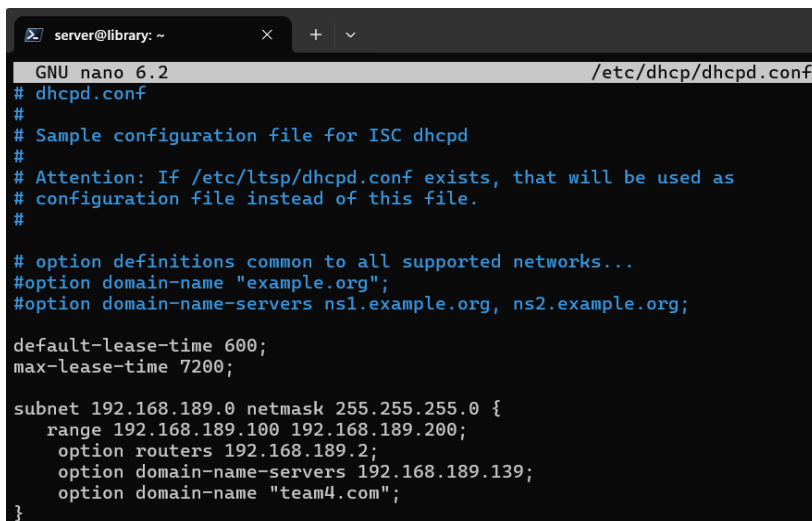


# Python: Create a port scanner

Team 2- Nsapi, Eliyar, Glen, Pacifique

## Prerequisites:

- 1.To perform TCP SYN scanning in Python, we'll need to use the socket library to create TCP socket connections and send SYN packets, and the struct library to craft and parse raw network packets.
- 2.Let's Understand ARP (Address Resolution Protocol) and ICMP (Internet Control Message Protocol) and how they are used in scanning:  
  
Address Resolution Protocol (ARP): ARP is a communication protocol used for discovering the hardware address (MAC address) associated with a given IPv4 address on a local area network (LAN). It is commonly used in Ethernet networks to map IP addresses to MAC addresses.  
  
Internet Control Message Protocol (ICMP):ICMP is a network layer protocol used to send control messages and provide feedback about the health and operation of an IP network. It is commonly used for diagnostic and error reporting purposes, as well as for network management tasks.
- 3.TCP SYN scanning, also known as SYN scanning or half-open scanning, TCP SYN scanning is a type of TCP port scanning technique that leverages the three-way handshake process of the TCP protocol.
- 4.Creating a range of IP addresses within your local network subnet:



```
server@library: ~  
GNU nano 6.2 /etc/dhcp/dhcpd.conf  
# dhcpd.conf  
#  
# Sample configuration file for ISC dhcpd  
#  
# Attention: If /etc/ltsp/dhcpd.conf exists, that will be used as  
# configuration file instead of this file.  
#  
# option definitions common to all supported networks...  
#option domain-name "example.org";  
#option domain-name-servers ns1.example.org, ns2.example.org;  
  
default-lease-time 600;  
max-lease-time 7200;  
  
subnet 192.168.189.0 netmask 255.255.255.0 {  
    range 192.168.189.100 192.168.189.200;  
    option routers 192.168.189.2;  
    option domain-name-servers 192.168.189.139;  
    option domain-name "team4.com";  
}
```

## Write a detailed software design report:

**\*\*Software Design Report: Python Network Scanner\*\***

### 1. **\*\*Service Name and Version:\*\***

- The service name refers to the type of service or application running on a specific port. For example, port 80 is commonly associated with the HTTP service used for web servers.
- The service version refers to the specific version of the service or application running on the port. For example, the HTTP service running on port 80 might be Apache HTTP Server version 2.4.18.
- Nmap retrieves this information by sending specific probes or queries to the open port and analyzing the responses received. These responses often contain banners or headers that identify the service and its version.

### 2. **\*\*Operating System Detection:\*\***

- In addition to identifying open ports and services, Nmap can also attempt to detect the operating system (OS) running on the target system.
- OS detection is performed by analyzing subtle differences in the way the target system responds to various network probes and packets.
- Nmap uses a combination of techniques, including TCP/IP stack fingerprinting and analysis of responses to specific probes, to determine the likely operating system running on the target.

