**A Mini Project Report**

**On**

**A LIGHT WEIGHT VULNERABILITY SCANNER**

***Submitted to* JNTU HYDERABAD**

**BACHELOR OF TECHNOLOGY IN**

**CSE-CYBER SECURITY**

*Submitted By*

|  |  |
| --- | --- |
| **MARUPAKA VARDHAN (218R1A6241)** |  |
|  |  |
|  |  |

*Under the Esteemed guidance of*

**Mrs. M.AMRUTHA**

Assistant Professor, Department of CSE-Cyber Security

**Department CSE-Cyber Security**

**CMR ENGINEERING COLLEGE**

**(UGC AUTONOMOUS)**

*(Accredited by NAAC Approved by AICTE,NEW DELHI, Affiliated to JNTU Hyderabad)*

*Kandlakoya, Medchal Road, Medchal, R.R. Dist. Hyderabad-501 401)*

**(2024-2025)**

CMRENGINEERINGCOLLEGE

**UGC AUTONOMOUS**

*(AccreditedbyNAAC,ApprovedbyAICTENEWDELHI,AffiliatedtoJNTU,Hyderabad)*

*(Kandlakoya, Medchal Road, R.R. Dist, Hyderabad-501 401)*

Department of Computer Science & Engineering(Cyber Security)



# CERTIFICATE

This is to certify that the project entitled“**A LIGHT WEIGHT VULNERABILITY SCANNER ”** is a bonafide work carried out by

|  |  |
| --- | --- |
| **MARUPAKA VARDHAN** | **(218R1A6241)** |

In partial fulfillment of the requirement for the award of the degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING (CYBER-SECURITY)** from CMR Engineering College, affiliated to JNTU, Hyderabad, under our guidance and supervision. The results presented in this project have been verified and are found to be satisfactory. The results embodied in this project have not been submitted to any other university for the award of any other degree or diploma.

|  |  |  |
| --- | --- | --- |
| **InternalGuide** | **Miniproject** | **HeadoftheDepartment** |
|  | **Coordinator** |  |
| **M.AMRUTHA** | **Ms.A.Anusha** | **Dr.V.SathiyaSuntharam** |
| AssistantProfessor | AssistantProfessor | Professor&HOD |
| CSE-CyberSecurity,  CMREC | CSE-CyberSecurity,  CMREC | CSE-CyberSecurity,  CMREC |
|  |  |  |
|  |  |  |

**DECLARATION**

#### This is to certify that the work reported in the present project entitled “**A LIGHT WEIGHT VULNERABILITY SCANNER**” is a record of bonafidework done by us in the Department of Computer Science and Engineering (Cyber Security), CMR Engineering College, JNTU Hyderabad. The reports are based on the project work done entirely by us and not copied from any other source. Wesubmit our project for further development by any interested students who share similar interests to improve the project in the future. The results embodied in this project report have not been submitted to any other University or Institute for theaward of any degree or diploma to the best of our knowledge and belief.

|  |  |
| --- | --- |
| **MARUPAKA VARDHAN** | **(218R1A6241)** |
|  |  |

**ACKNOWLEDGMENT**

We are extremely grateful to **Dr.A.Srinivasula Reddy** , Principal and **Dr.V.SathiyaSuntharam,** HOD **Department of CSE-Cyber Security Department** for their constant support.

We are extremely thankful to **M.Amrutha,** Assistant Professor and Internal guide,Department of CSE(CS) ,for her constant guidance, encouragement and moral support throughout the project.

We will be failing induty if we do not acknowledge with grateful thanks to the authors of the references and other literatures referred in this project.

We thank **Ms.A.Anusha** Mini Project Coordinator for her constant support in carrying out the project activities and reviews.

We express our thanks to all staff members and friends for all the help and co-ordination extended in the bringing out this project successfully on time.

