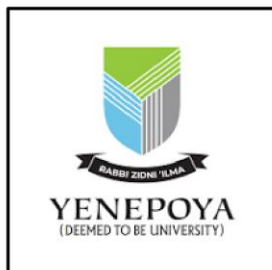


**YENEPOYA INSTITUTE OF ARTS, SCIENCE, COMMERCE AND MANAGEMENT A
CONSTITUENT UNIT OF YENEPOYA (DEEMED TO BE UNIVERSITY) BALMATTA,
MANGALORE**



Automated Penetration Testing Tool

FINAL REPORT

on Automated Penetration Testing Tool

BACHELOR OF SCIENCE

NAME	REGISTER NUMBER	EMAIL ID
ABHINAV K S	22BSCFDC02	22064@yenepoya.edu.in
GIBIN G	22BSCFDC17	21650@yenepoya.edu.in
ABHIJITH A NAIR	22BSCFDC01	21633@yenepoya.edu.in
MANURAJ MENON	22BSCFDC22	22062@yenepoya.edu.in
RAHUL KURUVATH	22BSCFDC35	23058@yenepoya.edu.in

Guided by,
Mr.Shashank

Executive Summary

InfoGather Pro is a comprehensive, full-stack cybersecurity reconnaissance tool developed to support ethical hackers and security analysts in performing automated intelligence gathering. The tool focuses on collecting publicly available information about internet-facing domains to streamline the reconnaissance phase of security assessments. Its primary goal is to reduce the manual effort involved in gathering preliminary data about a web target by automating key reconnaissance tasks.

InfoGather Pro performs various operations, including gathering DNS records such as A, AAAA, MX, NS, TXT, CNAME, SOA, and PTR, also parsing TXT records for SPF and DMARC configurations, and performing reverse DNS lookups. It extracts WHOIS information, including domain registration metadata, registrar, creation and expiration dates, name servers, contact emails, and country of registration. For subdomain enumeration, it utilizes Certificate Transparency Logs (via crt.sh) to discover associated SSL/TLS certificates, which helps reveal hidden infrastructure. The tool also performs SMTP diagnostics by identifying mail servers using MX records and attempting SMTP connections to assess server response, availability, and basic functionality like banner capture and HELO verification. Additionally, it checks IP addresses against DNS-based blacklists (DNSBLs) to determine if they are flagged for spam, abuse, or malware hosting, which is crucial for evaluating a domain's reputation.

The backend of InfoGather Pro is powered by the Python Flask framework (app.py), which manages scanning logic, handles API routes, and executes multi-threaded scans for non-blocking performance. It uses specialized libraries such as dnspython, python-whois, smtplib, socket, and requests to interact with external servers and services. The frontend is built with HTML, CSS (using Bootstrap 5), and JavaScript, offering a responsive, dark-themed interface that supports domain input validation, loading animations, and accordion-style result sections. This intuitive interface allows users to quickly visualize scan results categorized under metadata, DNS, WHOIS, subdomains, SMTP diagnostics, and blacklist checks.

In summary, InfoGather Pro addresses a fundamental need in the cybersecurity field: reliable, efficient, and automated reconnaissance. Its extensible architecture and real-time performance make it an ideal tool for ethical hackers, penetration testers, security analysts, and students in cybersecurity programs. Through automation and thoughtful interface design, it bridges the gap between manual intelligence gathering and scalable, modern threat assessment workflows.

Table of Contents

	Heading
	Executive Summary
1.	Background
1.1	Aim
1.2	Technologies
1.3	Hardware Architecture
1.4	Software Architecture
2	System
2.1	Functional requirements
2.1.2	User requirements
2.1.3	Environmental requirements
2.2	Design and Architecture
2.3	Implementation
2.4	Testing
2.4.1	Test Plan Objectives
2.4.2	Security
2.4.3	System Test
2.5	Graphical User Interface (GUI) Layout
2.6	Evaluation
3	Snapshots of the Project
4	Tools used
5	Conclusions
6	Further development or research
7	References

1. Background

As IT infrastructures continue to grow in complexity, encompassing cloud services, virtual environments, mobile platforms, and IoT devices, ensuring comprehensive cybersecurity coverage has become increasingly challenging. Traditional manual methods of conducting vulnerability assessments are often inadequate in addressing the dynamic nature of modern systems. These methods are not only labor-intensive and time-consuming but also susceptible to human error, which can result in the oversight of critical vulnerabilities. As cyber threats become more sophisticated and pervasive, organizations require more efficient and reliable ways to identify and address security weaknesses.

In response to these challenges, the adoption of automation in vulnerability scanning and information gathering has gained significant traction. Automated tools enable continuous and systematic assessments across expansive digital environments, providing rapid detection of vulnerabilities with improved accuracy. By streamlining routine tasks, these tools allow cybersecurity professionals to focus on higher-level analysis and strategic decision-making. The integration of automation into vulnerability management processes not only enhances operational efficiency but also delivers timely and actionable insights that are essential for mitigating potential security risks and ensuring regulatory compliance.

1.1 Aim

The development of InfoGather Pro is guided by key objectives focused on improving the efficiency, accuracy, and usability of the reconnaissance phase in cybersecurity assessments. These objectives include:

1. **Tool Development:** To create a comprehensive and automated cybersecurity tool.
2. **Vulnerability Detection:** To streamline the process of detecting vulnerabilities in systems and networks.
3. **Information Gathering:** To automate the collection of essential information about target systems.
4. **Efficiency Improvement:** To reduce manual effort and minimize human error in vulnerability assessments.
5. **Time and Accuracy:** To save time and improve the accuracy of threat detection.
6. **Proactive Threat Response:** To help cybersecurity professionals identify and fix issues early.
7. **Defense Enhancement:** To support proactive defense strategies and prevent cyberattacks.
8. **Security Strengthening:** To enhance the overall security posture of organizations.

1.2 Technologies

- **Programming Languages:**

The project leverages a combination of programming languages, each serving a specific purpose within the tool's architecture:

- **Python** is the core language driving the backend operations. It is responsible for executing the automated vulnerability scans, running various integrated security tools, and managing the overall workflow. Python's extensive libraries and ease of scripting make it ideal for orchestrating complex scanning and data collection processes.
- **JavaScript** enhances the user experience on the client side by providing dynamic and interactive elements within the web interface. This includes real-time updates, form validations, and responsive UI components that improve usability.
- **HTML** (HyperText Markup Language) and **CSS** (Cascading Style Sheets) are used to build the structural layout and visual styling of the web pages. HTML defines the content and organization of the interface, while CSS ensures the design is clean, consistent, and adaptable across different devices and screen sizes.

- **Frameworks:** Flask is a lightweight and flexible web framework written in Python. It acts as the bridge between the backend scanning logic and the frontend user interface. Flask handles incoming web requests, routes them to the appropriate Python functions that perform scanning or data processing and then renders the results back to the user in an accessible web format. Its modularity and simplicity allow for rapid development and easy integration of new features.
- **Security Tools and Integration:** The project incorporates a range of well-known security scanning tools, which are integrated and controlled through Python scripts. These tools perform critical functions such as port scanning to identify open services, vulnerability scanning to detect potential security weaknesses, and reconnaissance to gather detailed information about the target systems. By automating the execution and result parsing of these tools, the project enables efficient and comprehensive security assessments. Python's scripting capabilities ensure smooth coordination between different scanners, unify their outputs, and present actionable insights to users.

