A

MINI PROJECT REPORT

On

DEVELOPMENT OF BASIC WEBSITE VULNERABILITIES SCANNER USING PYTHON

A dissertation submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

By

D. Pallavi (22UP1A0542)
B. Sravani (22UP1A0517)
E. Tejaswini (22UP1A0551)
G. Babitha (22UP5A0503)

Under the guidance of

Dr. G.Rajesh
Associate Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
VIGNAN'S INSTITUTE OF MANAGEMENT AND TECHNOLOGY FOR WOMEN
(AN AUTONOMOUS INSTITUTION)

Kondapur (V), Ghatkesar (M), MedchalDist– 501 301.
Affiliated to Jawaharlal Nehru Technological University, Hyderabad.
www.vmtw.edu.in

2024-2025

VIGNAN'S INSTITUTE OF MANAGEMENT AND TECHNOLOGY FOR WOMEN
(An Autonomous Institution)
(Sponsored by Lavu Educational Society)
[Affiliated to JNTUH, Hyderabad & Approved by AICTE, New Delhi]
Kondapur (V), Ghatkesar (M), Medchal –Malkajgiri (D)- 501301.Phone:9652910002/3

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the project work entitled "**Development Of Basic Website Vulnerabilities Scanner Using Python**" submitted by **D. Pallavi (22UP1A0542), B. Sravani (22UP1A0517), E. Tejaswini (22UP5A0551) G. Babitha (23UP5A0503**) in the partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering**, **Vignan's Institute of Management and Technology for Women** is a record of bonafide work carried by them under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or institute for the award of any degree.

PROJECT GUIDE
Dr. G. Rajesh
(Associate Professor)

THE HEAD OF DEPARTMENT
Mrs. M. Parimala
(Associate Professor)

External Examiner

VIGNAN'S INSTITUTE OF MANAGEMENT AND TECHNOLOGY FOR WOMEN

(An Autonomous Institution)

(Sponsored by Lavu Educational Society)

[Affiliated to JNTUH, Hyderabad & Approved by AICTE, New Delhi]

Kondapur (V), Ghatkesar (M), Medchal –Malkajgiri (D)- 501301.Phone:9652910002/3

A+ NAAC

Accredited by NBA CSE & ECE

DECLARATION

We hereby declare that the results embodied in the project entitled "**Development Of Basic Website Vulnerabilities Scanner Using Python**" is carried out by us during the year 2025-2026 in partial fulfillment of the award of Bachelor of Technology in Computer Science and Engineering from Vignan's Institute of Management and Technology for Women is an authentic record of our work under the guidance of Dr. G. Rajesh. We have not submitted the same to any other institute or university for the award of any other Degree.

D. Pallavi (22UP1A0542)

B. Sravani (22UP1A0517)

E. Tejaswini (22UP1A0551)

G. Babitha (22UP5A0503)

ACKNOWLEDGEMENT

We would like to express sincere gratitude to **Dr G. APPARAO NAIDU**, **Principal, Vignan's Institute of Management and Technology for Women** for his timely suggestions which helped us to complete the project in time.

We would also like to thank our madam **Mrs. M. Parimala, Head of the Department and Associate Professor, Computer Science and Engineering** for providing us with constant encouragement and resources which helped us to complete the project in time.

We would also like to thank our Project guide **Mr Dr.G. Rajesh, Associate Professor, Computer Science and Engineering**, for providing us with constant encouragement and resources which helped us to complete the project in time with his valuable suggestions throughout the project. We are indebted to him for the opportunity given to work under his guidance.

Our sincere thanks to all the teaching and non-teaching staff of Department of Computer Science and Engineering for their support throughout our project work.

D. Pallavi (22UP1A0542)
B. Sravani (22UP1A0517)
E. Tejaswini (22UP1A0551)
G. Babitha (22UP5A0503)

INDEX

LIST OF FIGURES

1. ABSTRACT

This project focuses on the development of a basic website vulnerability scanner using Python. The scanner aims to identify common web security vulnerabilities, providing an introductory understanding of web application security principles. With the increasing reliance on web applications, cybersecurity risks such as misconfigurations, cross-site scripting (XSS), and SQL injection have each become important threats. This project mainly focuses on developing a functional website vulnerability scanner to detect common security weaknesses within web applications. The project emphasizes the importance of secure coding practices and highlights the prevalence of common web vulnerabilities. While this scanner is a simplified implementation for educational purposes and does not replace comprehensive vulnerability scanning tools, it serves as a valuable learning experience in web security concepts and programming. The output of the scanner includes a report of identified potential vulnerabilities, along with explanations of their potential impact and basic remediation advice. This project underscores the need for proactive security measures in web application development and aims to raise awareness about common web security risks. It also highlights the importance of ethical considerations in vulnerability scanning, emphasizing the need for explicit permission before scanning any website.The scanner serves as truly a foundational step forward toward improving cybersecurity by identifying of risks before attackers are able for to exploit them.

CHAPTER 2

INTRODUCTION

2.1 OBJECTIVE

The main objective of this project is to design and implement a **basic website vulnerability scanner using Python** that can detect common web-based security threats in a simplified and user-friendly manner. This tool is primarily developed for educational purposes.

With the growing reliance on web applications for communication, business transactions, data storage, and user interaction, securing web platforms has become essential. Many websites, especially small or educational ones, are vulnerable to various types of attacks due to a lack of security awareness during development. This project addresses that issue by offering a lightweight scanner capable of identifying several common vulnerabilities that can be exploited if not properly handled.

The vulnerabilities that this scanner aims to detect include:

- **SQL Injection** – A technique used to manipulate backend databases through insecure input fields.
- **Cross-Site Scripting (XSS)** – Injecting malicious scripts into websites to execute in users' browsers.
- **Directory Traversal** – Gaining unauthorized access to files and directories outside the intended web root.
- **Command Injection** – Executing arbitrary commands on the server through input fields.
- **Open Redirect** – Redirecting users to untrusted sites via manipulated URLs.
- **Host Header Injection** – Modifying HTTP headers to trick the server into generating malicious responses.

2.2 Existing System

In the current landscape of CYBERSECURITY, various vulnerability scanning tools are available in the market, such as Burp Suite, OWASP ZAP, NESSUS.They can perform deep scans, simulate real-world attack scenarios, and generate comprehensive vulnerability reports.

Most existing systems use advanced techniques like fuzzing, automated crawling, and heuristic analysis to discover hidden and complex security issues. These scanners also support integration with development environments, DevSecOps pipelines, and continuous monitoring tools, making them an essential part of modern web application security.

For educational institutions and beginner developers, the barrier to entry with these tools can be high. Many of them offer free versions, but with limited features, restricted scans, or time-bound access.

2.2.1 Limitations of Existing System

Despite their advantages, the existing systems have several limitations when considered from a
learning or small-scale development perspective:

- **Complexity**: Tools like Burp Suite and OWASP ZAP have a steep learning curve. New users may find it overwhelming to configure scans, understand the technical output, and apply appropriate fixes.
- **Cost and Licensing**: Many professional vulnerability scanners are commercial products. The full versions of these tools require paid licenses, which can be expensive for students or small teams.
- **Resource Intensive**: These tools often require powerful machines and significant system resources for full scans, which may not be feasible for every user.

- **Overkill for Basic Learning**: For educational purposes, these tools may offer more features than necessary, which can distract from fundamental learning objectives.
- **Limited Customization for Beginners**: Customizing scans or focusing on specific vulnerabilities requires a deeper understanding of security concepts and tool internals, which is often beyond the scope of beginner-level users.

2.3 Proposed System

To address the challenges and limitations of existing vulnerability scanning tools, this project proposes the development of a **basic website vulnerability scanner using Python**. The proposed system is intended to be lightweight, easy to understand, and focused on educational and small-scale usage. It provides a minimal yet functional web interface where users can input a URL and receive immediate feedback on basic security issues.

The scanner works by sending specially crafted HTTP requests to the provided URL and analysing the responses to detect patterns or behaviours indicative of common vulnerabilities. These include:

- SQL Injection
- Cross-Site Scripting (XSS)
- Directory Traversal
- Command Injection
- Open Redirect
- Host Header Injection

The tool is designed for simplicity. Users only need to enter the target website URL, and the scanner will perform checks for the above vulnerabilities automatically. After scanning, the system generates a report indicating whether the site is vulnerable or safe for each type of attack, and provides a short explanation with basic mitigation advice.

Ethical use is also emphasized, with clear instruction that the scanner should only be used on websites with proper authorization. This reinforces the importance of responsible CYBERSECURITY practices and discourages misuse.

2.3.1 Advantages of Proposed System

The proposed system offers several benefits over traditional and existing vulnerability scanners, especially in the context of learning and introductory usage:

- **User-Friendly Interface**: A clean, simple interface that allows users to initiate scans with minimal effort.
- **Educational Value**: Each vulnerability result is accompanied by an explanation and mitigation tip, promoting learning through practical use.
- **Lightweight and Fast**: Unlike enterprise tools, this scanner is lightweight, requires minimal system resources, and provides quick results.
- **Open Source and Customizable**: Built entirely using Python and Flask, it is easy to read, modify, and extend according to user needs.
- **No Installation Overhead**: Can be run locally on any machine with Python installed, without the need for complex configurations or installations.
- **Focus on Common Vulnerabilities**: Covers widely known and impactful vulnerabilities that are relevant to real-world applications.
- **Ethical Awareness**: Reinforces the importance of obtaining permission before scanning websites, promoting responsible cybersecurity behaviour.

