the vm terminals…):**



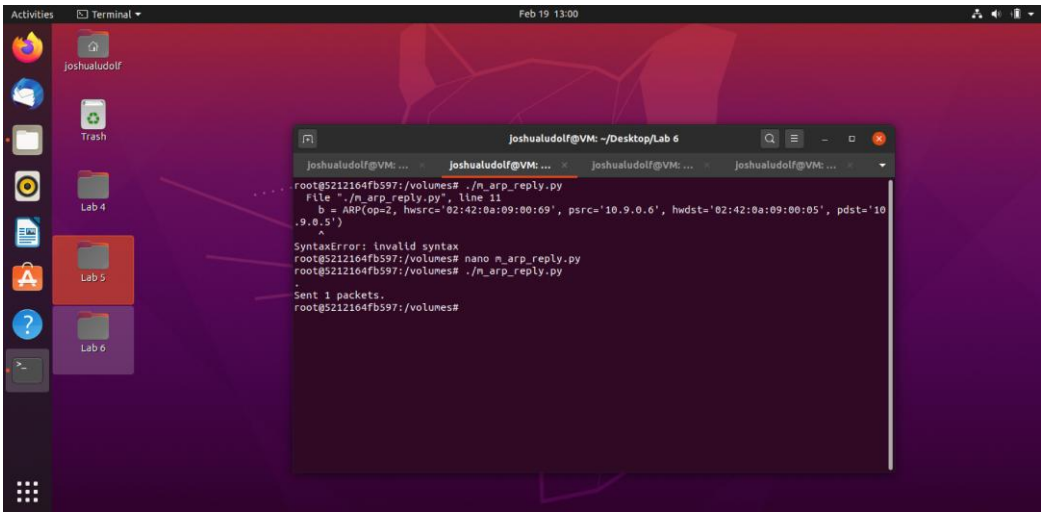


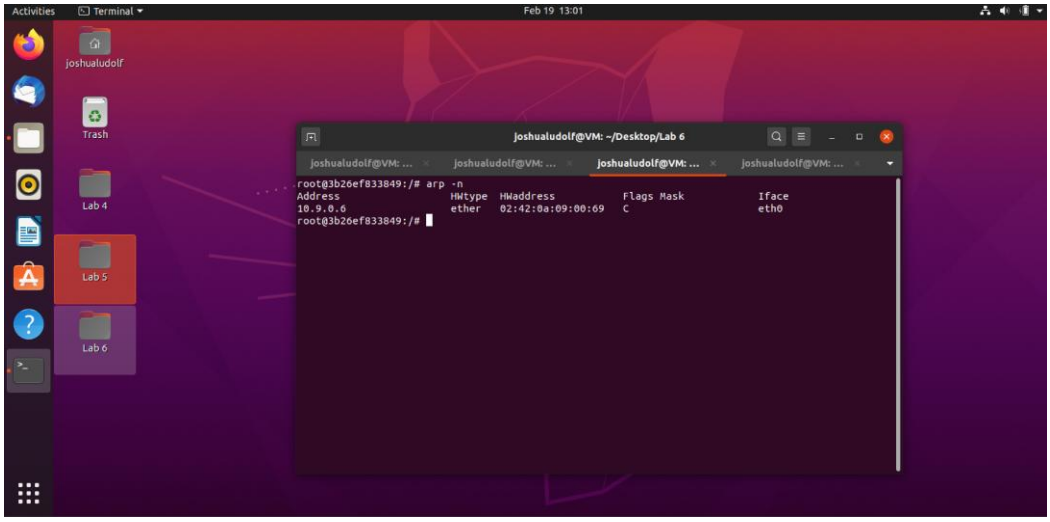


- **For task 1.b, we do the same thing but with ARP reply packet (by changing op=2 instead of op=1) for two scenarios: first with the arp cache and the second without:**

