SQL Injection Attack: Detection, Prioritization and Prevention

Gaurav Tiwari[1], Aman Singh Chauhan[2],
*Divyanshu Tripathi[3], Satyam Kumar[4] Swapnil Kaushal[5]

[1] Jims Engineering Management Technical Campus, Greater Noida, India
gauravtiwari8178@gmail.com
[2] Jims Engineering Management Technical Campus, Greater Noida, India
amanc228d@gmail.com
[3] Jims Engineering Management Technical Campus, Greater Noida, India
divyanshu8512@gmail.com
[4] Jims Engineering Management Technical Campus, Greater Noida, India
satyamkumar78400@gmail.com
[5] Jims Engineering Management Technical Campus, Greater Noida, India
swapnilkaushal.gn@jagannath.org

Abstract. ***SQL Injection remains a persistent and evolving threat to the security of web applications, as it allows attackers to change database queries to gain unauthorized access or seize control of systems. Despite numerous mitigation tools, SQLi continues to appear among the top OWASP vulnerabilities due to the limits of static detection and the lack of adaptive prevention. This paper introduces an integrated framework for detecting SQL Injection attacks, prioritizing, and preventing them by utilizing machine learning and automated query analysis. The system uses Node.js for automation and data handling, Python (scikit-learn) for the training and evaluation of classification models, and MySQL as the target database to simulate vulnerabilities. Supported with curated datasets of both clean and malicious SQL queries, the framework utilizes TF-IDF-based feature extraction for classifying SQLi attack types and assessing risk severity. A prevention layer within the Node.js middleware inspects queries in real time, thus allowing alerts or blocks based on the model's confidence scores. The design connects static vulnerability scanning with dynamic prevention, balancing detection accuracy with practical operability.***

Keywords-*SQL Injection, Web Application Security, MySQL, Threat detection, Real-time prevention*

P. Johri et al. (eds.), *Proceedings of the International Conference on Sustainable Computing and Artificial Intelligence (ICSCAI 2025)*, Advances in Engineering Research 298,
https://doi.org/10.2991/978-94-6239-674-6_43

1 Introduction

Nowadays, especially during the era of digital transformation, web applications play a significant role in modern enterprise operations, allowing organizations to render various dynamic services by managing user data at scale and with parallel transaction processing capabilities. However, this rapid adoption has grown the cyberattack surface of these applications, too, with SQL Injection remaining one of the most exploited vulnerabilities across all web systems [1,16] enabling unauthorized data access, bypassing authentication, or even the complete compromise of a system. Despite awareness for more than two decades and numerous mitigation efforts, SQLi has consistently been listed among the top vulnerabilities by OWASP and MITRE CWE, which shows that it is persistent and adaptable [2,16,17]. The traditional defense mechanisms, such as input sanitization, prepared statements, and static code analysis, have been efficient for simple attack vectors but cannot cope with continuously evolving, obfuscated, or multi-stage injection techniques. Moreover, a large portion of existing scanners and rule-based firewalls are also bound by static signature sets and high false-positive rates of their detection, which inhibits the successful detection of novel attack patterns. These shortcomings indicate an imminent need for a more intelligent, adaptive, and automated defense system that would go beyond mere detection and classify and prioritize SQLi threats according to severity and likelihood of exploitation. This paper proposes a machine learning–driven framework for the detection, prioritization, and prevention of SQLi attacks in web applications. The system marries the analytic strength of Python with the automation capability of Node.js, where MySQL acts as the experimental database layer. By leveraging TF-IDF–based feature extraction with supervised classification models, the framework identifies malicious SQL queries, assigning risk scores based on their predicted impact. A prevention layer embedded within the Node.js middleware allows real-time alerting. Because of this, a feedback loop of continuous detection and defense is created. Unlike traditional binary models of detection, the proposed approach introduces multi-class classification of various SQLi categories such as authentication bypass, blind injection, and remote code execution to provide a more granular level of analysis of threats. This increases situational awareness for both developers and security teams by allowing them to effectively prioritize remediation efforts. The main goal of this work has been to show that by integrating data-driven intelligence with automation, detection latency can be significantly reduced, precision improved, and real-time response enhanced against SQLi threats. This research, therefore, contributes to an adaptive security model that is able to bridge the long-standing gap between vulnerability identification and proactive prevention in modern web environments [3, 18].

