Port scanner tool

CODE :

```
import socket

import threading

import platform

import re

from tkinter import *

from tkinter import ttk, messagebox

import subprocess

from datetime import datetime

from queue import Queue

import concurrent.futures

import time

import nmap


class UnifiedPortScanner:

    def __init__(self, master):

        self.master = master

        master.title("Advanced Port Scanner with OS Detection")

        master.geometry("1000x800")


        self.scan_active = False

        self.thread_pool = None

        self.scan_queue = Queue()
        self.max_threads = 100
        self.nm = nmap.PortScanner() if self.check_nmap() else None

        # OS fingerprint patterns
        self.os_patterns = {
            'Linux': {'ttl_range': (30, 64), 'tcp_options': ['mss', 'sackOK', 'ts',
'nop', 'wscale']},
            'Windows': {'ttl_range': (113, 128), 'tcp_options': ['mss', 'nop',
'nop', 'sackOK']},
            'MacOS': {'ttl_range': (60, 64), 'tcp_options': ['mss', 'nop',
'wscale', 'nop', 'sackOK', 'ts']},
            'Router': {'ttl_range': (250, 255), 'tcp_options': ['mss']}
        }

        # Setup UI
        self.create_ui()

    def check_nmap(self):
        try:
            subprocess.run(["nmap", "--version"], check=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE)
            return True
        except:
            messagebox.showwarning("Nmap Not Available", "Advanced
OS detection features require Nmap (nmap.org)")
```

```
            return False

    def create_ui(self):
        """Create the UI components"""
        style = ttk.Style(self.master)
        style.theme_use('clam')

        # Main container
        self.main_frame = ttk.Frame(self.master, padding="10")
        self.main_frame.pack(fill=BOTH, expand=True)

        # Input Frame
        self.input_frame = ttk.LabelFrame(self.main_frame, text="Scan
Parameters", padding="10")
        self.input_frame.pack(fill=X, pady=5)

        # Target configuration
        ttk.Label(self.input_frame, text="Target
IP/Hostname:").grid(row=0, column=0, sticky=W)
        self.host_entry = ttk.Entry(self.input_frame, width=30)
        self.host_entry.grid(row=0, column=1, padx=5, pady=2,
sticky=W)
        self.host_entry.insert(0, "192.168.1.1")
        ttk.Label(self.input_frame, text="Port Range:").grid(row=1,
column=0, sticky=W)
        self.port_entry = ttk.Entry(self.input_frame, width=30)
        self.port_entry.grid(row=1, column=1, padx=5, sticky=W)
        self.port_entry.insert(0, "1-1024")

        ttk.Label(self.input_frame, text="Options:").grid(row=0,
column=2, padx=10, sticky=W)
        self.os_detect_var = IntVar(value=1)
        self.service_detect_var = IntVar(value=1)
        ttk.Checkbutton(self.input_frame, text="OS Detection",
variable=self.os_detect_var).grid(row=1, column=2, sticky=W)
        ttk.Checkbutton(self.input_frame, text="Service Detection",
variable=self.service_detect_var).grid(row=2, column=2, sticky=W)

        # Thread control
        ttk.Label(self.input_frame, text="Threads:").grid(row=2,
column=0, sticky=W)
        self.thread_entry = ttk.Entry(self.input_frame, width=10)
        self.thread_entry.grid(row=2, column=1, padx=5, sticky=W)
        self.thread_entry.insert(0, str(self.max_threads))

        # Buttons
        btn_frame = ttk.Frame(self.input_frame)
        btn_frame.grid(row=3, column=0, columnspan=3, pady=5)
        self.scan_btn = ttk.Button(btn_frame, text="Start Scan", command=self.start_scan)
        self.scan_btn.pack(side=LEFT, padx=5)
        self.stop_btn = ttk.Button(btn_frame, text="Stop", command=self.stop_scan, state=DISABLED)
        self.stop_btn.pack(side=LEFT, padx=5)

        # Progress bar
        self.progress_var = DoubleVar()
        self.progress = ttk.Progressbar(self.input_frame, variable=self.progress_var, maximum=100)
        self.progress.grid(row=4, column=0, columnspan=3, sticky=EW, pady=5)

        # Results area
        self.results_frame = ttk.LabelFrame(self.main_frame, text="Scan Results", padding="10")
        self.results_frame.pack(fill=BOTH, expand=True, pady=5)

        self.result_text = Text(self.results_frame, width=120, height=30, wrap=NONE, font=('Consolas', 10))
        scroll_y = ttk.Scrollbar(self.results_frame, command=self.result_text.yview)
