WEB VULNERABILITY SCANNER

A Project Report submitted in partial fulfillment of the requirement for the award of

Bachelor of Science

in

Computer Science

of Bharathiar University, Coimbatore-46.

by

DHANESH C

Reg. No: 2122K1326

Under the guidance of

Dr. Priya.R MCA., M.Phil., Ph.D.,

Associate Professor & Head

Department of Computer Science

SREE NARAYANA GURU COLLEGE

K.G.CHAVADI, COIMBATORE –641105.

(Affiliated to Bharathiar University)

(Re-Accredited by NAAC & An ISO 9001-2015 Certified Institution)

March - 2024

WEB VULNERABILITY SCANNER

Bonafide Work Done by

DHANESH C

Reg. No: 2122K1326

Under the guidance of

Dr. Priya.R MCA., M.Phil., Ph.D.,

Associate Professor & Head

Department of Computer Science

A project report submitted in partial fulfillment of the requirements for the award of

Bachelor of Science

in

Computer Science

of Bharathiar University

SREE NARAYANA GURU COLLEGE

(RE-ACCREDITED BY NAAC& AN ISO 9001-2015 CERTIFIED INSTITUTION)

(Affiliated to Bharathiar University)

K.G.Chavadi,Coimbatore-641105 March-2024

WEB VULNERABILITY SCANNER

Bonafide Work Done by

DHANESH C

Reg. No: 2122K1326

Under the guidance of

Dr. Priya.R MCA., M.Phil., Ph.D.,

Associate Professor & Head

Department of Computer Science

A project report submitted in partial fulfillment of the requirements for the award of

BACHELOR OF SCIENCE

in

COMPUTER SCIENCE

of Bharathiar University, Coimbatore-105.

Guide   **HOD**

Submitted for the viva voice examination held on ………………….

Internal Examiner   **External Examiner**

DECLARATION

        I, **DHANESH C** here by declaring that the project report entitled as "**WEB VULNERABILITY SCANNER"** done as the partial fulfillment of the requirement for The degree of Bachelor of Science in Computer Science is an independent project report done by me during the project duration of my period of study in **Sree Narayana Guru College**, K.G. Chavadi, Coimbatore under the guidance of **Dr. Priya.R MCA., M.Phil., Ph.D.,** Associate Professor & Head, Department of Computer Science.

Signature of the Guide   **Signature of the Student**

                                                                DHANESH C

CONTENTS   **Page No**

ACKNOWLEDGEMENT

 SYNOPSIS

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

First and foremost, I express my sincere thanks to our beloved and respected Principal **Dr. Kalpana.D, M.Com., B.Ed., M.Phil., Ph.D.,** Sree Narayana Guru College , K.G Chavadi ,Coimbatore.

I would like to this opportunity to express my sincere thanks to my project guide

Dr. Priya.R MCA., M.Phil., Ph.D., Associate Professor & Head Department of Computer Science, Sree Narayana Guru College, for valuable guidance, encouragement and helping me to do the project work.

I would like to express my heartfelt gratitude to all the Staff members of our department, Sree Narayana Guru College, K.G Chavadi, Coimbatore.

I would like to express my special thanks to my family and my friends who gave their assistance at each and every stage of this project.

SYNOPSIS

SYNOPSIS

The **"Web Vulnerability Scanner"** project is a Python-based tool designed to identify common vulnerabilities in web applications. With the increasing complexity of web technologies, vulnerabilities like Cross-Site Scripting (XSS) and SQL Injection pose significant risks to data security. This project aims to provide a solution for developers and security professionals to detect and mitigate these vulnerabilities effectively.

1. INTRODUCTION

The Web Vulnerability Scanner project aims to develop a tool using Python that can identify common vulnerabilities in web applications. These vulnerabilities include Cross-Site Scripting (XSS), SQL Injection and potentially others. The purpose of this project is to provide a basic but functional scanner that can help developers and security professionals identify and mitigate vulnerabilities in their web applications.

1.1 ORGANIZATION PROFILE

Sree Narayana Guru College

With a view of imparting quality education for the aspiring youth of Coimbatore, Sree Narayana Guru Educational Trust members came forward with a plan to start a college for Applied Sciences and Management Studies. Sree Narayana Guru College (SNGC) came into existence in 1994, Affiliated to Bharathiar University, Accredited by NAAC and Approved by Govt of Tamilnadu.

Sree Narayana Guru College strives to emerge as a premiere institute of international standards promoting excellence and equity in higher education. To mold the students of the institution through a socially committed, intellectually inclined, value based and culture driven paradigms of learning using the state-of-the-art educational technologies.

➢ To enhance the learning capacity and knowledge of the students by imparting quality education of national and international standards.

➢ To make the students think critically, objectively, creatively and to be life-long learners, engaged leaders and productive citizens.

➢ To motivate the students to pursue research, to advance knowledge and to address the state, national and global challenges.

➢ To mold their character and develop a value system so as to manifest oneness among students of diverse socio-cultural and economic backgrounds.

