**Comparative Analysis of SQL Injection Detection Techniques and Removal Methods: A Case Study Approach**

Rohana Mydin1, a), Abu Bakar Md Sultan2, b) and Hazura Zulzalil2, c)

*1Centre For Digital Innovations, Faculty of Computing and Informatics, Multimedia University, Persiaran Multimedia, 63100 Cyberjaya, Malaysia*

*2Faculty of Computer Science & Information Technology, Software Engineering Research Group, University Putra Malaysia, Serdang, Malaysia..*

*a) Corresponding Author: rohana.mydin@mmu.edu.my*

*b)abakar@upm.edu.my*

*c)hazura@upm.edu.my*

**Abstract.** SQL Injection (SQLi) attacks remain one of the most prevalent security threats targeting web applications, allowing attackers to manipulate database queries and compromise sensitive data. This study presents a comprehensive comparative analysis of various SQLi detection techniques and removal methods, incorporating a case study to evaluate their effectiveness in real-world scenarios. The research examines heuristic-based, signature-based, machine learning-based, and deep learning-based detection mechanisms, highlighting their strengths and limitations. Additionally, the study explores SQLi removal strategies, including input validation, prepared statements with parameterized queries, web application firewalls (WAFs), and automated sanitization techniques. The case study involves implementing and testing these methods on a controlled web environment, simulating various SQLi attack vectors. Experimental results indicate that while traditional heuristic and signature-based approaches are efficient in detecting known SQLi patterns, machine learning and deep learning models demonstrate superior accuracy in identifying zero-day attacks and obfuscated SQLi payloads. Furthermore, the adoption of prepared statements with PDO in PHP, coupled with a well-configured WAF, proves to be the most effective mitigation strategy, significantly reducing the risk of SQLi exploitation. This research contributes to the growing body of cybersecurity knowledge by providing an empirical evaluation of SQLi detection and removal methodologies, offering insights for developers, security practitioners, and researchers in fortifying web applications against SQLi threats. Future work will focus on enhancing machine learning models for real-time SQLi detection and developing hybrid approaches that integrate multiple defensive layers for improved security.

INTRODUCTION

Web applications have become integral to various aspects of modern life, facilitating e-commerce, communication, and data management. However, this increasing reliance has also led to a rise in cyber threats targeting these applications, with SQL injection being one of the most prevalent and dangerous vulnerabilities. SQL injection attacks occur when malicious SQL code is inserted into an application's input fields, allowing attackers to manipulate database queries and gain unauthorized access to sensitive information, modify data, or even execute administrative operations [1]. The consequences of successful SQL injection attacks can be severe, leading to financial losses, reputational damage, legal repercussions, and the compromise of vast amounts of personal and confidential data.

Despite being a well-known vulnerability for over two decades, SQL injection continues to be a significant security concern [2]. Attackers constantly evolve their techniques, making it crucial for security professionals and developers to stay informed about the latest detection and removal methods. This paper aims to provide a comprehensive comparative analysis of recent advancements in SQL injection detection and removal techniques, further illustrating their effectiveness and limitations through the examination of real-world case studies reported in academic literature and security research between 2020 and 2025 [3] - [9].

CATEGORIES OF SQL INJECTION ATTACKS

* SQL injection attacks can be broadly categorized based on the method of exploitation and the attacker's objectives [10]. Understanding these categories is essential for developing effective detection and prevention strategies.
* In-band SQL Injection: This is the most common type, where attackers use the same communication channel to both inject malicious code and retrieve results. This category includes:
* Error-based SQL Injection: Exploits error messages generated by the database server to gain information about the database structure.
* Union-based SQL Injection: Leverages the UNION SQL operator to combine the results of the original query with an injected malicious query, allowing the attacker to retrieve data from other database tables.
* Inferential (Blind) SQL Injection: Attackers do not receive direct error messages or data output. Instead, they infer information about the database by observing the application's response to various injected payloads. This includes:
  + Boolean-based Blind SQL Injection: Attackers send queries that cause the application to return different results (e.g., different page content) based on whether the injected condition is true or false.
  + Time-based Blind SQL Injection: Attackers send queries that force the database to wait for a specified duration if the injected condition is true, allowing them to infer information based on the response time [11].
  + Out-of-band SQL Injection: This less common type occurs when attackers use different channels (e.g., email, DNS) to retrieve data from the database server.
  + Second-order SQL Injection: The malicious payload is not immediately executed but is stored in the database and triggered later when the stored data is used in another SQL query [12].

