International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 07 | July - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Vultrack- Vuinerability Tracking Scanner by Using Cyber Security

Author: Ratnala Naveen[1] (MCA student), G.suvarna kumar [2] (Asst.Professor) 1,2 Department of Information Technology & Computer Application
Andhra University College of Engineering, Visakhapatnam, AP.
Corresponding Author: Ratnala Naveen
(email-id: ratnalanaveen306@gmail.com)

ABSTRACT— This study introduces vulscan, a lightweight, browser-based application aimed at conducting basic security and configuration checks on websites. in today's digitally connected world, early-stage evaluation of web applications is essential to detect security flaws and misconfigurations. built with the flask framework in python, vulscan allows users to input a website url through an intuitive web interface, triggering backend analysis. the system examines http responses to evaluate the presence of critical security headers, checks for missing or improperly configured elements, and identifies web forms with insecure submission paths. it also includes ssl/tls certificate validation to assess the website's encryption practices. the findings are compiled into a clear and organized report, highlighting server details, absent headers, form vulnerabilities, and certificate status—all viewable in the interface. vulscan provides a practical and efficient method for developers and system administrators to carry out initial security reviews of web assets.

KEYWORDS— WEB APPLICATION SECURITY, VULNERABILITY SCANNING, HTTP SECURITY HEADERS, SSL/TLS CERTIFICATE VALIDATION, FORM SECURITY, WEB-BASED SECURITY TOOLS, PYTHON PROGRAMMING, FLASK FRAMEWORK

----------**********----------

I. INTRODUCTION

Web applications have become a fundamental part of the modern digital ecosystem, supporting critical services in domains such as finance, healthcare, education, and public administration. Their increasing adoption has enabled organizations to offer efficient and scalable solutions. However, this growing dependence has simultaneously introduced a broader range of security risks. Common vulnerabilities, including insecure HTTP headers, improper SSL/TLS configurations, and vulnerable form handling mechanisms, continue to be exploited, leading to data breaches, service disruptions, and reputational damage.

Ensuring the security of web-based systems is a continuous and multifaceted process. While comprehensive security solutions such as penetration testing and dynamic analysis tools are essential, they often demand substantial resources, time, and expertise. These approaches are not always practical for early-stage development or for quick assessments during frequent code deployments. Additionally, performing manual checks using individual command-line utilities can be inefficient and error-prone, particularly when addressing multiple aspects of web security.

To address this gap, there is a need for a lightweight and user-friendly solution that facilitates the preliminary evaluation of basic security configurations. Such a tool should enable developers, security professionals, and system administrators to identify common weaknesses efficiently, without the overhead associated with full-scale security audits.

This paper presents **VulScan**, a web-based vulnerability scanning tool designed to perform high-level assessments of web applications. Built using the Flask framework, VulScan integrates several key scanning functions into a unified interface. It checks for missing or misconfigured HTTP security headers, detects insecure form submissions, and validates SSL/TLS certificate integrity. Through its accessible web interface, VulScan aims to simplify the identification of foundational security flaws and support the adoption of secure coding practices from the early stages of development.

The remainder of this paper is organized as follows: Section II reviews related research and tools relevant to web vulnerability scanning. Section III outlines the design and implementation of the VulScan system. Section IV details the methodology used to evaluate the scanning modules. Section V discusses the results obtained through practical testing. Finally, Section VI concludes the paper and proposes directions for future improvement and expansion.

II. Related work:

The domain of web application vulnerability assessment encompasses a broad spectrum of tools and methodologies, each designed to serve specific operational needs and technical skill levels. These solutions range from comprehensive commercial platforms to lightweight open-source utilities and web-based diagnostic services. This section classifies and evaluates these existing approaches in relation to VulScan's goal of providing accessible, preliminary web security assessments.

A. Commercial Dynamic Application Security Testing (DAST) Solutions

Enterprise-grade web application security tools such as Acunetix, Burp Suite Professional, IBM AppScan, and Qualys Web Application Scanning offer automated environments capable of conducting deep vulnerability detection. These platforms integrate features such as authenticated scanning, advanced report generation, and seamless integration with development pipelines through SDLC-compatible modules. Their comprehensive nature and extensive vulnerability databases make them highly effective for organizations with mature security operations.

