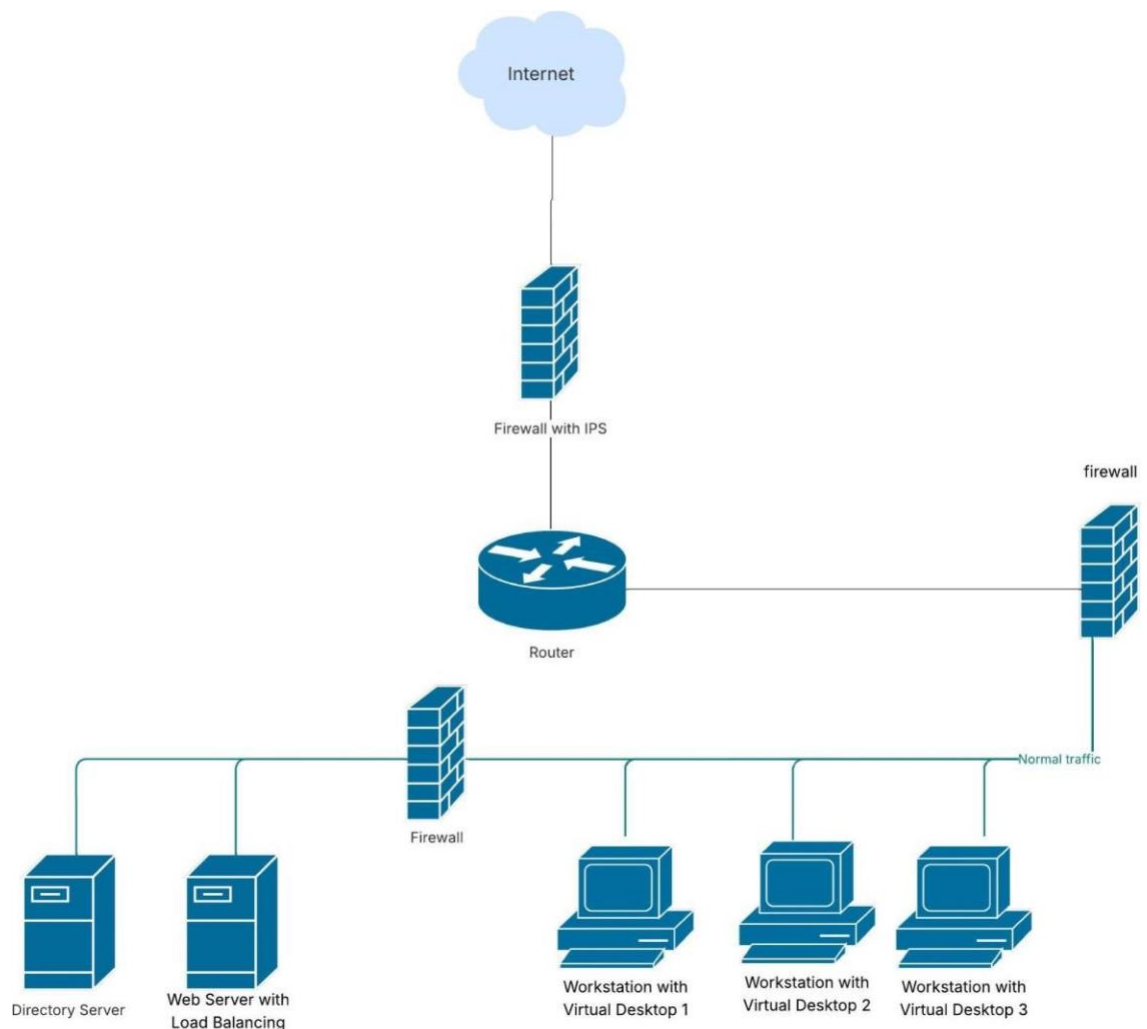


Building a small secure network

Objective:

- Establish basic firewall controls using UFW.
- Conduct port scanning and hardening.
- Monitor traffic with Wireshark.
- Simulate reconnaissance attacks and analyze detection mechanisms

