WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Web Vulnerability Scanner

Nagananthini T S[1], Baruni M K[2] , Laksitha N R G[3]

[1]Assistant Professor, Department of Information Technology, K.L.N. College of Engineering, Sivaganga– 630612, Tamil Nadu, India.
[2,3],UG Scholar, Department of Information Technology, K.L.N. College of Engineering, Sivaganga – 630612, Tamil Nadu, India
nanthinivirush.22@gmail.com, barunibaruni6@gmail.com, laksitha.n.r.g@gmail.com

ABSTRACT

This project introduces a lightweight, Python-based Web Vulnerability Scanner aimed at helping developers, students, and small organizations detect common web security flaws efficiently and ethically. Built using libraries like requests, Beautiful Soup, socket, and an optional Tkinter GUI, the scanner integrates non-destructive checks such as HTTPS and connectivity validation, security header inspection, SQL injection and XSS heuristics, clickjacking detection, directory and sensitive file discovery, cookie flag analysis, email harvesting, subdomain probing, and basic port scanning. It performs polite, rate-limited scans, tagging results with clear severity indicators-[VULN], [WARN], [INFO], or [SAFE]-and generates timestamped reports for remediation. Emphasizing authorized use and ethical testing, the tool serves as an accessible and extensible learning platform, bridging the gap between manual testing and enterprise-grade scanners. Future improvements may include authenticated scanning, JavaScript rendering, concurrency for faster performance, and advanced reporting formats such as JSON or HTML with severity scoring.

Keywords: Lightweight, Security Testing, SQL Injection (SQLi), Cross-Site Scripting (XSS) , Connectivity, Secure development, Open-source scanner.

1. INTRODUCTION

Web applications have become essential for businesses, education, and personal use, but they are frequently targeted by attackers due to security vulnerabilities. Issues such as SQL injection, cross-site scripting (XSS), weak security headers, and exposed files can compromise sensitive data, disrupt services, and damage trust. Many small organizations and student projects lack access to commercial security tools, leaving a gap in proactive vulnerability detection.

To address this challenge, this project develops a Python-based Web Vulnerability Scanner. The tool integrates multiple non-destructive checks—including connectivity and HTTPS verification, header analysis, XSS/SQLi detection, clickjacking checks, directory and backup file enumeration, sensitive file discovery, cookie security inspection, email harvesting, subdomain probing, and basic port scanning—into a single, easy-to-use application.

By providing clear, timestamped reports and emphasizing ethical, safe testing, the scanner helps users identify and remediate vulnerabilities early. It serves as both an educational platform for learning web security concepts and a practical solution for developers and small organizations to improve the overall security posture of their web applications.

2. METHODOLOGY

This project implements a Python-based Web Vulnerability Scanner designed to detect common security flaws in websites through a structured, modular, and ethical scanning process. The methodology follows a systematic workflow-from user input and URL validation to vulnerability detection and report generation-ensuring accuracy, efficiency, and safety. The approach is divided into the following phases:

1. **User Interface Design**

The scanner features an optional Graphical User Interface (GUI) built using Tkinter, designed for simplicity and clarity.

Key UI components include:

- Input fields for entering the target website URL.
- Checkbox options for enabling/disabling SSL verification and confirming authorization to scan.
- Buttons for starting, pausing, or terminating the scan safely.
- A text area or console to display real-time scan results with colored tags ([VULN], [WARN], [INFO], [SAFE]).

- This interface ensures that both beginners and professionals can easily perform scans with clear visibility of the process.

2. Target Validation and Setup

Before initiating any scan, the system verifies the validity and accessibility of the target URL:

- The scanner checks HTTP/HTTPS connectivity and handles SSL certificate issues if the user opts to ignore them.
- The domain is normalized, and potential subpages or directories are identified for deeper inspection.
- A disclaimer ensures the user has explicit permission, maintaining ethical and legal standards for web testing.

3. Crawling and Link Enumeration

- Internal links, forms, and directories are collected for recursive analysis.
- Robots.txt and sitemap.xml are parsed if available to identify allowed/disallowed paths.
- The crawling process is rate-limited to avoid overwhelming the target server, maintaining responsible scanning behavior.

