

Project Title: TCP Port Scanner Using Python

Name : S.Krishna Kumar

Platform / Context: Cybersecurity / Networking Project

1. Project Overview

This project is a TCP Port Scanner developed in Python that scans a target host to identify the status of TCP ports. It uses socket programming for network communication and multithreading to improve scanning speed. The scanner reports whether ports are OPEN, CLOSED, FILTERED, or TIMEOUT, and logs results for future reference.

This project is suitable for cybersecurity beginners, networking students, and SOC / VAPT fresher interviews as it demonstrates core networking and concurrency concepts.

2. Objectives

- Build a TCP port scanner to detect open ports on a target host
- Understand TCP socket communication
- Implement multithreading to improve scan performance
- Allow scanning of a single host with a user-defined port range
- Display scan results in real time and store them in log files
- Handle network errors and exceptions gracefully

3. Scope of the Project

This Scanner supports:

- Scanning a single host or IP address
- Scanning a specified range of TCP ports
- Classification of ports as Open, Closed, Filtered, or Timeout

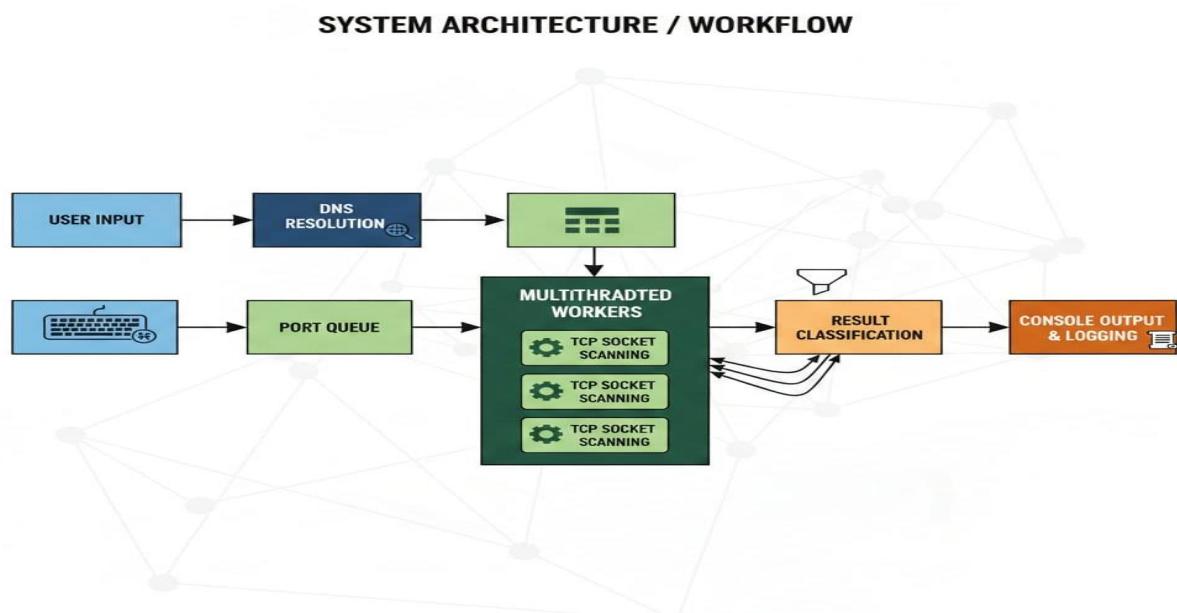
The project is intended for educational and authorized environments only.

4. Technologies & Tools Used

- **Programming Language:** Python 3
- **Libraries:** socket, threading, queue, logging, datetime
- **Operating System:** Cross-platform (Linux)
- **IDE/Editor:** VS Code / Terminal

5. System Architecture / Workflow

1. User Provides target host and port range
2. Hostname is resolved to IP address
3. Port numbers are added to a shared queue
4. Multiple threads retrieve ports from the queue
5. Each thread attempts a TCP connection
6. Results are printed and logged
7. Scan completion time is displayed



6. Methodology / Code Explanation

1. Configuration
 - Thread count, timeout, and log file defined at the start
2. Port Scanning Logic
 - TCP connection attempts using “socket.connect_ex()”
 - Return codes used to classify port state
3. Multithreading
 - Worker threads scan ports concurrently
 - Queue ensures thread-safe task distribution
4. Logging Mechanism
 - Logs stored with timestamps
 - Helps in audit and review
5. Exception Handling
 - Handles invalid hosts
 - Manages timeouts and socket errors
 - Prevents program crashes

