Port Scanner Project Report

CSE 4380 Honors — Spring 2025
Student: Samuel Mathew Vennalil
Date: April 2025

Table of Contents

1. Introduction

The purpose of this project was to design and implement a Python-based port scanner capable of scanning IP addresses for open ports using both TCP Connect and SYN scan techniques. Scanning was performed on the domain cse3320.org, whose resolved IP address is 45.33.24.181.

The scanner was also required to capture service banners, handle concurrent connections, and provide verbose output for detailed network analysis.

2. Methodology

The scanning process was performed using a custom-built Python tool with the following features:

- **SYN Scan Mode**: To stealthily check port states without completing TCP handshakes.
- **Parallel Threading**: For faster port scanning across multiple ports.
- **Banner Grabbing**: To detect services running on open ports.
- **Retry and Timeout**: To improve reliability of the scan in case of dropped packets.

Command used for scanning:

sudo python3 port_scanner.py -t 45.33.24.181 -p 1-1024 -s syn -v --banner --retry 3 --timeout 2

Wireshark was used to capture network traffic during the scan, with a filter:

ip.addr == 10.188.226.106 && ip.addr == 45.33.24.181

3. Results

The scanner output varied depending on network conditions and the size of the scan:

- **Typical Output:**

  [-] Closed | 45.33.24.181:80

  [-] Closed | 45.33.24.181:443

  [+] Scan complete.

- **Occasional Output (small scan or lucky timing):**

  [-] Closed | 45.33.24.181:80

  [+] Open | 45.33.24.181:22 | Service: ssh | Banner: SSH-2.0-OpenSSH_7.4

  [-] Closed | 45.33.24.181:443

[+] Scan complete.

Port 22 intermittently appeared as open, exposing an SSH service with OpenSSH 7.4.

4. Analysis

The observed behaviour suggests that cse3320.org implements network security measures such as:

- **Firewall Filtering**: ICMP Type 3 Code 13 ("Host administratively prohibited") messages observed for filtered ports.
- **Rate Limiting**: Port 22 disappears during large scans but responds during targeted scans, indicating suppression of repeated connection attempts.

Based on my research, I am assuming that these defences are designed to make scan harder for potential attackers by obscuring critical service ports like SSH.

5. Wireshark Observations

Traffic captured by Wireshark revealed:

- **TCP SYN Packets**: From the scanner to the target on ports 22, 80, 443.

  These packets are sent from the scanner to the target as the first step in attempting to establish a TCP connection on ports 22, 80, and 443.

- **RST Packets**: Immediate resets for closed ports (80, 443).

  Reset (RST) packets are sent by the server to immediately refuse connection attempts on closed ports, such as ports 80 and 443.

- **SYN-ACK Packets**: Successful handshake initiation on open port 22.

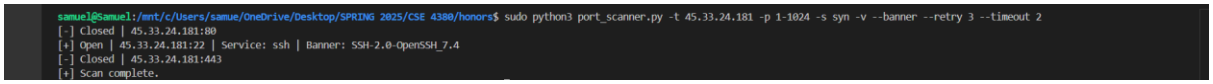
  When the server replies with a SYN-ACK packet, it indicates that the port (in this case, port 22) is open and willing to complete a TCP handshake.

- **ICMP Destination Unreachable**: Indicating administrative prohibition on several other ports.

