Jeffry Jimenez
CSE 331.01
Professor Rahmati
Fall 2020

# Report

Note: More examples are in the README.txt

**TASK 1:**

Synprobe.py uses the ipadress, sys, socket, time, and scapy python libraries. The script first creates a ports array with five default ports that will be searched in case no ports are specified by the user. The sendDummy() function sends http requests until it gets a response from the server (at most 3 times) using sockets. The code block for using sockets in python I got from #http://zetcode.com/python/socket/. After setting up the array with the default ports the script checks if the user provided the -p flag with the port ranges and appropriately parses it and updates the ports array. The script then checks the target IP (could be an IP, link or subnet), if it's a subnet I used the ipaddress librarie's ip_network method to get a list of IP's. For every IP target address The script does the following: For every port in the ports array use scapy.sr1 to send a sync request to the target, if sr1 returns none port is deemed as "filtered", if sr1 returns the flag RA (reset ack) port is "closed", in both cases the status is logged into terminal, with no further action. If scapy.sr1 returns the flag SA (sync Acknowledged) then the port is "open" and the script tries to get a response from the server using the sendDummy() function. If the server returns a payload then the payload is logged as a fingerprint, if no payload is returned in sendDummy() then the finger print for that port becomes "Port: {port#}, 3 requests transmitted, 0 bytes received". If the port is closed or open the script sends a packet with an 'R' (reset) flag to the server using the send() function from scapy. .

NOTE: If you are running the examples in README.txt, The program and nmap used to return some of the filtered ports as closed, maybe my firewall has something to do with this. All the output in the README.txt was compared to that of nmap -sSV and the results are the same.

```
$ sudo python3 synprobe.py -p 9929 scanme.nmap.org
        Host: scanme.nmap.org
        PORT      STATUS    FINGERPRINT
        9929      open      010100186a34f4675fa10f86000…
```

**TASK 2:**

arpwatch.py is simple. You will need to have arp in your computer if you plan to run this task. The only python libraries used are scapy, os and re. Pattern and Pattern2 in the code are simple regular expressions that help me fetch and filter the output from arp in the command line. In the code, I first check if the user asked for a specific interface, if not then the default interface is ens33. The dictionary address_to_mac contains the IP's as key and MAC's as values. I then run os.popen('arp -i'), the output of the first called to arp are going to be used as the based info, hence it is just added to the dictionary. sniff runs constantly returning a packet to the callback function arp_scan(). In arp_scan() If the IP of the packet sent by scapy.sniff() is not in the address_to_mac and it is simply added to it with its MAC of the packet as its value.  If the IP of the packet is in address_to_mac then MAC of the packet is compared to the one in the dictionary. In the case that the MAC's are different the user is notified that there was a change, otherwise nothing happens. This is done constantly until the used presses any key in their keyboard.

```
        #######TESTED WITH arpspoof  *DEFAULT Interface ens33*
                $ sudo python3 arpwatch.py
                192.168.60.1 changed from <pre MAC> to 00:50:56:c0:00:08
                192.168.60.1 changed from <pre MAC> to 00:50:56:c0:00:08
                192.168.60.1 changed from <pre MAC> to 00:50:56:c0:00:08
                192.168.60.1 changed from <pre MAC> to 00:50:56:c0:00:08
                192.168.60.1 changed from <pre MAC> to 00:50:56:c0:00:08
```