1. Network diagram



2. Basic security controls

- Firewall rules (UFW linux):
 - Started with a "deny all" default policy
 - Created specific allow rules for necessary services
 - Tested the rules to ensure they work as expected

```
[~]
oblee ➤ sudo ufw status
Status: active

[~]
oblee ➤ sudo ufw default deny incoming
Default incoming policy changed to 'deny'
(be sure to update your rules accordingly)

[~]
oblee ➤ sudo ufw default allow outgoing
Default outgoing policy changed to 'allow'
(be sure to update your rules accordingly)

[~]
oblee ➤ sudo ufw allow ssh
Rule added
Rule added (v6)

[~]
oblee ➤ sudo ufw allow 80/tcp
Rule added
Rule added (v6)

[~]
oblee ➤ sudo ufw allow 443/tcp
Rule added
Rule added (v6)

[~]
```

Figure 2.1: enabling UFW and setting up rules

```
[~]  
oblee ➤ python3 -m http.server 8080  
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...  
127.0.0.1 - - [06/Jun/2025 15:01:45] "GET / HTTP/1.1" 200 -
```

Figure 2.2: Starting a simple web server on port 8080

```
[~]  
oblee ➤ sudo ufw status verbose  
Status: active  
Logging: on (low)  
Default: deny (incoming), allow (outgoing), deny (routed)  
New profiles: skip  
  
To Action From  
--  
8080 ALLOW IN Anywhere  
8080 (v6) ALLOW IN Anywhere (v6)
```

