Automating NMAP with Python

Background and reason for Topic

- Nmap (Network Mapper): A powerful open-source tool for network discovery and security auditing. Widely used by network administrators, ethical hackers, and IT professionals for tasks such as host discovery, port scanning, and service detection.

- Need for Automation: Manual use of Nmap can be time-consuming for large-scale or repetitive tasks. Automation enhances efficiency, accuracy, and scalability.

- Why Python? Python offers robust libraries (e.g., nmap, python-nmap) to integrate Nmap seamlessly. Its simplicity and flexibility make it ideal for scripting complex tasks.

- Objective: Simplify and automate network scanning processes using Python scripts. Demonstrate real-world applications, including periodic scans (Automating daily or weekly network scans to detect changes or vulnerabilities.)

- Alert Systems: (Trigger alerts when critical vulnerabilities or open ports are found.)

- reporting, and integration systems. (Embedding Nmap into larger security workflows like SIEM (Security Information and Event Management) .

Security Concepts Applied

- Penetration Testing Phase - Reconnaissance
  - Involves gathering information about the target system, network, or organization.
  - Lays the foundation for identifying potential vulnerabilities
- OSI Model
  - Application Layer (Layer 7) nMap identifies services running on open ports - service discovery
  - Transport Layer (Layer 4) Scanning a computer or network to discover open ports and associated services
  - Network Layer (Layer 3) nMap performs host discovery (ping sweeps) to determine which IP addresses are active on the network
- Vulnerability Scanning
  - Automate advanced tasks like vulnerability scanning, information gathering, and testing for security issues using pre-written scripts
- CIA Triad
  - nMap can indirectly affect the confidentiality, integrity, and availability of targets by identifying vulnerabilities that could compromise these aspects

Research

- For research we started by looking over the previous project available online.
- We also referred to various other platforms to help with our project our main focus for research was nmap scanning, automation and python script writing.
- The research mainly focused on port scanning using python script, how to build a python port scanner script.
- We also did our research on how to automate and save nmap scan outputs in csv files.

Demo Preview - what our script does

- The script will scan a network for active hosts, detect open ports and services, and save results in a structured format.
- This script provides real-time feedback during the scan and helps users manage time expectations for long scans.
- We can analyze the data, administrators can identify critical vulnerabilities, misconfigured services, and potential entry points for attackers.
- The automated nature of the tool ensures rapid analysis, making it suitable for both scheduled checks and on-demand use.

Code explanation

```python
import nmap
import csv
import argparse
from datetime import datetime
import time
import requests

# Predefined scan profiles
SCAN_PROFILES = {
    "quick": "-F --reason",  # Fast scan with reasons
    "full": "-p- --reason",  # Scan all ports with reasons
    "web": "-p 80,443,8080 --reason",  # Common web ports with reasons
    "vuln": "--script vuln --reason",  # Vulnerability scan with reasons
}

# Function to get geolocation information for an IP address
def get_geolocation(ip):
    try:
        response = requests.get(f"https://ipinfo.io/{ip}/json")
        if response.status_code == 200:
            data = response.json()
            city = data.get('city', 'Unknown')
            country = data.get('country', 'Unknown')
            return city, country
    except requests.RequestException as e:
        print(f"Error fetching geolocation for {ip}: {e}")
    return 'Unknown', 'Unknown'

def scan_network(subnet, output_file, profile):
    nm = nmap.PortScanner()
    print(f"\nStarting network scan for subnet: {subnet}\n")

    # Determine scan arguments based on profile
    scan_args = SCAN_PROFILES.get(profile, '-sV -O -p- --reason')
    print(f"Using scan profile: {profile} with arguments: {scan_args}\n")
```

Explanation:

- Before running the script “sudo pip3 install python-nmap” is run
  - Installs the python-nmap library
  - Enabling automated nmap python scripts to be run on our system
- Importing essential libraries and modules
- Predefined Scan Profiles
  - SCAN_PROFILES Dictionary — Nmap scan profiles for different purposes.
    - Quick: Brief overview with justifications for host states.
    - Full: Scans all 65,535 ports.
    - Web: Common web ports (80, 443, 8080).
    - Vuln : Run Nmap vulnerability scripts.
- Geolocation Retrieval
  - get_geolocation(ip): Retrieves city and country information via the ipinfo. io API.
  - Gracefully handles exceptions for the API failures

Code Explanation

```
# Perform a ping scan to identify live hosts
print(f"Scanning for live hosts...\n")
nm.scan(hosts=subnet, arguments='-sn')
live_hosts = [host for host in nm.all_hosts() if nm[host].state() == 'up']
if not live_hosts:
    print("No live hosts found. Exiting.\n")
    return

print(f"Found {len(live_hosts)} live hosts:\n")
for idx, host in enumerate(live_hosts, 1):
    print(f"  {idx}. {host}")

results = []
total_hosts = len(live_hosts)
start_time = time.time()

for idx, host in enumerate(live_hosts, 1):
    print(f"\nPerforming detailed scan on host {idx}/{total_hosts}: {host}...")
    nm.scan(host, arguments=scan_args)

    # Calculate elapsed time and remaining time
    elapsed_time = time.time() - start_time
    avg_time_per_host = elapsed_time / idx
    remaining_time = avg_time_per_host * (total_hosts - idx)
    print(f"  Estimated time remaining: {int(remaining_time)} seconds")
```

Explanation:

- Ping Scan for Live Hosts
  - Uses the -sn option in nmap to find hosts on the subnet.
  - Derives the name of the host also known as "up" (reachable).
- If No Live Hosts Found
  - Exits with a message immediately if there are no live hosts.
- Detailed Scans on Live Hosts
  - All the arguments you would like to set for scanning of each live host.
  - Progress Updates: Displays current host index and total.
- Time Estimation
  - Measures elapsed time and computes average time from each host.
  - Calculates and shows remaining time.

Code Explanation

```
# Get geolocation information
city, country = get_geolocation(host)
os_match = nm[host]['osmatch'][0]['name'] if 'osmatch' in nm[host] and nm[host]['osmatch'] else 'Unknown'

scanned_ports = set()
for proto in nm[host].all_protocols():
    ports = nm[host][proto].keys()
    scanned_ports.update(ports)
    for port in sorted(ports):
        service = nm[host][proto][port]
        result = {
            'host': host,
            'port': port,
            'protocol': proto,
            'state': service['state'],
            'service': service['name'],
            'version': service['version'] or 'Unknown',
            'os': os_match,
            'city': city,
            'country': country
        }
        results.append(result)
        # Print each result with geolocation info
        print(f"    Host: {result['host']}")
        print(f"      Port: {result['port']}/{result['protocol']} - {result['state']}")
        print(f"      Service: {result['service']} (Version: {result['version']})")
        print(f"      OS: {result['os']}")
        print(f"      Location: {result['city']}, {result['country']}\n")
```

Explanation:

- Geolocation Retrieval
  - Gets city and country for each host's IP
- OS Detection
  - OS Details Extract from Nmap result, Unknown if not found
- Port and Protocol Analysis
  - Maps open ports of the identified protocols—like TCP, UDP, etc.
- Service & Version Detection
  - Identifies service name and version on a per port basis, showing "Unknown" where not found.
- Result Compilation
  - Building structured dictionaries with the gathered data
- Real-Time Output
  - shows information about each host:
    - Host IP, ports open, services, versions, OS, geolocation.

