NMAP – Network Exploration And Security Auditing: An Automated Python Approach

Harinarayanan R[1], Himanis Hariharha Kawsikan K S[2], Haswin T[3], Shiva Shankaran S[4], Dr. V. Ravindra Krishna Chandar[5]

[1, 2, 3, 4, 5] Dept of Cyber security

[1, 2, 3, 4, 5] Dhanalakshmi Srinivasan University

Abstract- *Maintaining visibility over an expanding network infrastructure is one of the biggest challenges faced by modern cybersecurity teams. Without regular auditing, open ports, outdated services, and unpatched operating systems become easy targets for attackers. For this mini-project, our team focused on automating network reconnaissance using Nmap, a powerful and widely used open-source security tool. Running Nmap commands manually for multiple targets is tedious, time-consuming, and prone to human error. To solve this, we developed a complete Python automation script that runs comprehensive network scans and generates structured security audit reports with a single click. By utilizing the 'python-nmap' library, our system seamlessly performs host discovery, TCP/UDP port scanning, service version detection, OS fingerprinting, and automated vulnerability checking using the Nmap Scripting Engine (NSE). Instead of presenting raw, complex terminal data, our script intelligently categorizes open ports into SAFE, RISK, or VULNERABLE statuses and outputs an easy-to-read text report. Testing in a controlled virtual environment against vulnerable targets like Metasploitable 2 proved that our tool accurately identifies active CVEs (Common Vulnerabilities and Exposures) much faster than manual auditing, making it highly effective for routine network security management.*

Keywords: Nmap, Network Exploration, Security Auditing, Python Automation, Vulnerability Scanning, CVE, Port Scanning, Operating System Fingerprinting.

I. INTRODUCTION

In the cybersecurity world, you cannot protect what you cannot see. Today's organizations operate complex networks with dozens or hundreds of connected devices, including servers, laptops, and IoT gadgets. Each device exposes certain network services (like HTTP, SSH, or FTP), and each of these services represents a potential door for a hacker to enter.

The fundamental problem is that administrators often lose track of their own networks. Devices are constantly added or removed, software gets outdated, and employees sometimes open unauthorized ports for testing and forget to close them. If these exposed ports have known weaknesses (CVEs), an attacker can easily break into the network. This is why regular network security auditing is a mandatory practice for any secure organization.

To perform these audits, professionals worldwide use Nmap (Network Mapper). Created in 1997, Nmap has grown from a simple port scanner into an advanced security tool capable of finding active hosts, identifying running services, guessing operating systems, and even running scripts to check for specific vulnerabilities. However, running Nmap manually involves typing long command-line arguments, waiting for the scan to finish, and then trying to understand hundreds of lines of raw text output.

For our mini-project, our team wanted to bridge the gap between Nmap's raw power and ease of use. We decided to build a Python automation layer on top of Nmap. Our goal was to create a "one-click" auditing tool that can take a target IP address, run all the complex Nmap scans automatically in the background, and generate a clean, structured security report. By automating this process, we prove that routine security checks can be made faster, easier, and much more consistent.

II. LITERATURE REVIEW

Before developing our automated tool, we reviewed how network auditing is currently performed and studied the tools available to security professionals.

First, we looked at manual scanning techniques. Traditional methods involve using basic command-line tools like 'ping' to find hosts and 'netcat' to check if a port is open. While these work, they provide very little information. They cannot tell you what version of a service is running or if it is vulnerable to a known exploit.

Next, we studied Enterprise Vulnerability Scanners. Tools like Nessus and OpenVAS are incredibly powerful and provide beautiful graphical reports. However, Nessus requires an expensive commercial license, making it inaccessible for

students and small businesses. OpenVAS is free, but it is very heavy, requires a dedicated server setup, and uses a massive amount of system memory.

We then researched Nmap in-depth. According to the official Nmap Project Guide by Gordon Lyon, Nmap uses a technique called TCP SYN scanning (stealth scanning) to rapidly check thousands of ports without fully establishing a connection. Furthermore, Nmap contains an advanced feature called the Nmap Scripting Engine (NSE), which uses Lua scripts to interact with services and detect vulnerabilities. We realized Nmap had all the power of Nessus, but lacked the automated reporting.

To connect Python with Nmap, we discovered the 'python-nmap' library. Literature on this library showed that it can run Nmap silently and convert the complex XML output into Python dictionaries. Based on our research, we concluded that combining Nmap's scanning engine with Python's data-handling capabilities would be the perfect way to build a lightweight, free, and highly effective security auditor.