### 3. **\*\*Additional Details:\*\***

- PC: VMware Workstation Pro
- OS: Kali Linux
- Software: Visual Studio Code + Python3

### **\*\*1. Introduction:\*\***

This report outlines the design of a Python-based network scanner script using the Nmap library. The script is designed to automate the process of scanning a target IP address for open ports using various scan types provided by Nmap.

### **\*\*2. Requirements Analysis:\*\***

#### **\*\*Functional Requirements:\*\***

- VM Kali Linux OS
- VS code + python3 programming language
- Accept user input for the target IP address from private network
- Allow users to choose the type of scan to perform (SYN ACK, UDP, Comprehensive).
- Utilize the Nmap library to perform the selected scan.
- Display scan results including Nmap version, scan information, IP status, and open ports.

#### **\*\*Non-functional Requirements:\*\***

- User-friendly interface for input and output.
- Robust error handling for invalid user input or scan errors.
- Compatibility with Python 3.x.
- Efficient utilization of system resources during the scanning process.

### **\*\*3. Architecture Design:\*\***

#### **\*\*3.1. High-Level Overview:\*\***

The network scanner script follows a procedural design approach with a clear separation of concerns between user interface, scan execution, and result presentation.

#### **\*\*3.2. Component Design:\*\***

- **\*\*User Interface:\*\*** Handles user input for the target IP address and scan type selection.
- **\*\*Scan Execution:\*\*** Utilizes the Nmap library to perform the selected scan type on the target IP address.
- **\*\*Result Presentation:\*\*** Formats and displays the scan results to the user in a clear and informative manner.

#### **\*\*3.3. External Libraries:\*\***

The script relies on the Nmap library for performing network scans. Nmap is a widely-used open-source tool for network exploration and security auditing.

### **\*\*4. Detailed Design:\*\***

#### **\*\*4.1. User Interface:\*\***

- Prompt the user to enter the target IP address.
- Present options for selecting the type of scan to perform (SYN ACK, UDP, Comprehensive).
- Validate user input and handle errors gracefully.

#### **\*\*4.2. Scan Execution:\*\***

- Utilize the Nmap library to perform the selected scan type on the target IP address.
- Pass appropriate arguments to Nmap based on the selected scan type.
- Capture and process the scan results for further presentation.

#### **\*\*4.3. Result Presentation:\*\***

- Display the Nmap version used for the scan.
- Present scan information including the scan type, protocol, and scanned ports.
- Display the status of the target IP address (up or down).
- Show a list of open ports detected during the scan.

### **\*\*5. Implementation Plan:\*\***

#### **\*\*5.1. Technologies:\*\***

- Python programming language (version 3.x).
- Nmap library for Python (`python3-nmap` package).

