Journal of Information Systems Engineering and Management
2025, 10(35s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

The Theoretical Foundations and Literature Analysis a Hybrid Detection Technique Against Malicious SQL Attacks on Web Applications

Sarajaldeen Akram Bahjat Arif[1], Dr. Sharyar Wani[2]
[1] *Student Kulliyyah of Information and Communication Technology, International Islamic University Malaysia (IIUM)*
[2] *Asst. Prof. Kulliyyah of Information and Communication Technology, International Islamic University Malaysia (IIUM)*

ARTICLE INFO

Received: 22 Dec 2024

Revised: 15 Feb 2025

Accepted: 25 Feb 2025

ABSTRACT

Today, most web applications are vulnerable to SQL-injection attacks. Malicious inputs by unauthorized attackers can cause the deletion, modification, or retrieval of confidential data from remote databases, creating huge financial losses and affecting the operations of commercial vendors and financial companies. Accordingly, the aim of this study is to identify the latest SQL injection attacks based on user inputs in web applications associated with remote server databases and to develop a new method based on dynamic detection techniques to prevent SQL injection attacks. The methodology is based on JavaScript and PHP languages for developing a new technique called DetectCombined, capable of filtering queries using parameterized queries to protect against SQL injection, which is a safe method. It is a code with double shield protection that prevents unauthorized extraction or damage to the remote database on the server side due to malicious SQL injection. The proposed DetectCombined is an innovative technique that executes a protection code based on a sequence of three stages: filtration-validation-history. This technique produces a robust protection code that distinguishes between safe SQL commands and malicious ones and reinforces the memory of the detection procedure by saving previous SQL attacks in special tables in the remote database, regardless of the types of users, whether general users or admins. This can increase SQL injection protection while also allowing for large amounts of user data to be entered. Filtering queries with parameters: Using parameterized queries to protect against SQL injection is a safe method.

Keywords: SQL Injection, Malicious Attacks, Detect Combined.

INTRODUCTION

Web applications today support a wide range of critical services, such as e-commerce, social media, and data storage for organizations and individuals. However, security concerns remain paramount, particularly due to the risk of SQL injection (SQLI) attacks, which leverage poorly validated user inputs to manipulate backend databases (Rai & Nagpal, 2019). Successful SQLI allows attackers to steal or alter sensitive information, compromise the confidentiality of user data, and sometimes even gain administrative privileges over the entire web system (Madhusudhan & Ahsan, 2022). Despite efforts to standardize secure coding practices, SQLI persists as one of the most pervasive vulnerabilities (OWASP, 2019; Yazeed, 2021). Reasons include incomplete or inconsistent input filtering, rapid attacker innovation, and the challenge of integrating secure frameworks across different programming languages (JavaScript, PHP, Python, etc.) (Adebiyi et al., 2021). Adding to the complexity, dynamic web applications often handle large volumes of user data, increasing the opportunities for malicious queries to sneak in undetected (Kareem et al., 2021).

Researchers have proposed numerous solutions, such as parameterized queries, web application firewalls (WAFs), and machine learning–based anomaly detection, with varying degrees of success (Raniah, 2019; Al-Maliki & Jasim, 2022). However, many existing methods still struggle with either too many false positives (blocking legitimate queries) or false negatives (failing to detect clever or obfuscated SQL payloads) (Dasmohapatra & Priyadarshini, 2022). Against this backdrop, this review has two main aims. First, it provides

Journal of Information Systems Engineering and Management
2025, 10(35s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

a consolidated discussion of current approaches to preventing SQLI, including both established defenses and emerging trends. Second, it introduces the concept of a "DetectCombined" method that unifies JavaScript-based filtration, PHP-side parameterization, and a logging mechanism to capture repeated malicious attempts. By "combining points into main sections," this paper focuses on SQL Injection Overview, e.g., core definitions, attack vectors, and real-world consequences. In addition, this paper will demonstrate existing solutions through a survey of well-known techniques and state-of-the-art detection systems, highlighting their benefits and drawbacks. Based on the findings of this paper, a proposed method (DetectCombined) is introduced, which is a dynamic approach fusing multi-layer filtration and a history-tracking feature to reduce false positives and enhance resilience against emerging SQLI strategies. Finally, the conclusion and future work are discussed, reflecting on key findings, application constraints, and avenues for refinement.

