**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

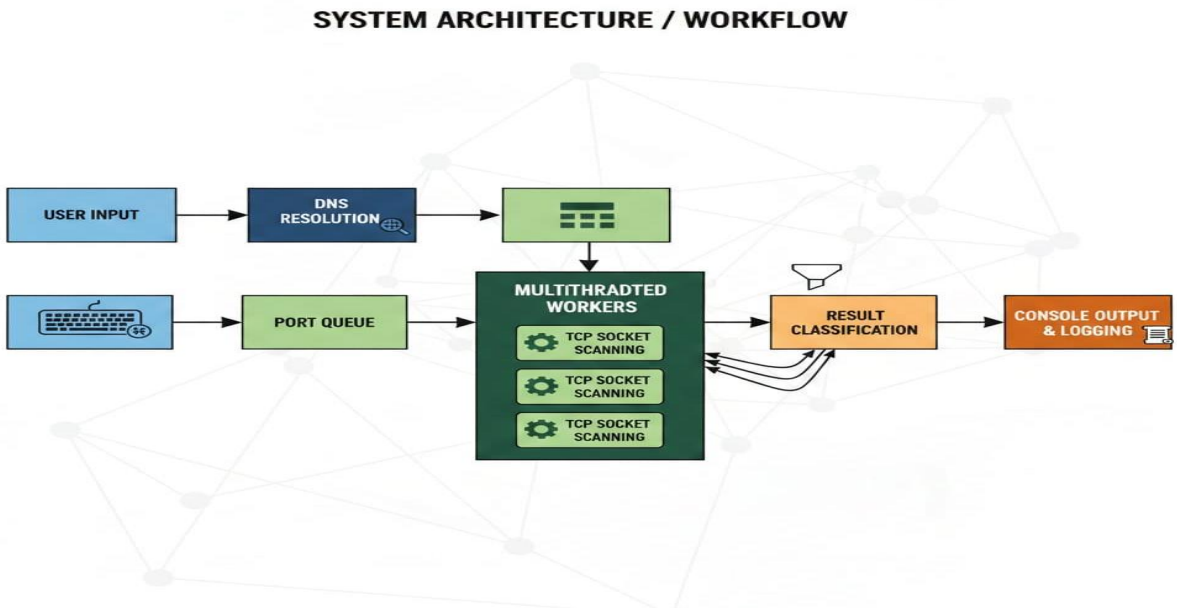
The project is intended for educational and authorized environments only.

## 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

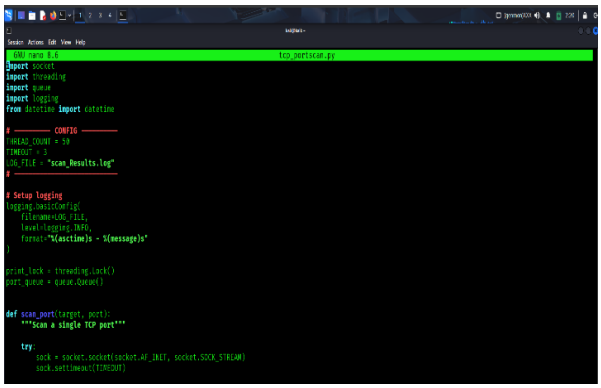
## Source Code:

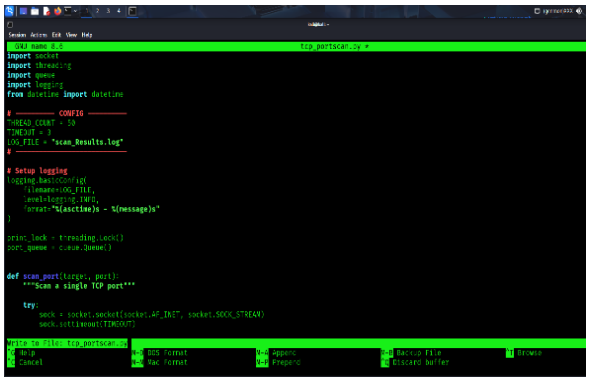
```
import socket
import threading
import queue
import logging
from datetime import datetime

# ---------- 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()
```