1.3 Hardware Architecture

The automated vulnerability detection and information gathering tool is designed to operate efficiently on commonly available computing hardware, making it accessible for a wide range of users without the need for specialized or high-end equipment. To ensure smooth and reliable performance, the tool requires the following minimum hardware specifications:

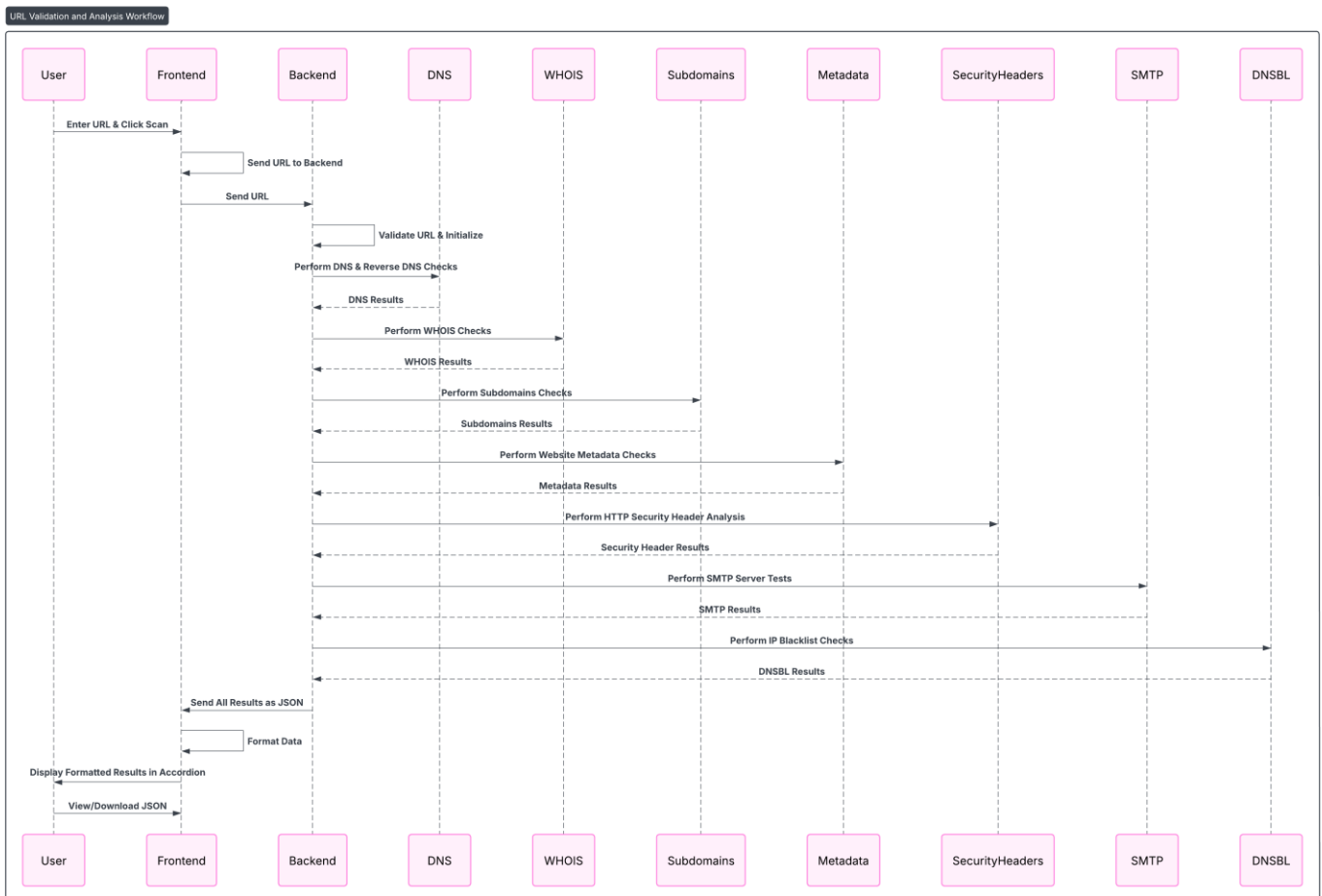
- **Processor:** A dual-core CPU is recommended as the minimum processing capability. This ensures that the tool can handle multiple scanning and data processing tasks concurrently without significant delays. The dual-core processor provides sufficient computational power to run the integrated security scanning tools and manage the web application backend smoothly.
- **Memory (RAM):** At least 4 GB of RAM is necessary to support the tool's operations. Adequate memory allows for efficient handling of multiple processes, such as executing scans, running scripts, and managing temporary data. This amount of RAM helps prevent performance bottlenecks and ensures that the tool can handle moderate-sized target systems and datasets effectively.
- **Storage:** A minimum of 100 GB of storage on either an HDD or SSD is required to accommodate the tool's software, scanning results, log files, and any auxiliary data generated during operations. While HDDs provide sufficient capacity, SSDs are preferred for faster data access speeds, which can improve the overall responsiveness and performance of the tool.
- **Network Connectivity:** Reliable internet connectivity is essential for the tool's full functionality. The internet connection enables the tool to download updates, security patches, and vulnerability databases to stay current with emerging threats. Additionally, network access is necessary for performing remote scans on target systems located outside the local environment, ensuring comprehensive coverage and flexibility.

1.4 Software Architecture

The application is designed following a modular software architecture, which promotes flexibility, maintainability, and scalability. This design approach divides the system into distinct components, each responsible for specific tasks, allowing independent development, testing, and updates. The primary components of the software architecture include:

- **Frontend:**
The user interface is built using HTML, CSS, and JavaScript. This combination enables the creation of a clean, intuitive, and responsive web interface where users can easily interact with the tool. HTML structures

- the content, CSS defines the visual style and layout, and JavaScript adds dynamic behaviour such as real-time feedback, interactive forms, and asynchronous data updates. The frontend ensures that users can submit scan requests, monitor progress, and view detailed results seamlessly.
- Backend:**
 The backend logic is implemented using the Flask framework, a lightweight and versatile Python web framework. Flask manages the core application logic, including handling HTTP requests, routing users to appropriate functionalities, and processing input data. It acts as the intermediary between the frontend and the underlying scanning modules, coordinating the execution of scanning tasks and compiling results. The Flask backend also manages error handling, user session management, and integration with external tools.
- Modules:**
 To maintain a clear separation of concerns, the application divides various scanning and reconnaissance tasks into distinct Python scripts or modules. Each module is specialized for a particular function, such as port scanning, vulnerability detection, or system information gathering. This modular approach allows developers to independently develop, update, or replace individual scanning components without affecting the overall system. It also facilitates easy integration of new scanning tools or techniques as cybersecurity needs evolve.



2. System

2.1. Functional requirements

The functional requirements specify the key features and behaviors that the automated vulnerability detection and information gathering tool must provide to fulfill its objectives. These requirements ensure the system operates as intended and meets user needs in performing comprehensive security assessments.