4. Vulnerability Detection and Analysis

- Checks for missing headers such as CSP, HSTS, X-Frame-Options, and X-Content-Type-Options.
- Sends heuristic payloads to form fields and URL parameters to detect possible vulnerabilities.
- Verifies if framing is allowed by inspecting the X-Frame-Options policy.
- Scans for sensitive or backup files.
- Ensures cookies have secure and HttpOnly flags enabled.
- Uses the socket library to check for open ports on the target domain.

5. Report Generation and Logging

After completing the scan, the tool compiles results into a timestamped text or CSV report:

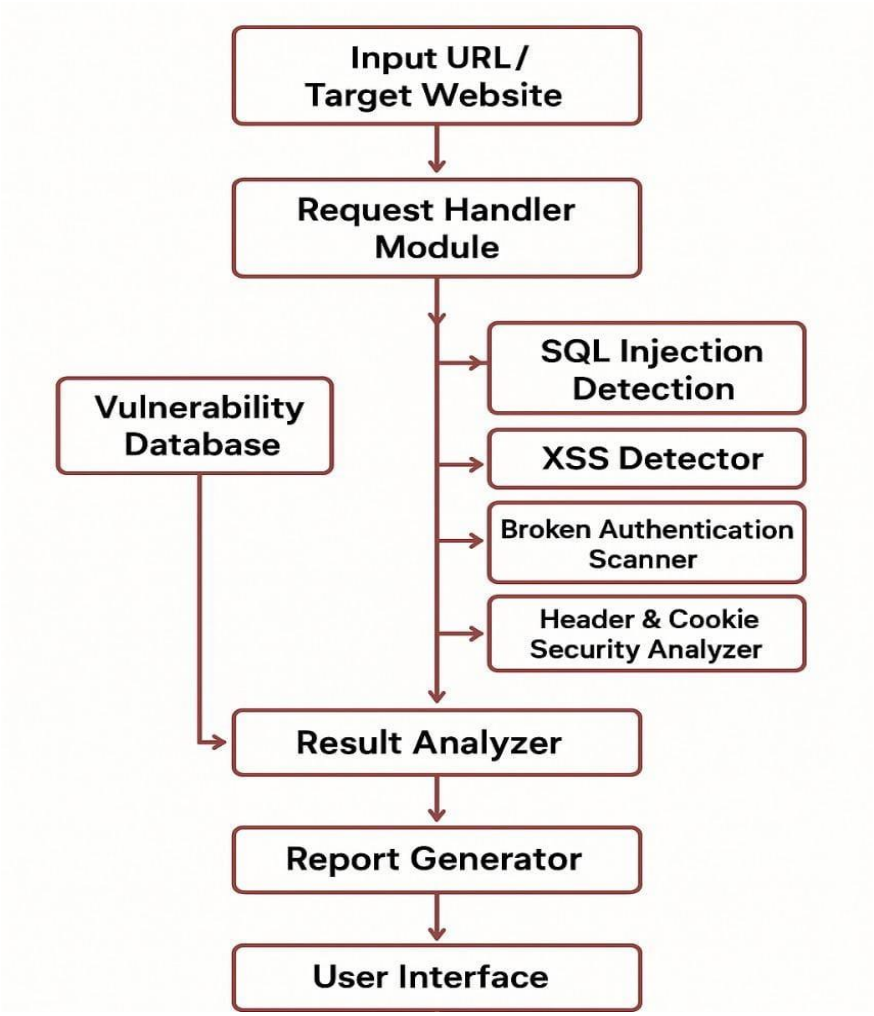
- Each entry includes the vulnerability type, affected URL or component, and recommended mitigation.
- Logs are also stored locally for reference and audit purposes.
- The GUI automatically displays a completion message, allowing users to view or export the findings.

6. Ethical Usage and Termination Control

To promote responsible use, the scanner emphasizes legal and ethical compliance throughout operation:

- Users must confirm authorization before scanning.
- The scanner includes a safe termination feature that gracefully stops ongoing scans while preserving partial results.
- This prevents accidental denial-of-service effects and ensures all data is saved before exit.

