

Comprehensive Report on Network Penetration Testing / Network INFORMATION Gathering Tool

Developed by Team Gamma



Contents

[**Introduction** 2](#_Toc171002077)

[**Functional Overview** 2](#_Toc171002078)

[**Analysis** 2](#_Toc171002079)

[**1. Domain Resolution and MX Record Retrieval** 2](#_Toc171002080)

[**Functionality:** 2](#_Toc171002081)

[**Code Implementation:** 3](#_Toc171002082)

[**Explanation:** 3](#_Toc171002083)

[**Usage:** 3](#_Toc171002084)

[**Example Scenario:** 3](#_Toc171002085)

[**2. Port Scanning** 3](#_Toc171002086)

[**Functionality:** 3](#_Toc171002087)

[**Code Implementation:** 4](#_Toc171002088)

[**Explanation:** 4](#_Toc171002089)

[**Usage:** 4](#_Toc171002090)

[Example Scenario: 4](#_Toc171002091)

[**3. OS Detection** 5](#_Toc171002092)

[**Functionality:** 5](#_Toc171002093)

[**Code Implementation:** 5](#_Toc171002094)

[**Explanation:** 5](#_Toc171002095)

[**Usage:** 6](#_Toc171002096)

[**Example Scenario:** 6](#_Toc171002097)

[**4. Service Detection** 6](#_Toc171002098)

[**Functionality:** 6](#_Toc171002099)

[**Code Implementation:** 7](#_Toc171002100)

[**Explanation:** 7](#_Toc171002101)

[**Usage:** 7](#_Toc171002102)

[**Example Scenario:** 7](#_Toc171002103)

[**5. Database Detection** 8](#_Toc171002104)

[**Functionality:** 8](#_Toc171002105)

[**Code Implementation:** 8](#_Toc171002106)

[**Explanation:** 8](#_Toc171002107)

[**Usage:** 9](#_Toc171002108)

[**Example Scenario:** 9](#_Toc171002109)

[**6. Network Mapping** 9](#_Toc171002110)

[**Functionality:** 9](#_Toc171002111)

[**Code Implementation:** 10](#_Toc171002112)

[**Explanation:** 10](#_Toc171002113)

[**Usage:** 10](#_Toc171002114)

[**Example Scenario:** 10](#_Toc171002115)

[**Conclusion** 10](#_Toc171002116)

[**Future Enhancements:** 11](#_Toc171002117)

[ **Automated Reporting:** 11](#_Toc171002118)

[ **Integration with Vulnerability Databases:** 11](#_Toc171002119)

[ **Improved User Interface:** 11](#_Toc171002120)

[**References** 11](#_Toc171002121)

[**Appendix:** 11](#_Toc171002122)

# **Introduction**

This report provides an in-depth analysis of a network penetration testing and gathering tool developed by Team Gamma. The tool is designed to perform a variety of network reconnaissance tasks, including domain resolution, MX record retrieval, port scanning, OS detection, service detection, database detection, and network mapping. The tool leverages the nmap library for most of its functionalities and the dns.resolver module for DNS-related tasks. This comprehensive report details the functionality, code implementation, and usage of each component of the tool, providing insights into its operation and practical applications.

## **Functional Overview**

The tool is structured to perform the following tasks:

1. **Domain Resolution and MX Record Retrieval**
2. **Port Scanning**
3. **OS Detection**
4. **Service Detection**
5. **Database Detection**
6. **Network Mapping**

Each function is user-driven, where inputs are taken through prompts, and the operations are carried out based on user preferences.

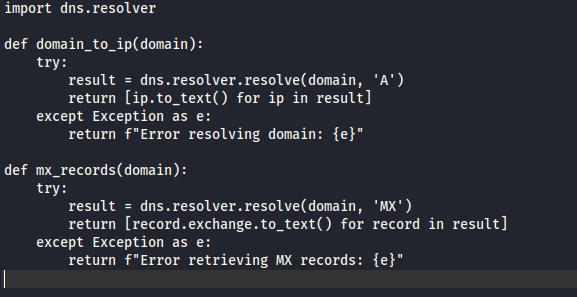
# **Analysis**

## **1. Domain Resolution and MX Record Retrieval**

### **Functionality:**

* **Domain Resolution:** Converts a domain name to its associated IP addresses.
* **MX Record Retrieval:** Fetches Mail Exchange (MX) records of a domain.

