Internship Report: Port Scanner Using Python

Intern Name: Prem Kumar

Project Title: Port Scanner Using Python

Internship Organization: Inlighn Tech

Abstract

The "Port Scanner Using Python" project was developed as part of my internship at **Inlighn Tech** to explore how open ports on a target system can expose critical services to cyber threats. This tool is designed to scan a range of TCP ports on a specified target IP address, identify which ports are open, and attempt to retrieve any associated service banners. The tool utilizes **Python's `socket` library** and **multithreading (via `ThreadPoolExecutor`)** for enhanced speed and performance. It serves as a foundational component for ethical hacking and penetration testing toolkits.

1. Introduction

In cybersecurity, **port scanning** is a vital reconnaissance technique used by both attackers and security analysts to discover active services on a networked device. Each service usually runs on a specific port, and discovering these ports helps analysts understand the attack surface of a system.

This project aimed to build a customizable Python script that performs multi-threaded TCP port scanning, banner grabbing, and presents a clean, structured output of open services. The scanner is intended for educational and internal network auditing purposes, adhering to ethical hacking principles.

2. Tools and Technologies Used

Tool/Library	Purpose

Python 3.x	Core language
`socket` module	Creating TCP socket connections
`concurrent.futures`	Handling multithreading using ThreadPoolExecutor
`sys`	Reading command-line arguments
Command Prompt / Linux Terminal	Executing and testing the scanner

3. Objectives

- Create a Python tool to scan open TCP ports on a remote host
- Use banner grabbing to identify exposed services
- Speed up scanning using multithreading
- Display results in a clean, readable format
- Understand real-world use of port scanners in ethical hacking
- Handle exceptions and errors gracefully

4. How the Tool Works

4.1 Target Input

The user is prompted to input:

- Target IP address (e.g., `192.168.1.10`)
- Start and end ports to define the scanning range (e.g., `20–1000`)

4.2 TCP Connection Attempt

For each port in the given range, the script attempts to establish a TCP connection using `socket.connect_ex()`. If the connection is successful, the port is marked as **open**.

4.3 Banner Grabbing

If the port is open, the tool sends a dummy request (or waits for a response) and captures any returned text. This is known as a **service banner**, which can reveal software names and versions.

4.4 Multithreading for Speed

Using `ThreadPoolExecutor`, the scanner scans multiple ports concurrently, significantly improving performance over sequential scanning.

4.5 Output Formatting

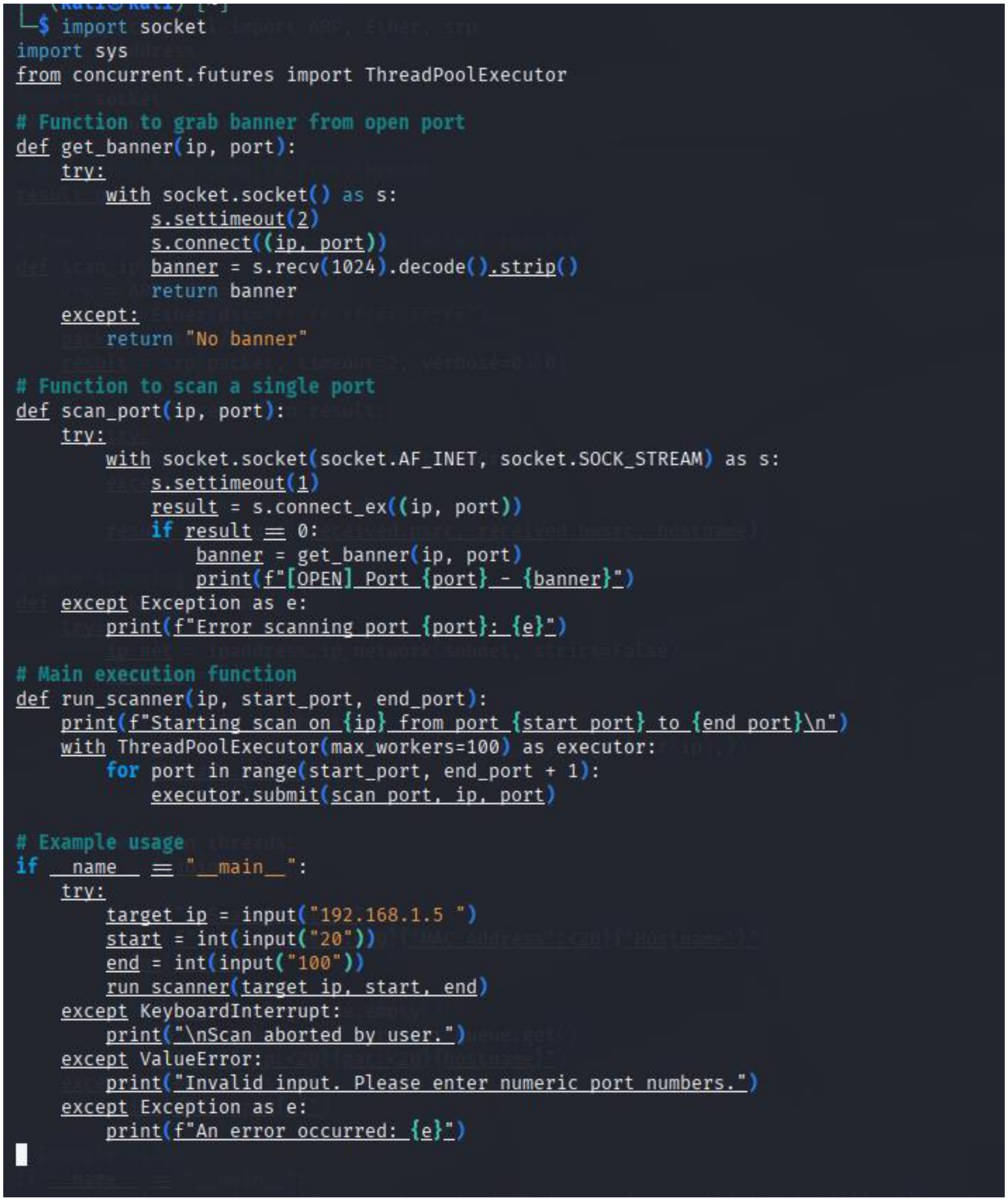
The tool neatly prints a list of open ports along with detected services or banners.

