COMP 6841PROJECT

Week 3:

In week 3, I have analyzed the basics of port, importance of port scanning, different types of port scanning and refreshed some network concepts like Socket Programming.

What is a port?

A port is a logical connection, not a physical one, used by programs and services to exchange information. It identifies which program or service on a computer or server will be used for tasks like accessing web pages, FTP services, or email.

PORT NUMBERS:

1. System or Well-Known Ports (0-1023): Common ports used daily, such as 80, 443, 25, and 21. These are typically used on servers.
2. User or Registered Ports (1024-49151): Ports that can be registered by companies and developers for specific services. These are also typically used on servers.
3. Dynamic or Private Ports (49152-65535): Client-side ports that your computer temporarily assigns to itself during a session (e.g., when viewing a web page).

The General idea for a port scanner is

```
for each computer:
    for each port:
        Test if the port is open
```

Even the famous Port Scanner Nmap scans only the 1,000 common ports by default. These ports were also chosen based on their prevalence like 22 (SSH), 80 (HTTP), and 443(HTTPS).

It offers the fast scan which scans only the top 1000 ports, and it doesn't scan the private ports unless specified because private ports are managed by the OS for new connections with different websites in your web browsers and are usually temporarily assigned.

Why Port Scanning is Important?

1. Detecting Unprotected Services:

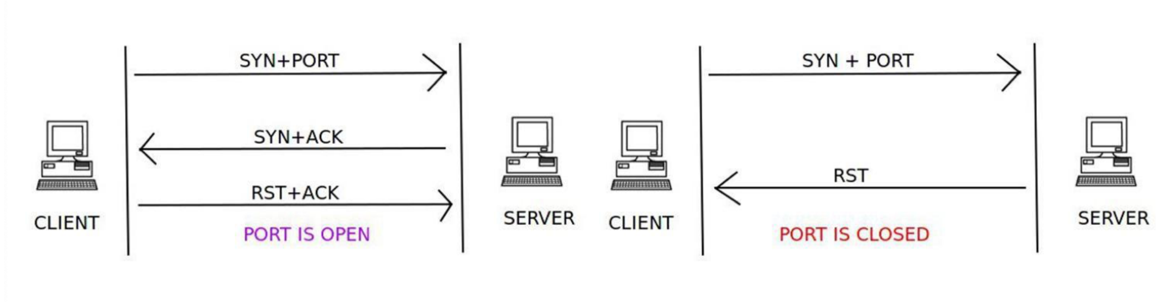
   Port scanning uncovers open ports that may be running remote access controls, exposed file servers, open webcams, or unauthorized services. These can include hidden or unsecured data and are often overlooked, leading to potential breaches if exploited.

2. Searching for Vulnerable Services:
   Open ports often correlate with services or software that could be outdated or unpatched. Cyber attackers actively look for these vulnerable services, as they provide avenues to exploit known vulnerabilities—many of which are tracked publicly in databases like CVE.

Generally, Developers are lazy to change some default passwords for web servers and port scanning helps us to mitigate such cyber risks.

Port Scanning Methods:

1. TCP Port Scanning



   a. Connect Scan: This method attempts to complete a three-way TCP handshake with the target host. If the handshake is completed, the port is open.
   b. SYN/Stealth Scan: This method involves sending only SYN packets without completing the three-way handshake. If the port is open, a SYN-ACK packet is returned, indicating the port is open without sending the final ACK packet. This was historically called "stealth" because it could bypass early firewall logging, but this is no longer the case with modern firewalls.

2. UDP Port Scanning
   An empty UDP packet is sent to the port. If the port is open, no reply is expected. If the port is closed, an ICMP "unreachable" packet should be sent back.
   UDP scanning is less reliable because firewalls and routers can drop ICMP packets, leading to misleading results.

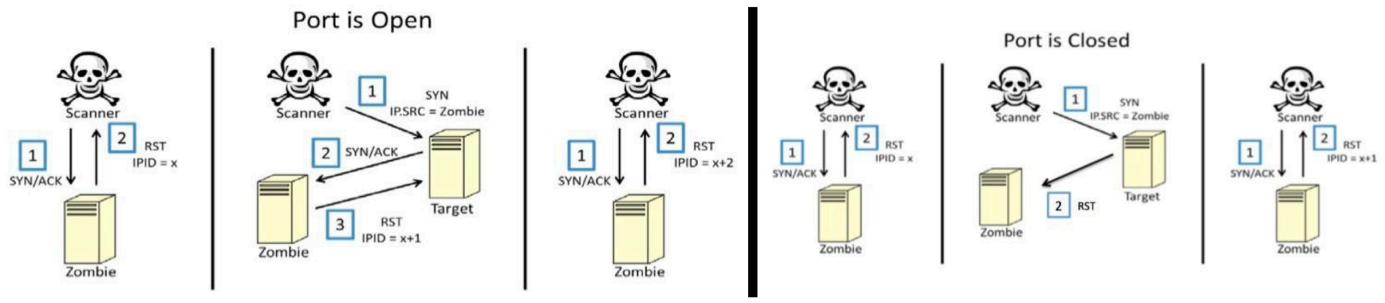
WEEK - 4:

In week 4, I decided to analyze and study the most popular Port Scanning tool Nmap and RUSTSCAN. I also studied more on port scanning ways and made report on problems and solution associated with it.

