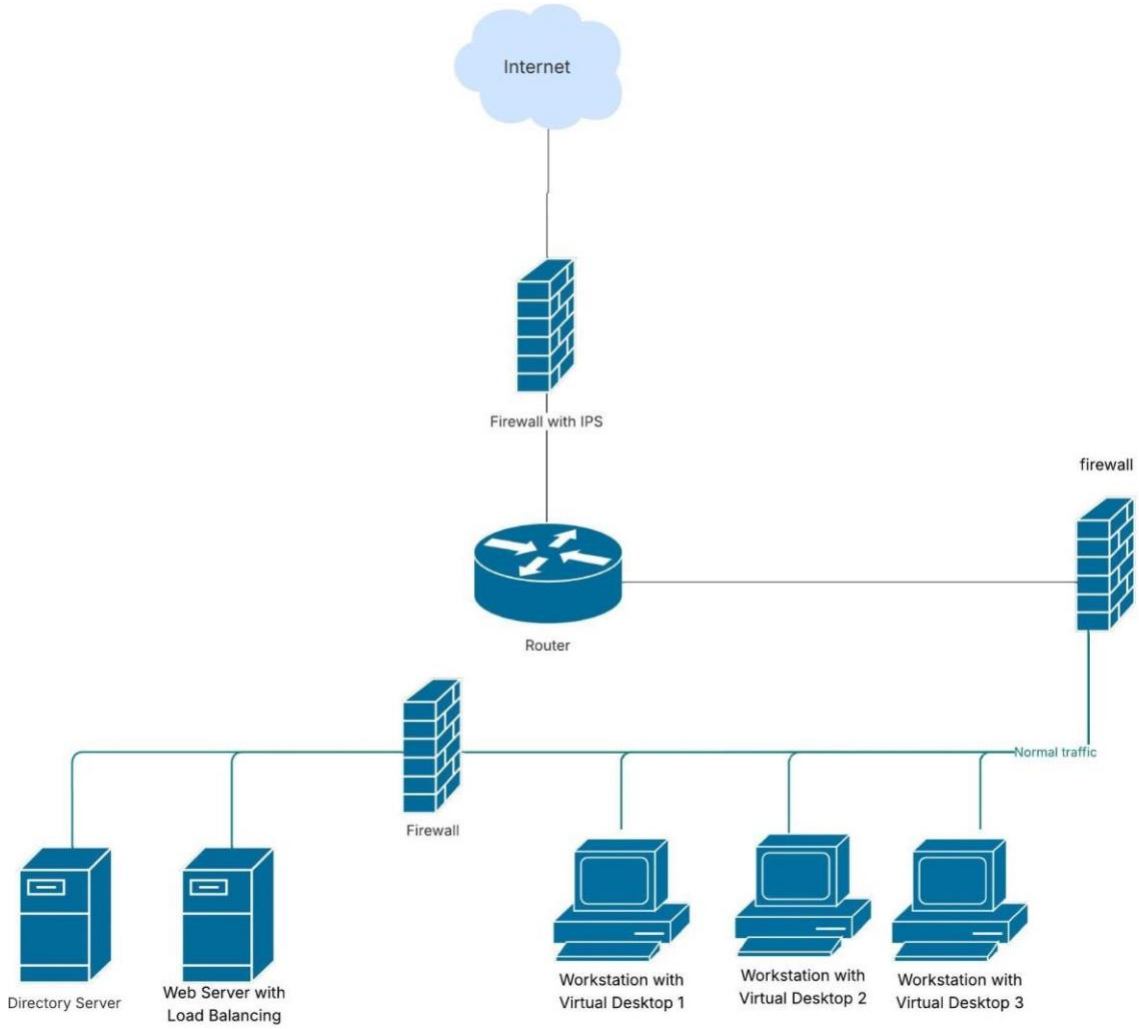
Building a small secure network

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