## **Code Implementation:**



## **Explanation:**

* The domain\_to\_ip function resolves the given domain to its corresponding IP addresses using DNS queries. This is essential for mapping a domain name to its IP addresses, which is a fundamental step in network reconnaissance.
* The mx\_records function retrieves MX records, which indicate the mail servers responsible for receiving emails on behalf of the domain. Understanding the email infrastructure of a target can be crucial for penetration testing, as it can reveal potential vulnerabilities in the mail servers.

## **Usage:**

* These functions are initiated by user input, where the domain is provided, and the respective IP addresses and MX records are displayed. This step helps in gathering preliminary information about the target domain, setting the stage for more detailed scanning and analysis.

## **Example Scenario:**

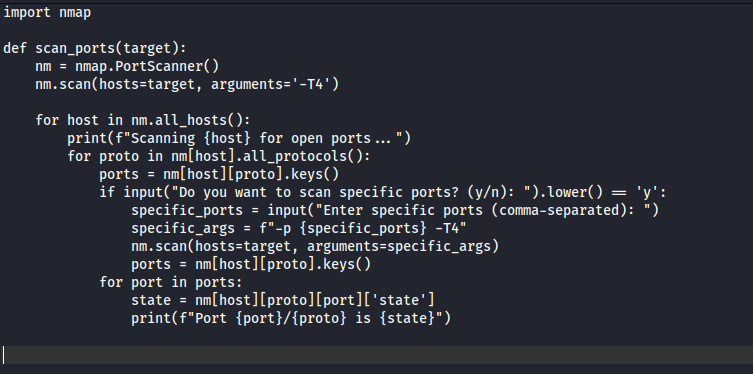
* A penetration tester starts by resolving the domain example.com to its IP addresses. This reveals multiple IPs associated with the domain, which could be load-balanced servers or different services hosted on subdomains. Next, the tester retrieves the MX records to identify the mail servers, checking for misconfigurations or outdated software that could be exploited.

# **2. Port Scanning**

## **Functionality:**

* Scans for open ports on a specified target (domain or IP address).

