Guarding the Gateways: Web Vulnerability Detection

1st Vibiksha Bharathi P K
(Department of Computer Technology)
(Madras Institute of Technology)
pkvibiksha@gmail.com

2nd Jincy J S
(Department of Computer Technology)
(Madras Institute of Technology)
jincyjs2003@gmail.com

3rd Priyasahaana M
(Department of Computer Technology)
(Madras Institute of Technology)
mspd3847@gmail.com

4th Nandhini V M
(Department of Computer Technology)
(Madras Institute of Technology)
nandhusuba621@gmail.com

***Abstract*—This study introduces an automated vulnerability detection script for web applications, employing Python along with various libraries such as 'requests' and 'BeautifulSoup". The script encompasses robust mechanisms for detecting SQL Injection, XXE (XML External Entity), XSS (Cross-Site Scripting), and SSRF (Server Side Request Forgery) vulnerabilities. Distinct features of this script include innovative caching techniques for optimizing repeated requests and efficient logging capabilities. Caching enhances the scan speed by storing previous responses and avoiding redundant requests, while logging meticulously records scan details such as detected vulnerabilities, affected URLs, payload specifics, and response times.The methodology involves systematic scanning processes that include payload generation for SQL Injection, asynchronous testing of diverse vulnerabilities, and a thorough log-based analysis. The script meticulously examines web forms, submits tailored payloads, and scrutinizes responses to identify potential security weaknesses.**

Overall, this comprehensive tool not only acts as an automated vulnerability scanner but also provides a detailed log file, enabling in-depth analysis of security posture, aiding in proactive security measures, and guiding effective risk mitigation strategies.

***Index Terms*—Automated Vulnerability Detection, Web Application Security, SQL Injection, XXE Attack, XSS Attack, SSRF Attack, Security Assessment, Cache, Logging.**

I. Introduction

Web applications are indispensable in today's digital landscape, offering convenience across e-commerce, social media, and enterprise systems. However, their susceptibility to security vulnerabilities necessitates robust solutions. This paper presents an automated tool designed to detect and mitigate common web application vulnerabilities.

The tool's automation ensures efficient identification and mitigation of security risks, overcoming the challenges of manual testing in rapidly evolving web environments. In addition to its automated capabilities, the tool integrates logging and caching features. Logging records detailed scan activities, providing a comprehensive account for post-scan analysis. Caching optimizes scanning efficiency by reducing redundant requests, contributing to faster scan times.

Emphasizing form analysis, payload injection, and HTTP response analysis, the tool offers versatility, catering to specific web application needs. The subsequent sections delve into the tool's architecture, functionality, and results, offering a comprehensive guide for securing web applications effectively.

II. Related Work

Web application security has been increasingly threatened by SQL injection attacks, a top concern per OWASP. These attacks continuously evolve, posing risks like data breaches and site disruptions. A novel approach presented in [1] offers an adaptive deep forest-based method to detect complex SQL injection attacks. It optimizes the model by enhancing deep forest structures and introducing an AdaBoost-based deep forest model, allowing dynamic feature weight assignment during training.

The persistent threat of Cross-Site Scripting (XSS) attacks is addressed on web applications. A multi-pronged approach, utilizing machine learning algorithms for XSS detection and classification is employed in [2]. Additionally, Content Security Policy (CSP) is introduced for real-time detection, reporting promising results. The critical issue of XML eXternal Entity (XXE) injection, a well-recognized vulnerability that poses a significant threat to web security is addressed isn [3]. While much of the existing literature has focused on vulnerability testing for XXE, this research stands out for its emphasis on prevention strategies.

SQL Injection is addressed in [4], a prevalent cyberattack that jeopardizes web application security. They focus on using machine learning to detect SQL Injection, achieving an impressive accuracy of up to 99.6 percent with various models. The insights into safeguarding web applications from threat are provided, with Logistic Regression emerging as the top-performing model.