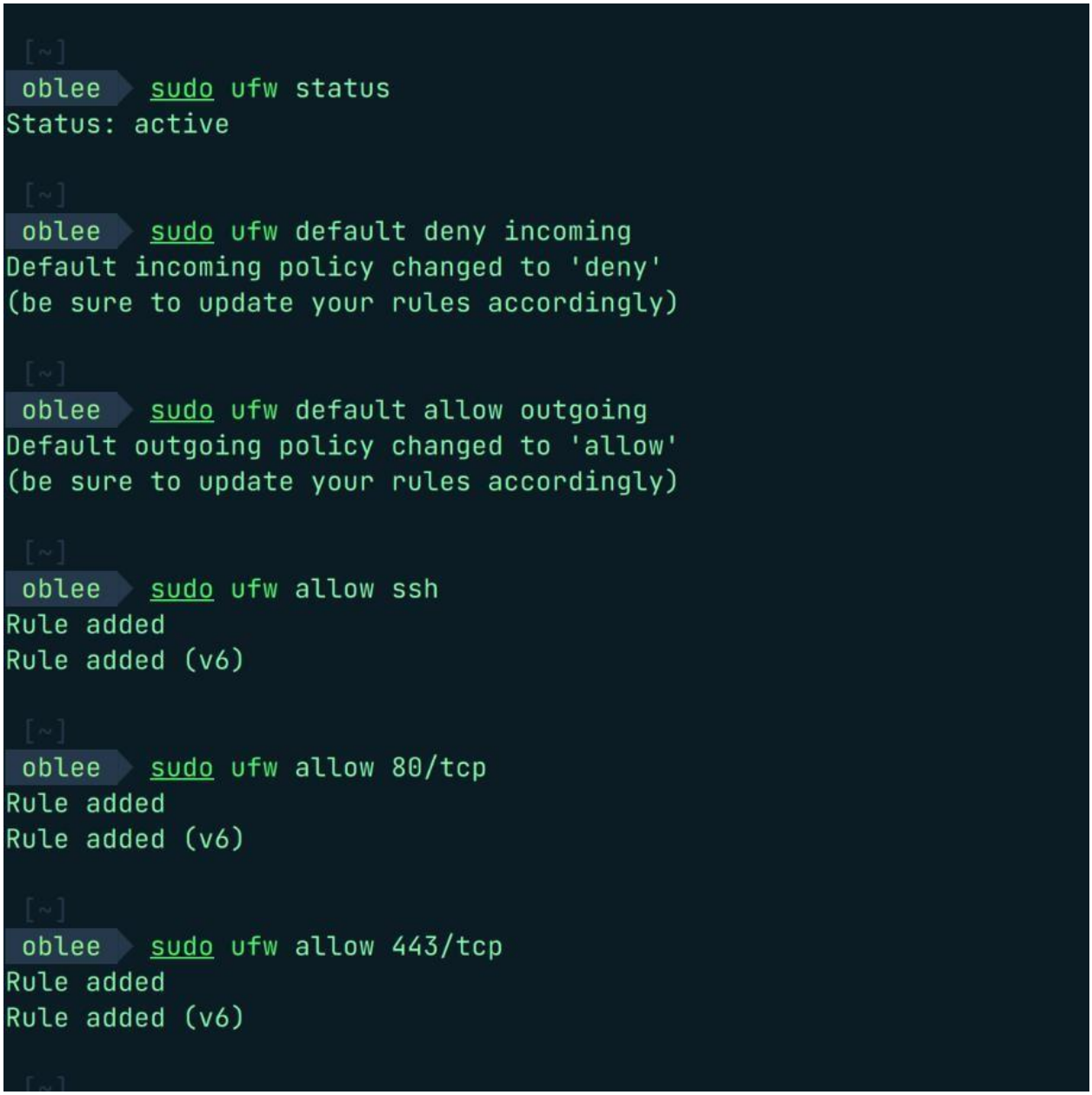


Figure 2.1: enabling UFW and setting up rules

Figure 2.2: Starting a simple web server on port 8080