Finally,we are very much thankful to our parents who guided us every step of the way.

|  |  |
| --- | --- |
| **MARUPAKA VARDHAN** | **(218R1A6241)** |

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Title** | **Page No.** |
| **1** | **Introduction** | **7** |
| **2** | **Literature Survey** | **8-9** |
| **3** | **System Analysis** | **10-11** |
| **4** | **System Specification** | **12-16** |
| **5** | **System Implementation** | **17-19** |
| **6** | **System Testing** | **20-21** |
| **7** | **Source code** | **22-45** |
| **8** | **Outputs** | **46** |
| **9** | **Conclusion** | **47** |
| **10** | **Future Enhancement** | **48** |
| **11** | **Reference** | **49** |

**ABSTRACT**

A lightweight vulnerability scanner is a streamlined tool designed to efficiently identify security weaknesses in systems, networks, or applications while minimizing resource usage. Its primary function is to detect vulnerabilities without placing a significant burden on the system or the scanning infrastructure. The scanner operates by first accepting input, such as IP addresses or URLs, to define the scope of the scan. It then performs targeted scans to identify open ports, running services, or potential configuration weaknesses. For web applications, the scanner might check for vulnerabilities like SQL injection or cross-site scripting (XSS). Once potential vulnerabilities are identified, the scanner cross-references them with a database of known vulnerabilities, such as the Common Vulnerabilities and Exposures (CVE) database. Finally, the tool generates a concise report that outlines the vulnerabilities, assesses their severity, and provides recommendations for remediation. To remain lightweight, the scanner uses techniques such as selective scanning of high-priority areas, parallel operations to increase speed, and a modular design that leverages existing tools like Nmap or OpenVAS. This approach ensures fast and frequent assessments while keeping the system’s footprint low, making it suitable for routine scans in resource-constrained environments.

**CHAPTER-1**

**INTRODUCTION**

**1.Introduction :**

In today's rapidly evolving digital landscape, maintaining robust security is paramount. This project focuses on creating a Lightweight Vulnerability Scanner (LWVS) to efficiently identify and address security vulnerabilities in various systems. The LWVS is designed to operate with minimal resource consumption, providing a quick and user-friendly solution ideal for small to medium-sized enterprises (SMEs) and individual developers. By ensuring essential security checks, the LWVS aims to enhance system integrity while remaining cost- effective. Integration with more comprehensive tools will also be explored to provide a well-rounded security approach.

**CHAPTER-2**

**LITERATURE SURVEY**

The literature on vulnerability scanning highlights the critical role these tools play in cybersecurity. Traditional scanners, such as Nessus and Qualys, offer comprehensive vulnerability assessments but often require significant system resources, making them less suitable for smaller organizations or frequent use. Research by Wang et al. (2019) emphasizes the need for lightweight solutions that balance efficiency and thoroughness. Tools like OpenVAS and Nmap have been noted for their effectiveness in specific contexts, such as open-source environments and network scanning, respectively. However, their scope can be limited compared to full-scale solutions. Studies by Johnson and Miller (2020) reveal that lightweight scanners like Nikto provide essential web server security checks quickly but lack broader application coverage. Recent advancements focus on improving the speed and user-friendliness of scanners without compromising accuracy. The integration of lightweight scanners with comprehensive tools, as suggested by Chen et al. (2021), is a promising approach to achieving balanced security measures. This project aims to build on these insights, developing a Lightweight Vulnerability Scanner that is both efficient and comprehensive enough for varied security needs.

**CHAPTER-3**

**SYSTER ANALYSIS**

**3.1 Existing System :**

Existing vulnerability scanning systems, such as Nessus, Qualys, and OpenVAS, provide comprehensive security assessments but often require significant system resources, making them less suitable for smaller organizations or frequent use. Tools like Nmap and Nikto offer more focused scanning capabilities, such as network mapping and web server vulnerability detection, but may lack the breadth and depth of full-scale solutions. These systems are often complex to set up and manage, posing challenges for users with limited technical expertise. This project addresses the need for a more efficient, user-friendly, and resource-effective vulnerability scanner**.**

**3.2 Proposing System**

The proposed Lightweight Vulnerability Scanner (LWVS) system aims to deliver an efficient, user-friendly solution for identifying security vulnerabilities with minimal resource usage, making it ideal for SMEs and individual developers. Key features include a web-based dashboard and CLI for initiating and managing scans, a robust scan engine, an up-to-date signature database, and a comprehensive reporting module. The system supports various deployment options, including local, network, and cloud-based setups, ensuring scalability and flexibility. It offers real-time progress updates, detailed reports, and seamless integration with other security tools through an API, enhancing overall security measures. The LWVS is designed to be cost-effective, easy to use, and secure, with regular updates and maintenance to ensure continued efficacy and relevance in identifying emerging vulnerabilities.