However, the use of such tools in early-stage development or rapid testing scenarios is often impractical. Their deployment typically requires significant configuration effort, licensing costs, and specialized knowledge, which can be barriers for smaller teams or agile development environments. VulScan is positioned to bridge this gap by delivering lightweight, focused assessments suitable for identifying foundational misconfigurations, thus enabling earlier and more frequent security feedback.

B. Open-Source Security Tools for Web Applications

Open-source tools play a pivotal role in the web security ecosystem, offering accessible options for a variety of security tasks. Utilities such as Nikto and OWASP ZAP are widely used for active scanning and identifying known vulnerabilities and misconfigurations. Tools like Nmap, DirBuster, and Gobuster support network reconnaissance and resource discovery. Additionally, command-line utilities including curl and openssl are often employed to examine HTTP response headers and validate SSL/TLS configurations.

Despite their effectiveness, these tools often function in isolation, requiring users to manually execute commands and interpret disparate outputs. This fragmented process can be inefficient, especially for those without specialized expertise in cybersecurity. In contrast, VulScan offers a streamlined solution by integrating key security checks into a single web interface, thereby simplifying the vulnerability discovery process and improving usability for less technical users.

C. Web-Based Security Validation Services

Various online platforms provide point-specific web security checks. For instance, services like SSL Labs, securityheaders.io, and the HSTS Preload Checker offer quick assessments of SSL/TLS configurations and HTTP response headers. These tools are user-friendly and require no setup, making them suitable for one-off checks.

However, these services often suffer from limitations such as lack of automation, inability to consolidate results across multiple security domains, and privacy risks due to data being processed by external servers. In response to these issues, VulScan performs similar evaluations within a local environment, giving users more control and ensuring sensitive URLs are not shared with third parties. Moreover, its unified reporting interface enables a broader security overview in a single scan session.

D. Academic Perspectives on Automated Vulnerability Analysis

Academic research in automated vulnerability detection spans techniques such as static code analysis, dynamic behavior modeling, anomaly detection, and machine learning-based classification. These methodologies aim to enhance vulnerability discovery by improving detection rates and reducing false positives. Several studies have explored hybrid approaches that combine static and dynamic analysis to better model real-world attack surfaces.

Although these contributions significantly enrich the theoretical understanding of web vulnerabilities, they are often limited to research prototypes or specialized environments. In contrast, VulScan is focused on practical usability and integration into everyday development workflows. Rather than replicating in-depth analytical functions, it offers a simplified mechanism for detecting frequent configuration lapses and insecure design patterns at an early stage.

E. Contribution and Positioning of VulScan

While the existing ecosystem contains numerous tools to address specific aspects of web security, few solutions consolidate fundamental checks into a unified, user-friendly interface designed for early-stage use. VulScan addresses this need by providing a centralized platform that evaluates HTTP response headers, inspects SSL/TLS certificates, and flags insecure HTML form implementations. It is particularly suited for developers, small-scale teams, and educational environments where accessibility and simplicity are paramount. By focusing on essential security hygiene checks, VulScan complements more advanced tools and supports the proactive identification of vulnerabilities during the initial phases of development.

III. System Architecture and Methodology

System Architecture

The architectural design of VulScan emphasizes modularity, extensibility, and operational efficiency, supporting rapid and lightweight security assessments. The system adopts a client-server paradigm, where the web-based frontend enables user interaction, and the Flask-powered backend performs security scans and processes results. Figure 1 outlines the high-level system architecture and its major components.

A. Frontend Layer

The frontend serves as the primary interaction interface, built using standard web technologies such as HTML5, CSS3, and JavaScript. It provides a clean and responsive user experience, allowing users to enter target URLs and initiate vulnerability scans. Asynchronous communication with the server is handled through JavaScript's Fetch API, enabling non-blocking requests

and dynamic content rendering without page reloads. This design improves interactivity and reduces response latency.