The critical domain of Server-Side Request Forgery (SSRF) attacks, a security vulnerability that poses a significant threat to web applications are addressed in [5]. Exploration of various SSRF attack types, the importance of securing web applications, and the use of machine learning techniques for

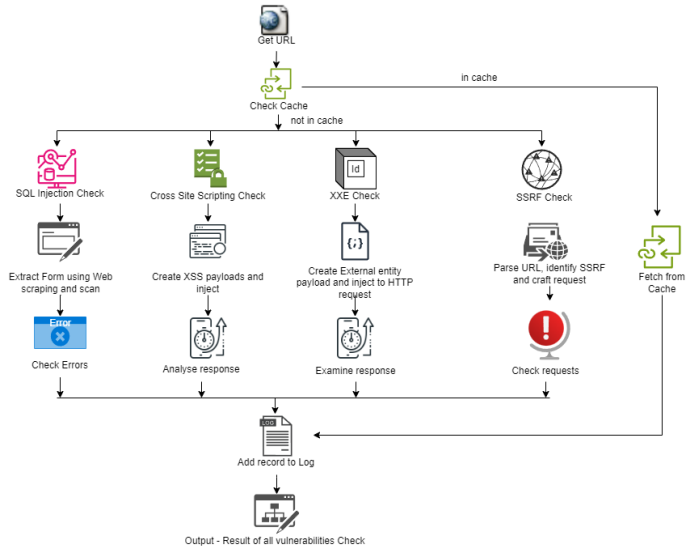

Fig. 1. WEB VULNERABILITY DETECTION

detection and mitigation are done.

The critical concern of Cross-Site Scripting (XSS) attacks in web applications, a pervasive threat to cybersecurity are addressed in [6]. Their research employs machine learning classifiers to detect XSS attacks, focusing on two key features: urls and JavaScript. The prevalent and critical issue of Cross-Site Scripting (XSS) attacks in web applications, a pervasive cybersecurity concern are discussed in [7].By conducting experiments on a dataset of real-world XSS attacks and non-attack web requests, a high degree of accuracy in identifying XSS attacks, outperforming several baseline approaches are demonstrated.

Countering SQL injection attacks, a top web security threat according to OWASP is focussed in [8]. The adaptive deep forest-based method offers an innovative solution to the challenges posed by diverse attack loads and methods. The contribution to the evolving field of web security, providing a robust defense against complex SQL injection attacks is done.

An overview of the SQL injection attack and identify the various attack sources, targets, and types, as well as classify the newly proposed detection and prevention solutions are provided. They discuss and categorize the most interesting and recent proposed mitigation solutions, especially those based on ontology and machine learning[9].

A machine learning algorithm that detects not only SQL injection attacks but also unauthorised users by keeping an audit record is developed [10]. A suggestion for a heuristic algorithm based on machine learning to avoid SQL injection attacks is given. To train and evaluate machine learning classifiers,a dataset of different SQL statements is used[11]. Pattern matching, which can be implemented using the machine learning paradigm, is capable of mitigating SQLIA requests via login, url, and search. If required precautions are not taken, machine learning algorithm selection for model building may be arbitrary and biased[12].

The detection of FDI attacks in a novel way uses Autoencoders. The advantage of sensor data similarity in time and space, which can assist in the identification of falsified data is taken. Furthermore, using the denoising audio encoder, the falsified data is cleaned[13].

Machine learning learns from previous attacks and blocks or bypasses the Web Application Firewall based on the previous performance. The SQL Injections and Phishing flaws, and how to prevent attackers from easily deceiving users are focussed[14].

III. PROPOSED WORK

In this section, we present our approach for comprehensive vulnerability detection in web applications, which seamlessly integrates caching into the detection process, improving efficiency without sacrificing the depth of security assessment.