III. SYSTEM REQUIREMENTS

Since our project acts as a wrapper around Nmap, we wanted to ensure it could run on standard academic laptops without needing heavy server equipment.

A. Hardware Requirements

The hardware needed to run our Python automation tool is very basic:

• Processor: An Intel Core i3 or equivalent is sufficient.
• Memory (RAM): 4 GB is the minimum requirement, though 8 GB is recommended if scanning a large network range (like an entire college Wi-Fi subnet).
• Storage: Only about 200 MB of free space is needed to install Nmap, Python, and store the generated text reports.
• Network Interface: A standard Wi-Fi or Ethernet card. To perform fast SYN stealth scans, the adapter must support raw packet transmission.

B. Software Requirements

We built the tool using entirely free and open-source software:

• Nmap Core: Nmap version 7.x or higher must be installed on the host machine. On Windows, the installer includes 'Npcap', which is required for raw packet sniffing.
• Programming Language: Python 3.8 or higher.
• Python Libraries: The project relies heavily on the 'python-nmap' library (installed via pip). We also used built-in libraries like 'subprocess' to run background commands, and 'datetime' to timestamp our reports.

Important Note: To run the tool effectively, the user must open their Command Prompt or Terminal as an Administrator (or Root). If run as a normal user, Nmap is forced to use slower TCP Connect scans instead of stealthy SYN scans.

IV. PROPOSED SYSTEM ARCHITECTURE

To make the code clean and easy to manage, we divided our Python project into five logical modules. Each module performs a specific task in the auditing pipeline.

A. Module 1: Target Input and Menu

This is the user interface of our tool. When the script starts, it asks the user to enter a target IP address or a subnet range (like 192.168.1.0/24). It then presents a simple menu where the user can choose to run a specific scan (like just checking for OS versions) or run the Full Automated Audit. This module initializes the python-nmap 'PortScanner' object and routes the user's choice to the correct function.

B. Module 2: Host Discovery and Port Scanning

If a full scan is chosen, this module runs first. It uses the Nmap command arguments '-sn' to perform a ping sweep. This tells us exactly which devices are currently turned on and connected to the network.

Once the active devices are found, the script automatically launches a port scan using the '-sS' (TCP SYN) and '-sU' (UDP) arguments. We programmed it to scan the top 1000 most common ports to save time. This module collects data on which ports are 'open', 'closed', or 'filtered' by a firewall.

C. Module 3: Service Detection and OS Fingerprinting

Knowing a port is open isn't enough; we need to know what software is running on it. This module runs Nmap with the '-sV' argument. Nmap sends special probe packets to the open ports and compares the responses to a database of 11,000 signatures to identify the exact software version (e.g., Apache 2.4.49).

At the same time, the script uses the '-O' argument for OS detection. By analyzing tiny details in how the target computer formats its TCP/IP packets (like Time-To-Live and

Window Size), Nmap can accurately guess if the target is running Windows 10, Ubuntu Linux, or a macOS device, and our Python script records the accuracy percentage.

D. Module 4: NSE Vulnerability Scanning

This is the most critical security module. After gathering the service versions, our script triggers the Nmap Scripting Engine using the '--script vuln' argument. This forces Nmap to run hundreds of community-written Lua scripts against the open ports to check for known Common Vulnerabilities and Exposures (CVEs). If a service is outdated or misconfigured, this module captures the exact CVE ID (like CVE-2014-0160 for Heartbleed) and the risk description.

E. Module 5: Automation & Report Generation

The final module acts as the brain that ties everything together. We programmed a logic system called the 'Trust Score' classifier to make the report easy to read:

• VULNERABLE: If the NSE script found a CVE, the port is marked as Vulnerable.
• RISK: If no CVE is found, but the port uses a dangerous cleartext protocol (like Telnet on port 23 or FTP on port 21), it is marked as a Risk.
• SAFE: If the service is up-to-date and secure, it is marked Safe.

The Python script organizes all this data, adds a timestamp, and uses standard file writing techniques to save a clean '.txt' audit report to the computer.

V. TESTING AND EXPERIMENTAL SETUP

To prove that our automation script works correctly, we needed to test it on real systems. However, scanning unauthorized networks is illegal. Therefore, we set up a safe, isolated virtual laboratory using Oracle VirtualBox on a laptop.
Our test network consisted of three virtual machines:

1. The Auditor: A Kali Linux VM running our Python script.
2. The Safe Target: A fully updated Windows 10 VM with a firewall enabled.
3. The Vulnerable Target: A VM running 'Metasploitable 2', which is a special Linux system intentionally loaded with outdated software and security holes for students to practice on.