        scroll_x = ttk.Scrollbar(self.results_frame, orient=HORIZONTAL, command=self.result_text.xview)
        self.result_text.config(yscrollcommand=scroll_y.set,
xscrollcommand=scroll_x.set)

        scroll_y.pack(side=RIGHT, fill=Y)
        scroll_x.pack(side=BOTTOM, fill=X)
        self.result_text.pack(side=LEFT, fill=BOTH, expand=True)

    def start_scan(self):
        """Start scanning with selected options"""
        target = self.host_entry.get()
        if not target:
            messagebox.showerror("Error", "Please enter a target host")
            return

        try:
            self.max_threads = max(10, min(500,
int(self.thread_entry.get())))
        except:
            self.max_threads = 100
            self.thread_entry.delete(0, END)
            self.thread_entry.insert(0, str(self.max_threads))

        self.clear_results()
        self.scan_active = True
        self.scan_btn.config(state=DISABLED)
        self.stop_btn.config(state=NORMAL)

        threading.Thread(target=self.run_scan, args=(target,)).start()

    def run_scan(self, target):
        """Run the scan process"""
        try:
            ip_addr = self.resolve_target(target)
            self.append_result(f"[*] Target: {target}\n")
            self.append_result(f"[*] Resolved IP: {ip_addr}\n")

            # OS Detection if enabled
            if self.os_detect_var.get():
                self.detect_os(ip_addr)

            # Port Scanning
            port_range = self.port_entry.get()
            start_port, end_port = map(int, port_range.split('-'))
            self.total_ports = end_port - start_port + 1
            self.scanned_ports = 0
            self.open_ports = []
            self.append_result(f"\n[*] Scanning ports {start_port}-
    {end_port}...\n")

            # Queue all ports and start workers
            self.scan_queue = Queue()
            for port in range(start_port, end_port + 1):
                self.scan_queue.put(port)

            self.thread_pool =
    concurrent.futures.ThreadPoolExecutor(max_workers=self.max_threa
    ds)
            for _ in range(self.max_threads):
                self.thread_pool.submit(self.port_scan_worker, ip_addr)

            # Wait for completion
            while not self.scan_queue.empty() and self.scan_active:
                time.sleep(0.5)

            # Service Detection if enabled and ports found
            if self.service_detect_var.get() and self.open_ports and
    self.scan_active:
                self.append_result("\n[*] Detecting services...\n")
                self.detect_services(ip_addr)

            self.append_result("\n[*] Scan completed\n")
        except Exception as e:
            self.append_result(f"\n[!] Error: {str(e)}\n")
        finally:
            self.scan_active = False
            self.master.after(0, self.finalize_scan)

    def detect_os(self, ip):
        """Perform OS detection using multiple techniques"""
        self.append_result("\n=== OS DETECTION ===\n")

        # Technique 1: TTL Analysis
        ttl = self.get_ttl(ip)
        if ttl:
            self.append_result(f"[*] Initial TTL value: {ttl}\n")
            os_guess = self.analyze_ttl(ttl)
            self.append_result(f"[+] TTL Analysis suggests: {os_guess}\n")

        # Technique 2: Advanced Nmap detection if available
        if self.nm:
            self.append_result("[*] Running advanced OS 
fingerprinting...\n")
            try:
                self.nm.scan(hosts=ip, arguments='-O')
            if ip in self.nm.all_hosts() and 'osmatch' in self.nm[ip]:
                self.append_result("[+] Nmap OS detection results:\n")
                for os_match in self.nm[ip]['osmatch']:
                    self.append_result(f"- {os_match['name']} (Accuracy: 
{os_match['accuracy']}%)\n")
        except Exception as e:
            self.append_result(f"[!] Nmap OS detection failed: 
{str(e)}\n")

    def get_ttl(self, ip):
        """Get initial TTL value from ping response"""
        try:
            cmd = ['ping', '-n', '1', ip] if platform.system().lower() == 
"windows" else ['ping', '-c', '1', ip]
            process = subprocess.Popen(cmd, stdout=subprocess.PIPE)
            stdout, _ = process.communicate()
            output = stdout.decode('utf-8', 'ignore')

            ttl_match = re.search(r'ttl=(\d+)', output.lower())