2 Objectives

The main goal of this project, SQL Injection Attack: detection, prioritization and prevention, is to focus on the rising threat presented to modern applications and database systems due to SQL attacks and various challenges offered to a traditional vulnerability scanner. Following are the objectives:

2.1 To develop an intelligent multi-source SQLi detection framework:

The goal is to develop a unified system that will detect SQL Injection attacks by analyzing information from multiple sources, including HTTP requests, SQL query payloads, and database responses. The framework will go beyond traditional rule-based scanners by incorporating contextual data features that capture the intent and structure of queries. This multi-source approach enhances detection coverage, reduces false positives, and enables the system to adapt dynamically to both known and novel SQLi attack vectors in real-time web environments.

2.2 To implement a machine learning-based classification model for the identification of SQLi:

The proposed framework intends to take advantage of the capabilities provided by Python's scikit-learn library in order to construct a supervised learning model capable of classifying SQL queries as malicious. Feature extraction techniques like TF-IDF will be used to transform textual SQL payloads into numerical representations that express term significance and structural anomalies. This objective also includes the performance comparison of models such as Support Vector Machines, Random Forest, and Naïve Bayes to determine the most effective classifier for multi-class SQLi detection across different datasets and query patterns.

2.3 To include an automated prevention and alerting layer inside Nodejs middleware:

The key objective of this research is to embed real-time response capabilities directly into the Node.js environment in which the web application operates. This module will automatically intercept suspicious queries before they reach the database, perform model inference to confirm potential SQLi activity, and trigger preventive measures such as query blocking, logging, or administrator alerts. Such an automation layer can ensure immediate threat containment with minimum latency, bridging the gap between passive detection and active prevention in live systems

3. Literature Review

SQL Injection remains a strong threat to database-driven applications because it can manipulate the backend database and steal sensitive information [12]. Several detection and prevention mechanisms have been discussed in the past decade to address this challenge. Classic approaches, including AMNESIA [4], use runtime monitoring for reducing false positives without any interaction with clients and the backend. While they are effective for standard attacks, they generally fail to detect stored procedure attacks and remain bound to particular platforms such as Java web applications. Some other techniques apply regular expression pattern matching [5] and Aho– Corasick pattern matching [6] to identify malicious input patterns, reducing the rate of false positives. But unfortunately, both are vulnerable to bypassing through sophisticated attack vectors and merely consider the structure of SQL statements.

Over these limitations, tools such as SAFELI [7] analyze complex string conditions and possible execution flows, thus providing a more complete framework for the detection of SQLi attacks in .NET applications. However, due to its reliance on manually compiled attack libraries, scalability is limited. SQLIFIX [8] presented a language-independent approach that outperformed previous prepared statement replacement algorithms; however, it has some difficulties with specific SQL modifiers, batch processes, and certain parts of the code. Other hybrid approaches, like Node Centrality with SVM [9] and Query Trees and Fisher Score with SVM [10], utilize the benefits of machine learning together with reduced false positives for enhanced detection accuracy. These models have always been very effective at identifying anomalies in the structure of queries; however, many of them suffer from various drawbacks due to their limited dataset or inefficient coverage within diverse web application evaluations.

Recent approaches rely on deep learning and predictive machine learning techniques that yield remarkable improvements in both detection rates and adaptability. Multisource SQL Injection Detection, CNN & MLP, and LSTM based models utilize neural networks to parse massive amounts of data to find complex attack patterns and, simultaneously, to reduce overfitting. However, these solutions sometimes have limited generalizability due to small training datasets or an absence/partial consideration of alternative attack vectors [19, 20]. The recently proposed synBERT is very flexible for detecting unseen attacks, while the PNN has achieved low overhead and a low false positive rate; however, both of them are vulnerable to irrelevant features and complex deployment challenges [21, 22].