## **Code Implementation:**



## **Explanation:**

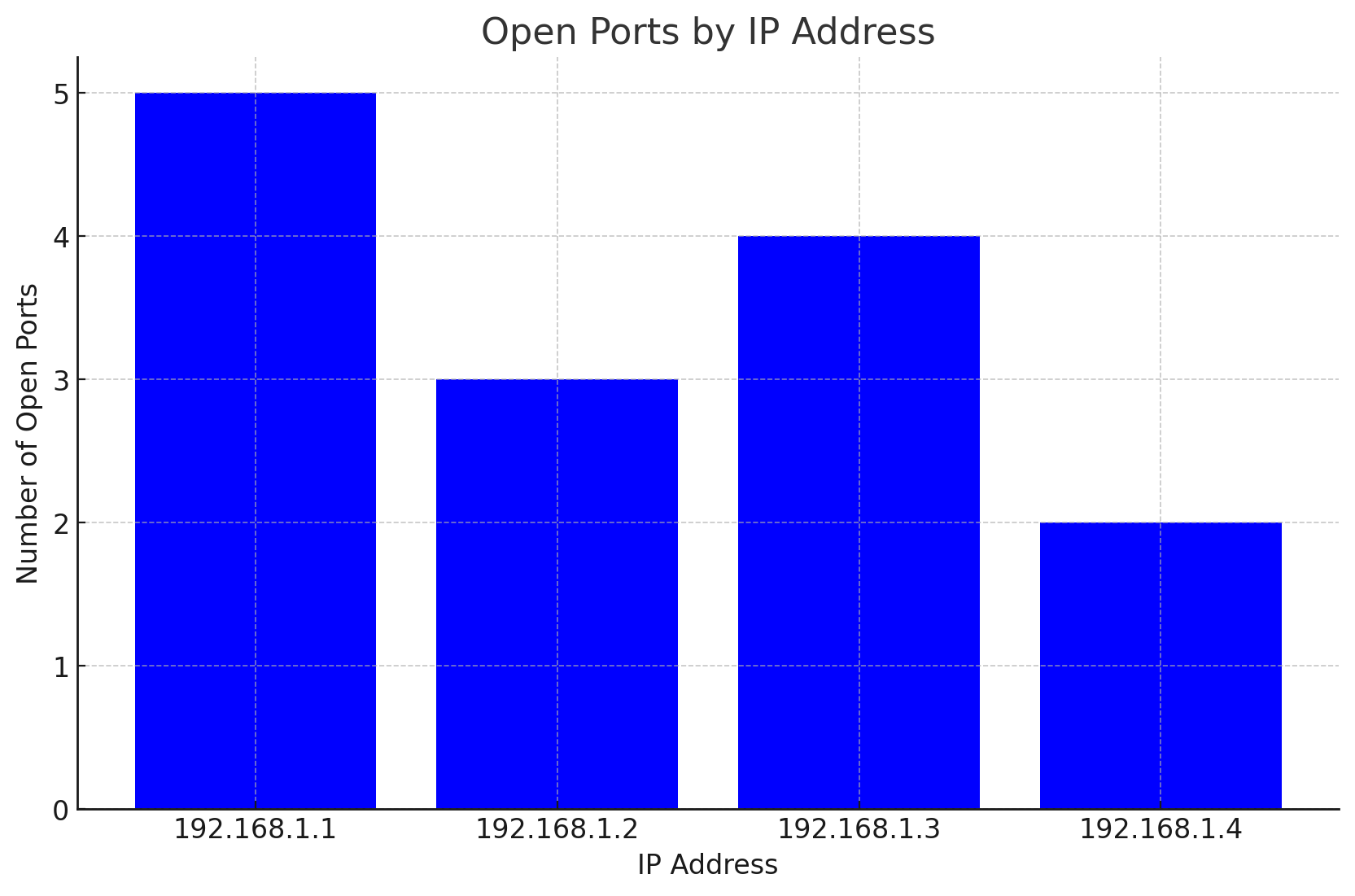
* The scan\_ports function utilizes nmap to scan for open ports on the specified target. Open ports are potential entry points into a system, and identifying them is a critical step in penetration testing.
* It allows the user to specify particular ports if desired, enhancing the granularity of the scan. This flexibility enables testers to focus on specific services or ranges of ports that are known to be problematic or require closer inspection.

## **Usage:**

* The user inputs a target domain or IP address. The tool scans the target and displays the status of each port. This information helps in identifying exposed services that might be vulnerable to attacks.

## Example Scenario:

* A penetration tester scans the target 192.168.1.1 and discovers several open ports, including 22 (SSH), 80 (HTTP), and 443 (HTTPS). The tester then decides to focus on port 22, checking for weak SSH credentials or outdated software that could be exploited.

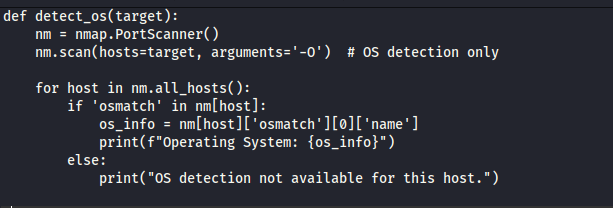


# **3. OS Detection**

## **Functionality:**

* Detects the operating system running on the target.