COMPARATIVE ANALYSIS OF SQL INJECTION DETECTION TECHNIQUES

Various techniques have been proposed and implemented to detect SQL injection attacks [13] - [18]. These techniques can be broadly categorized into signature-based, anomaly-based, code analysis, machine learning/deep learning, and hybrid approaches. The summaries of the techniques are tabulated in Table 1.

* Signature-based Detection: This traditional approach relies on predefined patterns or rules (signatures) to identify known SQL injection attack strings. These signatures often include specific SQL keywords or malicious code snippets.
* Strengths: Effective against known attack patterns and relatively easy to implement.
* Weaknesses: Struggles to detect novel or obfuscated attacks that do not match existing signatures [20]. Requires constant updates to the signature database to remain effective against new threats.
* Anomaly-based Detection: This approach establishes a baseline of normal SQL query behavior and flags any deviations from this baseline as potentially malicious [21]. It analyzes query structure, frequency, and other characteristics to identify anomalies.
* Strengths: Can potentially detect previously unknown or zero-day attacks by identifying unusual query patterns.
* Weaknesses: May generate a high number of false positives, especially in dynamic environments with legitimate variations in query patterns. Requires a significant amount of normal traffic data to establish an accurate baseline.

Code Analysis (Static and Dynamic):

* Static Analysis: Examines the source code of the web application without executing it to identify potential SQL injection vulnerabilities [22]. It looks for patterns where user input is directly incorporated into SQL queries without proper sanitization.
* Strengths: Can identify vulnerabilities early in the development lifecycle before deployment.
* Weaknesses: May produce false positives and negatives, especially with complex code or dynamic query generation. Cannot detect runtime behavior.
* Dynamic Analysis: Analyzes the application's behavior during runtime by injecting various inputs and observing the responses. This can involve techniques like fuzzing and penetration testing [23].
* Strengths: Can detect vulnerabilities that might be missed by static analysis by observing actual application behavior.
* Weaknesses: Can be time-consuming and may not cover all possible execution paths.
* Machine Learning and Deep Learning: These techniques involve training models on large datasets of both normal and malicious SQL queries to learn patterns and classify new queries as either legitimate or an attack. Various algorithms, including neural networks, support vector machines, and ensemble methods, have been employed. Natural Language Processing (NLP) techniques are also increasingly used to understand the semantics of SQL queries [1], [5], [7], [9], [10], [14], [17], [22], [23], [25] - [27].
* Strengths: Can detect novel and sophisticated attacks by learning complex patterns from data. Deep learning models have shown high accuracy in classification.
* Weaknesses: Requires large, high-quality datasets for training. Performance depends heavily on the features extracted from the SQL queries. Can be computationally expensive, especially deep learning models. May face challenges with adversarial attacks designed to evade detection [6], [15].
* Hybrid Approaches: Combine two or more detection techniques to leverage their individual strengths and mitigate weaknesses. For instance, combining static analysis with runtime monitoring or integrating signature-based detection with machine learning [8],[16].
* Strengths: Can offer a more comprehensive and robust defense against a wider range of SQL injection attacks.
* Weaknesses: Can be more complex to implement and manage.