➢ To identify the inherent talents of the students and provide a platform to exhibit them.

1.2   SYSTEM SPECIFICATION

1.2.1   HARDWARE CONFIGURATION:

Processor	:	AMD Ryzen 3 3200G with Radeon Vega Graphics.
RAM	:	8GB DDR4.
SSD	:	240GB.
Key Board	:	104 keys.

1.2.2   SOFTWARE SPECIFICATION:

Operating system	:	Ubuntu 23.04.
Front End Software	:	Python 3.
Back End Software	:	MySQL

SOFTWARE FEATURES

About Python 3

Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. Guido van Rossum initially created Python in the late 1980s, and its first public release occurred in 1991. Since then, Python has evolved into one of the most popular programming languages globally, with a vast and active community of developers contributing to its growth and ecosystem.

Key features and characteristics of Python include:

➢ **Readable and Simple Syntax**: Python emphasizes readability and simplicity, with a clean

and straightforward syntax that reduces the cost of program maintenance and debugging. Its code is often described as being close to natural language, making it accessible to beginners and experienced developers alike.

➢ **Interpreted and Interactive**: Python is an interpreted language, meaning that code is

executed line by line by the Python interpreter without the need for compilation. This allows for rapid development and prototyping, as developers can quickly test and execute code interactively in a Python shell or REPL (Read-Eval-Print Loop).

➢ **High-Level Language**: Python abstracts away low-level details, enabling developers to

focus on solving problems rather than managing system-level operations. It provides built-in data structures and high-level abstractions for common tasks, such as lists, dictionaries and file I/O operations.

➢ **Dynamic Typing and Strong Typing**: Python is dynamically typed, meaning that variable

types are determined at runtime and can change throughout the program's execution. However, Python is also strongly typed, enforcing type safety and preventing certain types of errors at runtime.

➢ **Extensive Standard Library**: Python comes with a comprehensive standard library that

provides modules and packages for performing a wide range of tasks, including file I/O, networking, web development, data processing and more. This rich library ecosystem simplifies development by providing pre-built solutions for common programming tasks.

➢ **Cross-Platform Compatibility**: Python is available on multiple platforms, including

Windows, macOS, Linux, and Unix-like operating systems, making it highly portable and suitable for developing cross-platform applications.

➢ **Large Ecosystem of Third-Party Libraries**: Python boasts a vast ecosystem of

third-party libraries and frameworks that extend its capabilities for various domains such as web development (Django, Flask), data science (NumPy, Pandas), machine learning (scikit-learn, TensorFlow), and more. These libraries enhance productivity and enable developers to leverage existing solutions to build complex applications efficiently.

➢ **Community Support**: Python has a vibrant and inclusive community of developers,

enthusiasts, and contributors who actively participate in discussions, contribute to open-source projects, organize conferences and events, and provide support through forums, mailing lists and online communities.

About MySQL

MySQL is an open-source relational database management system (RDBMS) that is widely used for building and managing databases in various applications and websites. Developed by MySQL AB, it was later acquired by Sun Microsystems and subsequently by Oracle Corporation. MySQL is named after its co-founder, Michael Widenius's daughter, My.

Key features of MySQL include:

➢ **Open Source**: MySQL is available as open-source software under the GNU General

Public License (GPL), allowing users to freely download, use, modify, and distribute it. This

open-source nature has contributed to its widespread adoption and extensive community support.

➢ **Relational Database**: MySQL is a relational database management system, meaning it organizes data into tables consisting of rows and columns, following the principles of the relational model. It supports SQL (Structured Query Language) for querying and manipulating data stored in these tables.

➢ **Cross-Platform Compatibility**: MySQL is compatible with various operating systems, including Linux, Windows, macOS, and Unix-like systems, making it highly versatile and suitable for deployment in diverse environments.

➢ **Scalability and Performance**: MySQL is known for its scalability and performance, capable of handling large datasets and high-traffic websites efficiently. It offers features like replication, clustering, partitioning, and caching to optimize performance and accommodate growing workloads.

➢ **High Availability and Reliability**: MySQL provides features such as replication, automatic failover, backup and recovery mechanisms, and transaction support to ensure high availability and reliability of data stored in databases.

➢ **Security**: MySQL offers robust security features to protect data from unauthorized access, including user authentication, access control, data encryption, and SSL/TLS support. It also supports auditing and logging functionalities to monitor database activities.

➢ **Community and Enterprise Editions**: MySQL is available in both community and enterprise editions. The community edition is free to use and is supported by the open-source community, while the enterprise edition includes additional features, support, and services tailored for enterprise deployments.

➢ **Extensive Ecosystem**: MySQL has a vast ecosystem of tools, utilities, connectors, and extensions that enhance its functionality and integration with other technologies. This ecosystem

includes management tools, graphical user interfaces (GUIs), connectors for various programming languages, and third-party plugins.

UBUNTU

Ubuntu is a popular open-source Linux distribution based on Debian. It is developed and maintained by Canonical Ltd. and was first released in October 2004. Ubuntu is named after the African philosophy of ubuntu, which emphasizes community, sharing, and interconnectedness.