## **Code Implementation:**



## **Explanation:**

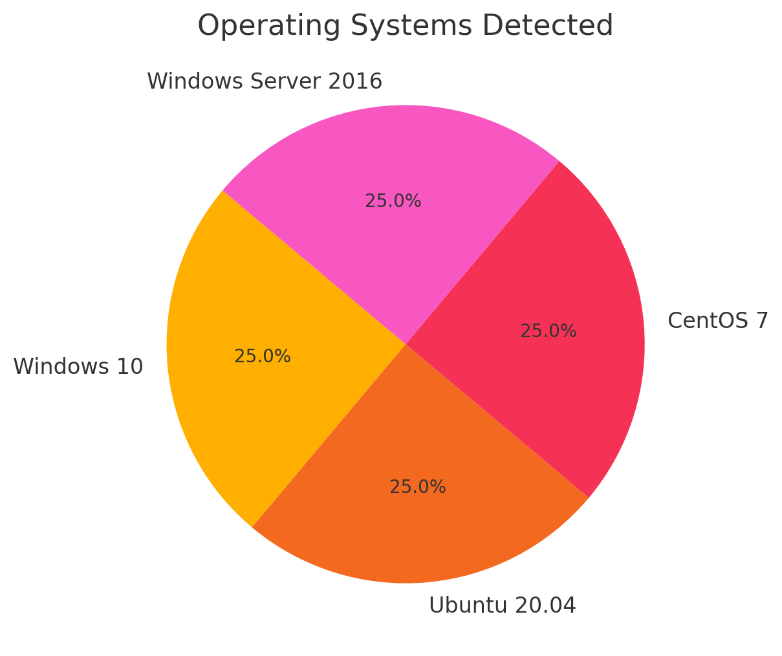
* The detect\_os function performs an OS detection scan using nmap. Identifying the operating system is crucial as it provides context about the target environment and potential vulnerabilities associated with specific OS versions.
* It retrieves and displays the name of the detected operating system for the target. This information can be used to tailor subsequent attacks or scans to the specific OS, increasing the likelihood of successful exploitation.

## **Usage:**

* This feature is optional and can be triggered by the user's choice during the tool's execution. OS detection is particularly useful in scenarios where understanding the target's environment is necessary for effective penetration testing.

## **Example Scenario:**

* After scanning the target 192.168.1.1, the penetration tester opts to detect the OS. The tool identifies the target as running Windows Server 2016. The tester then searches for known vulnerabilities and exploits specific to this OS version, planning the next steps accordingly.

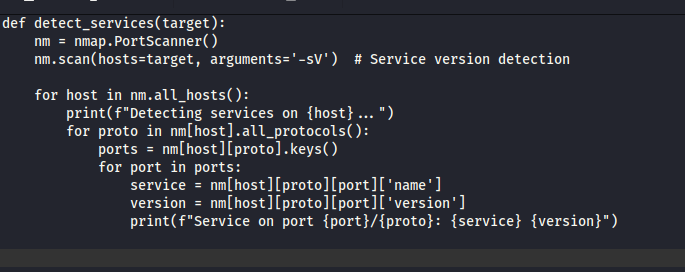


# **4. Service Detection**

## **Functionality:**

* Detects services running on open ports.