3.LITERATURE SURVEY

With the rapid growth of the internet and online services, web applications have become a central part of modern digital infrastructure. These applications handle a wide range of sensitive information such as personal data, payment details, and organizational records. As a result, web applications have increasingly become prime targets for CYBER attacks. Attackers commonly exploit vulnerabilities like SQL Injection, Cross-Site Scripting (XSS), Directory Traversal, and Command Injection to compromise systems and access unauthorized data.

To combat these threats, a variety of security tools and techniques have been developed over the years. Professional-grade tools like Burp Suite, OWASP ZAP, NIKTO, and ACUNETIX are widely used for automated vulnerability scanning, penetration testing, and generating detailed security reports. These tools offer advanced detection capabilities and are commonly used in industry by security professionals. Academic research in the field of web application security has proposed numerous defensive techniques, such as input validation, parameterized queries, output encoding, and content filtering. These practices help mitigate vulnerabilities by ensuring that user inputs are handled securely. Additionally, the concept of automated vulnerability scanning has received considerable attention, with research focusing on improving detection accuracy, minimizing false positives, and ensuring ethical usage.

Despite the availability of sophisticated tools and research findings, there remains a gap for lightweight, beginner-friendly tools that serve both functional and educational purposes. Many existing tools focus on deep scanning and professional reporting but are not designed with simplicity and learning in mind.

4.SYSTEM ANALYSIS

System analysis involves understanding the requirements and goals of the system to ensure it effectively addresses web security issues. The motivation behind this project is the rising number of CYBER threats targeting web applications and the lack of awareness among beginner developers.

This tool aims to fill that gap by offering a simple, lightweight scanner that detects common vulnerabilities like:

SQL Injection,XSS,Directory Traversal,Command Injection,Open Redirects,Host Header Injection.

The scanner sends test inputs to the target URL and analyzes responses to identify flaws, presenting results clearly with mitigation suggestions. Ethical use is emphasized—users are advised to scan only websites they have permission to test, promoting responsible CYBERSECURITY practices.

4.1 Purpose

System analysis involves understanding the requirements and goals of the system to ensure it effectively addresses web security issues. The motivation behind this project is the rising number of CYBER threats targeting web applications and the lack of awareness among beginner developers.

This tool aims to fill that gap by offering a simple, lightweight scanner that detects common vulnerabilities like:

SQL Injection,XSS,Directory Traversal,Command Injection,Open Redirects,Host Header Injection.

The scanner sends test inputs to the target URL and analyzes responses to identify flaws, presenting results clearly with mitigation suggestions. Ethical use is emphasized—users are advised to scan only websites they have permission to test, promoting responsible CYBERSECURITY practices.

4.2 Scope

The scope of this project encompasses the development of a basic yet functional website vulnerability scanner using Python and Flask that can identify commonly known web application vulnerabilities. The project is focused on educational and small-scale security testing, primarily targeting users such as students, beginner developers, and academic institutions that want to understand and experiment with web security principles in a safe and controlled environment.

The scanner is capable of testing for the following vulnerabilities:SQL Injection,Cross-Site Scripting (XSS),Directory Traversal,Command Injection,Open Redirect,Host Header Injection.

The system is limited to analyzing vulnerabilities based on simple crafted requests and observing the server's response. It does not perform deep crawling, authentication testing, or complex penetration testing simulations. The primary goal is to provide a straightforward scanning tool that demonstrates how basic security flaws can be identified through input manipulation and response analysis.

In terms of technical boundaries, the scanner:

- Operates on client-server architecture via Flask.
- Supports HTTP GET-based parameter manipulation.
- Assumes public-facing, unsecured endpoints for analysis.

- Runs locally without any external dependencies beyond standard Python libraries.

4.3 Feasibility Study

The feasibility study evaluates whether the proposed Basic Website Vulnerability Scanner (built with Python and Flask) can be successfully developed and used with the available resources.

This scanner is meant for educational and awareness purposes. It is low-cost, uses open-source tools, and meets the growing need for simple web security testing, especially in academic and developer communities.

The feasibility is assessed in three main areas:

- **Economic Feasibility:** Low development cost using free technologies.
- **Technical Feasibility:** Simple to build using known tools like Python and Flask.
- **Social Feasibility:** Easily accepted by students, learners, and beginner developers.
- This study supports the project's development and confirms that it is practical, useful, and sustainable.

4.3.1 Economic Feasibility

Economic feasibility assesses whether the project is cost-effective and can be developed within a reasonable budget compared to its benefits.

For this project, economic feasibility is highly favorable. It relies entirely on free, open-source tools such as Python, Flask, and libraries like requests and urllib, eliminating the need for paid software or licenses. The project is suitable for student-led development, minimizing labor costs. Since it is intended for educational purposes, there are no costs related to marketing, distribution, or commercial deployment. Maintenance is also low due to the simplicity of the system.

The return on investment (ROI) is gained through enhanced cybersecurity awareness, practical learning, and improved secure coding practices. These educational and long-term benefits make the project economically viable and valuable, especially in academic and training environments.

Expense Item	Cost Estimate
Software (Python, Flask, etc.)	₹0 (Open Source)
Development Labor	Minimal (Student-led)
Hardware (Laptop/PC)	Already available
Hosting (if needed)	Free or low-cost
Maintenance	Low

4.3.2 Technical Feasibility

Technical feasibility evaluates whether the proposed system can be effectively developed and operated using available tools, technologies, and skills.

For this project, the feasibility is strong. The scanner is built using Python, a beginner-friendly and widely supported language. It runs smoothly on basic hardware without requiring specialized setups. The system uses standard libraries like requests and URL lib.parse to send HTTP requests and analyze responses by manipulating URL parameters and observing behavior.

Key Components Supporting Technical Feasibility:

Programming Language: Python (easy to learn, well-documented)

Frontend: HTML/CSS (for simple user interface)

Libraries: requests, urllib.parse (standard Python libraries)

Platform: Cross-platform (Windows, macOS, Linux)

The required skills are minimal—basic programming and web concepts are sufficient. The system also allows for future upgrades such as adding advanced scan types, integrating APIs, or saving results in a database.

4.3.3 Social Feasibility

The proposed Basic Website Vulnerability Scanner, built with Python, is designed for students, beginner developers, and educational institutions. It serves both as a learning tool and a cybersecurity awareness platform.

This tool helps users explore web security in a hands-on way and fills the gap left by complex or costly commercial scanners. Its open-source and lightweight nature ensures it can be used even with limited resources, promoting equal learning opportunities.

Key Social Benefits:

- Raises awareness of common cybersecurity risks
- Encourages ethical and responsible use of security tools
- Makes security education more accessible
- Supports practical learning in classrooms
- Aligns with global efforts to improve cyber hygiene and digital literacy

4.4 Requirement Analysis

Requirement analysis is a crucial phase in software development that identifies and documents the system's needs, forming the foundation for designing a system that meets user expectations and functions efficiently

The analysis covers both functional and non-functional requirements. Functional needs include accepting user input, scanning URLs, detecting specific vulnerabilities, and displaying results. Non-functional requirements focus on usability, reliability, performance, and security. The target users—students, beginner developers, and cybersecurity learners—expect a simple, intuitive tool that helps them understand web vulnerabilities in real-world scenarios.

Key requirements identified are a web interface for URL input and result display, detection logic for vulnerabilities like SQL Injection and XSS, basic reporting with explanations and fixes, responsiveness across browsers, and minimal dependencies for easy deployment using Python and Flask. The system is designed to scan publicly accessible web pages via HTTP GET parameters without performing authenticated scans or exploiting vulnerabilities, only detecting their presence.

4.4.1 Functional Requirements

Functional requirements define the core operations, tasks, and features that the system must perform to meet its intended purpose. These requirements represent how it should interact with users, and what outputs it should generate.

For this project, the basic website vulnerability scanner is designed to allow users to input a target URL and receive a report highlighting any common security vulnerabilities found in the specified web application. The scanner focuses on detecting specific vulnerabilities using simple request/response analysis techniques, making the system useful and easy to understand for beginners and educational users.Below are the key functional requirements of the system:

1. User Input Interface

– The system must provide a web-based interface where users can enter a URL for scanning.

– The input must be validated to ensure it is in proper URL format.

2. Scan Initialization

– Upon receiving the URL, the system should trigger a scanning process.

– The scan must be performed using predefined test cases for each targeted vulnerability.

3. Vulnerability Detection

– The system must check the given URL for common vulnerabilities, including:

- SQL Injection
- Cross-Site Scripting (XSS)
- Directory Traversal
- Command Injection
- Open Redirect
- Host Header Injection

4. Report Generation

– The system must compile scan results and display them to the user.

– Each result must indicate whether a vulnerability was detected and provide a brief explanation.

5.Navigation and Usability

The web interface should include clear navigation, with options to input a new URL, rescan , or view previous results (if implemented).

4.4.2 Non-functional requirements

Non-functional requirements describe the attributes and qualities that the system must exhibit, rather than specific behaviors or tasks. While functional requirements focus on what the system should do, non-functional requirements define how the system should perform and behave in different situations. These requirements ensure that the system is efficient, user-friendly, secure, and maintainable.

