Scientific Research Publishing

Journal of Software Engineering and Applications, 2024, 17, 448-461
https://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

Design & Test of an Advanced Web Security Analysis Tool (AWSAT)

Meenakshi S. P. Manikandaswamy, Vijay Madisetti

School of Cybersecurity and Privacy, Georgia Institute of Technology, Atlanta, USA
Email: meenakshi@gatech.edu, vkm@gatech.edu

How to cite this paper: Manikandaswamy, M.S.P. and Madisetti, V. (2024) Design & Test of an Advanced Web Security Analysis Tool (AWSAT). *Journal of Software Engineering and Applications*, **17**, 448-461.
https://doi.org/10.4236/jsea.2024.175024

Received: April 28, 2024
Accepted: May 28, 2024
Published: May 31, 2024

Abstract

Considering the escalating frequency and sophistication of cyber threats targeting web applications, this paper proposes the development of an automated web security analysis tool to address the accessibility gap for non-security professionals. This paper presents the design and implementation of an automated web security analysis tool, AWSAT, aimed at enabling individuals with limited security expertise to effectively assess and mitigate vulnerabilities in web applications. Leveraging advanced scanning techniques, the tool identifies common threats such as Cross-Site Scripting (XSS), SQL Injection, and Cross-Site Request Forgery (CSRF), providing detailed reports with actionable insights. By integrating sample payloads and reference study links, the tool facilitates informed decision-making in enhancing the security posture of web applications. Through its user-friendly interface and robust functionality, the tool aims to democratize web security practices, empowering a wider audience to proactively safeguard against cyber threats.

Keywords

Web Security, Automated Analysis, Vulnerability Assessment, Web Scanning, Cross-Site Scripting, SQL Injection, Cross-Site Request Forgery

1. Introduction

In an era marked by the escalating frequency and complexity of cyber threats targeting web applications, the imperative for robust web security measures has never been more pronounced. According to projections, cybercrime is expected to reach a staggering $10.5 trillion by 2025 [1], with web application attacks comprising 26% of all breaches [2]. Compounding this challenge is the fact that 60% of organizations are operating with understaffed security teams [3], exacerbating the risks associated with potential security breaches. The average cost of a data breach stands at a daunting $9.48 million [4], underscoring the substantial financial ramifications of inadequate security measures. Against this backdrop, the ubiquitous utilization of web applications, accounting for 66.2% of all web traffic [5], underscores the critical need for accessible and effective web security solutions.

Traditional security tools, while indispensable in fortifying digital defenses, often present a formidable barrier to entry for individuals lacking specialized security expertise. This inherent complexity limits their accessibility, sidelining the active participation of non-security professionals, such as Quality Assurance (QA) specialists and developers, in the security testing process. To address this gap, our paper aims to "shift left" security by integrating security testing earlier into the software development lifecycle. The paper endeavors to democratize web security by developing an intuitive and user-friendly automated web security analysis tool.

The proposed tool aims to empower non-security professionals to efficiently comprehend and evaluate web application vulnerabilities without necessitating extensive security knowledge. By furnishing detailed reports, sample payloads, and reference study links, the tool equips users with the resources needed to bolster the security posture of their web applications comprehensively. Moreover, the tool's emphasis on usability and contextual reporting ensures that even individuals lacking a strong security background can contribute meaningfully to the protection of online assets.

The implementation of the proposed tool leverages the Selenium web driver on Python to accept the target webpage from the user. Subsequently, the tool traverses through the Document Object Model (DOM) elements, scouring for potential attack vectors such as input elements, forms, and more. Equipped with a preset list of payloads, the tool endeavors to execute common exploits including Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and others. The tool provides a quick summary of its findings via the terminal interface, along with a comprehensive report outlining identified vulnerabilities and actionable next steps.

The organization of this paper is structured as follows: Section 2 provides an overview of existing approaches to web security analysis, highlighting their respective limitations. Section 3 introduces the proposed approach, delineating how it addresses the shortcomings of existing methodologies. Section 4 delves into the implementation and testing of the proposed tool, elucidating its architecture, and presenting key results. Section 5 offers a comparative analysis of the proposed tool against prior approaches, showcasing its efficacy in overcoming existing limitations. Finally, Section 6 encapsulates the paper with a summary of findings and concluding remarks, followed by a comprehensive list of references in Section 7.

2. Existing Work

2.1. Evaluation Criteria

We evaluated and analyzed the performance of various tools available in the market. A diverse range of tools, including Owasp ZAP, Burp Suite, Invicti, Qualys WAS, AppSpider, Detectify, Skipfish, Nessus, and OpenVAS, were assessed against multiple criteria, including their web application scanning capabilities, ease of use, level of automation, contextual reporting features, resource efficiency, and cost considerations [6].

2.2. Assessment of Existing Tools

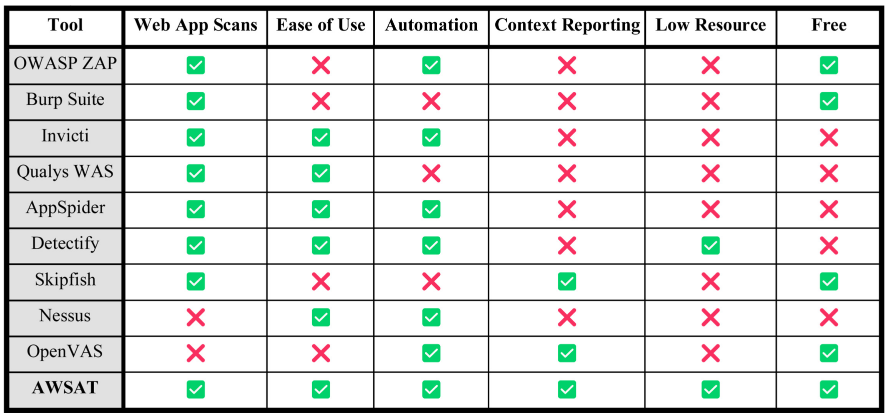
As depicted in **Figure 1**, Owasp ZAP was found to offer robust web application scanning capabilities, albeit lacking in ease of use and contextual reporting features [7]. Similarly, Burp Suite, while renowned for its comprehensive web security testing capabilities, posed challenges in terms of usability and automation [8]. Invicti and Qualys WAS excelled in ease of use but lacked automation and contextual reporting capabilities [9]. AppSpider and Detectify demonstrated commendable automation but fell short in contextual reporting and resource efficiency [10]. Skipfish showcased contextual reporting capabilities but lacked automation and ease of use [11]. Nessus, although user-friendly and highly automated, did not offer contextual reporting features [12]. OpenVAS, despite being open-source and featuring automation and contextual reporting capabilities, was found to be less user-friendly [13].

