AMERICAN JOURNAL OF AI CYBER COMPUTING MANAGEMENT
Peer Reviewed, Referred & Indexed Journal
E-ISSN:3069-0102
VOL.6, NO. 2(1) (2026)

AuthShieldX: A Self-Adaptive Exploit-Defense Mesh for Real-Time Injection Anomaly Surfacing in Credential Gateways

Ch. Jyothi[1*], K. Vamshee Krishna[1*], Vaggani Purushotham[2], Kotha Nishanth Reddy[2], Naspuri Rishik[2]

[1]Assistant Professor, [2]UG Student, [1,2]Department of Computer Science and Engineering

[1,2]Kommuri Pratap Reddy Institute of Technology, Ghanpur, Ghatkesar, 501301, Telangana, India.

*Correspondence: Ch. Jyothi (vampu.jyothi9@gmail.com), K. Vamshee Krishna(vamshik825@gmail.com)

ABSTRACT

The rapid growth of web-based applications has made secure authentication a critical requirement to protect user data from cyber threats such as Structured Query Language injection (SQLi) attacks. With the increasing reliance on digital platforms, ensuring data integrity, confidentiality, and controlled access has become essential. A major challenge arises from weak authentication implementations where improper input handling allows attackers to manipulate SQL queries and gain unauthorized access. In traditional systems, authentication is often implemented using dynamic query construction with minimal input validation, prioritizing functionality over security. Such systems fail to recognize malicious input patSterns and remain highly vulnerable to injection attacks. The limitations of these systems include unsafe query construction, weak validation techniques, lack of effective attack detection mechanisms, and insufficient focus on secure design principles, leading to risks such as data breaches and compromised system integrity. These issues highlight the need for a more secure and reliable authentication approach that balances protection with usability. To address these challenges, the proposed system introduces a Django-based application integrated with a MySQL (Structured Query Language-based relational database management system) backend, consisting of two modules: a vulnerable login module to demonstrate SQLi risks and a secured login module that applies input validation and detects suspicious patterns in user inputs. This dual-module design enables clear comparison between insecure and secure approaches. The significance of this research lies in enhancing awareness of web application security, promoting secure coding practices, improving authentication reliability, and providing a practical foundation for developing systems resistant to common cyber threats.

Key words: Structured Query Language Injection (SQLi), Web Application Security , Cyber Threat Detection, Query Sanitization, Data Integrity.

1. INTRODUCTION

Research on detecting SQLi vulnerabilities has gained significant momentum within the field of information security due to the increasing frequency of web-based attacks. A widely adopted approach for identifying such vulnerabilities is static analysis, which examines application source code without executing it. This method leverages structural and behavioral representations such as Abstract Syntax Tree (AST), Control Flow Graph (CFG), and Call Graph to analyze program logic and identify patterns associated with potential vulnerabilities [1]. Over time, static analysis techniques have evolved and achieved a reasonable level of maturity, enabling partial automation in detecting SQLi flaws across web applications [2]. Despite these advancements, they still face challenges in accurately identifying vulnerabilities in complex systems, particularly those utilizing Object-Oriented Database Extension (OODBE), where dynamic behaviors and abstractions reduce analysis precision.

A Web Application Vulnerability (WAV) is generally defined as a weakness originating from coding flaws that can lead to severe consequences when exploited. To uncover and mitigate such weaknesses, penetration testing—also known as ethical hacking—is commonly employed. Established frameworks such as the OWASP provide standardized methodologies and guidelines for conducting security

assessments and evaluating the effectiveness of detection tools [3]. These methodologies play a crucial role in ensuring systematic and comprehensive testing of web applications.

Web Application Vulnerability Scanners (WAVS) are essential tools used during penetration testing to automate the identification of security issues. They aim to detect vulnerabilities early and prevent potential breaches by scanning web applications for known attack patterns. However, different scanners often produce inconsistent results when analyzing the same target system due to variations in detection techniques, coverage, and performance, particularly for vulnerabilities such as Cross-Site Scripting (XSS) and SQLi. This inconsistency forces testers to rely on multiple tools and their own expertise to interpret results effectively. Furthermore, the reliance on manual processes in Web Application Penetration Testing (WAPT) introduces challenges such as increased time consumption, higher operational costs, and susceptibility to human error [4]. To address these limitations, recent research emphasizes the integration of intelligent and hybrid approaches that combine static analysis with dynamic testing and machine learning techniques. Such methods aim to improve detection accuracy, enhance coverage, and reduce dependency on human expertise, thereby making vulnerability assessment more efficient and reliable in modern web environments [5].