A. Attack Simulation Test Cases

We ran our script against the Metasploitable 2 machine to see if it would catch the deliberate vulnerabilities without any manual help from us.

Test 1 - Host and Port Detection: The script successfully found the Metasploitable IP and correctly listed 23 different open ports, including FTP, SSH, Telnet, and MySQL.

Test 2 - Service Version Accuracy: The script accurately identified that the FTP port was running 'vsftpd 2.3.4', which is a famously compromised version of the software.

Test 3 - Vulnerability Catching: The NSE module of our script worked perfectly. It cross-referenced the services and automatically generated alerts for major flaws, including the Samba MS-RPC flaw (CVE-2007-2447) and a critical backdoor in the FTP service (CVE-2011-2523).

Test 4 - Safe Network Check: When we ran the script against the updated Windows 10 VM, it correctly identified the OS, found only a few standard open ports, and correctly classified them as SAFE, proving our tool does not create false alarms.

VI. RESULTS AND DISCUSSION

The results of our mini-project were highly successful. We proved that we could take a complex command-line hacking tool and turn it into a streamlined, automated auditing program.

The most impressive result was the speed and efficiency of the automation. When we tried to scan the Metasploitable machine manually by typing individual Nmap commands for OS detection, then service detection, and then vulnerability scanning, it took us nearly 45 minutes to finish and write down the notes. When we ran our Python script, it executed all those phases perfectly in the background and generated a fully written text report in just 4 minutes and 32 seconds.

Below is a summarized example of how the script formats its final output, making it extremely easy for a network administrator to understand the threat level:

```
==============================
NMAP SECURITY AUDIT REPORT
==============================
Date   : 2026-05-20  10:45:02
Target : 192.168.1.50
HOST STATUS  : UP
OS DETECTED  : Linux 2.6.x [92% accuracy]
OPEN PORTS:
```

```
21  - vsftpd 2.3.4          [ VULNERABLE ] CVE-2011-2523
22  - OpenSSH 4.7p1         [ RISK - Outdated version ]
23  - Linux telnetd         [ RISK - Cleartext protocol ]
80  - Apache httpd 2.2.8    [ VULNERABLE ] CVE-2007-6750
3306 - MySQL 5.0.51a        [ RISK - Exposed DB Port ]
================================
```

The inclusion of the SAFE, RISK, and VULNERABLE tags is a major improvement over raw Nmap output. It means that even a junior IT staff member can run our Python tool, look at the text file, and instantly know which servers need to be patched immediately. The system reliably caught 100% of the critical CVEs on our test target.

VII. CONCLUSION AND FUTURE SCOPE

Through this mini-project, our team learned deep practical skills regarding network protocols, Python programming, and automated security auditing. We successfully achieved our objective of building an end-to-end vulnerability scanner. By using the 'python-nmap' library, we eliminated the tedious manual work associated with network reconnaissance.

Our tool proves that organizations do not necessarily need to spend thousands of dollars on enterprise vulnerability scanners to maintain basic network hygiene. A well-written Python script combined with the power of Nmap's open-source engine is more than capable of discovering rogue devices, detecting outdated software, and highlighting critical CVEs before hackers can exploit them.

For future enhancements, we have several exciting ideas. First, we would like to convert the plain-text report output into a PDF format using Python's 'ReportLab' library to make it look even more professional. Second, we plan to schedule the script using Windows Task Scheduler or Linux Cron jobs so it runs automatically every midnight, with an added module that emails the report to the administrator by morning. Finally, building a simple web-based GUI using Python Flask would allow users to run scans from their web browser instead of the command prompt. Overall, this project serves as a strong foundation for automated cybersecurity defense.

REFERENCES

[1] Gordon Lyon (Fyodor), "Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning," Nmap Project, 2009. Available at: https://nmap.org/book/

[2] Alexandre Norman, "python-nmap: Use nmap from Python," python-nmap documentation, PyPI, 2021. Available at: https://pypi.org/project/python-nmap/

[3] [3] MITRE Corporation, "Common Vulnerabilities and Exposures (CVE) Program," cve.mitre.org, 2024.

[4] [4] NIST National Vulnerability Database (NVD), nvd.nist.gov, 2024.

[5] NIST Special Publication 800-115: Technical Guide to Information Security Testing and Assessment, 2008.

[6] PTES (Penetration Testing Execution Standard), Technical Guidelines, 2012. Available at: http://www.pentest-standard.org/

[7] S. Bellovin and W. Cheswick, "Network Firewalls," IEEE Communications Magazine, 1994.

[8] Kali Linux Documentation, "Network Scanning with Nmap," Offensive Security, 2024. Available at: https://www.kali.org/tools/nmap/