Python Libraries Used

- scapy - as per the professor's recommendation, used for sending and receiving packets
- netaddr - used for getting all the ip addresses in a subnet
- python_arptable - used for getting the arp table

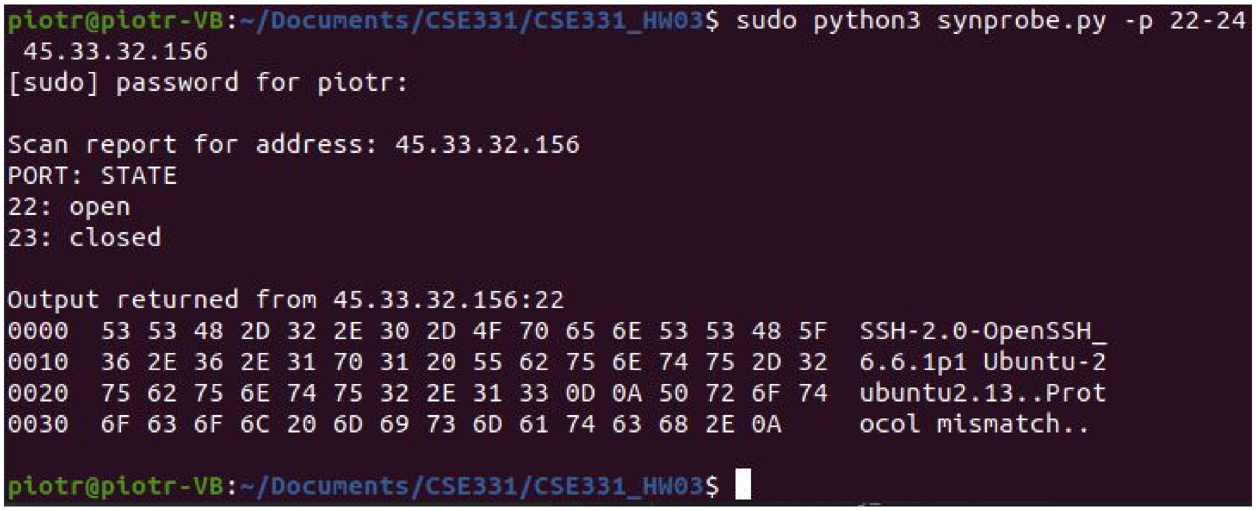
How to Run

- For arpwatch.py use the following format "sudo python3 arpwatch.py [-i interface]"
- For synprobe.py use the following format "sudo python3 synprobe.py [-p port_range] target" where the port_range is in the format of "a" or "a,b,c" or "a-d". The port_range "a-d" will be inclusive of 'a' and exclusive of 'd' so it is the same as "a,b,c"

