iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** VIII **Month of publication:** August 2026

DOI: https://doi.org/10.22214/ijraset.2026.84617

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue VIII Aug 2026- Available at www.ijraset.com

HawkEye: Web Vulnerability Analysis and Security Audit Tool

Dipak Patil[1], Varad Salgare[2], Devaj Arya[3], Prof. Suvarna Satkar[4]
Department of Computer Engineering, G H Raisoni College of Engineering and Management, Pune, India

Abstract: ***The widespread adoption of web applications has significantly increased exposure to security risks, making automated vulnerability assessment an essential component of modern software development. Although several open-source scanners provide strong detection capabilities, they typically operate independently, require technical expertise, and produce unstructured reports, limiting their accessibility and effectiveness. This paper introduces HawkEye, a modular, web-based vulnerability auditing platform designed to streamline security analysis by integrating multiple scanning tools within a unified dashboard. HawkEye uses Docker-isolated environments to orchestrate scanners and employs a Django–SQLite3 backend to manage scan workflows and result storage. The current implementation provides full integration for OWASP ZAP and Nikto, while modular support is included for incorporating Nmap, SQLMap, and Wapiti. The platform consolidates heterogeneous outputs into a standardized schema, applies CVSS-based severity classification, and generates exportable PDF/JSON reports with remediation guidance. Experimental testing on intentionally vulnerable applications demonstrates HawkEye's ability to detect prominent security weaknesses and illustrates how consolidated reporting improves vulnerability prioritization for development teams.***

Index Terms: ***Web Application Security, Vulnerability Scanning, OWASP ZAP, Nikto, SQLMap, Nmap, Wapiti, CVSS, CWE, Cybersecurity Automation, Penetration Testing, DAST, Network Scanning, Secure Software Development, Security Risk Assessment.***

I. INTRODUCTION

The rapid evolution of web technologies has fundamentally reshaped digital ecosystems, enabling seamless communication, online commerce, and large-scale data exchange across the globe. This shift, however, has also intensified the security challenges associated with modern web infrastructures, as attackers increasingly exploit weaknesses within application layers and server configurations [2]. Vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), and other categories outlined in the OWASP Top Ten continue to be widely targeted due to their prevalence and potential impact [3].

While the importance of vulnerability assessment is well recognized, conducting effective security testing remains difficult for many organizations. Developers, educational institutions, and small enterprises often lack the specialized expertise or resources required to perform comprehensive penetration testing. Traditional assessment tools typically rely on command-line interfaces or complex configurations, making them impractical for routine or large-scale deployment [6]. As a result, many systems remain insufficiently tested, increasing the likelihood of data breaches and service disruption.

A variety of open-source scanners address specific aspects of security analysis—such as Nikto for server misconfigu-rations, OWASP ZAP for dynamic analysis, SQLMap for injection flaws, Nmap for network exploration, and Wapiti for input validation checks [4], [7]. Although each tool is effective within its domain, they operate independently and output results in disparate formats, complicating the tasks of correlation, severity assessment, and remediation prioritization. Studies indicate that the combined use of multiple scanners improves detection coverage and accuracy, yet few platforms provide a unified interface that integrates these tools seamlessly [1],[8].

Another persistent limitation is the absence of standardized reporting across scanners. Many tools output raw findings without mapping vulnerabilities to frameworks such as the Common Weakness Enumeration (CWE) or the Common Vulnerability Scoring System (CVSS), making it difficult for developers to evaluate risk or determine remediation urgency [9]. Research suggests that incorporating structured scoring and automated analysis significantly enhances the vulnerability management workflow [10].

Usability is also a critical factor. The requirement for manual installation, local setup, and command-line interaction serves as a barrier for non-expert users. With the growing shift toward cloud-based and browser-accessible tools, there is a clear need for a web-based platform that supports automated scanning, real-time result visualization, and streamlined reporting without extensive configuration efforts [11].