## **Code Implementation:**



## **Explanation:**

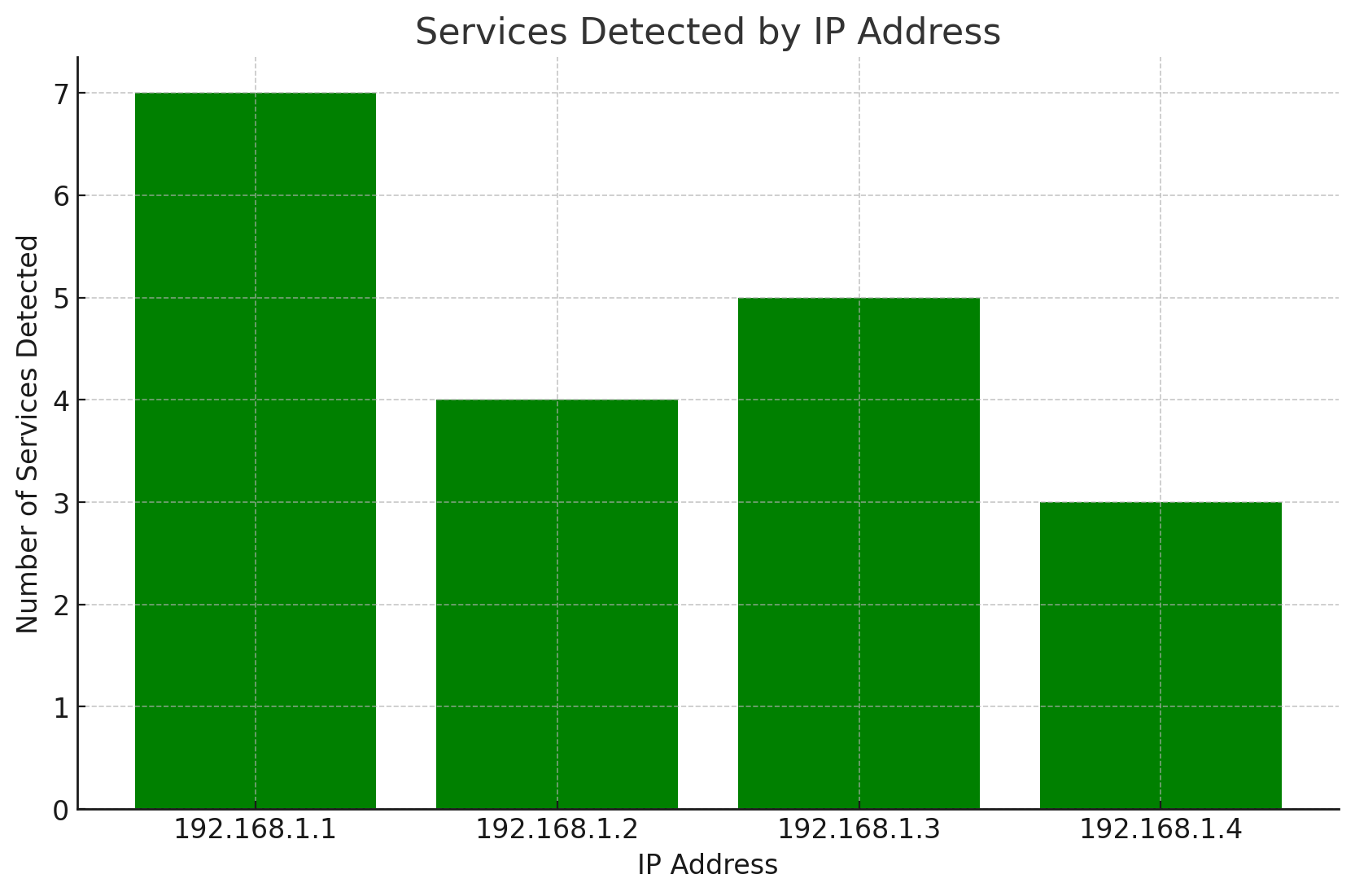
* The detect\_services function scans the target to detect running services and their versions. Knowing the specific services and their versions helps in identifying potential vulnerabilities associated with those services.
* This is achieved using nmap's service version detection capability. Accurate service detection enables penetration testers to focus their efforts on services with known security issues.

## **Usage:**

* The user can opt to use this feature to gain insights into the services running on the target. This step is essential for building a comprehensive understanding of the target's network and identifying potential weak points.

## **Example Scenario:**

* A penetration tester detects services on 192.168.1.1 and finds that the server is running Apache HTTP Server 2.4. The tester then researches vulnerabilities specific to this version, such as known exploits or misconfigurations, and plans the next phase of the attack.

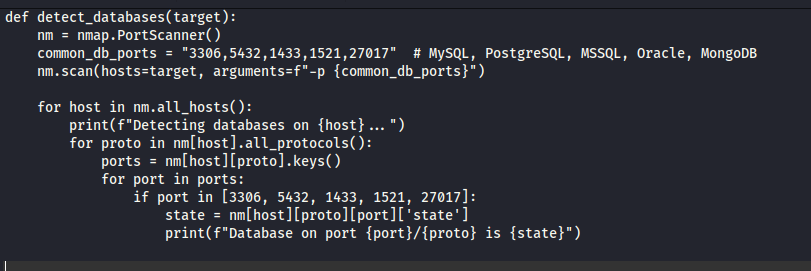


# **5. Database Detection**

## **Functionality:**

* Detects common database services running on the target.