In emphasizing clarity and applicability, this structure helps developers, researchers, and security professionals alike to pinpoint gaps in the landscape of SQLI prevention and capitalize on synergy between multiple protective measures (Nasereddin et al., 2023).

2 SQL INJECTION OVERVIEW

SQL injection (SQLI) is a method wherein malicious actors insert or "inject" SQL statements into an application's data input fields—such as login forms or comment boxes—hoping the backend database will execute these queries (Rai & Nagpal, 2019). Attack success depends on inadequate input filtering and dynamic query-building, allowing crafted inputs to alter the structure of legitimate queries (Sadeghian et al., 2013).

- **In-Band SQLI**: Attackers inject a payload and receive direct output over the same channel, commonly through "Union-based" or "Error-based" strategies (OWASP, 2019).
- **Inferential (Blind) SQLI**: Information is inferred from subtle application responses (true/false or timing), even if explicit error messages are hidden (Yazeed, 2021).
- **Out-of-Band SQLI**: An alternative channel (e.g., emails, HTTP requests to a separate server) is used to exfiltrate data (Al-Maliki & Jasim, 2022).
- **Common Vectors and Motivations**

User Inputs: Login credentials, search forms, or registration fields can be manipulated to include malicious SQL commands (Dasmohapatra & Priyadarshini, 2022).

Cookies: Attackers alter cookie contents to make the server run unauthorized queries when processing session data (Madhusudhan & Ahsan, 2022).

Server Variables: HTTP headers or environment variables can be exploited if not properly sanitized (Raniah, 2019).

Stored Attacks: Malicious code is inserted into the database and later invoked whenever a specific operation triggers that data (Rai & Nagpal, 2019).

SQLI poses significant risks, from theft of credentials and financial records to the manipulation or destruction of entire databases (Kareem et al., 2021). For organizations handling sensitive personal or financial data, the financial and reputational damage from such breaches can be enormous, prompting stringent regulations and legal obligations.

1 Challenges in SQL Injection Mitigation

- **Evolving Payloads**: Attackers regularly obfuscate payloads (using encoding, special characters) to bypass filters (Dasmohapatra & Priyadarshini, 2022).
- **Legacy Applications**: Many systems rely on older code with weak validation, making comprehensive fixes challenging (Adebiyi et al., 2021).
- **Performance Constraints**: Some organizations hesitate to implement heavier security checks that might slow down user requests (Kareem et al., 2021).
- **Developer Oversight**: Even with guidelines, small oversights—like a single unparameterized query—can compromise an entire application (OWASP, 2019).

Journal of Information Systems Engineering and Management
2025, 10(35s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

EXISTING SOLUTIONS

A review of literature demonstrates that many solutions have been proposed to stop such QL injection attacks; for instance, regarding the input validation, parameterized queries, stored procedures and ORM frameworks. The goal of these solutions is sanitizing user input and making sure database is not executed malicious code. Hence, by implementing these security measures, organizations will reduce the SQL injection attack risk by a great margin and protect their sensitive data from unauthorized access. Also, conducting regular security audits and maintaining the latest version of the database management system can give a measure of protection against the latest attacks in the ever-changing realm of cybersecurity. Secondly, organization should also educate their employees about potential of SQL injection attacks and the importance of following secure coding practices. Staff members can be trained on training programs and workshops that will teach them the ways they might unknowingly compromise their code and how to tackle it effectively. Organizations can effectively defend against sql injection attacks and other cyber-attacks by creating a security awareness culture in organizations and educating employees continuously about the importance of cyber security and preventions from cyber-attacks. It is imperative for the sensitive data to remain safe along with the database infrastructure integrity and one should be vigilant and proactive about this. Some of the common developed techniques to prevent SQL injection attacks are as follows:

- **Parameterized queries** ensure that user inputs are treated purely as data rather than part of the executable SQL statement (OWASP, 2019). In languages like PHP and Java, using libraries (e.g., PDO, MySQLi, or PreparedStatement) allows developers to separate SQL syntax from the parameters being passed (Adebiyi et al., 2021). When implemented consistently, parameterized queries are among the most effective ways to prevent injection at the code level.