Problem Definition

Modern web applications heavily depend on database-driven authentication systems to validate users. However, many of these systems use insecure query practices such as direct string concatenation, making them vulnerable to SQLi attacks. Attackers can manipulate inputs to bypass login checks, steal sensitive data, or corrupt the database. The core problem is the lack of secure input validation and poor query-handling mechanisms in traditional login systems. This research aims to identify these vulnerabilities and implement a secured authentication approach that prevents malicious SQL queries.

2. RELATED WORK

2.1 Web Application Vulnerabilities and Detection Approaches

Nam, et al. [6] analyzed vulnerabilities in web-based systems with emphasis on APIs and improper source code handling. Their findings showed that weak authentication and authorization mechanisms enable unauthorized access to sensitive data, making systems vulnerable to data leakage and manipulation. The study highlights the importance of secure API design practices such as token-based authentication, role-based access control, and strict input validation, along with regular vulnerability assessments to enhance overall security. Similarly, Moreira, et al. [7] proposed an automated vulnerability detection platform that integrates multiple security tools into a unified system. By enabling interoperability among tools, their approach improves detection coverage while reducing redundancy and false positives. The system is designed to be cost-effective and requires minimal user intervention. In the same direction, Seara, et al. [8] developed an automated framework using open-source tools that supports continuous monitoring and vulnerability scanning, demonstrating high usability and effectiveness even for non-expert users.

2.2 Vulnerability Detection Techniques and Evaluation

Bennouk, et al. [9] provided a comprehensive review of vulnerability detection methodologies, including signature-based, anomaly-based, and hybrid techniques. Their analysis emphasized the importance of proactive vulnerability management and highlighted that the effectiveness of detection systems depends on the quality and timeliness of vulnerability data. Extending this perspective, Althunayyan, et al. [11] evaluated black-box vulnerability scanners on modern web applications. Their results revealed limitations in detecting advanced injection attacks such as SQL injection, NoSQL injection, and SSTI, particularly in dynamic environments, indicating the need for more adaptive and intelligent scanning mechanisms.

2.3 AI-Based and Intelligent Security Solutions

Alaoui, et al. [10] conducted a Systematic Literature Review on the application of DL techniques for vulnerability and attack detection. Their study demonstrated that DL models can effectively detect both known and unknown attack patterns with higher accuracy compared to traditional methods, although challenges such as high computational requirements, dependency on large datasets, and limited interpretability remain. Li, et al. [12] introduced an intelligent penetration testing framework based on NMPT, integrating reinforcement learning and Markov Decision Processes to simulate realistic cyberattack scenarios and automate decision-making. Similarly, Pratama, et al. [13] developed CIPHER, an LLM-based system that assists ethical hackers by providing contextual recommendations and guidance, thereby improving efficiency and reducing the knowledge gap.

2.4 Advanced Frameworks for Vulnerability Management and Testing

Sotiropoulos, et al. [14] proposed a structured vulnerability management framework that incorporates risk-based prioritization and remediation strategies. Their approach considers factors such as severity, exploitation likelihood, and resource constraints, making it particularly effective for complex environments such as IoT systems. Furthermore, Chowdhary, et al. [15] introduced a GAN-based autonomous penetration testing framework capable of generating realistic attack payloads. Their system enhances the detection of vulnerabilities such as XSS by producing diverse and high-quality attack patterns, improving automation, scalability, and overall testing efficiency.

3. PROPOSED SYSTEM

The system architecture, as illustrated in Fig. 1, represents a comprehensive Django-based web application that integrates both insecure and secure authentication mechanisms while interacting with a MySQL database backend to demonstrate practical web security scenarios. The workflow begins at the client browser, where users input their login credentials through HTML forms, which are transmitted as HTTP POST requests containing structured key–value pairs to the Django server operating within a Python runtime environment. Upon receiving the request, the Django framework processes it through its URL routing mechanism and maps it to the appropriate view function based on the requested operation, such as user login or registration, while also verifying that the request method is valid. Within the business logic layer, the authentication process is divided into two distinct execution paths: insecure and secure. In the insecure path, user-provided inputs are directly embedded into SQL query strings using the pymysql interface without applying any validation or sanitization, resulting in raw queries that are highly susceptible to SQLi attacks, where attackers can inject malicious code to manipulate database operations and bypass authentication controls.