Key features and characteristics of Ubuntu include:

➢ **Free and Open Source**: Ubuntu is distributed as free and open-source software under the

GNU General Public License (GPL). Users are free to download, use, modify, and distribute Ubuntu without any licensing fees.

➢ **User-Friendly Interface**: Ubuntu provides a user-friendly desktop environment with a

graphical user interface (GUI), making it accessible to users who may not be familiar with Linux or command-line interfaces. It includes features such as the GNOME desktop environment, the Ubuntu Software Center for installing applications, and a range of pre-installed software packages.

➢ **Stable and Secure**: Ubuntu is known for its stability and security, with regular updates and

security patches provided by Canonical. It benefits from the robust security features inherent in Linux, including user permissions, access control, and built-in firewalls.

➢ **Package Management**: Ubuntu uses the Advanced Package Tool (APT) for package

management, allowing users to easily install, update, and remove software packages from the Ubuntu repositories. It also supports the Snap package format for distributing and installing software packages with dependencies bundled together.

➢ **Wide Hardware Support**: Ubuntu provides broad hardware compatibility, supporting a

wide range of hardware architectures and devices. It includes drivers for common hardware
components and peripherals, ensuring that Ubuntu can be installed and used on a variety of
systems.

➢ **Community Support**: Ubuntu benefits from a large and active community of users,

developers, and contributors who provide support, share knowledge, and collaborate on the
development of the operating system. Community forums, mailing lists, and online resources are
available for users to seek assistance and contribute to the Ubuntu ecosystem.

➢ **Customization and Flexibility**: Ubuntu offers flexibility and customization options,

allowing users to tailor their desktop environments, install additional software packages, and
configure system settings according to their preferences. Users can choose from various desktop
environments (e.g., GNOME, KDE, Xfce) and customize their Ubuntu experience accordingly.

SYSTEM STUDY

2.   SYSTEM STUDY

2.1   EXISTING SYSTEM

The existing system for scanning web vulnerabilities exhibits several drawbacks. Firstly, it relies on manual processes, necessitating human intervention for identifying and assessing vulnerabilities in web applications, which can introduce errors and inconsistencies. Moreover, the manual nature of the system contributes to its time-consuming nature, resulting in significant delays in completing scans and addressing security issues promptly. Additionally, the system lacks interactivity, impeding user engagement and feedback during the scanning process, thereby potentially reducing its effectiveness. Furthermore, the high costs associated with the manual system, including expenses related to manpower and specialized tools, make it financially burdensome for organizations to maintain.

2.1.1   DRAWBACKS

The drawbacks of the existing manual system for scanning web vulnerabilities, as described in the provided text, include:

➢   **Human Error**: Manual processes are prone to human error, leading to mistakes in

identifying or assessing vulnerabilities. This can result in false positives or negatives, compromising the effectiveness of the scanning process.

➢   **Time-Consuming**: Manual scanning is time-consuming as human operators need to

inspect each aspect of the web application individually. This leads to delays in identifying and addressing vulnerabilities, leaving the system exposed to potential threats for longer periods.

➢   **Limited Scalability**: Manual scanning may not be scalable, especially for large or

complex web applications. As the size and complexity of the application increases, the manual effort required for scanning also increases, making it difficult to scale the process effectively.

➢   **Inconsistency**: Different individuals may apply different methods or criteria for scanning

web vulnerabilities, leading to inconsistencies in the results. This lack of standardization makes it challenging to accurately assess the security posture of the web application.

➢ **Resource Intensive**: Manual scanning can be resource-intensive, requiring dedicated personnel and time to perform the scans. This can result in increased operational costs and may not be feasible for organizations with limited resources.

➢ **Limited Coverage**: Manual scanning may not cover all aspects of the web application comprehensively. Certain vulnerabilities or areas of the application may be overlooked due to human limitations or oversight, leaving potential security gaps.

➢ **Dependency on Expertise**: Effective manual scanning requires expertise in web security and vulnerability assessment. Organizations may struggle to find and retain skilled personnel to perform the scans, leading to gaps in security coverage.

2.2   PROPOSED SYSTEM

This system tends to replace the existing manual system for the scanning process which is time consuming, less interactive and highly expensive. The main features of this system will be creating reports and finding various types of vulnerabilities, storing Scanning data, process initiation and after that it generates a report of whole scanned websites.

2.2.1   FEATURES

The proposed Web Vulnerabilities Scanner (WVS) aims to overcome the limitations of the existing manual system for scanning web vulnerabilities. Here are the key aspects and advantages of the proposed system:

➢   **Automation**: The WVS automates the scanning process, reducing reliance on manual

effort. This automation improves efficiency and accuracy in identifying vulnerabilities across web applications.

➢   **Efficiency**: By automating the scanning process, the WVS significantly reduces the time

required to scan web applications for vulnerabilities. This efficiency allows for quicker detection and remediation of security issues, enhancing the overall security posture of the applications.

