Title: Networking Fundamentals, Nmap Scanning, and Automation Scripting

1. Introduction

This task focuses on understanding basic networking concepts, using Nmap for host and port discovery, and automating scans with Python. The work was performed in a controlled lab environment on an authorized target system. The objectives were to identify open ports and running services, evaluate potential security risks, and generate automated scan reports using a Python script that integrates with Nmap.

2. Lab Setup

The lab environment consists of a Windows host machine running Kali Linux as a guest operating system inside VirtualBox virtualization software. Kali Linux, This setup leverages VirtualBox's host-only or NAT networking modes to create an isolated lab network, ensuring scans remain contained within the virtual environment and do not affect the host Windows network or external systems.

The target is Metasploitable 2, a deliberately vulnerable virtual machine distributed by Rapid7 for security training and tool testing. Metasploitable exposes numerous outdated services across common ports (e.g., FTP, SSH, HTTP, SMB) with known exploits, making it ideal for practicing reconnaissance techniques like Nmap scanning.

3. Target

Item	Details
Target Machine	Host A (Metasploitable)
Target IP	192.168.56.7

4. Attacker

Item	Details
Attacker Machine	Kali Linux (Virtual Box)
Attacker IP	192.168.56.3

5. Tools

- Nmap
- Python 3
- python-nmap
- draw.io

6. Nmap Installation and Verification

Nmap was installed on the scanner host using the system's package manager or official installer appropriate for the operating system. After installation, Nmap was verified by checking the version and running a verbose help command to ensure the tool was correctly configured.

commands :

- nmap -v
- nmap --version

7. Nmap Scan Execution

Scan Commands Used

All scans were executed only against the authorized lab target. From this scan we can obtain several TCP and UDP ports were discovered in the open state, along with their associated services and, where possible, version information. The following commands were used, and the results were saved to text files for later analysis:

 # SYN scan (TCP half-open)

- *nmap -sS <target_ip> -oN syn_scan.txt*

 # TCP connect scan

- *nmap -sT <target_ip> -oN tcp_scan.txt*

 # UDP scan

- *nmap -sU <target_ip> -oN udp_scan.txt*

The SYN scan −sS−sS sends TCP SYN packets and infers port states from the responses without completing the full TCP handshake, making it relatively stealthy and fast. The TCP connect scan −sT−sT relies on the operating system's connect() call to complete full TCP connections, which is easier to perform but more likely to be logged. The UDP scan −sU−sU sends UDP probes and interprets responses or lack of responses, which can be slower due to timeouts but is necessary to discover UDP services.

8. Scan Result

1. SYN Scan - *nmap -sS 192.168.56.7 -oN syn_scan.txt*

The SYN scan −sS−sS sends TCP SYN packets and infers port states from the responses without completing the full TCP handshake, making it relatively stealthy and fast.