Advanced Port Scanning Technique – ZOMBIE SCANNING:

This scanning idea is to use a third-party machine (Zombie) to send spoofed packets to the target machine. It works by exploiting the IP ID Field in IP Headers. Many systems increment the IP ID field globally for each packet they sent. So, if zombie hasn't been communicating much, its IP ID value changes only when you interact with it, making it predictable. This is very stealthy way of achieving the things as your IP never touches the target machine. NMAP supports this through **nmap -Pn -sI <zombie_ip> <target_ip>.**

Check if the connection count has incremented by 1 (closed) or 2 (open).



Problems Associated with port scanning:

Firewalls control the traffic that flows into and out of the system. They block traffic based on some rules and usually Blacklist and whitelist certain IPs. They can also traffic block all traffic on certain ports.

Network Intrusion detection systems scan for known malicious patterns. Since, all the port scanning follows the same patterns its can easily detected by the NIDS as probing activity can quickly trigger alerts.

A default Nmap TCP scan checks 1000 popular ports, generating about 70 kilobytes of traffic. Scanning all 65,535 ports on a machine can generate around 4 megabytes of traffic.

Features like the Frequent changing access, short connections and bursts of traffic to many different local ports and destinations can help us to identify a port scan.

Intensive port scans can burden the network and impacting performance and gets easily detected by NIDS. So, it’s better to limit scans to the most common and vulnerable ones and follow priority order.

NMAP Demo:

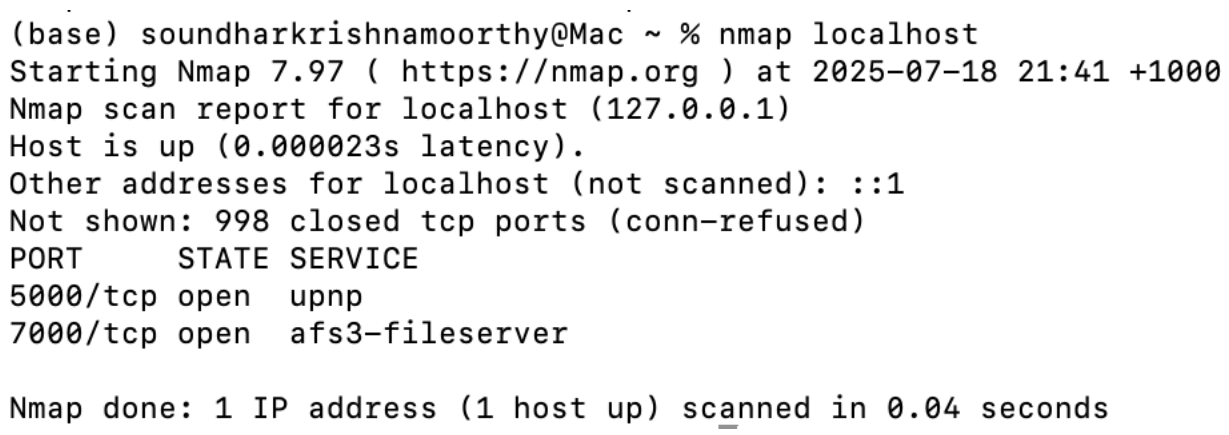
Nmap default scanning method is the TCP SYN scan mentioned above, which is used when you run Nmap with root (administrator) privileges. The SYN scan sends SYN packets to target ports and analyzes responses without completing the full TCP handshake, making it a stealthier and faster technique.

If you do not have root privileges, Nmap defaults to the TCP Connect scan, which uses the operating system's network functions to complete the full three-way TCP handshake on each port (less stealthy, potentially slower).

The UDP scan is another scanning method, but it is not the default. You must explicitly specify it using the -sU switch if you want Nmap to scan UDP ports.

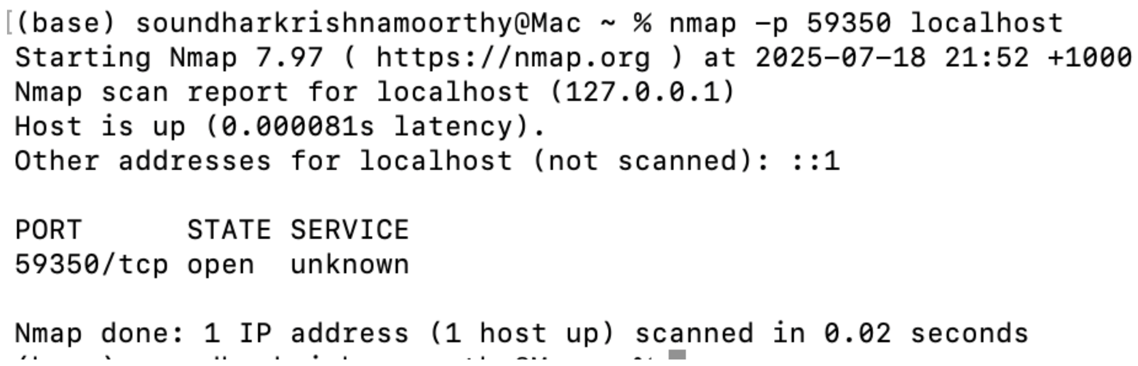
First, I did port scanning to my local machine which is not running any web server and then I decided to run web server at multiple ports and see the difference.

