IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 11, November 2026

# Web Application Vulnerability Scanner Using Python

**Mr. Thakare Yogesh Ravindra[1], Prof. Manoj V. Nikum[2], Prof. Shivam B. Limbhare[3]**

Research Scholar, MCA Student, Shri Jaykumar Rawal Institute of Technology, Dondaicha, India[1]

HOD MCA & Assistant Professor, Shri Jaykumar Rawal Institute of Technology, Dondaicha, India[2]

Project Guide, MCA Department, Shri Jaykumar Rawal Institute of Technology, Dondaicha, India[3]

**Abstract:** Web applications manage sensitive user data and are frequently targeted through weaknesses such as injection, cross-site scripting, insecure authentication, and broken access control. Manual security testing can be time-consuming and requires specialist knowledge. This project presents a Web Application Vulnerability Scanner that performs authorized, automated checks against a target web application to identify common security weaknesses. The system accepts a target URL, crawls reachable pages, and collects forms, links, headers, and input fields. It evaluates observable indicators including missing security headers, insecure HTTP use, exposed directory listings, weak form validation, and patterns associated with injection and cross-site scripting risk. Findings are categorized as low, medium, high, or critical and are presented in a report with remediation guidance. The proposed application uses Python, Flask, MySQL, Requests, and BeautifulSoup, with modules for target management, crawling, vulnerability checks, severity analysis, report generation, scan history, and an administrator dashboard. The scanner is intended only for locally hosted or legally authorized applications and is designed as an educational aid for students, developers, and security testers.

**Keywords:** Web security, vulnerability scanner, Python, Flask, OWASP, secure web application.

## I. INTRODUCTION

Web applications are central to education, commerce, public services, and enterprise operations. Their exposure to the Internet, frequent handling of user input, and dependence on multiple software components make security testing essential throughout development and deployment. The OWASP Top 10 identifies recurring web application risks, including broken access control, cryptographic failures, injection, security misconfiguration, and identification and authentication failures [1].

A practical security review combines automated checks with human judgement. For a student project, an automated scanner can make basic security assessment more repeatable by gathering target information, checking visible configuration signals, and organizing results in a single report. The proposed system is not a replacement for a professional penetration test; it is a controlled learning tool that helps users recognize common weaknesses early in the software life cycle.

## II. LITERATURE REVIEW

The OWASP Web Security Testing Guide provides a structured approach for evaluating web applications and services. Its coverage includes information gathering, configuration and deployment review, identity management, authentication, authorization, input validation, error handling, cryptography, business logic, and client-side security [2]. These areas guide the selection of checks in the proposed scanner.

Automated dynamic testing tools can crawl a running application and flag observable weaknesses, while manual testing remains important for context, exploitability validation, and business-logic defects. Existing tools such as OWASP ZAP demonstrate the value of combining discovery, passive analysis, and reporting. The present project focuses on a smaller, understandable Python implementation suitable for controlled laboratory environments [3].

## III. PROPOSED SYSTEM

The proposed system is a web-based scanner that accepts a permitted target URL and creates a bounded scan session. It discovers reachable same-origin pages, gathers metadata from HTTP responses and HTML documents, executes non-destructive checks, assigns a severity category, and stores the findings for later review. A Flask interface allows an authorized user to submit a target, review scan progress, inspect findings, and download a report.

© IARJSET This work is licensed under a Creative Commons Attribution 4.0 International License

TABLE I
SECURITY CHECKS INCLUDED IN THE PROPOSED SCANNER

| Check area | Observable evidence | Expected guidance |
|---|---|---|
| Transport security | HTTP use or missing transport policy | Use HTTPS and configure secure transport settings. |
| Response headers | Missing defensive response headers | Review headers such as Content Security Policy and HSTS. |
| Input handling | Forms or inputs requiring manual review | Validate input on the server and encode output. |
| Configuration exposure | Directory-listing or error-page indicators | Disable unnecessary listings and limit error details. |
| Authentication signals | Weak session or form-control indicators | Review identity, session, and access-control implementation. |

A. System Architecture

The architecture has five logical layers: a presentation layer for user interaction; a scan controller for job coordination; a crawler and collector for links, forms, and response data; a rule engine for security checks; and a reporting and persistence layer. MySQL stores scan metadata, findings, severity labels, and report history. This separation makes it easier to add checks without changing the user interface.

B. Functional Modules

Target management validates the submitted URL and applies the project scope. The crawler follows eligible links within the approved origin and records pages, forms, input fields, and headers. The detection module evaluates configuration and input-handling indicators. The severity module prioritizes findings, and the reporting module produces a readable summary that includes the affected URL, evidence, risk description, and a remediation suggestion.

## IV. METHODOLOGY

The scanner begins with authorization and scope validation. It then requests the target page through the Requests library, records the response status and headers, and parses HTML using BeautifulSoup. Extracted anchors and form elements are normalized, duplicate URLs are removed, and crawl depth is limited to avoid unnecessary load. Only requests that remain within the authorized target scope are considered.

Rules are designed to be non-destructive. Examples include checking whether HTTPS is used, whether selected security headers are present, whether directory listing indicators are exposed, and whether forms show basic client-side constraints. Pattern-based checks can identify inputs that warrant manual review for injection or cross-site scripting risk, but a finding is reported as an indicator rather than proof of exploitation. This preserves the tool's educational purpose and reduces the chance of unsafe testing behavior.