**TABLE 1.** Comparative analysis of SQL injection detection techniques

|  |  |  |
| --- | --- | --- |
| Technique | Strengths | Weaknesses |
| Signature-based | Effective against known attacks, easy to implement | Ineffective against novel attacks, requires constant updates |
| Anomaly-based | Detects unknown attacks | High false positives, requires accurate baseline |
| Static Code Analysis | Early vulnerability detection | False positives/negatives, limited runtime insight |
| Dynamic Code Analysis | Detects runtime vulnerabilities | Time-consuming, may not cover all paths |
| Machine/Deep Learning | Detects novel attacks, high accuracy potential | Requires large datasets, feature engineering, computationally intensive, susceptible to adversarial attacks |
| Hybrid Approaches | Comprehensive defense | Increased complexity |

COMPARATIVE ANALYSIS OF SQL INJECTION REMOVAL METHODS

Effective removal and prevention of SQL injection vulnerabilities require a multi-faceted approach encompassing secure coding practices and robust security mechanisms [20], [21]. Several methods are commonly employed to mitigate the risk of SQL injection attacks. The summaries of the techniques are tabulated in Table 2.

* Parameterized Queries (Prepared Statements): This is widely considered the most effective method to prevent SQL injection [12], [13]. It involves separating the SQL code structure from the user-provided data by using placeholders for input values, which are then passed separately to the database. This ensures that user input is treated as data, not executable code. Prepared statements are supported by most programming languages and database systems.
* Strengths: Highly effective in preventing SQL injection by clearly separating code and data. Relatively easy to implement. Can also offer performance benefits due to query pre-compilation.
* Weaknesses: Might not be suitable for all dynamic SQL scenarios involving table or column names.
* Stored Procedures: These are pre-compiled SQL code stored within the database. Applications call these procedures and pass parameters [15].
* Strengths: Enhance security by limiting the direct execution of arbitrary SQL queries. Can improve performance due to pre-compilation. Centralize database logic.
* Weaknesses: Can be less flexible than dynamic SQL. If stored procedures themselves construct dynamic SQL from parameters without proper sanitization, they can still be vulnerable. Overly broad database privileges granted for stored procedures can increase risk.
* Input Validation and Sanitization: This involves verifying that user-supplied input conforms to expected formats and types (validation) and removing or encoding potentially harmful characters (sanitization) before using it in SQL queries. Whitelisting (allowing only known safe inputs) is generally preferred over blacklisting (blocking known malicious inputs) as blacklists can be easily bypassed. Sanitization includes escaping special SQL characters using database-specific functions [16], [18].
* Strengths: Reduces the attack surface by ensuring only valid data is processed. Can catch many common injection attempts early.
* Weaknesses: Not a primary defense and can be insufficient against sophisticated attacks. Validation rules need careful definition and maintenance. Blacklisting is error-prone.
* Web Application Firewalls (WAFs): These act as a security layer between web applications and users, monitoring and filtering HTTP traffic to block malicious requests, including those containing SQL injection attempts. They use predefined rules and signatures to identify and block known attack patterns. [17]
* Strengths: Provide a centralized security layer. Can detect and block known SQL injection patterns. Offer virtual patching capabilities. AI-powered WAFs are emerging for more advanced threat detection.
* Weaknesses: Can be bypassed by sophisticated or novel attacks. Require proper configuration and tuning. Should not replace secure coding practices.
* Object Relational Mappers (ORMs): These libraries allow developers to interact with databases using object-oriented paradigms, abstracting away the need to write raw SQL. Many ORMs utilize parameterized queries by default [22], [23].
* Strengths: Reduce the likelihood of manual SQL injection vulnerabilities by abstracting database interactions. Often provide built-in protection mechanisms.
* Weaknesses: Performance overhead compared to raw SQL. Complexity in learning and configuration. Using raw SQL within ORMs can still introduce vulnerabilities. Potential for ORM-specific vulnerabilities.
* Principle of Least Privilege: This security principle dictates that applications should only be granted the necessary database privileges required for their functionality. This limits the potential damage an attacker can cause if they successfully exploit an SQL injection vulnerability. Applications should avoid connecting to the database with administrative or root user accounts [24].
* Strengths: Limits the scope of damage from a successful attack.
* Weaknesses: Requires careful planning and configuration of database roles and permissions.
* Secure Coding Practices and Developer Training: Educating developers on secure coding practices, including the proper use of parameterized queries, input validation, and awareness of common SQL injection attack vectors, is crucial for preventing vulnerabilities. Regular security training and awareness programs are essential. Utilizing static and dynamic analysis tools during development can also help identify potential vulnerabilities early [25].
* Strengths: Addresses the root cause of many SQL injection vulnerabilities by promoting secure development habits.
* Weaknesses: Requires ongoing effort and commitment from development teams.