#### **\*\*5.2. Development Phases:\*\***

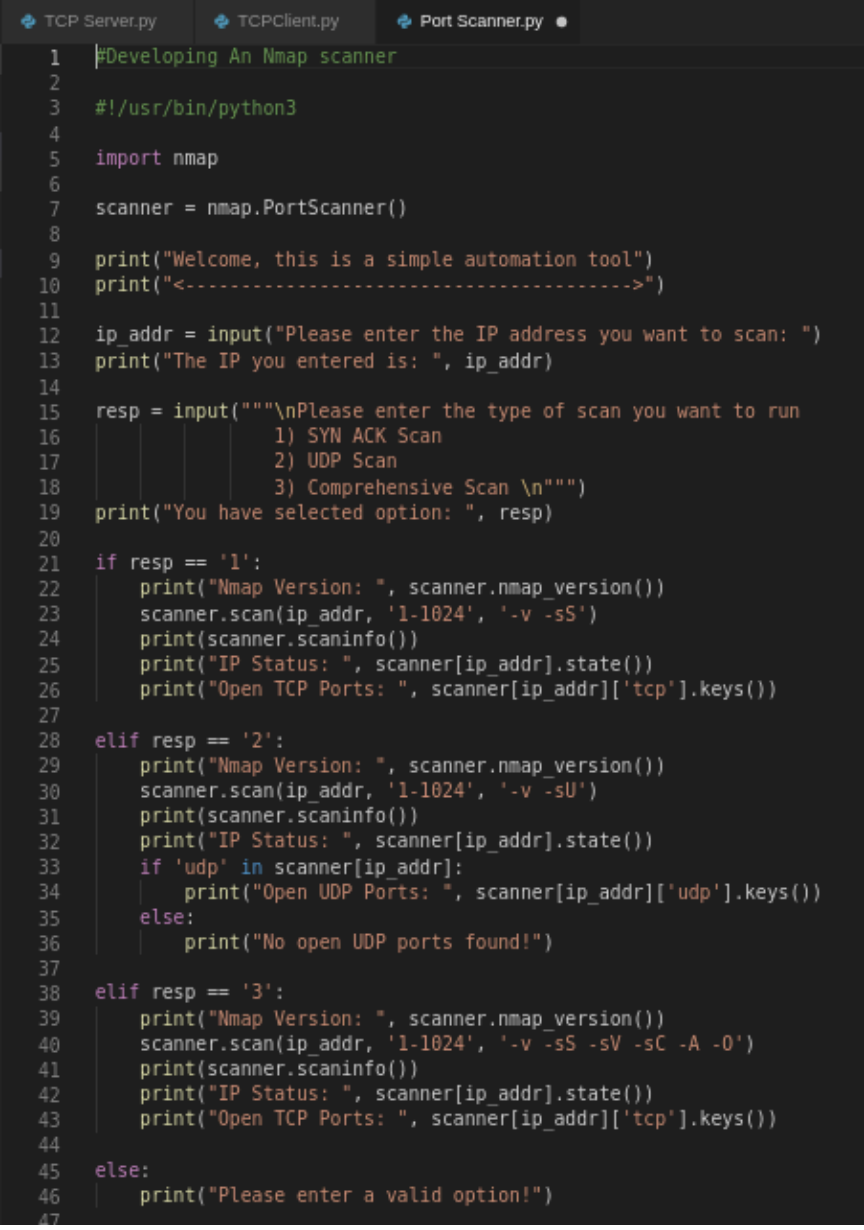
- **\*\*Phase 1: User Interface\*\***
  - Implement user input for target IP address and scan type selection.
- **\*\*Phase 2: Scan Execution \*\***
  - Integrate Nmap library for performing network scans.
  - Implement scan execution logic based on user-selected scan type.
- **\*\*Phase 3: Result Presentation \*\***
  - Format and display scan results to the user.
  - Handle errors and edge cases in result presentation.
- **\*\*Phase 4: Testing and Refinement\*\***

- Conduct comprehensive testing of the script.
- Refine user interface, scan execution, and result presentation based on feedback.

## **\*\*6. Conclusion:\*\***

The design outlined in this report provides a structured approach to developing a Python network scanner script using the Nmap library. By following a clear separation of concerns and implementing user-friendly interfaces, the script aims to provide an efficient and reliable tool for network exploration and security auditing. With further development and testing, the network scanner script will be a valuable asset for network administrators and security professionals.

### **Python Code: scan single IP address**



```

TCP Server.py TCPClient.py Port Scanner.py
1  #Developing An Nmap scanner
2
3  #!/usr/bin/python3
4
5  import nmap
6
7  scanner = nmap.PortScanner()
8
9  print("Welcome, this is a simple automation tool")
10 print("<----->")
11
12 ip_addr = input("Please enter the IP address you want to scan: ")
13 print("The IP you entered is: ", ip_addr)
14
15 resp = input("""\nPlease enter the type of scan you want to run
16              1) SYN ACK Scan
17              2) UDP Scan
18              3) Comprehensive Scan \n""")
19 print("You have selected option: ", resp)
20
21 if resp == '1':
22     print("Nmap Version: ", scanner.nmap_version())
23     scanner.scan(ip_addr, '1-1024', '-v -sS')
24     print(scanner.scaninfo())
25     print("IP Status: ", scanner[ip_addr].state())
26     print("Open TCP Ports: ", scanner[ip_addr]['tcp'].keys())
27
28 elif resp == '2':
29     print("Nmap Version: ", scanner.nmap_version())
30     scanner.scan(ip_addr, '1-1024', '-v -sU')
31     print(scanner.scaninfo())
32     print("IP Status: ", scanner[ip_addr].state())
33     if 'udp' in scanner[ip_addr]:
34         print("Open UDP Ports: ", scanner[ip_addr]['udp'].keys())
35     else:
36         print("No open UDP ports found!")
37
38 elif resp == '3':
39     print("Nmap Version: ", scanner.nmap_version())
40     scanner.scan(ip_addr, '1-1024', '-v -sS -sV -sC -A -O')
41     print(scanner.scaninfo())
42     print("IP Status: ", scanner[ip_addr].state())
43     print("Open TCP Ports: ", scanner[ip_addr]['tcp'].keys())
44
45 else:
46     print("Please enter a valid option!")
47