5. Python Implementation

Installation Requirements

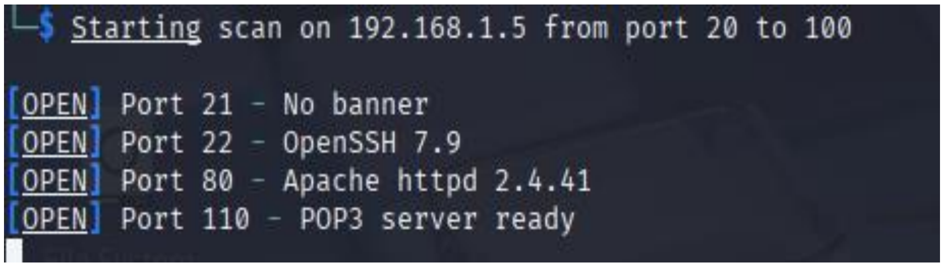
No external libraries are required. Python's standard library is sufficient.

Complete Code



```
└─$ import socket
import sys
from concurrent.futures import ThreadPoolExecutor

# Function to grab banner from open port
def get_banner(ip, port):
    try:
        with socket.socket() as s:
            s.settimeout(2)
            s.connect((ip, port))
            banner = s.recv(1024).decode().strip()
            return banner
    except:
        return "No banner"

# Function to scan a single port
def scan_port(ip, port):
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.settimeout(1)
            result = s.connect_ex((ip, port))
            if result == 0:
                banner = get_banner(ip, port)
                print(f"[OPEN] Port {port} - {banner}")
    except Exception as e:
        print(f"Error scanning port {port}: {e}")

# Main execution function
def run_scanner(ip, start_port, end_port):
    print(f"Starting scan on {ip} from port {start_port} to {end_port}\n")
    with ThreadPoolExecutor(max_workers=100) as executor:
        for port in range(start_port, end_port + 1):
            executor.submit(scan_port, ip, port)

# Example usage
if __name__ == "__main__":
    try:
        target_ip = input("192.168.1.5 ")
        start = int(input("20"))
        end = int(input("100"))
        run_scanner(target_ip, start, end)
    except KeyboardInterrupt:
        print("\nScan aborted by user.")
    except ValueError:
        print("Invalid input. Please enter numeric port numbers.")
    except Exception as e:
        print(f"An error occurred: {e}")
```

6. Output



```
└─$ Starting scan on 192.168.1.5 from port 20 to 100

[OPEN] Port 21 - No banner
[OPEN] Port 22 - OpenSSH 7.9
[OPEN] Port 80 - Apache httpd 2.4.41
[OPEN] Port 110 - POP3 server ready
```

7. Key Cybersecurity Concepts Covered

Concept	Description
Port Scanning	Identifies services running on different ports
Banner Grabbing	Reveals information about the software in use
Reconnaissance	First phase in ethical hacking to map attack surface
Multithreading	Improves performance for scanning large port ranges
TCP Protocol	Used to establish reliable socket connections

8. Learning Outcomes

Through this project, I gained:

- Practical skills in **Python socket programming**
- Ability to work with **concurrent executions**
- Understanding of how open ports expose vulnerabilities
- Insight into basic penetration testing techniques
- Awareness of common services and banners found on networks

9. Limitations

- Only supports TCP scanning (no UDP)
- No support for saving output to a file

- No OS fingerprinting or vulnerability scoring
- Cannot handle firewalled or filtered ports effectively

10. Future Enhancements

To expand this project, I could:

- Implement **UDP scanning** using raw sockets
- Add **output logging to text or CSV files**
- Develop a **Tkinter GUI interface**
- Integrate **nmap or masscan APIs** for deeper scans
- Add **OS detection** using TCP/IP stack fingerprinting

11. Real-World Applications

- Internal network auditing for open and insecure services
- Early-stage reconnaissance during penetration testing
- Teaching basic cybersecurity principles
- Asset discovery in unmanaged IT environments
- Automating routine security checks in DevOps pipelines

12. Ethical & Legal Considerations

Port scanning should only be performed on **networks you own or have written permission to test**. Unauthorized scanning can be considered illegal under various cybercrime laws. This tool was built and tested in a lab setup with full authorization, for **educational purposes only**.

13. Conclusion

The **Port Scanner Using Python** is a foundational project for anyone aspiring to work in cybersecurity. It demonstrates how simple scripts can reveal critical network information and highlights the importance of regularly auditing open ports in any environment. Through this internship task, I acquired real-world programming and security analysis experience that directly supports career paths in ethical hacking, red teaming, and DevSecOps.