            return int(ttl_match.group(1)) if ttl_match else None
        except:
            return None

    def analyze_ttl(self, ttl):
        """Analyze TTL value to guess OS"""
        if ttl <= 64:
            return "Linux/Unix"
        elif ttl <= 128:
            return "Windows"
        elif ttl <= 255:
            return "Router/Network Device"
        else:
            return f"Unknown (TTL: {ttl})"

    def port_scan_worker(self, ip):
        """Worker thread for port scanning"""
        while self.scan_active and not self.scan_queue.empty():
            try:
                port = self.scan_queue.get_nowait()
                if self.scan_port(ip, port):
                    self.open_ports.append(port)
                    self.master.after(0, self.append_result, f"[+] Port
{port}/tcp open\n")
                    if self.service_detect_var.get():
                        self.try_banner_grabbing(ip, port)

                self.scanned_ports += 1
                self.progress_var.set((self.scanned_ports / self.total_ports) *
100)
                self.master.update()
            except:
                continue

    def scan_port(self, ip, port):
        """Check if a port is open"""
        try:
            with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
                s.settimeout(0.5)
                return s.connect_ex((ip, port)) == 0
        except:
            return False

    def try_banner_grabbing(self, ip, port):
        """Attempt to get service banner"""
        try:
            with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
                s.settimeout(1)
                s.connect((ip, port))

                if port == 21:  # FTP
                    s.send(b"USER anonymous\r\n")
            elif port == 22:  # SSH
                s.send(b"SSH-2.0-OpenSSH_7.4\r\n")
            elif port in [80, 443]:  # HTTP/HTTPS
                s.send(b"GET / HTTP/1.1\r\nHost: example.com\r\n\r\n")

            banner = s.recv(1024).decode('utf-8', 'ignore').strip()
            if banner:
                self.master.after(0, self.append_result, f"  Service:
{banner[:200]}\n")
        except:
            pass

    def detect_services(self, ip):
        """Perform service detection using Nmap"""
        if not self.nm or not self.open_ports:
            return

        try:
            self.nm.scan(hosts=ip, ports=','.join(map(str, self.open_ports)),
arguments='-sV')

            if ip in self.nm.all_hosts():
                for proto in self.nm[ip].all_protocols():
                    for port, data in self.nm[ip][proto].items():
```

```
                service_info = f"  {port}/{proto}: {data['name']}"
                if 'product' in data:
                    service_info += f" {data['product']}"
                if 'version' in data:
                    service_info += f" {data['version']}"
                self.append_result(service_info + "\n")
    except Exception as e:
        self.append_result(f"[!] Service detection error: {str(e)}\n")

def resolve_target(self, target):
    """Resolve hostname to IP address"""
    try:
        return socket.gethostbyname(target)
    except socket.gaierror:
        return target

def append_result(self, text):
    """Thread-safe result display"""
    self.result_text.insert(END, text)
    self.result_text.see(END)
    self.master.update()

def clear_results(self):
    """Clear the results display"""
```

```
        self.result_text.delete(1.0, END)

    def stop_scan(self):
        """Stop the current scan"""
        if self.scan_active:
            self.scan_active = False
            if self.thread_pool:
                self.thread_pool.shutdown(wait=False)

            self.append_result("\n[*] Scan stopped by user\n")
            self.append_result(f"[*] Progress:
{self.scanned_ports}/{self.total_ports} ports scanned\n")

            self.scan_btn.config(state=NORMAL)
            self.stop_btn.config(state=DISABLED)

    def finalize_scan(self):
        """Clean up after scan completes"""
        self.scan_btn.config(state=NORMAL)
        self.stop_btn.config(state=DISABLED)

        if hasattr(self, 'open_ports'):
            self.append_result(f"[*] Found {len(self.open_ports)} open
ports\n")
```

```
if __name__ == "__main__":
    root = Tk()
    app = UnifiedPortScanner(root)
    root.mainloop()
```