Figure 2.3: The status of UFW as at the time the server was started

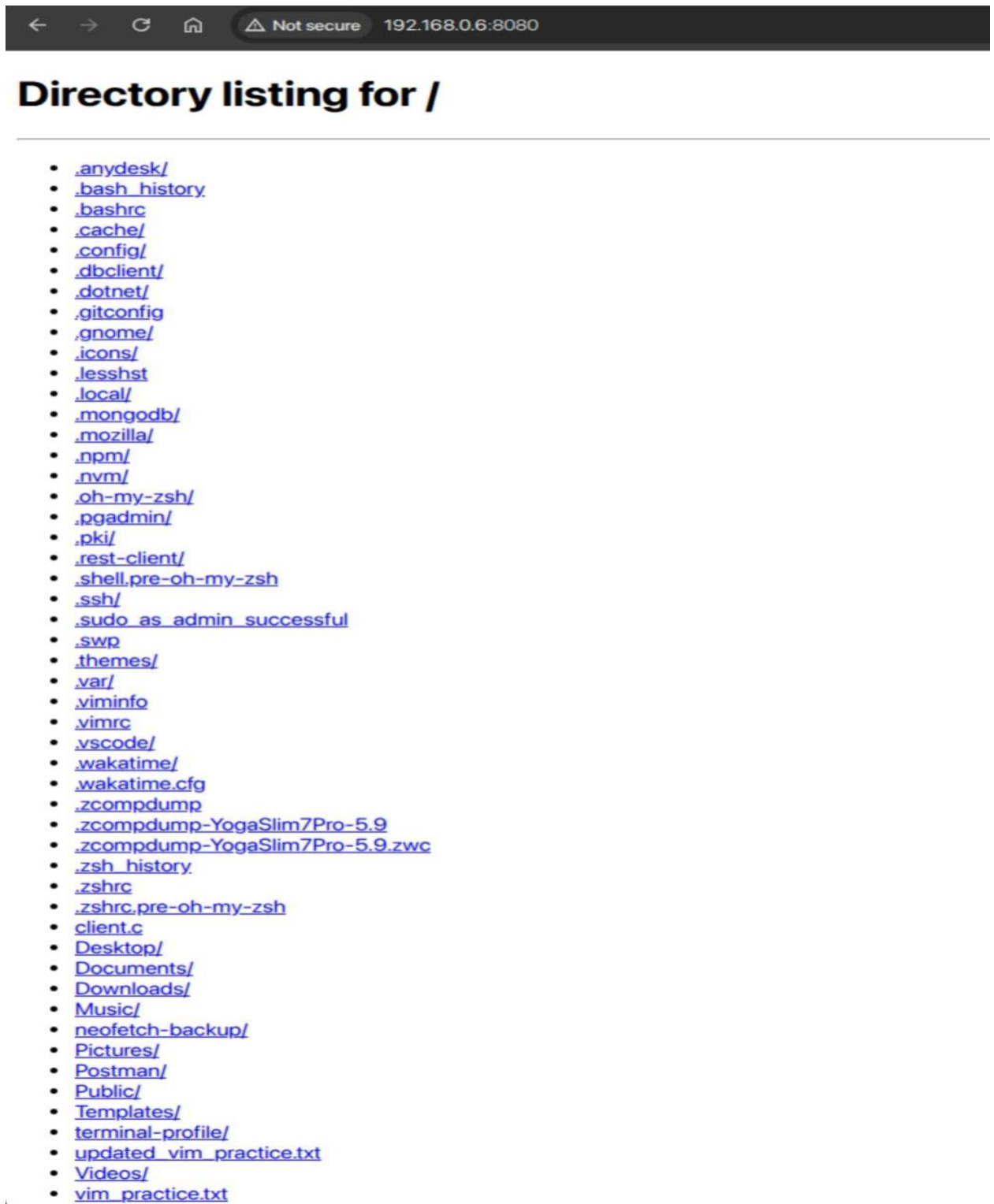


Figure 2.4: <http://192.186.0.6:8080> in a browser before access is denied

```
[~]
oblee ➤ sudo ufw deny 8080
Rule updated
Rule updated (v6)

[~]
oblee ➤ sudo ufw status
Status: active

To Action From
--
22/tcp ALLOW Anywhere
80/tcp ALLOW Anywhere
443/tcp ALLOW Anywhere
8080 DENY Anywhere
22/tcp (v6) ALLOW Anywhere (v6)
80/tcp (v6) ALLOW Anywhere (v6)
443/tcp (v6) ALLOW Anywhere (v6)
8080 (v6) DENY Anywhere (v6)
```