2.3. Proposed Tool Significance

Our proposed Automated Web Security Analysis Tool (AWSAT) integrates the best features of existing tools while addressing their limitations. AWSAT offers comprehensive web application scanning, user-friendly interface, advanced automation, contextual reporting capabilities, low resource usage, and free accessibility. This holistic approach positions AWSAT as a significant advancement in the field of web security analysis.

3. Proposed Approach

The proposed approach entails the development of an Automated Web Security Analysis Tool (AWSAT) designed to revolutionize web security testing by



Tool	Web App Scans	Ease of Use	Automation	Context Reporting	Low Resource	Free
OWASP ZAP	✅	❌	✅	❌	❌	✅
Burp Suite	✅	❌	❌	❌	❌	✅
Invicti	✅	✅	✅	❌	❌	❌
Qualys WAS	✅	✅	❌	❌	❌	❌
AppSpider	✅	✅	✅	❌	❌	❌
Detectify	✅	✅	✅	❌	✅	❌
Skipfish	✅	❌	❌	✅	❌	✅
Nessus	❌	✅	✅	❌	❌	❌
OpenVAS	❌	❌	✅	✅	❌	✅
AWSAT	✅	✅	✅	✅	✅	✅

Figure 1. Comparison of AWSAT with existing tools.

providing a fast, robust, comprehensive, and user-friendly solution. AWSAT aims to address the limitations of existing approaches by offering a low resource intensive, intuitive, and automated tool that enables users to efficiently understand and mitigate web application vulnerabilities.

Figure 2 block diagram illustrates the workflow of AWSAT, showcasing its seamless operation. Initially, the tool acquires the target URL input by the user, serving as the intuitive starting point for the security analysis process. Utilizing a Selenium web driver, AWSAT dynamically crawls through the webpage, meticulously inspecting DOM elements to identify potential threat vectors. Upon detection, the tool launches targeted exploits on these vectors, executing a series of predefined payloads to assess the susceptibility of the web application to common vulnerabilities. Subsequently, AWSAT compiles its findings into a comprehensive report, detailing the identified vectors, exploit details, and severity assessments. Moreover, the report includes step-by-step walkthroughs to reproduce and mitigate the detected vulnerabilities, ensuring clarity and actionable insights for users. By offering such usability and contextual reporting, AWSAT caters to both technical and non-technical users, facilitating informed decision-making and effective security measures.

Unlike traditional security tools that require significant expertise and manual intervention, AWSAT streamlines the scanning process, making it accessible to individuals with varying levels of security knowledge. By automating vulnerability detection, AWSAT significantly reduces the time and effort required for security testing, allowing users to focus on remediation efforts rather than the intricacies of scanning methodologies.

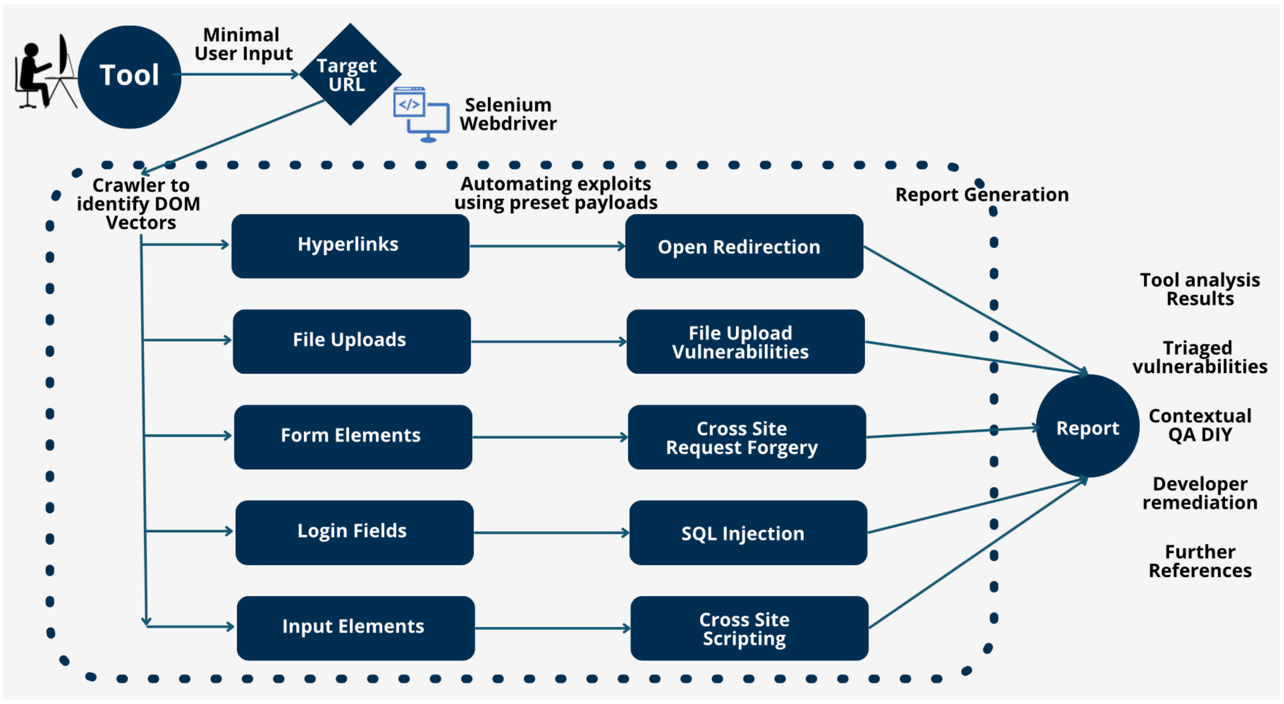


Figure 2. Block diagram of AWSAT.

One key aspect of AWSAT is its emphasis on comprehensive reporting. Traditional security tools often generate complex, technical reports that are challenging for non-security professionals to interpret. In contrast, AWSAT produces clear, actionable reports that provide detailed insights into identified vulnerabilities, including risk severity ratings, affected components, and recommended mitigation steps. By presenting information in a user-friendly format, AWSAT empowers users to understand the nature of security threats and take proactive measures to address them effectively.

Furthermore, AWSAT prioritizes accessibility and inclusivity in web security testing. Many existing tools are designed primarily for security experts, requiring extensive training and expertise to operate effectively. AWSAT, on the other hand, features an intuitive user interface that caters to individuals with limited security backgrounds, democratizing web security testing and enabling a broader audience to contribute to the protection of online assets.