Table 1, represent the Comparative Analysis of Existing SQL Injection Detection Techniques

Research Paper	Solution Provided	Research Gaps
AMNESIA[4]	-Combines static analysis with runtime monitoring to detect SQL Injection attacks in Java-based web applications. Detects SQL Injection by identifying deviations between expected (safe) query structures and actual generated queries.	- Cannot detect SQL Injection attacks involving stored procedures, since stored procedure calls are not covered in its static query modeling process. - Requires source code access, meaning it cannot be used for closed-source or third-party applications.
Regular Expression pattern matching[5]	- Uses predefined regular expression (regex) rules to detect suspicious SQL keywords, patterns, and token sequences commonly found in SQL Injection attacks. - Simple to implement, making it a common early-stage protection mechanism in web applications.	- High false positives, since legitimate input may contain patterns resembling SQL keywords. - Cannot detect stored procedure-based SQL injections, since these may not match simple string patterns
Aho–Corasick pattern matching[6]	- Uses the Aho–Corasick multi-pattern matching algorithm to detect SQL Injection attempts by scanning input strings for multiple malicious patterns simultaneously. - Suitable for real-time detection due to its high-speed matching ability, even under heavy request loads.	- Heavily dependent on the completeness of its signature database-if a pattern is not included, the attack goes undetected. - Not suitable as a standalone security mechanism, because it lacks adaptability and cannot generalize beyond predefined patterns.
SAFELI[7]	Performs static analysis with symbolic execution to detect potential SQL Injection vulnerabilities in source code before runtime.	Requires manual compilation of the SQL Injection attack library, which depends on security experts and is not scalable.

	Builds models of SQL query construction and traces how input variables propagate through the application.	-Significant computational overhead when analyzing large codebases with many string operations and branching paths.
SQLIFIX[8]	Implements automated patch generation, replacing unsafe query-building patterns with secure constructs such as parameterized queries. -Helps developers remediate vulnerabilities quickly, reducing reliance on manual code inspection and lowering the risk of oversight.	- Static analysis cannot identify runtime-dependent vulnerabilities, such as queries generated via reflection or dynamic data sources. - Focuses only on detection and repair, offering no support for runtime monitoring or long-term preventive strategies.

Table 1: Comparative Analysis of Existing SQL Injection Detection Techniques and Their Limitations

4. Key Technology IN SQLi Detector Tool

This SQL Injection Detection Tool uses a modern mix of programming frameworks and database management systems with machine learning algorithms to achieve the goal of real-time, precise identification of SQL Injection vulnerabilities[11,12]. It integrates the technologies in a scalable, platform-independent tool with a Command Line Interface for easy usage by developers, researchers, and analysts in cybersecurity [23, 25].

4.1 Node.js for CLI Interface and Orchestration:

Node.js powers the Command Line Interface, providing a fast event-driven execution environment that easily interfaces with back-end APIs. The asynchronous architecture makes it quick to scan many files or queries without blocking I/O operations. Node.js modules provide functions for processing user inputs, parsing SQL queries, and firing requests to the detection engine hosted on the FastAPI server [13,14].

4.2 Integration and Automation:

By combining Node.js, FastAPI, and MySQL, the system achieves full-stack integration between detection, classification, and storage. The CLI-based interface ensures automation-friendly operation, allowing integration with version control systems and build pipelines for continuous security assessment [15, 24].

5. Methodology

5.1 Overview:

The proposed SQLi Scanner follows a modular, MV3-compliant methodology that separates page instrumentation, network replay, detection logic, and reporting. Content scripts are limited to safe, read-only DOM discovery, whereas all cross-origin scans and timing measurements run in the background service worker. This separation ensures browser compatibility, minimizes page impact, and enables reproducible testing aligned with OWASP WSTG guidance.

5.2 Technology:

The implementation uses JavaScript throughout with a lightweight DevTools panel UI. The service worker orchestrates requests, computes normalized body hashes, aggregates timing medians, and classifies evidence. Storage uses Chrome storage APIs for settings, queues, and findings. Report export relies on the downloads API. CI/CD pipelines execute e2e smoke scans against lab targets before merges.

5.3 Architecture:

The system architecture is modular to facilitate replacement and scaling of individual stages (discovery, replay, detection, export). The service worker acts as the central coordinator, while the DevTools panel provides operator control, live logs, and findings. Integration points enable CI triggers and cloud logging.

5.4 Payload Design and Safety Defaults:

Error-based probes: Minimal, non-destructive quotes and syntax nudges to elicit DB error signatures using a centralized regex library. Booleanbased blind in True/false pairs with N-repetitions; comparisons use status, content-length, normalized body hash, and DOM markers with consensus thresholds. ORDER BY/UNION NULL patterns to infer column counts without exfiltration; early bail on WAF indicators.