Code Breakdown

1. **Imports**:

Code :

```
import socket
import threading
import platform
import re
from tkinter import *
from tkinter import ttk, messagebox
import subprocess
from datetime import datetime
from queue import Queue
import concurrent.futures
import time
import nmap
```

- **socket**: Used for network connections to check if ports are open.
- **threading**: Allows concurrent execution of port scanning tasks.
- **platform**: Used to determine the operating system for specific commands.
- **re**: Regular expressions for parsing output.
- **tkinter**: The standard GUI toolkit for Python, used to create the application interface.
- **subprocess**: Used to run system commands (like ping).

- **datetime**: For handling date and time (not used in the provided code).
- **queue**: Provides a thread-safe queue for managing tasks.
- **concurrent.futures**: For managing a pool of threads for concurrent execution.
- **time**: For sleep and timing operations.
- **nmap**: A library for interacting with the Nmap tool for advanced network scanning.

2. **Class Definition**:

Code :
```
class UnifiedPortScanner:
```

- This class encapsulates the entire functionality of the tool, including the GUI and the logic for port scanning and OS detection.

3. **Initialization**:

Code :
```
def __init__(self, master):
```

- The constructor initializes the main window and its components.

4. **Window Configuration**:

Code :
```
master.title("Advanced Port Scanner with OS Detection")
master.geometry("1000x800")
```

- Sets the title and size of the main window.

5. **Nmap Check**:

   Code : self.nm = nmap.PortScanner() if self.check_nmap() else None

   - Initializes the Nmap scanner if the Nmap tool is available on the system.

6. **OS Fingerprint Patterns**:

   Code :
   ```
   self.os_patterns = {
               'Linux': {'ttl_range': (30, 64), 'tcp_options': ['mss', 'sackOK', 'ts', 'nop', 'wscale']},
           ...
           }
   ```

   - Defines patterns for different operating systems based on TTL values and TCP options for OS detection.

7. **UI Setup**:

   Code : self.create_ui()

   - Calls the method to create the user interface components.

8. **Nmap Availability Check**:

   Code :
   ```
   def check_nmap(self):
           ...
   ```

   - Checks if Nmap is installed by running a command and shows a warning if it is not available.

9. **Create UI Method**:

Code : def create_ui(self):

...

- Sets up the GUI components, including input fields for target IP, port range, options for OS and service detection, buttons for starting/stopping the scan, and a results area.

10. **Start Scan Method**:

Code : def start_scan(self):

...

- Validates user input, initializes scanning parameters, and starts the scanning process in a separate thread.

11. **Run Scan Method**:

Code : def run_scan(self, target):

...

- Resolves the target IP, performs OS detection if enabled, and scans the specified port range using a thread pool.

12. **OS Detection Method**:

Code : def detect_os(self, ip):

...

- Performs OS detection using TTL analysis and Nmap if available.

13. **Get TTL Method**:

Code : def get_ttl(self, ip):

```
...
```

- Pings the target IP and retrieves the TTL value from the response.

14. **Analyze TTL Method**:

Code : def analyze_ttl(self, ttl):

```
...
```

- Analyzes the TTL value to guess the operating system.

15. **Port Scan Worker Method**:

Code : def port_scan_worker(self, ip):

```
...
```

- A worker thread that scans ports from the queue and checks if they are open.

16. **Scan Port Method**:

Code : def scan_port(self, ip, port):

```
...
```

- Checks if a specific port is open by attempting to connect to it.

17. **Service Detection Method**:

code :
```
def detect_services(self, ip):
    ...
```

- Uses Nmap to detect services running on open ports.

18. **Resolve Target Method**:

Code :
```
def resolve_target(self, target):
    ...
```

- Resolves a hostname to an IP address.

19. **Append Result Method**:

Code :
```
def append_result(self, text):
    ...
```

- Safely appends results to the results text area in the GUI.

20. **Clear Results Method**:

Code :
```
def clear_results(self):
    ...
```

- Clears the results display area.

21. **Stop Scan Method**:

Code :
```
def stop_scan(self):
    ...
```

- Stops the current scan and updates the UI accordingly.

22. **Finalize Scan Method**:

Code :
```
def finalize_scan(self):
    ...
```

- Cleans up the UI after the scan completes.

23. **Main Loop**:

Code :
```
if __name__ == "__main__":
        root = Tk()
        app = UnifiedPortScanner(root)
        root.mainloop()
```

- Initializes the Tkinter main loop, creating an instance of the application.

Features

- **Port Scanning**: The tool allows users to input a target IP/hostname and a range of ports to scan. It checks each port to see if it is open.
- **OS Detection**: The application can perform OS detection using TTL analysis and advanced techniques with Nmap if available.
- **Service Detection**: If enabled, the tool can detect services running on open ports using Nmap.
- **User -Friendly GUI**: The application features a clean and modern interface with input fields, buttons, and a scrollable text area for results.

- **Threading and Concurrency**: The use of threading and a thread pool allows the application to perform scans concurrently, improving performance.
- **Progress Bar**: A progress bar visually indicates the scanning progress.
- **Error Handling**: The application includes error handling for various operations, providing feedback to the user when issues arise.
- **Customizable Thread Count**: Users can specify the number of threads to use for scanning, allowing for performance tuning based on their system capabilities.

Key Features & Components

1. User Interface (Tkinter GUI)

- **Input Fields**:
    - **Target IP/Hostname** (**host_entry**): The IP or domain name to scan.
    - **Port Range** (**port_entry**): Range of ports to scan (e.g., **1-1024**).
    - **Thread Count** (**thread_entry**): Number of concurrent scanning threads (default: **100**).
    - **Checkboxes** (**os_detect_var**, **service_detect_var**): Enable OS and service detection.