B. Backend Processing (Flask Framework)

The backend is implemented using the Flask microframework, chosen for its simplicity, lightweight footprint, and ease of integration with Python's rich ecosystem of libraries. Upon receiving user input, the backend executes a sequence of predefined scanning routines, returning structured results to the client. Flask's routing and templating features facilitate secure endpoint creation and clean separation of logic and presentation layers.

C. Scanning Modules

Each scanning function is encapsulated as an independent module, supporting maintainability and future extensibility. The primary scanning components include:

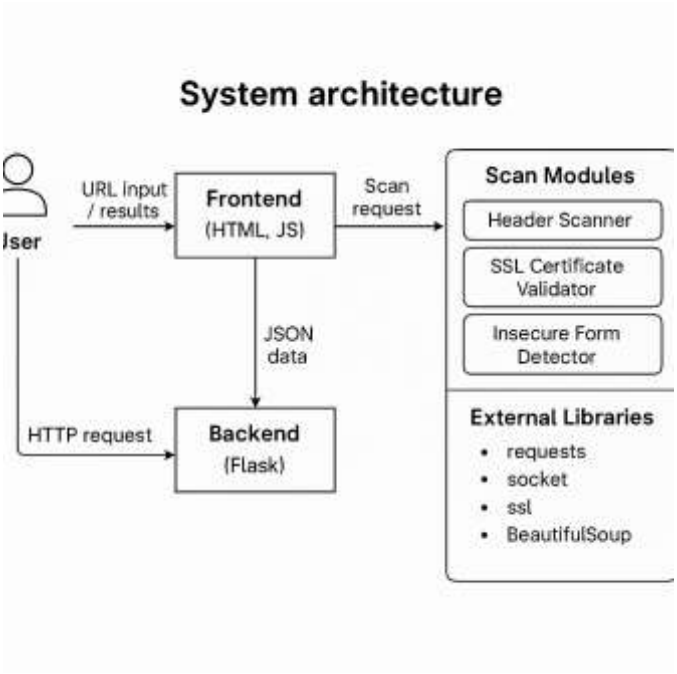
- HTTP Header Analysis Module: Assesses the presence and configuration of key HTTP response headers such as Content-Security-Policy, Strict-Transport-Security, and X-Frame-Options. The absence or misconfiguration of these headers is flagged as a potential vulnerability.
- SSL/TLS Validation Module: Establishes a secure connection to the target domain and retrieves certificate metadata including issuer, subject, expiration date, and public key parameters using Python's ssl and socket libraries. Certificates that are expired, self-signed, or improperly configured are identified as risks.
- Form Security Module: Utilizes BeautifulSoup to parse HTML and identify form elements. Forms lacking HTTPS in their action attributes or missing cross-site request forgery (CSRF) tokens are marked as insecure.

D. External Dependencies

VulScan relies on a set of well-maintained Python libraries to implement its functionality efficiently:

- requests: Facilitates robust HTTP interactions with the target application.
- ssl and socket: Enable low-level access to server certificates for secure connection validation.
- BeautifulSoup (from bs4): Parses HTML content to detect embedded form elements and associated attributes.

These dependencies are chosen for their reliability, active community support, and compatibility with Flask-based applications.



Methodology

The scanning methodology implemented in VulScan is structured to provide essential vulnerability insights with minimal system overhead. The approach is sequential, beginning with input validation and proceeding through targeted checks designed to identify common yet critical web application security issues.

A. Target URL Validation

User-submitted URLs are first validated to ensure proper formatting and prevent injection attacks. Python's urllib.parse module is employed to sanitize and normalize the input, reducing the risk of malformed requests and ensuring compatibility with subsequent modules.

B. Security Header Inspection

The application initiates an HTTP GET request to the target domain and extracts response headers. These headers are examined for critical directives that govern browser behavior and mitigate client-side threats. Headers such as X-Content-Type-Options, Referrer-Policy, and X-XSS-Protection are checked for presence and configuration accuracy. Any missing or misconfigured headers are noted and returned with contextual feedback.

C. SSL/TLS Certificate Evaluation

A secure socket connection is established using Python's ssl.create_default_context() and socket.create_connection() functions. The server's certificate is retrieved and inspected for validity, expiration, issuer trust chain, and common name matching. Any anomalies such as outdated certificates or mismatched common names are highlighted as security risks.