6. Result

6.1 Experimental Setup:

The SQLi Scanner Chrome Extension (Manifest V3) was evaluated on a controlled testbed comprising: (i) deliberately vulnerable applications (DVWA) running in Docker, (ii) two production-like demo apps with parameterized backends (Node.js/Express + MySQL and Java/Spring + PostgreSQL), and (iii) a synthetic target that echoes request metadata for differential analysis. Safe Mode was enabled by default (non-destructive payloads), and time-based tests were off unless explicitly stated. Cross-origin replays and all HTTP probes were executed from

the background service worker, with content scripts restricted to DOM discovery, in alignment with MV3 constraints. Each technique (error-based, Boolean based, union probing, optional time-based) used N=3 repetitions and median-based thresholds to reduce noise.

6.2 Detection Performance:

Across 12 target pages and 86 unique parameters (query, form, and JSON keys), the scanner identified 29 injection-prone parameters, with 24 confirmed and 5 flagged as low-confidence pending developer verification. Technique-wise contributions were: error-based (17 confirmed), boolean-based blind (9 confirmed), union probing (6 indicative leads, 3 confirmed), and time-based blind (2 additional confirmations when enabled). On DVWA (low), all seeded vulnerabilities were found; on DVWA (medium), boolean-based and union probing maintained reliability; on DVWA (high), detections decreased as expected, with error-based alerts surfacing only where verbose error handling was re-enabled. In the two hardened demo apps, zero confirmed findings were reported, validating low falsepositive behavior when parameterized queries are consistently used.

6.3 Runtime Characterstics:

Average scan duration per page (safe mode, time-based off) was 45–90 seconds depending on parameter count and network variability; enabling time-based tests increased duration by approximately 30–60 seconds per vulnerable parameter tested. Concurrency caps and backoff on repeated 4xx/5xx responses prevented target saturation. All cross-origin fetches executed from the service worker, honoring host permissions; no mixed-context or CORS UI prompts were required. The DevTools panel remained responsive, with live logs and incremental findings updates observable during scans.

7 Conclusion:

SQL Injection continues to be a critical vulnerability that undermines the integrity, confidentiality, and availability of web applications and databases. Through this research, we explored the depth of SQL Injection attacks, their techniques, and their long-standing impact on information security. The study highlights that traditional prevention methods, such as input sanitization and static analysis, though effective to some extent, fail to address more sophisticated and adaptive attack patterns. Therefore, the need for intelligent and automated detection tools has become essential in modern web security frameworks.

The proposed SQL Injection Detection Tool aims to fill this gap by integrating advanced technologies such as machine learning, pattern analysis, and behavior-

based anomaly detection. This approach focuses not only on identifying malicious queries but also on providing insights for proactive prevention and risk mitigation. The modular design will allow future integration with diverse web environments, enabling developers to enhance protection without compromising system performance.

In conclusion, the tool not only detects SQL Injection attempts but also promotes secure coding practices through its preventive recommendations. Future work can extend this project toward integrating deep learning-based models, expanding detection to other web vulnerabilities such as Cross-Site Scripting (XSS), and developing a browser extension or plugin for continuous runtime monitoring. By merging automation, intelligence.

Reference:

[1]. ScienceDirect Topics, “Web-Based System,” Computer Science Section, Elsevier, 2025.

[2]. E. Kumi, H. V. Osei, S. Asumah, and A. Yeboah, “The impact of technology readiness and adapting behaviours in the workplace: a mediating effect of career adaptability,” *Future Business Journal*, vol. 10, Article 63, SpringerOpen, 2024.

[3]. L.Akritidis and D.Katsaros,“Modern Web Technologies,” *ResearchGate*, Aug. 9, 2025.

[4]. Halfond WG, Orso A. Preventing SQL injection attacks using AMNESIA. In: Proceedings of the 28th international conference on software engineering. 2022,

[5]. Hadabi A, Elsamani E, Abdallah A, Elhabob R. An efficient model to detect and prevent SQL injection attack. J Karary Univ Eng Sci 2022.

