CYBERSECURITY INTERNSHIP PYTHON PROJECTS(Inlighn Tech)

1.SUBDOMAIN ENUMERATION TOOL

The Subdomain Enumeration Tool is a Python-based cybersecurity project that automates the process of discovering subdomains of a given target domain. Subdomains often host applications, services, or test environments that may not be publicly known but could contain vulnerabilities. By checking which subdomains are live, this tool helps security professionals identify potential attack surfaces during reconnaissance.

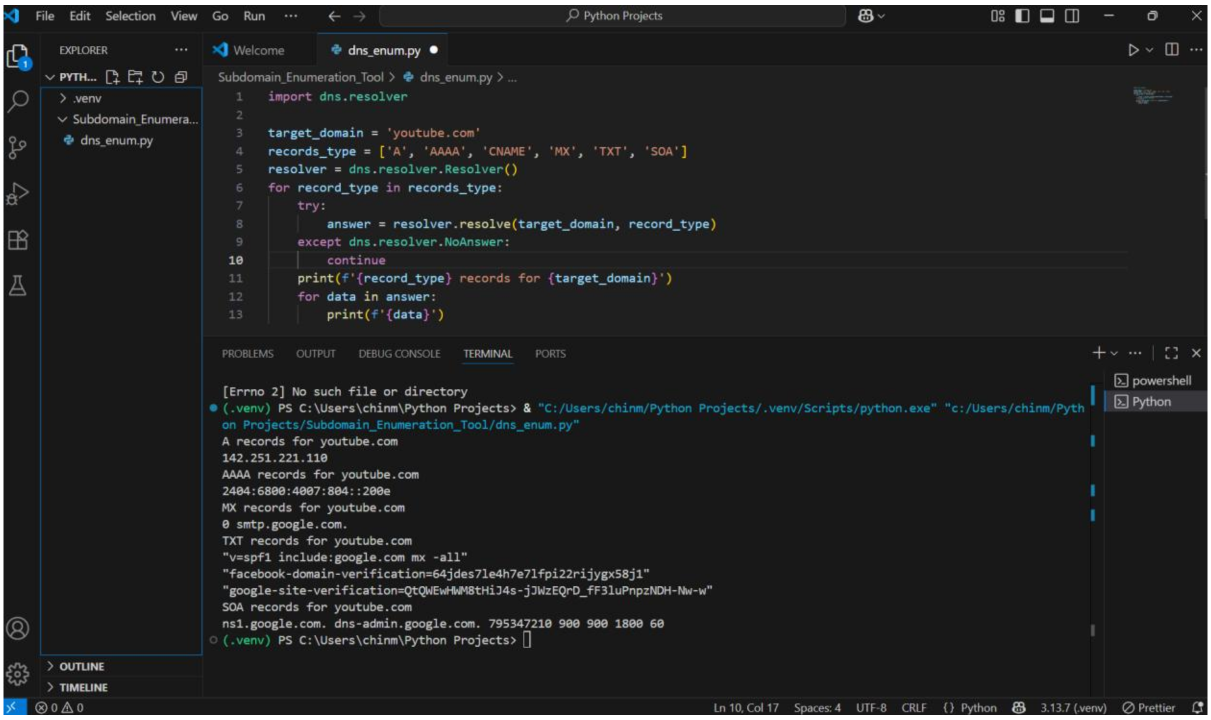
Project Functionality:

- Built a Python tool that discovers valid subdomains for a target domain (example: youtube.com).
- Used a wordlist (subdomains.txt) containing potential subdomains.
- Implemented multi-threading so multiple subdomains can be tested in parallel, speeding up execution.
- The script attempts HTTP requests to check if each subdomain is live, then logs valid results.
- Output is stored/printed so that the enumerated subdomains can be used for further reconnaissance.