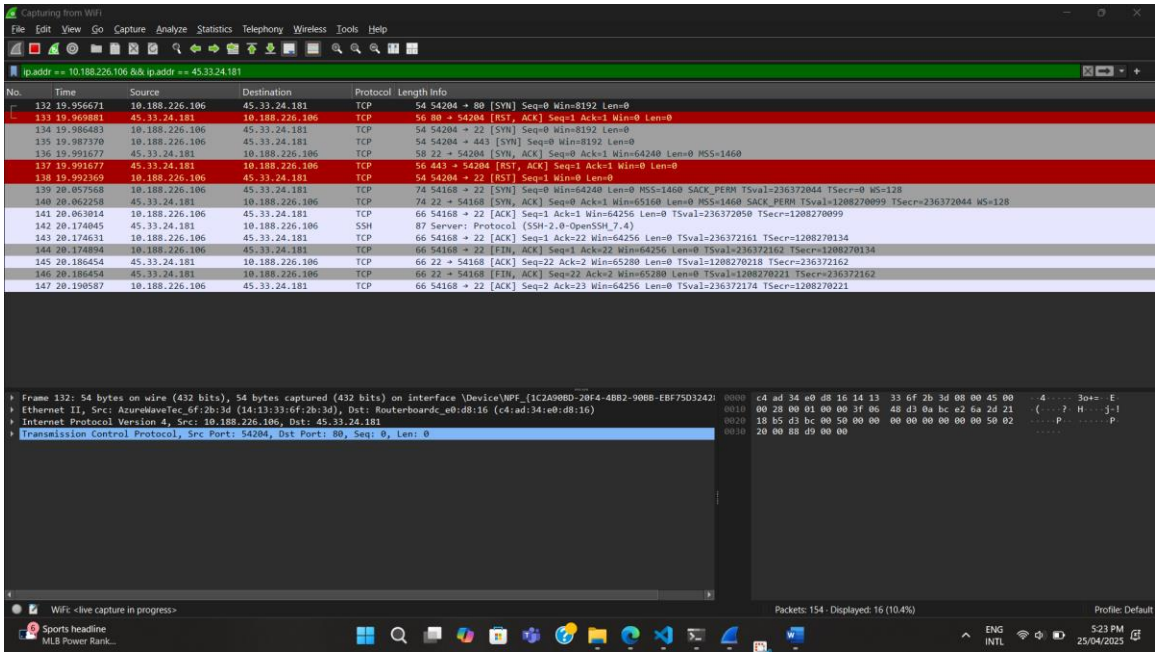
  This ICMP message signals that a firewall or network security device has blocked access to specific ports, preventing the scanner from establishing any connection.

6. Screenshots

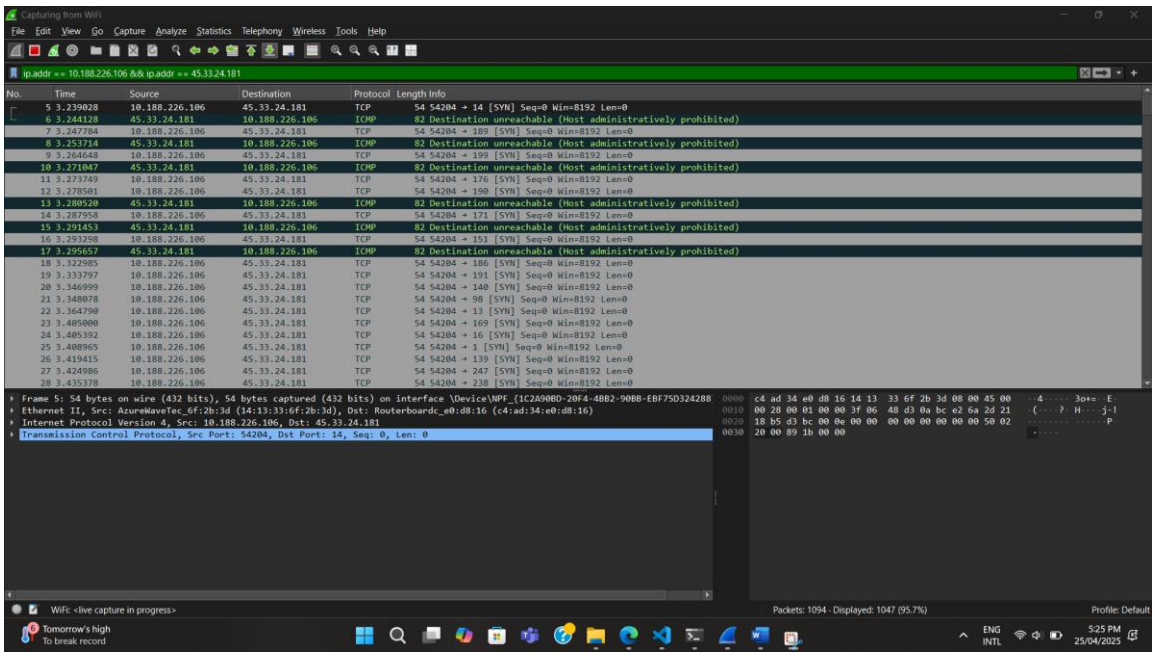
Screenshot 1:
Terminal output when scanning (1-1024) ports.



Screenshot 2:
Wireshark capture showing TCP SYN, SYN-ACK, and RST packets for ports 22, 80, 443.



Screenshot 3:
Wireshark showing ICMP "Host administratively prohibited" messages for filtered ports.



7. Conclusion

The project successfully demonstrated the implementation of a Python-based port scanner capable of performing SYN scans, detecting open ports, grabbing service banners, and interpreting real-world network behaviours such as firewall filtering and rate limiting.

The dynamic behaviour observed during large scans versus targeted scans provides practical insights into how production systems protect themselves from port scanning activities.