Code Explained

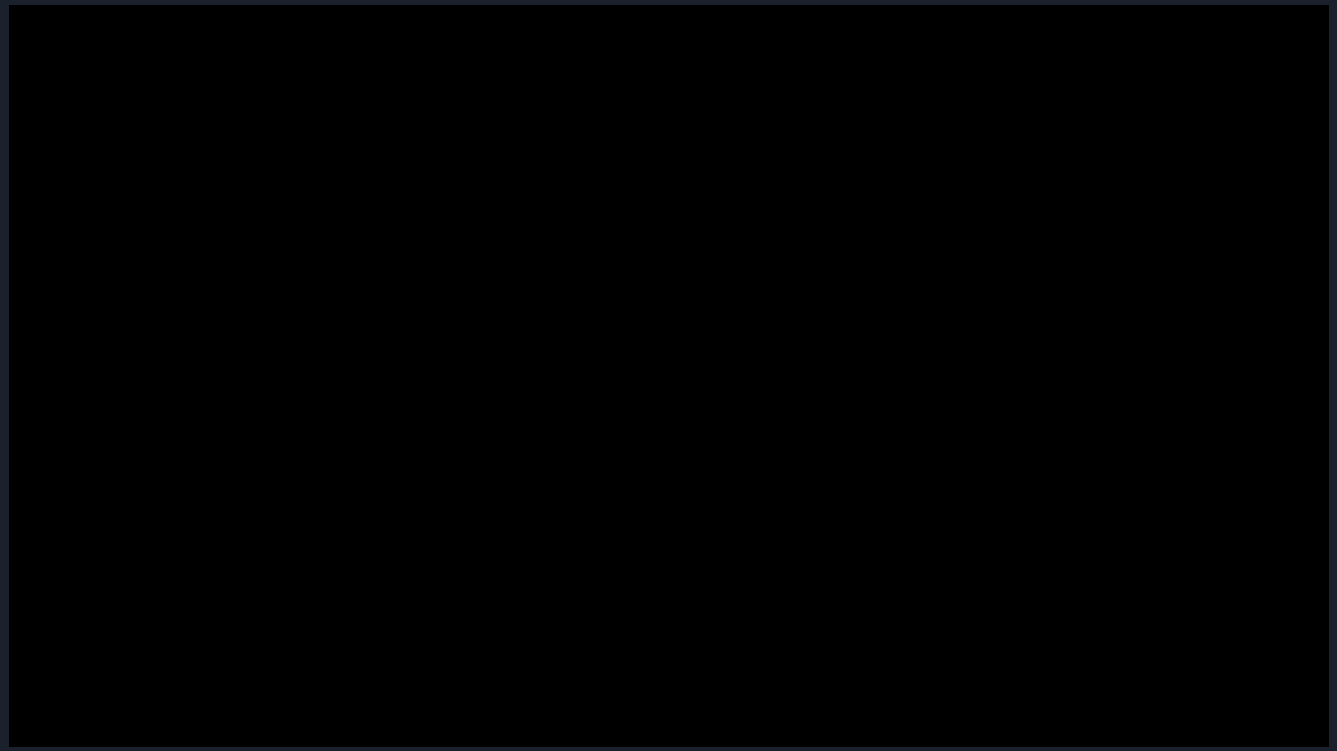
```
        closed_ports = all_ports - scanned_ports
        for port in sorted(closed_ports):
            result = {
                'host': host,
                'port': port,
                'protocol': 'tcp',
                'state': 'closed',
                'service': 'N/A',
                'version': 'N/A',
                'os': os_match,
                'city': city,
                'country': country
            }
            results.append(result)

    save_results(results, output_file)

def save_results(results, output_file):
    print(f"\nSaving results to {output_file}...")
    with open(output_file, 'w', newline='') as csvfile:
        fieldnames = ['host', 'port', 'protocol', 'state', 'service', 'version', 'os', 'city', 'country']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()
        for row in results:
            writer.writerow(row)
    print(f"Results successfully saved to {output_file}\n")
```