- **Results Display** (**result_text**):
    - Real-time output logging of scan results.
    - Scrollable text area for viewing findings.

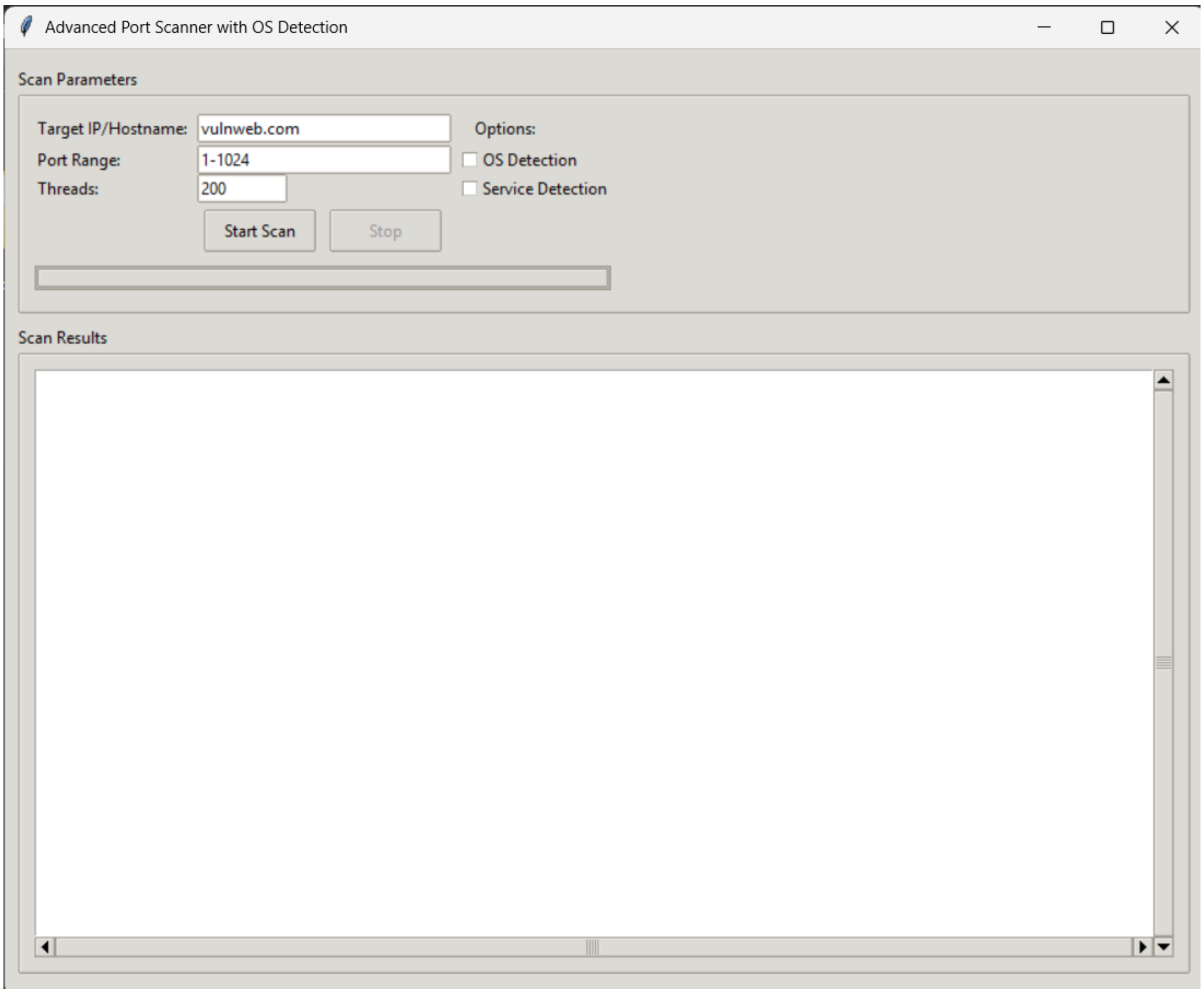
- **Controls**:
    - **Start Scan** (**scan_btn**): Initiates scanning.
    - **Stop Scan** (**stop_btn**): Stops the scan prematurely.
    - **Progress Bar** (**progress_var**): Shows scanning progress.

Workflow

1. **User Input**:
    - Enter target (e.g., **192.168.1.1**), port range (**1-1024**), and select options.
2. **Scan Execution**:
    - Resolves the target IP (**socket.gethostbyname**).
    - Performs **OS detection** (ping + TTL analysis, Nmap if enabled).
    - Scans ports (multithreaded) and checks for open ones (**scan_port()**).
    - Optionally detects services (**try_banner_grabbing()**, Nmap **-sV**).
3. **Results Logging**:
    - Shows real-time scan updates (**append_result()**).
    - Logs open ports, detected OS, and services.