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

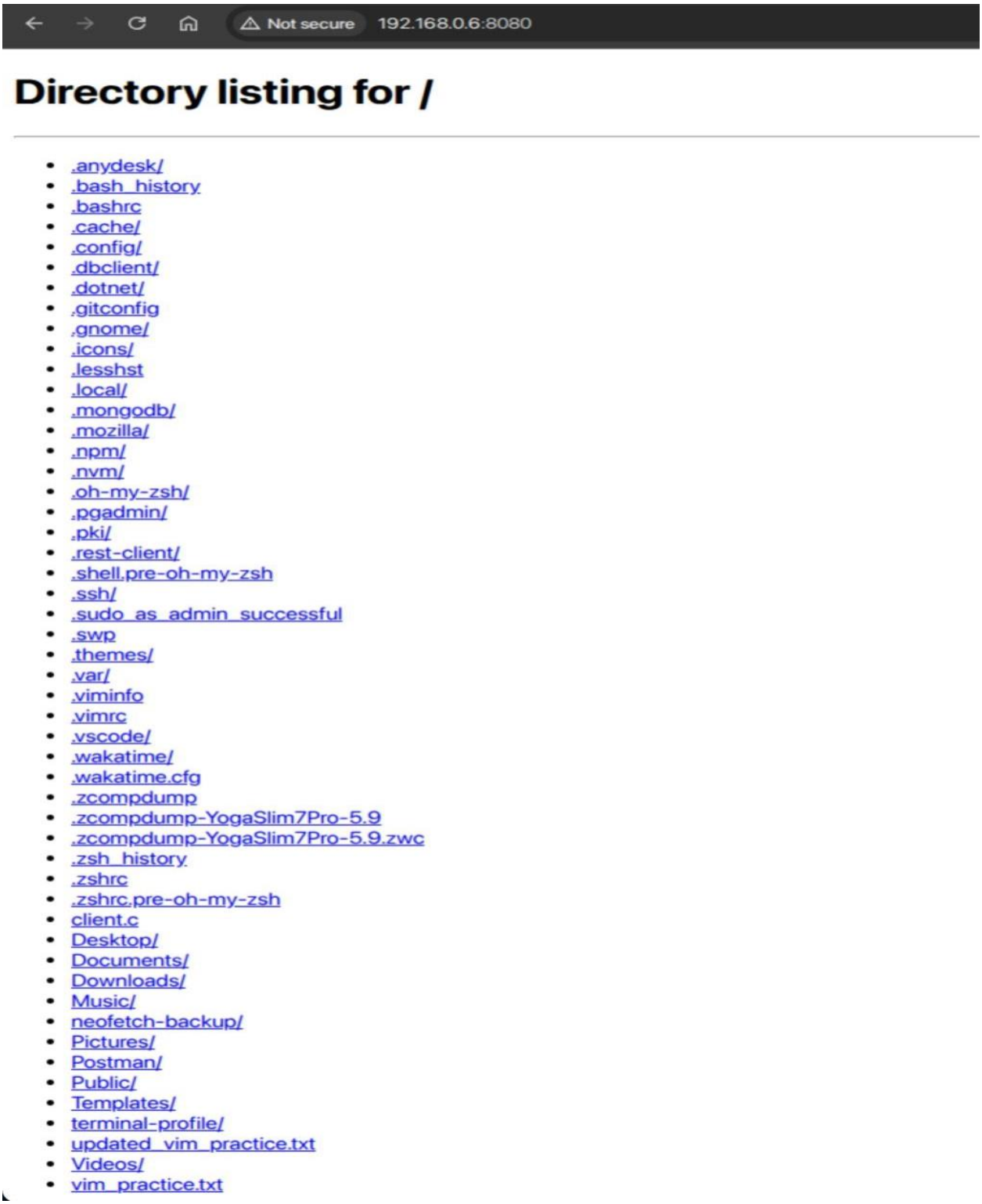


Figure 2.4: http://192.186.0.6:8080 in a browser before access is denied