To address these challenges, this paper presents HawkEye, a modular and automated vulnerability analysis platform that consolidates multiple scanning tools into a single web interface. Built using the Django framework with an SQLite3 backend, HawkEye leverages Docker containers to isolate scanning processes and ensure safe execution. The platform standardizes vulnerability outputs using CWE identifiers and CVSS severity ratings while enabling live scan tracking, consolidated visualization, and automated PDF reporting. By integrating accessibility, automation, and standardized analysis, HawkEye aims to make effective security auditing achievable for users with varying technical backgrounds.

The key contributions of this work are as follows:

1) Development of a unified web-based framework that integrates multiple vulnerability scanners for automated detection and reporting.
2) A standardized parsing and normalization pipeline with CWE and CVSS-based severity classification.
3) A browser-accessible dashboard supporting real-time scanning, visualization, and automated report generation.
4) Evaluation of HawkEye on intentionally vulnerable applications to assess usability, coverage, and accuracy.
5) A modular architecture designed for future integration of AI-assisted analysis, adaptive scanning, and real-time trend monitoring.

II. RELATED WORK

Research in web application security has expanded steadily over the past two decades, driven by the increasing complexity of online systems and the growing sophistication of cyberattacks. Prior studies have explored automated vulnerability detection, dynamic application testing, and penetration-testing frameworks, with emphasis on improving accuracy, integration, and usability. Despite these advancements, existing tools still tend to function independently, produce disparate outputs, and require considerable technical expertise to operate effectively [4]. HawkEye seeks to address these limitations by integrating established open-source scanners into a unified, accessible web platform.

One of the earliest and most influential contributions to network-level reconnaissance is the Nmap tool, widely cited in vulnerability assessment literature for its ability to enumerate services, ports, and network configurations [5]. Although Nmap remains a valuable resource for identifying exposed infrastructure, it does not provide comprehensive dynamic testing of web applications. To address server-side weaknesses, scanners such as Nikto were introduced to identify outdated components and configuration flaws, offering insight into common server vulnerabilities [6]. However, Nikto's output typically requires manual interpretation, and its static reporting can slow the remediation workflow. Further research expanded toward dynamic, application-layer vulnerability scanning. Studies such as those presented in [7] examined the effectiveness of automated black-box scanners and highlighted substantial variation in their ability to detect common threats, including SQL Injection and Cross-Site Scripting (XSS). These findings emphasized the importance of combining multiple tools to achieve broader coverage. Complementing this, the OWASP Zed Attack Proxy (ZAP)—listed among widely adopted security tools in contemporary evaluations [8]—introduced a comprehensive open-source platform for automated crawling and dynamic testing. Owing to its extensibility and active community support, ZAP has become a standard in both academic and professional security assessments.

Specialized tools such as SQLMap streamline the detection and exploitation of SQL Injection vulnerabilities through automated payload generation and database-level testing, while scanners like Wapiti perform black-box testing for a range of web application flaws. Although effective within their respective domains, many of these tools present operational barriers for non-technical users due to command-line interfaces and limited reporting standardization.

HawkEye distinguishes itself by consolidating these diverse scanning capabilities into a single web-based environment. By centralizing execution, normalizing outputs, and applying recognized standards such as CWE classifications and CVSS scoring, HawkEye aims to simplify analysis and enhance the usability of automated vulnerability assessments. Through its modular design, result-unification pipeline, and automated reporting, HawkEye represents a practical step toward improving accessibility and efficiency in web application vulnerability management.

III. PROPOSED SOLUTION

HawkEye is designed to address the limitations of traditional vulnerability scanners, which often work in isolation and require considerable technical expertise. To overcome these challenges, the platform introduces a unified, web-based framework capable of automating vulnerability detection, consolidating multi-tool results, and producing standardized security reports. The architecture emphasizes five core principles: multi-layered detection, consistent result normalization, automated and containerized execution, high accessibility through a browser interface, and secure deployment.