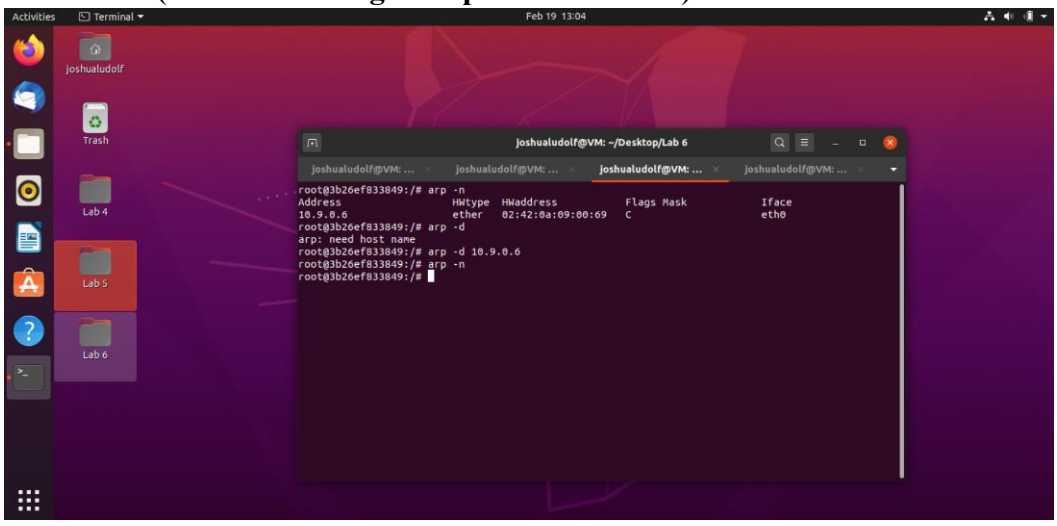


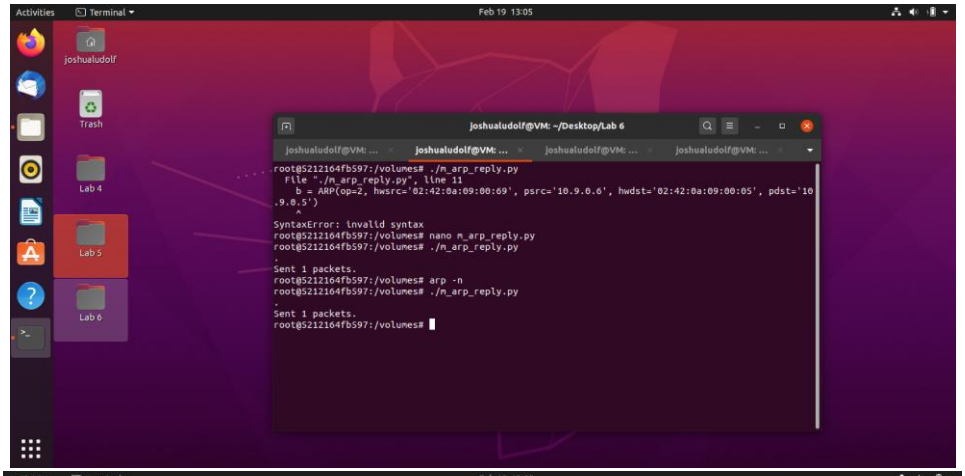
  - **For scenario 1 (which I got output in A's cache 😎):**

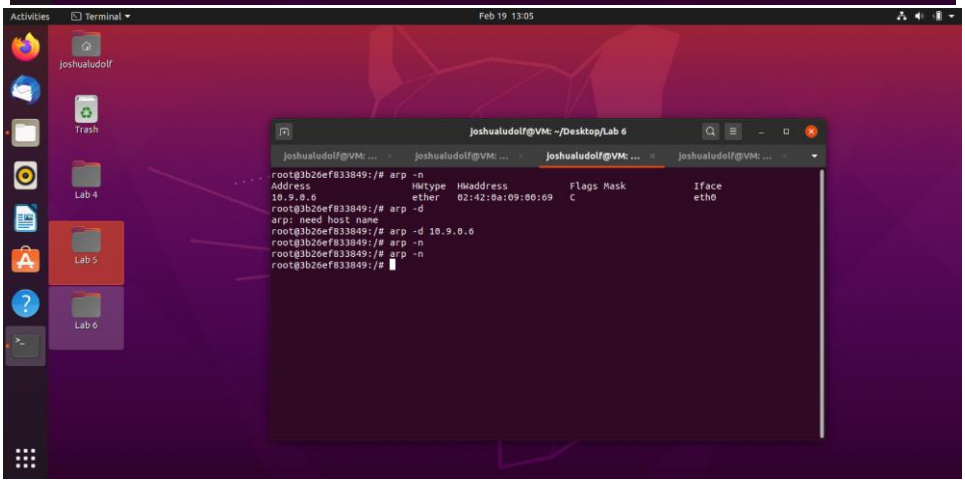




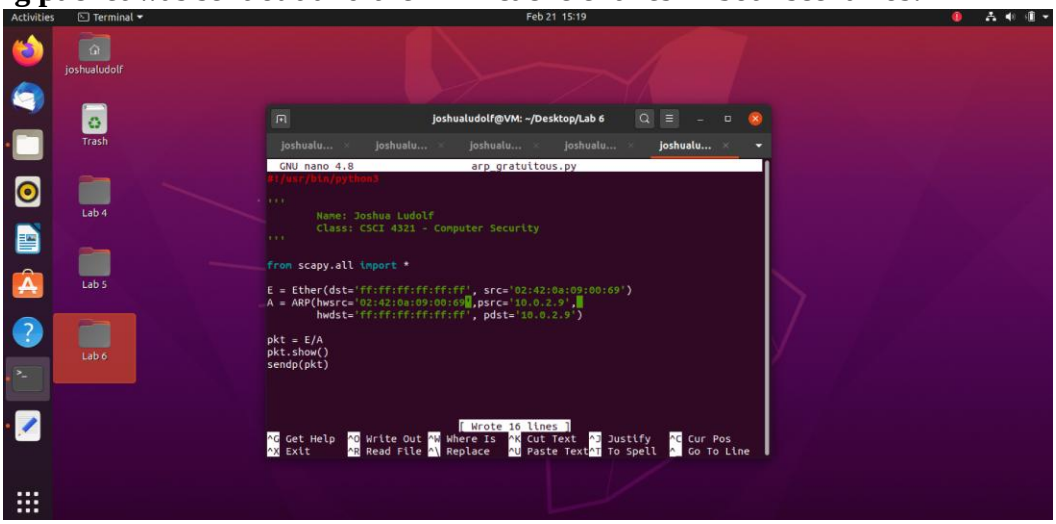
- **For scenario 2 (which I didn't get output in A's cache):**