The following non-functional requirements have been identified.

1. Usability

– The system must be simple and intuitive to use, even for users with limited technical knowledge.

– Scan results should be easy to interpret, with clear indications of vulnerabilities and their descriptions.

2. Performance

– The scanner should return results within a reasonable amount of time (typically a few seconds for basic checks).

– It should handle multiple consecutive scans without crashing or slowing down significantly.

3. Portability

– The system should be cross-platform and capable of running on Windows, Linux, or macOS without major changes.

– It should only require standard Python packages and Flask, ensuring minimal dependency issues.

4. Reliability

– The scanner must handle unexpected inputs or errors gracefully without crashing.

– The system should be tested to ensure that it consistently produces accurate results under different conditions.

5. Security

– While the scanner tests for security vulnerabilities, it must also avoid becoming a source of risk itself.

– Input validation and proper output handling must be implemented to prevent misuse.

– The system should not store or transmit sensitive information.

6.maintainability

– The codebase should be clean, well-documented, and modular to facilitate updates and improvements.

4.5 Requirements Specification

The requirements specification provides a structured summary of all essential hardware, software, and programming tools needed to develop, deploy, and run the proposed system a basic website vulnerability scanner using Python. It also helps stakeholders understand the technological scope and environment in which the application will function.

This project is designed to be lightweight and compatible with most standard computing systems, emphasizing accessibility and ease of deployment. The application relies on minimal resources, as it primarily performs lightweight HTTP requests and string-matching techniques to detect common vulnerabilities.

The requirements are categorized into three major parts:

- Hardware Requirements
- Software Requirements
- Language Specification

These requirements also support portability, as the system can be run on various operating systems (Windows, Linux, macOS) as long as Python is installed. Additionally, by using open-source technologies, the project ensures cost effectiveness and wide accessibility, making it ideal for academic and personal use cases.

4.5.1 Hardware requirements

Hardware requirements refer to the physical components and system specifications needed to develop, deploy, and run the website vulnerability scanner efficiently. Since this project is lightweight and designed for educational purposes, it does not require high-end or specialized hardware. The system operates mainly by sending basic HTTP requests and processing responses, without heavy graphical interfaces or databases.

Here is the recommended hardware specification for running the system:

Component	**Minimum Requirement**
Processor (CPU)	Intel Core i3 or equivalent (1.8 GHz or above)
RAM	4 GB (8 GB recommended for multitasking)
Storage	1 GB free disk space
Monitor	13-inch or larger display (HD resolution or above)
Input Devices	Keyboard and Mouse
Network	Internet connection (for accessing external websites during scan)
Operating System	Windows 10/11, Linux (Ubuntu), or macOS

The frontend runs in a web browser, while the backend uses lightweight Python scripts with the Flask framework, making it highly portable and resource-friendly. Although the scanner can run on low-end systems, better performance is achieved with faster processors and more RAM, especially when handling multiple scans or multitasking. Overall, the system's minimal resource consumption makes it suitable for a wide range of devices.

4.5.2 Software requirements

The website vulnerability scanner is developed using Python, a versatile and beginner-friendly programming language widely used in cybersecurity and automation. The project relies on two main Python libraries:

- Requests – Used to send HTTP GET requests to the target URLs.
- urllib.parse – Helps manipulate and modify URLs during vulnerability testing.

All required tools and libraries are open-source and freely available, making the project affordable and accessible, especially for students, educators, and individual learners.

Software Component	**Specification / Version**
Operating System	Windows 10/11, Ubuntu/Linux, or macOS
Programming Language	Python 3.8 or above
Web Framework	Flask 2.0 or above
Python Libraries	requests, urllib, re (all standard libraries)
Web Browser	Google Chrome, Mozilla Firefox, or equivalent
Code Editor (Optional)	Visual Studio Code

Terminal / Command Prompt	For running Python scripts and Flask server

The simplicity of the software requirements supports quick installation, smooth testing, and a strong focus on core learning objectives.

4.5.3 language specification

The language specification outlines the programming languages and scripting technologies used in the development of the website vulnerability scanner. The choice of language is crucial, as it directly impacts the readability, maintainability, performance, and scalability of the application.

For this project, the scanner is built using the following technologies:

Component	**Language Used**
Backend Logic	Python
Web Framework	Flask (Python-based)
Frontend Interface	HTML, CSS (Basic Styling)
HTTP Handling	Python (requests, urllib)
Template Rendering	Jinja2 (Flask default)

1. **Python**
2. Python is used as the core backend language for the scanner due to its simplicity, readability, and cross-platform compatibility, making it ideal for students and beginners.

It is used to:

- Send HTTP requests to target URLs
- Craft payloads for testing vulnerabilities (e.g., SQL Injection, XSS)
- Parse and analyze server responses

2.Flask

Flask is a lightweight and modular web framework built in Python. It allows quick development of small web applications with minimal configuration. Flask handles routing, request/response management, and template rendering.

Key reasons for choosing Flask:

- Easy integration with Python scripts.
- Lightweight and flexible ideal for a single-page scanner interface.
- Built-in support for Jinja2 templating engine.

3.HTML&CSS

HTML & CSS is used to create the frontend user interface where users can input URLs and view scan results. Basic CSS is used for styling to ensure the page is visually clean and easy to navigate.HTML forms and buttons are used for user input, while Flask dynamically renders the output using Jinja2 templates.

Jinja2 : Jinja2 is the default templating engine for Flask. It is used to pass dynamic data (such as scan results) from the backend to the frontend, rendering them into HTML pages in real time.

5.SYSTEM DESIGN

The system design outlines the structure and workflow of the website vulnerability scanner. It focuses on creating a lightweight, modular, and user-friendly framework. The design is divided into the following components:

- **Backend Logic:** Performs scanning and processes responses.
- **Frontend Interface:** Allows users to input URLs and view results.
- Controller/Router: Connects frontend and backend using Flask.
- Scanner Engine: Executes security checks and analyzes responses.

It ensures ethical use by detecting, not exploiting, vulnerabilities and promoting user awareness.

5.1 System Architecture

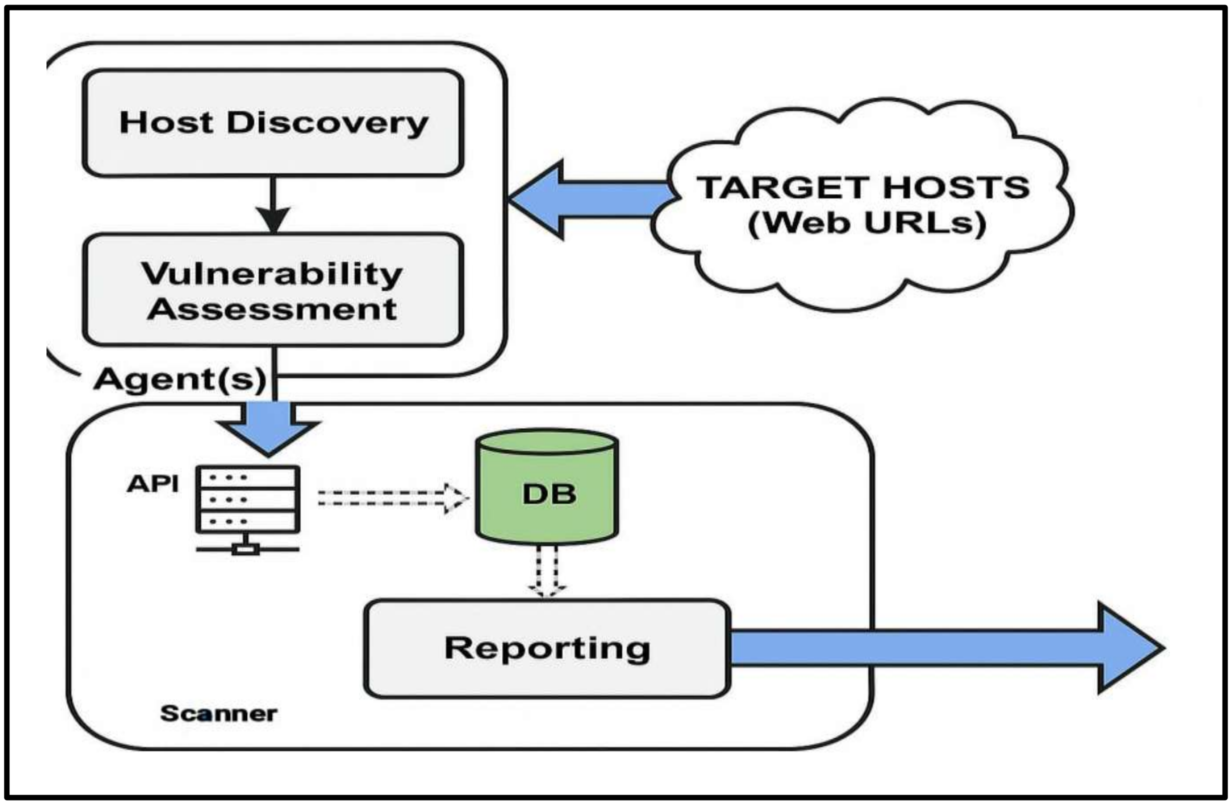


Fig no: 5.1.1 System architecture

Overall System Layout:
The system is divided into two main functional blocks:

1. Host Discovery & Vulnerability Assessment (upper box)
2. Scanner Engine with API, Database, and Reporting (lower box)

5.2 Description

1. Host Discovery & Vulnerability Assessment

Host Discovery: Identifies and verifies target web applications by resolving domain names and checking connectivity.