3.MODELING AND ANALYSIS



System Modeling Overview

The proposed system is designed to provide a lightweight, ethical, and efficient web vulnerability scanning solution that **detects common web application security flaws** through a modular and extensible architecture. Developed in Python with an optional Tkinter-based GUI, the system integrates components such as target validation, web crawling, vulnerability detection, and report generation. It follows the **Incremental Software Development Model**, enabling step-by-step implementation, testing, and enhancement of each module to ensure reliability and accuracy. This model supports continuous improvement while maintaining safe and authorized scanning practices, making the system suitable for **educational use, small-scale security audits, and early-stage vulnerability analysis.**

Software Development Life Cycle (SDLC) Model

This project follows the **Incremental Model**, where each development phase adds a functional module-such as target validation, crawling, vulnerability detection, or reporting. Each increment undergoes complete SDLC stages, ensuring continuous testing, integration, and improvement of the Web Vulnerability Scanner.

Reasons for Choosing the Incremental Model:

- Enables early development and testing of key scanning features like target validation and vulnerability detection.
- Allows each module (crawling, analysis, reporting) to be built, tested, and improved independently.
- Supports iterative refinement based on performance feedback and new security needs.
- Ideal for research-oriented projects like vulnerability scanners, where new threats and testing techniques evolve continuously.

Functional and Non Functional Requirements

Functional Requirements

Functionality	**Description**
Target URL Input	Accepts the Target URL.
Target Validation	Checks URL and SSL.
Web Crawling	Gathers Site or Forms.
Vulnerability Detection	Scan for Common Flaws.
Subdomain Scanning	Find Subdomain and email.
Port Scanning	Detects Open Server Ports.

Non-Functional Requirements

- **Security:** Ensures ethical and authorized scanning by requiring user consent and preventing destructive tests.
- **Usability:** Provides a clear, user-friendly Tkinter GUI for both beginners and professionals.
- **Performance:** Executes scans efficiently with rate-limiting to avoid server overload and ensure smooth operation.
- **Portability:** Runs seamlessly on any system with Python installed, requiring minimal setup.
- **Scalability:** Designed to easily integrate new modules such as authenticated scanning, concurrency, or advanced reporting.
- **Reliability:** Delivers consistent, accurate results with proper error handling and secure report logging.

System Architecture

The system is composed of five key modules:

Module	**Description**
GUI Module	Provides a user friendly interface for input and displaying results.
Entering Target URL Module	Takes and validates the target URL.
Web VAPT Module	Scans and test for vulnerabilities.
Web Vulnerability Scanning Module	Detects security flaws in web application.
Faculty / Department Info Module	Displays faculty and department details.

Data Flow Representation

The **Data Flow Diagram (DFD)** represents the logical flow from user input to system output.

- **Input:** Target URL + Scan options (SSL ignore, authorization).
- **Processes:** URL Validation → Web Crawling → Vulnerability Detection → Subdomain & Email Scanning → Port Scanning → Report Generation.
- **Output:** Vulnerability report (HTML/JSON/Console).
- **Storage:** Optional local report storage; no external server required.

UML Design Overview

The system is modeled using the following **UML diagrams** to ensure complete design visualization:

- **Use Case Diagram:** Shows user interactions (e.g., enter URL, start scan, view report).
- **Sequence Diagram:** Models the timeline of function calls between UI, scanning engine, and report generator.
- **Activity Diagram:** Describes workflows from target input to final vulnerability report.
- **Class Diagram:** Defines class responsibilities for modules like ScannerEngine, VulnerabilityDetector, ReportManager, UIManager.
- **Collaboration Diagram:** Maps interactions between modules using numbered messages for clarity.

Technological Stack and Implementation Layers

- **Python 3.x :** Core language powering scanner logic, network requests, threading, and orchestration of all modules.
- **Tkinter :** GUI layer handling input forms, buttons, progress bars, and live scrollable console output for user interaction.
- **Presentation Layer (GUI):** Tkinter interface with URL input, SSL ignore option, permission checkbox, progress bar, scan output console, and buttons for starting/stopping scans and saving reports.
- **BeautifulSoup4:** Parses HTML content to extract forms, links, and other page elements.