```
[~]
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

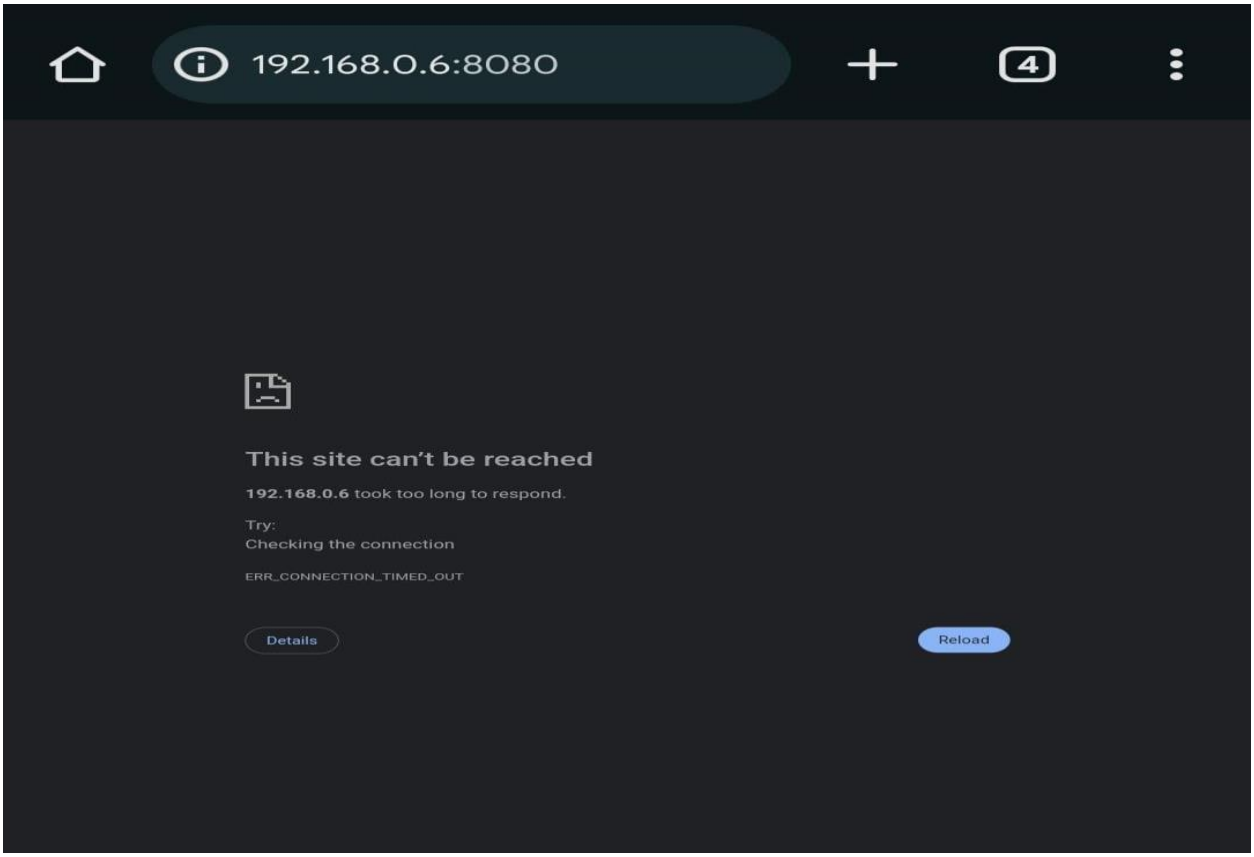


Figure 2.6: http://192.186.0.6:8080 