Vulnerability Assessment: Performs security checks for common vulnerabilities like SQL Injection and XSS.

Input: List of web URLs to scan.
Output to Agent(s): Discovered hosts and scan scope are passed to the scanner engine.

2. Scanner (Core Engine)

API: Connects the user interface or scripts with backend logic; handles scan requests.

Database: Stores scan results, vulnerabilities, and related data.

Reporting: Creates readable reports with findings and basic remediation suggestions.

Final Output: Reports are shared with the user via a web interface, download link, or email.

5.3 UML Diagrams

Unified Modeling Language (UML) diagrams visually represent the structure and behavior of a software system. They help in understanding system functionality, guiding development, and improving communication among stakeholders.

For the Basic Website Vulnerability Scanner, three key UML diagrams are used:

- **Use Case Diagram** – Shows how users interact with the system.
- **Activity Diagram** – Illustrates the step-by-step scanning workflow and decisions.
- **Sequence Diagram** – Depicts the interaction between system components over time.

Each diagram highlights different parts of the scanner's design, aligned with its features and use cases.

5.3.1 Use Case Diagram

This UML Use Case Diagram shows how a user (actor) interacts with the vulnerability scanner system.

Actor: Represents the end-user (developer, student, or tester) using the scanner.

Use Cases

Submit Target URL - User enters the website address to scan.

Configure Scan Options – Optionally select specific vulnerabilities or scan depth.

Start Vulnerability Scan – System sends crafted requests and logs results.

View/Download Report – User can see or export scan results (e.g., PDF/CSV).

Relationships - Lines connect the actor to each use case, indicating what actions the user can perform**.**

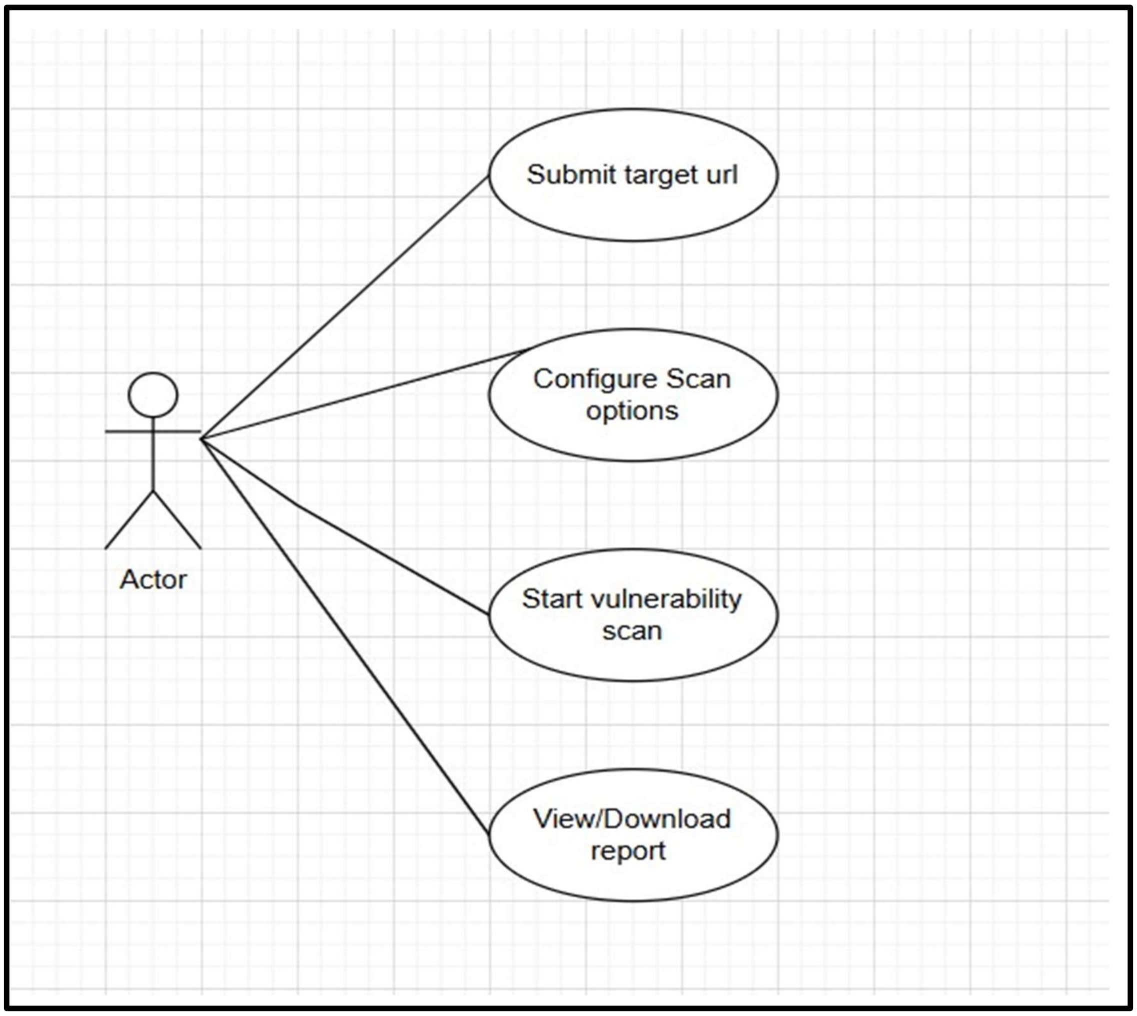


Fig no: 5.3.1.1 Use case Diagram

.

5.3.2 Activity Diagram

Activity Diagram a type of UML diagram that represents the dynamic flow of actions and decisions within a system. It visually maps out how a process proceeds step-by-step, including decision points and alternate paths.

Initial Node (Solid black circle)

- Marks the start of the process.

Start

- A starting action where the system begins processing.

Accept input URL

- The user provides a URL that will be scanned for vulnerabilities.
- This is the first user interaction and entry point for initiating a scan.

Decision: Connection error?

- The system checks whether it can reach the target website.
- If yes (i.e., there's a connection error), it generates a Report indicating the failure.
- If no, it proceeds to perform vulnerability checks.

Check for vulnerabilities

- This is the core scanning logic.
- The system tests for known vulnerabilities such as SQL Injection, XSS, Directory Traversal, etc.

Decision: Vulnerable?

- The system analyzes the results of its tests.
- If it detects a vulnerability, it goes to Output.
- If no vulnerabilities are found, it proceeds to output a clean result.

Output / Output result

- Displays or generates the results of the scan, whether vulnerabilities were found or not.
- This may include a summary report with suggestions or confirmations of safety.
- Final Node Marks the end of the process flow.