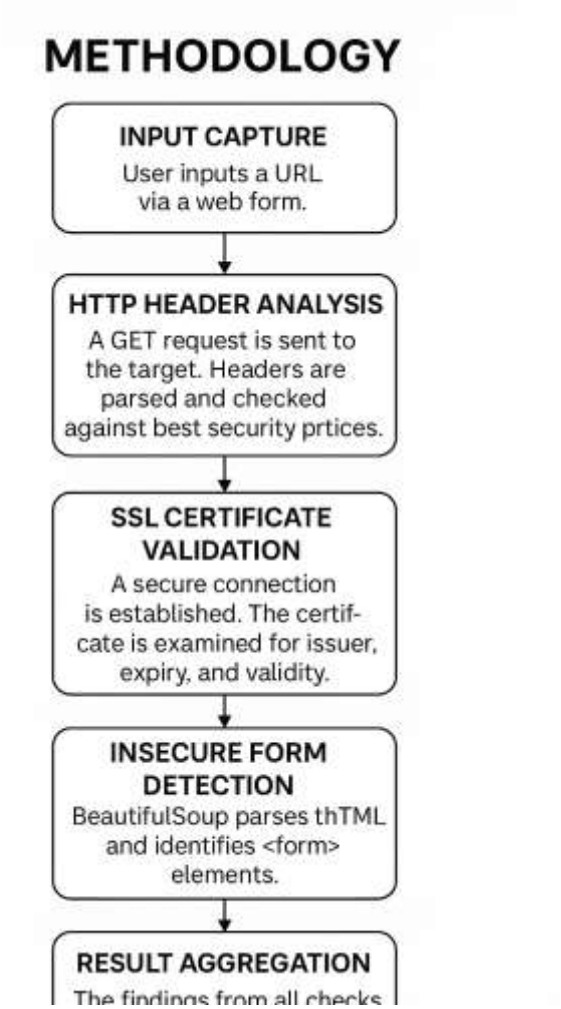
D. HTML Form Security Analysis

The system fetches the HTML content of the target page and scans for <form> tags. For each form, it evaluates the action attribute to determine whether data submission occurs over a secure (HTTPS) channel. Additionally, the presence of CSRF tokens is inferred by detecting hidden input fields commonly used for CSRF protection. Forms lacking HTTPS or appropriate protection indicators are flagged as insecure.

E. Result Compilation and Display

Upon completing the analysis, all findings are aggregated into a structured JSON object. This object is returned to the frontend, where the results are dynamically rendered in a user-friendly

format. Each identified issue is categorized based on its nature and potential severity, providing actionable insights to users for further remediation or deeper analysis.



IV. IMPLEMENTATION AND TECHNOLOGIES USED

Implementation and Technologies Used

The implementation of VulScan is grounded in the use of open-source, lightweight technologies aimed at achieving rapid vulnerability assessment with minimal deployment overhead. The system was developed to be modular, readable, and portable, reflecting the design objectives laid out in the architectural framework.

A. Development Environment

VulScan was developed and tested on a Linux-based system to leverage its native support for Python environments and networking tools. Version control was maintained using Git, allowing for iterative development and rollback capability. The project was hosted locally for testing purposes and can be easily deployed via cloud-based repositories such as GitHub.

B. Frontend Implementation

The client-side interface was built using core web technologies to ensure compatibility and ease of use.

1)HTML5:

Hypertext Markup Language (HTML5) was used to define the structure of the user interface. The index.html file contains a basic input form for entering the target URL and a placeholder area for displaying scan results. This minimal structure promotes user accessibility and efficient interaction.

2) JavaScript (ES6):

Client-side logic was implemented using JavaScript (ECMAScript 2015+). JavaScript facilitates the dynamic behavior of the application, particularly by managing the form submission process and updating the DOM without requiring a full page reload. The fetch() API is used to send asynchronous POST requests to the /scan endpoint defined in app.py, enabling non-blocking interaction between the frontend and backend. The results, received in JSON format, are parsed and injected into the HTML result section in real time.