➢   **Comprehensive Coverage**: The WVS is designed to provide comprehensive coverage by

scanning for a wide range of vulnerabilities. It can identify various types of security flaws, including input injection attacks, cross-site scripting (XSS), SQL injection, and more.

➢   **User-Friendly Interface**: The system features a user-friendly interface that simplifies the

scanning process and allows users to easily navigate through the application. This intuitive interface makes it accessible to users with varying levels of technical expertise.

➢   **Reporting Capabilities**: The WVS generates detailed reports that summarize the findings

of the scanning process. These reports provide actionable insights into detected vulnerabilities,

enabling users to prioritize and address security issues effectively.

➢ **Scalability**: The WVS is scalable and can accommodate the scanning needs of both small and large web applications. It can handle multiple scans simultaneously, allowing organizations to scale their security efforts as their applications grow.

➢ **Cost-Effectiveness**: By automating the scanning process and reducing the need for manual intervention, the WVS offers cost savings for organizations. It minimizes the resources required for scanning, including time, manpower, and specialized tools.

➢ **Continuous Monitoring**: The WVS supports continuous monitoring of web applications, allowing organizations to regularly assess their security posture and detect vulnerabilities in real-time. This proactive approach helps prevent security breaches and minimize potential risks.

SYSTEM DESIGN AND DEVELOPMENT

3. SYSTEM DESIGN AND DEVELOPMENT

System design involves translation of information, requirements and conceptual design into technical specification and general flow of processing. After the requirements are identified, related information is gathered to verify the problem and after evaluating the existing system, a new system is proposed. The proposed system consists of various tables, their maintenance and report generation.

The proposed system is designed in such a way that it will compensate for all the deficiencies of the existing system and to regenerate the system with optimum efficiency. The software is developed in user friendly mode. The software also generates reports of permanent. It can store large amounts of data by occupying very less space.

3.1   FILE DESIGN

In a web vulnerability scanner, file design refers to the organization and structure of the files and directories that make up the scanner's codebase. A well-designed file structure helps in managing the code, improving maintainability, and enhancing collaboration among developers.

3.2   INPUT DESIGN

Input design for a web vulnerability scanner involves defining how users input URLs, configurations, and other parameters required for scanning. Here's a breakdown of input design considerations:

URL Input:

Users should be able to input URLs of web applications to be scanned. Considerations include:

➢   **Single URL Input**: Provide a text field where users can input a single URL.

➢   **Batch URL Input**: Allow users to input multiple URLs either by typing or uploading a file containing a list of URLs.

➢   **Validation**: Implement validation to ensure that URLs are properly formatted and accessible.

User Interface:

Ensure that the input interface is user-friendly and intuitive. Considerations include:

➢   **Clear Labels and Instructions**: Provide clear labels and instructions to guide users in inputting the required information.

➢   **Error Handling**: Implement error messages and validation to inform users of any input errors and how to correct them.

➢   **Feedback**: Provide feedback to users during the scanning process to indicate progress and any issues encountered.

Security:

Implement security measures to protect sensitive input data and prevent misuse. Considerations include:

➢ **Secure Transmission**: Use HTTPS to encrypt data transmission between the user's browser and the scanner.

➢ **Data Sanitization**: Sanitize user input to prevent injection attacks.

➢ **Access Control**: Implement authentication and authorization mechanisms to control access to the scanner and its input features.

3.3   OUTPUT DESIGN

Output design for a web vulnerability scanner involves defining how the results of the scanning process are presented to users. Here are some considerations for designing the output of a web vulnerability scanner:

Vulnerability Report:

The primary output of the scanner is a comprehensive report detailing the vulnerabilities detected during the scanning process. Considerations include:

➢ **Format**: Decide on the format of the report (e.g., HTML, JSON, PDF) based on user preferences and ease of consumption.

➢ **Structure**: Organize the report into sections for easy navigation, such as by vulnerability type or severity.

➢ **Details**: Include detailed information about each vulnerability, including the affected URL, description, severity level, and recommended mitigation steps.

➢ **Evidence**: Provide evidence of each vulnerability, such as screenshots or HTTP request/response data, to help users understand the issue.

➢ **Summary**: Include a summary section highlighting key statistics, such as the total number of vulnerabilities detected and their distribution by severity.

User Interface:

Ensure that the output interface is user-friendly and intuitive. Considerations include:

➢ **Clear Labels and Instructions**: Provide clear labels and instructions to guide users in interpreting the vulnerability report.

➢ **Interactive Elements**: Implement interactive elements, such as collapsible sections or filters, to allow users to explore the report in more detail.

➢ **Feedback**: Provide feedback to users during the report generation process to indicate

progress and any issues encountered.

Security:

Implement security measures to protect sensitive information included in the vulnerability report. Considerations include:

➢ **Access Control**: Ensure that only authorized users have access to view or download the

vulnerability report.

➢ **Data Encryption**: Use encryption to protect the vulnerability report during transmission

and storage.