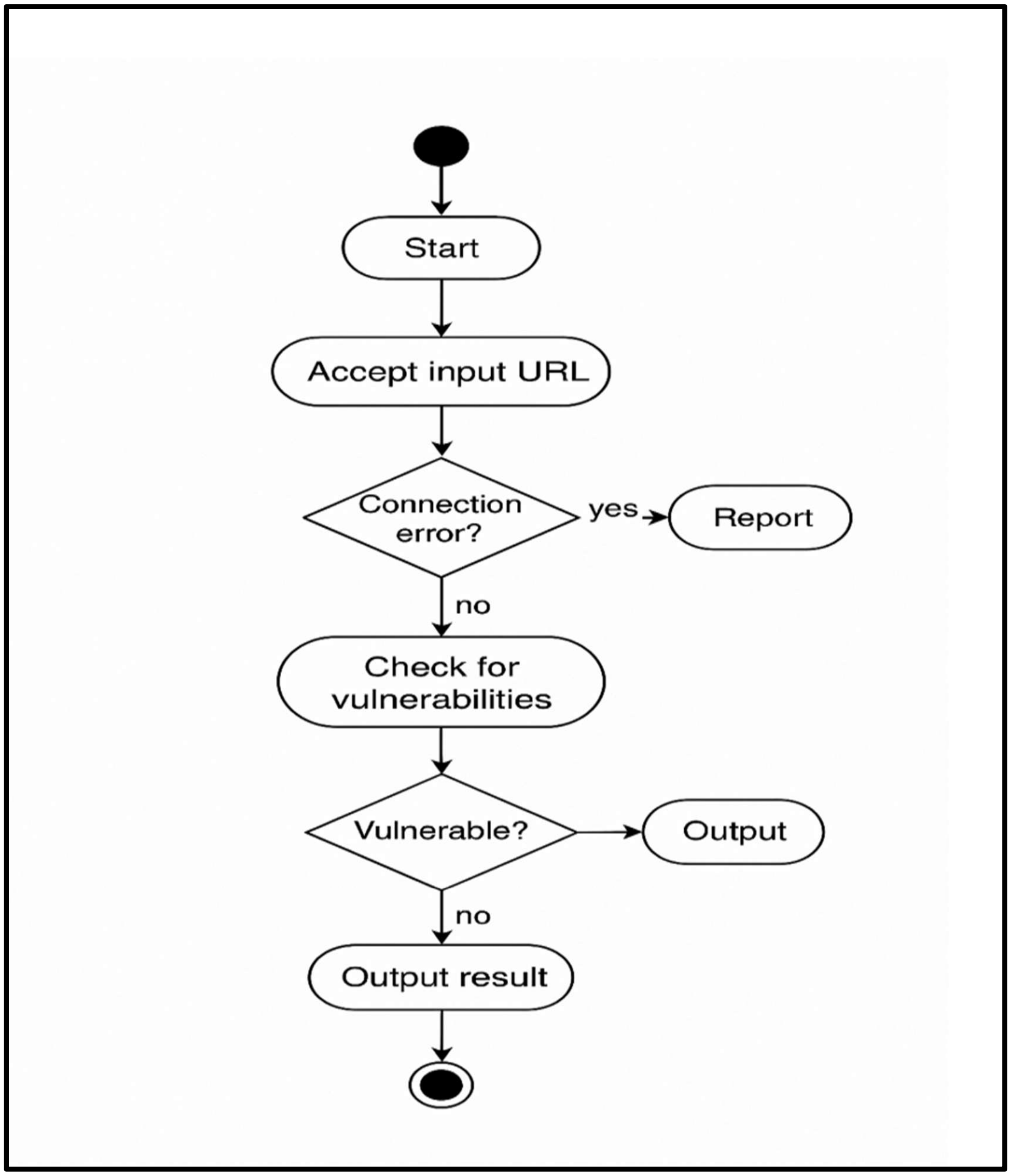


Fig no: 5.3.2.1 Activity Diagram

5.3.3 Sequence Diagram

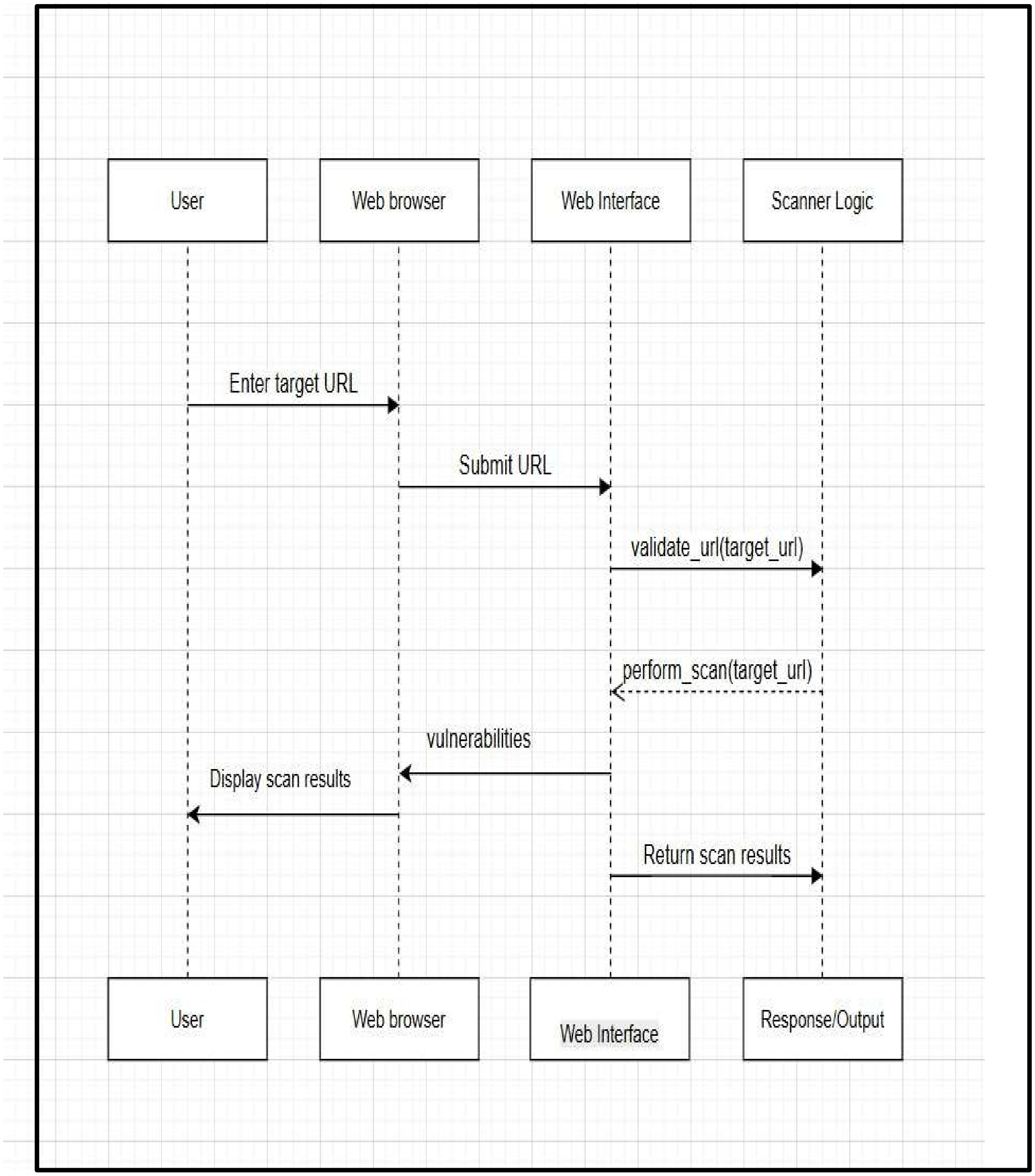


Fig no: 5.3.2.2 sequence Diagram

Participants (from left to right):

User – The person interacting with the system

Web Browser – The medium used by the user to access the web application

Web Interface – The frontend part of the vulnerability scanner (built using Flask/HTML)

Scanner Logic – The backend that contains the core scanning functionality

Sequence of Actions:

1. Enter target URL:

The user enters a web address (URL) into a form field on the web interface through their browser.

2.Submit URL:

The web browser submits the entered URL to the web application backend.

3.validate_url(target_url):

The Web Interface passes the target URL to the Scanner Logic to validate whether it's correctly formatted and safe for scanning.

4.perform_scan(target_url):

Once validated, the Scanner Logic performs the actual scanning by sending requests with test payloads to the given URL to check for vulnerabilities (e.g., SQL Injection, XSS).

5.Return scan results:

The scan results (a list of detected vulnerabilities, if any) are sent back from the Scanner Logic to the Web Interface.

vulnerabilities → Display scan results:

The Web Interface sends the results back to the browser, which displays them to the user in a readable format.

6.IMPLEMENTATION AND RESULTS

6.1 Algorithm used

1. Accept the Target URL as input (e.g., http://example.com/page?id=1)
2. Parse the URL and identify parameters (like id=1)
3. Inject predefined malicious payloads into the parameters

   Examples:

   - SQLi: ' OR '1'='1
   - XSS: <script>alert(1)</script>
   - LFI: ../../../../etc/passwd
4. Reconstruct the URL with the payload injected

   Example: http://example.com/page?id=' OR '1'='1
5. Send an HTTP GET request to the modified URL using requests.get()
6. Capture and analyze the response content (HTML/text returned by server)
7. Search for known indicators of vulnerability using pattern matching:
   - SQL Error messages (e.g., "You have an error in your SQL syntax")
   - Reflection of script tags (indicating XSS)
   - Presence of sensitive file contents (e.g., "root:x:0:0" from /etc/passwd)
8. If any patterns are detected → Mark as Vulnerable

   Otherwise → Mark as Safe
9. Return the scan result to the user in the report format