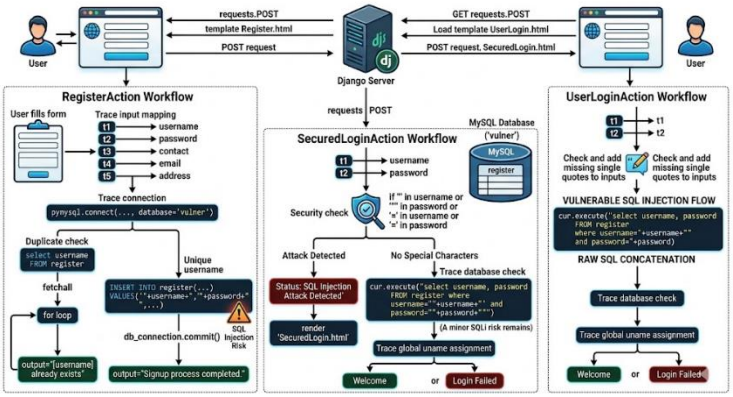

Fig. 1: Proposed system architecture of real-time SQLi defense.

In contrast, the secure path incorporates robust input validation and filtering techniques to detect suspicious patterns, including special characters and injected SQL fragments, ensuring that only sanitized and safe inputs are used in query construction. Following query generation, both secure and insecure queries are executed on the MySQL database, which processes them by comparing the

provided credentials against stored user records and returns corresponding outcomes such as successful matches or null results. The response handling component within Django then interprets these results, constructs appropriate context data, and forwards it to the template engine. The template rendering layer dynamically generates HTML responses based on the authentication outcome, which are then sent back to the client browser, completing the request-response lifecycle. This architecture not only maintains modularity and clear separation of concerns but also effectively highlights the contrast between vulnerable and secure implementations, providing a practical and insightful framework for understanding, analyzing, and mitigating web application security threats.

3.1 SQLi Detection Workflow

The SQLi Detection Workflow as illustrated in Fig. 2 is designed to safeguard the authentication system by identifying input patterns that indicate malicious intent. This module monitors user-submitted credentials and analyses them for harmful characters or SQL keywords commonly used in exploits, such as ', --, OR, or conditional expressions like 1=1. By performing server-side validation before constructing or executing database queries, the system prevents attackers from manipulating SQL logic or bypassing authentication. This workflow adds an essential defensive layer, ensuring that only clean and trusted input is processed while potential injection attempts are immediately blocked. Through this proactive detection mechanism, the system enhances overall application security and protects sensitive user data from unauthorized access.

User Enters Username and Password: The detection workflow begins when the user inputs their login credentials into the login form. Both fields username and password are directly submitted to the server without any client-side validation. Since the system allows free-form text, users may unknowingly or intentionally enter special characters or SQL fragments, creating a potential risk for SQLi. This step marks the starting point where malicious input can enter the system.

Server Receives User Input: Once the form is submitted, Django processes the POST request and retrieves the entered values using request.POST.get (). At this stage, the system treats these values as plain text without performing sanitization. Whatever is typed by the user, including harmful SQL operators, reaches the backend unchanged. This step highlights the importance of server-side filtering because client-side validation cannot be trusted in security-sensitive workflows.

System Checks for Malicious Characters or Patterns: The system performs a defensive check by scanning the input for dangerous characters such as ', ", =, --, OR, or patterns like 1=1. These symbols are red flags for SQLi attempts. The detection logic evaluates if the input contains anything that could manipulate the structure of the SQL query. This step serves as the primary layer of security to identify abnormal or suspicious login attempts before processing them further.

Detection of SQLi Attempt: If the system identifies any suspicious character or keyword in the input, it immediately classifies the request as a potential SQLi attack. This is done before any database query execution, ensuring that harmful SQL does not reach the database layer. The detection mechanism effectively prevents attackers from crafting queries that bypass authentication or manipulate data. At this point, the workflow branches into the defensive response sequence.

Block the Login Attempt and Display Warning: Upon detecting malicious input, the system halts the login process entirely and avoids executing any database query. Instead, it displays a warning message such as “SQLi Attack Detected” to the user. This prevents unauthorized access and helps administrators monitor security events. This step ensures that harmful payloads cannot exploit vulnerabilities in the authentication logic.