➢ **Data Sanitization**: Sanitize sensitive information included in the report, such as

authentication credentials or sensitive URLs.

3.4   DATABASE DESIGN

The activity deals with the design of the database. A key is to determine how the access paths are to be implemented. A physical path is derived from a logical path. The general theme behind databases is to handle information as a whole. A database is a collection of interrelated data stored with minimum redundancy to serve many users quickly and efficiently.

Database design is the most critical part of the design phase. An elegantly designed, well-defined database is a strong foundation for the whole system. The general objective is to make information access easy, quick, inexpensive and flexible for the user Files in a relational database are called tables. Columns of tables represent data and rows represent the records in conventional technology. During database design, the following objectives are concerned:-

➢   Controlled redundancy.

➢   Easy to learn and use.

➢   More information and low cost.

➢   Accuracy.

➢   Integrity.

.

3.5 SYSTEM DEVELOPMENT

System development for a web vulnerability scanner involves several stages, including requirements gathering, design, implementation, testing, deployment, and maintenance. Here's a high-level overview of each stage:

Requirements Gathering:

➢　　Define the objectives and scope of the web vulnerability scanner.

➢　　Gather requirements from stakeholders, including users, developers, and security experts.

➢　　Identify the types of vulnerabilities to be scanned, supported platforms, and desired

features.

Design:

➢　　Define the system architecture, including components, modules, and their interactions.

➢　　Design the database schema to store scan results, configurations, and other relevant data.

➢　　Create wireframes or mockups to visualize the user interface and user experience.

Implementation:

➢　　Develop the backend of the scanner, including scanning algorithms, data processing logic,

and API endpoints.

➢　　Implement the frontend user interface for configuring scans, viewing results, and

generating reports.

➢　　Integrate with third-party libraries or services for functionalities such as authentication,

logging, and reporting.

Testing:

➢   Conduct unit tests to ensure the functionality of individual components.

➢   Perform integration tests to validate interactions between different modules.

➢   Conduct security testing to identify and address vulnerabilities in the scanner itself.

➢   Test the scanner against various web applications to verify its effectiveness and accuracy.

Deployment:

➢   Deploy the web vulnerability scanner on a secure server infrastructure.

➢   Configure monitoring and logging mechanisms to track system performance and detect
        issues.

➢   Perform user acceptance testing (UAT) to ensure that the scanner meets stakeholders'
        requirements.

Maintenance:

➢   Provide ongoing support and maintenance for the web vulnerability scanner.

➢   Release updates and patches to address security vulnerabilities and add new features.

➢   Monitor user feedback and performance metrics to identify areas for improvement.

➢   Continuously update the scanner's vulnerability database to detect emerging threats.

TESTING AND IMPLEMENTATION

4.   TESTING AND IMPLEMENTATION

SYSTEM IMPLEMENTATION

The implementation phase is less creative than system design. It is primarily concerned with user training site preparation and file conversation. Implementation is the stage where the theoretical design is turned into a working system. The most crucial stage in achieving a new system is that it will work efficiently and effectively after implementation. The system can be implemented only after thorough several testing and should work according to the user's specification.

The user training is a very important aspect in developing a project. The software developer gives the details and the methods to use the software. The user should be provided with enough details for the successful operation and maintenance of the system.

The user manual should contain the details and description of the system, which is essential for the proper operation. In addition, the system should accommodate future enhancements and also the error handling techniques should be perfect.

During the final testing the user acceptance is tested and followed by user training depending on the nature of the system extensive user training may be required. This is the final stage of the system development.

SYSTEM TESTING

System testing focuses verification effort on the smallest unit of software design of the module. Using the detailed design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and the errors detected as a result is limited by the constrained scope established for unit testing.   The unit is always white box- oriented, and the step can be conducted in parallel for multiple modules. System testing is normally considered an adjunct to the coding step. After source level code has been developed, reviewed, and verified for correct syntax, unit test case design begins.

A review of design information provides guidance for establishing test cases that are likely to uncover errors in each of the categories discussed above. Each test case should be coupled with a set of expected results.

Because a module is not a stand-alone program, driver and / or stub software must be developed for each system test. In most applications a driver is nothing more than a "main program" that accepts test case data, passes such data to the module, and prints the relevant results. Stubs serve to replace modules that are subordinates (called by) the module to be tested.

WHITE BOX TESTING

In this approach of testing, tests are made with complete knowledge about the internal working of the product. Tests can be conducted to ensure that all the internal operations of the product perform according to the specifications and all the internal components have been adequately tested. Check points at various places could be placed, so that checks are made to see if the status corresponds to the actual status.

Test cases designed through this approach ensure that all independent paths within a module have been exercised at least once, executing all logical decisions depending on their true or false values.

➢ Execute all loops well within their bounds

➢ Exercise internal data structure to ensure their validity etc.

BLACK BOX TESTING

In this approach to testing, test cases are prepared with the knowledge about the function that the product is designed to perform. Tests can be conducted which demonstrates the complete functionality of the product. Tests are done to check if the product is fully operational rather than checking each loop are conditions of the function. At the software level of testing, this approach revolves around testing the software interface. This approach has very little regard for the internal logical structure of the software. This approach saves time and energy.