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Nmap automation script with scan progress")
    parser.add_argument("subnet", help="Subnet to scan (e.g., 192.168.1.0/24)")
    parser.add_argument("-o", "--output", help="Output file name", default=f"nmap_scan_{datetime.now().strftime('%Y%m%d_%H%M%S')}.csv")
    parser.add_argument("-p", "--profile", help=f"Scan profile to use (options: {', '.join(SCAN_PROFILES.keys())})", default="quick")
    args = parser.parse_args()
    scan_network(args.subnet, args.output, args.profile)
```

Explanation:

- Closed Ports Detection
  - Full (1–65,535) versus common (1–1,024) port ranges per scan profile.
  - Determines which ports are closed by subtracting scanned ports from the 0-65535 range.
- Appending Closed Port Details
  - Results will include additional information (host, port, protocol, state, etc.) about the closed ports.
- Results Saving
  - Output all results (open and closed ports) to a CSV file.
  - Host, port, protocol, state, service, version, OS and geolocation fields included.
- Command-Line Interface
  - Lets user define their own subnet, output file, scan profile, etc.
  - Default: "Quick" profile with timestamped CSV output.

Demo Video



Demo Summary

Explanation:

- Running The Script
  - Starts a nmap scan for the subnet: 45.33.32.156
  - Using scan profile: quic with arguments: -sV -O -p- --reason
    - -sV: Detects versions of services running on open ports.
    - -O: Attempts to identify the operating system of the target.
    - -p-: Scans all 65,535 TCP ports.
    - --reason: Provides the reason why each port is considered open, closed, or filtered.

- Visually Shown Output Of Script
  - Outputs the location (Fremont, US), OS (Linux) and shows the open ports
    - Port 22/tcp: SSH (Version: 6.6.1p1 Ubuntu 2ubuntu2.13)
    - Port 80/tcp: HTTP (version unknown)
    - Port 9929/tcp: nping-echo (version unknown)
    - Port 31337/tcp: tcpwrapped (version unknown)

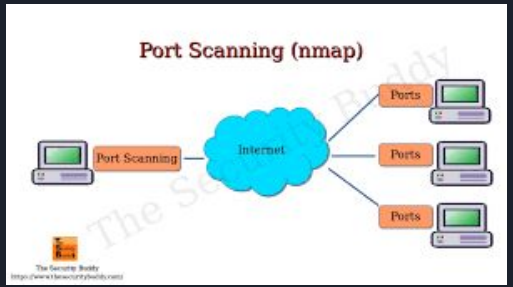
- Information Saved To File
  - The output information then gets saved to “output.csv” to keep the information gathered readable and easy to deliver over.

Impact & Benefits

- Impact:
  - Greater Efficiency: Automates complex Nmap scans making it easy and quick.
  - In-depth Information: Gather detailed information on open/closed ports, services, OS, geolocation, etc.
  - Scalability: Scales with part of the network, from small subnets to full scale scan.
  - Precision Improvement: Automating set up and information gathering minimizes wrong doing.

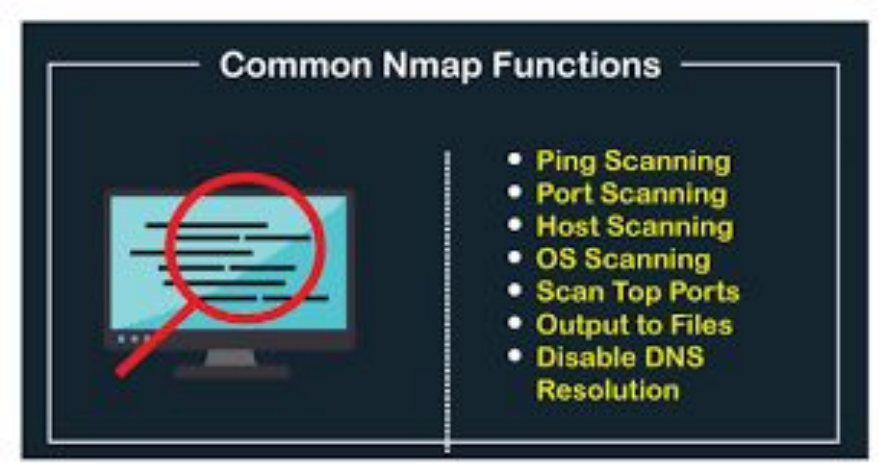
- Benefits:

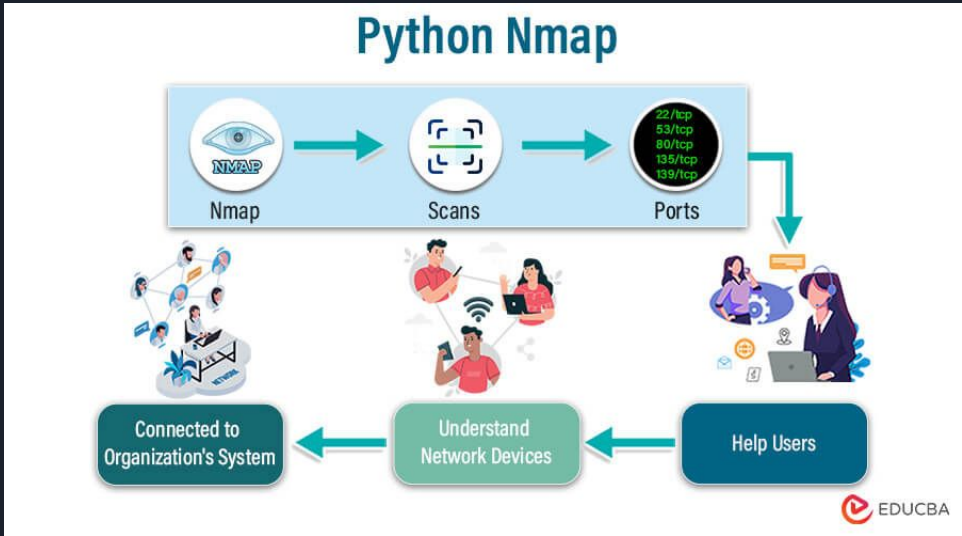
  - Cost- Open source and free making it's accessibility to users very friendly.
  - Time-Saving – Real-time feedback and progress tracking make easier to monitor and scan efficiently.
  - Generates Report — Creates structured CSV files for easy data analysis for audit compliance.
  - Ease of Use: By using CLI options, its highly user-friendly for technical and non-technical users.
  - Security-Driven: It detects vulnerabilities and contextualizes them with geolocation insights.



Conclusion

Nmap is a powerful tool for network scanning and can be used to identify open ports, hosts and vulnerabilities on a network. By using Nmap with python, you can automate the scanning process and integrate it into your existing security tools. Nmap has a wide range of uses from finding live hosts, port scanning to OS detection and is becoming increasingly used because of its ability to monitor traffic between web servers and IoT devices. NMap is also used to interrogate botnets which connect to devices using UPnP protocol. The rising complexity means as a cyber security professional we must understand how to use Nmap and it's viability to perform ethical hacking and for what illegal purposes as well as there is a huge responsibility with corporations and liabilities which arises with it for users to remain within the boundary of the legal grounds.





