**Full Code Explanation for the Network Port Scanner**

**📆 Author Credits**

*Project by*: **Youssef Amr, Youssef Hisham, Zeyad Sayed, Sam Malak**

This document provides a detailed explanation of each component of the Python script used for port scanning and network monitoring using Wireshark.

**✨ Overview**

This Python-based tool scans a target IP address to identify open ports using either TCP or UDP protocols. It supports multi-threaded scanning, CSV result export, real-time progress display, optional log file analysis, and optional Wireshark integration.

**🔢 Modules and Their Purpose**

import socket

import argparse

import csv

import subprocess

import re

import sys

from datetime import datetime

from concurrent.futures import ThreadPoolExecutor

from tqdm import tqdm

* socket: Enables low-level networking to check if a port is open.
* argparse: Handles command-line arguments.
* csv: Writes results into a CSV file.
* subprocess: Launches Wireshark externally.
* re: For regular expression matching in log files.
* sys: For script termination using sys.exit().
* datetime: Timestamps for filenames and reporting.
* ThreadPoolExecutor: Enables concurrent port scanning.
* tqdm: Provides a progress bar during scanning.

**⚙️ Constants**

COMMON\_PORTS = { ... }

Defines a dictionary of well-known ports and their associated services.

def get\_service\_name(port):

return COMMON\_PORTS.get(port, "Unknown")

Returns a human-readable service name for a given port.

**📝 Port Scanning Logic**

**Single Port Scanner**

def scan\_single\_port(ip, port, protocol, timeout=1):

* Tries to connect to a given port using TCP or UDP.
* Returns the port number if it's open; otherwise None.

**Multi-threaded Scanning with Progress**

def scan\_ports\_with\_progress(ip, start\_port, end\_port, protocol, max\_workers=50):

* Scans a range of ports using multiple threads.
* Displays a progress bar using tqdm.
* Returns a sorted list of open ports.

**📊 Output Formatting**

**Table Formatter**

def format\_results\_table(open\_ports):

* Formats the list of open ports into a clean ASCII table.

**CSV Saver**

def save\_results\_to\_csv(ip, open\_ports, filename=None):

* Saves the scan results in CSV format.
* Filenames include timestamps unless specified.

**📃 Log File Analysis**

def analyze\_log\_file(log\_file):

* Reads a log file line-by-line.
* Searches for potential security issues like error, timeout, attack, etc.
* Highlights timestamps, IPs, and line numbers for easier analysis.

**🔢 IP Validation**

def validate\_ip(ip):

* Ensures the IP address entered is valid using socket.inet\_aton().

**⚡ Main Function**

def main():

Handles:

* Argument parsing (IP, port range, protocol, CSV output, Wireshark, logs).
* IP and port validation.
* Starts the scan and prints results.
* Optionally saves results or launches Wireshark.
* Gracefully handles errors and interruptions.

**⛏ Command-Line Usage**

python port\_scanner.py <IP> <start\_port> <end\_port> [--protocol tcp|udp] [--save] [--output filename.csv] [--wireshark] [--log file.log] [--threads N]

**Example:**

python port\_scanner.py 192.168.1.1 20 100 --protocol tcp --save --wireshark

This command:

* Scans ports 20 to 100 on 192.168.1.1.
* Uses TCP protocol.
* Saves results to CSV.
* Launches Wireshark for live packet capture on discovered open ports.

**✅ Advantages**

* Easy to use and extend.
* Integrates with real-time monitoring (Wireshark).
* Useful for diagnostics and security analysis.
* Includes log file analysis.
* Multi-threaded for speed.

**🔹 Suggestions for Future Improvement**

* Add support for OS detection and banner grabbing.
* Integrate with vulnerability scanning APIs.
* Add email notifications on scan completion.
* Create a GUI using Tkinter or PyQt.
* Export to JSON format for automation use.