## **Code Implementation:**



## **Explanation:**

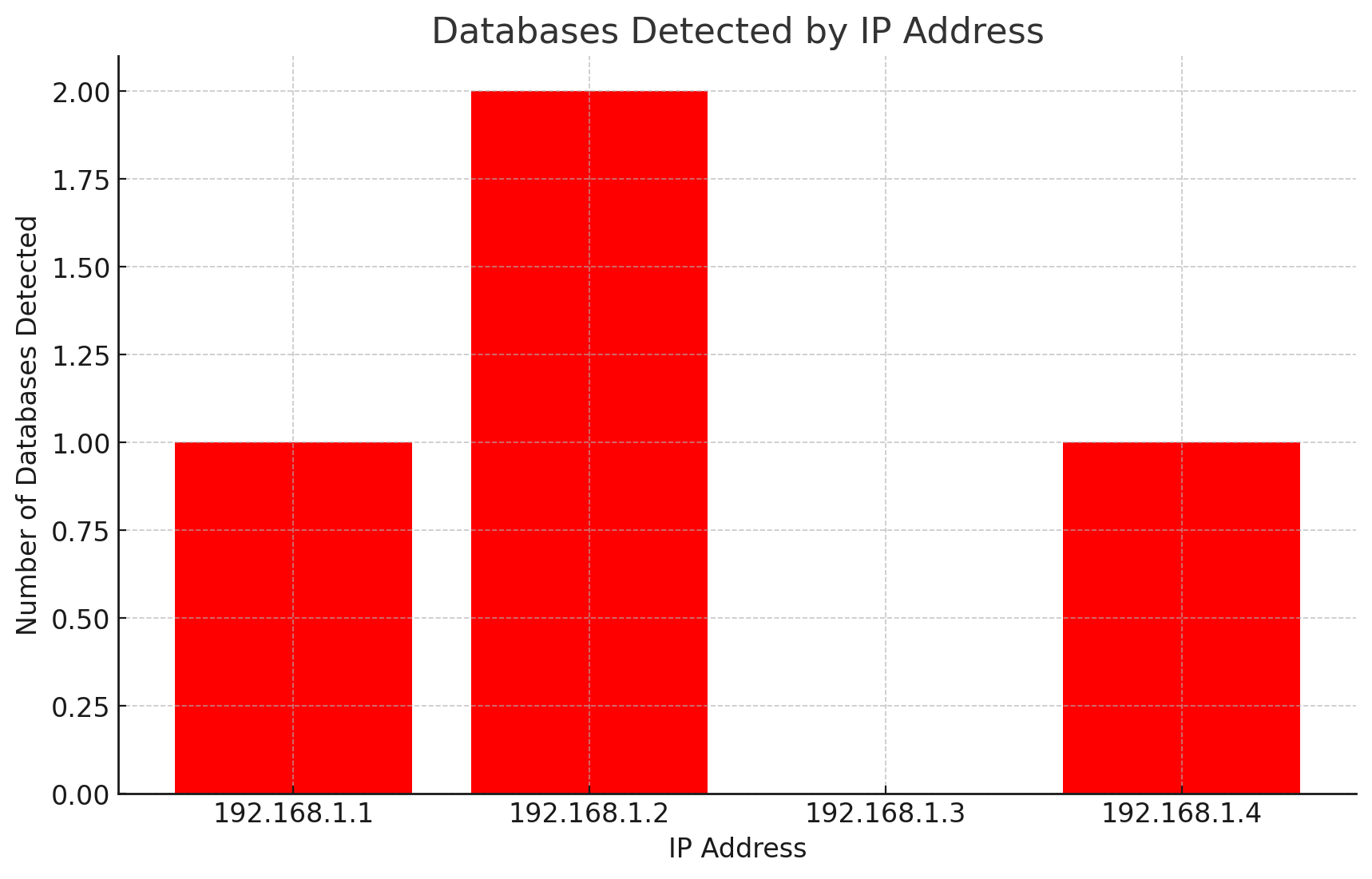
* The detect\_databases function focuses on identifying common database services by scanning known database ports. Databases are critical components of many systems, and detecting them can reveal significant vulnerabilities.
* It checks the state of these ports to determine if database services are running. Identifying database services helps penetration testers understand the target's data storage infrastructure and potential entry points for data breaches.