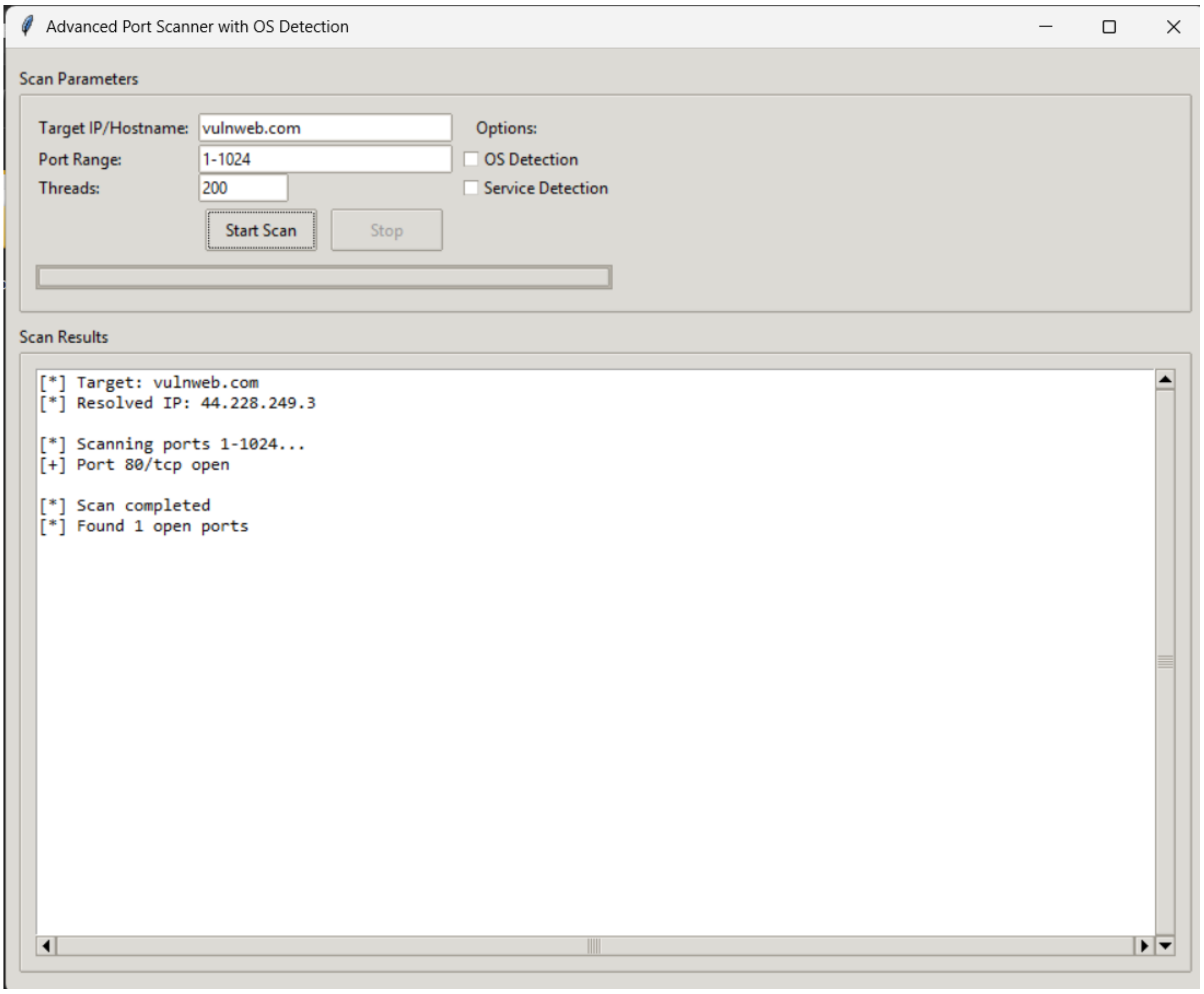
Screenshots :

1. Entered vulnweb.com as target, port rage for scanning is set to 1-1024, threads set to 200 for fast scanning



Target IP/Hostname: vulnweb.com
Port Range: 1-1024
Threads: 200
☐ OS Detection
☐ Service Detection