Figure 2.5: The status of UFW showing port 8080 blocked

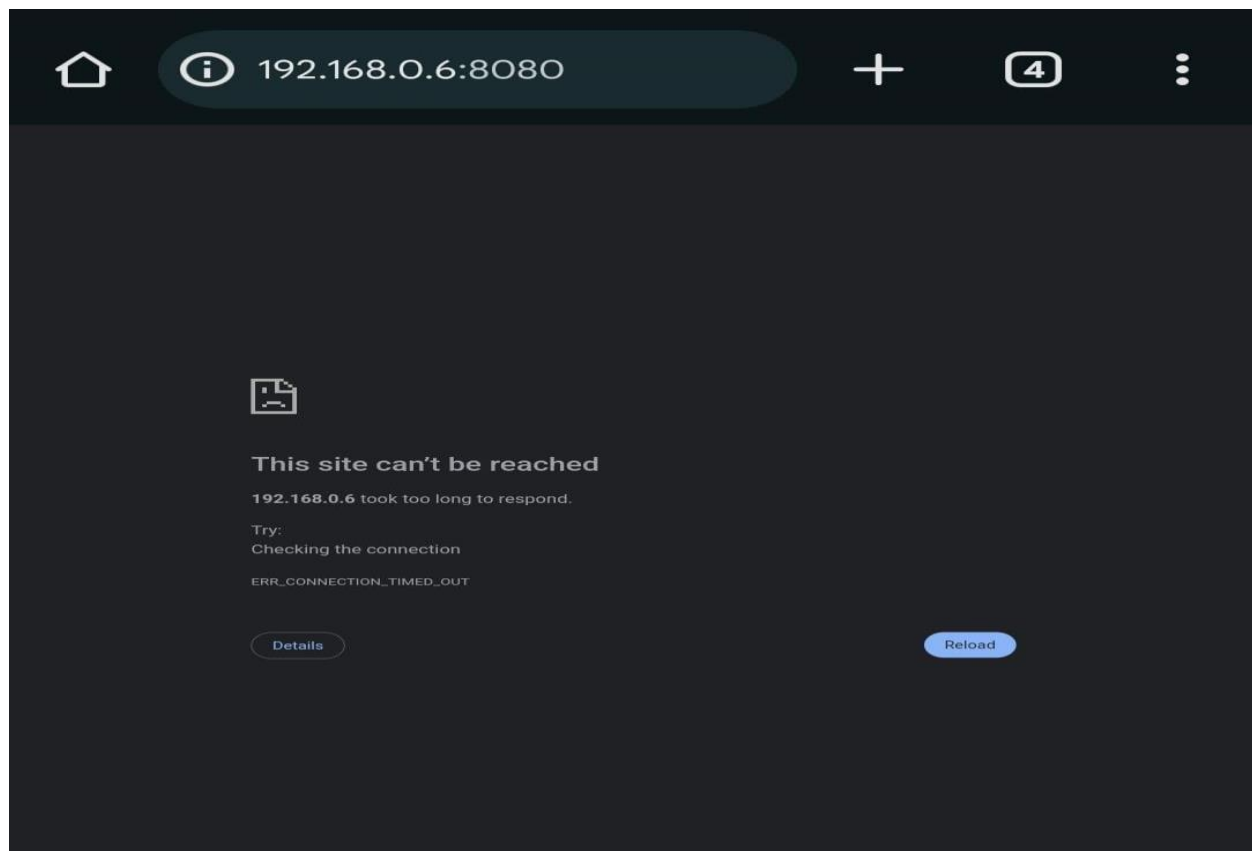


Figure 2.6: <http://192.186.0.6:8080> in a browser after access is denied

- Port scanning and hardening: Used Nmap to discover what's actually running

```
oblee nmap 192.168.0.1
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-06-06 14:49 WAT
Nmap scan report for _gateway (192.168.0.1)
Host is up (0.029s latency).
Not shown: 997 closed tcp ports (conn-refused)
PORT      STATE SERVICE
53/tcp    open  domain
80/tcp    open  http
443/tcp   open  https

Nmap done: 1 IP address (1 host up) scanned in 0.40 seconds

[~]
oblee nmap localhost
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-06-06 14:51 WAT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000074s latency).
Not shown: 997 closed tcp ports (conn-refused)
PORT      STATE SERVICE
80/tcp    open  http
631/tcp   open  ipp
5432/tcp  open  postgresql

Nmap done: 1 IP address (1 host up) scanned in 0.08 seconds
```

Figure 2.7: Port scanning using nMap

```
[~]
oblee sudo nmap -sS -O localhost
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-06-06 14:43 WAT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000071s latency).
Not shown: 997 closed tcp ports (reset)
PORT      STATE SERVICE
80/tcp    open  http
631/tcp   open  ipp
5432/tcp  open  postgresql
Device type: general purpose
Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6.32
OS details: Linux 2.6.32
Network Distance: 0 hops

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1.42 seconds

[~]
oblee sudo nmap -sS -O 192.168.0.1
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-06-06 14:44 WAT
Nmap scan report for _gateway (192.168.0.1)
Host is up (0.0030s latency).
Not shown: 997 closed tcp ports (reset)
PORT      STATE SERVICE
53/tcp    open  domain
80/tcp    open  http
443/tcp   open  https
MAC Address: B8:D4:BC:8E:1B:AD (zte)
Device type: general purpose
Running: Linux 3.X
OS CPE: cpe:/o:linux:linux_kernel:3
OS details: Linux 3.2 - 3.16
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1.80 seconds
```

Figure 2.8 Checking to see what's running using nMap


```
[~]
oblee sudo systemctl stop postgresql

[~]
oblee nmap localhost
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-06-06 14:58 WAT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000073s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT      STATE SERVICE
80/tcp    open  http
631/tcp   open  ipp

Nmap done: 1 IP address (1 host up) scanned in 0.07 seconds
```

Figure 2.9: closing used ports

- Wireshark ICMP Ping Capture:
 - I captured 4 ICMP Echo Request and 4 Echo Reply packets between the machine (192.168.0.6) and 8.8.8.8.
 - The Echo Requests (Type 8) were sent from the machine, and Echo Replies (Type 0) were received in response.
 - Each packet includes an identifier and sequence number to track responses.
 - The round-trip time (RTT) was visible in the ping output and the ICMP reply delay in the packet timestamps.