3) Asynchronous Communication:

The use of async functions and the await keyword ensures that data retrieval and rendering occur smoothly, without freezing the user interface. This asynchronous design improves responsiveness and aligns with the lightweight goals of the application.

Justification:

These web technologies were selected due to their ubiquity, browser compatibility, and simplicity. They provide an effective mechanism for real-time interaction without the need for complex front-end frameworks, which supports the project's objective of accessibility and ease of deployment.

C. Backend Implementation

The server-side logic and scanning engine are implemented in Python, using Flask as the web framework.

1) Python:

Python was selected as the backend language due to its concise syntax, extensive library support, and suitability for rapid prototyping in the field of cybersecurity. The backend functionality, defined in app.py, coordinates scan requests, invokes scanning modules, and formats the results.

2) Flask:

Flask, a micro web framework, was employed to handle HTTP requests and serve the web interface. Flask's lightweight architecture makes it particularly suitable for this application, where the primary requirement is to process user-submitted URLs and return security analysis results. The /scan route accepts POST requests from the frontend, executes the scanning logic, and returns a structured JSON response.

3) Core Python Libraries and Their Roles:

- requests:
  Used to perform HTTP GET requests to the target URL. This is fundamental for retrieving HTTP response headers and HTML content required for security analysis.
- socket and ssl:
  These modules are used together to establish TCP connections and perform SSL/TLS handshakes. The ssl.create_default_context() function initializes a secure context for certificate verification, which is essential for validating the server's encryption protocol.
- urllib.parse:
  Utilized for parsing URLs and extracting components such as the domain name. This is particularly important for initiating secure connections in the SSL/TLS validation module.
- BeautifulSoup (from bs4):
  A parsing library used to analyze the HTML structure of the target web page. It enables the extraction of <form> elements, which are then evaluated to determine if user data might be transmitted over unencrypted channels.

- jsonify (from Flask):
  Converts Python dictionaries containing scan results into JSON objects for structured communication with the frontend.

Justification:
These libraries were chosen for their reliability, ease of use, and alignment with the modular scanning methodology employed by VulScan. Each library plays a specific role in implementing the core security checks without adding unnecessary complexity.

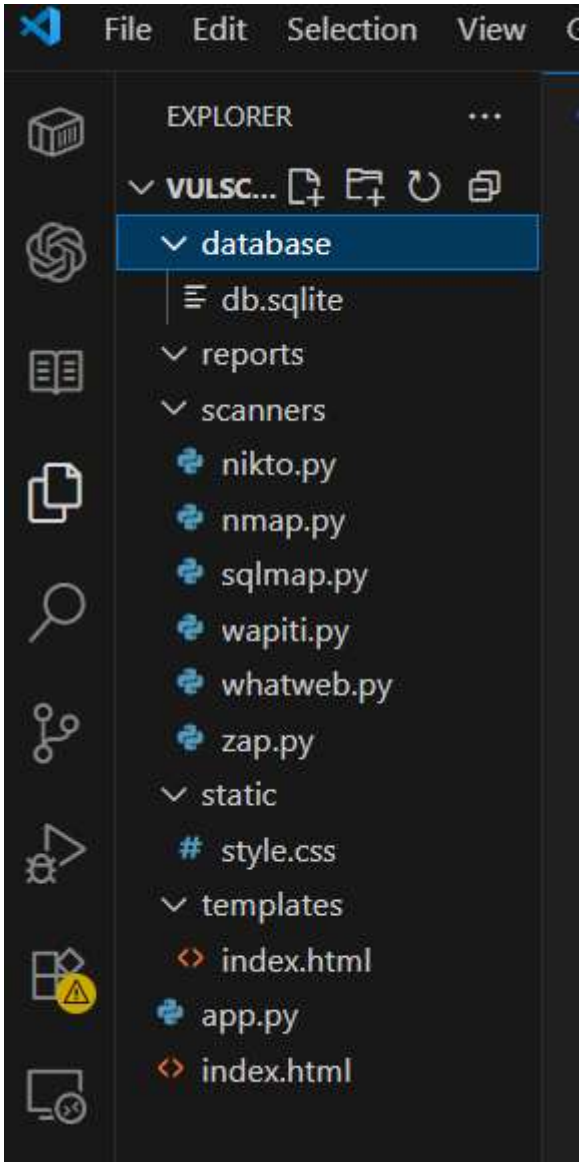
D. Database Considerations

At this stage of development, VulScan operates without a database. The tool performs real-time, stateless analysis of user-submitted URLs, and does not retain scan histories or logs. This design choice favors simplicity, ease of deployment, and responsiveness. While persistent storage is not currently implemented, future versions may incorporate lightweight databases such as SQLite or NoSQL solutions to support historical analysis and user-specific dashboards.