in a browser after access is denied

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

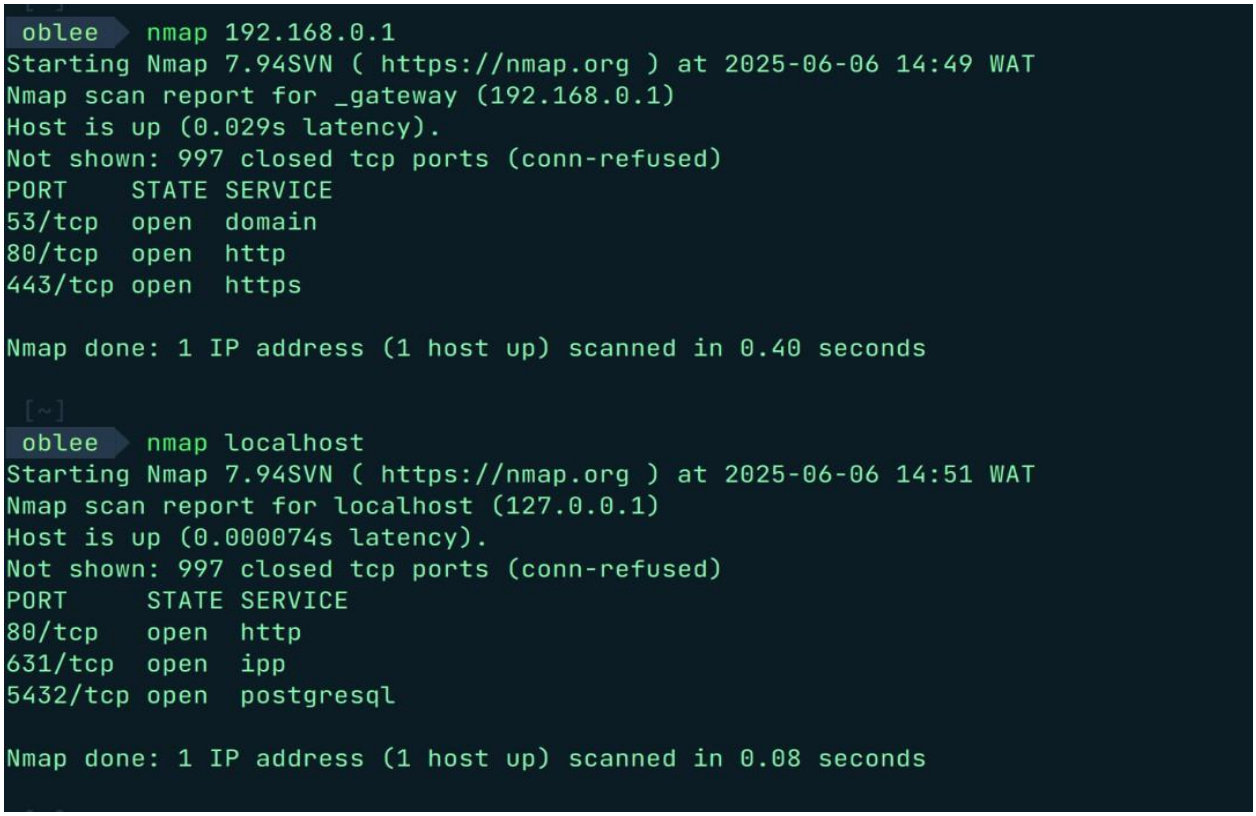