A. Core Design Concept

Conventional scanners typically operate as standalone utilities, forcing users to perform manual correlation of findings or rely on command-line execution. In contrast, HawkEye centralizes the entire assessment workflow through a modular architecture that separates scanning, data processing, visualization, and reporting components. Each module operates independently but communicates through well-defined internal interfaces, enabling scalability and easier maintenance. This modularity aligns with emerging multi-component scanning approaches highlighted in modern vulnerability research [5].

B. Integrated Vulnerability Scanning Engine

At the center of HawkEye is its scanning engine, which coordinates multiple open-source tools to provide broad vulnerability coverage. The current implementation supports full Docker-isolated integration for OWASP ZAP and Nikto, which perform dynamic application scanning and server configuration analysis, respectively. Additional modules are structured to incorporate tools such as SQLMap, Nmap, and Wapiti, enabling deeper inspection of injection flaws, network exposure, and input-validation weaknesses. All scanners are executed through Python-based orchestration—either via subprocess calls or containerized execution—to ensure isolation, stability, and controlled resource usage. The outputs from each tool are collected into a unified dataset for further processing.

C. Result Normalization and Severity Mapping

Because scanners produce results in different formats, HawkEye introduces a normalization layer that translates heterogeneous outputs into a consistent internal schema. Each identified issue is mapped to a relevant CWE classification and assigned a CVSS v3.1 severity score, allowing users to understand the risk level of each finding. Overlapping or duplicate results across scanners are automatically merged to avoid redundancy and improve precision. The normalized data is stored in the Django–SQLite3 backend and forms the basis for generating structured PDF and JSON reports suitable for audits, compliance, and development workflows.

D. Accessibility and Reporting Interface

To make vulnerability assessment more approachable, HawkEye provides a browser-based interface built using HTML, CSS, and JavaScript. Through this dashboard, users can submit target URLs, initiate scans, and monitor progress in real time. The reporting module automatically generates downloadable PDF summaries that include vulnerability descriptions, affected endpoints, severity levels, and recommended remediation steps. Visual indicators and filter-based navigation further enhance usability for developers and non-technical users.

E. Scalability and Deployment

HawkEye employs Docker-based deployment to ensure modularity, isolation, and portability. Each scanner and service runs within its own container, simplifying updates and preventing interference between components. The architecture also supports integration with CI/CD workflows—such as GitHub Actions—to streamline automated builds, testing, and version management. This containerized approach enables high availability, fault tolerance, and seamless scalability, making it suitable for future enhancements such as AI-driven risk scoring or continuous security monitoring.

F. Summary of the Proposed Framework

By combining multi-tool scanning, standardized reporting, automation, and web accessibility, HawkEye presents a forward-looking solution for modern vulnerability analysis. Its emphasis on modularity and secure orchestration ensures that users can perform comprehensive assessments without deep technical knowledge. Overall, the proposed framework delivers a robust, scalable, and user-friendly environment for conducting effective web application security audits.

IV. METHODOLOGY

The development of HawkEye follows a modular and automation-centric methodology designed to support continuous and reliable web vulnerability assessment. The overall design incorporates principles from modern DevSecOps, microservice-oriented workflows, and automated penetration-testing practices, enabling scalability, maintainability, and secure operation [4],[5]. This section outlines the methodology used to design, implement, and evaluate the HawkEye platform.

A. *Development Approach*

HawkEye was developed using an Agile Software Development Life Cycle (SDLC), enabling incremental evolution of system components and continuous refinement through iterative sprints. Each development cycle focused on a dedicated module—such as scanner integration, result normalization, report generation, or frontend improvements. This iterative approach ensured rapid feedback, modular growth, and smoother integration of new features. The workflow aligns with hybrid analysis methodologies presented in prior research on vulnerability assessment systems [4], supporting flexibility and long-term maintainability.

B. *System Architecture*

The architecture of HawkEye adopts a layered microservice-inspired structure, organizing the system into independent yet interconnected components (Fig. 1). This design enhances modularity and allows individual layers to evolve without affecting the rest of the system [5].