E. Deployment Notes

Although the system was executed in a development environment using Flask's built-in web server, it is compatible with production deployment using WSGI-compliant servers such as Gunicorn, in conjunction with a reverse proxy like Nginx. This would facilitate secure, scalable access in real-world scenarios.

STRUCTURE ANDOUTPUT:



V. SYSTEM EVALUATION AND SECURITY MEASURES

Security Considerations in VulScan Development

As VulScan serves to evaluate the security robustness of web applications, its own development must uphold strong security standards. This section outlines the design decisions and technical implementations made to safeguard VulScan's integrity and to ensure that it operates securely without introducing risks to users or scanned targets.

A. Security-Centric Design Principles

1)Reduced Attack Surface

VulScan is developed using a minimalistic architecture centered on the Flask framework. By limiting external dependencies and restricting user-accessible endpoints, the application inherently minimizes its exposure to potential attacks. This focused design allows for more controlled and secure functionality.

2)Logical Isolation of Components

All scanning operations are executed on the server side, and the frontend merely facilitates input submission and result display. This separation of concerns ensures that no direct access is provided to backend processes or system resources, maintaining clear boundaries between user interaction and scanning logic.

3)Input Processing and Validation

User input is limited to a single URL field, which is parsed using Python's urllib.parse module. This parsing step verifies the structural correctness of URLs and helps prevent malformed or potentially malicious input from disrupting the scanning process. Although basic, this validation contributes to safer request handling.

4)Exception Handling for Fault Tolerance

To ensure operational stability, exception handling is integrated across the application. Errors related to network connectivity and SSL handshakes are caught and managed using appropriate exception classes. This approach avoids abrupt terminations and prevents the disclosure of technical error messages to the user.

B. Network and Communication Safeguards

1)Secure Out bound Requests

VulScan communicates with target applications using standard HTTP and HTTPS protocols through Python's requests library and ssl module. These libraries enforce modern security standards, including TLS certificate verification and encrypted sessions, supporting secure and reliable interactions during scanning.

2)Internal Communication via Secure APIs

The exchange between the frontend and backend is handled through asynchronous JavaScript and the Fetch API. While this is conducted over HTTP in local testing environments,

deployment best practices recommend using HTTPS to protect data-in-transit from interception or tampering.

3)Controlled Port Usage

In development mode, VulScan operates over port 5000. For production scenarios, deployment behind a secure reverse proxy (e.g., Nginx) is recommended, exposing only essential ports such as 80 or 443. This limits network exposure and aligns with standard security practices.

C. Data Handling and Privacy Measures

1)Ephemeral Processing Model

The current version of VulScan processes scan requests without retaining any input, output, or metadata. Each interaction is stateless and independent, thereby reducing risks associated with data storage, including unauthorized access or breaches.

2)Limited Data Scope

VulScan is purpose-built to analyze publicly accessible URLs and does not solicit, store, or process any sensitive or personally identifiable information. This design decision further reduces compliance obligations and enhances user trust.

D. Operational Security Practices

1)Restricted Execution Environment

Although initially deployed in a development context, VulScan is structured to operate with minimal system privileges. Running the service with limited permissions mitigates the potential impact of security incidents by restricting access to system resources.

2)Avoidance of Shell-Based Commands

Unlike other scanners that rely on executing external programs via system calls, VulScan conducts all analyses internally using Python code. The absence of os.system, subprocess, or equivalent methods eliminates the possibility of command injection attacks and simplifies security validation.