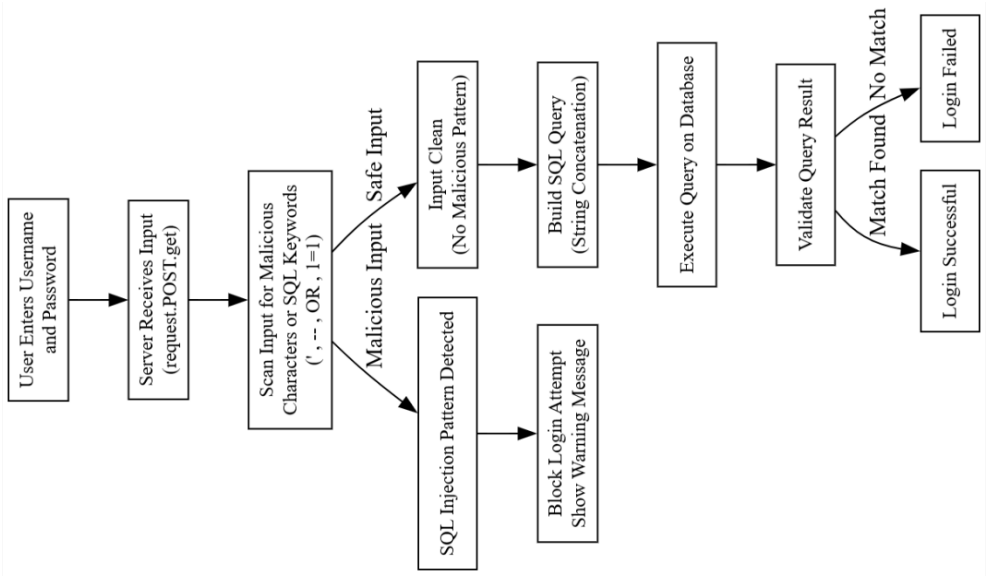

Fig. 2: SQLi detection workflow

Proceed with Normal Login if No Malicious Pattern Found: If the input does not contain any harmful characters or suspicious patterns, it is classified as clean. The system then proceeds to build the SQL query for verifying the credentials. This step ensures legitimate users can authenticate normally while keeping attackers out. Clean input represents a safe pathway that allows the workflow to continue into the database query stage.

Build SQL Query with Clean Input: Once input is validated, the backend constructs a SQL SELECT query to match the username and password in the database. Although the query still uses string concatenation, the risk is reduced because malicious characters were filtered earlier. The query is structured to retrieve only the matching user record. This step acts as the bridge between detection and actual authentication processing.

Execute SQL Query Against Database: The constructed SQL query is now sent to the database via PyMySQL for execution. Since harmful payloads have already been blocked, the query executes safely without altering the logic. The database responds with matching records if the credentials are correct. This step marks the transition from detection to final authentication validation.

Validate Query Results and Decide Authentication Outcome: The system checks the returned rows to verify whether the credentials matched a record in the database. If a row is found, the login is considered valid, and the user is successfully authenticated. If the query returns no rows, the login attempt is marked as failed. This step finalizes the workflow by deciding whether access should be granted or denied.

Display Login Success or Failure Message: Based on the validation result, the system either redirects the user to the authenticated interface or displays a "Login Failed" message. Clean users proceed to their dashboard, while incorrect inputs result in a failed login state. This step completes the SQLi Detection Workflow, ensuring secure and controlled access to the system.

4. RESULTS AND DESCRIPTION

Fig. 3 depicts the secure login module screen, representing the authentication interface enhanced with SQLi protection mechanisms. It illustrates how login inputs are processed under security-aware validation to prevent malicious query execution. The figure reflects the system's approach to safeguarding database interactions through structured input handling. It demonstrates the integration of real-time security monitoring within the authentication process.

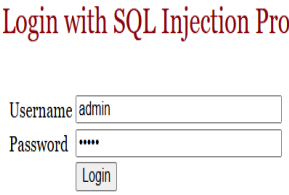

Fig. 3: SQLi protection Module Screen

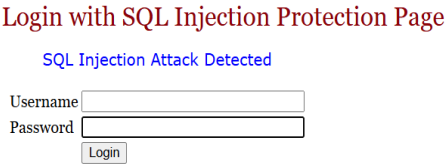

Fig. 4: SQLi Detection Screen

Fig. 4 depicts the SQLi detection screen, illustrating the system's capability to identify and respond to malicious input attempts in real time. It represents the detection and prevention mechanism that safeguards database queries from unauthorized manipulation. The figure reflects the integration of exploit-aware monitoring that ensures system resilience against common injection attacks. It demonstrates how alerts or detection outcomes are communicated within the interface.

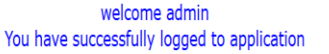

Fig. 5: Secure Login Authentication Screen

Fig. 5 illustrates the secure login authentication screen, representing the stage where validated credentials allow access under enforced security policies. It depicts the confirmation of successful authentication while maintaining protection against potential vulnerabilities. The figure reflects how security checks are embedded throughout the login lifecycle to ensure safe system usage. It highlights the continuity of protection even after authentication is completed.