Modules Installed:

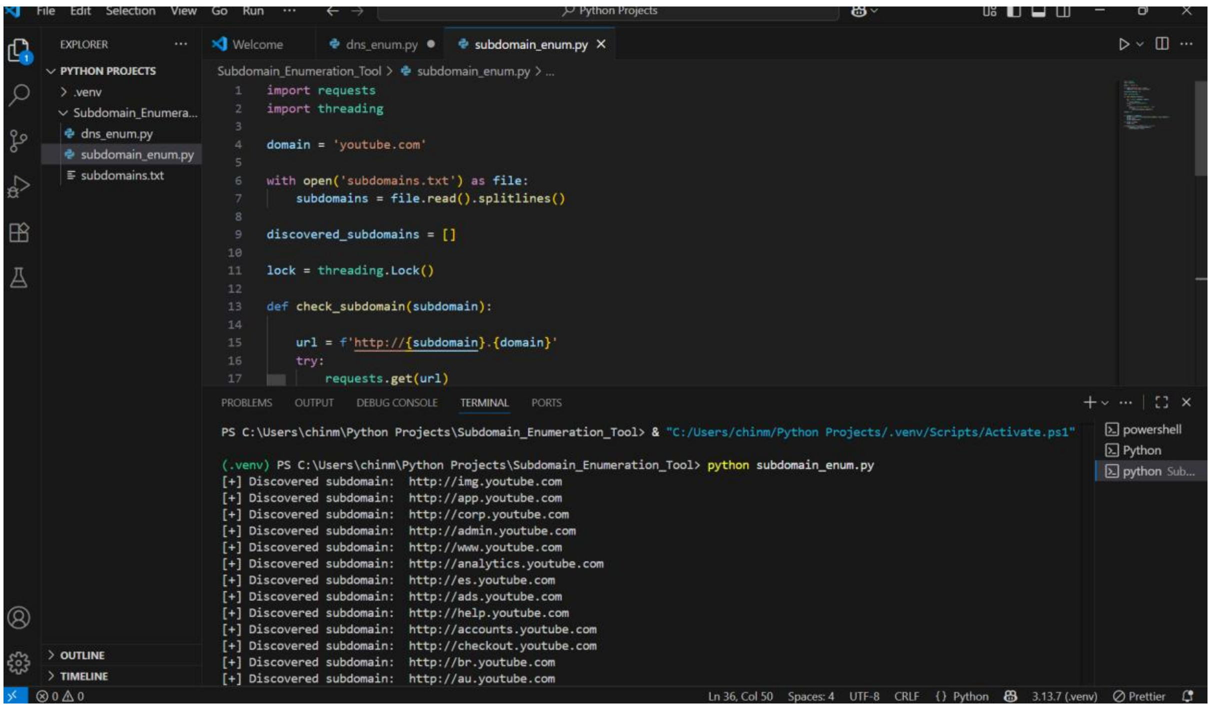
1. **requests** – to send HTTP GET requests and check if subdomains respond.
2. **threading** – to create and manage multiple threads for faster subdomain checking.
3. **dns.resolver** from dnspython – to resolve DNS records of subdomains.

Code:



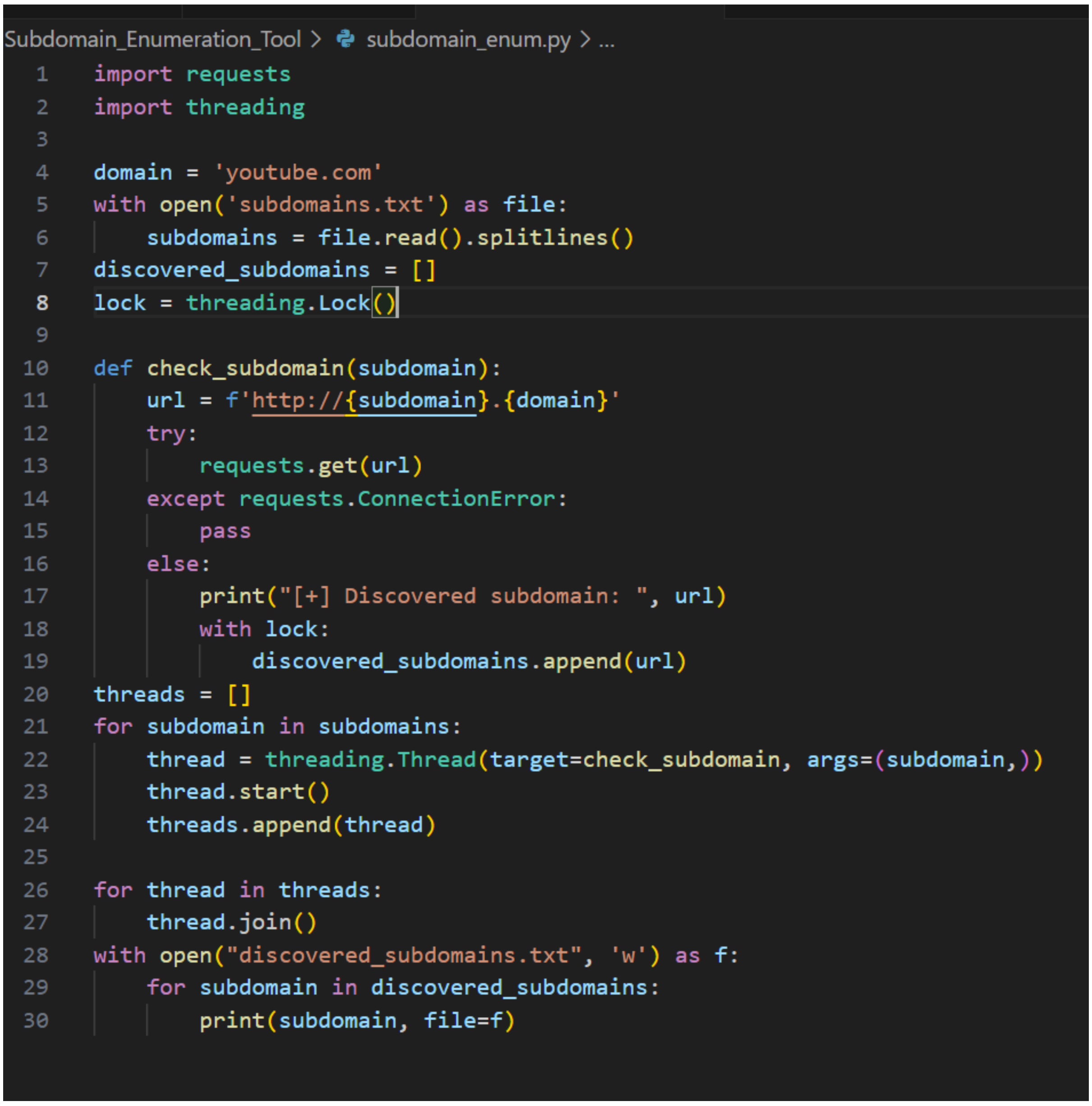
```python
import dns.resolver

target_domain = 'youtube.com'
records_type = ['A', 'AAAA', 'CNAME', 'MX', 'TXT', 'SOA']
resolver = dns.resolver.Resolver()
for record_type in records_type:
    try:
        answer = resolver.resolve(target_domain, record_type)
    except dns.resolver.NoAnswer:
        continue
    print(f'{record_type} records for {target_domain}')
    for data in answer:
        print(f'{data}')
```

```
[Errno 2] No such file or directory
(.venv) PS C:\Users\chinm\Python Projects> & "C:/Users/chinm/Python Projects/.venv/Scripts/python.exe" "c:/Users/chinm/Pyth
on Projects/Subdomain_Enumeration_Tool/dns_enum.py"
A records for youtube.com
142.251.221.110
AAAA records for youtube.com
2404:6800:4007:804::200e
MX records for youtube.com
0 smtp.google.com.
TXT records for youtube.com
"v=spf1 include:google.com mx -all"
"facebook-domain-verification=64jdes7le4h7e7lfpi22rijygx58j1"
"google-site-verification=QtQWEwHWM8tHiJ4s-jJWzEQrD_fF3luPnpzNDH-Nw-w"
SOA records for youtube.com
ns1.google.com. dns-admin.google.com. 795347210 900 900 1800 60
(.venv) PS C:\Users\chinm\Python Projects>
```



```python
import requests
import threading

domain = 'youtube.com'

with open('subdomains.txt') as file:
    subdomains = file.read().splitlines()

discovered_subdomains = []

lock = threading.Lock()

def check_subdomain(subdomain):

    url = f'http://{subdomain}.{domain}'
    try:
        requests.get(url)
```

```
PS C:\Users\chinm\Python Projects\Subdomain_Enumeration_Tool> & "C:/Users/chinm/Python Projects/.venv/Scripts/Activate.ps1"

(.venv) PS C:\Users\chinm\Python Projects\Subdomain_Enumeration_Tool> python subdomain_enum.py
[+] Discovered subdomain:  http://img.youtube.com
[+] Discovered subdomain:  http://app.youtube.com
[+] Discovered subdomain:  http://corp.youtube.com
[+] Discovered subdomain:  http://admin.youtube.com
[+] Discovered subdomain:  http://www.youtube.com
[+] Discovered subdomain:  http://analytics.youtube.com
[+] Discovered subdomain:  http://es.youtube.com
[+] Discovered subdomain:  http://ads.youtube.com
[+] Discovered subdomain:  http://help.youtube.com
[+] Discovered subdomain:  http://accounts.youtube.com
[+] Discovered subdomain:  http://checkout.youtube.com
[+] Discovered subdomain:  http://br.youtube.com
[+] Discovered subdomain:  http://au.youtube.com
```