*wlp0s20f3

No.	Time	Source	Destination	Protocol	Length	Info
13511	979.941904325	192.168.0.6	8.8.8.8	ICMP	98	Echo (ping) request id=0x3b9f, seq=1/256, ttl=64 (reply in 13512)
13512	979.981785195	8.8.8.8	192.168.0.6	ICMP	98	Echo (ping) reply id=0x3b9f, seq=1/256, ttl=111 (request in 13511)
13517	971.943127839	192.168.0.6	8.8.8.8	ICMP	98	Echo (ping) request id=0x3b9f, seq=2/512, ttl=64 (reply in 13518)
13518	971.962799855	8.8.8.8	192.168.0.6	ICMP	98	Echo (ping) reply id=0x3b9f, seq=2/512, ttl=111 (request in 13517)
13519	972.945493917	192.168.0.6	8.8.8.8	ICMP	98	Echo (ping) request id=0x3b9f, seq=3/768, ttl=64 (reply in 13520)
13520	972.970834507	8.8.8.8	192.168.0.6	ICMP	98	Echo (ping) reply id=0x3b9f, seq=3/768, ttl=111 (request in 13519)
13522	973.947097366	192.168.0.6	8.8.8.8	ICMP	98	Echo (ping) request id=0x3b9f, seq=4/1024, ttl=64 (reply in 13523)
13523	973.965906357	8.8.8.8	192.168.0.6	ICMP	98	Echo (ping) reply id=0x3b9f, seq=4/1024, ttl=111 (request in 13522)
13526	974.948270980	192.168.0.6	8.8.8.8	ICMP	98	Echo (ping) request id=0x3b9f, seq=5/1280, ttl=64 (reply in 13527)
13527	974.969811423	8.8.8.8	192.168.0.6	ICMP	98	Echo (ping) reply id=0x3b9f, seq=5/1280, ttl=111 (request in 13526)
13528	975.950540774	192.168.0.6	8.8.8.8	ICMP	98	Echo (ping) request id=0x3b9f, seq=6/1536, ttl=64 (reply in 13529)
13529	975.978662357	8.8.8.8	192.168.0.6	ICMP	98	Echo (ping) reply id=0x3b9f, seq=6/1536, ttl=111 (request in 13528)
13530	976.952012466	192.168.0.6	8.8.8.8	ICMP	98	Echo (ping) request id=0x3b9f, seq=7/1792, ttl=64 (reply in 13531)
13531	976.969808470	8.8.8.8	192.168.0.6	ICMP	98	Echo (ping) reply id=0x3b9f, seq=7/1792, ttl=111 (request in 13530)
13535	977.953522058	192.168.0.6	8.8.8.8	ICMP	98	Echo (ping) request id=0x3b9f, seq=8/2048, ttl=64 (reply in 13536)
13536	977.984747832	8.8.8.8	192.168.0.6	ICMP	98	Echo (ping) reply id=0x3b9f, seq=8/2048, ttl=111 (request in 13535)
13538	978.954799257	192.168.0.6	8.8.8.8	ICMP	98	Echo (ping) request id=0x3b9f, seq=9/2304, ttl=64 (reply in 13539)
13539	978.977778406	8.8.8.8	192.168.0.6	ICMP	98	Echo (ping) reply id=0x3b9f, seq=9/2304, ttl=111 (request in 13538)
13541	979.956134395	192.168.0.6	8.8.8.8	ICMP	98	Echo (ping) request id=0x3b9f, seq=10/2560, ttl=64 (reply in 13542)
13542	979.981872834	8.8.8.8	192.168.0.6	ICMP	98	Echo (ping) reply id=0x3b9f, seq=10/2560, ttl=111 (request in 13541)

- Wireshark HTTP capture:

- Started a test HTTP server: `python3 -m`

- `http.server 8080 --bind 192.168.0.6`

This hosted a simple file server on port **8080**, accessible from devices on the local network.

- Launched Wireshark, capturing on the `wlp0s20f3` interface.