Without webserver running:



```
(base) soundharkrishnamoorthy@Mac ~ % nmap localhost
Starting Nmap 7.97 ( https://nmap.org ) at 2025-07-18 21:41 +1000
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000023s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 998 closed tcp ports (conn-refused)
PORT     STATE SERVICE
5000/tcp open  upnp
7000/tcp open  afs3-fileserver

Nmap done: 1 IP address (1 host up) scanned in 0.04 seconds
```

The default is configured to run scans for the thousand ports with some selected port numbers based on prevalence. So,it couldn’t identify the service running at port 59350 which was identified through Rustscan.

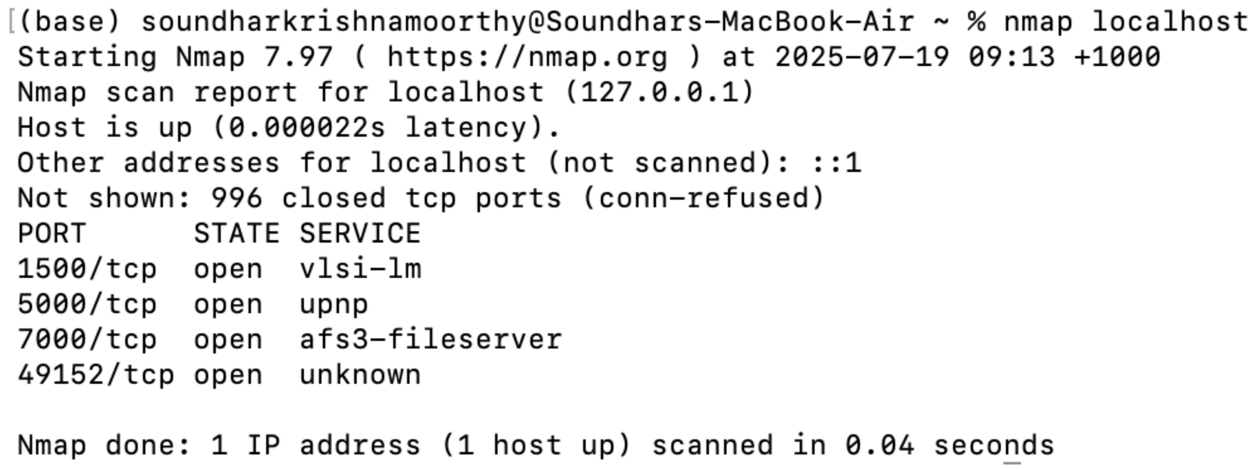


```
(base) soundharkrishnamoorthy@Mac ~ % nmap -p 59350 localhost
Starting Nmap 7.97 ( https://nmap.org ) at 2025-07-18 21:52 +1000
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000081s latency).
Other addresses for localhost (not scanned): ::1

PORT      STATE SERVICE
59350/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 0.02 seconds
```

I confirmed the extra port told by rustscan by specifying manually with nmap and gave the confirmation that it is open.

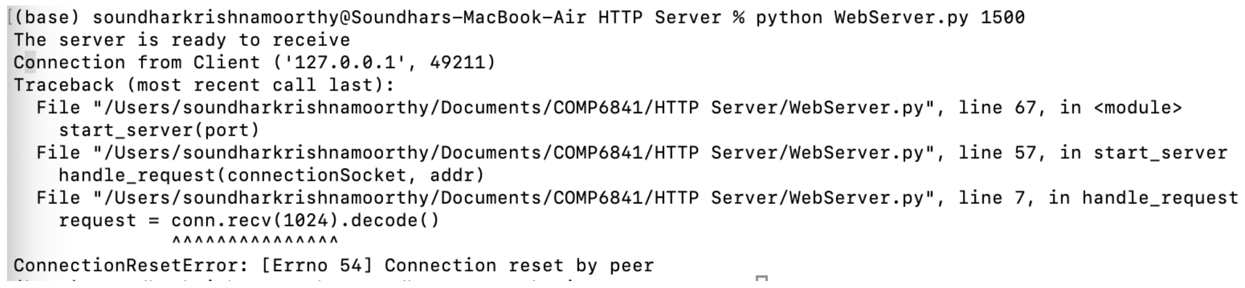
With webserver running at multiple ports:

I started two simple TCP based Simple HTTP webservers at port 1200 and 1500 and I could see the difference with the use of the Sudo i.e. without Sudo it runs a full connect scan and with Sudo it can run SYN scan. Here is the difference what happened in the webserver.