A. Algorithm Integration

Our approach combines two essential algorithms, as follows:

Algorithm 1: Vulnerability Detection with Caching

Input: url to be tested
Input: XML payload for XXE test
Output: Detection results for SQL injection, XXE, XSS, and SSRF vulnerabilities

if *cache file* CACHE_FILE() *exists* **then**
  cache() ← load cache data from CACHE_FILE()
end
if *url is in* cache() **then**
  `cached_request(`*url*`)`
end
else
  Fetch the response from the requests package
end
`test_sqlInjection_vulnerability(`*url*`)`
`test_xxe_vulnerabilities(`*url, XML payload*`)`
`test_xss_vulnerabilities(`*url*`)`
`test_ssrf_vulnerabilities(`*url*`)`

1) Algorithm 1: Vulnerability Detection with Caching: Algorithm 1 focuses on enhancing the efficiency of vulnerability detection. It leverages caching mechanisms to optimize resource usage and reduce redundant requests. This algorithm is an integral part of our approach and involves the following key steps:

The caching algorithm efficiently uses cached data, if available, to minimize redundant data retrieval. Subsequently, it fetches web page content, optimizing network usage. The algorithm rigorously tests for SQL injection, XXE, XSS, and SSRF vulnerabilities, covering various potential security threats in web applications.

Algorithm 2: Vulnerability Detection

```
Data: url, XML payload
Result: SQL Injection, XXE, XSS, and SSRF
        Detection
Initialize XML payload;
Retrieve web page content from the provided url;
Parse web page content;
Identify forms in the web page;
for each form in forms do
    Extract form action url;
    if valid POST form then
        Construct input field payloads;
        for each payload in payloads do
            Send POST request with payload;
            Check for SQL injection;
            if SQL injection found then
                SQL Injection Vulnerability Detected;
            end
        end
    end
end
Send POST request with XML payload for XXE
 detection;
Check for XXE indications;
if XXE indications found then
    XXE Vulnerability Detected;
end
else
    No XXE Vulnerability Detected;
end
Construct XSS payload;
Send GET request with XSS payload;
Check for XSS payload;
if XSS payload found then
    XSS Vulnerability Detected;
end
else
    No XSS Vulnerability Detected;
end
Send GET request for SSRF detection;
Check response status code;
if Status code is 200 then
    No SSRF Vulnerability Detected;
end
else
    SSRF Vulnerability Detected
end
Log the vulnerability detection analysis results.
```

The integration of caching into this algorithm significantly enhances its efficiency and response times, making it an invaluable component of our vulnerability detection approach.

2) Algorithm 2: Vulnerability Detection: Algorithm 2 is designed for comprehensive security assessment in web applications. The algorithm includes the following fundamental steps:

The vulnerability detection algorithm initializes the XML payload, retrieves and parses web content, identifies page structures, and detects SQL injection, XXE, XSS, and SSRF vulnerabilities. These steps ensure real-time examination, detailed analysis, and robust defense against diverse security threats. Finally, the algorithm stores detection specifics in a logging record file, providing an exhaustive overview of the application's security posture.

Our proposed approach intelligently integrates these two algorithms to offer a robust, efficient, and comprehensive method for identifying vulnerabilities in web applications. Algorithm 1 optimizes efficiency through caching, while Algorithm 2 ensures the depth of security assessment remains uncompromised. This combination results in a powerful and balanced approach to web application security.

IV. Result and Analysis

The result and analysis section presents the findings of applying the automated Web Vulnerability Detection tool to target web applications and provides a critical assessment of its performance and effectiveness. Adding a caching mechanism in the requests package improves performance of detection tool by reducing server requests and response times, thereby minimizing server load and bandwidth usage, and enhancing overall user experience while mitigating the risk of web vulnerabilities.

A. Experimental Setup

To evaluate the tool's capabilities, a series of experiments were conducted using a diverse set of web applications known to have different vulnerabilities. The following details the experimental setup:

1) Test Environments: A variety of web applications with known web vulnerabilities were selected as test cases. These applications represent different domains and complexities.
2) Tool Configuration: The Web Vulnerability detection tool was configured to target each test application by including payload variations and detection criteria, were optimized for accuracy. The tool also included web scraping to extract forms from provided website. Caching mechanism is included to increase accuracy.
3) Data Collection: For each test, data was collected on the tool's actions, detected vulnerabilities, and false positives, if any. This data includes the url of the tested web application, form details, and detection results.