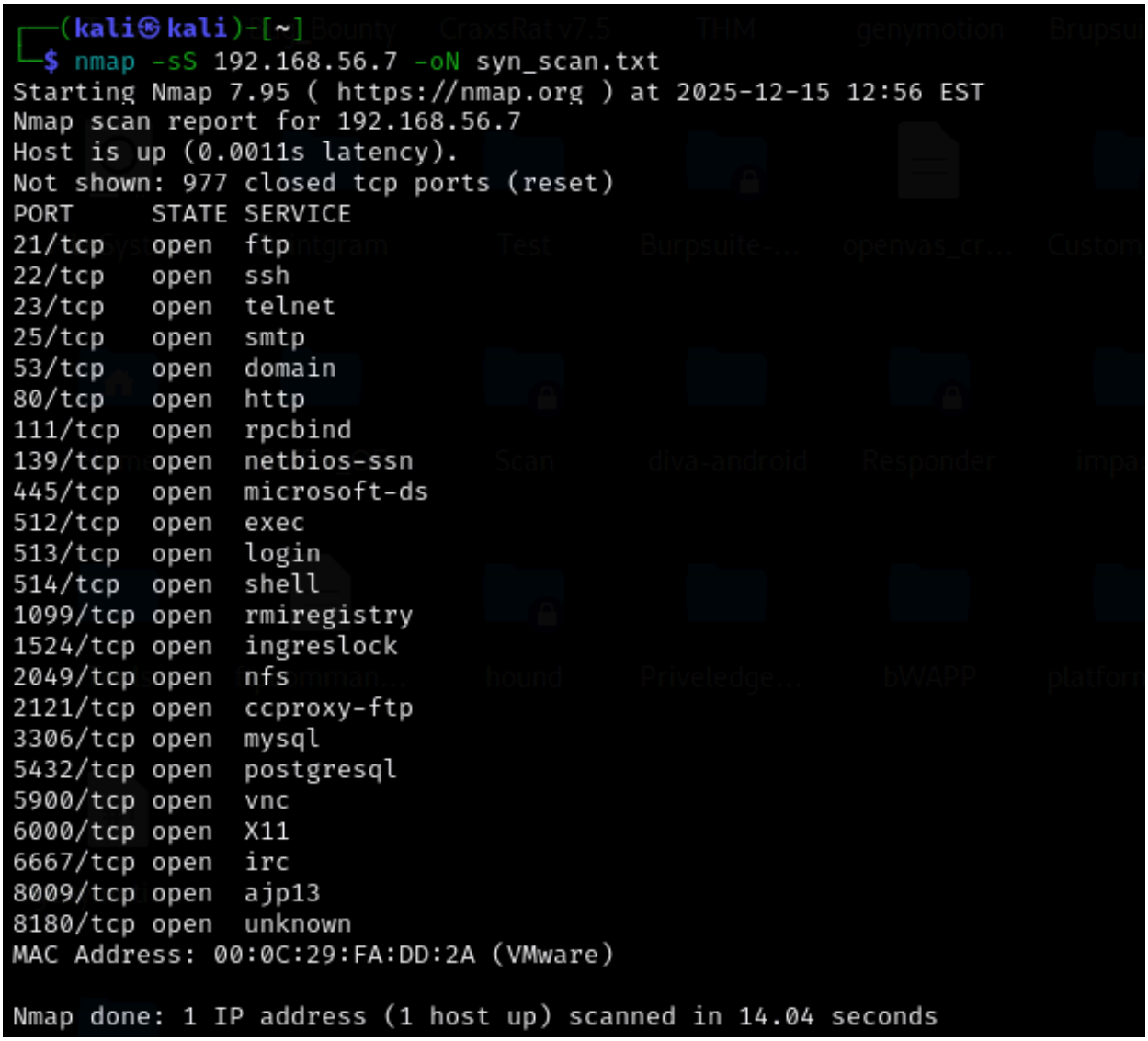


Fig.1

Raw outputs for each command were saved as syn_scan.txt

2. TCP Scan - *nmap -sT 192.168.56.7 -oN tcp_scan.txt*

The TCP connect scan −sT−sT relies on the operating system's connect() call to complete full TCP connections, which is easier to perform but more likely to be logged.

```
┌──(kali㉿kali)-[~]
└─$ nmap -sT 192.168.56.7 -oN tcp_scan.txt
Starting Nmap 7.95 ( https://nmap.org ) at 2025-12-15 13:10 EST
Stats: 0:00:03 elapsed; 0 hosts completed (0 up), 1 undergoing ARP Ping Scan
Parallel DNS resolution of 1 host. Timing: About 0.00% done
Nmap scan report for 192.168.56.7
Host is up (0.0085s latency).
Not shown: 977 closed tcp ports (conn-refused)
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
23/tcp   open  telnet
25/tcp   open  smtp
53/tcp   open  domain
80/tcp   open  http
111/tcp  open  rpcbind
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
512/tcp  open  exec
513/tcp  open  login
514/tcp  open  shell
1099/tcp open  rmiregistry
1524/tcp open  ingreslock
2049/tcp open  nfs
2121/tcp open  ccproxy-ftp
3306/tcp open  mysql
5432/tcp open  postgresql
5900/tcp open  vnc
6000/tcp open  X11
6667/tcp open  irc
8009/tcp open  ajp13
8180/tcp open  unknown
MAC Address: 00:0C:29:FA:DD:2A (VMware)

Nmap done: 1 IP address (1 host up) scanned in 13.44 seconds
```

Fig.2

Raw outputs for each command were saved as tcp_scan.txt

3. UDP Scan - *nmap -sU 192.168.56.7 -oN udp_scan.txt*

The UDP scan −sU−sU sends UDP probes and interprets responses or lack of responses, which can be slower due to timeouts but is necessary to discover UDP services.