1) Client Layer: Implemented using HTML, CSS, and JavaScript, the frontend enables users to submit target URLs, initiate scans, and monitor scanning progress through an intuitive web interface.
2) API Layer: The Django REST Framework (DRF) acts as the gateway for all backend interactions. It manages request routing, validation, and secure communication among internal services.
3) Scanning Layer: Python-based orchestration modules execute tools such as OWASP ZAP and Nikto, with support for extending to SQLMap, Nmap, and Wapiti. Tools run via Docker containers or subprocess calls, depending on configuration, providing isolation and controlled resource use.
4) Data Layer: SQLite3 serves as the database backend, storing scan results, parsed vulnerability records, metadata, and logs.
5) Reporting Layer: Standardized PDF reports are generated from normalized data, incorporating CWE classification and CVSS scoring to support consistent risk assessment.

This architecture ensures flexibility, ease of deployment, and clear separation of responsibilities across layers.

C. *Scanning and Analysis Pipeline*

HawkEye's vulnerability detection workflow is modeled as a multi-stage pipeline, inspired by hybrid scanning methodologies discussed in prior studies [3],[4]. The pipeline includes:

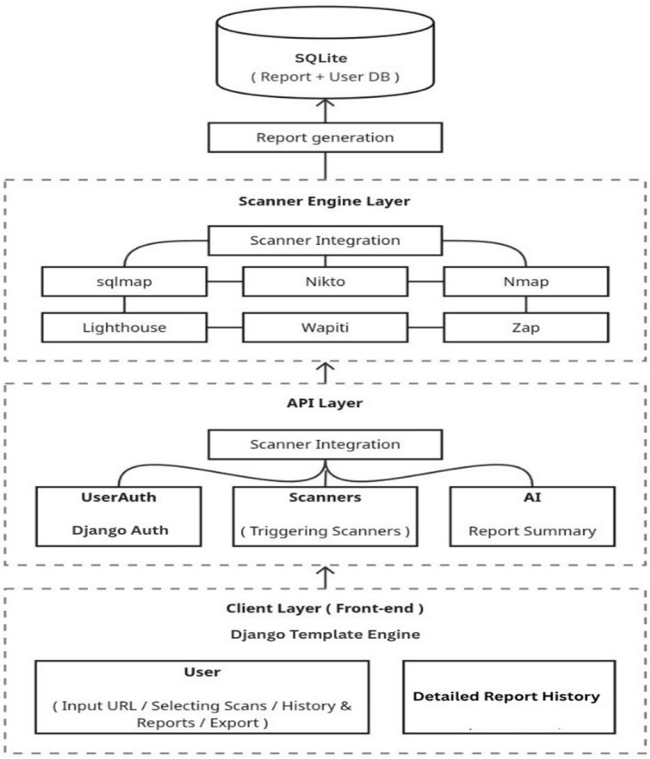


Fig. 1. Architecture Diagram

1) Target Validation: The submitted URL or domain is verified for correctness and scanning eligibility.
2) Tool Execution: The system launches the configured scan-ners—primarily OWASP ZAP and Nikto—using Dockerized or local subprocess execution.
3) Output Parsing: Raw output is extracted to obtain end-points,vulnerability descriptions, affected parameters, and severity indicators.
4) Result Correlation: Parsed results are normalized into a unified schema. Duplicate or overlapping findings across tools are merged to ensure consistency.
5) Report Generation: Consolidated results are transformed into structured PDF reports that include mitigation guidance and CVSS-based severity levels.

This pipeline provides comprehensive and consistent detection aligned with common threat categories described in the OWASP framework [7].

D. Automation and Security Controls

To ensure ethical, secure, and controlled operation, Hawk-Eye integrates several safety and automation features.

The backend uses JWT-based authentication and role-based access control (RBAC) to limit unauthorized usage. Input sanitization prevents command injection or malicious payload execution during scans. Docker sandboxing ensures that each scanning task runs in an isolated container, preventing interference with host resources. Scheduled scanning workflows support periodic assessments for continuous security monitoring. These security practices reflect recommended guidelines in modern cybersecurity frameworks [6], [7].