**3.3Requirement Analysis :**

The project involved analyzing the design of a few applications so as to make the application more user-friendly. To do so, it was really important to keep the navigations from one screen to the other well-ordered and at the same time reduce the amount of typing the user needs to do. In order to make the application more accessible, the browser version had to be chosen so that it is compatible with most of the Browsers.

**3.4 Functional Requirements :**

The Functional requirements for a system describe the functionality or the servicesthatthesystemisexpectedtoprovide.Thesearethestatementsofservicesthesystemshould provide how the system should react to particular inputs and how thesystemshould behaveinparticularsituations.

**3.5Feasibility Study**

The feasibility of the project is analyzed in this phase and the business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis, the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

**3.5.1.Economical Feasibility**

This study is carried out to check the economic impact that the system will have on the organization. The amount of funds that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

### 3.5.2 TechnicalFeasibility

### This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

**3.5.3 Social Feasibility**

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

**CHAPTER-4**

**4. SystemSpecifications :**

**4.1 HardwareRequirements :**

**System:** Dual core CPU with 2.4 GHz clock speed.

**Hard Disk:**10 GB and more.

**Monitor**: Colour Monitor.

**Mouse:** Optical Mouse.

**Ram:** 1gb and above.

**4.2 SoftwareRequirements :**

**Operating system:** Windows 7 and above.

**Coding Language:** Python.

**Front-End:** Python.

**Designing:**CLI

**Data Base:** SQLite.

**4.3 System Design**

**4.3.1 Unified Modelling Language Diagrams :**

UML is a standard language for specifying, visualizing, constructing, and documenting the artefacts of software systems. UML was created by the Object Management Group (OMG) and the UML1.0 specification draft was proposed to the OMG in January 1997.

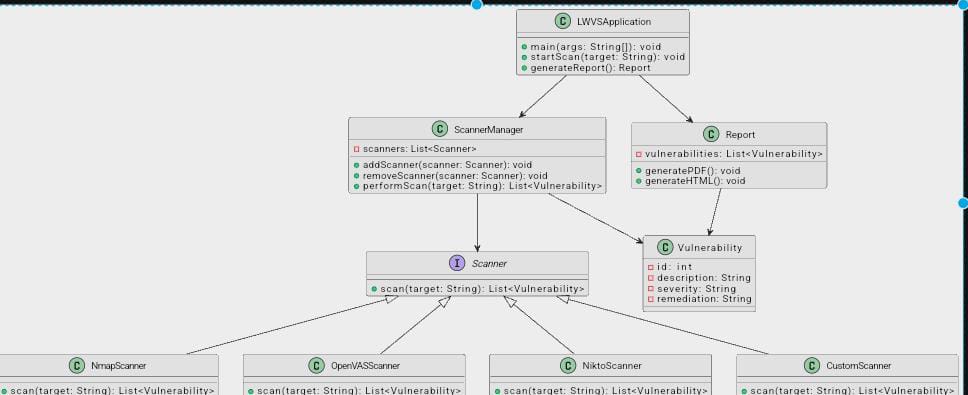
There are several types of UML diagrams and each one of them serves a different purposeregardless of whether it is being designed before the implementation or after (as part ofdocumentation). UML has a direct relation with object-oriented analysis and design. Aftersomestandardization, UML has becomeanOMGstandard.

The two broadest categories that compass all other types are:

1. **Behavioural UML diagram**
2. **Structural UML diagram.**

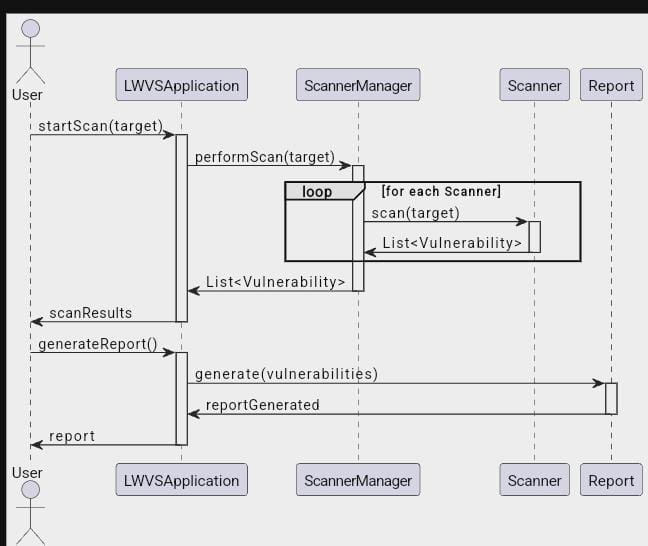
As it suggests, some UML diagrams try to analyze and depict the structure of a system or process, whereas others describe the behaviour of the system, its actors, and its building components.