Overall, AWSAT represents a significant advancement in web security testing, offering an automated, user-friendly solution that addresses the limitations of existing approaches. By streamlining the scanning process, providing comprehensive reporting, and prioritizing accessibility, AWSAT empowers users to enhance the security posture of their web applications effectively and efficiently.

4. Implementation and Test of AWSAT

In implementing our proposed approach, we leverage the Selenium web driver on Python to execute our Automated Web Security Analysis Tool (AWSAT). The tool begins by accepting the target webpage URL from the user, initiating a dynamic crawl through the DOM elements of the page to identify potential attack vectors such as input fields, login inputs, forms, file uploads, and hyperlinks.

Using a preset list of payloads, AWSAT then systematically launches common exploits including Cross-Site Scripting (XSS), SQL Injection (SQLi), Cross-Site Request Forgery (CSRF), file upload vulnerabilities, and open redirection vulnerabilities. Upon completion, the tool provides users with a quick summary of the analysis in the terminal, highlighting detected attack vectors and successful exploits.

Additionally, AWSAT generates a comprehensive report detailing each attempted exploit, including a brief description, the steps taken by the tool, findings, severity assessment, and guided next steps for remediation. This reporting structure aims to empower both technical and non-technical users to understand and mitigate identified vulnerabilities effectively.

To ensure the effectiveness of AWSAT, iterative testing and continuous improvements were needed. One significant hurdle encountered was the scarcity of authentic testing environments closely mirroring real-world scenarios. Many testing platforms necessitated the deployment of virtual machines or the establishment of personal server sandboxes, complicating the testing process. Moreover, certain platforms relied heavily on iFrames, which failed to accurately replicate actual sites, thereby hindering the tool's evaluation under realistic conditions.

Despite these obstacles, AWSAT persevered, ultimately undergoing rigorous testing across a diverse array of websites known to harbor vulnerabilities as listed below. This proactive approach sought to ensure a multifaceted evaluation of the tool's capabilities.

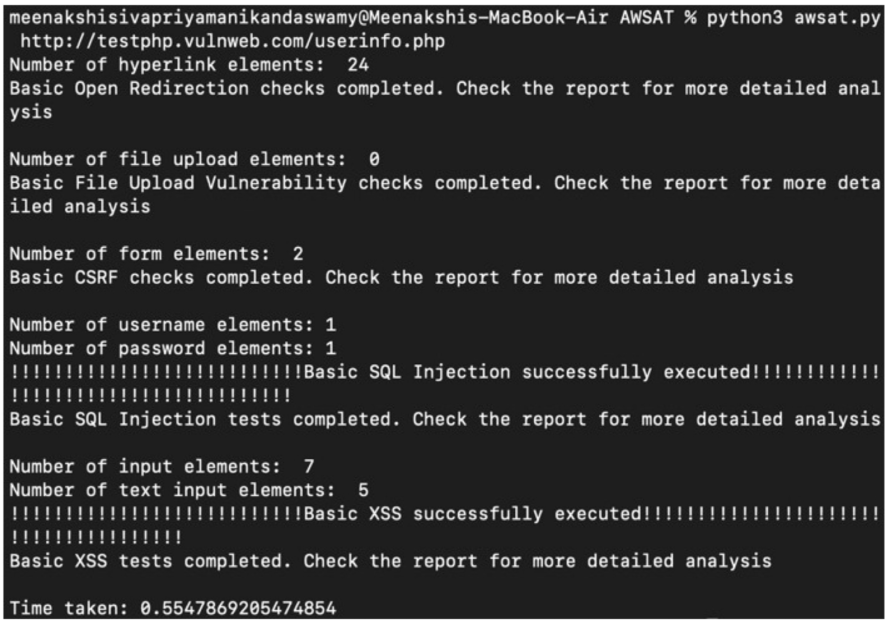
1) https://xss-quiz.int21h.jp
2) http://sudo.co.il/xss/level0.php
3) http://www.xssgame.com/f/m4KKGHi2rVUN/
4) https://demo.testfire.net/login.jsp
5) http://testphp.vulnweb.com/userinfo.php
6) https://xss-game.appspot.com/level1/frame

The insights gained from these tests have been instrumental in refining the tool's capabilities, enhancing its efficiency in vulnerability identification and exploitation. Through this iterative process, AWSAT continues to evolve, ensuring its effectiveness and reliability in real-world web application security assessments.

Figure 3 shows a representative output generated from the tool execution on a susceptible website. The illustration provides a succinct overview of the identified threat vectors and the successful exploitation of associated vulnerabilities.

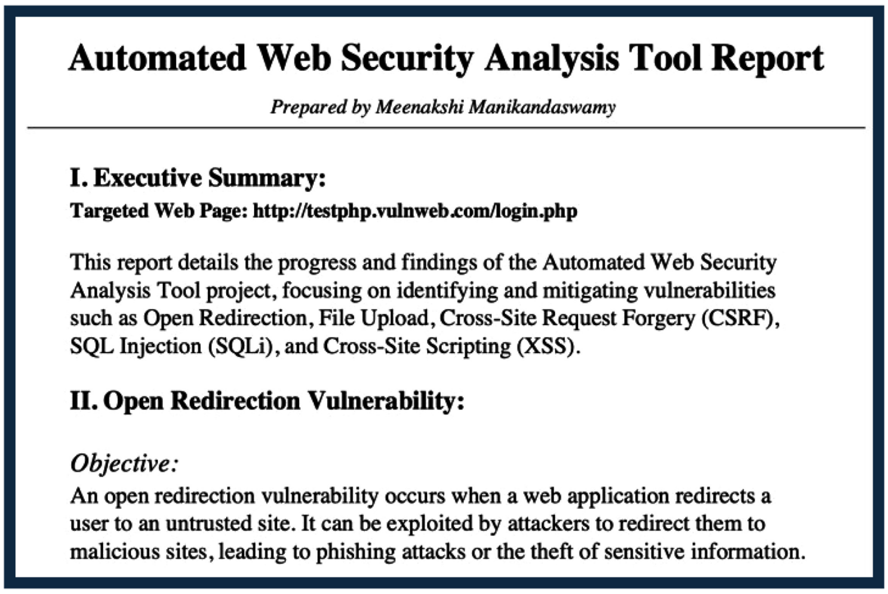
Figure 4 depicts the preliminary layout of the comprehensive report generated by the tool. It encompasses an executive summary detailing the report's scope and content, along with the URL specific to the website subjected to testing.