A. Severity Classification

Each finding receives a severity level based on the affected control, the apparent exposure, and the potential impact if the issue is confirmed. Critical and high findings require prompt review; medium findings require planned remediation; low findings indicate security hygiene improvements. The report clearly states that severity is an initial triage value and that an authorized security reviewer should validate material findings before remediation decisions are made.

## V. IMPLEMENTATION

Python is used for the core scanning logic because it supports rapid development and has mature libraries for HTTP communication, HTML parsing, and database integration. Flask provides the web application framework, Requests manages HTTP operations, BeautifulSoup parses returned HTML, and MySQL persists scan history and results. The interface presents scan status, a finding list, severity filters, and report-generation controls.

The system records the scan start time, target, visited URLs, observed response properties, findings, and completion state. Reporting converts stored findings into a concise document that supports remediation planning. Recommended actions include enabling HTTPS, configuring appropriate response headers, validating server-side input, encoding output, enforcing secure session handling, and reviewing access-control rules against OWASP guidance [1], [2].

 This work is licensed under a Creative Commons Attribution 4.0 International License 

IARJSET ISSN (O) 2393-8021, ISSN (P) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology

Impact Factor 8.311 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 11, November 2026

TABLE II
PROPOSED TECHNOLOGY STACK

| Layer | Technology | Purpose |
|---|---|---|
| User interface | HTML, CSS, JavaScript, Flask templates | Target submission, scan status, and findings dashboard. |
| Application logic | Python and Flask | Scope checks, scan orchestration, and report workflow. |
| HTTP collection | Requests | Controlled retrieval of authorized pages and headers. |
| HTML analysis | BeautifulSoup | Extraction of forms, links, and input fields. |
| Persistence | MySQL | Storage of scans, findings, severity, and history. |

## VI. TESTING AND RESULTS

Testing is limited to locally hosted applications or targets for which written authorization has been obtained. Test cases include pages with missing response headers, HTTP-only access, exposed directory-listing behavior, and forms lacking expected validation signals. The expected outcome is that the scanner records each observable indicator, assigns a clear severity label, and produces a readable report without performing destructive actions.

The primary evaluation criteria are correct collection of pages and forms, consistent application of rules, understandable severity labeling, report completeness, and adherence to the authorized scope. Findings should be manually reviewed because configuration signals and pattern matches may produce false positives or may not reveal application-specific business-logic weaknesses.

TABLE III
EVALUATION CRITERIA FOR THE AUTHORIZED TEST ENVIRONMENT

| Test case | Expected scanner output | Validation method |
|---|---|---|
| Missing response header | Finding identifies the absent control and provides guidance. | Compare reported header status with the local test application. |
| HTTP-only page | Finding is categorized for transport-security review. | Verify the observed scheme and report entry. |
| Exposed directory listing | Finding records the affected path and configuration indicator. | Review the controlled server response manually. |
| Form review | Finding flags input handling for manual confirmation. | Compare extracted form fields with the page source. |

## VII. ETHICAL AND SECURITY CONSIDERATIONS

Security scanning must be conducted only with explicit permission. The application therefore limits its intended use to controlled laboratory systems, locally hosted systems, or web applications for which the user has legal authorization. It avoids destructive payloads, credential attacks, and uncontrolled crawling. Scan logs should be protected because they can contain URLs, headers, and other operational information.

The project follows the principle that responsible testing should improve security without disrupting services or exposing user data. The scanner's output is advisory: it identifies areas for validation and remediation, not a license to test unrelated systems.

## VIII. LIMITATIONS AND FUTURE SCOPE

The current design focuses on common, externally observable weaknesses. It cannot fully verify business-logic flaws, authorization errors across complex roles, source-code defects, or every client-side behavior. JavaScript-heavy applications may require browser automation for complete discovery, and rule-based checks require regular updates as technologies and security guidance evolve.

© IARJSET This work is licensed under a Creative Commons Attribution 4.0 International License

Future work may add controlled integration with OWASP ZAP, authenticated test profiles for explicitly approved systems, browser-based crawling, export formats for security teams, and machine-learning-assisted prioritization. Any such enhancement should retain explicit scope controls, rate limits, audit logging, and a non-destructive default mode.

## IX. CONCLUSION

This paper presents a Python-based Web Application Vulnerability Scanner for educational and authorized security assessment. By combining target management, bounded crawling, rule-based checks, severity classification, history tracking, and report generation, the system gives users a structured way to recognize common web security concerns. Its greatest value is as an early-review aid: it encourages developers and students to identify, validate, and remediate weaknesses before deployment while maintaining ethical testing boundaries.

## ACKNOWLEDGMENT

The authors thank the Department of MCA, Shri Jaykumar Rawal Institute of Technology, Dondaicha, for the facilities and guidance provided during this project.

## REFERENCES

[1] OWASP Foundation, “OWASP Top 10:2021,” 2021. [Online]. Available: https://top10.owasp.org/2021/.
[2] OWASP Foundation, “OWASP Web Security Testing Guide,” version 4.2, 2023. [Online]. Available: https://wstg.owasp.org/v4.2/.
[3] OWASP Foundation, “Zed Attack Proxy,” [Online]. Available: https://www.zaproxy.org/.
[4] Python Software Foundation, “Python 3 Documentation,” [Online]. Available: https://docs.python.org/3/.
[5] Pallets Projects, “Flask Documentation,” [Online]. Available: https://flask.palletsprojects.com/.
[6] K. Reitz, “Requests: HTTP for Humans,” [Online]. Available: https://requests.readthedocs.io/.
[7] L. Richardson, “Beautiful Soup Documentation,” [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

© IARJSET This work is licensed under a Creative Commons Attribution 4.0 International License