2. This image shows normal port scan without os and service detection



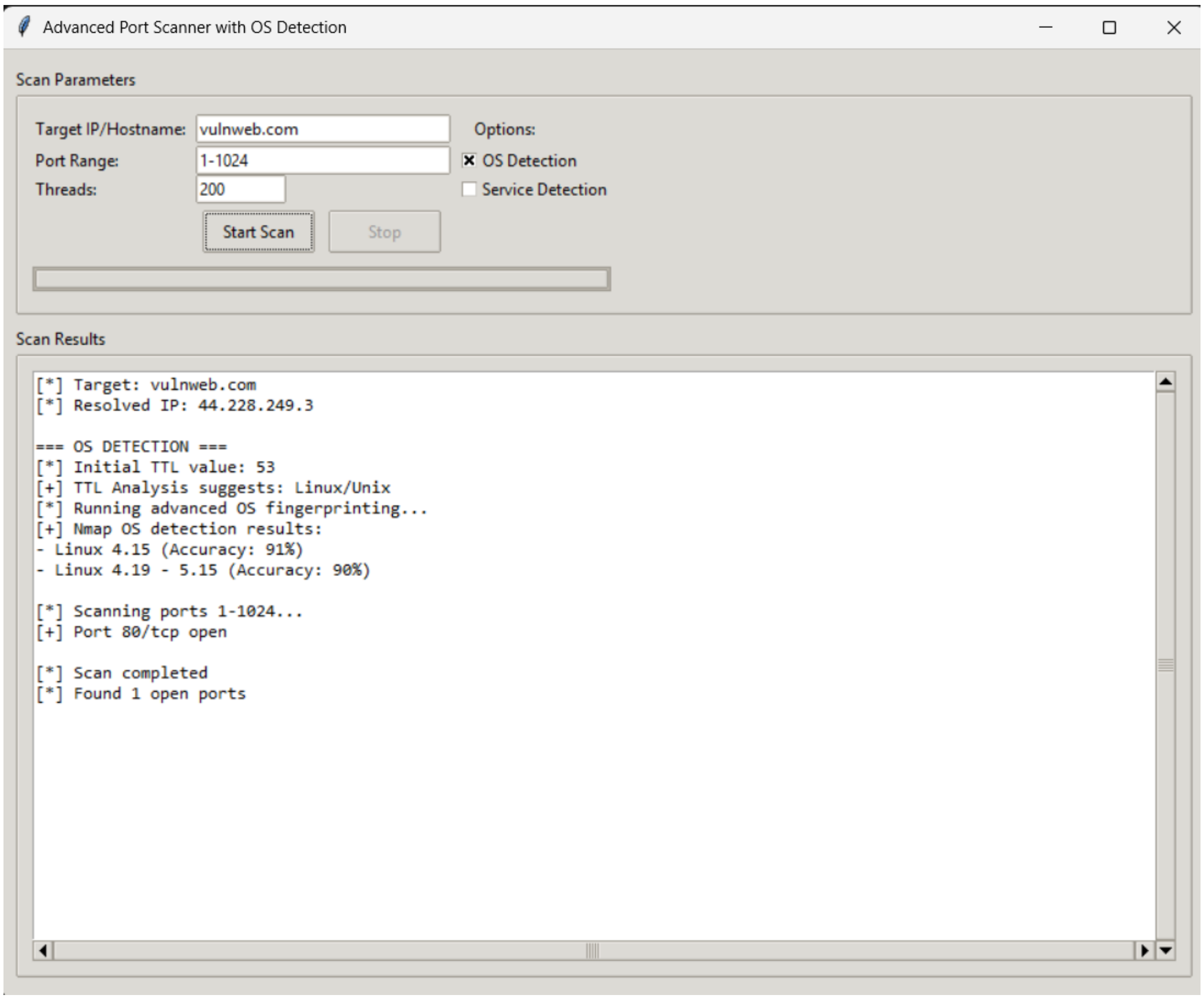
Target IP/Hostname: vulnweb.com
Port Range: 1-1024
Threads: 200
OS Detection
Service Detection

```
[*] Target: vulnweb.com
[*] Resolved IP: 44.228.249.3

[*] Scanning ports 1-1024...
[+] Port 80/tcp open

[*] Scan completed
[*] Found 1 open ports
```

3. This image shows port scan with os detection feature on



Target IP/Hostname: vulnweb.com
Port Range: 1-1024
Threads: 200
Options:
[✓ OS Detection]
Service Detection