Subdomain_Enumeration_Tool > subdomain_enum.py > ...

```python
import requests
import threading

domain = 'youtube.com'
with open('subdomains.txt') as file:
    subdomains = file.read().splitlines()
discovered_subdomains = []
lock = threading.Lock()

def check_subdomain(subdomain):
    url = f'http://{subdomain}.{domain}'
    try:
        requests.get(url)
    except requests.ConnectionError:
        pass
    else:
        print("[+] Discovered subdomain: ", url)
        with lock:
            discovered_subdomains.append(url)
threads = []
for subdomain in subdomains:
    thread = threading.Thread(target=check_subdomain, args=(subdomain,))
    thread.start()
    threads.append(thread)

for thread in threads:
    thread.join()
with open("discovered_subdomains.txt", 'w') as f:
    for subdomain in discovered_subdomains:
        print(subdomain, file=f)
```

Code Breakdown :-

1) **Open subdomains.txt**
   - Reads possible subdomains from a text file.
   - Each line becomes a subdomain candidate (e.g., www, mail, login).
2) **Initialize storage**
   - A list is created to store valid (discovered) subdomains.
   - A threading.Lock() is used to handle safe access when multiple threads write results.
3) **Function: check_subdomain(subdomain)**
   - Builds a URL in the format http://<subdomain>.<target-domain>.
   - Sends an HTTP GET request using requests.
   - If the request succeeds → the subdomain exists.

- Prints and saves the discovered subdomain to the results list using the lock.

4) **Thread creation and execution**
   - A thread is created for each subdomain in the wordlist.
   - Each thread runs the check_subdomain function.
   - All threads are started in parallel for faster execution.
5) **Thread synchronization**
   - The script waits (join) until all threads finish checking their assigned subdomains.
6) **Output handling**
   - The list of discovered subdomains is written to an output file (discovered.txt).
   - This ensures results are saved for later use in security testing.

Challenges Faced:

- **Handling Large Wordlists Efficiently** – Needed multithreading to speed up enumeration.
- **Avoiding False Positives & Errors** – Had to manage connection errors and timeouts properly.
- **Thread Synchronization** – Required locks to prevent race conditions when saving results.
- **Environment/Dependency Issues** – Faced installation problems with Python packages in virtual environments.

2. NETWORK SCANNER TOOL

The Network Scanner is a Python tool designed to discover and profile devices on a local network. It automates the process of identifying active hosts, resolving their hostnames, detecting their operating systems, and scanning for open ports. This comprehensive approach helps security professionals map a network's attack surface by identifying all reachable devices and their potential vulnerabilities.

Project Functionality:

The following key functionalities were developed and integrated into the final tool:

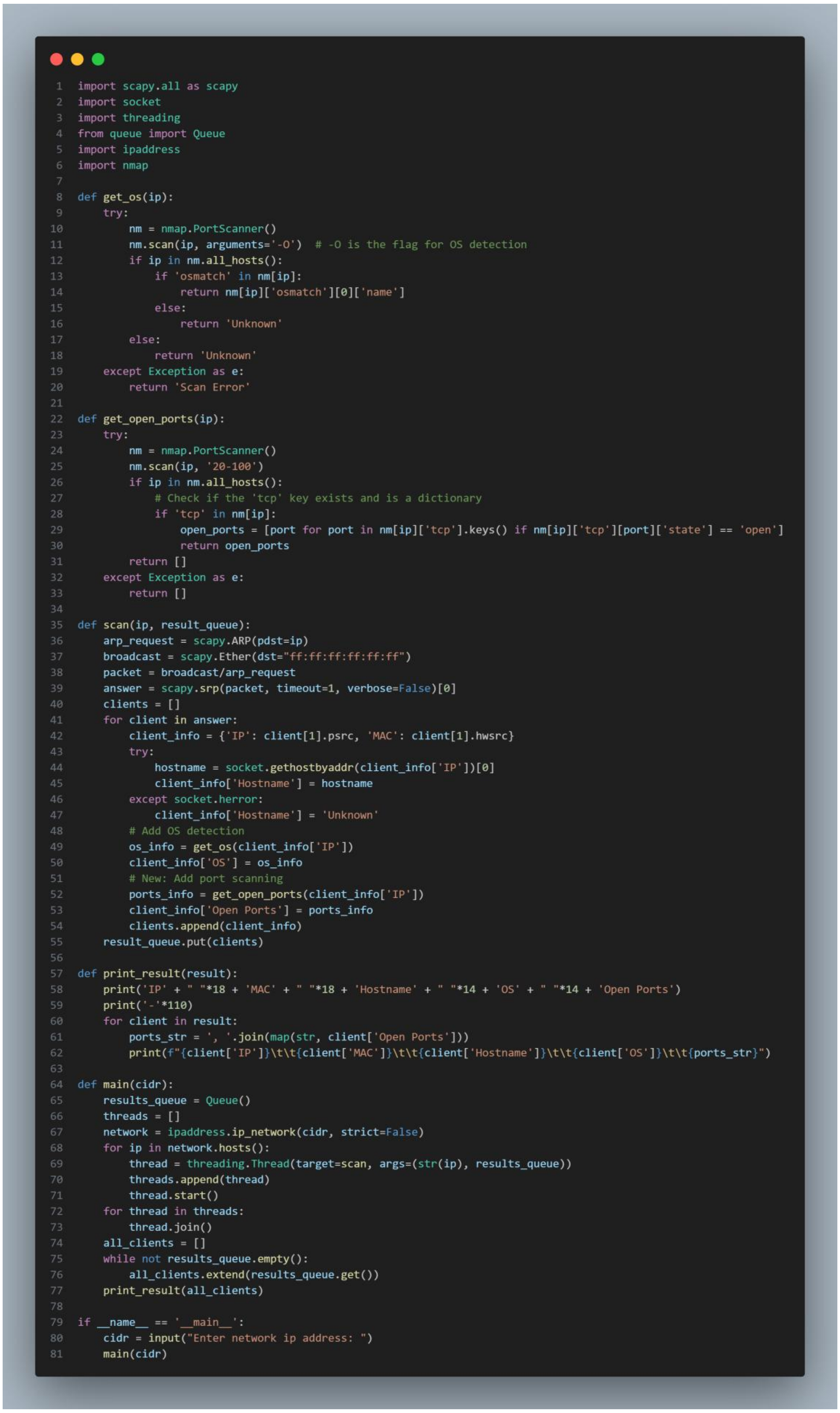
- **Host Discovery**: The tool sends ARP (Address Resolution Protocol) broadcasts to scan a target IP range and identify all active devices on the local network.
- **Hostname Resolution**: For each discovered device, the tool performs a reverse DNS lookup to identify its hostname.
- **OS Detection**: The scanner uses TCP/IP stack fingerprinting with the **python-nmap** library to identify the operating system of each discovered host.
- **Port Scanning**: The tool performs a targeted port scan on a range of common ports to detect active services, which could indicate potential vulnerabilities.
- **Multi-threading**: The final code uses multi-threading to speed up the scanning process by checking each host concurrently.
- **Output**: The results are printed to the console in a clear, formatted table and are available for further use in security assessments.

Modules Installed:

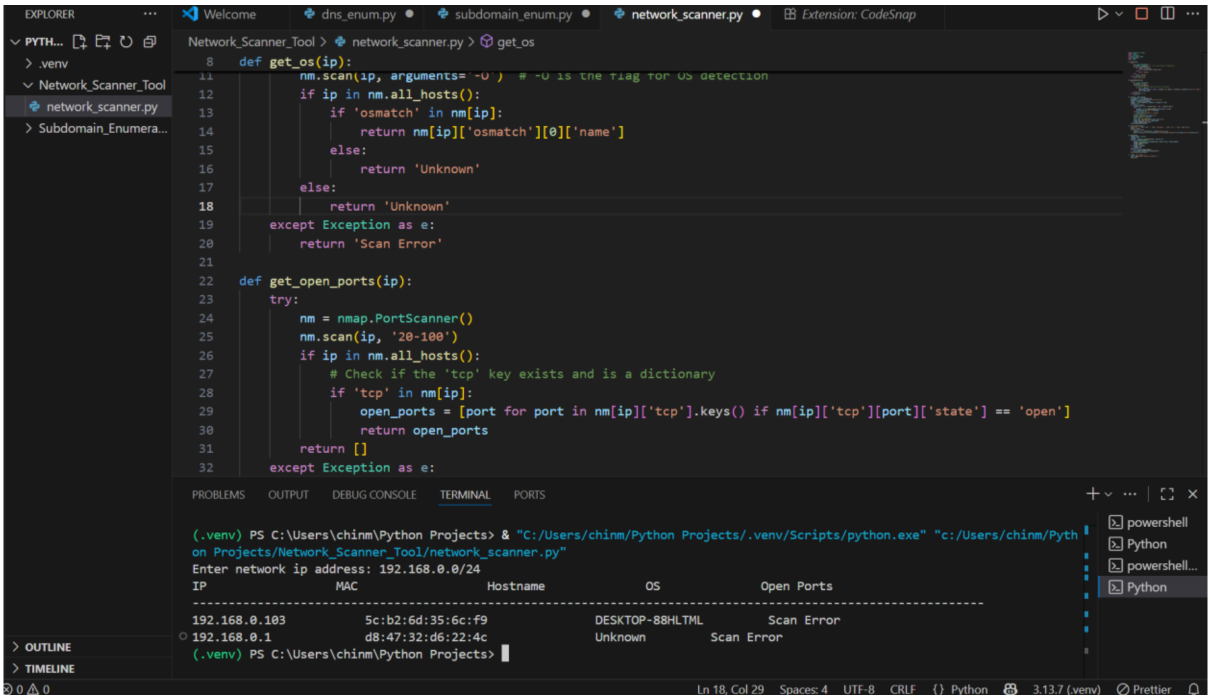
The following Python modules are required to run this project:

- **scapy**: Used for crafting and sending custom network packets at Layer 2 (ARP).
- **socket**: Used for network operations, specifically for hostname resolution.
- **threading**: Used to create and manage multiple threads, enabling parallel scanning for improved performance.
- **nmap**: A Python wrapper for the Nmap tool, used to perform OS detection and port scanning.
- **ipaddress**: Used for handling and iterating over IP addresses within a given CIDR network range.
- **queue**: Used to safely collect results from multiple threads.

CODE:



```python
import scapy.all as scapy
import socket
import threading
from queue import Queue
import ipaddress
import nmap

def get_os(ip):
    try:
        nm = nmap.PortScanner()
        nm.scan(ip, arguments='-O')  # -O is the flag for OS detection
        if ip in nm.all_hosts():
            if 'osmatch' in nm[ip]:
                return nm[ip]['osmatch'][0]['name']
            else:
                return 'Unknown'
        else:
            return 'Unknown'
    except Exception as e:
        return 'Scan Error'

def get_open_ports(ip):
    try:
        nm = nmap.PortScanner()
        nm.scan(ip, '20-100')
        if ip in nm.all_hosts():
            # Check if the 'tcp' key exists and is a dictionary
            if 'tcp' in nm[ip]:
                open_ports = [port for port in nm[ip]['tcp'].keys() if nm[ip]['tcp'][port]['state'] == 'open']
                return open_ports
        return []
    except Exception as e:
        return []

def scan(ip, result_queue):
    arp_request = scapy.ARP(pdst=ip)
    broadcast = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
    packet = broadcast/arp_request
    answer = scapy.srp(packet, timeout=1, verbose=False)[0]
    clients = []
    for client in answer:
        client_info = {'IP': client[1].psrc, 'MAC': client[1].hwsrc}
        try:
            hostname = socket.gethostbyaddr(client_info['IP'])[0]
            client_info['Hostname'] = hostname
        except socket.herror:
            client_info['Hostname'] = 'Unknown'
        # Add OS detection
        os_info = get_os(client_info['IP'])
        client_info['OS'] = os_info
        # New: Add port scanning
        ports_info = get_open_ports(client_info['IP'])
        client_info['Open Ports'] = ports_info
        clients.append(client_info)
    result_queue.put(clients)

def print_result(result):
    print('IP' + " "*18 + 'MAC' + " "*18 + 'Hostname' + " "*14 + 'OS' + " "*14 + 'Open Ports')
    print('-'*110)
    for client in result:
        ports_str = ', '.join(map(str, client['Open Ports']))
        print(f"{client['IP']}\t\t{client['MAC']}\t\t{client['Hostname']}\t\t{client['OS']}\t\t{ports_str}")

def main(cidr):
    results_queue = Queue()
    threads = []
    network = ipaddress.ip_network(cidr, strict=False)
    for ip in network.hosts():
        thread = threading.Thread(target=scan, args=(str(ip), results_queue))
        threads.append(thread)
        thread.start()
    for thread in threads:
        thread.join()
    all_clients = []
    while not results_queue.empty():
        all_clients.extend(results_queue.get())
    print_result(all_clients)

if __name__ == '__main__':
    cidr = input("Enter network ip address: ")
    main(cidr)
```



```python
def get_os(ip):
        nm.scan(ip, arguments='-O')  # -O is the flag for OS detection
        if ip in nm.all_hosts():
            if 'osmatch' in nm[ip]:
                return nm[ip]['osmatch'][0]['name']
            else:
                return 'Unknown'
        else:
            return 'Unknown'
    except Exception as e:
        return 'Scan Error'

def get_open_ports(ip):
    try:
        nm = nmap.PortScanner()
        nm.scan(ip, '20-100')
        if ip in nm.all_hosts():
            # Check if the 'tcp' key exists and is a dictionary
            if 'tcp' in nm[ip]:
                open_ports = [port for port in nm[ip]['tcp'].keys() if nm[ip]['tcp'][port]['state'] == 'open']
                return open_ports
        return []
    except Exception as e:
```

```
(.venv) PS C:\Users\chinm\Python Projects> & "C:/Users/chinm/Python Projects/.venv/Scripts/python.exe" "c:/Users/chinm/Pyth
on Projects/Network_Scanner_Tool/network_scanner.py"
Enter network ip address: 192.168.0.0/24
IP                  MAC                    Hostname            OS              Open Ports
-------------------------------------------------------------------------------------------------------
192.168.0.103          5c:b2:6d:35:6c:f9            DESKTOP-88HLTML          Scan Error
192.168.0.1            d8:47:32:d6:22:4c            Unknown         Scan Error
(.venv) PS C:\Users\chinm\Python Projects> 
```

Code Breakdown:

1. get_os(ip): This function uses nmap with the -O flag to perform OS detection on a single IP address. It returns the detected OS or "Unknown" if the scan fails.

2. get_open_ports(ip): This function uses nmap to perform a basic port scan on a specified range of TCP ports. It returns a list of open ports, handling potential errors and returning an empty list if no ports are open or the scan fails.

3. scan(ip, result_queue): This is the core function for a single host. It performs the initial ARP scan to get the IP and MAC address, then calls socket.gethostbyaddr for the hostname, get_os for the operating system, and get_open_ports for the active services. It then places the comprehensive results into a queue.

4. print_result(result): This function takes the final list of discovered clients and formats it into a clean, readable table with columns for IP, MAC, Hostname, OS, and Open Ports.

5. main(cidr): This is the orchestrator of the project. It takes a network address (in CIDR notation) and creates a separate thread for each IP address on the network. It starts all threads in parallel, waits for them to complete, and then gathers and prints the final results.

6. if __name__ == '__main__': :-This block prompts the user to enter the network IP address to scan and calls the main function to start the process.

Challenges Faced:

- Permissions and Dependencies: A key challenge was the need for elevated permissions and platform-specific drivers (like Npcap) to enable low-level network operations.

- Error Handling and Unexpected Output: Debugging required handling unexpected output from the nmap library, such as empty lists instead of dictionaries, to prevent the script from crashing.

- External Data Limitations: The scanner's output was limited by the network environment itself, as reverse DNS lookups often failed for local IP addresses, resulting in "Unknown" hostnames.

-Chinmayi Sangaraju