**TABLE 2.** Comparative analysis of SQL injection removal methods

|  |  |  |
| --- | --- | --- |
| Method | Strengths | Weaknesses |
| Parameterized Queries | Highly effective, easy to implement, performance benefits | Not suitable for all dynamic SQL |
| Stored Procedures | Enhanced security, improved performance, centralized logic | Less flexible, risk if poorly written or dynamically construct SQL , potential for overly broad privileges |
| Input Validation/Sanitization | Reduces attack surface, early detection | Not a primary defense, can be bypassed, requires careful maintenance, blacklisting is error-prone |
| Web Application Firewalls | Centralized security, detects known attacks, virtual patching | Can be bypassed, requires proper configuration, not a replacement for secure coding |
| Object Relational Mappers | Reduces manual SQL, often uses parameterized queries by default | Performance overhead, complexity, risk of raw SQL usage, potential for ORM-specific vulnerabilities |
| Principle of Least Privilege | Limits the scope of damage | Requires careful planning and configuration |
| Secure Coding/Training | Addresses the root cause | Requires ongoing effort and commitment |

CASE STUDY APPROACH

To further illustrate the complexities and impact of SQL injection attacks, this section examines two significant case studies reported in recent years.

## Case Study 1: MOVEit Transfer Breach (2023)

In late May 2023, a critical zero-day SQL injection vulnerability (CVE-2023-34362) was discovered in Progress Software's MOVEit Transfer platform, a widely used managed file transfer application [26]. This vulnerability allowed unauthenticated attackers to gain unauthorized access to the MOVEit Transfer database. The CXXp ransomware group, also known as Cl0p, exploited this flaw to install a web shell called LEMURLOOT and exfiltrate massive amounts of sensitive data. The attack chain involved exploiting the SQL injection vulnerability to achieve remote code execution, enabling the installation of the web shell [27].

The impact of this breach was extensive, affecting over 2,700 organizations worldwide across various sectors, including education, government, healthcare, and financial services. The personal data of over 93.3 million individuals was exposed, leading to an estimated global financial loss of up to $12.15 billion. Affected organizations included over 1,000 U.S. colleges and universities via the National Student Clearinghouse and the healthcare sector, where the Maximus breach compromised 11.3 million patient records.

Detection of this vulnerability might have been possible through static analysis tools identifying unsanitized user input being used in SQL queries. Dynamic analysis could have flagged unusual database activity or attempts to execute commands on the server. Prevention could have been achieved by adhering to secure coding practices, such as using parameterized queries, and by promptly applying security patches released by the software vendor. The MOVEit Transfer breach underscores the critical importance of timely patching and secure development practices, as vulnerabilities in widely used software can have far-reaching consequences.

## Case Study 2: PostgreSQL psql SQL Injection (CVE-2025-1094)

In February 2025, Rapid7 discovered a high-severity SQL injection vulnerability, tracked as CVE-2025-1094, affecting the PostgreSQL interactive tool psql. This vulnerability received a high CVSS 3.1 base score of 8.1. The flaw stemmed from an incorrect assumption that attacker-controlled untrusted input, even after being safely escaped via PostgreSQL's string escaping routines, could not be leveraged to generate a successful SQL injection attack. Rapid7 found that SQL injection was still possible in certain scenarios when escaped untrusted input was included as part of a SQL statement executed by psql, particularly due to how PostgreSQL handles invalid UTF-8 characters. [14]

An attacker successfully exploiting this vulnerability could achieve arbitrary code execution by leveraging psql's ability to run meta-commands, which allow for operating system shell command execution. Exploitation of this PostgreSQL vulnerability was also linked to attacks targeting BeyondTrust Privileged Remote Access and Remote Support products.