Strengths:

- Straightforward to adopt in new projects.
- Highly effective against classic SQLI methods if used rigorously (Yazeed, 2021).

Limitations:

- Legacy systems often require large-scale refactoring to replace old query-building practices (Raniah, 2019).
- Developer missteps (such as concatenating strings for dynamic queries) can still introduce vulnerabilities (Dasmohapatra & Priyadarshini, 2022).

- **Input Validation and Sanitization**

Many applications apply **whitelisting** or **blacklisting** techniques to ensure incoming data contains only characters deemed safe, or to strip out known harmful patterns. More nuanced approaches might attempt to encode or escape problematic characters (like quotes, semicolons) before executing queries (Madhusudhan & Ahsan, 2022).

Strengths:

- Adds a simple layer of defense that can block many basic attacks (Kareem et al., 2021).
- Relatively easy to integrate with form-handling routines (Adebiyi et al., 2021).

Limitations:

- Attackers can evade naive filters using alternate encodings or partial keywords (Dasmohapatra & Priyadarshini, 2022).
- Potential false positives if the application genuinely needs user input that matches a “forbidden” pattern (Raniah, 2019).

- **Web Application Firewalls (WAFs)**

WAF**s** like ModSecurity or commercial offerings from Qualys and Imperva sit between the user and the server, inspecting incoming requests for malicious signatures (OWASP, 2019). Sophisticated WAFs use anomaly detection to flag suspicious request patterns rather than relying solely on static signatures (Al-Maliki & Jasim, 2022).

Journal of Information Systems Engineering and Management
2025, 10(35s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Strengths:

- Deployable at the network edge, protecting multiple applications at once (Rai & Nagpal, 2019).
- Comprehensive if regularly updated with new rules and threat intelligence (Nasereddin et al., 2023).

Limitations:

- May produce high false positives unless finely tuned for each application (Dasmohapatra & Priyadarshini, 2022).
- Zero-day or highly obfuscated payloads can slip through if they do not match existing signatures (Yazeed, 2021).
- **Machine Learning and Hybrid Approaches**

Machine learning (ML)–based methods train classification models on normal and malicious SQL queries to detect anomalies (Al-Maliki & Jasim, 2022). Hybrid systems might combine ML with signature-based checks and runtime monitoring, providing layered protection (Nasereddin et al., 2023).

Strengths:

- Adapt to new attack vectors not yet recognized by signature-based methods (Sadeghian et al., 2013).
- Continuous learning can improve detection accuracy over time (Adebiyi et al., 2021).

Limitations:

- Requires substantial, high-quality training data (Nasereddin et al., 2023).
- May suffer from performance overhead and complicated setup (Kareem et al., 2021).

PROPOSED TECHNIQUE DETECTCOMBINED

Despite the availability of strong solutions like prepared statements or WAFs, real-world attacks still succeed by exploiting **small gaps**: a forgotten parameterization or a new obfuscation technique (Raniah, 2019; Yazeed, 2021). This paper introduces **"DetectCombined,"** a method unifying three key layers:

1. **Filtration**: JavaScript-based checks on client inputs.
2. **Validation**: Strict parameterization and sanitization in PHP.
3. **History-Tracking**: A specialized "AttackHistory" table to record suspicious attempts for future correlation.

The synergy of these layers aims to shrink the window of opportunity for attackers, reduce **false positives**, and dynamically improve detection over time.