E. Implementation

Communication between the frontend and Django backend occurs through RESTful APIs, enabling asynchronous updates during active scans. Python's subprocess and threading modules manage scanner execution, while Django ORM handles all database interactions with SQLite3. The ReportLab library is used to generate structured PDF reports that summarize detected vulnerabilities, severity ratings, and remediation recommendations. TLS encryption secures client-server communication, ensuring confidentiality throughout the assessment workflow. Collectively, these implementation choices reinforce modularity, security, and adherence to established secure development standards [7].

F. Testing and Validation

The system undergoes multi-level testing to ensure performance, reliability, and correctness:

1) Unit Testing: Validates individual components, including parser modules and API endpoints.
2) Integration Testing: Ensures correct interaction between Django services and scanning subsystems.
3) System Testing: Assesses the full end-to-end scanning pipeline across supported tools.
4) User Acceptance Testing (UAT): Conducted with students and security practitioners to evaluate usability, clarity of reporting, and platform responsiveness, following methodologies described in modern scanning research [8].

Testing tools such as PyTest, Postman, and Selenium assist in evaluating stability, correctness, and user experience across multiple environments.

G. Deployment

Deployment is handled through a CI/CD workflow based on GitHub Actions, enabling automated builds, validation, and deployment routines. All major components—including scanner modules—run within Docker containers, ensuring consistency, service isolation, and portability. This approach supports scalable deployments and simplifies updates, aligning with the best practices recommended for automated security frameworks [5].

V. CONCLUSION AND FUTURE SCOPE

The rapid expansion of digital services has heightened the exposure of web applications to security threats, placing increased pressure on developers and organizations to maintain strong security practices. Although numerous tools exist for detecting vulnerabilities, their fragmented nature and reliance on technical expertise often limit effectiveness, especially for small teams and educational environments [4], [5]. HawkEye was developed to address these challenges by providing an integrated, automated, and standardized platform for web vulnerability assessment.

HawkEye combines the strengths of widely used open-source scanners—such as OWASP ZAP and Nikto—with a modular architecture that supports future integration of SQLMap, Nmap, and Wapiti. Through its layered design, the platform consolidates key components including scan orchestration, output parsing, severity scoring, and PDF/JSON report generation. By unifying these functionalities into a browser-accessible interface, HawkEye improves coverage across multiple vulnerability types while enabling consistent risk evaluation through CWE and CVSS mappings. This standardized approach ensures that developers, students, and security practitioners can obtain clear, actionable insights without the need for specialized expertise.

A central contribution of HawkEye is its normalization and correlation engine, which resolves the limitations of standalone scanners that often produce disjointed outputs. By merging multi-tool findings, removing redundant entries, and assigning uniform severity scores, HawkEye enhances the reliability and clarity of vulnerability reports. This automation not only accelerates the assessment process but also improves remediation planning by highlighting high-risk issues in a structured format.

Usability is another key focus of the platform. HawkEye's web interface simplifies complex security tasks by enabling users to initiate scans, track scan progress, and download reports directly through a browser. This shift from manual command-line tools to a streamlined dashboard democratizes security testing and encourages broader adoption among institutions and small organizations that lack advanced penetration-testing resources.

From a technical standpoint, HawkEye leverages a Django-based backend, SQLite3 for persistent storage, and Docker for secure and isolated scanner execution. CI/CD integration through GitHub Actions supports automated testing and deployment, ensuring system stability and scalability. Security controls—including authentication, encrypted communication, and sandboxed scanning—further reinforce safe and ethical usage. Experimental validation shows that HawkEye improves detection consistency, reduces duplication of findings, and enhances reporting accuracy compared to using standalone tools independently.