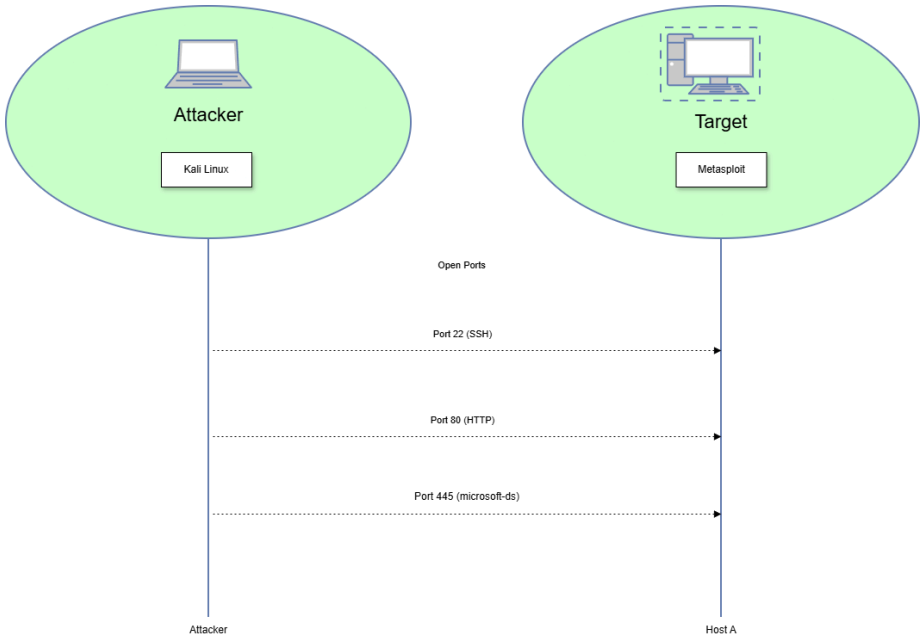
```
┌──(kali㉿kali)-[~]
└─$ nmap -sU 192.168.56.7 -oN udp_scan.txt
Starting Nmap 7.95 ( https://nmap.org ) at 2025-12-15 13:14 EST
Nmap scan report for 192.168.56.7
Host is up (0.0017s latency).
Not shown: 992 closed udp ports (port-unreach)
PORT      STATE         SERVICE
53/udp    open          domain
68/udp    open|filtered dhcpc
69/udp    open|filtered tftp
111/udp   open          rpcbind
137/udp   open          netbios-ns
138/udp   open|filtered netbios-dgm
2049/udp  open          nfs
49211/udp open          unknown
MAC Address: 00:0C:29:FA:DD:2A (VMware)

Nmap done: 1 IP address (1 host up) scanned in 1094.25 seconds
```

Fig.3

Raw outputs for each command were saved as udp_scan.txt

9. Network Diagram



10. Open Ports

SYN and TCP scan open ports

Port	Proto	State	Service
21	tcp	open	ftp
22	tcp	open	ssh
23	tcp	open	telnet
25	tcp	open	smtp
53	tcp	open	domain
80	tcp	open	http
111	tcp	open	rpcbind
139	tcp	open	netbios-ssn
445	tcp	open	microsoft-ds
512	tcp	open	exec
513	tcp	open	login
514	tcp	open	shell
1099	tcp	open	rmiregistry
2049	tcp	open	nfs
3306	tcp	open	mysql
5432	tcp	open	postgresql
5900	tcp	open	vnc
6000	tcp	open	X11
6667	tcp	open	irc
8009	tcp	open	ajp13

UDP scan open ports

Port	Proto	State	Service
53	udp	open	domain
68	udp	open|filtered	dhcpc
69	udp	open|filtered	tftp
111	udp	open	rpcbind
137	udp	open	netbios-ns
138	udp	open|filtered	netbios-dgm
2049	udp	open	nfs
2111	udp	open	unknown

11. Service Research

Two services from the scan were selected for deeper research to understand their purpose and common vulnerabilities: SSH (port 22) and HTTP (port 80).