**Class diagram :**



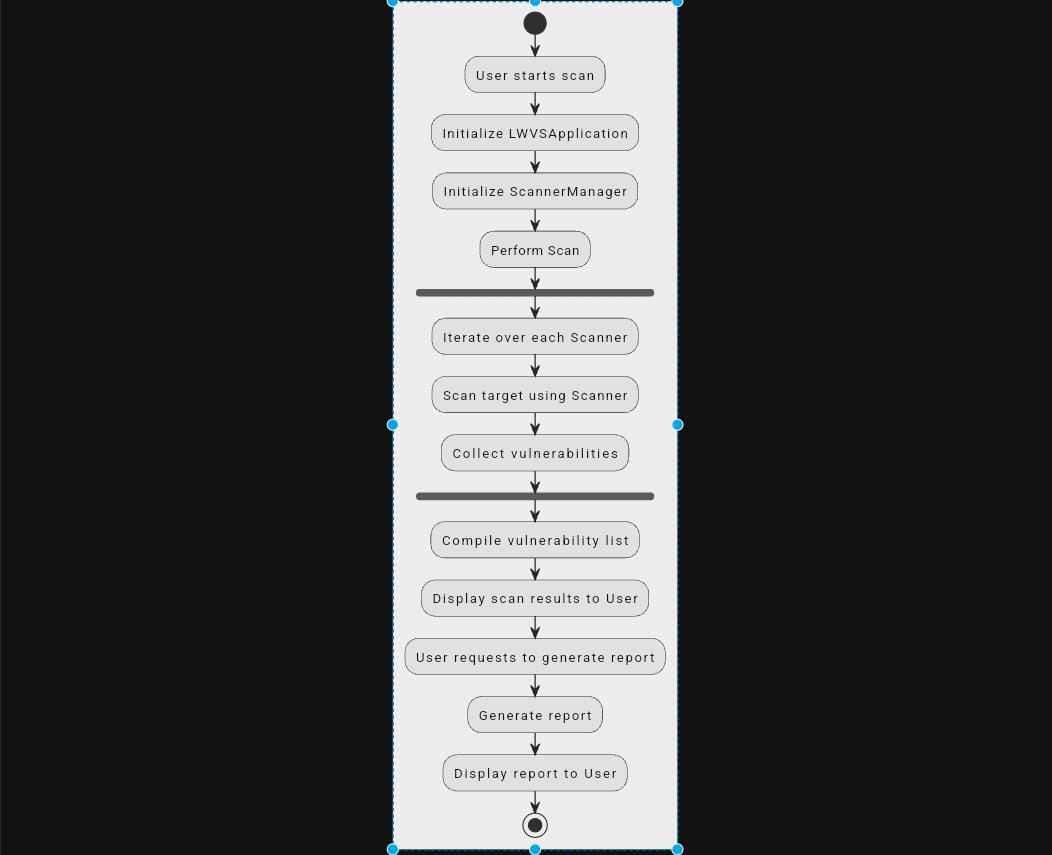
Class diagrams are a type of [UML](https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/)(Unified Modeling Language) diagram used in software engineering to visually represent the structure and relationships of classes in a system. UML is a standardized modeling language that helps in designing and documenting software systems. They are an integral part of the software development process, helping in both the design and documentation phases.