Figure 5 illustrates the structure of each subsection within the report, with each subsection dedicated to a distinct vulnerability detected and exploited.



```
meenakshisivapriyamanikandaswamy@Meenakshis-MacBook-Air AWSAT % python3 awsat.py
 http://testphp.vulnweb.com/userinfo.php
Number of hyperlink elements:  24
Basic Open Redirection checks completed. Check the report for more detailed anal
ysis

Number of file upload elements:  0
Basic File Upload Vulnerability checks completed. Check the report for more deta
iled analysis

Number of form elements:  2
Basic CSRF checks completed. Check the report for more detailed analysis

Number of username elements: 1
Number of password elements: 1
!!!!!!!!!!!!!!!!!!!!!!!!!!Basic SQL Injection successfully executed!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!
Basic SQL Injection tests completed. Check the report for more detailed analysis

Number of input elements:  7
Number of text input elements:  5
!!!!!!!!!!!!!!!!!!!!!!!!!!Basic XSS successfully executed!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!
Basic XSS tests completed. Check the report for more detailed analysis

Time taken: 0.5547869205474854
```

Figure 3. Terminal Output summary from a run.



Automated Web Security Analysis Tool Report

Prepared by Meenakshi Manikandaswamy

I. Executive Summary:

Targeted Web Page: http://testphp.vulnweb.com/login.php

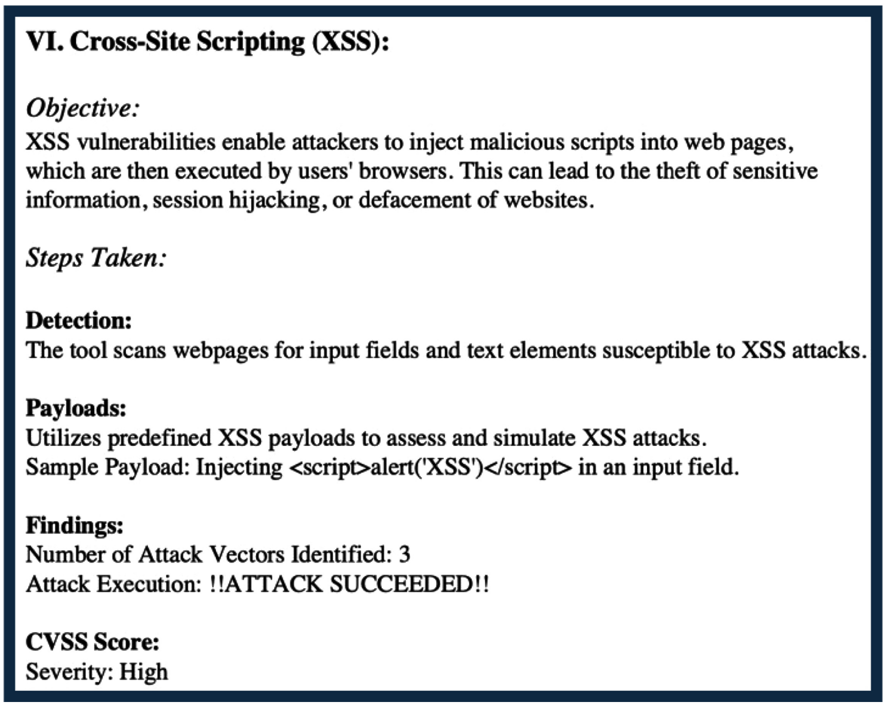
This report details the progress and findings of the Automated Web Security Analysis Tool project, focusing on identifying and mitigating vulnerabilities such as Open Redirection, File Upload, Cross-Site Request Forgery (CSRF), SQL Injection (SQLi), and Cross-Site Scripting (XSS).

II. Open Redirection Vulnerability:

Objective:

An open redirection vulnerability occurs when a web application redirects a user to an untrusted site. It can be exploited by attackers to redirect them to malicious sites, leading to phishing attacks or the theft of sensitive information.

Figure 4. Basic Structure of the AWSAT's report.



VI. Cross-Site Scripting (XSS):

Objective:

XSS vulnerabilities enable attackers to inject malicious scripts into web pages, which are then executed by users' browsers. This can lead to the theft of sensitive information, session hijacking, or defacement of websites.

Steps Taken:

Detection:

The tool scans webpages for input fields and text elements susceptible to XSS attacks.

Payloads:

Utilizes predefined XSS payloads to assess and simulate XSS attacks.
Sample Payload: Injecting <script>alert('XSS')</script> in an input field.

Findings:

Number of Attack Vectors Identified: 3
Attack Execution: !!ATTACK SUCCEEDED!!

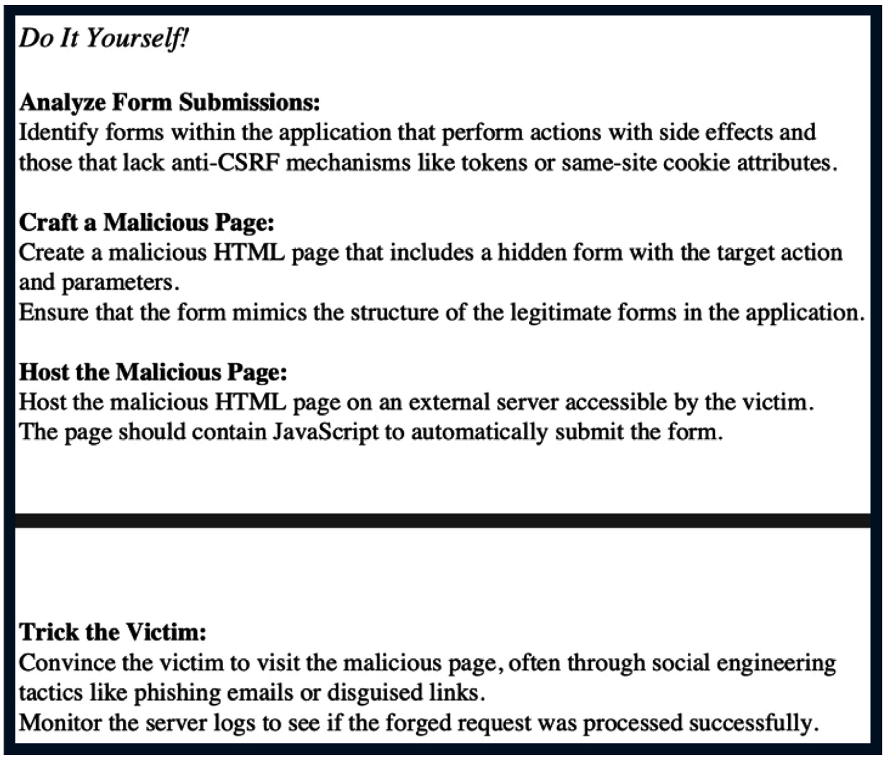
CVSS Score:

Severity: High

Figure 5. Details of AWSAT actions on each vulnerability.