This case highlights that even seemingly robust security mechanisms like string escaping can have underlying vulnerabilities. Static analysis might have difficulty identifying such complex issues related to character encoding and the interaction between different software components. Dynamic analysis, particularly penetration testing, could potentially flag unusual command execution attempts originating from database interactions. The lesson learned from CVE-2025-1094 emphasizes that a defense-in-depth strategy is crucial, and regular security audits and penetration testing are essential to uncover unexpected vulnerabilities, even in well-established and seemingly secure systems.

CONCLUSION AND FUTURE DIRECTIONS

The comparative analysis of SQL injection detection techniques reveals a continuous evolution in response to increasingly sophisticated attacks. While traditional signature-based and anomaly-based methods still play a role, the growing reliance on machine learning and deep learning [1], [7], [9], [10], [13], [ 17] demonstrates the potential for more adaptive and accurate detection capabilities. Hybrid approaches that combine the strengths of different techniques offer a promising direction for enhanced detection accuracy and coverage [ 5], [8], [15].

Similarly, the analysis of removal methods underscores the importance of a layered security approach. Parameterized queries remain the most effective primary defense, but should be complemented by other methods such as input validation, WAFs, and the principle of least privilege to provide a robust defense-in-depth strategy. The case studies of the MOVEit Transfer breach and the PostgreSQL psql vulnerability highlight the real-world impact of SQL injection attacks and the diverse ways in which they can manifest, even in seemingly well-protected systems. [12], [20] - [23]

Despite significant advancements, several challenges and open research questions remain in the field. Detecting increasingly sophisticated and obfuscated SQL injection attacks requires continuous innovation in both detection and analysis techniques. The need for more robust and generalizable machine learning models that can effectively handle the evolving threat landscape is also critical. Furthermore, the security implications of emerging technologies, such as the potential for prompt-to-SQL injections in applications integrated with large language models (LLMs) , warrant further investigation.

Future research directions could focus on developing more integrated and automated defense frameworks that seamlessly combine various detection and removal methods [6], [14], [23], [25]. Exploring the adoption of deep reinforcement learning for creating adaptive security systems that can learn and respond to new attack patterns in real-time is another promising avenue. The use of artificial intelligence for automated security auditing and vulnerability assessment could also significantly enhance the proactive identification and mitigation of SQL injection risks. Finally, advancements in differential privacy techniques could enable secure analysis of attack data to improve detection models while protecting sensitive information [27].