7. Implementation (Source Code)

File: Port_scanner.py

Source Code:

```
import socket
import threading
import queue
import logging
from datetime import datetime

# ----- CONFIG -----
THREAD_COUNT = 50
TIMEOUT = 3
LOG_FILE = "scan_results.log"
# -----

# Setup logging
logging.basicConfig(
    filename=LOG_FILE,
    level=logging.INFO,
    format="%(asctime)s - %(message)s"
)

print_lock = threading.Lock()
port_queue = queue.Queue()

def scan_port(target, port):
    """Scan a single TCP port"""
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(TIMEOUT)
```

```
result = sock.connect_ex((target, port))

with print_lock:

    if result == 0:
        print(f"[OPEN] Port {port}")
        logging.info(f"OPEN Port {port}")

    elif result == 111:
        print(f"[CLOSED] Port {port}")
        logging.info(f"CLOSED Port {port}")

    else:
        print(f"[FILTERED] Port {port}")
        logging.info(f"FILTERED Port {port}")

    sock.close()

except socket.timeout:

    with print_lock:
        print(f"[TIMEOUT] Port {port}")
        logging.info(f"TIMEOUT Port {port}")

except Exception as e:

    with print_lock:
        print(f"[ERROR] Port {port}: {e}")
        logging.error(f"ERROR Port {port}: {e}")

def worker(target):
    """Thread worker function"""

    while not port_queue.empty():

        port = port_queue.get()
        scan_port(target, port)
        port_queue.task_done()
```

```
def main():
    print("\n===== TCP Port Scanner =====\n")
    target = input("Enter target host/IP: ")
    start_port = int(input("Enter start port: "))
    end_port = int(input("Enter end port: "))

    try:
        target_ip = socket.gethostbyname(target)
    except socket.gaierror:
        print(" Hostname could not be resolved.")
        return

    print(f"\nScanning {target_ip} from port {start_port} to {end_port}")
    print("-" * 50)

    start_time = datetime.now()

    # Fill queue with ports
    for port in range(start_port, end_port + 1):
        port_queue.put(port)

    # Create threads
    threads = []
    for _ in range(THREAD_COUNT):
        thread = threading.Thread(target=worker, args=(target_ip,))
        thread.daemon = True
        thread.start()
        threads.append(thread)

    # Wait for completion
    port_queue.join()
```

```

end_time = datetime.now()

print("-" * 50)

print(f"Scan completed in: {end_time - start_time}")

print(f"Results saved to: {LOG_FILE}")

if __name__ == "__main__":
    main()

```

```

Session Actions Edit View Help
[001] new 8.6          top_portscan.py
Import socket
Import threading
Import queue
Import time
Import datetime
From datetime import datetime

# ----- CONFIG -----
THREAD_POOL = 50
TIMEOUT = 3
LOGFILE = "scan_Results.log"
#-----


# Setup logging
logging.basicConfig(
    filename=LOGFILE,
    level=logging.INFO,
    format="%(asctime)s - %(message)s"
)

print_lock = threading.Lock()
port_queue = queue.Queue()

def scan_port(target, port):
    """Scan a single TCP port"""
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(TIMEOUT)
        sock.connect((target, port))
        print_lock.acquire()
        print(f"[OPEN] Port {port}")
        sock.close()
    except:
        print_lock.acquire()
        print(f"[FILTERED] Port {port}")
        pass
    finally:
        port_queue.task_done()

```

```

Session Actions Edit View Help
[001] new 8.6          top_portscan.py
Import socket
Import threading
Import queue
Import time
Import datetime
From datetime import datetime

# ----- CONFIG -----
THREAD_POOL = 50
TIMEOUT = 3
LOGFILE = "scan_Results.log"
#-----


# Setup logging
logging.basicConfig(
    filename=LOGFILE,
    level=logging.INFO,
    format="%(asctime)s - %(message)s"
)

print_lock = threading.Lock()
port_queue = queue.Queue()

def scan_port(target, port):
    """Scan a single TCP port"""

    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(TIMEOUT)
        sock.connect((target, port))
        print_lock.acquire()
        print(f"[OPEN] Port {port}")
        sock.close()
    except:
        print_lock.acquire()
        print(f"[FILTERED] Port {port}")
        pass
    finally:
        port_queue.task_done()

```

GitHub Repo : https://github.com/Premkumar-pro/Syntecxhub_TCPportscanning_using_python/blob/main/tcp_portscanning.py

8. Execution Steps

1. Run the script
2. Enter target host/IP
3. Enter Starting Port
4. Enter Ending Port
5. View real-time results
6. Check log file for stored output

```

#python3 tcp_portscan.py
#IP : 192.168.1.16
#Enter starting port : 100
#Enter end port :4000