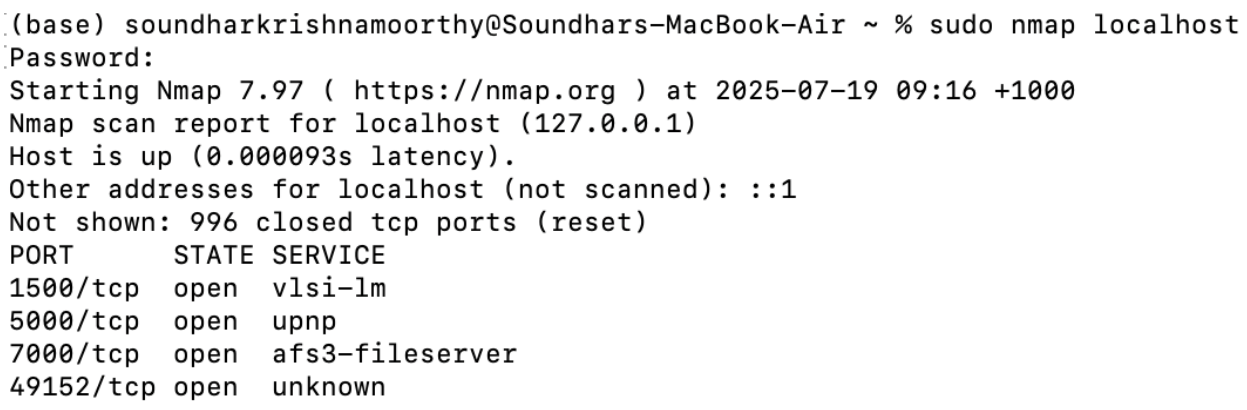


```
(base) soundharkrishnamoorthy@Soundhars-MacBook-Air ~ % nmap localhost
Starting Nmap 7.97 ( https://nmap.org ) at 2025-07-19 09:13 +1000
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000022s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 996 closed tcp ports (conn-refused)
PORT      STATE SERVICE
1500/tcp  open  vlsi-lm
5000/tcp  open  upnp
7000/tcp  open  afs3-fileserver
49152/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 0.04 seconds
```

Though it found one of the web servers running at port 1500. Since, it used the Full Connect it was spotted by my server as it didn't give the request as my server expected.

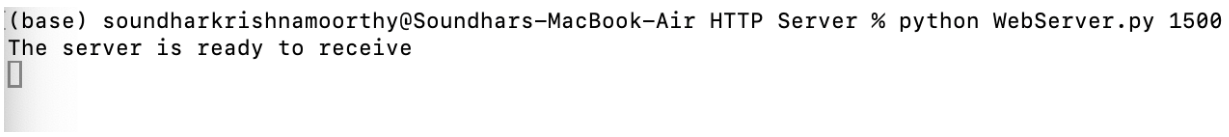


```
(base) soundharkrishnamoorthy@Soundhars-MacBook-Air HTTP Server % python WebServer.py 1500
The server is ready to receive
Connection from Client ('127.0.0.1', 49211)
Traceback (most recent call last):
  File "/Users/soundharkrishnamoorthy/Documents/COMP6841/HTTP Server/WebServer.py", line 67, in <module>
    start_server(port)
  File "/Users/soundharkrishnamoorthy/Documents/COMP6841/HTTP Server/WebServer.py", line 57, in start_server
    handle_request(connectionSocket, addr)
  File "/Users/soundharkrishnamoorthy/Documents/COMP6841/HTTP Server/WebServer.py", line 7, in handle_request
    request = conn.recv(1024).decode()
              ^^^^^^^^^^^^^^^
ConnectionResetError: [Errno 54] Connection reset by peer
```



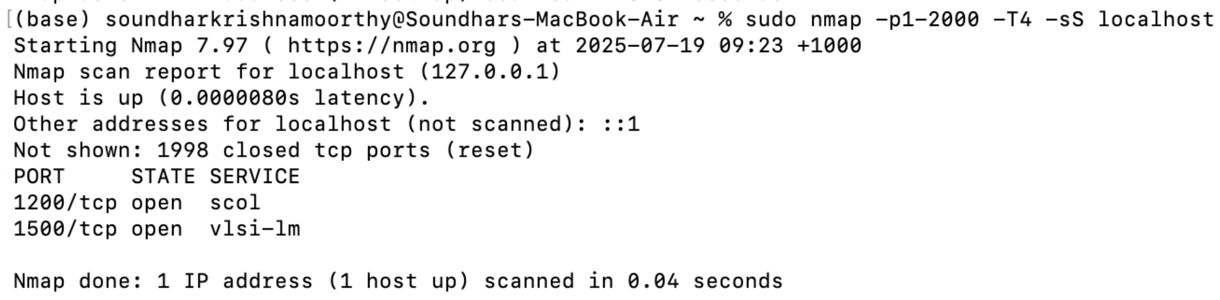
```
(base) soundharkrishnamoorthy@Soundhars-MacBook-Air ~ % sudo nmap localhost
Password:
Starting Nmap 7.97 ( https://nmap.org ) at 2025-07-19 09:16 +1000
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000093s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 996 closed tcp ports (reset)
PORT      STATE SERVICE
1500/tcp  open  vlsi-lm
5000/tcp  open  upnp
7000/tcp  open  afs3-fileserver
49152/tcp open  unknown
```

Now it was on SYN Mode for port scan, and it didn't affect by web server as the TCP three-way handshake is not complete.



```
(base) soundharkrishnamoorthy@Soundhars-MacBook-Air HTTP Server % python WebServer.py 1500
The server is ready to receive
```

I manually specified the port range to scan used stealth mode and then scanned it.