4.RESULTS AND DISCUSSION

The proposed GUI Vulnerability Scanner was successfully developed and tested as a fully client-Python desktop application using Tkinter. It performs safe, non-destructive vulnerability checks and reconnaissance on authorized web targets. The system met all core functionality objectives, including connectivity checks, header and cookie analysis, form and link parsing, directory and file inspection, subdomain probing, and basic port scanning.

Key outcomes:

- The scanner reliably connected to target URLs and retrieved HTTP/HTTPS responses with optional SSL verification bypass.
- Security headers, cookies, and server banners were analyzed to detect potential misconfigurations.
- Common backup files, sensitive files, and directory listings were detected and logged without causing harm.
- Forms, links, and emails were successfully extracted from pages, and basic XSS reflection tests were performed safely.
- Scan results were logged in real-time within the GUI and could be saved as timestamped reports for offline review.

Observations from User Interface

1. TargetInputPhase

Users could enter a target URL, select options like ignoring SSL errors, and confirm explicit permission before scanning. The interface ensured proper URL formatting and prevented scans without authorization.

2. ScanControl

The GUI provided clear buttons to start, stop, and save scans. The progress bar indicated ongoing activity, and the scrollable console displayed real-time scan output for transparency.

3. ScanResultsPhase

Scan results were logged live, including connectivity status, header analysis, cookie security, directory checks, exposed files, forms, emails, subdomains, and basic port scan outcomes. Users could save the full report as a timestamped text file.

4. UserFeedback and ErrorHandling

Validation prompts ensured are Alerts when the target URL was missing or invalid, Warnings if scans were attempted without confirming permission. Clear error messages for network failures, inaccessible targets, or unexpected scanner errors.

5. Responsiveness and Usability

The interface remained responsive during scanning thanks to threading. Users could stop scans gracefully, and all output was easily readable and scrollable for review.

6. ConfigurationOptions

Users could customize scanning behavior via checkboxes and options, such as ignoring SSL certificate errors for lab testing. These settings allowed flexible scans while maintaining safe defaults for ethical testing.

Test case result:

Test case result – Web Vulnerability Scanner

Test Case	Input/Action	Expected Output	Result
URL Validation	Enter Valid URL	URL Accepted,Ready For Scan	Pass
Invalid URL	Enter malformed URL	Error Message/invalid URL alert	Pass
SQL injection Detection	URL With test parameter	SQLi vulnerability flagged	Pass
XSS Detection	URL With test parameter	XSS vulnerability flagged	Pass
Security-Header Check	Target Website	Missing Headers Reported	Pass
Directory Enumeration	Target Website with Wordlist	Found Directories/files listed	Pass
Port Scanning	Target Host	Open Ports Displayed	Pass