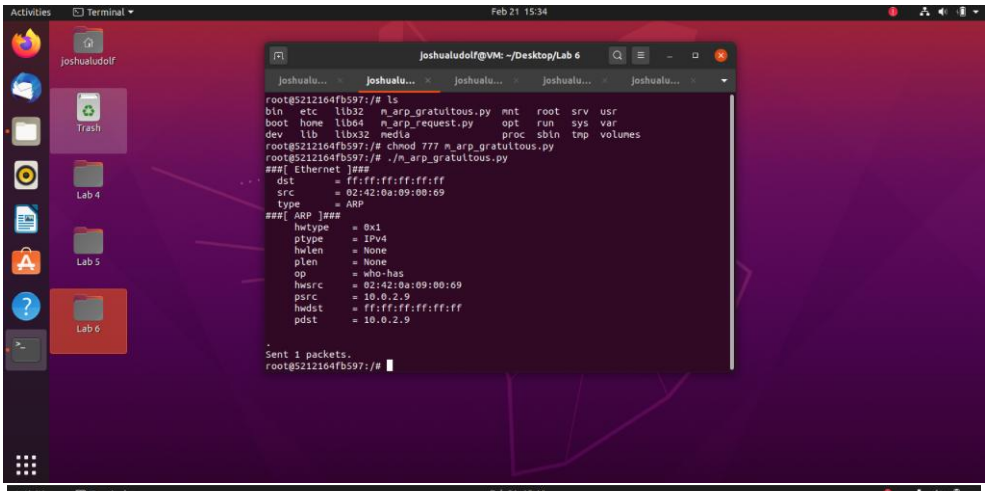


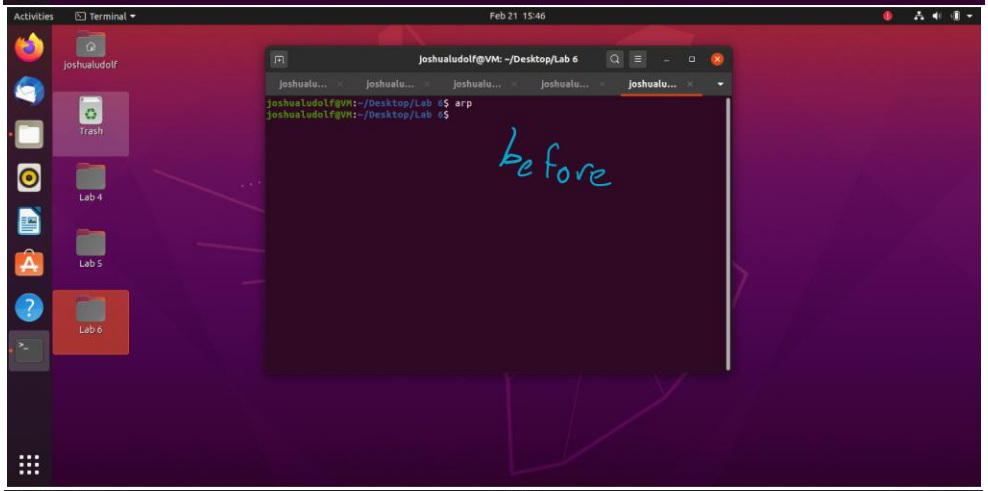


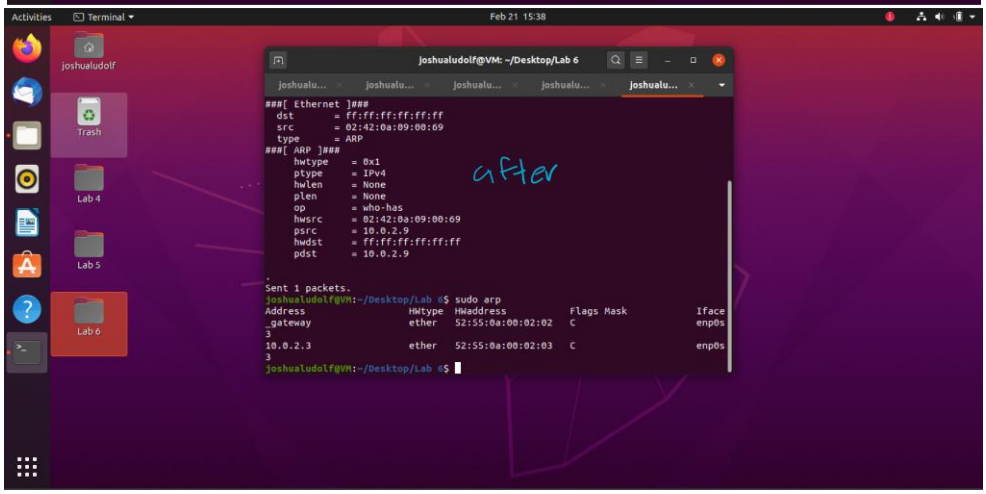


- **For task 1.c, I created arp_gratuitous.py as I wanted to construct an ARP gratuitous packet on host M, and use it to map B's IP address to M's MAC address, and we see the following packet was sent out and the ARP cache entries in both scenarios:**