Within each section, a concise overview of the attack is provided, detailing the method of detection by the tool, the payload responsible for successful execution, the tool's findings, and the Common Vulnerability Scoring System (CVSS) score indicative of the exploit's severity.

Figure 6 depicts the step-by-step process for reproducing each vulnerability. It offers a comprehensive guide on configuring the environment to craft the exploit and subsequently exploit the identified vulnerability. This guide is designed


Do It Yourself!

Analyze Form Submissions:
Identify forms within the application that perform actions with side effects and those that lack anti-CSRF mechanisms like tokens or same-site cookie attributes.

Craft a Malicious Page:
Create a malicious HTML page that includes a hidden form with the target action and parameters.
Ensure that the form mimics the structure of the legitimate forms in the application.

Host the Malicious Page:
Host the malicious HTML page on an external server accessible by the victim.
The page should contain JavaScript to automatically submit the form.

Trick the Victim:
Convince the victim to visit the malicious page, often through social engineering tactics like phishing emails or disguised links.
Monitor the server logs to see if the forged request was processed successfully.

Figure 6. Steps to reproduce the attack.

to be highly inclusive, employing straightforward language to ensure readability and execution accessibility for individuals, regardless of their familiarity with security concepts.

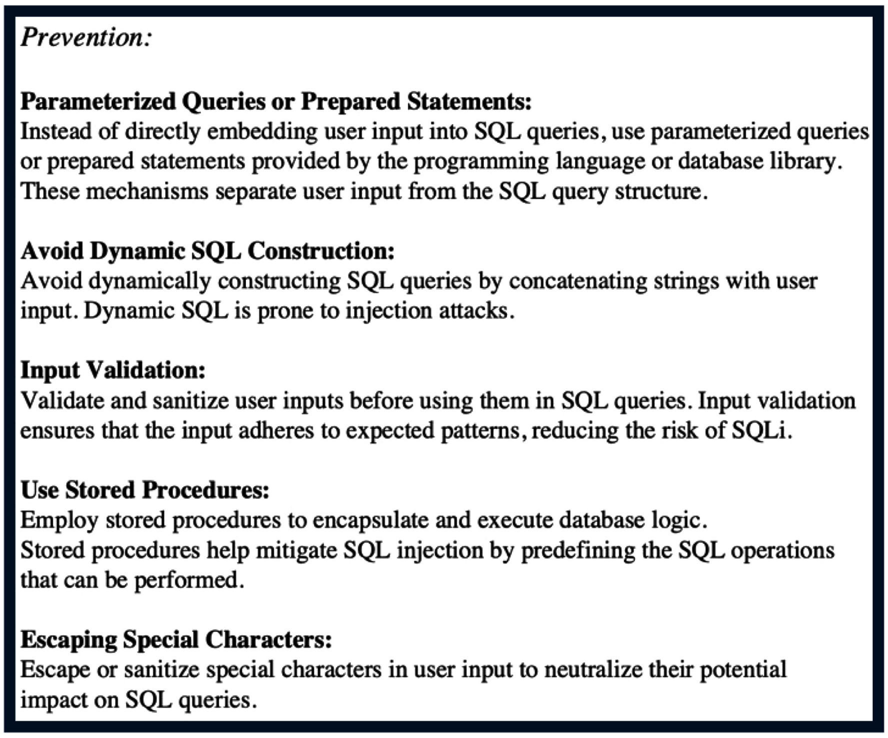
In Figure 7, the report presents mitigation recommendations in easily understandable terms, catering to individuals with varying levels of security expertise. The recommendations offer multiple avenues for mitigation, allowing organizations to tailor their security measures according to their specific use case and desired level of protection.

In Figure 8, the references section is depicted, which is present under each vulnerability entry. This section provides users with additional resources for delving deeper into the intricacies of the vulnerability. Additionally, a comprehensive conclusion section is included at the end of the report to summarize key findings and insights gleaned from the analysis.

Thus, during the implementation and testing phase, significant progress was made in enhancing the functionality and reliability of the Automated Web Security Analysis Tool. Despite encountering challenges related to the scarcity of suitable testing environments, the tool was successfully tested on diverse websites. These tests yielded valuable feedback, enabling iterative improvements to the tool’s efficiency and effectiveness in identifying and mitigating vulnerabilities in real-world web applications.

5. Comparison with Prior Work

During the evaluation phase, the focus was on comparing the performance of



Prevention:

Parameterized Queries or Prepared Statements:
Instead of directly embedding user input into SQL queries, use parameterized queries or prepared statements provided by the programming language or database library. These mechanisms separate user input from the SQL query structure.

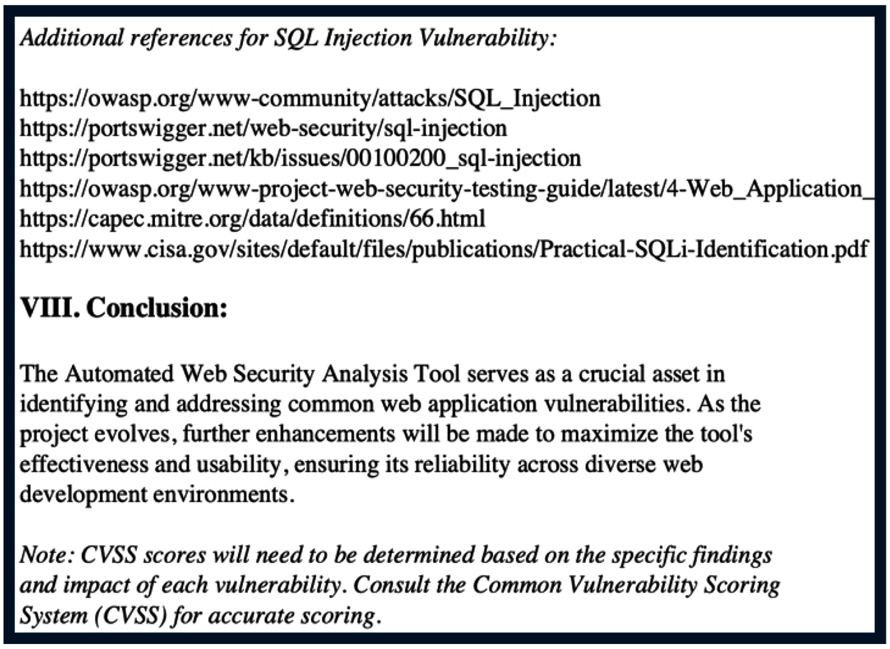
Avoid Dynamic SQL Construction:
Avoid dynamically constructing SQL queries by concatenating strings with user input. Dynamic SQL is prone to injection attacks.