6.2 Sample code

Main : app.py

```
from flask import Flask, request, render_template
import requests
from urllib.parse import urlparse, parse_qs, urlencode, urlunparse
app = Flask(__name__)
# 1. SQL Injection
def check_sql_injection(raw_url):
    parts = urlparse(raw_url)
    params = parse_qs(parts.query)
    params['id'] = ["' OR '1'='1"]
    new_query = urlencode(params, doseq=True)
     test_url = urlunparse((parts.scheme, parts.netloc, parts.path, parts.params,
new_query, parts.fragment))
    try:
        response = requests.get(test_url, timeout=5)
        body = response.text.lower()
        for error in ["sql syntax", "mysql", "ora-", "query failed"]:
            if error in body:
                return ("Vulnerable", "Use parameterized queries or prepared statements
to prevent SQL Injection.")
        return ("Safe", "No SQL Injection vulnerability detected.")
    except Exception as e:
        return ("Error", str(e))
# 2. XSS
def check_xss(raw_url):
     parts = urlparse(raw_url)
    params = parse_qs(parts.query)
    params['q'] = ['<script>alert("XSS")</script>']
    new_query = urlencode(params, doseq=True)
    test_url = urlunparse((parts.scheme, parts.netloc, parts.path, parts.params,
new_query, parts.fragment))
    try:
        response = requests.get(test_url, timeout=5)
        if '<script>alert("XSS")</script>' in response.text:
            return ("Vulnerable", "Use output encoding and input validation to prevent
XSS.")
        return ("Safe", "No XSS vulnerability detected.")
except Exception as e:
        return ("Error", str(e))
```

```
# 3. Directory Traversal
def check_directory_traversal(raw_url):
    parts = urlparse(raw_url)
    params = parse_qs(parts.query)
    params['file'] = ['../../../../etc/passwd']
    new_query = urlencode(params, doseq=True)
     test_url = urlunparse((parts.scheme, parts.netloc, parts.path, parts.params,
new_query, parts.fragment))
    try:
        response = requests.get(test_url, timeout=5)
        if "root:x:" in response.text:
               return ("Vulnerable", "Validate and sanitize file path input to prevent
directory traversal.")
        return ("Safe", "No Directory Traversal vulnerability detected.")
    except Exception as e:
        return ("Error", str(e))
# 4. Command Injection
def check_command_injection(raw_url):
parts = urlparse(raw_url)
    params = parse_qs(parts.query)
    params['cmd'] = ['; ls']
    new_query = urlencode(params, doseq=True)
     test_url = urlunparse((parts.scheme, parts.netloc, parts.path, parts.params,
new_query, parts.fragment))
    try:
        response = requests.get(test_url, timeout=5)
        indicators = ['bin', 'usr', 'lib', 'home']
        if any(i in response.text.lower() for i in indicators):
            return ("Vulnerable", "Sanitize and validate command inputs. Avoid using
shell commands directly with user input.")
        return ("Safe", "No Command Injection vulnerability detected.")
    except Exception as e:
        return ("Error", str(e))
# 5. Open Redirect
def check_open_redirect(raw_url):
    parts = urlparse(raw_url)
    params = parse_qs(parts.query)
    params['redirect'] = ['https://evil.com']
    new_query = urlencode(params, doseq=True)
     test_url = urlunparse((parts.scheme, parts.netloc, parts.path, parts.params,
new_query, parts.fragment))
    try:
        response = requests.get(test_url, allow_redirects=False, timeout=5)
       if 'Location' in response.headers and 'evil.com' in response.headers['Location']:
                  return ("Vulnerable", "Validate and whitelist redirect URLs to prevent
Open Redirect attacks.")
        return ("Safe", "No Open Redirect vulnerability detected.")
    except Exception as e:
        return ("Error", str(e))
# 6. Host Header Injection
def check_host_header_injection(url):
    try:
         headers = {'Host': 'malicious.com'}
         response = requests.get(url, headers=headers, timeout=5)
        if 'malicious.com' in response.text:
             return ("Vulnerable", "Use strict host header validation and do not reflect
user-supplied host headers.")
        return ("Safe", "No Host Header Injection vulnerability detected.")
    except Exception as e:
        return ("Error", str(e))
@app.route('/')
def home():
    return render_template("index.html")
@app.route('/scan', methods=["POST"])
def scan():
    url = request.form['url']
    results = {
        "SQL Injection": check_sql_injection(url),
        "XSS": check_xss(url),
        "Directory Traversal": check_directory_traversal(url),
            "Command Injection": check_command_injection(url),
        "Open Redirect": check_open_redirect(url),
        "Host Header Injection": check_host_header_injection(url)
    }
    return render_template("result.html", url=url, results=results)
if __name__ == "__main__":
    app.run(debug=True)
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
   <head>
    <meta charset="UTF-8">
     <title>Website Vulnerability Scanner</title>
      <style>
      body {
        font-family: Arial, sans-serif;
        background-color: #f9f9f9;
         padding: 40px;
          color: #333;
}h1 {
  text-align: center;
  color: #444;
  margin-bottom: 30px;
}
.form-container {
  max-width: 500px;
  margin: auto;
  background-color: #fff;
  border: 1px solid #ddd;
  padding: 30px;
  border-left: 6px solid #5a5a5a;
  border-radius: 4px;
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.05);
}
input[type="text"] {
  width: 100%;
  padding: 12px;
  margin: 10px 0 20px 0;
  border: 1px solid #ccc;
  border-radius: 4px;
  font-size: 16px;
}
button {
  width: 100%;
  padding: 12px;
  background-color: #007bff;
  color: white;
  border: none;
  border-radius: 4px;
  font-size: 16px;
cursor: pointer;
}
button:hover {
  background-color: #0056b3;
}
.footer {
  text-align: center;
  margin-top: 20px;
  font-size: 0.9em;
  color: #666;
}
</style>
</head>
<body>
  <h1>Website Vulnerability Scanner</h1>
   <div class="form-container">
     <form action="/scan" method="post">
            <input type="text" name="url" placeholder="Enter target URL (e.g.,
http://example.com)" required>
       <button type="submit">Start Scan</button>
      </form>
    </div>
     <div class="footer"> &copy; 2025 Vulnerability Scanner </div>
     </body> </html>
```

Result.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Scan Results</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 40px;
            background-color: #f9f9f9;
            color: #333;
        }
        h2 {
            color: #444;
        }
        .vuln {
            margin-bottom: 20px;
            padding: 15px;
            border: 1px solid #ddd;
            border-left: 6px solid #5a5a5a;
            background-color: #fff;
        }
        .safe {
            border-left-color: green;
        }
        .vulnerable {
            border-left-color: red;
        }
    .error {
            border-left-color: orange;
        }
        .status {
            font-weight: bold;
        }
        .advice {
            margin-top: 5px;
            font-size: 0.95em;
        }
        a {
            text-decoration: none;
            color: #007bff;
        }
        a:hover {
            text-decoration: underline;
        }
    </style>
</head>
<body>
    <h2>Scan Results for {{ url }}</h2>

    {% for vuln, result in results.items() %}
        {% set status, advice = result %}
         <div class="vuln {% if status == 'Safe' %}safe{% elif status == 'Vulnerable'
%}vulnerable{% else %}error{% endif %}">
                <div><strong>{{ vuln }}:</strong> <span class="status">{{ status
}}</span></div>
            <div class="advice">{{ advice }}</div>
        </div>
{% endfor %}
    <a href="/">Scan another site</a>
</body>
</html>
```

7.SYSTEM TESTING

System testing ensures the complete website vulnerability scanner functions as intended. It verified both functional and non-functional aspects, including accuracy, usability, performance, and reliability.

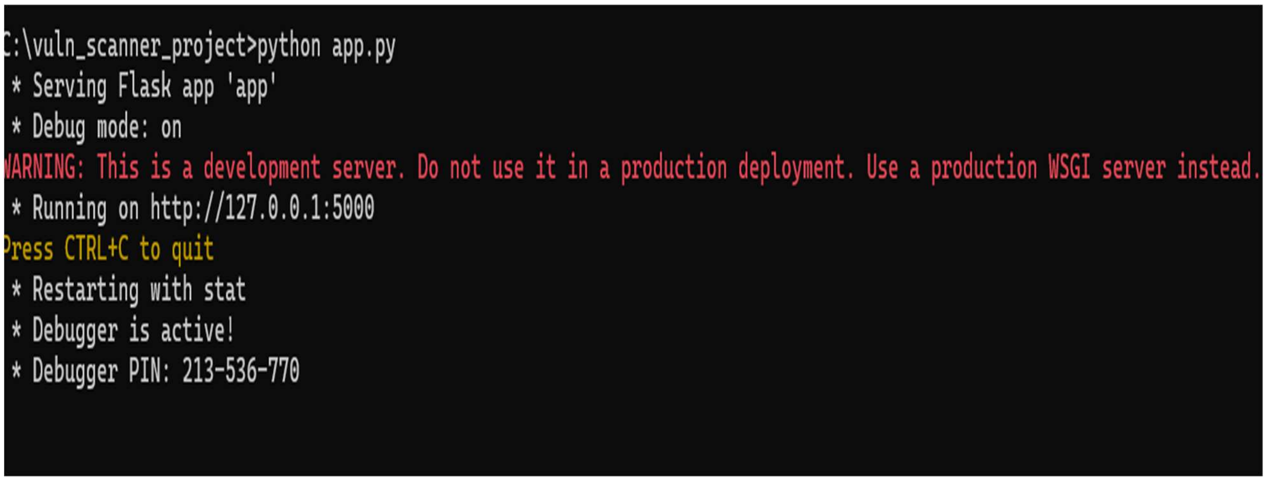
Types of Tests:

1. Functional Testing – Verified scan accuracy using known vulnerable pages and correct report generation.
2. Input Validation – Checked with valid, invalid, and malformed URLs, ensuring proper error messages.
3. Usability Testing – Tested navigation, clarity of results, and responsive messages.
4. Performance Testing – Measured response times and checked system behavior under load.
5. Error Handling – Ensured smooth handling of errors like 404/500 without crashing.
6. Ethical Use – Verified presence of a disclaimer to promote responsible use.

Test Case ID	**Test Scenario**	**Input Data**	**Expected Output**	**Actual Output status**	**Status**
TC_01	Valid URL input	http://testphp.vulnweb.com	Scanner should start and perform all checks	Scanner executed pass	Pass
TC_02	Invalid URL input	http://testphp.vulnweb.com/listproducts.php?cat=1'	Detect SQL error in response	SQLi detected	Pass

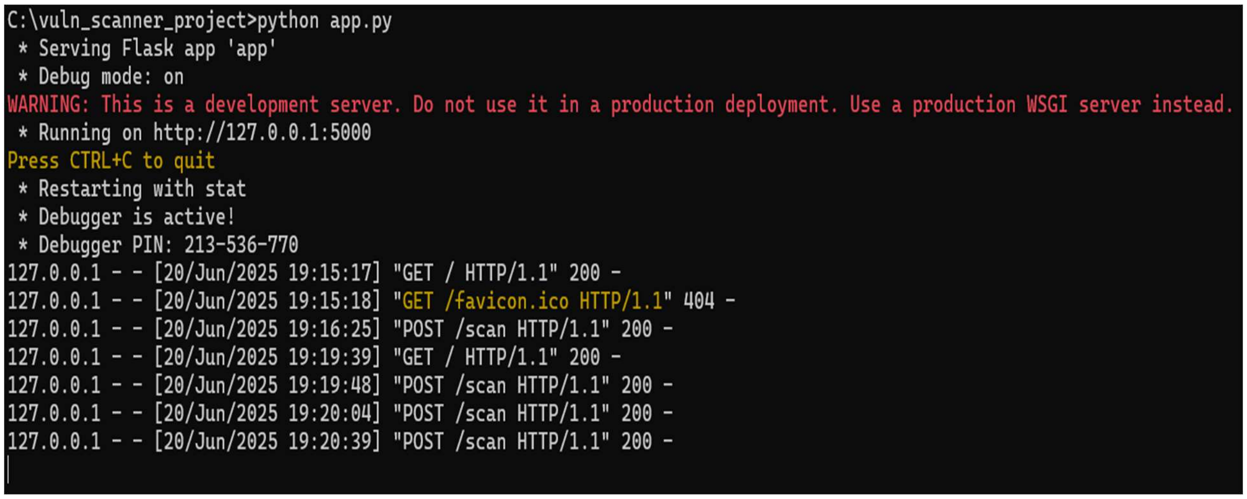
TC_03	Blank URL submission	(empty field)	Prompt user to enter a valid URL	Prompt shown	Pass
TC_04	SQL Injection detection	http://testphp.vulnweb.com/listproducts.php?cat=1'	Detect SQL error in response	SQLi detected	Pass
TC_05	XSS Detection	http://testphp.vulnweb.com/search.php?q=alert(1)	XSS vulnerability identified	XSS detected	Pass
TC_06	Directory Traversal	http://testphp.vulnweb.com/page?file=../../etc/passwd	Directory traversal detected	Vulnerability detected	Pass
TC_07	Command Injection	http://testphp.vulnweb.com/cmd?input=;ls	Response reveals command execution	Detected	Pass
TC_08	Open Redirect Detection	http://example.com/redirect?url=http://malicious.com	Detect possible redirect behavior	Redirect warning	Pass
TC_09	Host Header Injection	Custom Host header value	Server responds with manipulated content	Detected	Pass

8.Screenshots