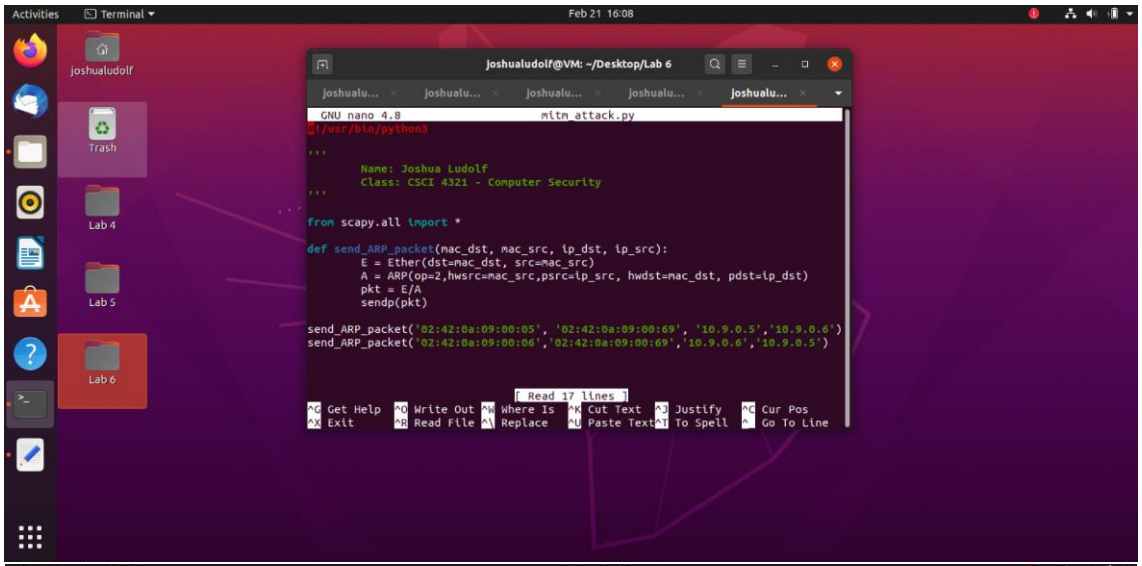


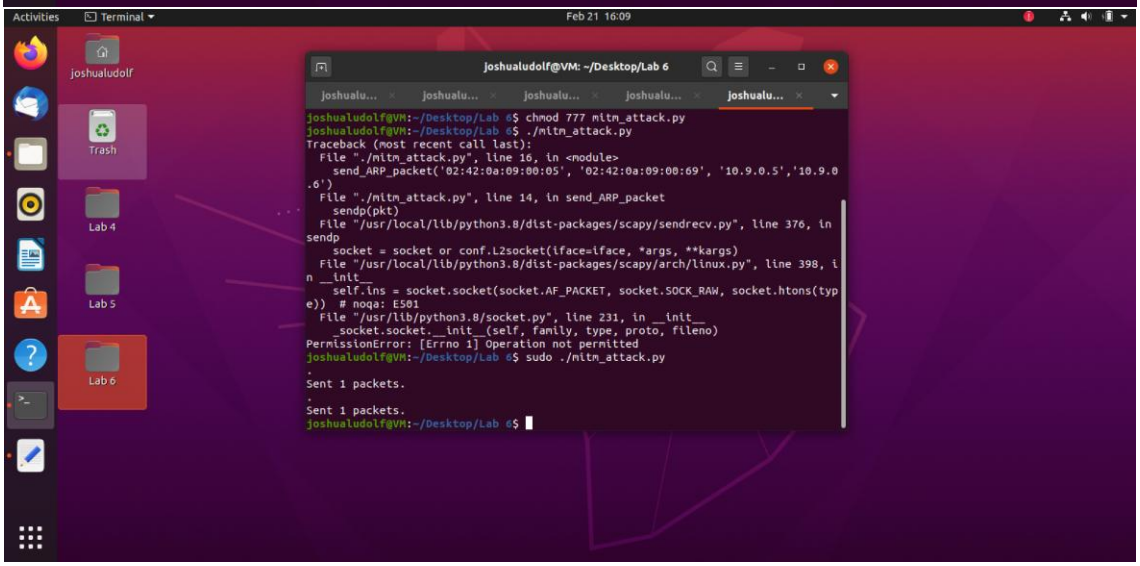


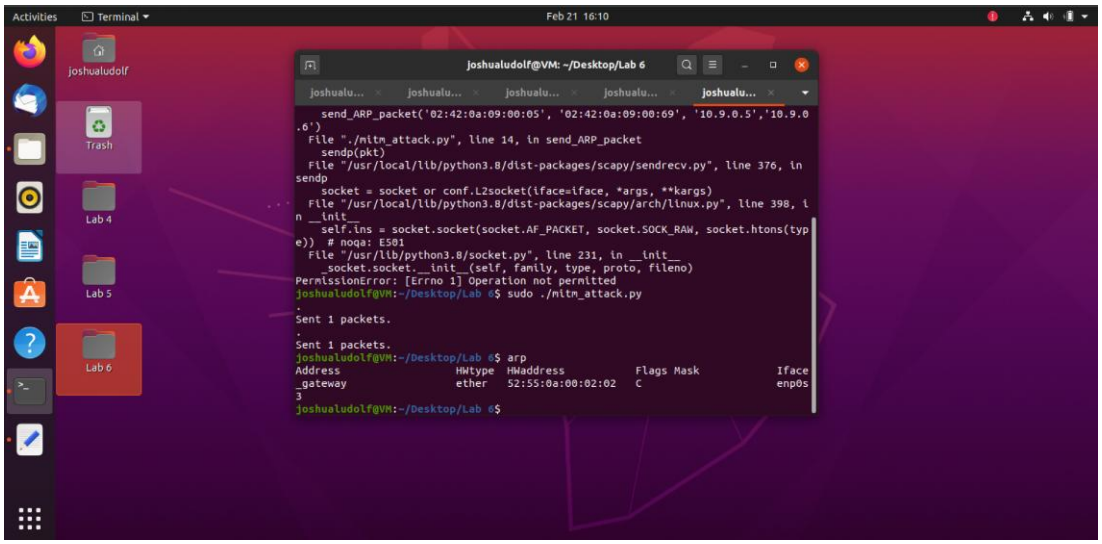


- **In the above output, we see that only A's ARC Cache changes and even though B received the packet (due to the packet being broadcast on the network), B's ARP Cache remains unchanged. This was because the sender's IP address matches B's IP address and hence B assumes that the packet was sent by it. The ARP Cache only consists of those IP addresses that doesn't belong to the host.**

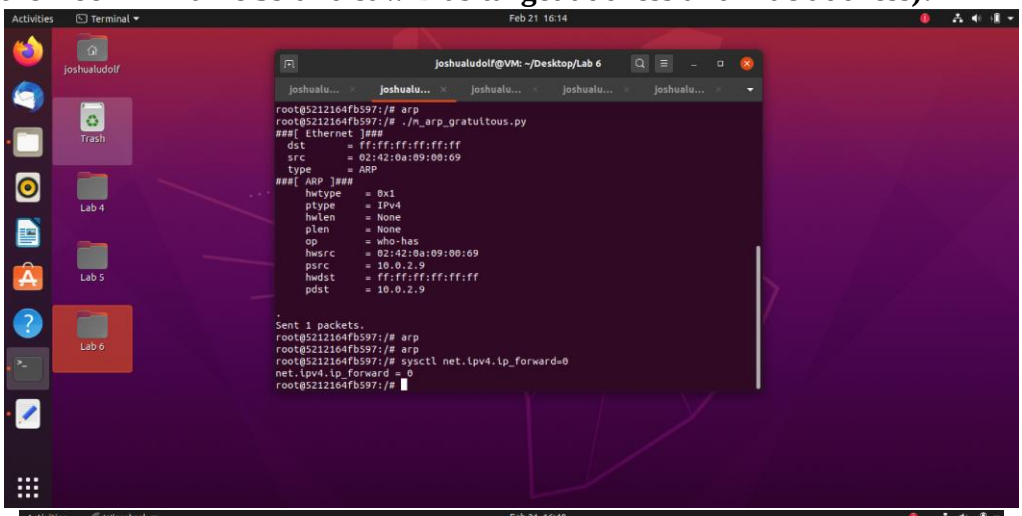
❖ **From there, I constructed a python script to launch MITM on Telnet using the ARP Cache Poisoning:**