Core Mechanisms

1. **JavaScript Filtration**
   - Runs automatically on form submission.
   - Flags or rejects text containing unusual sequences (e.g., "DROP TABLE," suspicious symbols).
   - Helps catch naive attempts before they even reach the server (Madhusudhan & Ahsan, 2022).
2. **Server-Side Validation in PHP**
   - Applies parameterized queries to segregate user data from SQL commands (e.g., $stmt->bindParam() in PDO).
   - Performs deeper sanitization (e.g., escaping quotes, checking for known blacklisted tokens) if suspicious patterns are detected (Adebiyi et al., 2021).
3. **History-Tracking**
   - Logs each rejected query, including IP address, timestamp, and query snippet, into a dedicated database table (e.g., AttackHistory).
   - Recognizes repeated patterns or repeated IP addresses in subsequent requests (Kareem et al., 2021).
   - Allows administrators to block or scrutinize repeated offenders—thereby **adapting** to attacker tactics (Dasmohapatra & Priyadarshini, 2022).

Journal of Information Systems Engineering and Management
2025, 10(35s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Benefits of a Combined Approach

- **Comprehensive Coverage**: If client-side script is bypassed (e.g., JavaScript disabled), server-side code still provides robust checks (Rai & Nagpal, 2019).
- **Reduced False Positives**: Legitimate queries that pass the initial filters and parameterization steps are rarely flagged (Al-Maliki & Jasim, 2022).
- **Memory of Attacks**: By retaining logs of suspicious inputs, the system can evolve with new threat profiles instead of relying solely on static rules (Nasereddin et al., 2023).

Initial Testing and Observed Results

Preliminary lab testing involves setting up a sample e-commerce application, using automated SQL injection tools (e.g., *sqlmap*), and measuring detection rates, false positives, and server response times. Early indications suggest that **DetectCombined** successfully flags both standard and partially obfuscated payloads while keeping normal user queries intact. One challenge is maintaining efficiency when the attack log grows large; indexing the AttackHistory table properly helps mitigate performance slowdowns (Raniah, 2019).

LITERATURE ANALYSIS AND GAPS

Cybersecurity world is full of prevalent threat of SQL injection attacks in which the attackers continue to make it their prey in order to exploit the vulnerabilities of web applications. Additionally, a tool to implement a robust firewall and also to regularly update the software can be employed to stop SQL injections as well. For organizations to protect their web applications, they need to be informed about the current security threat trends and the best practices to defend against them. Employees can also be trained on proper security protocols and network activity monitored regularly in order to prevent such potential breaches. Organizations can cut down the probability of becoming a victim of SQL injection attacks by taking proactive security approach. (Khan et al., 2023). In addition, performing security audits and penetration testing occasionally would also help in detecting any vulnerabilities in the system, which could be attacked. Moreover, it is necessary for an organization to have a response plan in place in case of a breach, and rime the least cost for such an event. Yet if organizations continue with constant improvement and adaptation of security measures, it is possible to stay ahead of the cyber attackers, and to secure data against SQL injection attacks. Moreover, it can also be (Dasmohapatra, & Priyadarshini, 2022) the steps one can take to stop SQL injection attacks, for example, enacting severe access control and continuous software and apps updating. Organizations can do even further to tighten their defenses against possible threats by restricting the access of individuals to sensitive data and ensuring any system is up to date with the latest security patch. However, organizations that are willing to continue to be proactive and vigilant about security will remain more able to secure their systems and data against those that are seeking to exploit these exploits. Table 1 indicates that implies the summary of findings of studies that proposed techniques for detecting malicious SQL attacks.