Although this approach of testing is designed to uncover errors, they demonstrate that the functions are fully operational. They ensure that the inputs are properly accepted and desired output is obtained. They also provide for maintaining the complete integrity of the external input.

The attributes of both black and white box testing can be combined to provide an approach that validates the software interfaces and selectively ensures that the internal workings of the software are correct.

UNIT TESTING

Unit testing focuses verification effort on the smallest unit of software design, module. This testing was carried out during the coding itself. In this testing step, each module is found to be working satisfactorily as per the expected output from the module.

SYSTEM TESTING

The next level of testing is system testing where the system analyst tests all of the components to see that they interact correctly when combined as a system.   A series of testing are performed for the newly developed system before the system is ready for user acceptance testing.

VALIDATION TESTING

Validation succeeds when the software functions in a manner that is reasonably expected by the customer. Software validation is achieved through a series of black box test that demonstrate conformity with the requirements. Deviations or errors at this step are corrected.

OUTPUT TESTING

The output generated by the system under consideration is in the format required by the user and the information in the reports is accurate. It is possible to generate timely reports without any errors.

USER ACCEPTANCE TESTING

User acceptance of a system is a key factor for the success of any system. The system under consideration was tested for user acceptance by constantly keeping in touch with prospective system users at the time of developing and making changes wherever required.

QUALITY ASSURANCE

Software quality assurance involves the entire software development process monitoring and improving the process, making sure that any agreed-upon standards and ensuring that problems are found and dealt with. It is oriented to prevention.

Testing involves operation of a system or application under controlled conditions and evaluating the results. The controlled conditions should include both normal and abnormal conditions. Testing should intentionally attempt to make things go wrong to determine if things happen when they should not or things don't happen when they should. It is oriented to 'detection'.

Organizations vary considerably in how they assign responsibilities, quality assurance and testing. Sometimes they are the combined responsibility of one group or individual. Also common are project teams that include a mix of testers and developers who work closely together, with the overall quality assurance process monitored by project managers. It will depend on what best fits an organization's size and business structures.

SYSTEM MAINTENANCE

Maintenance involves the software industry captive, typing of system resources. It means restoring something to its original condition. Maintenance involves a wide range of activities including correcting, coding and design errors, updating documentation and test data, and upgrading user support. Maintenance was done after the successful implementation. Maintenance is continued till the product is re-engineered or deployed to another platform. Maintenance is also done based on fixing the problem reported, changing the interface with other software or hardware, enhancing the software.

CONCLUSION

5. CONCLUSION

The strengths of the project are that it is relatively fast and performed well on the tests, with very few errors. It can scan for all functions at once and output what it found in a clear way. It also has few false positives as the scans can be both performed manually and by using the application.The application is easy to use as there is only 1 input, the website address.

The limitations of the project is currently it can only do one web page at a time unless the list of websites to test is coded into the application to have the functions run multiple times and it does not have the complexity or volume as that of more popular tools or commercial based tools. It therefore can't be used in a commercial environment or in large websites with potentially hundreds of pages.

It also does not check for a wide variety of vulnerabilities, just the most common ones present as outlined in the OWASP Top 10.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] OWASP. OWASP Top 10 - 2021: The ten most critical web application security risks.

[2] OWASP. XSS filter evasion cheat sheet. <https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet>.

[3] OWASP. XSS prevention cheat sheet. <https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html>.

[4] thepythoncode.com. 2020. How to Build a XSS Vulnerability Scanner in Python - Python Code. [online] Available at: <https://www.thepythoncode.com/article/make-a-xss-Vulnerability-scanner-in-python>.

[5]thepythoncode.com. 2021. How to Build a SQL Injection Scanner in Python - Python Code. [online] Available at: <https://www.thepythoncode.com/article/sql-injection-vulnerability-detector-in-python>.

[6]ChatGPT : <https://chat.openai.com/>.