```
:\vuln_scanner_project>python app.py
 * Serving Flask app 'app'
 * Debug mode: on
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
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 213-536-770
```

Fig no : 8.1 Development Server Status



```
C:\vuln_scanner_project>python app.py
 * Serving Flask app 'app'
 * Debug mode: on
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
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 213-536-770
127.0.0.1 - - [20/Jun/2025 19:15:17] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [20/Jun/2025 19:15:18] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [20/Jun/2025 19:16:25] "POST /scan HTTP/1.1" 200 -
127.0.0.1 - - [20/Jun/2025 19:19:39] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [20/Jun/2025 19:19:48] "POST /scan HTTP/1.1" 200 -
127.0.0.1 - - [20/Jun/2025 19:20:04] "POST /scan HTTP/1.1" 200 -
127.0.0.1 - - [20/Jun/2025 19:20:39] "POST /scan HTTP/1.1" 200 -
```

Fig no : 8.2 Server Running locally on port 5000

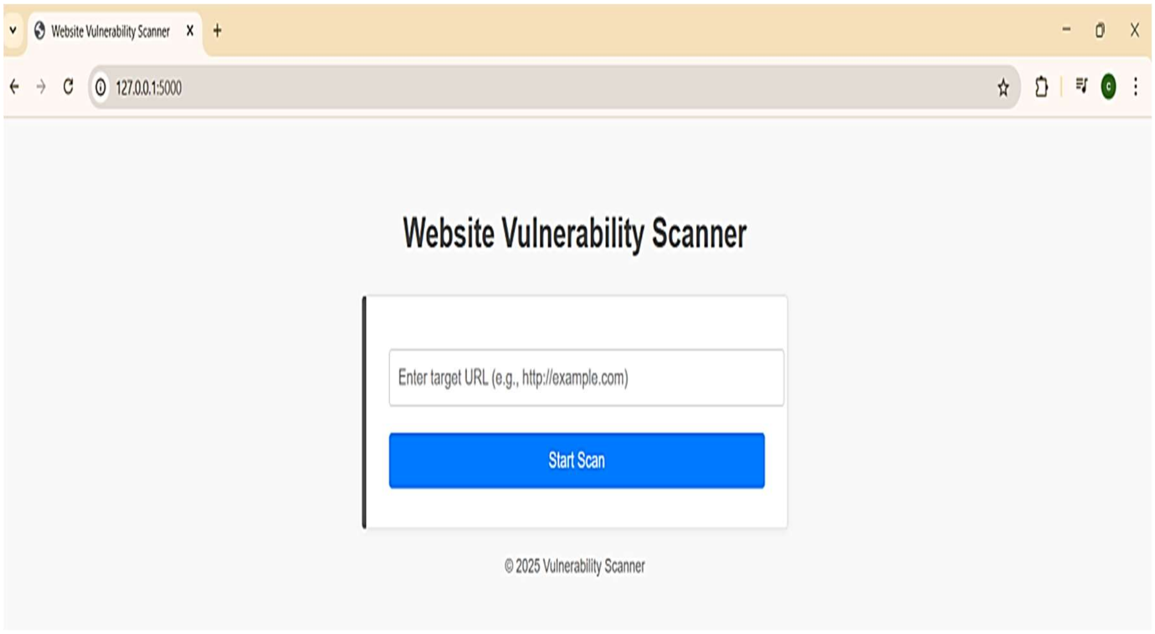

Website Vulnerability Scanner
127.0.0.1:5000
Website Vulnerability Scanner
Enter target URL (e.g., http://example.com)
Start Scan
© 2025 Vulnerability Scanner

Fig no: 8.3 Index Page

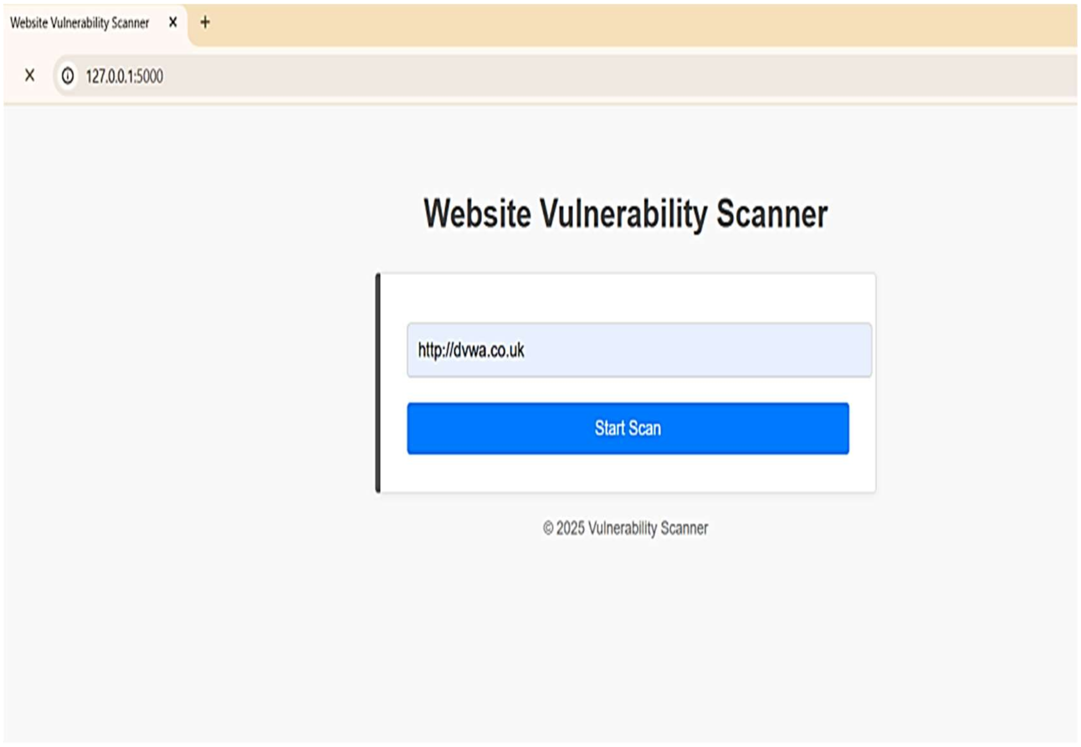
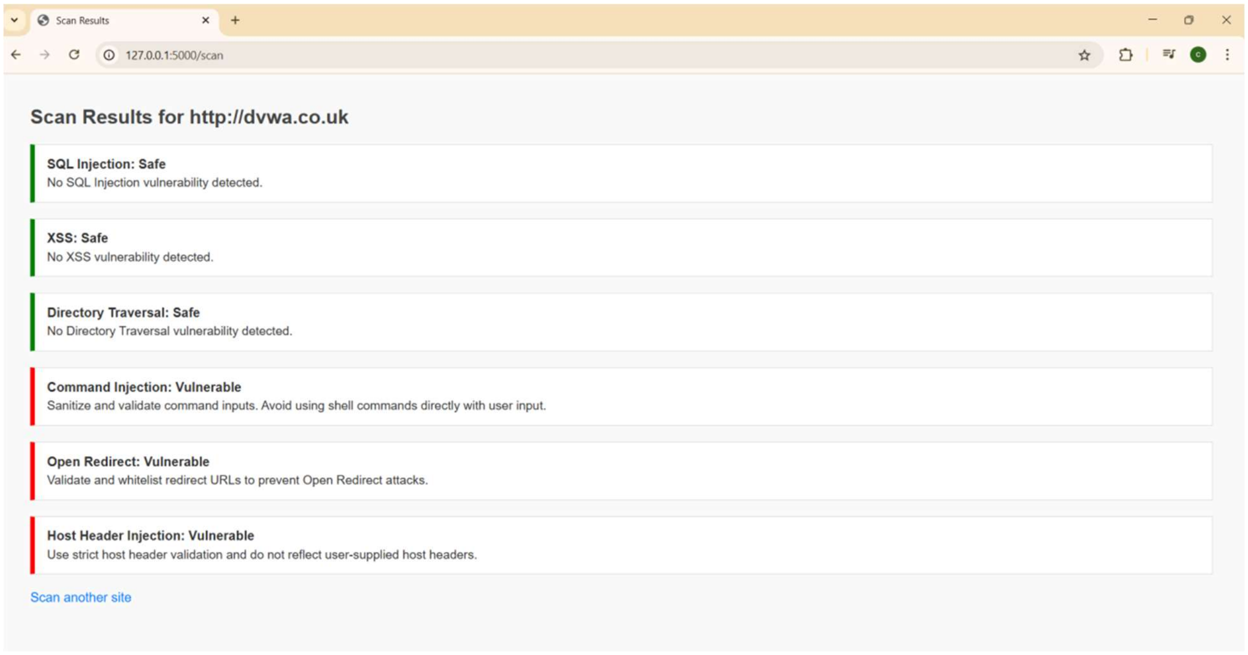

Website Vulnerability Scanner
127.0.0.1:5000
Website Vulnerability Scanner
http://dvwa.co.uk
Start Scan
© 2025 Vulnerability Scanner

Fig no : 8.4 Entering URL



Scan Results for http://dvwa.co.uk

SQL Injection: Safe
No SQL Injection vulnerability detected.

XSS: Safe
No XSS vulnerability detected.

Directory Traversal: Safe
No Directory Traversal vulnerability detected.

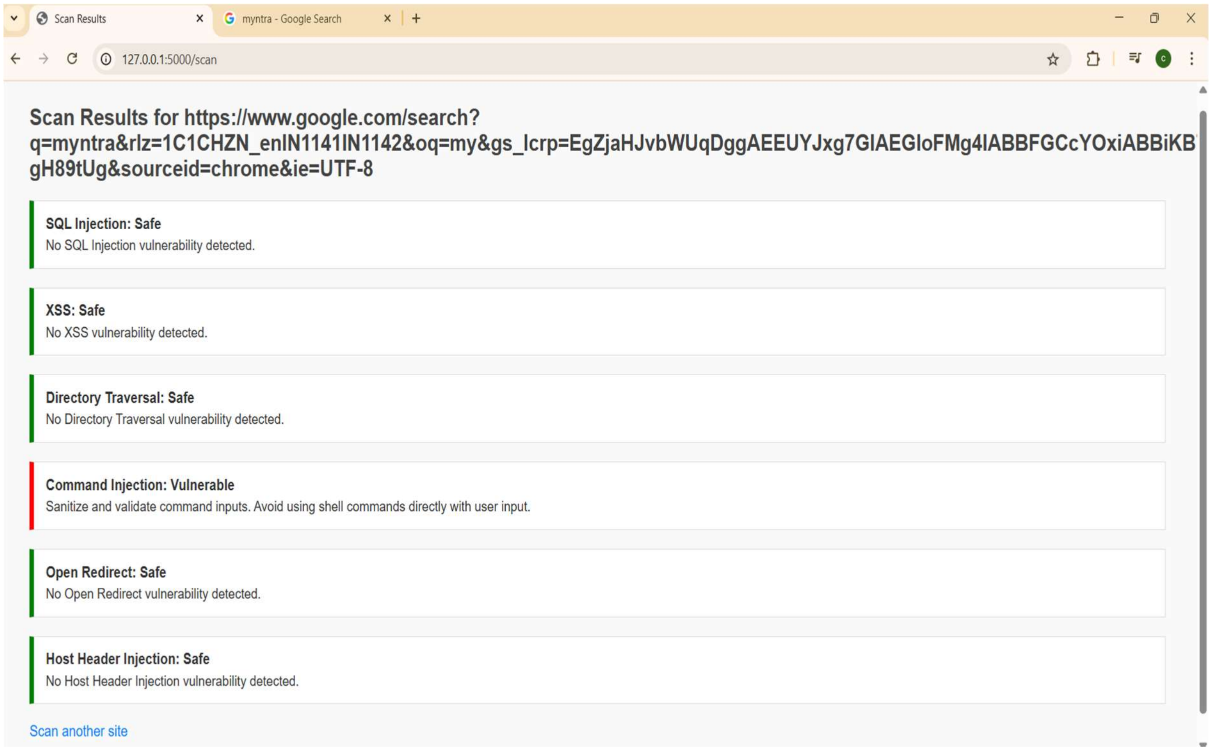
Command Injection: Vulnerable
Sanitize and validate command inputs. Avoid using shell commands directly with user input.

Open Redirect: Vulnerable
Validate and whitelist redirect URLs to prevent Open Redirect attacks.

Host Header Injection: Vulnerable
Use strict host header validation and do not reflect user-supplied host headers.

Scan another site

Fig No: 8.5 output 1



Scan Results for https://www.google.com/search?q=myntra&rlz=1C1CHZN_enIN1141IN1142&oq=my&gs_lcrp=EgZjaHJvbWUqDggAEEUYJxg7GIAEGIoFMg4IABBFGCcYOxiABBiKB gH89tUg&sourceid=chrome&ie=UTF-8

SQL Injection: Safe
No SQL Injection vulnerability detected.

XSS: Safe
No XSS vulnerability detected.

Directory Traversal: Safe
No Directory Traversal vulnerability detected.

Command Injection: Vulnerable
Sanitize and validate command inputs. Avoid using shell commands directly with user input.

Open Redirect: Safe
No Open Redirect vulnerability detected.

Host Header Injection: Safe
No Host Header Injection vulnerability detected.

Scan another site

Fig No: 8.6 output 2

9.CONCLUSION

The development of the Basic Website Vulnerability Scanner using Python and Flask has been successfully completed, fulfilling the objective of creating an educational, lightweight, and functional tool for detecting common web vulnerabilities. This project serves as a stepping stone for students, developers, and cybersecurity enthusiasts to understand the fundamental concepts of web application security and ethical hacking.