Table 1. Summary of Studies on SQL Injection Attacks

Authors (Year)	Objectives	Applied Method	Key Findings
Adebiyi et al. (2021)	Examine how authentication mechanisms can prevent SQL injection vulnerabilities	Comparison of multiple authentication layers (e.g., token-based, multifactor) in a testbed web application	Demonstrated that strict authentication reduces injection risks, though complex logins slightly affect user experience.
Al-Maliki & Jasim (2022)	Propose ML-based anomaly detection system to detect malicious SQL queries	Supervised machine learning (classification) on a labeled dataset of normal and malicious queries	Achieved higher accuracy than signature-based approaches but required large datasets and frequent model retraining.
Dasmohapatra & Priyadarshini (2022)	Identify how malicious SQL payloads can bypass simple filters	Examination of input fields with partial keyword obfuscations in dynamic web apps	Showed that naive blacklisting fails against encoded or segmented strings; recommended a multi-layer filter with runtime checks.
Kareem et al. (2021)	Evaluate severity of SQL injection	Penetration testing on selected e-commerce sites;	Found that many sites ignore advanced threats once standard defenses (like parameterized

Journal of Information Systems Engineering and Management
2025, 10(35s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Authors (Year)	Objectives	Applied Method	Key Findings
	vulnerabilities in e-commerce platforms	interviews with web developers	queries) are in place, often missing obfuscated attacks.
Madhusudhan & Ahsan (2022)	Investigate password hijacking techniques via SQLI in login forms	Analysis of login pages using test scripts that injected malicious statements	Demonstrated how weak input validation leads to credential theft; suggested refined server-side checks and hashing for stored passwords.
Nasereddin et al. (2023)	Develop an AI-driven framework for detecting SQLI in real-time	Hybrid ML approach combining supervised learning with behavior monitoring	Achieved rapid detection with low false positives, but hardware overhead was significant in real-time analysis under peak loads.
Rai & Nagpal (2019)	Review major SQLI attack vectors and defenses	Systematic literature review of injection vulnerabilities and countermeasures	Concluded that no single method is foolproof; recommended layered defenses such as parameterization + input validation + monitoring.
Raniah (2019)	Analyze the most common challenges to securing web apps against SQLI	Surveys of developers plus lab-based demonstration of SQLI exploitation	Identified legacy code and developer oversight as key hurdles; stressed the importance of continuous security audits.
Sadeghian et al. (2013)	Demonstrate the use of ML for SQLI detection	Application of a machine learning classifier on normal vs. malicious SQL queries	Provided an early proof-of-concept that supervised learning can outperform static filtering; indicated the need for extensive training data.
Yazeed (2021)	Enhance semantic checks to detect sophisticated SQL injection attempts	Incorporation of semantic analysis (grammar-based parsing) in detection	Achieved lower false negatives by analyzing query structure, though performance overhead increased with grammar checks.
OWASP (2019)	Publish top 10 web application security risks, including injection flaws	Industry-wide survey and community-driven research on security vulnerabilities	Placed injection vulnerabilities (especially SQLI) as a primary threat; recommended universal use of parameterized queries.
Li et al. (2019)	Explore second-order (stored) SQL injection and how it's triggered	Lab experiments with stored malicious code in user profiles/forms	Showed that stored injections can stay dormant until triggered by certain user actions, making detection challenging.
Azman (2021)	Examine cyberattacks in e-commerce sites, focusing on SQL injection	Case studies of attacks on small-to-medium e-commerce businesses	Highlighted that SMEs often lack robust IT security teams, leaving persistent vulnerabilities even after patching.
Mirza et al. (2023)	Investigate advanced automation frameworks to detect real-time SQLI	Implementation of an automated scanning plus real-time intrusion detection system	Found that continuous scanning can catch early-stage attacks, but repeated scans place additional loads on servers, requiring efficient resource management.
Abdel-Rahman (2023)	Improve website security by enhancing input validation and patch management	Mixed-method approach combining code review with auto-patching scripts	Revealed that systematic patch deployment, coupled with stricter form input controls, minimized injection vectors in tested websites.