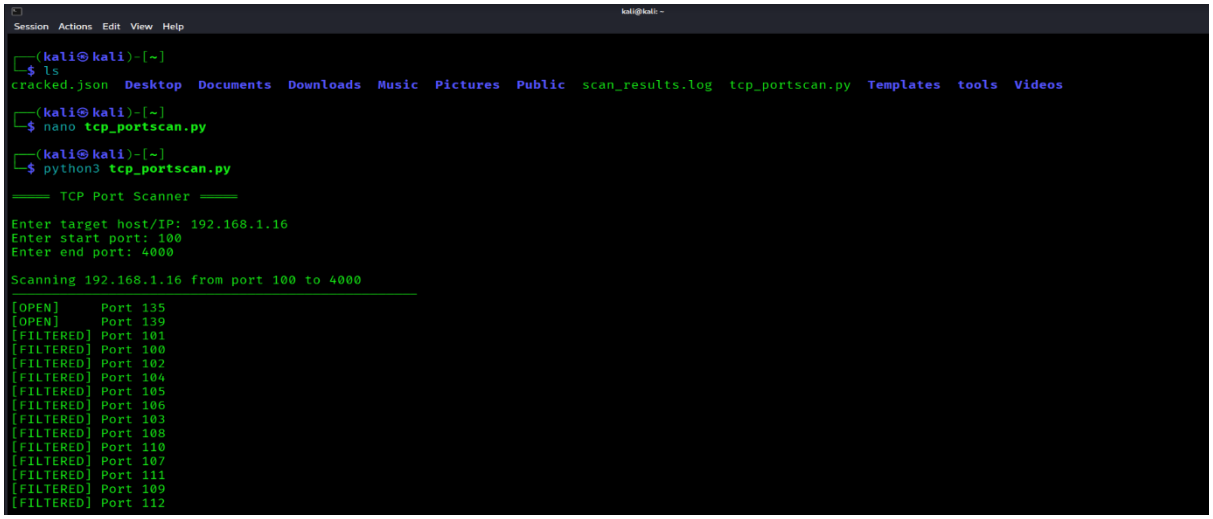




**GitHub Repo** : https://github.com/Premkumar-pro/Syntecxhub_TCPportscanning_using_python/blob/main/tcp_portscanning.py

## 8. Execution Steps

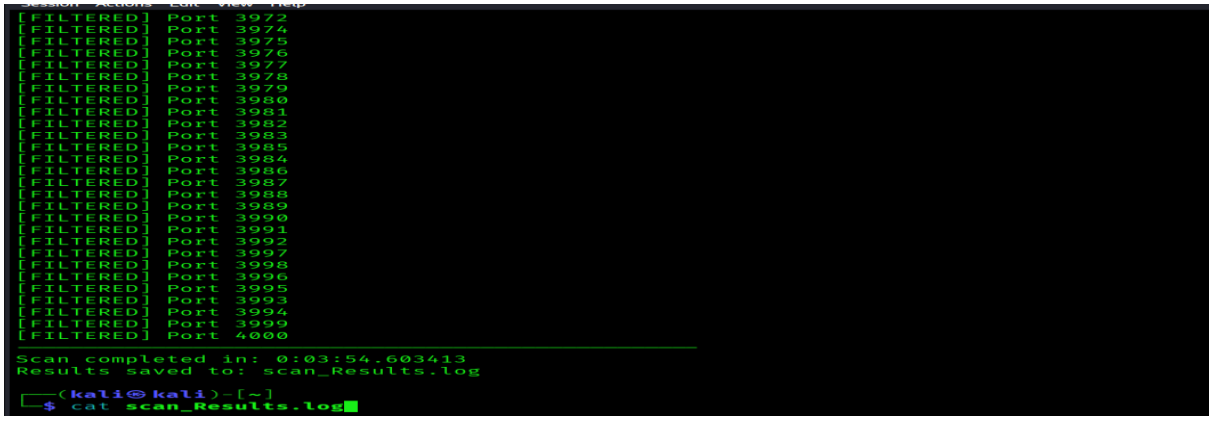
1. Run the script #python3 tcp_portscan.py
2. Enter target host/IP #IP : 192.168.1.16
3. Enter Starting Port #Enter starting port : 100
4. Enter Ending Port #Enter end port :4000
5. View real-time results
6. Check log file for stored output



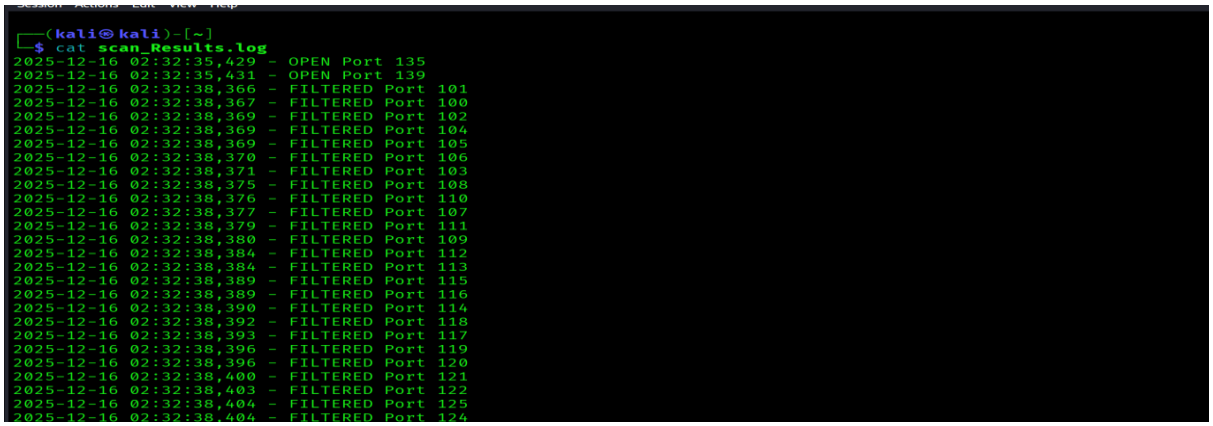
## 9. Results & Output

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

**Output:**




**Log file:**




## 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.