Through this project, a working prototype was implemented that allows users to scan target URLs for several widely known vulnerabilities including SQL Injection, Cross-Site Scripting (XSS), Directory Traversal, Command Injection, Open Redirect, and Host Header Injection.

The system was developed using open-source technologies Python, and basic HTML—which makes it accessible, cost-effective, and easy to maintain or extend. The use of modular code structure allows for future scalability.

Most importantly, the project emphasizes ethical usage by including warnings against scanning unauthorized websites and encouraging responsible behavior in security testing. This promotes a strong foundation in cyber security ethics among learners.

In conclusion, this project not only meets its technical and educational goals but also raises awareness about real-world security issues in web applications. It provides a valuable learning platform and can be further improved into a more advanced tool with features like authentication scanning, reporting logs, or integration with databases. Ultimately, the scanner serves as a small but meaningful contribution toward building more secure and resilient web applications in the digital age.

10.FUTURE SCOPE

The current Basic Website Vulnerability Scanner offers foundational features, but it can be enhanced significantly. Key areas for future development include:

More Vulnerability Types

Add detection for CSRF, SSRF, RCE, IDOR, Broken Authentication, and Clickjacking.

Deep Link Crawling

Implement automatic link discovery, input detection, and recursive scanning for better coverage.

Authentication Support

Enable scanning of login-protected areas by adding login form handling, session management, and credential-based access.

Advanced Reporting

Include PDF/CSV export, visual dashboards, vulnerability severity levels, and scan history tracking.

Database Integration

Store scan data, support user accounts, and track remediation progress over time.

These enhancements would make the tool more robust, user-friendly, and suitable for broader real-world and academic use.

11.BIBLIOGRAPHY

11.1 References

These are the academic and technical resources referred to during the development and writing of this project:

Stallings, William. Network Security Essentials: Applications and Standards. Pearson Education, 2016.

Kurose, James F., and Keith W. Ross. Computer Networking: A Top-Down Approach. Pearson, 2017.

Christou, George. Web Application Security: A Beginner's Guide. McGraw Hill, 2012.

OWASP Foundation. OWASP Testing Guide v4. The Open Web Application Security Project, 2014.

11.2 Websites

These websites were used as resources for tools, documentation, and vulnerability research:

https://owasp.org — OWASP: Open Web Application Security Project

https://www.acunetix.com/websitesecurity/cross-site-scripting/ - XSS

https://www.acunetix.com/websitesecurity/sql-injection/ - SQL Injection

https://www.w3schools.com — W3Schools: HTML, Python, Flask tutorials

https://flask.palletsprojects.com — Flask Official Documentation.