**SEQUENCE DIAGRAM :**



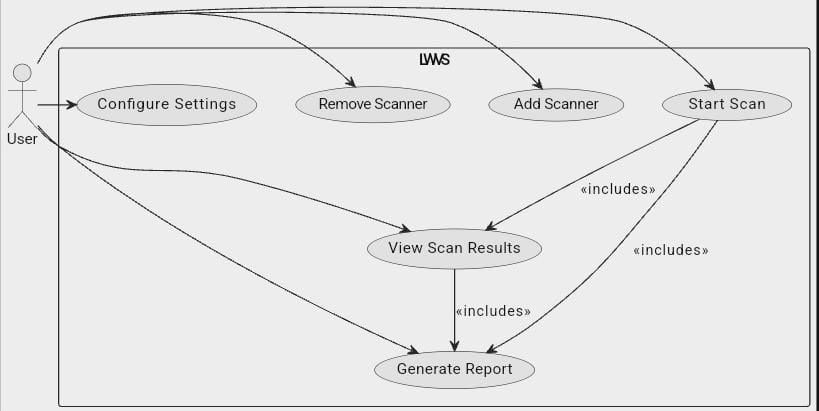
A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams

**ACTIVITYDIAGRAM :**

****

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

**USE CASE DIAGRAM :**



A use-case diagram in the Unified Modeling Language (UML) is a type of behavioural diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. The roles of the actors in the system can be depicted

**CHAPTER-5**

**System Implementation**

**5.1 Implementation :**

Implementing a lightweight vulnerability scanner involves creating a tool that can efficiently identify potential security vulnerabilities in a system without consuming too many resources. Here's a high-level approach for building such a system:

1. Define the Scope and Objectives

Target Environment: Decide whether the scanner will focus on web applications, networks, or system-level vulnerabilities.

Types of Vulnerabilities: Identify what types of vulnerabilities the scanner will target (e.g., OWASP Top 10 for web apps, port vulnerabilities for networks, or CVEs for systems).

2. Technology Stack

Programming Language: Python is an excellent choice due to its simplicity and rich set of libraries.

Libraries/Tools:

Nmap: For network scanning and detecting open ports.

OpenVAS: For vulnerability scanning.

SQLmap: For SQL injection testing.

Nikto: For web application vulnerability scanning.

Python Libraries: socket, scapy (for network analysis), requests, BeautifulSoup (for web application analysis), and subprocess (for integrating command-line tools).

Database: Store scan results (e.g., SQLite or PostgreSQL).

3. System Architecture

Input Module: Accepts user input such as the target IP, URL, or system.

Scan Module:

Port Scanning: Use Nmap or a custom-built solution with scapy to scan open ports.

Service Identification: Identify services running on open ports.

Vulnerability Scanning: Match identified services with known vulnerabilities using a CVE database or tools like OpenVAS.

Web Application Scanning: Perform checks for common vulnerabilities such as SQL Injection (using SQLmap), XSS, CSRF, etc.

Reporting Module: After scanning, generate a detailed report of vulnerabilities.

Results Management: Store the results in a database for future reference and further analysis.

4. Steps for Implementation

a. Port Scanning

You can integrate Nmap for port scanning using Python's subprocess module:

import subprocess

def run\_nmap(target\_ip):

result = subprocess.run(['nmap', '-sS', '-p-', target\_ip], stdout=subprocess.PIPE)

return result.stdout.decode('utf-8')

b. Vulnerability Identification

You can map services on open ports to known vulnerabilities by cross-referencing with a CVE database.

Example using requests to fetch CVEs:

import requests

def get\_cves(service\_name):

url = f"https://cve.circl.lu/api/search/{service\_name}"

response = requests.get(url)

if response.status\_code == 200:

return response.json()

return None

c. Web Application Vulnerability Scanning

Use existing tools like SQLmap or Nikto to check for web application vulnerabilities. These can also be integrated into Python using the subprocess module.

def run\_sqlmap(url):

result = subprocess.run(['sqlmap', '-u', url, '--batch'], stdout=subprocess.PIPE)

return result.stdout.decode('utf-8')

d. Results Reporting

After gathering scan results, store them in a database and generate a report. Example for generating a simple text-based report:

def generate\_report(scan\_results, target):

with open(f"{target}\_scan\_report.txt", 'w') as report\_file:

report\_file.write("Scan Results:\n")

for result in scan\_results:

report\_file.write(result + "\n")

e. Optimizing for Lightweight Operation

Parallel Scanning: Use Python’s concurrent.futures or asyncio to perform scanning operations concurrently, reducing the time taken.

Selective Scanning: Implement options for users to select specific types of scans (e.g., network, web app) to reduce overhead.