SSH (Port 22)

Secure Shell (SSH) is used for encrypted remote login, command execution, and file transfer between clients and servers, replacing legacy protocols like Telnet and rlogin that send data in clear text. While SSH improves confidentiality and integrity, exposed SSH services are often targeted with credential brute-forcing, password spraying, and attempts to reuse stolen SSH keys. Misconfigurations such as allowing root login, using outdated protocol versions, or failing to restrict access by IP can significantly increase risk, especially when combined with unpatched vulnerabilities in specific SSH implementations. Recommended hardening includes disabling direct root login, enforcing key-based authentication, using strong passphrases, and limiting access via firewalls or TCP wrappers.

HTTP (Port 80)

HTTP is the core protocol for delivering web pages and APIs between clients and servers, commonly exposed on port 80 for unencrypted traffic or 443 for HTTPS. A web server may host static content, dynamic applications, or administrative interfaces, each of which can introduce different types of vulnerabilities.
Common security issues associated with HTTP-based services include cross-site scripting, SQL injection, directory traversal, insecure direct object references, and insufficient session management. Mitigation strategies involve secure coding practices, robust input validation, least-privilege database access, use of security headers, and keeping the web server and application stack regularly patched.

12. Key Learnings

Through this task, it became clear how IP addresses, ports, and protocols interact to form the basis of network communication, and how open ports map directly to running services and potential attack surfaces. Working with Nmap provided hands-on experience with different scan types such as SYN, TCP connect, and UDP scans, highlighting trade-offs between speed, stealth, and reliability.

Analyzing the open services emphasized the security impact of legacy protocols like FTP and Telnet compared with more secure alternatives such as SSH, as well as the importance of minimizing exposed services and hardening those that must remain reachable. Finally, integrating Nmap with Python demonstrated how automation can streamline reconnaissance and reporting, enabling more efficient and repeatable security assessments in a professional penetration testing or VAPT workflow

13. Conclusion

This task demonstrated a complete mini-VAPT workflow against an intentionally vulnerable Metasploitable VM from a Kali Linux attacker machine running in VirtualBox. Using Nmap, you performed SYN, TCP connect, and UDP scans to systematically enumerate open ports and identified a wide range of exposed services, including legacy protocols (FTP, Telnet, r-services), web services, databases, RPC, SMB, NFS, and remote access interfaces. These findings highlighted how a poorly hardened host can present a broad attack surface across multiple layers of the stack.

14. References

- Nmap Documentation – https://nmap.org
- Python-nmap library - https://pypi.org/project/python-nmap
- Github - https://github.com/topics/port-scanner-in-python

15. Python Automation Script

Purpose and Library

To automate the scanning process, a Python script named nmap_automation.py was developed using the python-nmap library, which provides a wrapper around the Nmap binary and allows direct access to scan results from Python code. The script accepts a target IP or hostname, performs a SYN scan, extracts information about open ports and services, and writes a human-readable text report.

The python-nmap library was installed with:

- pip install python-nmap

or an equivalent package such as python3-nmap depending on the environment.

This implementation demonstrates how Nmap's functionality can be automated, making it easier to repeat scans, store results, and integrate port scanning into larger workflows or tools. By wrapping Nmap in a Python script, you remove the need to retype complex command-line options for each assessment and instead trigger consistent scans with a single command or function call. The script can automatically capture structured results (host IPs, open ports, services, and versions) and write them to files, which simplifies documentation, comparison between runs, and integration with other security tooling such as vulnerability scanners, reporting pipelines, or custom dashboards.

In addition, programmatic access to Nmap results allows you to post-process the data, for example by filtering only high-risk ports, tagging legacy or insecure services, or feeding the findings into other components such as vulnerability scanners, ticketing systems, or dashboards. This type of automation is especially useful in larger penetration tests or continuous assessment setups, where recurring scans need to be scheduled, logged, and correlated without manual intervention

Script Logic and Structure

The script's high-level workflow is:

1. Prompt the user to enter a target IP or hostname.
2. Create a PortScanner object and run an Nmap SYN scan using the appropriate arguments.
3. Iterate over discovered hosts, protocols, and ports, selecting only those with state "open".
4. Extract service names and, where available, product and version strings from the scan results.
5. Generate a scan_report.txt file containing a timestamp, target, and a table of open ports and services, followed by a simple completion note.