## **Usage:**

* Users can choose this option to check for database services, aiding in database vulnerability assessments. Detecting databases is crucial for identifying potential risks to sensitive data.

## **Example Scenario:**

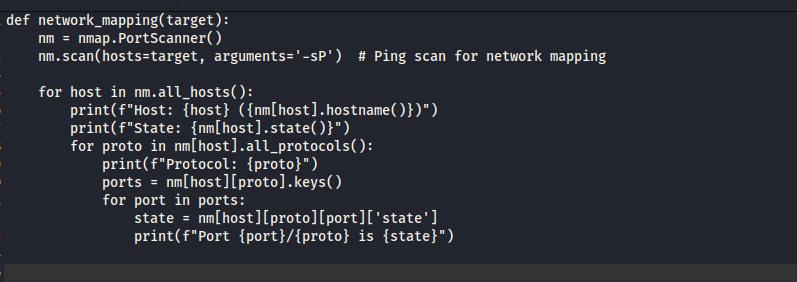
* A penetration tester scans the target 192.168.1.1 and detects a running MySQL service on port 3306. The tester then attempts to access the database using default credentials or exploits known vulnerabilities in the MySQL service to gain unauthorized access to the data.



# **6. Network Mapping**

## **Functionality:**

* Maps the network to discover hosts within a specified range.

**Code Implementation:**

## **Explanation:**

* The network\_mapping function performs a ping scan to map out the network and discover active hosts. This function helps in identifying all devices within a network segment, providing a comprehensive view of the network topology.
* It provides details about each host, including state and active protocols. Understanding the network layout is critical for effective penetration testing, as it reveals potential targets and network structure.

## **Usage:**

* This function is useful for network administrators to get an overview of the network and identify active devices. Network mapping is the first step in understanding the scope of a network and planning subsequent penetration tests.

## **Example Scenario:**

* A penetration tester maps the network 192.168.1.0/24 and identifies multiple active hosts, including workstations, servers, and network devices. The tester then prioritizes targets based on their roles and potential vulnerabilities, planning detailed scans and attacks for each device.

# **Conclusion**

The network penetration testing and gathering tool developed by Team Gamma is a comprehensive solution for network reconnaissance and security assessments. It combines domain resolution, port scanning, OS and service detection, database identification, and network mapping into a single, user-friendly script. The tool's modular design allows users to selectively perform tasks based on their requirements, making it versatile for different network security scenarios. By leveraging powerful libraries such as nmap and dns.resolver, the tool ensures reliable and accurate results.

## **Future Enhancements:**

* **Automated Reporting:** Implementing automated report generation after scans to provide detailed and structured reports for analysis and documentation.
* **Integration with Vulnerability Databases:** Linking detected services and OS information with known vulnerabilities for proactive security measures, enabling testers to identify and address vulnerabilities more efficiently.
* **Improved User Interface:** Developing a graphical user interface (GUI) for better usability, making the tool accessible to users with varying levels of technical expertise.

This tool is a valuable asset for network security professionals, enabling them to conduct thorough and effective network penetration testing and gathering with ease. The detailed insights provided by the tool help in identifying and mitigating security risks, ensuring the robustness and security of network infrastructures.

# **References**

1. Nmap, 2024. Nmap Reference Guide
2. Radding, A., 2024. DNS Resolver Library Documentation. dnspython.readthedocs.io. Available at: <https://dnspython.readthedocs.io/en/latest/> [Accessed 4 July 2024].
3. OWASP, 2024. Penetration Testing Methodologies. OWASP.