Resource Management: Limit the number of simultaneous scans to prevent overloading the system.

5. Integration with CVE Databases

Integrating a vulnerability database, such as the National Vulnerability Database (NVD) or CIRCL CVE, will allow your scanner to identify known vulnerabilities associated with the services and software it detects.

**5.2.3 The Project Structure**

**Client-Server Architecture:**

**Integration with OWASP ZAP:**

**Database Integration:**

**CHAPTER-6**

**SYSTEM TESTING**

**6.1. UNIT TESTING:**

Unit testing is like checking each piece of a puzzle to make sure it fits perfectly before putting the whole puzzle together. For a website vulnerability scanner, this means testing each small part of the scanner individually to make sure it works correctly.

* **Accuracy:** Helps make sure that the scanner finds vulnerabilities correctly and doesn’t miss any issues.
* **Reliability:** Ensures that the scanner works well and doesn’t break when changes are made.
* **Efficiency:** Makes it easier to find and fix problems early, before they become big issues.

**6.2. INTEGRATION TESTING:**

Integration testing checks how well different parts of a vulnerability scanner work together as a complete system.

* **To Ensure Everything Works Together:** Even if each part works fine alone, they need to function well together to be effective.
* **To Find Issues in Interaction:** This testing helps identify problems that occur when different parts of the scanner are used together.
* Integration testing ensures that all parts of the vulnerability scanner work together smoothly. It helps make sure the scanner finds and reports security issues accurately when used as a complete tool.

**6.3. COMPONENT TESTING :**

Component testing checks if each individual part of the vulnerability scanner works correctly on its own.

* **To Ensure Each Part Works Well:** Each component (like scanning, analyzing, or reporting parts) needs to function properly by itself.
* **To Catch Issues Early:** Finding and fixing problems in each part before they affect the whole system.
* Component testing focuses on making sure each individualpart of the vulnerability scanner works correctly by itself. It helps identify and fix issues in each component before they affect the entire scanner.

**6.4. FUNCTIONAL TESTING :**

Functional testing checks if the website vulnerability scanner does what it's supposed to do. It focuses on verifying that all the features and functions work correctly according to their requirements.

**To Ensure All Features Work:** Confirms that every function of the scanner performs as expected.

**To Validate Requirements:** Ensures the scanner meets the specified requirements and performs its tasks correctly.

Functional testing makes sure that the website vulnerability scanner performs all its intended tasks correctly. It checks each feature to ensure it works as expected and meets the requirements.

**CHAPTER-7**

**SOURCE CODE**

import requests

from bs4 import BeautifulSoup

# Function to check for SQL Injection vulnerability

def check\_sql\_injection(url):

sql\_payloads = ["'", '"', "' OR '1'='1", '" OR "1"="1']

vulnerable = False

for payload in sql\_payloads:

full\_url = f"{url}{payload}"

response = requests.get(full\_url)

if "error" in response.text.lower() or "sql" in response.text.lower():

vulnerable = True

print(f"Possible SQL Injection vulnerability detected with payload: {payload}")

return vulnerable

# Function to check for XSS vulnerability

def check\_xss(url):

xss\_payloads = ["<script>alert('XSS')</script>", '"><script>alert(1)</script>']

vulnerable = False

for payload in xss\_payloads:

full\_url = f"{url}{payload}"

response = requests.get(full\_url)

if payload in response.text:

vulnerable = True

print(f"Possible XSS vulnerability detected with payload: {payload}")

return vulnerable

# Function to parse forms and check for vulnerabilities

def scan\_forms(url):

response = requests.get(url)

soup = BeautifulSoup(response.text, 'html.parser')

forms = soup.find\_all('form')

for form in forms:

action = form.get('action')

method = form.get('method', 'get').lower()

inputs = form.find\_all('input')

form\_data = {}

for input in inputs:

input\_name = input.get('name')

input\_value = input.get('value', 'test')

form\_data[input\_name] = input\_value

if method == 'post':

response = requests.post(url + action, data=form\_data)

else:

response = requests.get(url + action, params=form\_data)

# Check for vulnerabilities in the form response

if "error" in response.text.lower() or "sql" in response.text.lower():

print(f"Possible SQL Injection vulnerability detected in form action: {action}")

if "<script>alert('XSS')</script>" in response.text:

print(f"Possible XSS vulnerability detected in form action: {action}")

# Main function to start the scan

def start\_scan(url):

print(f"Scanning URL: {url}")

if check\_sql\_injection(url):

print("SQL Injection vulnerability detected!")

else:

print("No SQL Injection vulnerability detected.")

if check\_xss(url):

print("XSS vulnerability detected!")

else:

print("No XSS vulnerability detected.")

scan\_forms(url)

print("Scan completed.")