Python Code

```
#!/usr/bin/env python3
"""
nmap_automation.py
Automates an Nmap SYN scan and writes results to scan_report.txt
"""

import nmap         # python-nmap library wrapper around Nmap
import datetime     # for timestamps


def run_syn_scan(target):
    """
    Run an Nmap SYN (-sS) scan against the given target.
    Returns a PortScanner object containing the scan results.
    """
    nm = nmap.PortScanner()
    # -sS = SYN scan, -Pn = skip host discovery (assume host is up), -T4 = faster timing
    nm.scan(target, arguments='-sS -Pn -T4')
    return nm


def generate_report(target, nm, filename="scan_report.txt"):
    """
    Generate a text report from an Nmap PortScanner result.
    The report includes timestamp, target IP, and a table of open ports/services.
    """
    with open(filename, "w") as f:
        # Header information
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        f.write("==== Nmap Automated Scan Report ====\n")
        f.write(f"Scan timestamp : {timestamp}\n")
        f.write(f"Target         : {target}\n\n")

        # Iterate over all discovered hosts
 for host in nm.all_hosts():
 f.write(f"Host IP : {host}\n")
 f.write(f"Host state : {nm[host].state()}\n\n")

 # Table header
 f.write("Open Ports and Services:\n")
 f.write("{:<8} {:<8} {:<20} {:<30}\n".format(
 "Port", "Proto", "Service", "Version"))
 f.write("-" * 70 + "\n")
```

```
        # For each protocol (usually 'tcp' and/or 'udp')
        for proto in nm[host].all_protocols():
            ports = nm[host][proto].keys()
            for port in sorted(ports):
                port_data = nm[host][proto][port]
                if port_data.get("state") == "open":
                    # Extract service name and version info if present
                    name = port_data.get("name", "")
                    product = port_data.get("product", "")
                    version = port_data.get("version", "")
                    extrainfo = port_data.get("extrainfo", "")

                    # Build a version string if we have any details
                    version_str = " ".join(
                        x for x in [product, version, extrainfo] if x
                    )
                    f.write("{:<8} {:<8} {:<20} {:<30}\n".format(
                        port, proto, name, version_str))

        f.write("\n")

    # Completion note
    f.write("Scan completed successfully.\n")
    f.write("=====================================\n")

def main():
 target = input("Enter target IP or hostname: ").strip()
 if not target:
 print("No target provided. Exiting.")
 return
 print(f"[+] Running SYN scan on {target} ...")
 nm = run_syn_scan(target)
 print("[+] Scan finished, generating report ...")
 generate_report(target, nm)
 print("[+] Report saved to scan_report.txt")


if __name__ == "__main__":
 main()
```

Script Result

```
┌──(venv)─(kali㉿kali)-[~/Desktop/Test]
└─$ python3 nmap_automation.py
Enter target IP or hostname: 192.168.56.7
[+] Running SYN scan on 192.168.56.7 ...
[+] Scan finished, generating report ...
[+] Report saved to scan_report.txt

┌──(venv)─(kali㉿kali)-[~/Desktop/Test]
└─$ cat scan_report.txt
=== Nmap Automated Scan Report ===
Scan timestamp : 2025-12-15 14:34:44
Target         : 192.168.56.7

Host IP        : 192.168.56.7
Host state     : up

Open Ports and Services:
Port     Proto    Service              Version
------------------------------------------------------------
21       tcp      ftp
22       tcp      ssh
23       tcp      telnet
25       tcp      smtp
53       tcp      domain
80       tcp      http
111      tcp      rpcbind
139      tcp      netbios-ssn
445      tcp      microsoft-ds
512      tcp      exec
513      tcp      login
514      tcp      shell
1099     tcp      rmiregistry
1524     tcp      ingreslock
2049     tcp      nfs
2121     tcp      ccproxy-ftp
3306     tcp      mysql
5432     tcp      postgresql
5900     tcp      vnc
6000     tcp      X11
6667     tcp      irc
8009     tcp      ajp13
8180     tcp

Scan completed successfully.
============================================================
```

Fig.4