Input Validation:
Validate and sanitize user inputs before using them in SQL queries. Input validation ensures that the input adheres to expected patterns, reducing the risk of SQLi.

Use Stored Procedures:
Employ stored procedures to encapsulate and execute database logic.
Stored procedures help mitigate SQL injection by predefining the SQL operations that can be performed.

Escaping Special Characters:
Escape or sanitize special characters in user input to neutralize their potential impact on SQL queries.

Figure 7. Mitigation recommendations.



Additional references for SQL Injection Vulnerability:

https://owasp.org/www-community/attacks/SQL_Injection
https://portswigger.net/web-security/sql-injection
https://portswigger.net/kb/issues/00100200_sql-injection
https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_
https://capec.mitre.org/data/definitions/66.html
https://www.cisa.gov/sites/default/files/publications/Practical-SQLi-Identification.pdf

VIII. Conclusion:

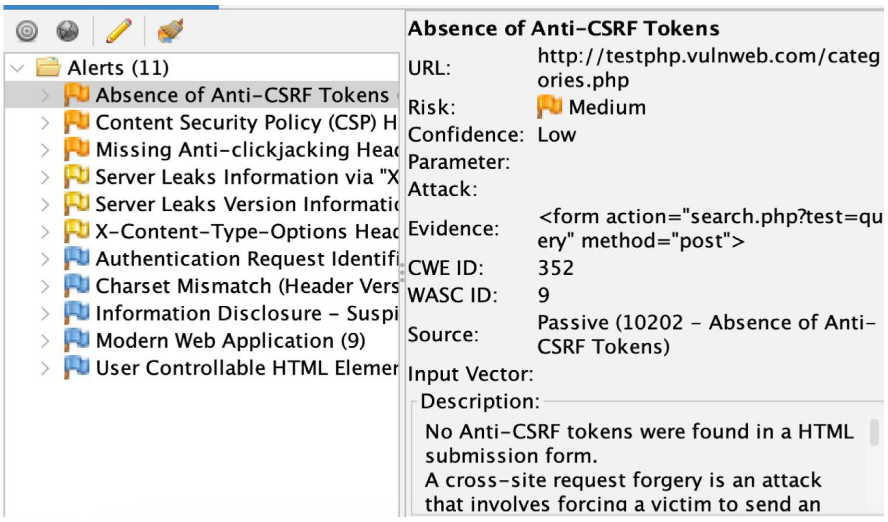
The Automated Web Security Analysis Tool serves as a crucial asset in identifying and addressing common web application vulnerabilities. As the project evolves, further enhancements will be made to maximize the tool's effectiveness and usability, ensuring its reliability across diverse web development environments.

Note: CVSS scores will need to be determined based on the specific findings and impact of each vulnerability. Consult the Common Vulnerability Scoring System (CVSS) for accurate scoring.

Figure 8. References and conclusion.

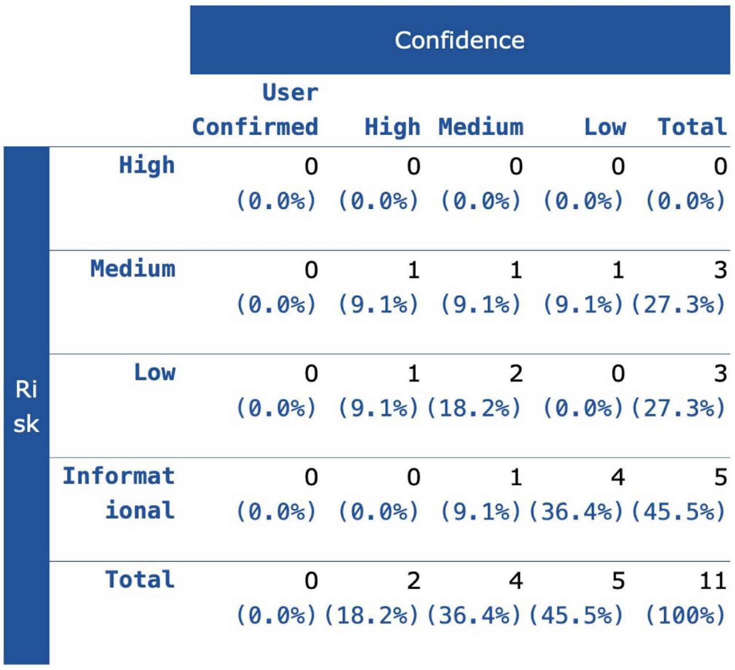
existing market tools against the Automated Web Security Analysis Tool (AWSAT) on the same set of websites used in testing. OpenVAS, primarily designed as a system and network scanner, struggled to address web security vulnerabilities effectively. Its setup was challenging, requiring users to navigate through complex configurations, and the generated reports lacked actionable insights for users. Despite offering manual penetration testing options, OpenVAS lacked automated web security analysis capabilities, which limited its effectiveness in identifying and mitigating web-based threats.

Our analysis of OWASP ZAP is shown in Figure 9 and Figure 10, and of BurpSuite in Figures 11-13. Similarly, while the free version of BurpSuite demonstrated robustness in web security testing, its complexity and lack of automated scanning in the free version posed usability challenges. Setting up BurpSuite proved to be notably more complex due to its various components like the proxy, target, intruder, and repeater. The free version's limitations hindered its usability, particularly for users without extensive security knowledge.



```
Alerts (11)
  Absence of Anti-CSRF Tokens
  Content Security Policy (CSP) H
  Missing Anti-clickjacking Head
  Server Leaks Information via "X
  Server Leaks Version Informati
  X-Content-Type-Options Head
  Authentication Request Identifi
  Charset Mismatch (Header Vers
  Information Disclosure - Suspi
  Modern Web Application (9)
  User Controllable HTML Elemer

Absence of Anti-CSRF Tokens
URL:           http://testphp.vulnweb.com/categories.php
Risk:          Medium
Confidence:    Low
Parameter:
Attack:
Evidence:      <form action="search.php?test=query" method="post">
CWE ID:        352
WASC ID:       9
Source:        Passive (10202 - Absence of Anti-CSRF Tokens)
Input Vector:
Description:
No Anti-CSRF tokens were found in a HTML submission form.
A cross-site request forgery is an attack that involves forcing a victim to send an
```

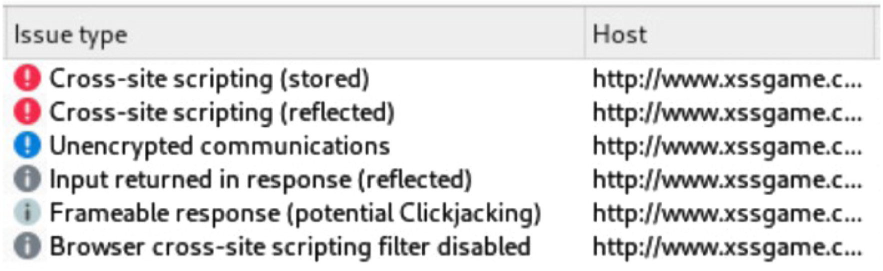
Figure 9. Output from OWASP ZAP.



