INLIGHN TECH

EXPERIENCE. LEARN. THRIVE

Prerequisites for the Port Scanner Project

By: **Inlighn Tech**

Overview

The Port Scanner project involves creating a Python script that scans a target host for open ports within a specified range, identifying services and grabbing banners where possible. The solution uses the socket library for network communication, concurrent.futures for multithreaded scanning, and ANSI color codes for formatted output, with progress tracking via sys.stdout. To write this script independently, students must master several key concepts and skills from the curriculum. These prerequisites, paired with practical Python code examples, ensure students understand how port scanning works and can implement the script effectively.

Prerequisite Knowledge and Skills

1. **Networking Basics – Ports and Protocols**
    - **What to Know**:
        - Ports (0-65535) are endpoints for network communication; common ones include 80 (HTTP), 22 (SSH), and 443 (HTTPS).

        - TCP connects via a handshake; open ports accept connections, while closed ones reject them.
        - Port scanning tests these endpoints to identify active services.
    - **How It Applies**:
        - The script uses socket.connect_ex() to check port status and getservbyport() to identify services on open ports.
    - **Curriculum Source**:
        - "NETWORKING, SOCKETS, AND CYBERSECURITY" PDFs (e.g., "Understanding Ports, IP Addresses, and Protocols in Cybersecurity.pdf").
    - **Practical Prep**:
        - Use telnet 127.0.0.1 22 or nc -zv 127.0.0.1 22 to manually test a port (if SSH is running locally).
    - **Python Code Block**:

```
# Check if a single port is open
import socket

target = "127.0.0.1"  # Localhost
port = 22  # SSH port
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.settimeout(1)
result = sock.connect_ex((target, port))
if result == 0:
    print(f"Port {port} is open")
else:
    print(f"Port {port} is closed")
sock.close()
```

        - **Run It**: Save as port_test.py, run with python port_test.py (ensure an SSH server runs on localhost). Introduces port checking, like scan_port().

2. **Socket Programming**
    - **What to Know**:

        - The socket module creates network sockets; AF_INET uses IPv4, SOCK_STREAM uses TCP.
        - connect_ex() returns 0 for success (open port) or an error code (closed); recv() grabs banners from open ports.
        - gethostbyname() resolves hostnames to IPs; getservbyport() maps ports to service names.
    - **How It Applies**:
        - The script uses sockets to test ports, resolve the target IP, identify services, and fetch banners.
    - **Curriculum Source**:
        - "NETWORKING, SOCKETS, AND CYBERSECURITY" PDF ("Introduction to Sockets.pdf").
    - **Practical Prep**:
        - Practice socket basics.
    - **Python Code Block**:

```python
# Resolve hostname and test a port with banner
import socket

host = "example.com"
port = 80  # HTTP
ip = socket.gethostbyname(host)
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.settimeout(1)
if sock.connect_ex((ip, port)) == 0:
    banner = sock.recv(1024).decode().strip()
    print(f"IP: {ip}, Port {port} open, Banner: {banner}")
else:
    print(f"Port {port} closed on {ip}")
sock.close()
```

        - **Run It**: Save as socket_test.py, run with python socket_test.py. Shows IP resolution and banner grabbing, like in port_scan() and get_banner().

3. **Exception Handling**
    - **What to Know**:

        - Use try/except to manage network errors (e.g., timeouts, decode failures).
        - Return defaults (e.g., empty strings) when operations fail.
    - **How It Applies**:
        - scan_port() and get_banner() catch connection and banner retrieval errors to keep scanning robust.
    - **Curriculum Source**:
        - "PYTHON BASICS" Lesson 21 (Errors and Exception Handling).
    - **Practical Prep**:
        - Test error handling with sockets.
    - **Python Code Block**:

```
# Handle socket errors
import socket

target = "127.0.0.1"
port = 9999  # Unlikely to be open
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.settimeout(1)
try:
    sock.connect_ex((target, port))
    banner = sock.recv(1024).decode()
    print(f"Banner: {banner}")
except socket.timeout:
    print(f"Timeout on port {port}")
except:
    print(f"Failed to get banner on port {port}")
sock.close()
```

        - **Run It**: Save as error_test.py, run with python error_test.py. Prepares for error handling in scan_port().

4. **Multithreading with concurrent.futures**
    - **What to Know**:
        - concurrent.futures.ThreadPoolExecutor runs tasks in parallel; submit() schedules tasks, as_completed() retrieves results.
        - max_workers controls thread count for efficiency.