# **Appendix:**

import nmap

import dns.resolver

def domain\_to\_ip(domain):

    try:

        result = dns.resolver.resolve(domain, 'A')

        return [ip.to\_text() for ip in result]

    except Exception as e:

        return f"Error resolving domain: {e}"

def mx\_records(domain):

    try:

        result = dns.resolver.resolve(domain, 'MX')

        return [record.exchange.to\_text() for record in result]

    except Exception as e:

        return f"Error retrieving MX records: {e}"

def scan\_ports(target):

    nm = nmap.PortScanner()

    nm.scan(hosts=target, arguments='-T4')

    for host in nm.all\_hosts():

        print(f"Scanning {host} for open ports...")

        for proto in nm[host].all\_protocols():

            ports = nm[host][proto].keys()

            if input("Do you want to scan specific ports? (y/n): ").lower() == 'y':

                specific\_ports = input("Enter specific ports (comma-separated): ")

                specific\_args = f"-p {specific\_ports} -T4"

                nm.scan(hosts=target, arguments=specific\_args)

                ports = nm[host][proto].keys()

            for port in ports:

                state = nm[host][proto][port]['state']

                print(f"Port {port}/{proto} is {state}")

def detect\_os(target):

    nm = nmap.PortScanner()

    nm.scan(hosts=target, arguments='-O')  # OS detection only

    for host in nm.all\_hosts():

        if 'osmatch' in nm[host]:

            os\_info = nm[host]['osmatch'][0]['name']

            print(f"Operating System: {os\_info}")

        else:

            print("OS detection not available for this host.")

def detect\_services(target):

    nm = nmap.PortScanner()

    nm.scan(hosts=target, arguments='-sV')  # Service version detection

    for host in nm.all\_hosts():

        print(f"Detecting services on {host}...")

        for proto in nm[host].all\_protocols():

            ports = nm[host][proto].keys()

            for port in ports:

                service = nm[host][proto][port]['name']

                version = nm[host][proto][port]['version']

                print(f"Service on port {port}/{proto}: {service} {version}")

def detect\_databases(target):

    nm = nmap.PortScanner()

    common\_db\_ports = "3306,5432,1433,1521,27017"  # MySQL, PostgreSQL, MSSQL, Oracle, MongoDB

    nm.scan(hosts=target, arguments=f"-p {common\_db\_ports}")

    for host in nm.all\_hosts():

        print(f"Detecting databases on {host}...")

        for proto in nm[host].all\_protocols():

            ports = nm[host][proto].keys()

            for port in ports:

                if port in [3306, 5432, 1433, 1521, 27017]:

                    state = nm[host][proto][port]['state']

                    print(f"Database on port {port}/{proto} is {state}")

def network\_mapping(target):

    nm = nmap.PortScanner()

    nm.scan(hosts=target, arguments='-sP')  # Ping scan for network mapping

    for host in nm.all\_hosts():

        print(f"Host: {host} ({nm[host].hostname()})")

        print(f"State: {nm[host].state()}")

        for proto in nm[host].all\_protocols():

            print(f"Protocol: {proto}")

            ports = nm[host][proto].keys()

            for port in ports:

                state = nm[host][proto][port]['state']

                print(f"Port {port}/{proto} is {state}")

if \_\_name\_\_ == "\_\_main\_\_":

    domain = input("Enter a domain to resolve: ")

    ips = domain\_to\_ip(domain)

    print(f"IP addresses for {domain}: {ips}")

    mx = mx\_records(domain)

    print(f"MX records for {domain}: {mx}")

    target = input("Enter a domain or IP address to scan: ")

    scan\_ports(target)

    if input("Do you want to see OS detection results? (y/n): ").lower() == 'y':

        detect\_os(target)

    if input("Do you want to detect services? (y/n): ").lower() == 'y':

        detect\_services(target)

    if input("Do you want to detect databases? (y/n): ").lower() == 'y':

        detect\_databases(target)

    if input("Do you want to map the network? (y/n): ").lower() == 'y':

        network\_mapping(target)