Overall, HawkEye demonstrates how automation, modularity, and standardization can significantly improve web vulnerability assessment workflows. By simplifying the testing process and providing comprehensive, structured reports, the platform offers a practical and scalable solution suitable for developers, educators, and security teams. Looking ahead, HawkEye lays the groundwork for future innovations aimed at enhancing intelligence, adaptability, and integration within modern cybersecurity infrastructures.

VI. FUTURE SCOPE

Although the current prototype effectively demonstrates the feasibility of automated, web-based vulnerability assessment, several avenues remain open for expanding HawkEye's capabilities:

1) AI-Assisted Vulnerability Prioritization: Incorporating machine learning models to evaluate exploitability, contextual severity, and business impact can help prioritize findings and streamline remediation workflow.

2) Predictive Threat Intelligence: Leveraging historical vulnerability patterns and emerging attack trends may enable the system to anticipate potential threats before exploitation occurs.

3) Blockchain-Based Log Integrity: Using blockchain to store scan logs and reports can provide tamper-resistant audit trails, improving transparency, traceability, and trust in the assessment process.

4) Continuous Monitoring and DevSecOps Integration: Embedding HawkEye directly into CI/CD pipelines would enable automated scans during development and deployment cycles, aligning with modern DevSecOps practices.

5) Cloud-Based Distributed Scanning: Scaling the platform across distributed cloud nodes can enhance performance and allow assessment of large enterprise systems with high availability.

6) Advanced Visualization Dashboards: Interactive analyt-ics—such as heat maps, vulnerability timelines, and trend charts—can improve situational awareness and assist security teams in tracking remediation progress.

7) AI-Driven Auto-Remediation Suggestions: Integrating large language models and static code analysis tools can enable automated generation of context-specific fixes and patch recommendations within reports.

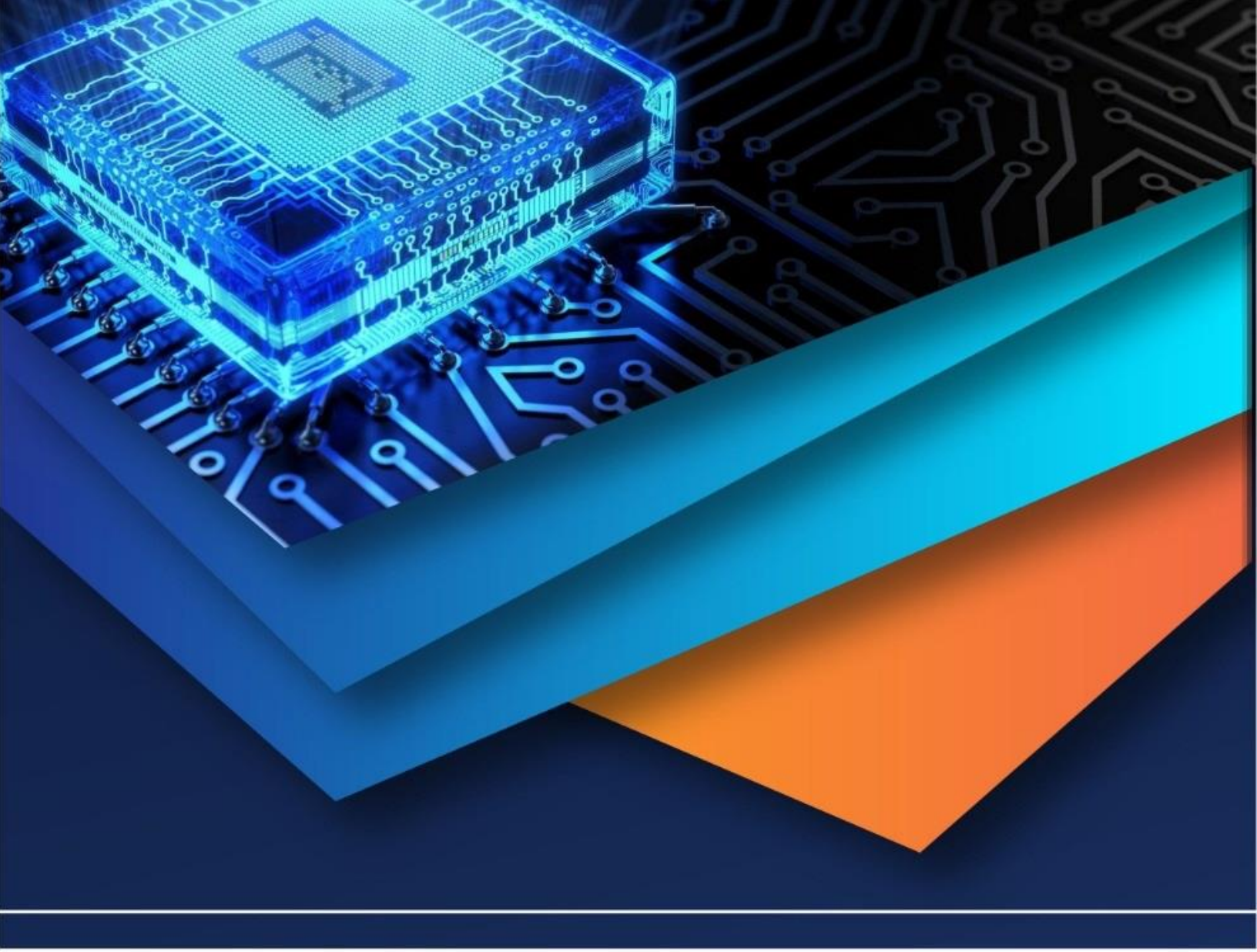
In summary, HawkEye represents a forward-looking approach to web application security auditing, blending automation, accessibility, and standardized analysis within a unified platform. With continued development, the framework has the potential to evolve into a cloud-ready, intelligence-driven ecosystem capable of safeguarding modern applications against emerging cyber threats.