Figure 2.7: Port scanning using nMap

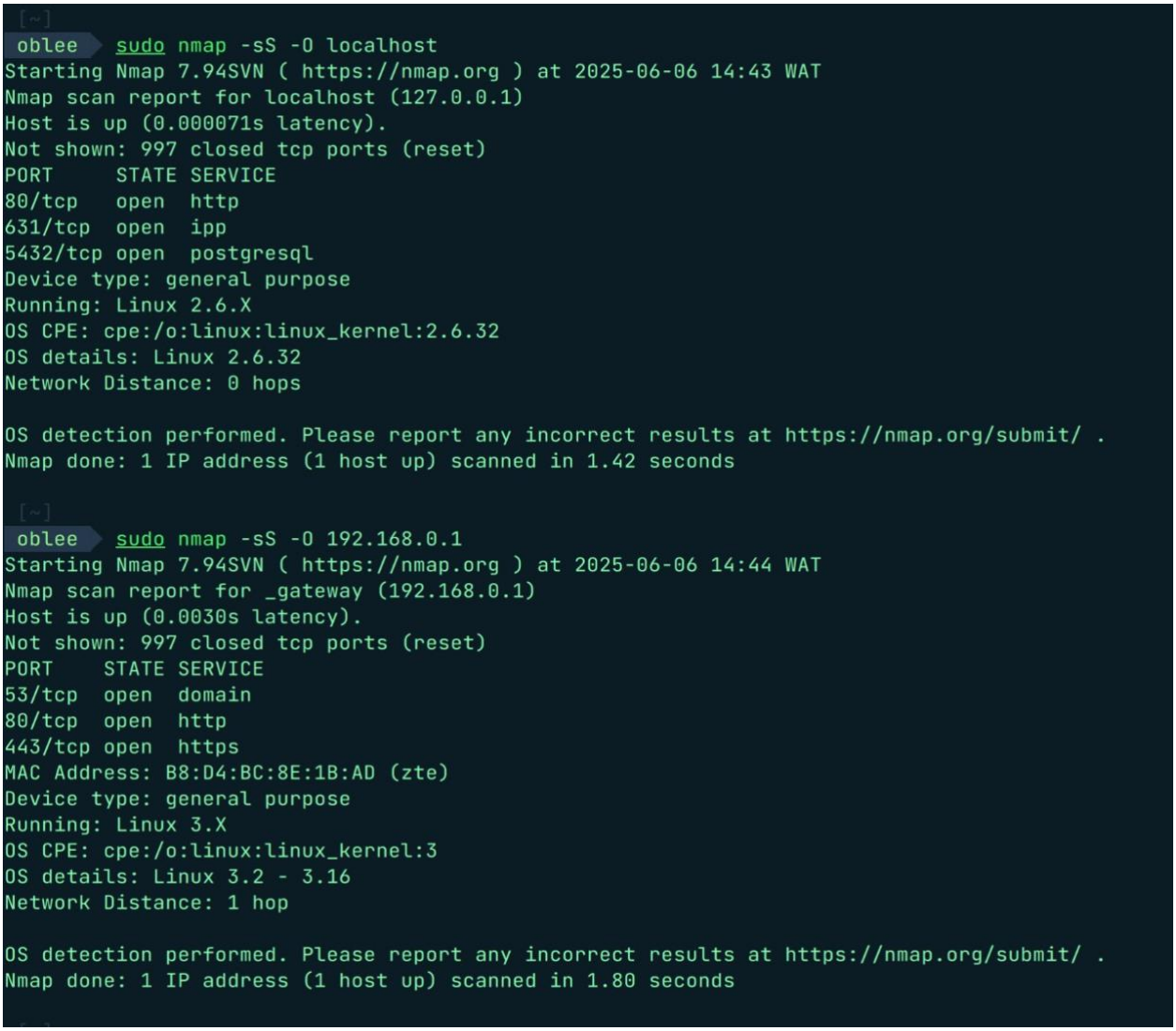


Figure 2.8 Checking to see what's running using nMap

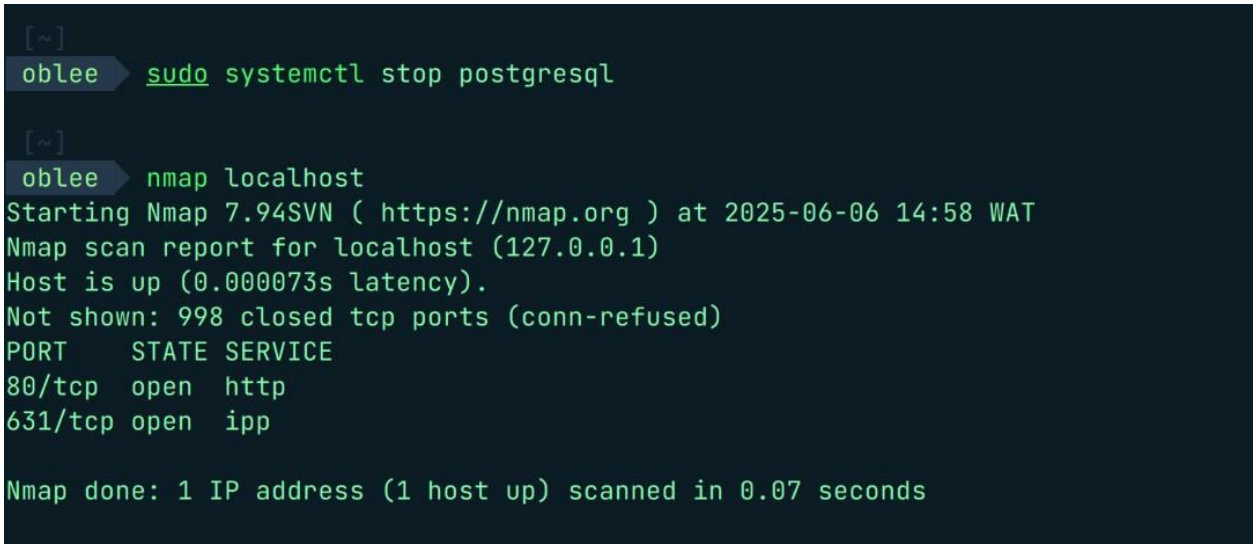
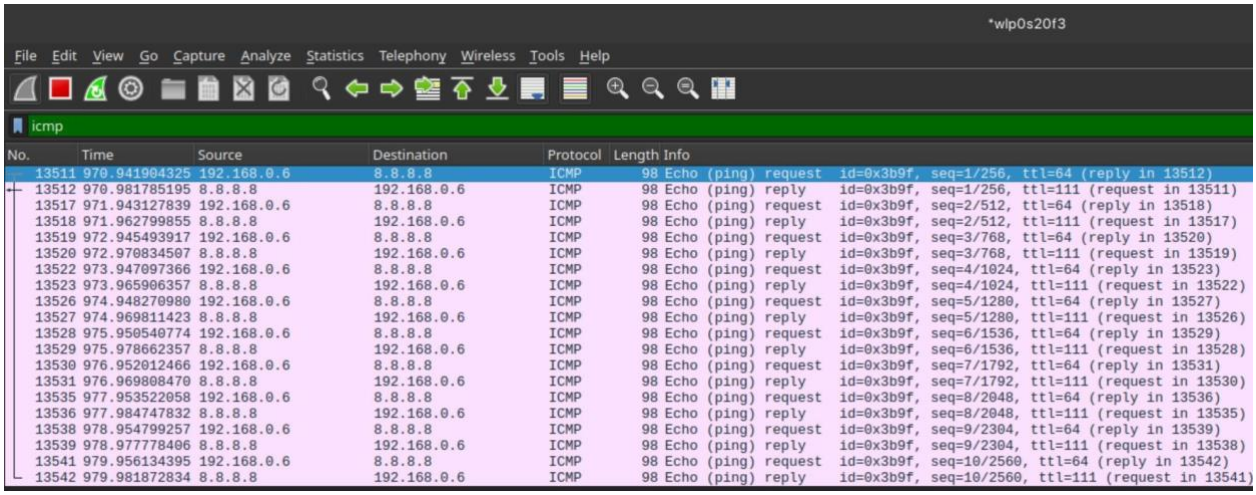


Figure 2.9: closing used ports

- 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.