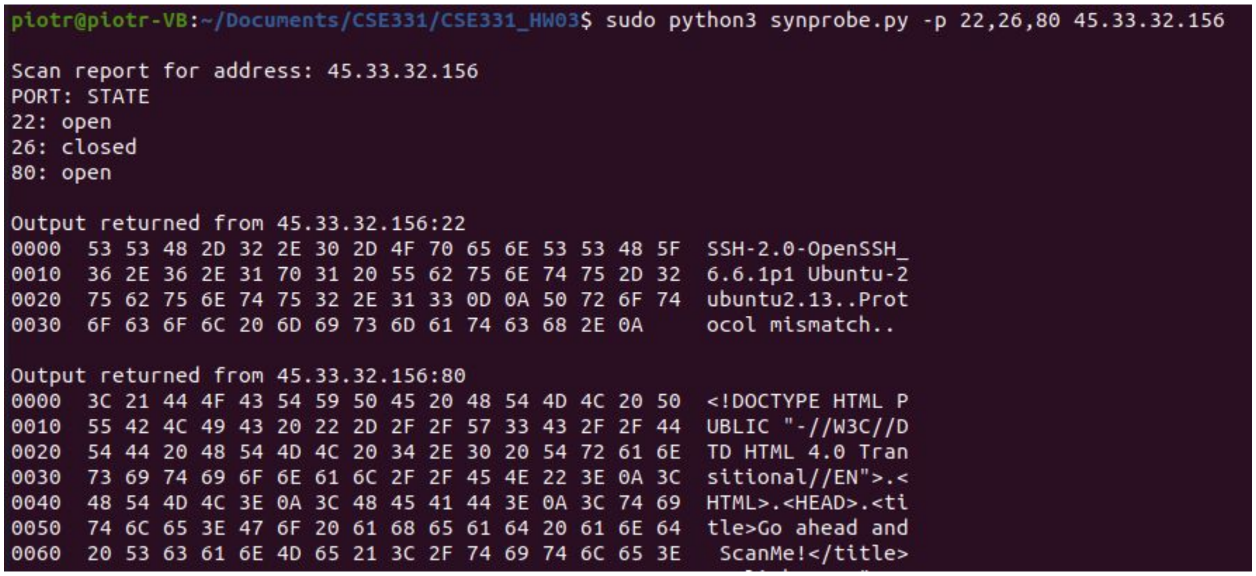
How I accomplished synprove.py

I started by looking through the scapy usage documentation located here: https://scapy.readthedocs.io/en/latest/usage.html and looked at the methodology that they used for doing a syn scan. Next I did argument processing in my main function and once that was done I would check if the user was requesting a scan for a subnet and if so then I would loop through all the IPs in the specific subnet which I would get from the IPNetwork function from netaddr which I decided to do after reading this post on stack overflow https://stackoverflow.com/questions/1942160/python-3-create-a-list-of-possible-ip-addresses-from-a-cidr-notation . Then for each ip I would call a function called checkIP which I would also call if only 1 ip was specified. checkIP's job is to first check if the target ip is running by sending a single packet to the ip and checking if a reply was received. Then if it is running for each port in the ip I will check the port in that ip by sending a syn packet and checking if the response was a TCP response and if the flag inside was set to 0x12 which I determined by scanning an open port and seeing what the flag value was set to. Then I will print out all the ports and their states and then run printResp() for each open port to try and elicit a response from the server by connecting to the server and sending it 2 messages and listening for a response. The messages were initially different but afterwards I set them to the same message which is GET but with different timeouts so in the case of the first packet being missed or not having enough time to reply a second packet will be sent with a longer time to wait for a response. Then I just hexdump'd the response into the terminal as per the professor's recommendation.

Example output for synprobe.py:



```
piotr@piotr-VB:~/Documents/CSE331/CSE331_HW03$ sudo python3 synprobe.py -p 22-24
 45.33.32.156
[sudo] password for piotr:

Scan report for address: 45.33.32.156
PORT: STATE
22: open
23: closed

Output returned from 45.33.32.156:22
0000  53 53 48 2D 32 2E 30 2D 4F 70 65 6E 53 53 48 5F  SSH-2.0-OpenSSH_
0010  36 2E 36 2E 31 70 31 20 55 62 75 6E 74 75 2D 32  6.6.1p1 Ubuntu-2
0020  75 62 75 6E 74 75 32 2E 31 33 0D 0A 50 72 6F 74  ubuntu2.13..Prot
0030  6F 63 6F 6C 20 6D 69 73 6D 61 74 63 68 2E 0A     ocol mismatch..

piotr@piotr-VB:~/Documents/CSE331/CSE331_HW03$ 
```

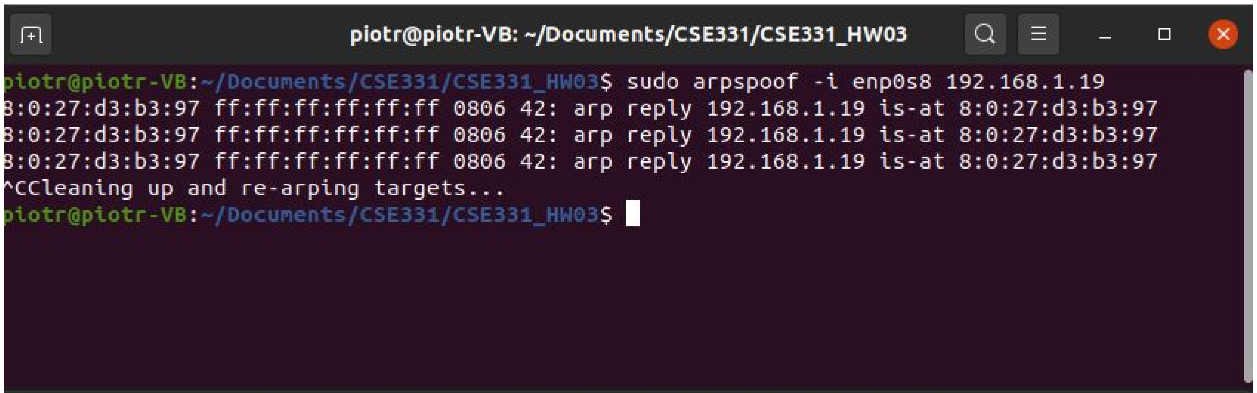


```
piotr@piotr-VB:~/Documents/CSE331/CSE331_HW03$ sudo python3 synprobe.py -p 22,26,80 45.33.32.156

Scan report for address: 45.33.32.156
PORT: STATE
22: open
26: closed
80: open

Output returned from 45.33.32.156:22
0000  53 53 48 2D 32 2E 30 2D 4F 70 65 6E 53 53 48 5F  SSH-2.0-OpenSSH_
0010  36 2E 36 2E 31 70 31 20 55 62 75 6E 74 75 2D 32  6.6.1p1 Ubuntu-2
0020  75 62 75 6E 74 75 32 2E 31 33 0D 0A 50 72 6F 74  ubuntu2.13..Prot
0030  6F 63 6F 6C 20 6D 69 73 6D 61 74 63 68 2E 0A     ocol mismatch..

Output returned from 45.33.32.156:80
0000  3C 21 44 4F 43 54 59 50 45 20 48 54 4D 4C 20 50  <!DOCTYPE HTML P
0010  55 42 4C 49 43 20 22 2D 2F 2F 57 33 43 2F 2F 44  UBLIC "-//W3C//D
0020  54 44 20 48 54 4D 4C 20 34 2E 30 20 54 72 61 6E  TD HTML 4.0 Tran
0030  73 69 74 69 6F 6E 61 6C 2F 2F 45 4E 22 3E 0A 3C  sitional//EN">.<
0040  48 54 4D 4C 3E 0A 3C 48 45 41 44 3E 0A 3C 74 69  HTML>.<HEAD>.<ti
0050  74 6C 65 3E 47 6F 20 61 68 65 61 64 20 61 6E 64  tle>Go ahead and
0060  20 53 63 61 6E 4D 65 21 3C 2F 74 69 74 6C 65 3E   ScanMe!</title>
```