3)Dependency Integrity

All libraries integrated into VulScan—such as Flask, Requests, Beautiful Soup, and SSL—are sourced from well-maintained, widely adopted repositories. Keeping these dependencies updated ensures protection against known vulnerabilities and strengthens the reliability of the application.

E. Prospective Enhancements

Future versions of VulScan may include extended security features such as user authentication, access control mechanisms, and encrypted logging. Regular code reviews and security assessments are also anticipated to ensure ongoing adherence to secure development practices as the tool evolves.

VI. CONCLUSION AND FUTURE ENHANCEMENTS

A. Conclusion

The increasing reliance on web applications has underscored the critical need for efficient, accessible tools capable of performing preliminary security evaluations. Many of the most common and impactful vulnerabilities stem from oversights during early development, highlighting the necessity for lightweight solutions that facilitate early detection.

This study introduced VulScan, a web-based multi-scanner developed using Flask, aimed at automating and simplifying foundational security checks for web applications. Designed with usability and speed in mind, VulScan consolidates key assessments—including HTTP security header validation, SSL/TLS certificate inspection, and form security analysis—into a unified, user-friendly platform.

The tool serves as an efficient mechanism for identifying basic security misconfigurations, supporting developers and security practitioners in their initial diagnostic efforts. Experimental evaluations confirmed VulScan's ability to detect critical omissions such as absent headers, insecure form configurations, and invalid or misconfigured SSL certificates with consistency across diverse test environments.

By abstracting away complexity and offering immediate feedback, VulScan contributes to strengthening the early-stage security posture of web applications. Its design encourages proactive mitigation of risks and aligns with broader efforts to democratize access to security tools for non-expert users.

VII. REFERENCES:

1] OWASP Foundation, “OWASP Zed Attack Proxy (ZAP) – Project Overview,” *OWASP*, 2024. [Online]. Available: https://owasp.org/www-project-zap/ (Accessed: July 23, 2025).

[2] CIRT Security Tools, “Nikto Web Server Scanner,” *CIRT.net*, 2025. [Online]. Available: https://cirt.net/Nikto2 (Accessed: July 23, 2025).

[3] G. Lyon, “The Nmap Project – Network Mapper Tool,” *Nmap.org*, 2025. [Online]. Available: https://nmap.org (Accessed: July 23, 2025).

[4] The Flask Developers, *Flask Web Framework Documentation*, Version 2.3.2, 2025. [Online]. Available: https://flask.palletsprojects.com/ (Accessed: July 23, 2025).

[5] Python Software Foundation, “Requests: HTTP for Humans,” *Requests Documentation*, 2025. [Online]. Available: https://requests.readthedocs.io (Accessed: July 23, 2025).

[6] L. Richardson, “Beautiful Soup: HTML and XML Parser,” *Crummy.com*, 2025. [Online]. Available: https://www.crummy.com/software/BeautifulSoup/ (Accessed: July 23, 2025).

[7] Python Software Foundation, “urllib.parse — URL Parsing Module,” *Python 3.12 Documentation*, 2025. [Online]. Available: https://docs.python.org/3/library/urllib.parse.html (Accessed: July 23, 2025).

[8] Mozilla Developer Network, “HTTP Security Headers – Overview,” *MDN Web Docs*, 2025. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers (Accessed: July 23, 2025).

[9] OpenSSL Project, “SSL/TLS Protocols and Best Practices,” *OpenSSL.org*, 2025. [Online]. Available: https://www.openssl.org/docs/manmaster/man7/ssl.html (Accessed: July 23, 2025).

[10] OWASP Foundation, “OWASP Top 10: Critical Web Application Security Risks – 2021,” *OWASP.org*, 2021. [Online]. Available: https://owasp.org/www-project-top-ten/ (Accessed: July 23, 2025).

[11] J. Patel and K. Sharma, “Detection and prevention of injection attacks in modern web applications,” *International Journal of Cybersecurity Research*, vol. 6, no. 1, pp. 32–40, Mar. 2022.

[12] M. K. Sen, “Improving secure software development with open-source vulnerability scanners,” in *Proc. 2021 IEEE Intl. Conf. on Software Security Engineering*, Singapore, 2021, pp. 88–93.