Screenshots

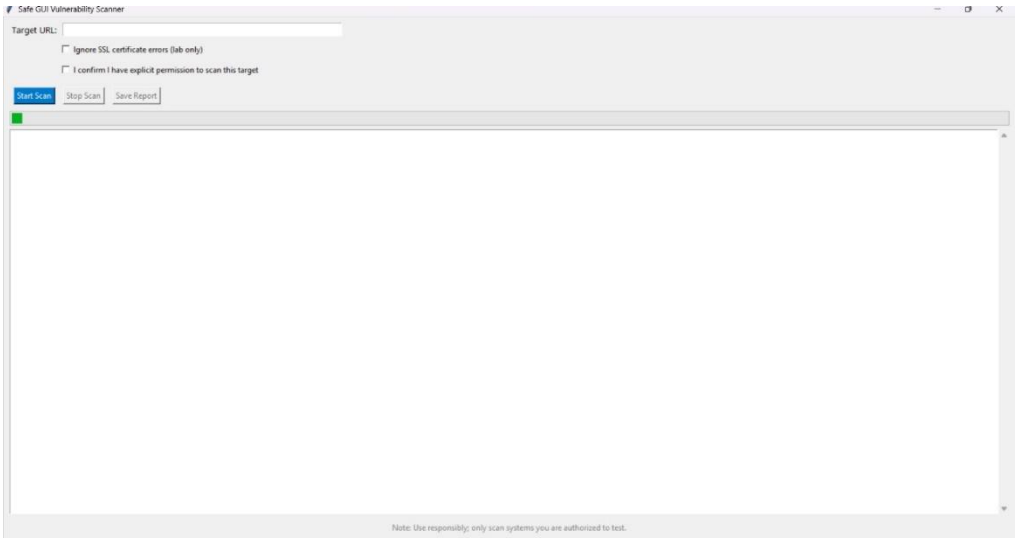


Fig:4.1 User Interface of Safe GUI Vulnerability Scanner

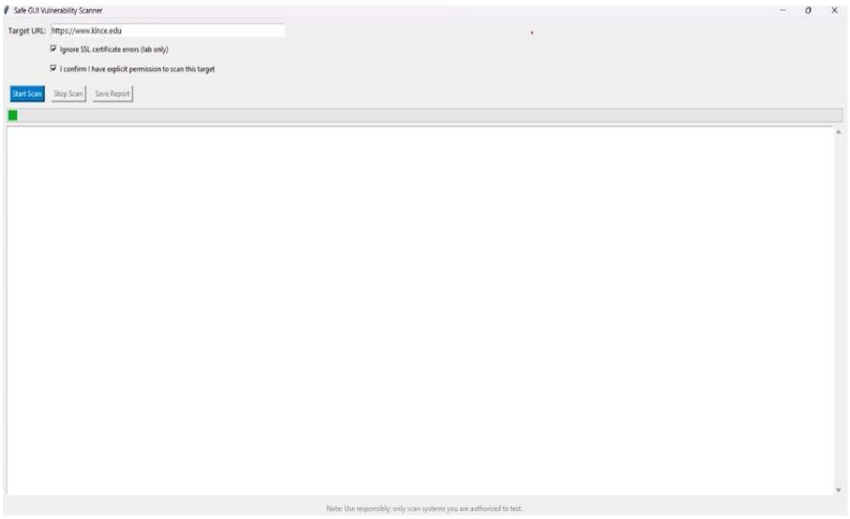


Fig:4.2 Entering Target URL and Selecting the checkbox Options

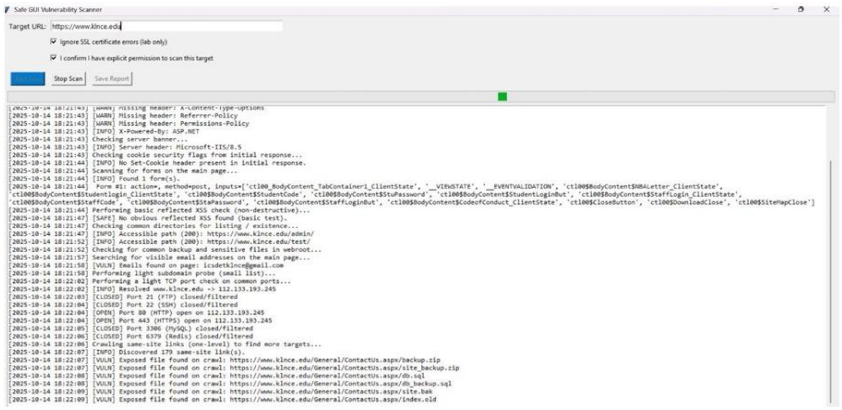


Fig 4.3 Starting the Scan and Verifying Website Open, Close, Safe , Info & Scan Status.

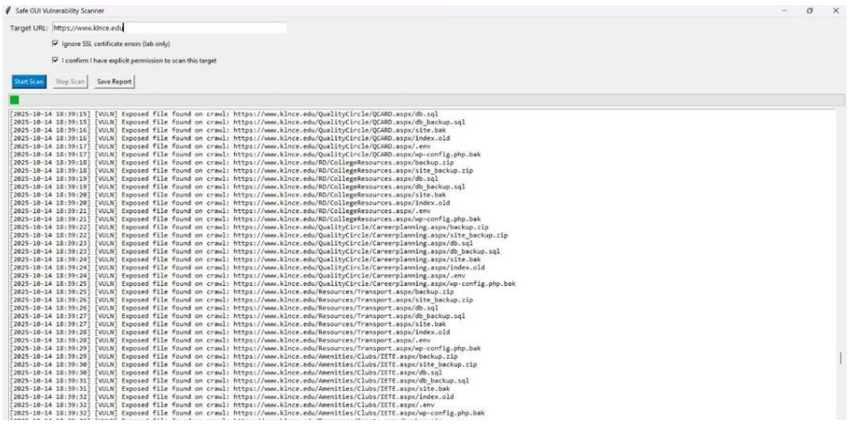


Fig 4.4 Vulnerability Scan Identifying and Displaying Multiple Exposed Files.