- **How It Applies**:
  - The script uses ThreadPoolExecutor to scan multiple ports concurrently, speeding up the process.
- **Curriculum Source**:
  - "Port Scanner (Solution)" Lesson 3 ("Working with Threads.mp4"); extends "Network Scanner" Lesson 4.
- **Practical Prep**:
  - Practice thread pools.
- **Python Code Block**:

```
# Use ThreadPoolExecutor for parallel tasks
from concurrent.futures import ThreadPoolExecutor
import time

def task(n):
    time.sleep(0.5)  # Simulate work
    return f"Task {n} done"

with ThreadPoolExecutor(max_workers=2) as executor:
    futures = [executor.submit(task, i) for i in range(4)]
    for future in executor.as_completed(futures):
        print(future.result())
```

  - **Run It**: Save as threadpool_test.py, run with python threadpool_test.py. Demonstrates parallel execution, like in port_scan().

5. **String Formatting and ANSI Colors**
   - **What to Know**:
     - Use .format() or f-strings for structured output; ANSI escape codes (e.g., \033[91m) colorize terminal text.
     - Align text with padding (e.g., {:<8}).
   - **How It Applies**:
     - format_port_results() creates a table with colored output for open ports and banners.
   - **Curriculum Source**:
     - "WORKING WITH DATA STRUCTURES & FILE HANDLING" Lesson 2 (Print Formatting with Strings).

        - Note: ANSI colors may need a brief intro.
    - **Practical Prep**:
        - Test formatted output with colors.
    - **Python Code Block**:

```
# Format output with ANSI colors
RED = "\033[91m"
RESET = "\033[0m"

print("{:<10} {:<10}".format("Name", "Status"))
print("-" * 20)
print(f"{RED}Alice{RESET:<10} {'Active':<10}")
```

        - **Run It**: Save as format_test.py, run with python format_test.py (works best in a terminal supporting ANSI, e.g., Linux/macOS). Prepares for format_port_results().

6. **Progress Tracking with sys.stdout**
    - **What to Know**:
        - sys.stdout.write() and flush() update console output dynamically (e.g., progress counters).
        - \r returns the cursor to the line start for overwriting.
    - **How It Applies**:
        - The script shows scan progress with a dynamic counter.
    - **Curriculum Source**:
        - Not explicitly taught; assumes basic Python I/O knowledge.
    - **Practical Prep**:
        - Test dynamic output.
    - **Python Code Block**:

```
# Dynamic progress display
import sys
import time

for i in range(5):
    sys.stdout.write(f"\rProgress: {i+1}/5")
    sys.stdout.flush()
    time.sleep(0.5)
print("\nDone")
```

- **Run It**: Save as progress_test.py, run with python progress_test.py. Shows progress updates, like in port_scan().

How Port Scanning Works

- **Concept**:
  - Port scanning tests a range of ports on a target host to identify open ones, revealing running services (e.g., web servers, SSH).
  - This is a reconnaissance technique to assess a target's attack surface.
- **Script Workflow**:
  - Accept target hostname and port range from user input.
  - Resolve hostname to IP with gethostbyname().
  - Use ThreadPoolExecutor to scan ports in parallel with scan_port().
  - For open ports, identify services and grab banners.
  - Format and display results with progress updates.
- **Why Multithreading?**:
  - Speeds up scanning by checking multiple ports simultaneously, essential for large ranges.

How to Write the Port Scanner Script

Using these prerequisites, students can build the script step-by-step:

1. **Setup**:
   - Import socket, concurrent.futures, and sys. Define ANSI color codes.
2. **Scan Function**:

   - Write scan_port() to test ports with connect_ex(), get services with getservbyport(), and fetch banners with recv().
3. **Banner Function**:
   - Create get_banner() to retrieve data from open ports, handling errors.
4. **Format Results**:
   - Define format_port_results() to build a colored table.
5. **Main Scan**:
   - Implement port_scan() to resolve IP, launch threads, track progress, and display results.
6. **Entry Point**:
   - Use if __name__ == "__main__": to prompt for inputs and run port_scan().

Notes for Students

- **Setup**: No external libraries needed beyond Python's standard library. Test on a local machine (e.g., 127.0.0.1) or a known host with open ports (e.g., scanme.nmap.org, ports 1-100).
- **Engagement**: Run each code block to practice socket connections, threading, and formatting. Start with a small range (e.g., 1-10) to see quick results.
- **Tips**: Ensure your firewall allows outbound connections; some ports may be filtered by the target.

Jump to the Port Scanner Project Description if you are ready now.