```
import nmap
import csv
import argparse
from datetime import datetime
import time
import requests
# Predefined scan profiles
SCAN_PROFILES = {
    "quick": "-F --reason",  # Fast scan with reasons
    "full": "-p- --reason",  # Scan all ports with reasons
    "web": "-p 80,443,8080 --reason",  # Common web ports with reasons
    "vuln": "--script vuln --reason",  # Vulnerability scan with reasons
}
# Function to get geolocation information for an IP address
def get_geolocation(ip):
    try:
        response = requests.get(f"https://ipinfo.io/{ip}/json")
        if response.status_code == 200:
            data = response.json()
            city = data.get('city', 'Unknown')
            country = data.get('country', 'Unknown')
            return city, country
    except requests.RequestException as e:
        print(f"Error fetching geolocation for {ip}: {e}")
    return 'Unknown', 'Unknown'
def scan_network(subnet, output_file, profile):
    nm = nmap.PortScanner()
    print(f"\nStarting network scan for subnet: {subnet}\n")
    # Determine scan arguments based on profile
    scan_args = SCAN_PROFILES.get(profile, '-sV -O -p- --reason')
    print(f"Using scan profile: {profile} with arguments: {scan_args}\n")
    # Perform a ping scan to identify live hosts
    print(f'Scanning for live hosts...\n")
    nm.scan(hosts=subnet, arguments='-sn')
    live_hosts = [host for host in nm.all_hosts() if nm[host].state() == 'up']
    if not live_hosts:
        print("No live hosts found. Exiting.\n")
        return
    print(f"Found {len(live_hosts)} live hosts:\n")
    for idx, host in enumerate(live_hosts, 1):
        print(f"  {idx}. {host}")
    results = []
    total_hosts = len(live_hosts)
    start_time = time.time()
```