1. **User Interface**
 - The system shall provide an intuitive web-based interface accessible through standard web browsers.
 - The interface shall allow users to initiate scans by specifying target systems or IP addresses.
 - Users shall be able to view real-time progress updates during scanning processes.
 - The system shall display detailed scan results in an organized, easy-to-understand format.
 - Users shall have options to export scan reports in common formats (e.g., PDF, CSV).
2. **Target Specification**
 - The system shall accept single or multiple targets (IP addresses, domain names, or ranges) for scanning.
 - Input validation shall ensure targets are correctly formatted and reachable.
3. **Port Scanning**
 - The tool shall scan specified targets to detect open ports using standard scanning techniques.
 - It shall identify and report open TCP and UDP ports.
 - The system shall provide options for both quick scans and in-depth scanning modes.
4. **Service Enumeration**
 - The system shall identify services running on detected open ports, including service type and version where possible.
 - The tool shall detect common services such as HTTP, FTP, SSH, SMTP, and others.
5. **Vulnerability Scanning**
 - The tool shall check the target for known vulnerabilities based on up-to-date vulnerability databases and security advisories.
 - It shall highlight critical and high-severity vulnerabilities requiring immediate attention.
 - The system shall provide detailed information on each vulnerability, including description, risk level, and mitigation suggestions.
6. **Information Gathering**
 - The system shall collect relevant publicly available information about the target, such as DNS records, WHOIS data, and subdomain enumeration.
 - It shall incorporate OSINT techniques to enrich the target profile with external intelligence.
7. **Performance and Reliability**
 - The tool shall handle concurrent scanning tasks without significant degradation in performance.
 - It shall provide error handling and notify users of any issues during scans, such as unreachable targets or timeouts.
8. **Security and Access Control**
 - The system shall ensure secure communication between the frontend and backend components.
 - User inputs shall be sanitized to prevent injection attacks or misuse.
 - The system may include user authentication features to restrict access to authorized personnel only (optional based on deployment).

2.1.2 User requirements

To ensure the tool effectively meets the needs of its users, the following user requirements have been identified:

- **Ease of Use**

The application shall provide an intuitive and user-friendly interface designed to accommodate users with varying levels of technical expertise. Navigation and operation should be straightforward to minimize the learning curve and enhance overall usability.

- **Customization**

Users shall have the ability to customize scanning parameters according to their specific requirements. This includes selecting scan types, specifying target ranges, and adjusting scan depth to focus on areas of interest or to comply with organizational policies.

- **Performance**

The tool shall perform scans efficiently, optimizing the use of system resources such as CPU and memory to ensure minimal impact on the host system. Scanning processes should balance thoroughness with speed to provide timely results.

- **Support and Documentation**

Comprehensive documentation, including user guides and troubleshooting resources, shall be made available to assist users in effectively utilizing the tool and resolving common issues. Additional support mechanisms, such as FAQs or contact options, should be considered to enhance user experience.

2.1.3 Environmental requirements

The successful deployment and operation of the automated vulnerability scanning tool depend on the following environmental conditions:

- **Operating System Compatibility**

The tool shall be compatible with major operating systems including **Windows**, **Linux**, and **macOS**. This cross-platform support ensures flexibility in deployment across different user environments and IT infrastructures.

- **Software Dependencies**

The system requires **Python 3.x** as the core runtime environment. Additionally, the **Flask** framework is necessary for the web application functionality. Other Python libraries and packages, as specified in the project's dependency files (e.g., requirements.txt), must be installed to support various scanning and automation features.

- **Network Connectivity**

A stable internet connection is essential for the tool to perform certain scanning operations, such as retrieving vulnerability updates, accessing remote targets, and conducting information gathering from online public sources. Network stability directly impacts the accuracy and completeness of scan results.

2.2 Design and Architecture

The application is built upon a **client-server architecture** model, which separates user interaction from backend processing to ensure modularity, scalability, and ease of maintenance. The design consists of two primary components:

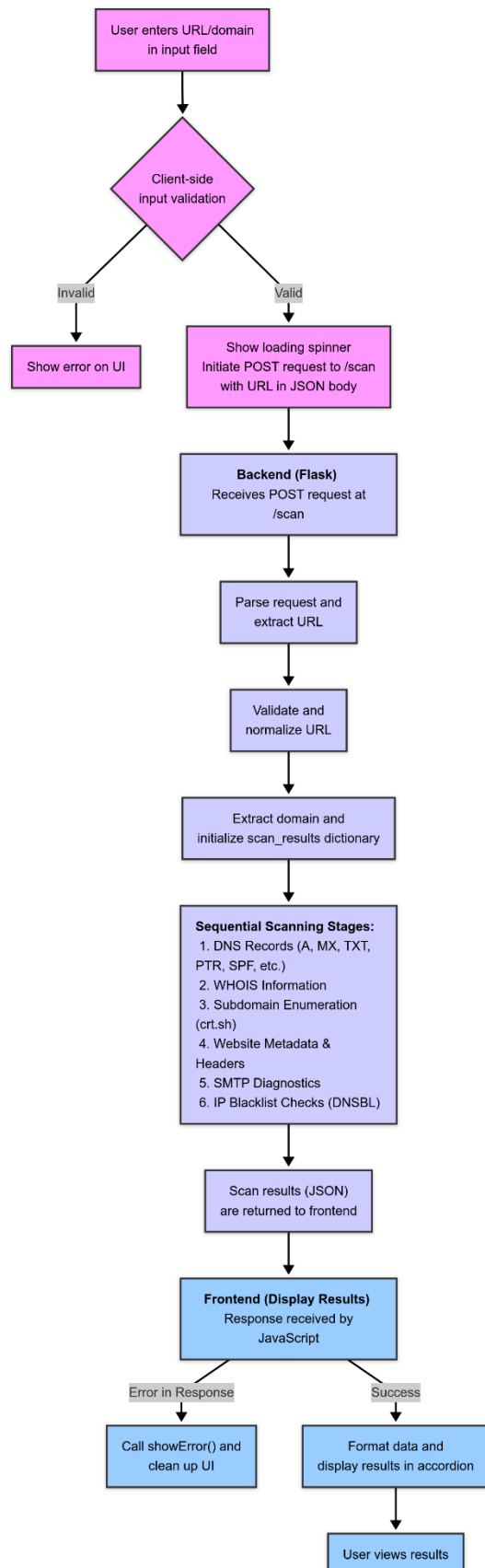
- **Client Side (Frontend)**

The client side comprises a web-based interface accessible through standard web browsers. This interface is developed using **HTML**, **CSS**, and **JavaScript** to provide users with an interactive and responsive experience. Users can input targets, configure scan options, monitor progress, and review detailed scan reports. The client side communicates with the server by sending requests and receiving results asynchronously, enabling smooth and real-time interaction.

- **Server Side (Backend)**

The server side is implemented using the Flask web framework in Python. It serves as the central processing unit of the application, responsible for handling incoming requests from the client, orchestrating the execution of various scanning and information gathering modules, and compiling the outputs into coherent results. The server manages task scheduling, integrates third-party security tools via Python scripts, and ensures efficient

resource utilization. By isolating the backend logic from the user interface, the system promotes better security, easier updates, and enhanced performance.