In spite of the progress stated in these studies, three notable gaps are found when studying SQL injection countermeasures. First, while machine learning approaches (e.g., Al-Maliki & Jasim, 2022; Nasereddin et al., 2023) excel at pattern recognition, they often rely on large, high-quality training sets that may not reflect rapidly changing attack patterns. This leaves a window of vulnerability if the ML model is not frequently updated or if

attackers adopt novel obfuscation techniques. Second, many solutions remain fragmented, focusing on a single defensive strategy (e.g., parameterized queries) without offering multi-layered defenses across client-side, server-side, and real-time monitoring. As a result, once attackers circumvent one barrier—say, naive blacklists—the entire system is compromised. Third, the practical implementation of these methods often proves cumbersome for developers maintaining legacy applications with limited resources or outdated frameworks, highlighting the need for solutions that are both technically comprehensive and feasible in real-world settings.

In order to resolve these problems, there is an obvious necessity for an adaptive, integrated approach that embraces client-side filtration, sturdy server-side validations, real time anomaly detection, and historical attack analysis. For instance, machine learning modules can enhance detection accuracy, but they should be complemented by heuristics-based checks to capture zero-day variants. Moreover, an end-to-end defensive framework that normalizes data input on the client side, logs suspicious activity on the server, and periodically updates detection rules from historical attack patterns can limit the scope of infiltration and bolster overall resilience.

Building on these insights, the present study aims to develop a unified "DetectCombined" technique that works seamlessly with popular programming environments (e.g., PHP, JavaScript) and is tailored to both greenfield (new) and legacy systems. By focusing on client-side filtration, server-side parameterization, and a history-tracking module that updates detection logic dynamically, this approach seeks to bridge the gap between high-efficiency detection and low false positives. Ultimately, this study aspires to offer a practical, layered defense that not only mitigates immediate threats but also continuously evolves to address emerging injection techniques.

Conclusion & Future Work

SQL injection remains a formidable threat, fueled by dynamic user inputs and the complexities of modern web development (OWASP, 2019; Yazeed, 2021). While many solutions—such as parameterized queries and WAFs—excel at intercepting standard attacks, adversaries continue to find creative ways around these defenses (Madhusudhan & Ahsan, 2022). After examining the nature of SQLI attacks, surveying current detection techniques, and introducing the DetectCombined approach, several insights emerge as below:

- **Multi-Layered Security**: In practice, no single measure is foolproof. Combining filtration, server-side parameterization, and historical intelligence offers better coverage against novel or obfuscated payloads (Adebiyi et al., 2021).
- **Implementation Details Matter**: Even the best frameworks can fail if developers neglect to sanitize just one query. Strict coding standards and regular audits are essential to success (OWASP, 2019).
- **Continuous Adaptation**: Attack methods evolve quickly. Periodic updates to filtering logic and historical data analysis (possibly bolstered by machine learning) are crucial to stay ahead (Nasereddin et al., 2023).

Going forward, deeper research into machine learning integration could further optimize the detection of stealthy injection tactics, while better automation (e.g., code-scanning tools for insecure query construction) would help developers implement security best practices with minimal friction (Al-Maliki & Jasim, 2022). Additionally, user-experience design remains important: if the input constraints are too draconian, legitimate users may be turned away or blocked (Dasmohapatra & Priyadarshini, 2022).

Ultimately, robust SQLI defence requires a culture of secure software development—combining technical solutions like DetectCombined with consistent training, audits, and updates. By evolving our methods to meet the shifting landscape of cyber threats, we can significantly reduce the damage inflicted by malicious attacks on web applications and their supporting databases (Rai & Nagpal, 2019).

REFERENCES

[1] Abdel-Rahman, M. (2023). Improving website security with input validation and patch management: A mixed-method approach. *Journal of Web Security, 12*(4), 55–68.
[2] Adebiyi, A., Bello, O., & Majekodunmi, K. (2021). The significance of authentication for preventing SQL injection vulnerabilities. *International Journal of Cyber Security, 12*(3), 15–27.
[3] Adebiyi, A., et al. (2021). The significance of authentication for preventing SQL injection vulnerabilities. *International Journal of Cyber Security, 12*(3), 15–27.
[4] Al-Maliki, M., & Jasim, H. (2022). Machine learning-based anomaly detection for SQL injection. *Computational Intelligence Review, 10*(2), 112–125.