```

## Python Code: scan a range of IP addresses

```
PortScanner.py x ScannerSocket.py
1  # with Bonus
2
3  #!/usr/bin/python3
4
5  import nmap
6  import logging
7  import sys
8  import socket
9
10 # Configure logging
11 logging.basicConfig(filename='port_scanner.log', level=logging.INFO, format='%(asctime)s - %(levelname)s: %(message)s')
12
13 def scan_ports(ip_addr, scan_type):
14     try:
15         scanner = nmap.PortScanner()
16         logging.info(f"Starting {scan_type} scan for IP: {ip_addr}")
17
18         if scan_type == '1':
19             scanner.scan(ip_addr, '1-100', arguments='-v -sS')
20             logging.info(f"SYN ACK Scan completed for IP: {ip_addr}")
21             return scanner[ip_addr]['tcp']
22
23         elif scan_type == '2':
24             scanner.scan(ip_addr, '1-100', arguments='-v -sU')
25             logging.info(f"UDP Scan completed for IP: {ip_addr}")
26             return scanner[ip_addr]['udp'] if 'udp' in scanner[ip_addr] else {}
27
28         elif scan_type == '3':
29             scanner.scan(ip_addr, '1-100', arguments='-v -sS -sV -sC -A -O')
30             logging.info(f"Comprehensive Scan completed for IP: {ip_addr}")
31             return scanner[ip_addr]['tcp']
32
33         else:
34             logging.error("Invalid scan type")
35             print("Please enter a valid scan type (1, 2, or 3)")
36             sys.exit(1)
37
38     except Exception as e:
39         logging.error(f"Error scanning ports for IP: {ip_addr} - {str(e)}")
40         print(f"Error scanning ports for IP: {ip_addr} - {str(e)}")
41         sys.exit(1)
42
43 def main():
44     print("Welcome, this is a simple automation tool")
45     print("<----->")
46
47     ip_range_start = input("Please enter the starting IP address: ")
48     ip_range_end = input("Please enter the ending IP address: ")
49     scan_type = input("""\nPlease enter the type of scan you want to run
50     1) SYN ACK Scan
51     2) UDP Scan
52     3) Comprehensive Scan \n""")
53     print("You have selected option: ", scan_type)
54
55     logging.info(f"Scanning IP range: {ip_range_start} - {ip_range_end} with scan type: {scan_type}")
56
57     try:
58         # Convert IP range to list of IP addresses
59         ip_range = [f"192.168.189.{i}" for i in range(int(ip_range_start.split('.')[3]), int(ip_range_end.split('.')[3])+1)]
60
61         for ip_addr in ip_range:
62             print(f"\nScanning ports for IP: {ip_addr}")
63             ports = scan_ports(ip_addr, scan_type)
64
65             if ports:
66                 print(f"Open Ports: {list(ports.keys())}")
67             else:
68                 print("No open ports found")
69
70     except ValueError:
71         logging.error("Invalid IP range")
72         print("Invalid IP range. Please enter valid IP addresses.")
73         sys.exit(1)
74     except socket.gaierror:
75         logging.error("Invalid IP address")
76         print("Invalid IP address. Please enter valid IP addresses.")
77         sys.exit(1)
78
79 if __name__ == "__main__":
80     main()
```