2.3 Implementation

The implementation phase focuses on bringing together the different components of the system to create a fully functional automated vulnerability scanning tool. Key steps involved in the implementation include:

- **Flask Application Setup**

The backend is developed using the Flask framework. This involves configuring the application, defining URL routes, and setting up request handlers that respond to client interactions. Each route corresponds to specific functionalities such as initiating scans, retrieving results, or managing configurations.

- **Development of Scanning Modules**

Individual Python scripts are created for each core scanning functionality — including port scanning, service enumeration, vulnerability detection, and information gathering. These scripts leverage existing security tools and libraries, automate their execution, and process their output to extract meaningful information.

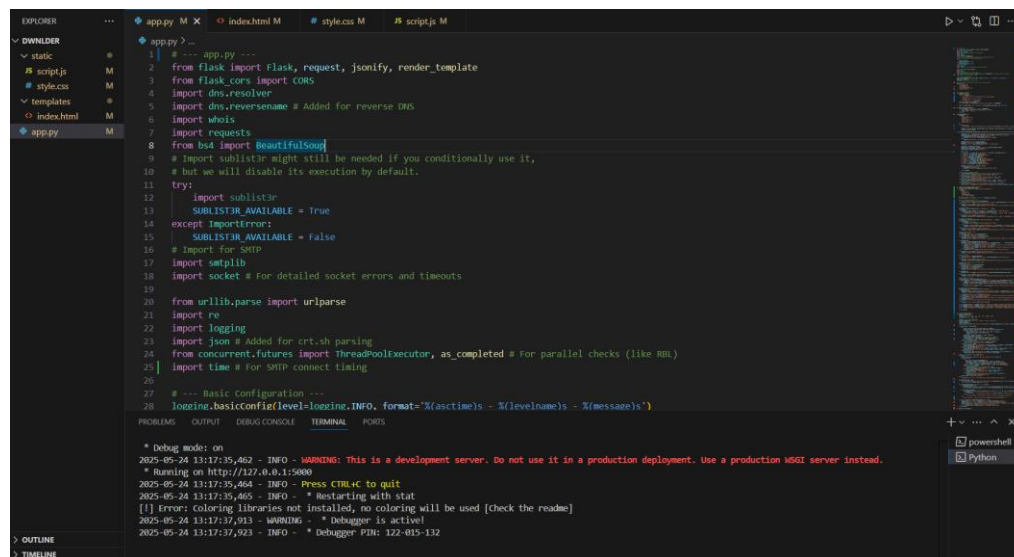
- **Frontend Integration**

The user interface is built using HTML and CSS to provide a clean and organized layout. These frontend templates are integrated within the Flask application using its templating engine (Jinja2), allowing dynamic rendering of scan results and interactive elements. JavaScript is incorporated to handle asynchronous requests and enhance user experience.

- **Frontend-Backend Communication**

To ensure smooth interaction, the system uses HTTP requests (GET and POST) to exchange data between the client and server. AJAX techniques allow asynchronous communication, enabling the frontend to send scan parameters and receive updates without reloading the page. This seamless communication pipeline enhances responsiveness and usability.

Through these coordinated implementation steps, the project delivers an automated tool that efficiently performs vulnerability assessments while providing a user-friendly interface for cybersecurity professionals.



```

1 # --- app.py ---
2 from flask import Flask, request, jsonify, render_template
3 from flask_cors import CORS
4 import dns.resolver
5 import dns.reversename # Added for reverse DNS
6 import whois
7 import requests
8 from bs4 import BeautifulSoup
9 # Import sublist3r might still be needed if you conditionally use it,
10 # but we will disable its execution by default.
11 try:
12     import sublist3r
13     SUBLIST3R_AVAILABLE = True
14 except ImportError:
15     SUBLIST3R_AVAILABLE = False
16 # Import for 2019
17 import smtplib
18 import socket # For detailed socket errors and timeouts
19
20 from urllib.parse import urlparse
21 import re
22 import logging
23 import json # Added for crt.sh parsing
24 from concurrent.futures import ThreadPoolExecutor, as_completed # For parallel checks (like RBL)
25 import time # For SMTP connect timing
26
27 # --- Basic Configuration ---
28 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

* Debug mode: on
2025-05-24 13:17:35,462 - INFO - WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
2025-05-24 13:17:35,464 - INFO - Press CTRL+C to quit
2025-05-24 13:17:35,465 - INFO - * Restarting with stat
[!] Error: coloring libraries not installed, no coloring will be used [check the readme]
2025-05-24 13:17:37,913 - WARNING - * Debugger is active!
2025-05-24 13:17:37,923 - INFO - * Debugger PIN: 122-015-132

```

2.4 Testing

Test Plan Objectives

The test plan for the automated vulnerability scanning tool is designed to achieve the following objectives:

- **Functionality Verification**

Confirm that all implemented features and modules perform according to the specified functional requirements, including scanning, enumeration, and information gathering capabilities.

- **Bug Identification and Resolution**

Detect any defects, errors, or performance bottlenecks within the system, and facilitate timely correction to enhance stability and user experience.

- **Cross-Platform Compatibility**

Ensure the application operates consistently and reliably across multiple operating systems, including Windows, Linux, and macOS, accommodating diverse user environments.

- **Accuracy and Reliability Validation**

Verify that the scanning results are precise, comprehensive, and dependable, providing trustworthy information for cybersecurity analysis and decision-making.

2.4.2 Security

To safeguard the automated vulnerability scanning tool and protect both the system and user data, several security measures have been implemented:

- **Input Validation**

All user inputs are rigorously validated and sanitized to prevent injection attacks such as SQL injection, command injection, and cross-site scripting (XSS). This ensures that only properly formatted and safe data is processed by the system, reducing the risk of exploitation.

- **Authentication**

The application incorporates secure login mechanisms to restrict access to authorized users only. Authentication protocols enforce strong password policies and may support additional security features such as multi-factor authentication (MFA) to enhance protection against unauthorized access.

- **Logging and Monitoring**

The system maintains detailed logs of user activities, scan executions, and system events. These logs support auditing and monitoring efforts, enabling administrators to track actions, detect suspicious behavior, and investigate security incidents effectively.

- **Error Handling**

Robust exception handling routines are implemented to gracefully manage errors and failures without exposing sensitive information. Error messages are designed to be informative yet generic, preventing leakage of internal system details that could aid attackers.

2.4.5 System Test

System testing is a critical phase in the development lifecycle aimed at validating the complete functionality and performance of the application in an environment that closely resembles real-world usage. The following activities are carried out during system testing:

- **Controlled Deployment**

The application is deployed in a test environment that simulates production conditions, including hardware specifications, network configurations, and operating system diversity. This ensures the testing environment is suitable for identifying environment-specific issues.

- **End-to-End Scan Execution**

Comprehensive scans are performed on predefined test targets using the full set of available features—such as port scanning, service enumeration, vulnerability analysis, and information gathering. This end-to-end testing ensures all modules work in coordination and produce the expected outcomes.