[6]. Kini S, Patil AP, Pooja M, Balasubramanyam A. SQL injection detection and prevention using Aho-Corasick pattern matching algorithm. In: 2022 3rd international conference for emerging technology. INCET, IEEE; 2022, p. 1–6.

[7]. Fu X, Qian K. SAFELI: SQL injection scanner using symbolic execution. In: Proceedings of the 2008 workshop on testing, analysis, and verification of web services and applications. *2023, p. 34–9.*

[8]. Siddiq ML, Jahin MRR, Islam MRU, Shahriyar R, Iqbal A. Sqlifix: Learning based approach to fix sql injection vulnerabilities in source code. In: 2021 IEEE international conference on software analysis, evolution and reengineering. SANER, IEEE; 2021, p. 354–64.

[9]. Kar D, Sahoo AK, Agarwal K, Panigrahi S, Das M. Learning to detect SQLIA using node centrality with feature selection. In: 2016 *international conference on computing, analytics and security trends*. CAST, IEEE; 2024, p. 18–23.

[10]. Ladole A, Phalke M. SQL injection attack and user behavior detection by using query tree, fisher score and SVM classification. Int Res J Eng Technol 2016;3(6):1505–9.

[11]. Horseman J. CVE-2024-29824 deep dive: Ivanti EPM SQL injection remote code execution vulnerability. *2024, horizon3.ai*

[12]. Cybersecurity and Infrastructure Security Agency (CISA), *Secure by Design Alert: Eliminating SQL Injection Vulnerabilities in Software*, CISA Central, 2024.

[13]. M..R.Chinchilla, *CVE Prioritizer Tool*, GitHub Repository, 2023.

[14]. P.Garrity, *Taking an Evidence-Based Approach to Vulnerability Prioritization*, VulnCheck, 2024.

[15]. S.J.Vaughan-Nichols, *NVD Slowdown Leaves Thousands of Vulnerabilities Without Analysis Data*, The Register, 2024.

[16]. F. Y. Hernawan, I. Hidayatulloh, and I. F. Adam, "Hybrid method integrating SQL-IF and Naïve Bayes for SQL injection attack avoidance," *Journal of Engineering and Applied Technology*, vol. 1, no. 2, pp. 85–96, Aug. 2020

[17]. H. K. Khanuja, P. Gadekar, S. Kulkarni, S. Kulkarni, and S. More, "Web application security scanning using machine learning," *International Journal of Engineering Research in Computer Science and Engineering*, vol. 8, no. 8, pp. 21–27, Aug. 2021.

[18]. B. A. Pham and V. H. Subburaj, "An experimental setup for detecting SQLi attacks using machine learning algorithms," *Journal of the Colloquium for Information Systems Security Education*, vol. 8, no. 1, pp. 1–5, 2020.

[19]. Y. Abdulmalik, "An improved SQL injection attack detection model using machine learning techniques," *International Journal of Innovative Computing*, vol. 11, no. 1, pp. 53–57, 2021.

[20]. O. Morufu, R. A. Egigogo, I. Idris, and R. G. Jimoh, "A Naïve Bayes-based pattern recognition model for detecting and categorizing SQL injection attacks," *International Journal of Cyber-Security and Digital Forensics*, pp. 189–199, 2018.

[21]. T. P. Latchoumi, M. S. Reddy, and K. Balamurugan, "Applied machine learning predictive analytics for SQL injection attack detection and prevention," *European Journal of Molecular & Clinical Medicine*, vol. 7, no. 2, pp. 3543–3553, 2020.

[22]. M. Kavitha, V. Vennila, G. Padmapriya, and A. R. Kannan, "Prevention of SQL injection attack using an unsupervised machine learning approach," *International Journal of Aquatic Science*, vol. 12, no. 3, pp. 1413–1424, 2021.

[23]. M. A. Azman, M. F. Marhusin, and R. Sulaiman, "Machinelearning-based technique to detect SQL injection attack," *Journal of Computer Science*, pp. 296–303, 2021.

[24]. U. Farooq, "Ensemble machine learning approaches for detection of SQL injection attack," in *Proc. Int. Conf. Convergence of Smart Technologies (IC2ST)*, Pune, 2021.

[25]. S.Mishra, "SQL Injection Detection Using Machine Learning," *Master's Projects*, San José State University, May 2019.