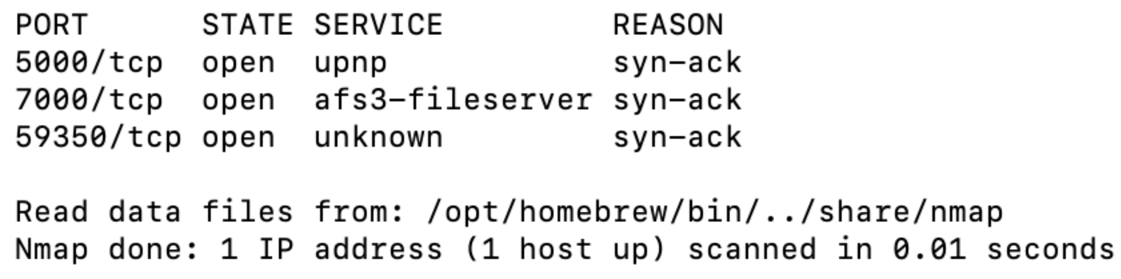


```
(base) soundharkrishnamoorthy@Soundhars-MacBook-Air ~ % sudo nmap -p1-2000 -T4 -sS localhost
Starting Nmap 7.97 ( https://nmap.org ) at 2025-07-19 09:23 +1000
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000080s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 1998 closed tcp ports (reset)
PORT     STATE SERVICE
1200/tcp open  scol
1500/tcp open  vlsi-lm

Nmap done: 1 IP address (1 host up) scanned in 0.04 seconds
```

The problem here is the 1200 and 1500 are mapped to some other services which is not running there which made me realize that it follows some priority list of ports and services running there.

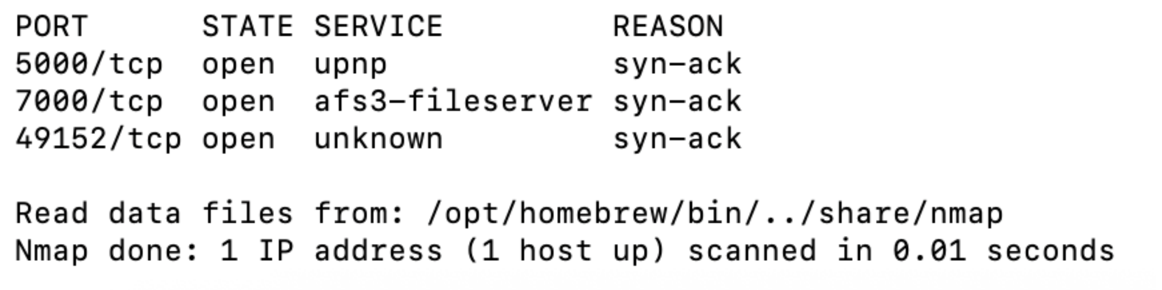
RUSTSCAN DEMO:

Without Webserver running:



```
PORT      STATE SERVICE         REASON
5000/tcp  open  upnp            syn-ack
7000/tcp  open  afs3-fileserver syn-ack
59350/tcp open  unknown         syn-ack

Read data files from: /opt/homebrew/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 0.01 seconds
```

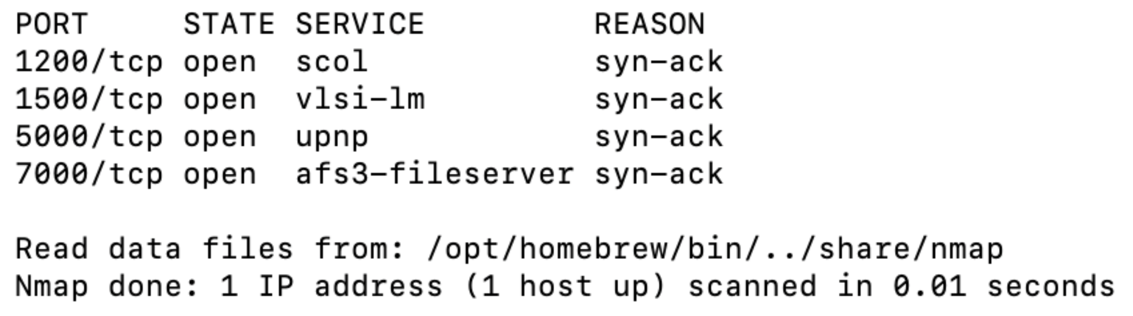
With Webserver running:



```
PORT      STATE SERVICE          REASON
5000/tcp  open  upnp             syn-ack
7000/tcp  open  afs3-fileserver  syn-ack
49152/tcp open  unknown          syn-ack

Read data files from: /opt/homebrew/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 0.01 seconds
```

Rustscan didn't spot my webserver instances running. So, I decided to keep my give all ports to scan.

```
rustscan -a 127.0.0.1 --range 1-10000 --scan-order "Random"
```



```
PORT     STATE SERVICE          REASON
1200/tcp open  scol             syn-ack
1500/tcp open  vlsi-lm          syn-ack
5000/tcp open  upnp             syn-ack
7000/tcp open  afs3-fileserver  syn-ack

Read data files from: /opt/homebrew/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 0.01 seconds
```