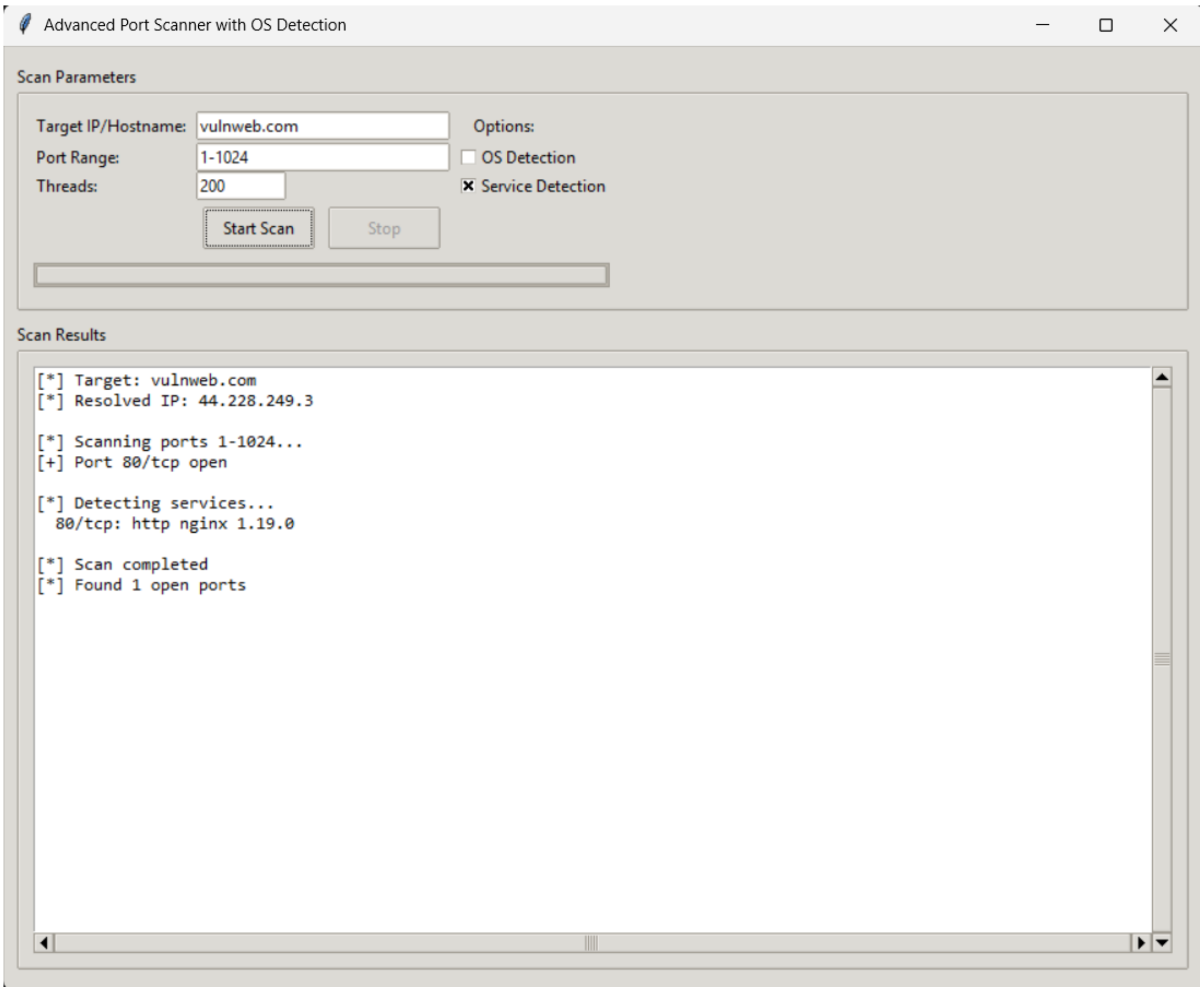
```
[*] Target: vulnweb.com
[*] Resolved IP: 44.228.249.3

=== OS DETECTION ===
[*] Initial TTL value: 53
[+] TTL Analysis suggests: Linux/Unix
[*] Running advanced OS fingerprinting...
[+] Nmap OS detection results:
- Linux 4.15 (Accuracy: 91%)
- Linux 4.19 - 5.15 (Accuracy: 90%)

[*] Scanning ports 1-1024...
[+] Port 80/tcp open

[*] Scan completed
[*] Found 1 open ports
```

4. This image shows port scan with service detection feature on



Target IP/Hostname: vulnweb.com
Port Range: 1-1024
Threads: 200
Options:
[ ] OS Detection
[✓] Service Detection

```
[*] Target: vulnweb.com
[*] Resolved IP: 44.228.249.3

[*] Scanning ports 1-1024...
[+] Port 80/tcp open

[*] Detecting services...
  80/tcp: http nginx 1.19.0

[*] Scan completed
[*] Found 1 open ports
```