```

```

(kali㉿kali)-[~]
$ ls
cracked.json  Desktop  Documents  Downloads  Music  Pictures  Public  scan_results.log  tcp_portscan.py  Templates  tools  Videos
(kali㉿kali)-[~]
$ nano tcp_portscan.py
(kali㉿kali)-[~]
$ python3 tcp_portscan.py
===== TCP Port Scanner =====

Enter target host/IP: 192.168.1.16
Enter start port: 100
Enter end port: 4000

Scanning 192.168.1.16 from port 100 to 4000
[OPEN]  Port 135
[OPEN]  Port 139
[FILTERED] Port 101
[FILTERED] Port 100
[FILTERED] Port 102
[FILTERED] Port 104
[FILTERED] Port 105
[FILTERED] Port 106
[FILTERED] Port 103
[FILTERED] Port 108
[FILTERED] Port 110
[FILTERED] Port 107
[FILTERED] Port 111
[FILTERED] Port 109
[FILTERED] Port 112

```

9. Results & Output

- Displays port status in terminal
- Saves scan results to scan_results.log
- Shows total scan duration

Output:

```
[FILTERED] Port 3972
[FILTERED] Port 3974
[FILTERED] Port 3975
[FILTERED] Port 3976
[FILTERED] Port 3977
[FILTERED] Port 3978
[FILTERED] Port 3979
[FILTERED] Port 3980
[FILTERED] Port 3981
[FILTERED] Port 3982
[FILTERED] Port 3983
[FILTERED] Port 3985
[FILTERED] Port 3986
[FILTERED] Port 3987
[FILTERED] Port 3988
[FILTERED] Port 3989
[FILTERED] Port 3990
[FILTERED] Port 3991
[FILTERED] Port 3992
[FILTERED] Port 3997
[FILTERED] Port 3998
[FILTERED] Port 3996
[FILTERED] Port 3995
[FILTERED] Port 3993
[FILTERED] Port 3994
[FILTERED] Port 3999
[FILTERED] Port 4000
Scan completed in: 0:03:54.603413
Results saved to: scan_results.log

```

Log file:

```
(kali㉿kali)-[~]
$ cat scan_results.log
2025-12-16 02:32:35,429 - OPEN Port 135
2025-12-16 02:32:35,431 - OPEN Port 139
2025-12-16 02:32:38,366 - FILTERED Port 101
2025-12-16 02:32:38,367 - FILTERED Port 100
2025-12-16 02:32:38,369 - FILTERED Port 102
2025-12-16 02:32:38,369 - FILTERED Port 104
2025-12-16 02:32:38,369 - FILTERED Port 105
2025-12-16 02:32:38,370 - FILTERED Port 106
2025-12-16 02:32:38,371 - FILTERED Port 103
2025-12-16 02:32:38,375 - FILTERED Port 108
2025-12-16 02:32:38,376 - FILTERED Port 110
2025-12-16 02:32:38,377 - FILTERED Port 107
2025-12-16 02:32:38,379 - FILTERED Port 111
2025-12-16 02:32:38,380 - FILTERED Port 109
2025-12-16 02:32:38,380 - FILTERED Port 112
2025-12-16 02:32:38,384 - FILTERED Port 113
2025-12-16 02:32:38,389 - FILTERED Port 115
2025-12-16 02:32:38,389 - FILTERED Port 116
2025-12-16 02:32:38,390 - FILTERED Port 114
2025-12-16 02:32:38,392 - FILTERED Port 118
2025-12-16 02:32:38,393 - FILTERED Port 117
2025-12-16 02:32:38,396 - FILTERED Port 119
2025-12-16 02:32:38,396 - FILTERED Port 120
2025-12-16 02:32:38,400 - FILTERED Port 121
2025-12-16 02:32:38,403 - FILTERED Port 122
2025-12-16 02:32:38,404 - FILTERED Port 125
2025-12-16 02:32:38,404 - FILTERED Port 124
```

10. Limitations

- Scans TCP ports only
- No service/version detection
- No stealth or rate-limiting features

11. Learning Outcomes

- Gained hands-on experience with socket programming
- Learned multithreading and synchronization
- Understood basic port scanning techniques
- Improved debugging and logging skills

12. Conclusion

The TCP Port Scanner successfully demonstrates fundamental networking and cybersecurity concepts using Python. The project provides a strong foundation for understanding network reconnaissance and serves as a base for building advanced scanning tools.