# Example usage

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

target\_url = "http://example.com"

start\_scan(target\_url)

import requests

def check\_sql\_injection(url):

sql\_payload = "' OR '1'='1"

target\_url = f"{url}{sql\_payload}"

try:

response = requests.get(target\_url)

if "syntax error" in response.text or "mysql" in response.text:

return True

return False

except requests.exceptions.RequestException:

return False

def check\_xss(url):

xss\_payload = "<script>alert('XSS')</script>"

target\_url = f"{url}{xss\_payload}"

try:

response = requests.get(target\_url)

if xss\_payload in response.text:

return True

return False

except requests.exceptions.RequestException:

return False

from flask import Flask, render\_template, request

from scanner import check\_sql\_injection, check\_xss

app = Flask(\_name\_)

@app.route('/')

def index():

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),

}

return render\_template('results.html', url=url, results=results)

if \_name\_ == '\_main\_':

app.run(debug=True)

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Scan Results</title>

</head>

<body>

<h1>Scan Results for {{ url }}</h1>

<ul>

<li>SQL Injection: {{ 'Vulnerable' if results['sql\_injection'] else 'Not Vulnerable' }}</li>

<li>XSS: {{ 'Vulnerable' if results['xss'] else 'Not Vulnerable' }}</li>

</ul>

<a href="/">Scan another URL</a>

</body>

</html>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Vulnerability Scanner</title>

</head>

<body>

<h1>Lightweight Vulnerability Scanner</h1>

<form action="/scan" method="POST">

<label for="url">Enter URL:</label>

<input type="text" id="url" name="url" required>

<button type="submit">Scan</button>

</form>

</body>

</html>

**8. Outputs:**

**CHAPTER-9**

**CONCLUSION**

Applications :

This tools is very helpful for finding vulnerabilities present in the Website .

A web application scanner explores a web application by crawling through its web pages and examines it for security vulnerabilities, which involves generation of malicious inputs and evaluation of application's responses.

These scanners are automated tools that scan web applications to lookforsecurity vulnerabilities. They test web applications for common security problems such as cross-site scripting (XSS), SQL injection, and cross-site request forgery (CSRF).

This scanner uses different tools like nmap, dnswalk, dnsrecon, dnsenum, dnsmapetc in order to scan ports, sites, hosts and network to find vulnerabilites like OpenSSL CCS Injection, Slowloris, Denial of Service, etc.

**CHAPTER-10**

**FUTURE SCOPE**

**Future Enhancements:**

Real-Time Scanning:

Implement continuous monitoring for real-time detection of vulnerabilities.

Integration with CI/CD Pipelines:

Integrate with CI/CD pipelines for automated security testing during development.

Advanced Reporting:

Enhance reporting capabilities with visualizations and trend analysis.

Support for Additional Vulnerabilities:

Expand scanning capabilities to detect emerging vulnerabilities.

**CHAPTER-11**

**REFERENCES**

1. OWASP Foundation. "OWASP Testing Guide."

Available at: https://owasp.org/www-project-web-security-testing-guide/

2. Nikto Web Vulnerability Scanner.

Available at: https://cirt.net/Nikto2

3. Nmap Security Scanner Documentation.

Available at: https://nmap.org/book/

4. Rapid7. "Introduction to Vulnerability Management."

Available at: <https://www.rapid7.com/fundamentals/vulnerability-management/>

5. Qualys. "Vulnerability Management."

Available at: https://www.qualys.com/

6. Beck, Kent. "Test Driven Development: By Example."

Addison-Wesley, 2003.

7. Martin, Robert C. "Clean Code: A Handbook of Agile Software Craftsmanship."

Prentice Hall, 2008.

8. OWASP Foundation. "OWASP Top 10 Vulnerabilities."

Available at: https://owasp.org/www-project-top-ten/

**Mini Project Co-Ordinator Internal Guide**