- **Report Accuracy Verification**

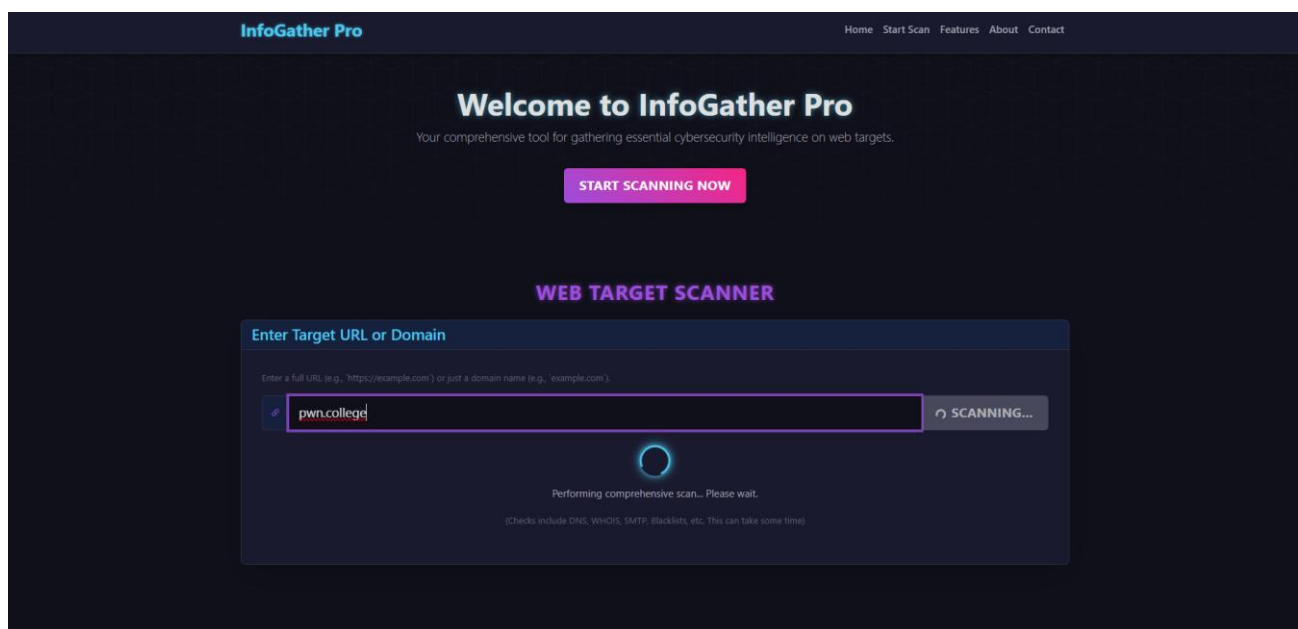
The results generated by the tool, including vulnerability details and scanned information, are carefully analyzed for accuracy and completeness. This step ensures that the tool provides reliable data that cybersecurity professionals can act upon.

- **Stability and Performance Evaluation**

The application is tested under varying loads to assess its stability, resource utilization, and response time. Stress tests are conducted to determine how the system behaves under heavy scanning conditions, helping to identify any potential performance bottlenecks or failures.

2.5 Graphical User Interface (GUI) Layout

The application features a clean, intuitive, and user-friendly graphical user interface (GUI) designed to streamline the vulnerability scanning and reporting process. The layout is structured into the following key components:



- **Dashboard**

The dashboard serves as the main interface upon login, providing a real-time overview of scanning activities, system status, and recent results. It offers quick access to initiate new scans, view historical data, and monitor ongoing operations, ensuring users stay informed briefly.

- **Scan Configuration**

This section allows users to input target details (such as IP addresses or domain names) and select specific scanning modules or options. The configuration panel supports customizable settings including scan depth, port range, and tool preferences, giving users control over the assessment scope.

- **Reports**

The reports section displays detailed output from completed scans in a readable and organized format. Users can navigate through findings, categorized by type or severity, and download reports in various formats (e.g., PDF or JSON) for documentation or further analysis.

- **Settings**

The settings panel provides options to customize the behavior of the tool, including user preferences,

scheduling automated scans, enabling/disabling specific features, and managing user accounts or authentication settings. This ensures flexibility in adapting the tool to different operational requirements.

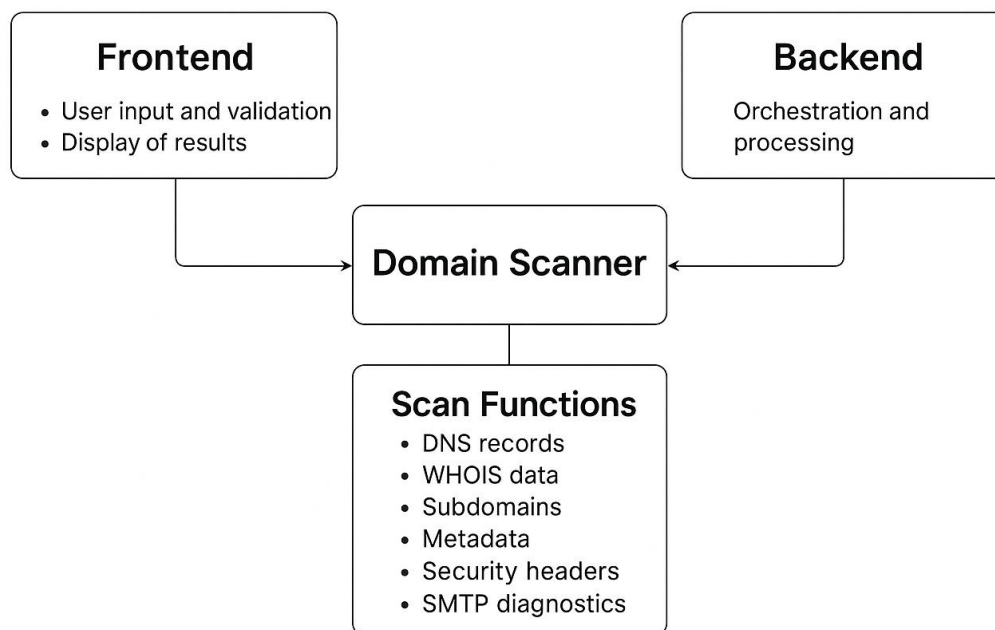
2.6 Valuation

The developed tool successfully achieves its primary goal of automating vulnerability scanning and information gathering, proving to be both efficient and reliable during testing and simulated use cases. By integrating multiple scanning techniques and data collection modules, it provides cybersecurity professionals with timely, actionable insights into the security posture of target systems.