		Confidence				
		User Confirmed	High	Medium	Low	Total
Risk	High	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
	Medium	0 (0.0%)	1 (9.1%)	1 (9.1%)	1 (9.1%)	3 (27.3%)
	Low	0 (0.0%)	1 (9.1%)	2 (18.2%)	0 (0.0%)	3 (27.3%)
	Informational	0 (0.0%)	0 (0.0%)	1 (9.1%)	4 (36.4%)	5 (45.5%)
	Total	0 (0.0%)	2 (18.2%)	4 (36.4%)	5 (45.5%)	11 (100%)

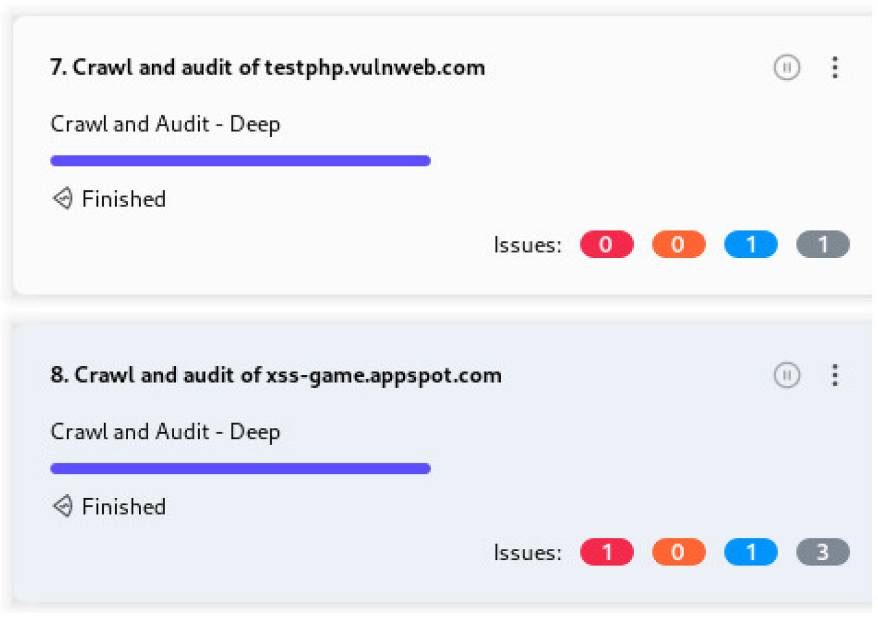
Figure 10. Triaging output from OWASP ZAP.

ZAP, another widely used tool, exhibited promising results during evaluation but had notable limitations compared to AWSAT. Although relatively easy to set up and use, ZAP was slower in identifying vulnerabilities compared to AWSAT and lacked contextual report generation. Additionally, ZAP consumed considerable resources during operation, leading to performance issues, particularly on more vulnerable sites. AWSAT's streamlined architecture and efficient resource utilization mitigate these issues, ensuring faster scanning speeds and smoother operation across various web applications.



Issue type	Host
Cross-site scripting (stored)	http://www.xssgame.c...
Cross-site scripting (reflected)	http://www.xssgame.c...
Unencrypted communications	http://www.xssgame.c...
Input returned in response (reflected)	http://www.xssgame.c...
Frameable response (potential Clickjacking)	http://www.xssgame.c...
Browser cross-site scripting filter disabled	http://www.xssgame.c...

Figure 11. Scan output from BurpSuite premium.



7. Crawl and audit of testphp.vulnweb.com

Crawl and Audit - Deep

Finished

Issues: 0 0 1 1

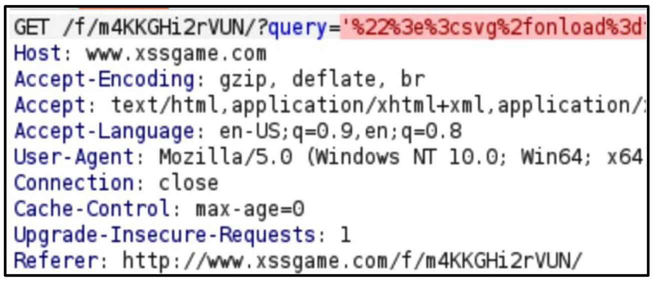
8. Crawl and audit of xss-game.appspot.com

Crawl and Audit - Deep

Finished

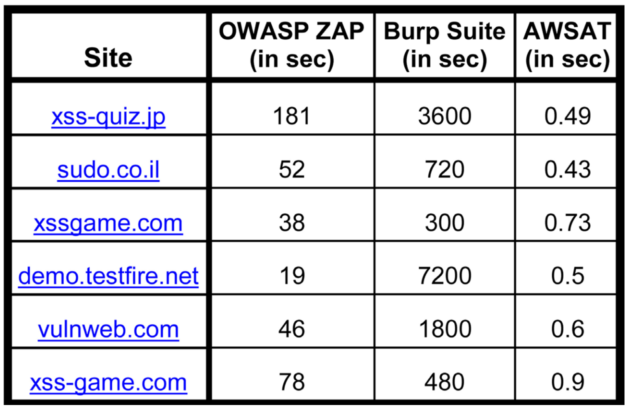
Issues: 1 0 1 3

Figure 12. Overall scan results from BurpSuite premium.



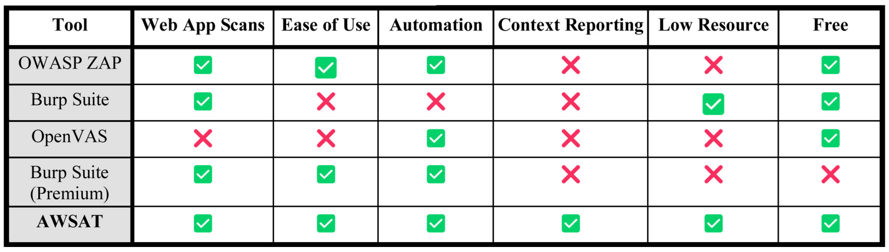
```
GET /f/m4KKGHi2rVUN/?query='%22%3e%3csvg%2fonload%3d
Host: www.xssgame.com
Accept-Encoding: gzip, deflate, br
Accept: text/html,application/xhtml+xml,application/
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64
Connection: close
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Referer: http://www.xssgame.com/f/m4KKGHi2rVUN/
```

Figure 13. Succeeded Payload from BurpSuite Premium.