```
    for idx, host in enumerate(live_hosts, 1):
        print(f"\nPerforming detailed scan on host {idx}/{total_hosts}: {host}...")
        nm.scan(host, arguments=scan_args)
        # Calculate elapsed time and remaining time
        elapsed_time = time.time() - start_time
        avg_time_per_host = elapsed_time / idx
        remaining_time = avg_time_per_host * (total_hosts - idx)
        print(f"  Estimated time remaining: {int(remaining_time)} seconds")
        # Get geolocation information
        city, country = get_geolocation(host)
        os_match = nm[host]['osmatch'][0]['name'] if 'osmatch' in nm[host] and nm[host]['osmatch']
else 'Unknown'
        scanned_ports = set()
        for proto in nm[host].all_protocols():
            ports = nm[host][proto].keys()
            scanned_ports.update(ports)
            for port in sorted(ports):
                service = nm[host][proto][port]
                result = {
                    'host': host,
                    'port': port,
                    'protocol': proto,
                    'state': service['state'],
                    'service': service['name'],
                    'version': service['version'] or 'Unknown',
                    'os': os_match,
                    'city': city,
                    'country': country
                }
                results.append(result)
                # Print each result with geolocation info
                print(f"    Host: {result['host']}")
                print(f"      Port: {result['port']}/{result['protocol']} - {result['state']}")
                print(f"      Service: {result['service']} (Version: {result['version']})")
                print(f"      OS: {result['os']}")
                print(f"      Location: {result['city']}, {result['country']}\n")
        # Add closed ports explicitly
        if "full" in scan_args or "-p-" in scan_args:
            all_ports = set(range(1, 65536))  # Full range of ports
        else:
            all_ports = set(range(1, 1025))  # Common ports range
        closed_ports = all_ports - scanned_ports
        for port in sorted(closed_ports):
            result = {
```

```
                'host': host,
                'port': port,
                'protocol': 'tcp',
                'state': 'closed',
                'service': 'N/A',
                'version': 'N/A',
                'os': os_match,
                'city': city,
                'country': country
            }
            results.append(result)
    save_results(results, output_file)
def save_results(results, output_file):
    print(f"\nSaving results to {output_file}...")
    with open(output_file, 'w', newline='') as csvfile:
        fieldnames = ['host', 'port', 'protocol', 'state', 'service', 'version', 'os', 'city', 'country']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()
        for row in results:
            writer.writerow(row)
    print(f"Results successfully saved to {output_file}\n")
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Nmap automation script with scan progress")
    parser.add_argument("subnet", help="Subnet to scan (e.g., 192.168.1.0/24)")
    parser.add_argument("-o", "--output", help="Output file name",
default=f"nmap_scan_{datetime.now().strftime('%Y%m%d_%H%M%S')}.csv")
    parser.add_argument("-p", "--profile", help=f"Scan profile to use (options: {',
'.join(SCAN_PROFILES.keys())})", default="quick")
    args = parser.parse_args()
    scan_network(args.subnet, args.output, args.profile)
```