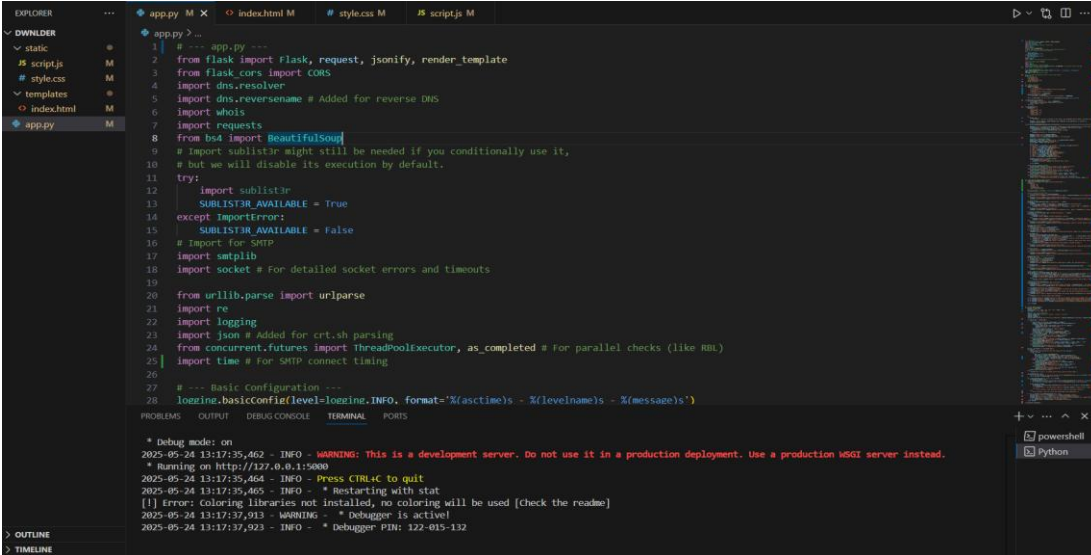
One of the standout features of the tool is its modular design, which promotes flexibility and scalability. Each scanning function is encapsulated in its own module, making the system easy to maintain and extend. New tools or capabilities can be added with minimal disruption to the existing architecture. This design not only supports future development but also enables customization based on evolving security needs.

Overall, the tool demonstrates strong performance, accurate reporting, and high usability. It is particularly valuable in environments where quick threat identification and proactive security assessments are crucial.

Domain Scanner: Structural Diagram

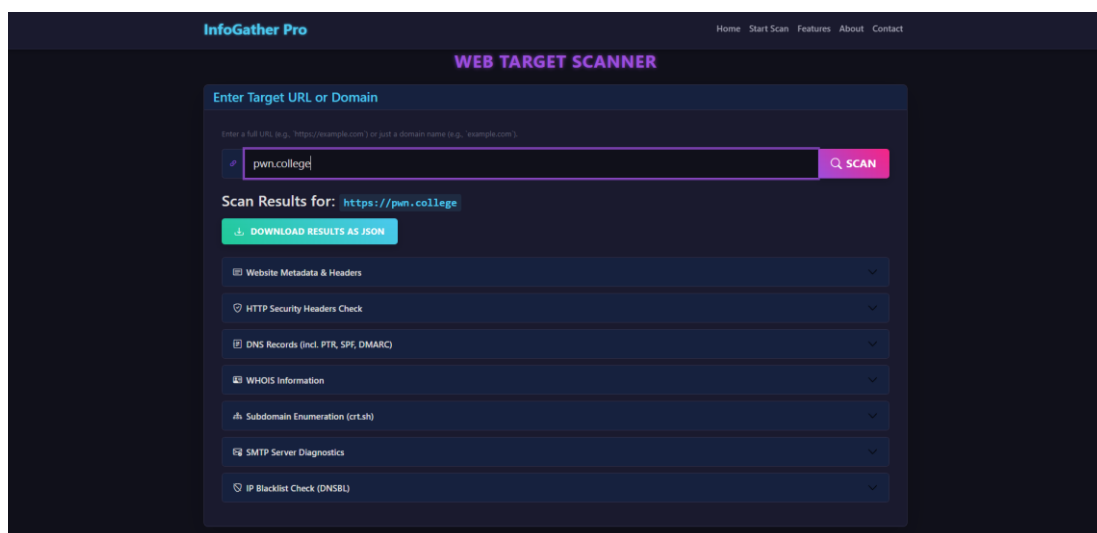
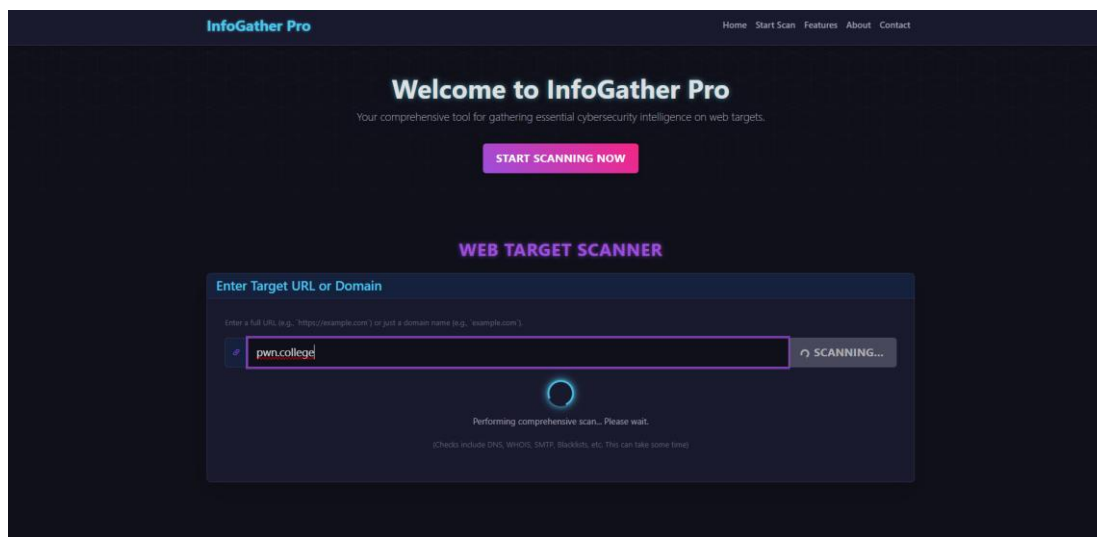


3. Snapshots of the Project



```
1 # --- app.py ---
2 from flask import Flask, request, jsonify, render_template
3 from flask_cors import CORS
4 import dns.resolver
5 import dns.reversename # Added for reverse DNS
6 import whois
7 import requests
8 from bs4 import BeautifulSoup
9 # import sublistr might still be needed if you conditionally use it,
10 # but we will disable its execution by default.
11 try:
12     import sublistr
13     SUBLISTR_AVAILABLE = True
14 except ImportError:
15     SUBLISTR_AVAILABLE = False
16 # Import for SMTP
17 import smtplib
18 import socket # For detailed socket errors and timeouts
19
20 from urllib.parse import urlparse
21 import re
22 import logging
23 import json # Added for crt.sh parsing
24 from concurrent.futures import ThreadPoolExecutor, as_completed # for parallel checks (like RBL)
25 import time # for SMTP connect timing
26
27 # --- Basic Configuration ---
28 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
```

* Debug mode: on
2025-05-24 13:17:35,462 - INFO - WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
2025-05-24 13:17:35,464 - INFO - Press CTRL+C to quit
2025-05-24 13:17:35,465 - INFO - * Restarting with stat
[!] Error: coloring libraries not installed, no coloring will be used [check the readme]
2025-05-24 13:17:37,913 - WARNING - * Debugger is active
2025-05-24 13:17:37,923 - INFO - * Debugger PID: 122-015-132