Site	OWASP ZAP (in sec)	Burp Suite (in sec)	AWSAT (in sec)
xss-quiz.jp	181	3600	0.49
sudo.co.il	52	720	0.43
xssgame.com	38	300	0.73
demo.testfire.net	19	7200	0.5
vulnweb.com	46	1800	0.6
xss-game.com	78	480	0.9

Figure 14. Speed comparison.



Tool	Web App Scans	Ease of Use	Automation	Context Reporting	Low Resource	Free
OWASP ZAP	✓	✓	✓	✗	✗	✓
Burp Suite	✓	✗	✗	✗	✓	✓
OpenVAS	✗	✗	✓	✗	✗	✓
Burp Suite (Premium)	✓	✓	✓	✗	✗	✗
AWSAT	✓	✓	✓	✓	✓	✓

Figure 15. Market comparison with existing tools and desired metrics practically.

Notably, testing on the paid version of Burp Suite's automated scanning tool revealed slower speeds and occasional missed exploits. By leveraging Selenium over Python, AWSAT streamlines the web scanning process, making it faster and more accessible to users with limited security backgrounds. These comparative analyses underscore AWSAT's advancements in efficiency and usability over existing market tools.

The speed comparison in **Figure 14** and market comparison in **Figure 15** reveal AWSAT's remarkable efficiency compared to industry-leading tools like OWASP ZAP and BurpSuite, both in the free and paid segments. While the latter tools may excel in identifying a higher number of low-severity issues, AWSAT is faster than them in detecting high-severity vulnerabilities at a remarkable response time, often surpassing them by a factor of 100. Additionally, AWSAT demonstrates superior reporting capabilities, providing users with comprehensive insights and actionable recommendations.

6. Summary and Conclusions

The development of the Automated Web Security Analysis Tool (AWSAT) represents a significant milestone in the realm of web application security. The journey from inception to implementation has been guided by a meticulous approach encompassing thorough market analysis, inclusive design, efficient implementation, iterative testing, and comparative evaluation against existing tools.

The market analysis phase served as the foundation for AWSAT's development, offering deep insights into the landscape of automated security tools. This comprehensive examination involved scrutinizing various tools available in the market, identifying their strengths, weaknesses, and areas for improvement. By leveraging this detailed understanding, AWSAT was conceptualized to address the inherent limitations of existing solutions while anticipating the evolving needs of web security.

The design phase of AWSAT was characterized by a commitment to accessibility and inclusivity. The user centered design approach ensured that the tool caters to a diverse audience, including individuals with limited security expertise. Through intuitive interfaces and contextual reporting features, AWSAT aims to empower users of all backgrounds to conduct effective web security analysis.

Efficient implementation using Selenium with Python was instrumental in ensuring that AWSAT remains lightweight and resource-efficient while delivering robust security analysis capabilities. The choice of technology not only streamlines the scanning process but also enhances the tool's scalability and compatibility across different web environments.

The iterative testing phase of AWSAT involved meticulous evaluation on a diverse set of websites closely resembling real-world scenarios. This rigorous testing approach enabled the refinement of AWSAT's capabilities, ensuring its effectiveness in identifying and mitigating vulnerabilities across various web applications.

Comparative evaluation against existing market tools provided valuable insights into AWSAT's competitive edge. By benchmarking performance against established metrics, AWSAT demonstrated superior efficiency, usability, and effectiveness in web security analysis. This comparative analysis underscored AWSAT's potential to set a new standard for automated web security tools.

In conclusion, AWSAT represents a culmination of careful market analysis, inclusive design, efficient implementation, iterative testing, and comparative evaluation. With its comprehensive feature set, intuitive interface, and robust security analysis capabilities, AWSAT empowers a broader range of users in enhancing the security posture of their web applications. For those interested in exploring further, the AWSAT repository is available on GitHub: AWSAT GitHub Repository.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] https://www.cobalt.io/blog/cybersecurity-statistics-2024

[2] https://expertinsights.com/insights/50-web-security-stats-you-should-know

[3] https://www.ponemon.org/research/ponemon-library/security/reducing-enterprise-application-security-risks-more-work-needs-to-be-done.html

[4] https://www.statista.com/statistics/273575/us-average-cost-incurred-by-a-data-breach/

[5] https://www.statista.com/statistics/617136/digital-population-worldwide

[6] Daud, N.I., Bakar, K.A.A. and Hasan, M.S.M. (2014) A Case Study on Web Application Vulnerability Scanning Tools. 2014 *Science and Information Conference*, London, 27-29 August 2014, 595-600. https://doi.org/10.1109/SAI.2014.6918247

[7] Alzahrani, A., Alqazzaz, A., Zhu, Y., Fu, H. and Almashfi, N. (2017) Web Application Security Tools Analysis. In 2017 *IEEE 3rd International Conference on Big Data Security on Cloud*, Beijing, 26-28 May 2017, 237-242. https://doi.org/10.1109/BigDataSecurity.2017.47

[8] Curphey, M. and Arawo, R. (2006) Web Application Security Assessment Tools. *IEEE Security & Privacy*, **4**, 32-41. https://doi.org/10.1109/MSP.2006.108

[9] Mohammed, R. (2016) Assessment of Web Scanner Tools. *International Journal of Computer Applications*, **133**, 1-4. https://doi.org/10.5120/ijca2016907794

[10] Dukes, L., Yuan, X. and Akowuah, F. (2013) A Case Study on Web Application Security Testing with Tools and Manual Testing. In 2013 *Proceedings of IEEE Southeastcon*, Jacksonville, 4-7 April 2013, 1-6. https://doi.org/10.1109/SECON.2013.6567420

[11] Wakhale, A. (2018) Web Application Vulnerability Assessment Tools Analysis. UMBC Student Collection.

[12] Joshi, C. and Singh, U.K. (2016) Performance Evaluation of Web Application Security Scanners for More Effective Defense. *International Journal of Scientific and Research Publications* (*IJSRP*), **6**, 660-667.

[13] http://sectooladdict.blogspot.com/2017/11/wavsep-2017-evaluating-dast-against.html