- Accessed the server from the same machine using: `curl`

- `http://192.168.0.6:8080`

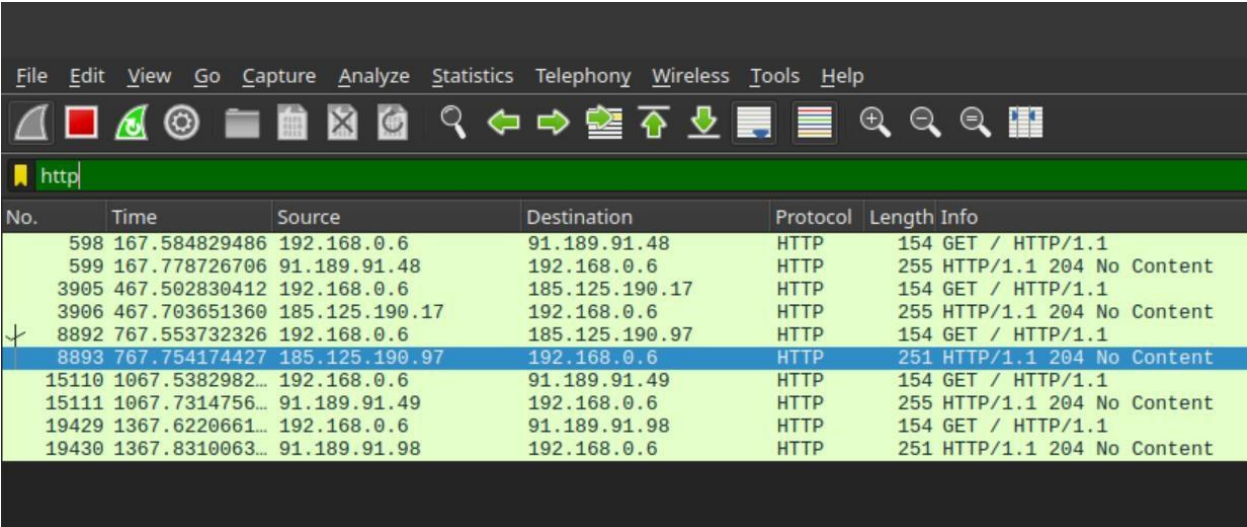
- Filtered packets in Wireshark using: `http`

- Wireshark successfully captured the HTTP GET request:

- Method: `GET`

■ Host: 192.168.0.6:8080

- The server responded with HTTP/1.0 200 OK, indicating the page was served correctly.



No.	Time	Source	Destination	Protocol	Length	Info
598	167.584829486	192.168.0.6	91.189.91.48	HTTP	154	GET / HTTP/1.1
599	167.778726706	91.189.91.48	192.168.0.6	HTTP	255	HTTP/1.1 204 No Content
3905	467.502830412	192.168.0.6	185.125.190.17	HTTP	154	GET / HTTP/1.1
3906	467.703651360	185.125.190.17	192.168.0.6	HTTP	255	HTTP/1.1 204 No Content
8892	767.553732326	192.168.0.6	185.125.190.97	HTTP	154	GET / HTTP/1.1
8893	767.754174427	185.125.190.97	192.168.0.6	HTTP	251	HTTP/1.1 204 No Content
15110	1067.5382982...	192.168.0.6	91.189.91.49	HTTP	154	GET / HTTP/1.1
15111	1067.7314756...	91.189.91.49	192.168.0.6	HTTP	255	HTTP/1.1 204 No Content
19429	1367.6220661...	192.168.0.6	91.189.91.98	HTTP	154	GET / HTTP/1.1
19430	1367.8310063...	91.189.91.98	192.168.0.6	HTTP	251	HTTP/1.1 204 No Content

3. Simulated Threat & Response

Reconnaissance Attack

Reconnaissance is the first phase of any attack, where an adversary gathers information about the target. I used Nmap, a powerful network scanning tool, to identify open ports and services running on the target machine.

I ran the command `nmap -A 192.168.0.6`: (the local machine).

```
[~]
oblee nmap -A 192.168.0.6
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-06-07 07:03 WAT
Nmap scan report for YogaSlim7Pro (192.168.0.6)
Host is up (0.000078s latency).
Not shown: 999 closed tcp ports (conn-refused)
PORT      STATE SERVICE VERSION
80/tcp    open  http      Apache httpd 2.4.58 ((Ubuntu))
|_http-server-header: Apache/2.4.58 (Ubuntu)
|_http-title: Apache2 Ubuntu Default Page: It works

Service detection performed. Please report any incorrect results at https://nmap.org/submit/
Nmap done: 1 IP address (1 host up) scanned in 6.40 seconds
```

This step revealed valuable information, such as open SSH and HTTP ports, which could be leveraged in subsequent attack phases.

Detection: I captured scan activity using Wireshark and filtered by `tcp.flags.syn == 1`

Mitigation: IDS (Wireshark for this demo) can detect unusual scan behavior.

Conclusion: This simple port scan demonstrated how an attacker might map open services. Basic firewall logging and packet analysis tools like Wireshark can effectively detect and mitigate such reconnaissance.