B. Detection Results

The results of the experiments are presented in this subsection. They include:

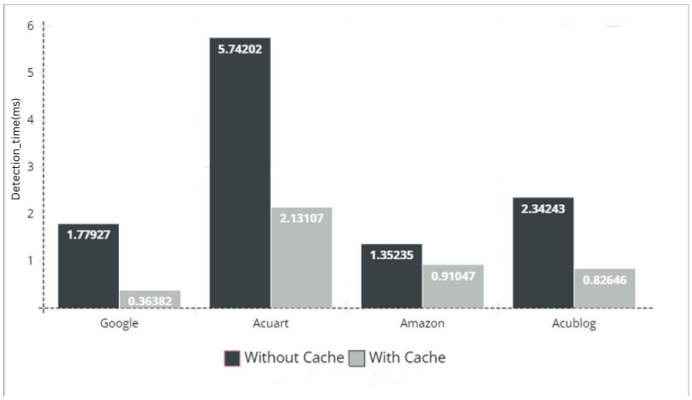

Fig. 2. Comparative Analysis of Vulnerability Detection: A Time-Efficiency Perspective

1) Identified Vulnerabilities: The tool successfully identified different web vulnerabilities in several web applications. Each detected vulnerability is detailed, including the specific form and payload that triggered the detection.

2) False Positives: Occurrences of false positives, where the tool incorrectly flagged a form as vulnerable, are documented. These cases are analyzed to determine the reasons behind false positives and potential areas for improvement.

C. Performance Evaluation

The performance evaluation section assesses the effectiveness and efficiency of the Web Vulnerability detection tool:

1) Accuracy: The tool's accuracy in identifying true web vulnerabilities is measured. The true positive rate and false positive rate are calculated and discussed.

2) Response Time: The time taken by the tool to identify vulnerabilities is analyzed. Higher response times due to repeated fetching of data from the server for every request but after addition of caching mechanism, response time reduces as cached data is served eliminating the need for repeated server queries.

3) Server Requests: Increased number of server requests as data is fetched afresh for each user interaction before caching is done. Once including the caching, there is decreased number of server requests since cached responses fulfill many user requests.

D. Discussion

The discussion section provides a deeper analysis of the results and their implications:

1) Latency Evaluation: Cached responses can be served quickly without the need to fetch data from the server, reducing the round-trip time and, consequently, lowering latency. Lower latency means faster loading times for users, leading to a more responsive and snappier user experience.

2) Effectiveness of Payloads: The effectiveness of different payload variations is examined. Insights are provided into which types of payloads were most successful in triggering vulnerability alerts.

3) Challenges and Limitations: Challenges faced during the experiments, such as evasive techniques used by web applications, are discussed. The limitations of the tool and areas for future improvement are highlighted.

4) Real-World Applicability: The practical applicability of the tool in real-world scenarios is considered. Its potential impact on web application security and the prevention of web attacks is discussed.

V. Conclusion and Future Work

In summary, the paper introduces a robust web security approach, detecting vulnerabilities (SQL Injection, XXE, XSS, SSRF) and optimizing performance via caching. Our system reduces data breach risks, improves response times for popular URLs, empowers developers to proactively address security issues, and includes logging records for tested URLs, enhancing transparency and accountability in maintaining a secure digital environment. Future work may involve leveraging machine learning for adaptive vulnerability detection and exploring distributed caching for improved scalability. Investigating advanced caching strategies and implementing a user-friendly monitoring dashboard could further enhance the system's efficiency and user experience. Exploring real-time threat intelligence integration could provide proactive defense against emerging vulnerabilities, contributing to a continuously secure web application environment.

References

[1] Q. Li, W. Li, J. Wang and M. Cheng, "A SQL Injection Detection Method Based on Adaptive Deep Forest," in IEEE Access, vol. 7, pp. 145385-145394, 2019, doi: 10.1109/ACCESS.2019.2944951.