APPENDICES

A. DATA FLOW DIAGRAM
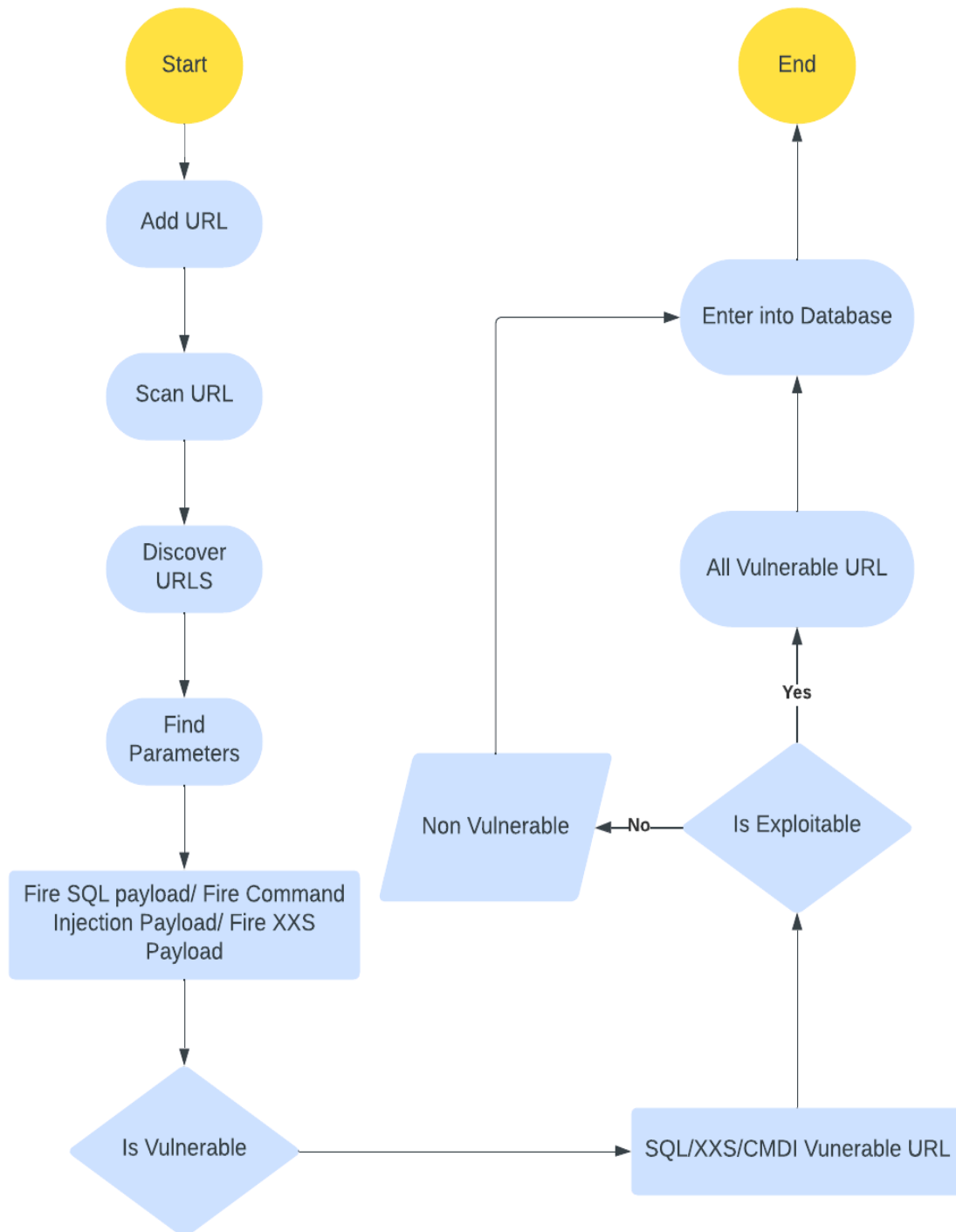
B. TABLE STRUCTURE

Database Table

NO	NAME	TYPE	SIZE	CONSTRAIN	DESCRIPTION
1	id	Int	20	Primary Key	Vulnerability Id
2	url	Text	255	Not Null	Vulnerable Url
3	vulnerability	Text	255	Not Null	Vulnerability Type
4	attack_method	Text	255	Not Null	Attack Method