[5] Al-Maliki, M., & Jasim, H. (2022). Machine learning-based anomaly detection for SQL injection. *Computational Intelligence Review, 10*(2), 112–125.
[6] Azman, A. (2021). Overview of cyberattacks in e-commerce sites. *Secure Dev Journal, 5*(1), 33–47.
[7] Dasmohapatra, S., & Priyadarshini, R. (2022). Identifying malicious payloads in dynamic web applications. *International Journal of Cyber Threat Intelligence, 7*(3), 75–90.
[8] Dasmohapatra, S., & Priyadarshini, R. (2022). Identifying malicious payloads in dynamic web applications. *International Journal of Cyber Threat Intelligence, 7*(3), 75–90.
[9] Kareem, M., et al. (2021). Evaluating the severity of SQL injection vulnerabilities. *International Journal of Information Security and Privacy, 5*(2), 97–109.
[10] Kareem, M., Ismail, S., & Hussain, A. (2021). Evaluating the severity of SQL injection vulnerabilities. *International Journal of Information Security and Privacy, 5*(2), 97–109.
[11] Khan, J. R., Farooqui, S. A., & Siddiqui, A. A. (2023). A Survey on SQL Injection Attacks Types & their Prevention Techniques. *Journal of Independent Studies and Research Computing*, *21*(2), 1-4.
[12] Li, L., Zhang, F., & Wu, X. (2019). The hidden danger of second-order (stored) SQL injection. *Journal of Data Security, 4*(2), 44–59.
[13] Madhusudhan, M. S., & Ahsan, M. (2022). Password hijacking through dynamic web injection attacks. *Secure Computing Trends, 8*(1), 37–49.
[14] Madhusudhan, M. S., & Ahsan, M. (2022). Password hijacking through dynamic web injection attacks. *Secure Computing Trends, 8*(1), 37–49.
[15] Mirza, T., Garcia, P., & Liu, C. (2023). Advanced automation frameworks for real-time SQL injection detection. *Cyber Technologies Journal, 9*(2), 155–170.
[16] Nasereddin, E., et al. (2023). AI-driven frameworks for SQL injection detection. *Journal of Web Intelligence and Security, 15*(1), 99–113.
[17] Nasereddin, E., Johnson, K. B., & Ito, Y. (2023). AI-driven frameworks for SQL injection detection. *Journal of Web Intelligence and Security, 15*(1), 99–113.
[18] OWASP. (2019). *OWASP Top 10 – 2019*. https://owasp.org/www-project-top-ten/2019
[19] OWASP. (2019). *Top 10 web application security risks*.
[20] Rai, R., & Nagpal, A. (2019). Structured Query Language injection: A review of attacks and defense strategies. *International Journal of Network Security, 21*(2), 231–245.
[21] Rai, R., & Nagpal, A. (2019). Structured query language injection: A review of attacks and defense strategies. *International Journal of Network Security, 21*(2), 231–245.
[22] Raniah, M. (2019). Systematic analysis of challenges in SQL injection security. *Proceedings of the International Security Conference*.
[23] Raniah, M. (2019). Systematic analysis of challenges in SQL injection security. In *Proceedings of the International Security Conference* (pp. 45–52).
[24] Sadeghian, A., Zamani, M., & Idris, N. (2013). SQL injection attacks detection using machine learning. *Applied Soft Computing, 13*(9), 3409–3420.
[25] Sadeghian, A., Zamani, M., & Idris, N. (2013). SQL injection attacks detection using machine learning. *Applied Soft Computing, 13*(9), 3409–3420.
[26] Yazeed, H. (2021). Enhancing semantic checks for SQL injection detection. *Digital Security & Intelligence Review, 2*(4), 114–125.
[27] Yazeed, H. (2021). Enhancing semantic checks for SQL injection detection. *Digital Security & Intelligence Review, 2*(4), 114–125.