- 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

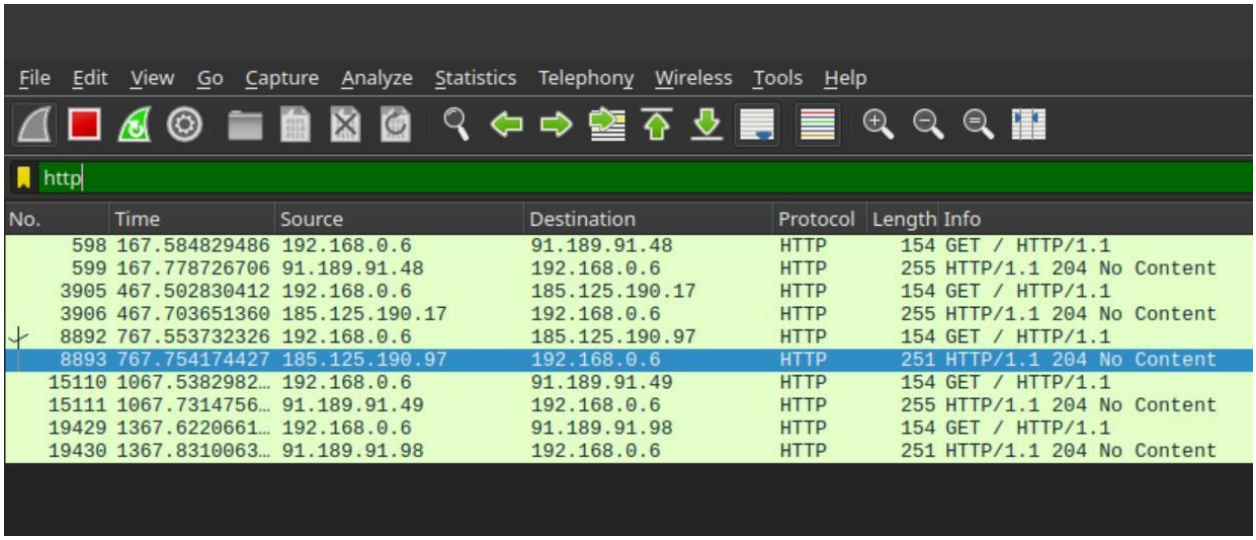
  - **Filtered packets** in Wireshark using: http

  - 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.

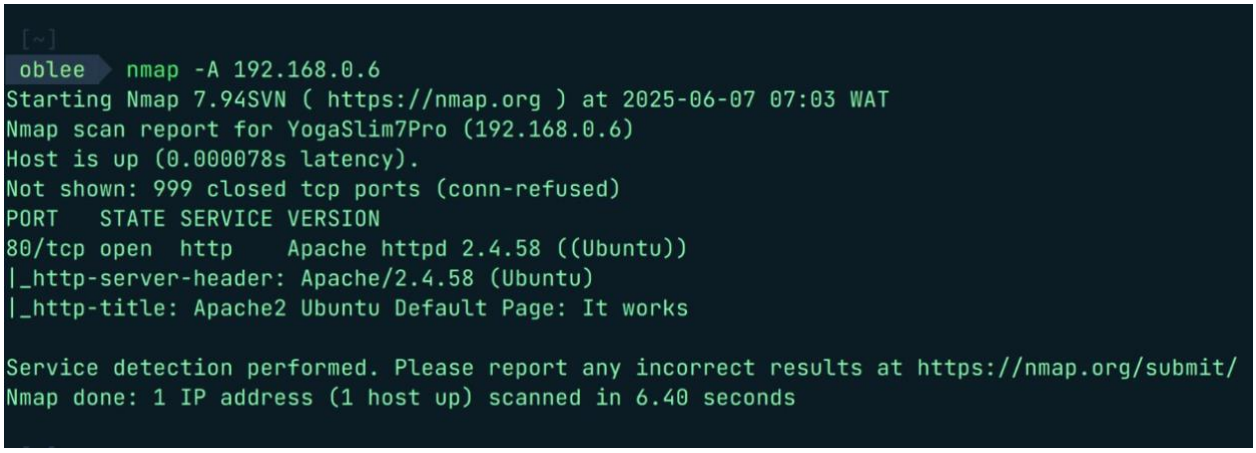


3. Simulated Threat & Response

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).



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.