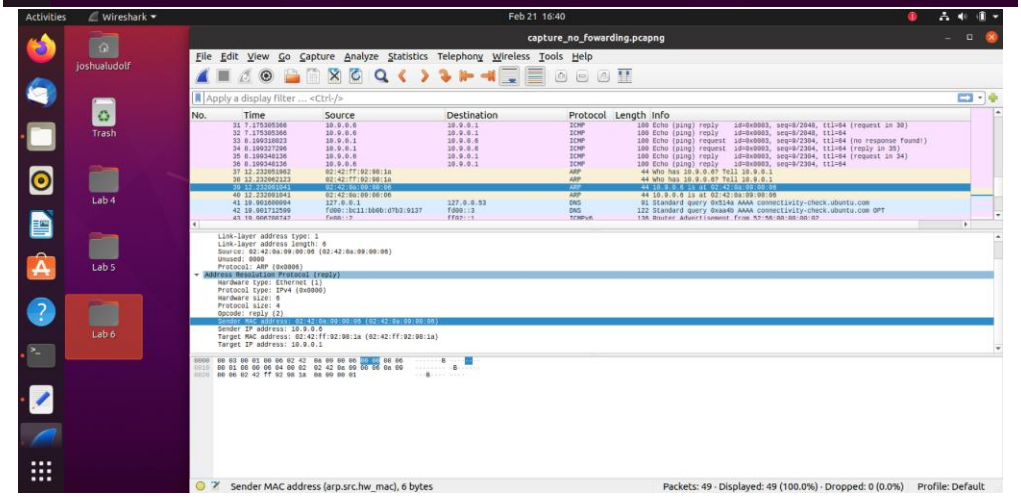




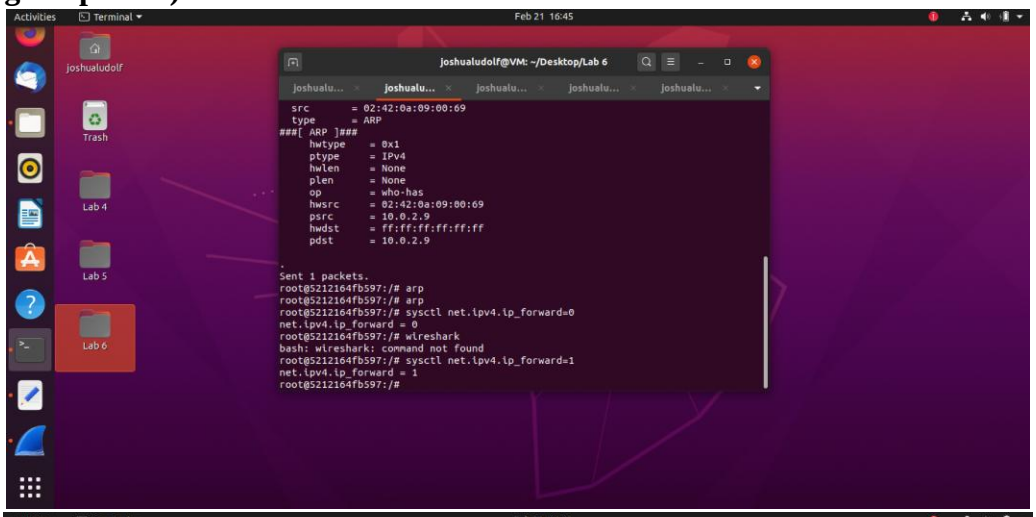


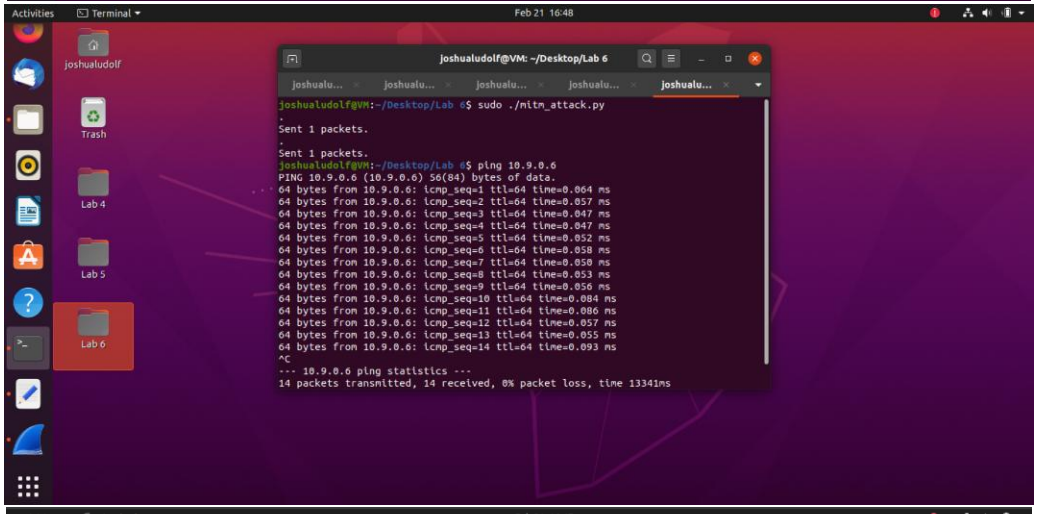
❖ **After that I turned off port forwarding from Host M for part 2 of task 2 to observe pinging from Host A and Host B and noticed in frames 37 – 40, was starting to get weird frames (I took further look in frame 39 and saw B as target address and mac address):**