C. SAMPLE CODING

```
import requests
import re
from bs4 import BeautifulSoup
from urllib.parse import urljoin
from cmdi import *
from sqli import *
from xxe import *


def scan_website(url):
    # Step 1: Discover URLs on the website
    discovered_urls = discover_urls(url)
    print(f"Discovered {len(discovered_urls)} URLs on {url}:\n")
    for i, discovered_url in enumerate(discovered_urls, start=1):
        print(f"{i}. {discovered_url}")
    # Step 2: Scan discovered URLs for vulnerabilities
    for page_url in discovered_urls:
        vulnerabilities = scan_url(page_url)
        if vulnerabilities:
            print(f"\nVulnerabilities found on {page_url}:")
            for vulnerability, attack_method in vulnerabilities.items():
                print(f"\nVulnerability: {vulnerability}")
                print(f"Attack Method: {attack_method}")
def discover_urls(url):
    discovered_urls = []
    # Send a GET request to the given URL
    response = requests.get(url)
    if response.status_code == 200:
        # Parse the HTML content of the response
        soup = BeautifulSoup(response.text, "html.parser")
        # Find all anchor tags and extract URLs
        for anchor_tag in soup.find_all("a"):
            href = anchor_tag.get("href")
```

```python
        if href:
            absolute_url = urljoin(url, href)
            # Filter out non-HTTP(S) URLs and other invalid URLs
            if absolute_url.startswith("http://") or absolute_url.startswith("https://"):
                discovered_urls.append(absolute_url)
    return discovered_urls
def scan_url(url):
    vulnerabilities = {}
    # Step 1: Perform vulnerability scans using a vulnerability scanner or custom checks
    # Example: Check for SQL injection vulnerability
    if is_sql_injection_vulnerable(url):
        vulnerabilities["SQL injection vulnerability"] = "Injecting SQL code into input fields"
    # Example: Check for cross-site scripting (XSS) vulnerability
    if is_xss_vulnerable(url):
        vulnerabilities["Cross-site scripting (XSS) vulnerability"] = "Injecting malicious scripts into input fields"
    #Example: Check for command injection vulnerability
    if is_cmdi_vulnerable(url):
        vulnerabilities["Command injection vulnerability"] = "Injecting commands into input fields"
    # Step 2: Perform additional vulnerability checks or manual code review
    # Example: Check for insecure server configuration
    if has_insecure_configuration(url):
        vulnerabilities["Insecure server configuration"] = "Exploiting insecure communication protocols"
    return vulnerabilities
def has_insecure_configuration(url):
    # Perform checks for insecure server configuration
    # Example: Check if the website uses HTTP instead of HTTPS
    if not url.startswith("https"):
        return True
    return False
```
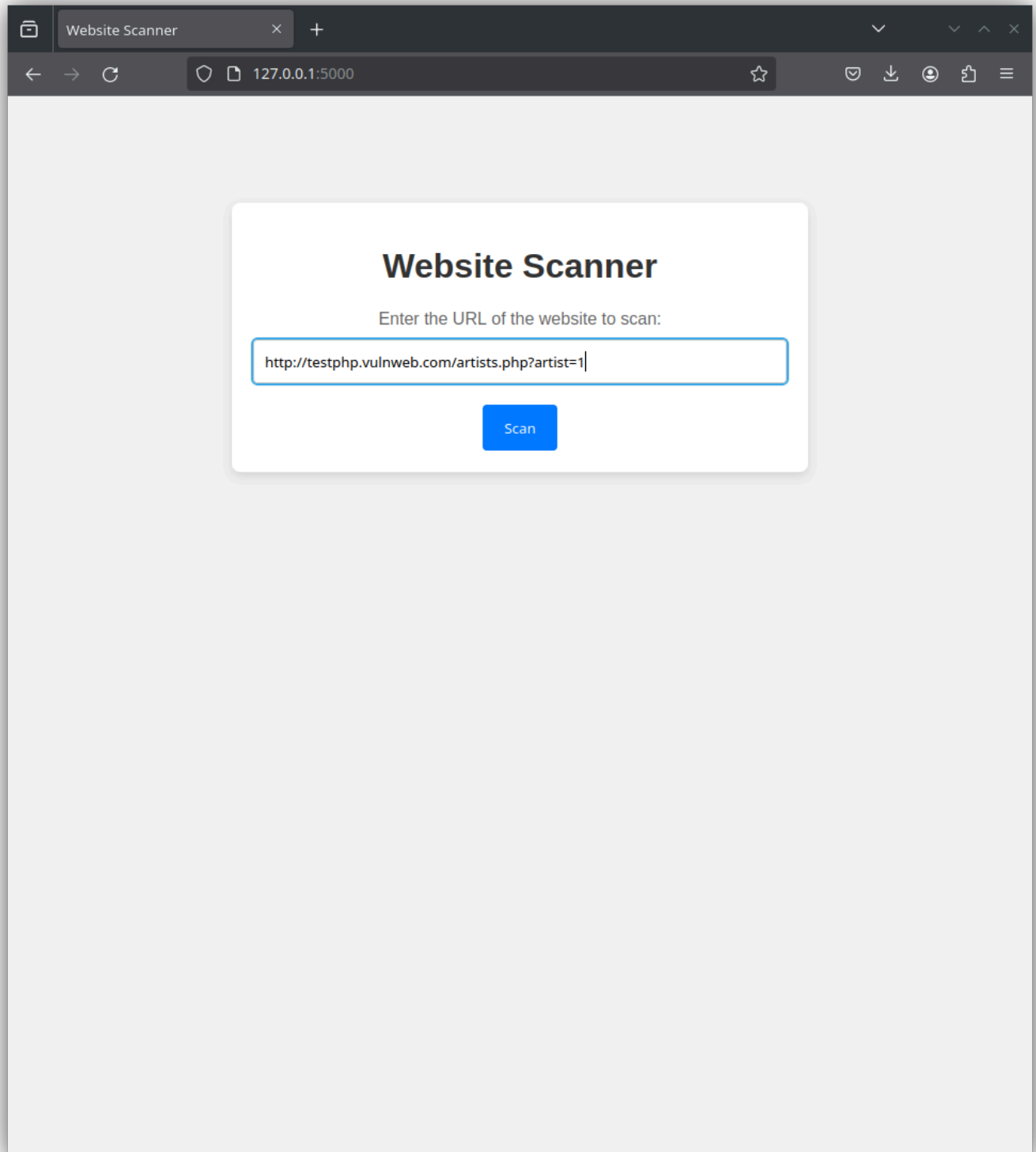
D. SAMPLE INPUT

SCANNER PAGE

E. SAMPLE OUTPUT

RESULT PAGE