[2] H. -C. Chen, A. Nshimiyimana, C. Damarjati and P. -H. Chang, "Detection and Prevention of Cross-site Scripting Attack with Combined Approaches," 2021 International Conference on Electronics, Information, and Communication (ICEIC), Jeju, Korea (South), 2021, pp. 1-4, doi: 10.1109/ICEIC51217.2021.9369796.

[3] R. Shahid, S. N. K. Marwat, A. Al-Fuqaha and G. B. Brahim, "A Study of XXE Attacks Prevention Using XML Parser Configuration," 2022 14th International Conference on Computational Intelligence and Communication Networks (CICN), Al-Khobar, Saudi Arabia, 2022, pp. 830-835, doi: 10.1109/CICN56167.2022.10008276.

[4] M. Alghawazi, D. Alghazzawi, and S. Alarifi, "Detection of SQL Injection Attack Using Machine Learning Techniques: A Systematic Literature Review," Journal of Cybersecurity and Privacy, vol. 2, no. 4, pp. 764–777, Sep. 2022, doi: 10.3390/jcp2040039.

[5] Al-talak, Khadejah, and Onytra Abbass. "Detecting server-side request forgery (SSRF) attack by using deep learning techniques." International Journal of Advanced Computer Science and Applications 12.12 (2021).

[6] R. Banerjee, A. Baksi, N. Singh and S. K. Bishnu, "Detection of XSS in web applications using Machine Learning Classifiers," 2020 4th International Conference on Electronics, Materials Engineering Nano-Technology (IEMENTech), Kolkata, India, 2020, pp. 1-5, doi: 10.1109/IEMENTech51367.2020.9270052.

[7] J. Harish Kumar and J. J Godwin Ponsam, "Cross Site Scripting (XSS) vulnerability detection using Machine Learning and Statistical Analysis," 2023 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 2023, pp. 1-9, doi: 10.1109/ICCCI56745.2023.10128470.

[8] Q. Li, W. Li, J. Wang and M. Cheng, "A SQL Injection Detection Method Based on Adaptive Deep Forest," in IEEE Access, vol. 7, pp. 145385-145394, 2019, doi: 10.1109/ACCESS.2019.2944951.

[9] nes Jemal, Omar Cheikhrouhou, Habib Hamam and Adel Mahfoudhi," SQL Injection Attack Detection and Prevention Techniques Using Machine Learning", International Journal of Applied Engineering Research ISSN 0973-4562 Volume 15, Number 6 (2020) pp. 569-580.

[10] Musaab Hasan ; Zayed Balbahaith ; Mohammed Tarique," Detection of SQL Injection Attacks: A Machine Learning Approach,International Conference on Electrical and Computing Technologies and Applications (ICECTA), Ras Al Khaimah, United Arab Emirates, 2019, pp. 1-6.

[11] Musaab Hasan ; Zayed Balbahaith ; Mohammed Tarique," Detection of SQL Injection Attacks: A Machine Learning Approach,International Conference on Electrical and Computing Technologies and Applications (ICECTA), Ras Al Khaimah, United Arab Emirates, 2019, pp. 1-6.

[12] Akinsola, Jide E. T, Awodele, Oludele and 3 Idowu, Sunday A," SQL Injection Attacks Predictive Analytics Using Supervised Machine Learning Techniques", International Journal of Computer Applications Technology and Research Volume 9–Issue 04, 139- 149, 2020.,

[13] Mariam M. N. Aboelwafa, Karim G. Seddik, Mohamed H. Eldefrawy, Yasser Gadallah, and Mikael Gidlund, A Machine Learning-Based Technique for False Data Injection Attacks Detection in Industrial IoT", IEEE,2020.

[14] J.Jagadessan, Akshat Shrivastava, Arman Ansari, Laxmi Kanta Kar, Mukul Kumar," Detection and Prevention Approach to SQLi and Phishing Attack using Machine Learning", International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-8, Issue-A, April 2019