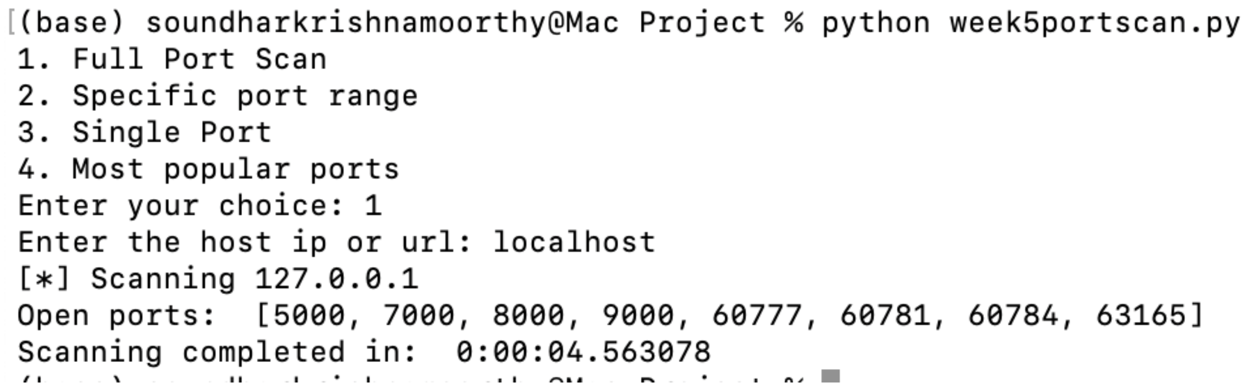
The main idea of this activity is to understand various commands and how Nmap and rustscan performs and to gather the working principle to understand my own port scanning tool for it.

Week 5:

I wrote a basic port scanning tool which can take IP address and port range or port as input and identify the open ports and closed ports. This scanning involved using the TCP Full connection and took lot of time in scanning every port one by one if my option was to scan all ports.

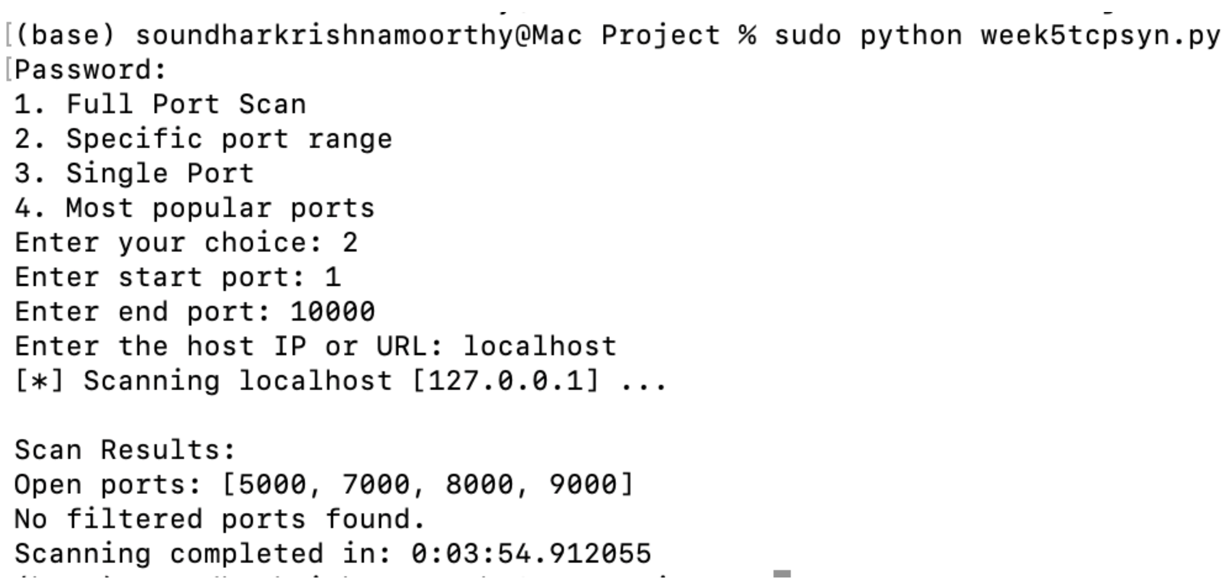
I ran netcat on port 8000 and port 9000.

With TCP Full connect Scan (Normal):



```
[(base) soundharkrishnamoorthy@Mac Project % python week5portscan.py
 1. Full Port Scan
 2. Specific port range
 3. Single Port
 4. Most popular ports
 Enter your choice: 1
 Enter the host ip or url: localhost
 [*] Scanning 127.0.0.1
 Open ports:  [5000, 7000, 8000, 9000, 60777, 60781, 60784, 63165]
 Scanning completed in:  0:00:04.563078
```

With TCP SYN Connect Scan:



```
[(base) soundharkrishnamoorthy@Mac Project % sudo python week5tcpsyn.py
[Password:
 1. Full Port Scan
 2. Specific port range
 3. Single Port
 4. Most popular ports
 Enter your choice: 2
 Enter start port: 1
 Enter end port: 10000
 Enter the host IP or URL: localhost
 [*] Scanning localhost [127.0.0.1] ...

 Scan Results:
 Open ports: [5000, 7000, 8000, 9000]
 No filtered ports found.
 Scanning completed in: 0:03:54.912055
```

Then, I used **scapy** library to make TCP SYN Connect based port scanning. I needed root user privileges to alter my TCP segments and set flag S in SYN Flag field in TCP segment and send the IP packet. This scan also helps to know which ports are blocked by the firewall and which ports are Open.