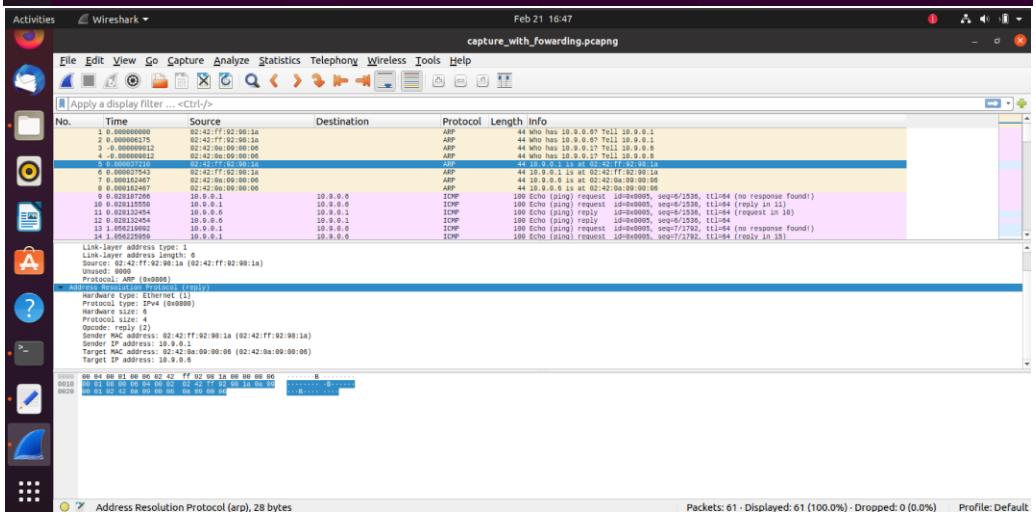




- **Next, I needed to observe similarly but with port forwarding enabled (this time, it echos as we see duplicate of some of the frames, basically meaning M forwards the packet instead of dropping the packet):**