Fig 4.5 In Previous fig By copying and pasting the required sub-website URLs into the browser, we examined their page content, forms, and elements to identify details useful for testing.

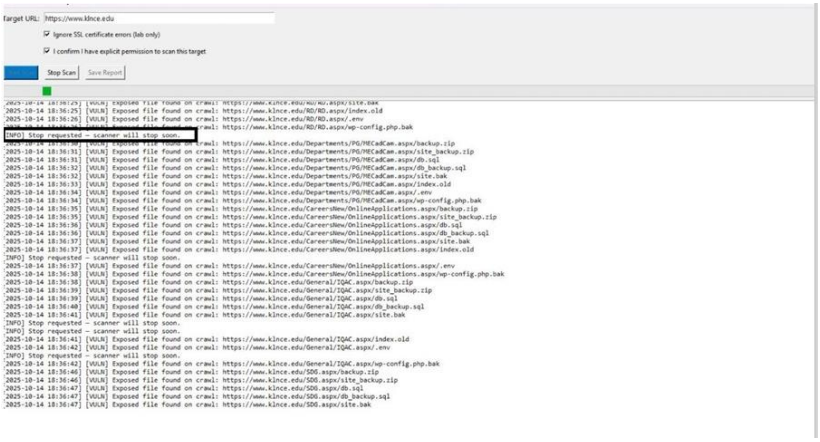


Fig 4.6 Select the Stop Scan Button that Comes Info as Stopping the Vulnerability Scan Process soon.

5.CONCLUSION

The project successfully demonstrates a safe, efficient, and user-friendly method for performing vulnerability reconnaissance on authorized web targets. By integrating multiple non-destructive checks—including connectivity, security headers, cookie analysis, directory and file inspection, form and email parsing, subdomain probing, and basic port scanning—the system provides a comprehensive overview of potential web security issues. The fully client-side desktop implementation ensures user privacy and control, while the intuitive GUI allows even non-technical users to conduct scans, monitor progress in real time, and save deleted report for further analysis. This approach highlights the practical application of ethical vulnerability scanning in modern cybersecurity, enabling responsible testing

6.REFERENCES

1. Building a Simple Web Application Security Scanner with Python — freeCodeCamp

https://www.freecodecamp.org/news/build-a-web-application-security-scanner-with python/

2. Nikto – Web Server Scanning With Nikto — freeCodeCamp

https://www.freecodecamp.org/news/an-introduction-to-web-server-scanning-with-nikto/

3. TPSQLi: Test Prioritization for SQL Injection Vulnerability Detection in Web Applications — arXiv

https://www.arxiv.org/abs/2509.10920

4. Vulnerability Scanning Tools — OWASP Foundation

https://owasp.org/www-community/Vulnerability_Scanning_Tools

5. Testing Tools Resource — WSTG (OWASP Web Security Testing Guide)

https://owasp.org/www-project-web-security-testing-guide/v42/6-Appendix/A Testing_Tools_Resource

6. 10 Vulnerability Scanning Tools to Know in 2025 — Pynt

https://www.pynt.io/learning-hub/application-security/10-vulnerability-scanning-tools-to

know-in-2025

7. 12 Popular Vulnerability Scanning Tools in 2025 — Red Canary

https://redcanary.com/cybersecurity-101/security-operations/vulnerability-scanning-tools/

8. OWASP Vulnerability Management Guide

https://owasp.org/www-project-vulnerability-management-guide/

9. TPSQLi: Test Prioritization for SQL Injection Vulnerability Detection in Web Applications -arXiv

https://arxiv.org/abs/2509.10920

10. VULNERABILITY SCANNERS: A Proactive Approach to Assess Web Application Security

https://www.researchgate.net/publication/261182006_Vulnerability_Scanners-A_Proactive_Approach_To_Assess_Web_Application_Security