Week 5 Conclusion:

When I used the normal TCP Full connect tool it stopped my netcat server which were running on the ports 8000 and 9000.But, TCP SYN Connect port scanner didn't stop them running. This happens because TCP full connections exhaust available slots for new connections, but SYN scans won't disturb them as they don't consume network resource.

Week 6:

What I observed from last week is single threaded port scanning was very slow, and I decided to use multi-threaded port scanning and decided to use the Nmap services file which gives the service, port number and frequency of that service running to name the services for the open ports I found during the port scanning.

I used these commands to extract the nmap information to my local file directory.

This command gave me the 1000 ports and their services which were sorted based on the third column frequency, which I can use as the base scan.

```
awk '!/^#/ && $2 ~ /\/tcp$/ { print $1, $2, $3 }' /opt/homebrew/share/nmap/nmap-services | sort -k3,3nr | head -n 1000 > top_tcp_ports.txt
```

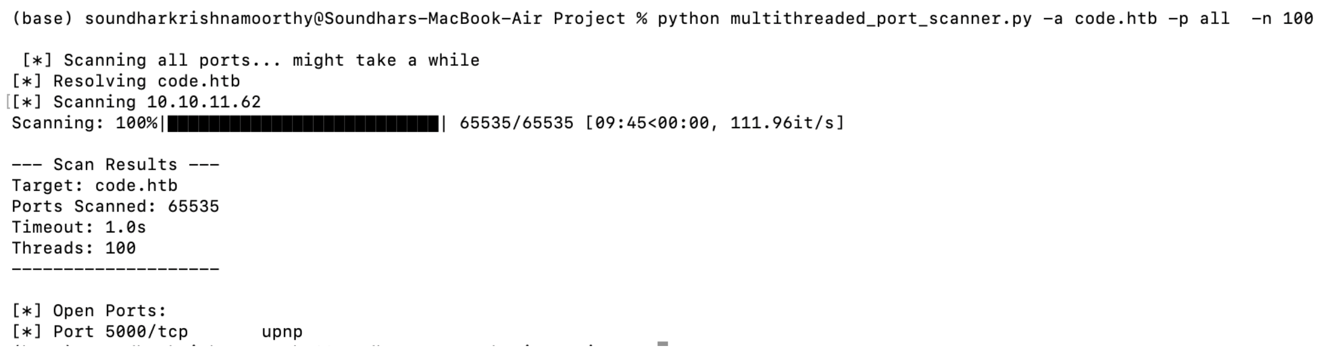
This file got all the services, and their port numbers associated with it.

```
cat /opt/homebrew/share/nmap/nmap-services > nmap_default_ports.txt
```

I had multiple options to use the port scanner, and I have the description and usage of them below:

Arguments	Description
-a	Required. This is to give the Target IP address or domain name which will be later resolved.
-p	Optional. This is to mention the port range or even single port. If nothing is given, it performs the basic port scanning of top 1000 most common ports. It takes port range as argument like 1-100, or all.
-T	Timeout for the TCP packets.
-n	Number of threads (default value is 10 and it can go up to 100)
-o	To save the results to the output file.

I have tried this port scanning on the Week 7 CTFD machines to understand which ports are open in that machine.



```
(base) soundharkrishnamoorthy@Soundhars-MacBook-Air Project % python multithreaded_port_scanner.py -a code.htb -p all  -n 100

 [*] Scanning all ports... might take a while
[*] Resolving code.htb
[*] Scanning 10.10.11.62
Scanning: 100%|█████████████████████████| 65535/65535 [09:45<00:00, 111.96it/s]

--- Scan Results ---
Target: code.htb
Ports Scanned: 65535
Timeout: 1.0s
Threads: 100
--------------------

[*] Open Ports:
[*] Port 5000/tcp       upnp
```

Week 7:

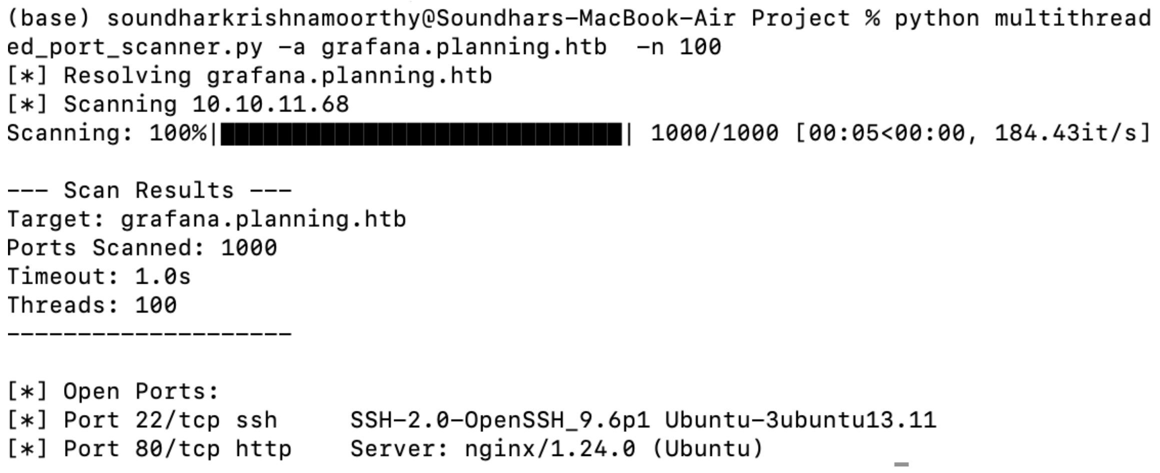
I had discussions with my tutor and asked for additional service implementations. He suggested me the idea of getting version from the service.