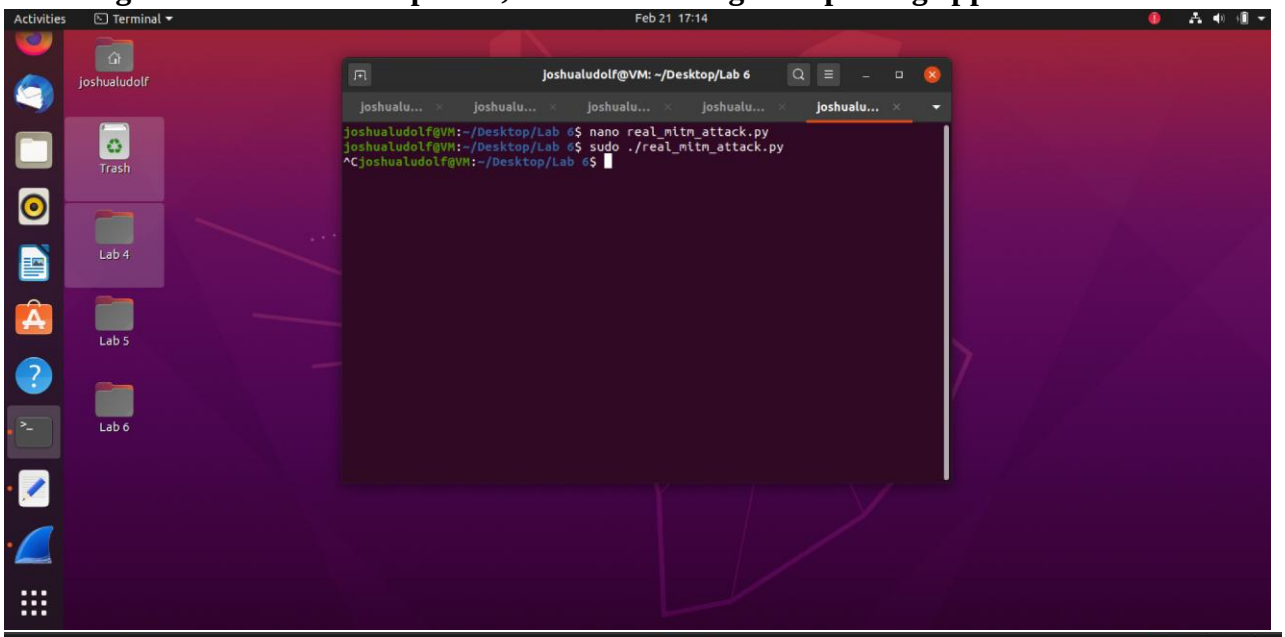


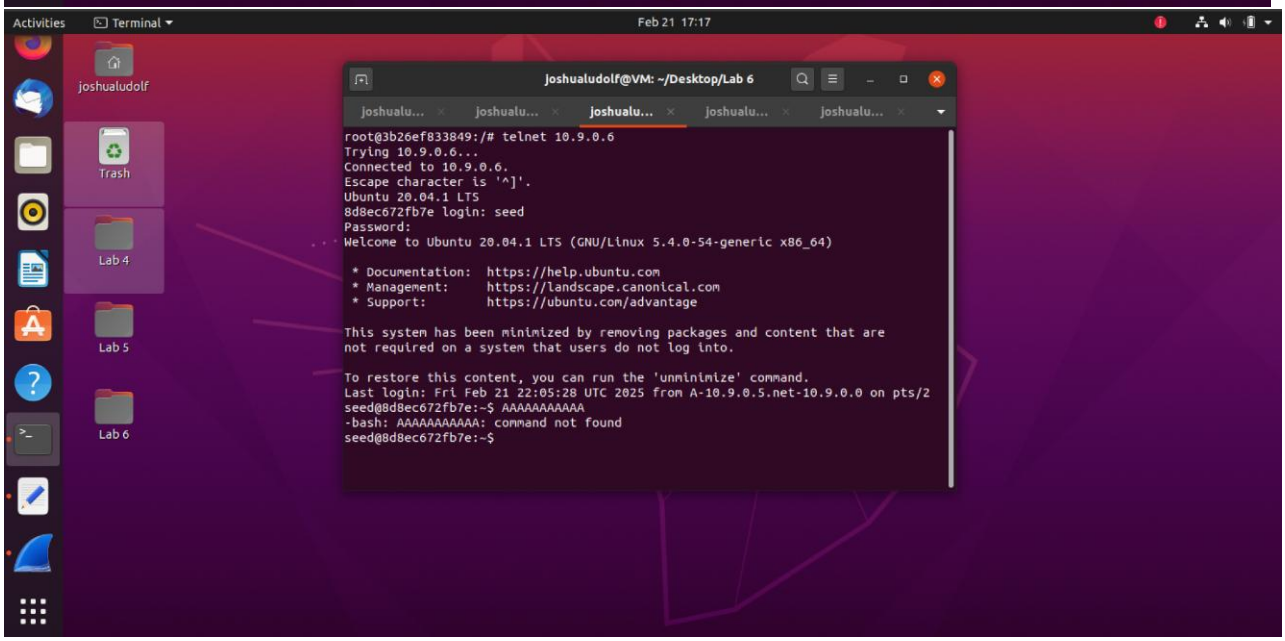


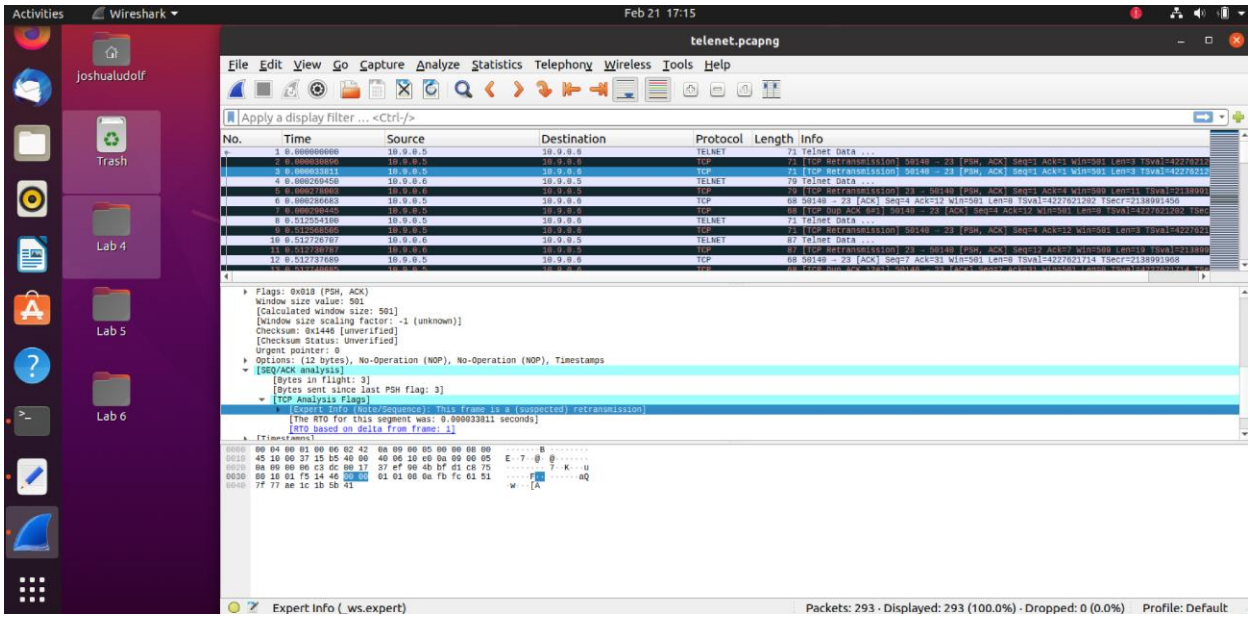


- **Finally, I created python script named – real_mitm_attack.py, like mitm_attack.py performing the ARP cache poisoning, however this time after connection is established**

between A and B, I turn off the IP port forwarding so that I could manipulate the packet. To change the contents of the packet, I use the sniffing and spoofing approach:







❖ What I learned from this lab:

In this lab, I focused on understanding and executing ARP spoofing attacks. Through tasks 1 and 2, I learned how to manipulate ARP tables to intercept and modify network traffic between two virtual machines. Task 1 involved setting up the environment and ensuring that ARP packets could be sent and received correctly. Task 2 required me to implement the ARP Cache Poisoning attack, allowing me to see firsthand how an attacker can position themselves as a man-in-the-middle. This experience highlighted the importance of network security measures to prevent such attacks. Overall, these tasks provided valuable insights into the mechanics of ARP spoofing and the potential vulnerabilities in network protocols.