Website Metadata & Headers

```
{
  "description": "No description found",
  "headers": {
    "Connection": "keep-alive",
    "Content-Length": "37908",
    "Content-Type": "text/html; charset=utf-8",
    "Date": "Sat, 24 May 2025 07:49:37 GMT",
    "Server": "nginx/1.26.0",
    "Set-Cookie": "session=378e8385-1062-40e3-b3fa-b978c0ff3feb.9KsGg9csvgkKgKjuFips9Xe9Yo; HttpOnly; Path=/; SameSite=Lax",
    "Strict-Transport-Security": "max-age=31536000"
  },
  "server": "nginx/1.26.0",
  "status_code": 200,
  "technologies": [
    "Nginx"
  ],
  "title": "pwn.college"
}
```

HTTP Security Headers Check

Analysis of common HTTP security headers and their configurations.

```
Strict-Transport-Security (HSTS): max-age=31536000

--- Missing Headers (-) ---
X-Frame-Options
X-Content-Type-Options
Content-Security-Policy (CSP)
Referrer-Policy
Permissions-Policy

--- Configuration Warnings/Observations (!) ---
Server header may reveal specific software/version: 'nginx/1.26.0'. Consider minimizing this information.

--- Recommendations (*) ---
Implement X-Frame-Options (e.g., 'DENY' or 'SAMEORIGIN') to protect against clickjacking attacks.
Implement X-Content-Type-Options: nosniff to prevent browsers from MIME-sniffing a response away from the declared content-type.
Implement Content-Security-Policy (CSP) as a powerful defense against XSS and data injection attacks.
Consider implementing Referrer-Policy to control how much referrer information is sent with requests.
Consider implementing Permissions-Policy to control which browser features can be used by the page.
Obscure or remove detailed version information from the 'Server' header to reduce information leakage.
```

DNS Records (incl. PTR, SPF, DMARC)

Note: Reverse DNS (PTR), SPF, and DMARC results are included within this section.

```
{
  "A": [
    "206.206.192.59"
  ],
  "AAAA": [
    "No record found"
  ],
  "CNAME": [
    "No record found"
  ],
  "DKIM_Note": "DKIM records usually exist at selector._domainkey subdomain (e.g., google._domainkey.domain.com) and require separate lookups.",
  "DMARC": [
    "No DMARC record found"
  ],
  "MX": [
    "1 aspmx.l.google.com.",
    "10 aspmx2.googlemail.com.",
    "10 aspmx3.googlemail.com."
  ]
}
```

WHOIS Information

```
{
  "address": [
    "Kalkofnsvegur 2"
  ],
  "city": [
    "Reykjavik"
  ],
  "country": [
    "IS"
  ],
  "creation_date": "2018-08-22T06:33:56",
  "dnssec": "unsigned",
  "domain_name": [
    "PWN.COLLEGE"
  ],
  "email": [
    "abuse@namecheap.com",
    "fe058779b334f249b0852330c1bd1a0.protect@withheldforprivacy.com"
  ]
}
```

Subdomain Enumeration (crt.sh)

```
--- Subdomain Enumeration Summary ---
Found 16 total unique subdomains (using crt.sh).

--- Source Status ---
- crtsh: Success (16 found)
- sublist3r: Disabled

--- Errors Encountered ---
No critical errors encountered during subdomain enumeration.

--- Discovered Subdomains ---
autotest.pwn.college
beta.pwn.college
cse466.pwn.college
cse5194.pwn.college
ctf.pwn.college
dojo.pwn.college
dreamport-1.pwn.college
```

IP Blacklist Check (DNSBL)

Checks domain/mail server IPs against common DNS-based blacklists.

```
--- Blacklist Check (DNSBL) ---

--- Summary ---
IPs Checked: 1
Blacklists Queried (per IPv4): 5
Listings Found: 2
Timeouts During Check: 0
Errors During Check: 0

--- Details per IP ---

[IP: 206.206.192.59]
LISTED on: cbl.abuseat.org, zen.spamhaus.org
(Checked 5 lists for this IP: 3 clean, 2 listed, 0 timeouts, 0 errors)
```


4.Tools used

The tool has 7 main feature categories, each providing specific insights:

- **Website Metadata & Headers:**

What it does:

Fetches the main page of the target URL, extracts its title, meta description, HTTP response headers (like Server, Content-Type, Set-Cookie), and tries to identify common web technologies used (e.g., WordPress, jQuery, Nginx).

Results Meaning:

Title/Description: Useful for understanding the website's purpose, can sometimes reveal internal project names or hint at the target's nature.

Reveal server software (e.g., Server: Apache/2.4.52), caching policies, cookies being set, and potentially other backend technologies (X-Powered-By: PHP/8.1).

Technologies: Knowing the tech stack (e.g., "WordPress", "React") helps an attacker focus on vulnerabilities specific to those technologies or their versions.

Pen Testing Application: Identifying the server software (e.g., "Apache", "Nginx", "IIS") and its version can lead to searching for known exploits for that specific version. The "X-Powered-By" header can reveal backend languages (PHP, ASP.NET), guiding further attack vectors. Technologies like WordPress, if outdated, have many known vulnerabilities.

- **HTTP Security Headers Check:**

What it does:

Analyses the HTTP response headers (obtained from the metadata check) for the presence and correct configuration of important security-related headers.

Results Meaning:

Present/Missing: Shows which recommended security headers are implemented (e.g., Strict-Transport-Security, X-Frame-Options, Content-Security-Policy) and which are missing.

Warnings/Recommendations: Highlights misconfigurations (e.g., a weak X-Frame-Options value) or suggests improvements.

Overall Info: Gives a summary of the security header posture.

Pen Testing Application:

Missing or misconfigured security headers can indicate vulnerabilities:

Missing X-Frame-Options or Content-Security-Policy frame-ancestors can lead to Clickjacking.

Missing Strict-Transport-Security (HSTS) can allow SSL stripping attacks.

Weak Content-Security-Policy (CSP) might not effectively prevent XSS.

Presence of Server or X-Powered-By with detailed versions is an information disclosure.

- **DNS Records (incl. PTR, SPF, DMARC):**

What it does:

Queries DNS servers for various records associated with the domain:

A/AAAA: IP addresses (IPv4/IPv6) of the domain.

MX: Mail exchange servers responsible for handling email.

NS: Name servers authoritative for the domain.

TXT: Text records, often used for SPF (Sender Policy Framework), DMARC (Domain-based Message Authentication, Reporting & Conformance), domain verification, etc.

CNAME: Canonical Name, an alias for another domain.

SOA: Start of Authority, provides authoritative information about the DNS zone.

PTR: Pointer record (Reverse DNS), maps an IP address back to a hostname. (Done for IPs found in A/AAAA). It specifically extracts SPF and DMARC from TXT records.

Results Meaning:

A/AAAA: The actual servers hosting the website/services.

MX: Mail server hostnames; crucial for email security checks.

NS: The DNS servers managing the domain's records; could be a target or point to hosting provider.

TXT/SPF/DMARC: Indicate email security configurations. Weak SPF/DMARC can make the domain vulnerable to email spoofing.

PTR: Can reveal hostnames associated with IPs, sometimes exposing internal naming conventions or other services on the same IP.