I tried understanding how Nmap does it. The NMAP Commands file contained lot of different HTTP request message which is unique to each service which was very hard to understand.

I decided to do it only for the most famous ports like HTTP (PORT 80 or 8080) , HTTPS (443) , SSH and Telent (22 and 23), Redis (6379) or MySQL (3306).

I also decided to do CVE Lookup to spot the vulnerabilities in the version detected. I had the CVE dictionary which contained the CVE ID and used the description field to match my vulnerability.

I added new modules like service detection and CVE Lookup this week.



```
(base) soundharkrishnamoorthy@Soundhars-MacBook-Air Project % python multithread
ed_port_scanner.py -a grafana.planning.htb  -n 100
[*] Resolving grafana.planning.htb
[*] Scanning 10.10.11.68
Scanning: 100%|█████████████████████████████| 1000/1000 [00:05<00:00, 184.43it/s]

--- Scan Results ---
Target: grafana.planning.htb
Ports Scanned: 1000
Timeout: 1.0s
Threads: 100
--------------------

[*] Open Ports:
[*] Port 22/tcp ssh     SSH-2.0-OpenSSH_9.6p1 Ubuntu-3ubuntu13.11
[*] Port 80/tcp http    Server: nginx/1.24.0 (Ubuntu)
```

You can see the service version detections in the above output. I have decided to give those service versions as Input to my CVE Look up program which would give me if any vulnerabilities were present.

Week 8:

This week I have decided to visualize the entire flow for a basic penetration testing along with my tools. Port scanning an unknown machine or a university network is illegal to do and remembering Good Faith Policy I have attempted all this in my home network and used the HackTheBox machines for the same purposes.

Only in a place where you are allowed to do Pen testing in a network. You can do these things.

The flow starts with running a

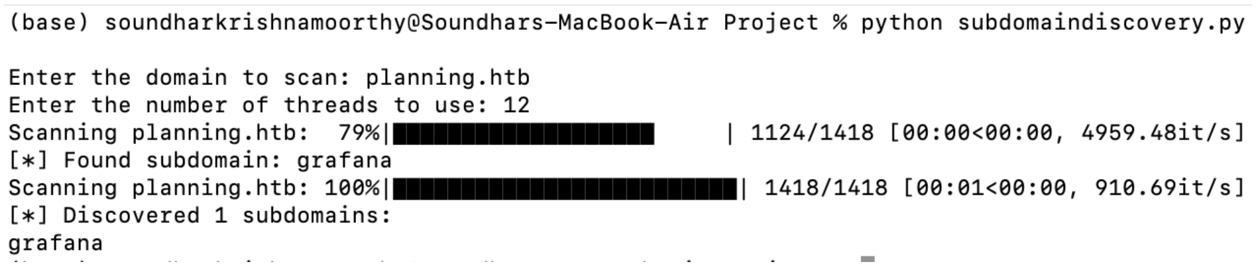
1. Address Resolution Protocol in my local network which would get me the IP address and MAC address of the devices associated in the network.
2. Port Scanning and Service Enumeration.
3. Subdomain Enumeration.
4. Vulnerability CVE ID Lookup.

I have spent my time in writing the arpscan.py which does the functionality of scanning your local network if you give the IP address in CIDR formatting and gives all the reachable hosts in that network. Then, I decided to look how FFUF works and decided to write similar in a low level to stimulate that tool. Considering the time constraint, I decided to write only the sub domain discovery but FFUF lot of offensive security things like

1. content discovery – which takes the end points and word list, which is to be used, then lists the files present in the server.
   ffuf -w /path/to/wordlist -u https://target/FUZZ
2. Virtual Host discovery
   which also has a parameter to filter out the file size.
3. GET Parameter fuzzing
4. POST Data fuzzing

It looked like an amazing tool when I tested it during the CTFD Challenges for Pen Testing. For subdomain discovery it also used the HTTP request and analyses the responses based on status code and response size.

I decided to use the common.txt from the SecLists under DNS subdirectory. The point of using the sub domain discovery is to identify the potential ways to open the server. Sometimes, we might find the API end points, unsecured test end points and some monitoring tools. These tools or servers sometimes might not be patched to the latest version giving us a chance to attack.



```
(base) soundharkrishnamoorthy@Soundhars-MacBook-Air Project % python subdomaindiscovery.py

Enter the domain to scan: planning.htb
Enter the number of threads to use: 12
Scanning planning.htb:  79%|███████████████████     | 1124/1418 [00:00<00:00, 4959.48it/s]
[*] Found subdomain: grafana
Scanning planning.htb: 100%|█████████████████████████| 1418/1418 [00:01<00:00, 910.69it/s]
[*] Discovered 1 subdomains:
grafana
```

I have also done proper formatting and commenting of my codes and pushed it to a public GitHub repository and did all the documentation works.