REFERENCES

1. D. Ajasa, H. Chizari, and A. Alam, “SQL injection detection using machine learning: a comparative study,” *Applied Sciences*, vol. 13, no. 4, p. 156, 2023.
2. J. M. Alkhathami and S. M. Alzahrani, “A survey of SQL injection detection and prevention techniques based on machine learning,” *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 11, pp. 781–787, 2022.
3. M. Alsalamah, H. Alwabli, H. Alqwifli, and D. M. Ibrahim, “SQL injection attacks detection and prevention using machine learning,” *The ISC International Journal of Information Security*, vol. 13, no. 3, pp. 1–10, 2021.
4. S. Nethala, S. Kampa, and S.R. Kosna, “Cyber Security Threats of Using Generative Artificial Intelligence in Source Code Management,” *Journal of Informatics and Web Engineering* **4**(2), 114–124 (2025).
5. A. Al Madani, S.A. Lashari, S.S. Uddin, A. Khan, M.N.A. Muhammad Attaullah, and D.A. Ramli, “Detecting Black Hole Attack using Support Vector Machine with XGBoosting in Mobile Ad-Hoc Networks,” *Journal of Informatics and Web Engineering* **4**(2), 209–224 (2025).
6. Appiah, E. Opoku-Mensah, and Z. Qin, “A hybrid deep learning model for detection of SQL injection attack,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 11, no. 1, pp. 249–266, 2020.
7. N. Augustine, A. Md. Sultan, M. Osman, and K. Sharif, “SQL injection attack detection using hybrid feature engineering and machine learning,” *JOIV: International Journal on Informatics Visualization*, vol. 8, no. 4, pp. 2131–2138, 2024.
8. N. Cahyadi, F. Pratama, A. N. Hidayanto, and B. I. Indrawan, “SQL injection detection using machine learning with improved feature extraction,” *Journal of ICT*, vol. 3, pp. 1–11, 2023.
9. N. S. Dasari, A. Badii, A. Moin, and A. Ashlam, “A novel approach for SQL injection detection using machine learning algorithms,” *Journal of Cybersecurity and Data Science*, vol. 1, no. 1, pp. 1–15, 2025.
10. H. Fu, C. Guo, C. Jiang, Y. Ping, and X. Lv, “Deep learning-based detection of SQL injection attacks in web applications,” *Sensors*, vol. 23, no. 11, p. 5213, 2023.
11. Z. Habibi, “SQL injection attack detection using decision tree classifier,” *International Journal of Advanced Academic Studies*, vol. 2, no. 1, pp. 6–10, 2020.
12. A. Jana and D. Maity, “SQL injection attack detection and prevention using machine learning techniques,” in *Proc. 11th Int. Conf. on Computing, Communication and Networking Technologies (ICCCNT 2020)*, Kharagpur, India: IEEE, 2020, pp. 1–6.
13. J. Kareem, A. M. F. Hussein, and H. J. Fadhil, “Detecting SQL injection attacks using supervised machine learning techniques,” *Asian Journal of Research in Computer Science*, vol. 10, no. 3, pp. 13–32, 2021.
14. S. Li, Y. Yu, J. Li, Y. Zhang, and Y. Zhang, “A convolutional neural network model for detecting SQL injection attacks,” *Applied Sciences*, vol. 15, no. 2, p. 571, 2025.
15. P. Lv, C. Yue, R. Liang, Y. Yang, S. Zhang, H. Ma, and K. Chen, “Detecting SQL injection attacks with syntax-aware neural networks,” in *Proc. 32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA: USENIX Association, 2023.
16. R. Manikandan, “Web application filter using regular expressions and sanitization for injection attack prevention,” 2020.
17. H. Matallah, A. N. Belkacem, and A. Bensaoucha, “Detecting SQL injection attacks using machine learning classifiers,” *International Journal of Information Technologies and Systems Approach*, vol. 14, no. 1, pp. 1–18, 2021.
18. S. Mishra, “Algorithms called gradient boosting are used from ensemble ML approaches to classify and detect SQLIAs,” 2019.
19. K. Ross, “SQL injection detection using machine learning techniques and multiple data sources,” Master’s Project, San Jose State University, 2018. [Online]. Available: <https://scholarworks.sjsu.edu/etd_projects/650/>
20. S. Sharma, “A novel approach for SQL injection attack detection during login phase using secure hash algorithm (SHA) based OTP generation,” 2018.
21. P. Sultana and N. Sharma, “SQL injection attack detection using random forest classifier,” in *Recent Developments in Electronics and Communication Systems*, 2023, pp. 276–282.
22. Y. Tang, Y. Zhou, J. Zhang, and Y. Zhang, “SQL injection detection using deep learning models,” *Knowledge-Based Systems*, vol. 190, p. 105528, 2020.
23. N. Yadav and N. M. Shekokar, “SQL injection attack detection using hybrid machine learning model,” in *Cyber Security: Threats and Challenges Facing Human Life*, Boca Raton, FL: Chapman and Hall/CRC, 2023, pp. 153–170.
24. W. Y. Win and H. H. Htun, “Detection of SQL injection attacks based on query pattern matching,” 2020.
25. Y. Yuan, “SQL injection detection with ensemble machine learning models,” *Applied Sciences*, vol. 13, no. 21, p. 11763, 2023.
26. H. Zhang and X. Zhang, “SQL injection detection using hybrid deep neural networks,” *Knowledge-Based Systems*, vol. 190, p. 105528, 2020.
27. Z. Zhuo, T. Cai, X. Zhang, and F. Lv, “SQL injection detection based on improved random forest algorithm,” *Computers & Security*, vol. 105, p. 102236, 2021.