5. CONCLUSION

The research effectively illustrates the critical role of secure authentication mechanisms in web applications by presenting a clear comparison between an insecure login implementation and a protected module resistant to SQLi attacks. The implementation reveals how poorly handled user inputs can be exploited by attackers to bypass access controls and interfere with database operations. It exposes the flaws in conventional query construction approaches and underlines the necessity of adopting secure development practices such as proper input validation and structured interaction with databases. By utilizing Django along with PyMySQL, the system creates a practical environment that demonstrates both the occurrence and prevention of security vulnerabilities. The secured module successfully identifies and restricts suspicious input patterns, allowing access only to authorized users. Overall, the research enhances understanding of SQLi threats and promotes the adoption of safer programming practices, including input sanitization, credential verification, and avoiding direct concatenation of SQL statements.

REFERENCE

[1] Altulaihan, E.A.; Alismail, A.; Frikha, M. A Survey on Web Application Penetration Testing. Electronics 2023, 12, 1229.

[2] Sadqi, Y.; Maleh, Y. A systematic review and taxonomy of web applications threats. Inf. Secur. J. Glob. Perspect. 2022, 31, 1–27.

[3] Trickel, E.; Pagani, F.; Zhu, C.; Dresel, L.; Vigna, G.; Kruegel, C.; Wang, R.; Bao, T.; Shoshitaishvili, Y.; Doupé, A. Toss a Fault to Your Witcher: Applying Grey-box Coverage-Guided Mutational Fuzzing to Detect SQL and Command Injection Vulnerabilities. In Proceedings of the 2023 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 21–25 May 2023; pp. 2658–2675.

[4] Deepa, G.; Thilagam, P.S. Securing web applications from injection and logic vulnerabilities: Approaches and challenges. Inf. Softw. Technol. 2016, 74, 160–180.

[5] Alhamed, M.; Rahman, M.M.H. A Systematic Literature Review on Penetration Testing in Networks: Future Research Directions. Appl. Sci. 2023, 13, 6986.

[6] Nam Y, Choi S. Analysis of Vulnerabilities in College Web-Based System. Electronics. 2024; 13(12):2261. https://doi.org/10.3390/electronics13122261

[7] Moreira D, Seara JP, Pavia JP, Serrão C. Intelligent Platform for Automating Vulnerability Detection in Web Applications. Electronics. 2025; 14(1):79. https://doi.org/10.3390/electronics14010079

[8] Seara JP, Serrão C. Automation of System Security Vulnerabilities Detection Using Open-Source Software. Electronics. 2024; 13(5):873. https://doi.org/10.3390/electronics13050873

[9] Bennouk K, Ait Aali N, El Bouzekri El Idrissi Y, Sebai B, Faroukhi AZ, Mahouachi D. A Comprehensive Review and Assessment of Cybersecurity Vulnerability Detection Methodologies. Journal of Cybersecurity and Privacy. 2024; 4(4):853-908. https://doi.org/10.3390/jcp4040040

[10] Alaoui RL, Nfaoui EH. Deep Learning for Vulnerability and Attack Detection on Web Applications: A Systematic Literature Review. Future Internet. 2022; 14(4):118. https://doi.org/10.3390/fi14040118

[11] Althunayyan M, Saxena N, Li S, Gope P. Evaluation of Black-Box Web Application Security Scanners in Detecting Injection Vulnerabilities. Electronics. 2022; 11(13):2049. https://doi.org/10.3390/electronics11132049

[12] Li Y, Wang Y, Xiong X, Zhang J, Yao Q. An Intelligent Penetration Test Simulation Environment Construction Method Incorporating Social Engineering Factors. Applied Sciences. 2022; 12(12):6186. https://doi.org/10.3390/app12126186

[13] Pratama D, Suryanto N, Adiputra AA, Le T-T-H, Kadiptya AY, Iqbal M, Kim H. CIPHER: Cybersecurity Intelligent Penetration-Testing Helper for Ethical Researcher. Sensors. 2024; 24(21):6878. https://doi.org/10.3390/s24216878

[14] Sotiropoulos P, Mathas C-M, Vassilakis C, Kolokotronis N. A Software Vulnerability Management Framework for the Minimization of System Attack Surface and Risk. Electronics. 2023; 12(10):2278. https://doi.org/10.3390/electronics12102278

[15] Chowdhary A, Jha K, Zhao M. Generative Adversarial Network (GAN)-Based Autonomous Penetration Testing for Web Applications. Sensors. 2023; 23(18):8014. https://doi.org/10.3390/s23188014