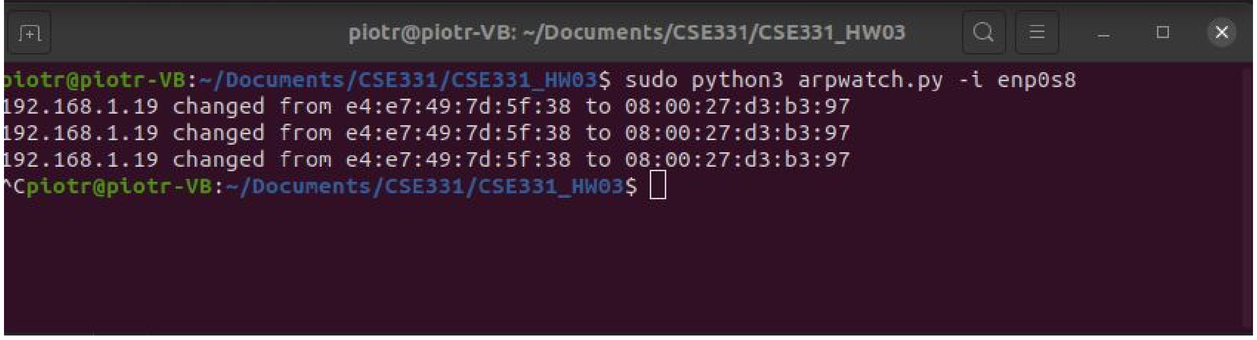
How I accomplished arpwatch.py

For arpwatch.py I started off by initializing an arp table using python_arptable which I found by just googling python arp table and finding this link https://pypi.org/project/python_arptable/ and afterwards I processed the command line arguments. I then used the sniff method after reading about it in the scapy usage documentation such that it will only look for arp packets on the interface set by the user and it will run this indefinitely until the user exits the program. Once a packet is received I will check its source ip, get the original mac address associated with that ip if it exists, and if it does exist then I will check if the new packet has a different mac address than the original and if not then I will print a message that it has changed.

Example output for arpwatch.py:



```
piotr@piotr-VB:~/Documents/CSE331/CSE331_HW03$ sudo arpspoof -i enp0s8 192.168.1.19
8:0:27:d3:b3:97 ff:ff:ff:ff:ff:ff 0806 42: arp reply 192.168.1.19 is-at 8:0:27:d3:b3:97
8:0:27:d3:b3:97 ff:ff:ff:ff:ff:ff 0806 42: arp reply 192.168.1.19 is-at 8:0:27:d3:b3:97
8:0:27:d3:b3:97 ff:ff:ff:ff:ff:ff 0806 42: arp reply 192.168.1.19 is-at 8:0:27:d3:b3:97
^CCleaning up and re-arping targets...
piotr@piotr-VB:~/Documents/CSE331/CSE331_HW03$
```



```
piotr@piotr-VB:~/Documents/CSE331/CSE331_HW03$ sudo python3 arpwatch.py -i enp0s8
192.168.1.19 changed from e4:e7:49:7d:5f:38 to 08:00:27:d3:b3:97
192.168.1.19 changed from e4:e7:49:7d:5f:38 to 08:00:27:d3:b3:97
192.168.1.19 changed from e4:e7:49:7d:5f:38 to 08:00:27:d3:b3:97
^Cpiotr@piotr-VB:~/Documents/CSE331/CSE331_HW03$
```