Pen Testing Application:

Discovering all associated IP addresses and hostnames helps map the target's infrastructure. Weak SPF/DMARC records can be exploited in phishing campaigns (making spoofed emails appear more legitimate). PTR records might reveal other domains or services hosted on the same server.

- **WHOIS Information:**

What it does:

Queries WHOIS databases for domain registration details.

Results Meaning:

Provides information like:

Registrar (e.g., GoDaddy, Namecheap).

Registration, expiration, and update dates.

Name servers (often matches DNS NS records).

Registrant contact information (name, organization, email, address, phone) – though often masked by privacy services.

Pen Testing Application:

Contact details (if not private) can be used for social engineering or phishing. Registration dates can sometimes indicate how established a domain is. Name servers can confirm DNS information or point to the hosting provider.

- **Subdomain Enumeration (crt.sh):**

What it does:

Queries Certificate Transparency (crt.sh) logs. Crt.sh aggregates SSL/TLS certificates issued by public Certificate Authorities. If a certificate was issued for sub.example.com, then sub.example.com is a known subdomain.

Results Meaning:

Provides a list of subdomains associated with the target domain that have had SSL/TLS certificates issued for them.

Pen Testing Application:

Subdomains often host different applications, forgotten services, or development environments that might be less secure than the main website. Discovering these expands the attack surface. For example, dev.example.com or internal.example.com could be interesting targets.

- **SMTP Server Diagnostics:**

What it does:

Identifies mail servers from MX DNS records.

For each mail server, it resolves its hostname to an IP address.

It attempts to connect to the mail server on port 25 (standard SMTP port).

If successful, it records the connection time, captures the server's welcome banner, and attempts a HELO command.

Results Meaning:

Status: Indicates if the connection was successful, failed, or timed out.

Banner: The welcome message from the mail server, often revealing the SMTP server software and version (e.g., Postfix, Exim, Microsoft Exchange).

HELO response: Shows if the server accepts basic SMTP commands.

Errors: Details any connection or command errors.

Pen Testing Application: The banner can reveal the mail server software and version, which can be checked for known vulnerabilities. Open mail relays (though not directly tested here, a successful connection is the first step) could be abused for spam. Understanding mail infrastructure is key for phishing attack planning.

- **IP Blacklist Check (DNSBL):**

What it does:

Collects IP addresses associated with the domain (from A/AAAA records) and its mail servers (from MX records).

For each unique IP address, it queries a predefined list of DNS-based Blackhole Lists (DNSBLs like SpamCop, Spamhaus).

Results Meaning:

Listed/Not Listed: Indicates if an IP address is found on any of the checked blacklists.

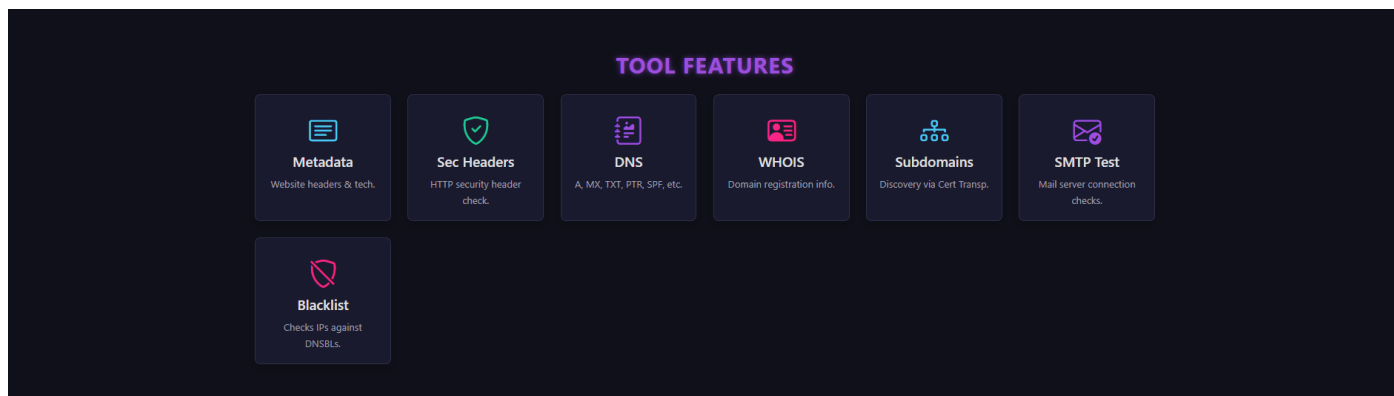
Summary: Shows how many IPs were checked, how many listings were found, and if any errors/timeouts occurred during the checks.

Pen Testing Application: If a target's mail server IP is on a blacklist, it suggests potential issues with spamming or malware distribution from that IP, which might indicate a compromised server or poor email practices. For a web server IP, being on a blacklist could mean it was previously associated with malicious activity. This is generally more of an operational/reputational concern but can provide context.

5. Conclusions

The **Automated Vulnerability and Information Gathering Tool** proves to be a significant asset in the field of cybersecurity, offering a streamlined and efficient approach to identifying security weaknesses in target systems. By integrating a variety of scanning techniques and information-gathering methods within a modular, user-friendly framework, the tool reduces the time and effort required for manual assessments.

Its automation capabilities not only improve the speed and accuracy of vulnerability detection but also support proactive threat management by delivering timely insights. With features such as customizable scan configurations, real-time reporting, and an intuitive GUI, the tool is well-suited for use by both security experts and IT teams aiming to strengthen their organization's cybersecurity posture. Its design allows for continuous improvement, ensuring adaptability to emerging threats and evolving security requirements.



5. Further development or research

While the current version of the **Automated Vulnerability and Information Gathering Tool** provides a solid foundation for cybersecurity assessments, there are several opportunities for enhancement and expansion that could further increase its effectiveness and versatility:

- **Integration with Databases**
Incorporating persistent storage through database integration would allow scan results, user preferences, and historical data to be stored and queried efficiently. This would enable trend analysis, long-term monitoring, and better incident response tracking.
- **Machine Learning for Predictive Analysis**
Implementing machine learning models could enhance the tool's intelligence by enabling predictive analytics. By learning from previous scans and detected vulnerabilities, the system could anticipate potential weak points and suggest proactive mitigation strategies.
- **Expanded Scanning Capabilities**
Future versions can include modules specifically designed for web application vulnerability scanning, such as detecting SQL injection, XSS, and authentication flaws. This would broaden the tool's applicability to more diverse cybersecurity environments.

6. References

- **Flask Documentation**
Flask, a lightweight WSGI web application framework, was used for backend development.
Available at: <https://flask.palletsprojects.com>
- **OWASP Top Ten**
A standard awareness document for developers and web application security, providing insights into the most critical security risks.
Available at: <https://owasp.org/www-project-top-ten/>
- **Python Official Documentation**
The primary programming language used in this project. The documentation was referenced for syntax, libraries, and best practices.
Available at: <https://docs.python.org/3/>
- **Nmap Security Scanner**
A powerful network scanning tool used as part of the vulnerability assessment modules.
Available at: <https://nmap.org/>
- **GITHUB PROJECTS**
- **LINUX BASICS FOR HACKERS -[OccupyTheWeb](#)**