REFERENCES

[1] B. Mburano and W. Si, "Evaluation of Web Vulnerability Scanners Based on OWASP Benchmark," in Proc. 26th Int. Conf. Syst. Eng. (ICSEng), 2018, pp. 1–6

[2] S. Alazmi and D. C. De Leon, "A Systematic Literature Review on the Characteristics and Effectiveness of Web Application Vulnerability Scanners," IEEE Access, vol. 10, pp. 33200–33219, 2022

[3] OWASP Foundation, “OWASP Top Ten,” OWASP Foundation, 2024. [Online]. Available:https://owasp.org/www-project-top-ten/ [Accessed: Aug. 8, 2024].
[4] A. I. Mohaidat and A. Al-Helali, “Web Vulnerability Scanning Tools: A Comprehensive Overview, Selection Guidance, and Cyber Security Recommendations,” Int. J. Res., vol. 10, no. 1, pp. 8–15, 2024.
[5] M. A. Yalc¸inkaya and E. U. Ku¨c¸u¨ksille, “Artificial Intelligence and Dynamic Analysis-Based Web Application Vulnerability Scanner,” ISe-Cure, vol. 16, no. 1, 2024.
[6] D. Desai, H. Baria, A. Chauhan, G. Yadav, M. Patidar, and M. Mahale, “Website Vulnerability Scanning Extension,” IEEE Conf. Publ., 2024. [Online]. Available: https://ieeexplore.ieee.org/document/11074512/.
[7] D. Dumaniya, Y. N. Makawana, and N. D. Bhagchandani, “An Automated Web Vulnerability Scanner for Detecting Common Security Risks,” Int. J. Comput. Techniques, vol. 12, no. 5, pp. 1–8, Sep.–Oct. 2025. [Online]. Available: https://ijctjournal.org/web-vulnerability-scanner-automation/. [Accessed: Oct. 13, 2025].
[8] “Top Vulnerability Scanning Tools 2025,” EscapeTech Blog, Aug. 2025. [Online]. Available: https://escape.tech/blog/top-vulnerability-scanning-tools-2025/. [Accessed: Oct. 13, 2025].



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY