A Machine Learning Approach for Detecting SQL Injection Attacks in Database System

Abdul Rahman K[1], Ajmal Ahamed A[2], J. Maria Shyla[3]
[1,2,3]*Department of Information Technology, Nehru Arts and Science College, Coimbatore, India*

***Abstract*—SQL Injection (SQLi) remains a pervasive threat to database-driven applications, often bypassing traditional signature-based defenses through polymorphic and obfuscated query patterns. This paper proposes an intelligent detection framework leveraging supervised Machine Learning (ML) to classify SQL queries as either benign or malicious. By transitioning from rigid rule-based filtering to dynamic feature extraction and pattern recognition, the proposed system demonstrates high classification accuracy and adaptability to zero-day vulnerabilities. Our methodology utilizes a multi-algorithmic approach—including Random Forest and Support Vector Machines—to provide a scalable, real-time security layer for modern database architectures.**

***Index Terms*—SQL Injection, Machine Learning, Cybersecurity, Database Security, Anomaly Detection, Supervised Learning.**

I. INTRODUCTION

As web-centric ecosystems have evolved, it has fundamentally changed the cybersecurity world, and the database layer is now considered the "crown jewel" for sophisticated cyber adversaries. One of the most enduring threats is SQL Injection (SQLi), which is a critical security concern that involves adversaries attempting to manipulate back-end database queries through unsensitized input. SQLi is particularly significant as it allows adversaries to have the ability to spoof identities, edit or delete sensitive information, and in the worst case, have total administrative access over the entire database server.

Traditionally, Web Application Firewalls (WAFs) have been used as the primary means of defense. Unfortunately, although WAFs provide an essential foundation, it is now considered an inadequate response to emerging threats. These outdated solutions are largely dependent upon static detection mechanisms, which compare incoming data streams to known attack signatures. This is an inherently weak mechanism, which fails to address the polymorphism of modern exploit vectors and is frequently evaded through obfuscation or 'zero-day' attacks, which possess no signature.

To address this systemic issue, this research introduces a paradigm shift from reactive filtering to a more proactive and "intelligent" form of detection, utilizing a machine learning-based approach. In this regard, this research introduces a more accurate and efficient form of SQLi detection by conceptualizing it as a binary classification problem. This allows us to look beyond the simplistic nature of string-based detection. Instead, our proposed system utilizes complex models that are trained on the underlying "syntax and intent" of SQL-based queries. This allows us to accurately identify legitimate user requests from malicious injections, even for highly sophisticated and unseen attack vectors.

Key Enhancements in This Version

- Thematic Depth: Expanding upon the rationale for why databases are targeted in the first place ("crown jewel" approach), and the inherent limitation of WAFs in the first place (polymorphism and zero-day attacks).
- Technical Nuance: Using more descriptive and academic wording such as "polymorphic nature," "obfuscation techniques," and "syntax and intent" in place of more general and vague terms.
- Structural Flow: The structure now flows more logically from the Problem statement to the Current Limitation and then the Proposed Solution.

II. CRITIQUE OF EXISTING SYSTEMS

Taxonomy of Current Defensive Methodologies

The current cybersecurity paradigms classify the various database protection methodologies into three technical tiers. However, these tiers are based on the logic of exclusion rather than intelligence.

- Signature based detection systems basically check incoming SQL queries against a big list of known bad stuff, like attack strings or malicious payloads they have on record. If something like that common tautology shows up, say OR 1=1 followed by a comment, it gets flagged right away. I think it's kind of limited though because it only catches what it has seen before, nothing new or sneaky.
- Heuristic filtering uses these allow lists or deny lists to control what interacts with the database. It blocks out high risk words or characters, for example DROP or UNION, even semicolons at the app level. That works okay for basic cleaning up queries, but it seems like it struggles with context. Like, how do you tell if a command is actually bad or just part of some legit admin job that is more complicated.
- Static web application firewalls sit at the edge of the network and grab traffic before it hits the server. They use regex patterns, you know regular expressions, to spot and filter suspicious things. Its a good first line of defense I guess, but the rules are so fixed that people can get around them easily, maybe by encoding differently or tweaking the syntax a little. That part feels a bit weak, does it not.

Overall, these methods have their places, but none seem perfect for everything. The signature one is reactive, heuristics miss nuances, and WAFs can be tricked.

Traditional defenses, you see them everywhere in cybersecurity. But honestly, they have these big problems built right in. Sophisticated attackers just keep finding ways around them, exploiting the weak spots over and over. It seems like no matter how common these measures are, they still fall short against real threats. I am not totally sure why they havent improved more, but that is the part that stands out.

- Architectural Inelasticity and "Zero-Day" Risk: The most glaring omission in signature-based systems is their inability to adjust to the unknown. Signature-based systems are strictly reactive systems, which means that if an attacker can devise a new "Zero-Day" exploit or use an unconventional obfuscation technique that does not fit any known signature, the system remains blind to it. This leads to a perpetual "cat-and-mouse" game, with the defender always remaining a step behind.
- The Burden of High False Positives: The problem with these systems is that they are based on the blunt instrument of keyword blocking and therefore often incorrectly identify complex and legitimate queries as suspicious. Advanced administrative activities, such as those that involve deeply nested sub-queries, complex and multifaceted joins, and unusual data types, are common sources of false positives.
- Unsustainable Maintenance Overhead: These systems require constant human intervention in order to be at least moderately effective. Security analysts need to continually update signature libraries and tweak rules in order to keep up with new exploit techniques that are being developed. This is a tremendous overhead on IT departments and creates a huge window of exposure that exists between the time that a new threat is first encountered and the time that its associated signature is available.

III. PROPOSED METHODOLOGY

The proposed framework facilitates a fundamental shift in the defensive paradigm, moving from a philosophy of "static matching" to one of "dynamic learning." Rather than searching for a specific sequence of characters, the core objective is to construct a predictive model that internalizes the underlying syntax, logical structure, and statistical distribution of legitimate SQL traffic. By establishing a robust baseline of "normal" behaviour, the system can autonomously flag deviations that signal malicious intent.

The technical realization of this framework is divided into three sophisticated operational phases:

1. Advanced Data Acquisition & Preprocessing

To ensure the model is both resilient and accurate, data is sourced from diverse, curated repositories containing tens of thousands of labelled SQL samples (e.g., Kaggle, OWASP, and CVE-linked datasets). Before training, the raw query data undergoes a rigorous "cleaning" process to eliminate noise:

- Normalization: All queries are converted to a uniform lowercase format, and excessive whitespace or hidden characters are stripped to prevent simple evasion techniques.
- Lexical Tokenization: The system decomposes complex queries into their atomic components—identifying specific keywords, operators, and literals. This allows the model to analyse the "grammar" of the SQL statement rather than just the text string.

2. Multidimensional Feature Engineering

This represents the most critical phase of the pipeline, where qualitative SQL strings are transformed into quantitative numerical vectors. By extracting high-signal features, the model can "see" the fingerprints of an attack:

- Syntax Complexity Metrics: The system calculates the density of specific command keywords (e.g., SELECT, UNION, EXEC, INFORMATION_SCHEMA). A sudden spike in command keywords often indicates an injection attempt.
- Character Distribution Analysis: This tracks the frequency and positioning of high-risk special characters (e.g., ', --, *, /*).
- Logical Risk Assessment: The model looks for the presence of tautologies—logical statements that are always true (e.g., 1=1 or 'a'='a')—which are hallmark indicators of authentication bypass attempts.
- Structural Anomalies: Standard application queries typically follow a predictable length and complexity. Malicious payloads often deviate significantly from these established averages, providing a clear statistical red flag.

3. Model Selection, Training, and Evaluation

The final phase transforms raw data into a functional security engine by prioritizing a balance between detection accuracy and operational speed. This stage ensures the system can identify sophisticated threats in real-time without interrupting legitimate traffic:

- Algorithmic Evaluation: The framework compares three distinct approaches: Random Forest for its ability to handle non-linear attack patterns, Support Vector Machines (SVM) to establish precise boundaries in high-dimensional vector spaces, and Logistic Regression to serve as a high-speed probabilistic baseline.
- Validation Protocols: To ensure the model generalizes to "Zero-Day" exploits rather than just memorizing data, the system utilizes an 80/20 train-test split combined with k-fold cross-validation. This rigorous testing ensures the model remains resilient against evolving query structures.
- Performance Benchmarking: Success is measured through the F1-Score to maintain a balance between precision and recall. Additionally, Inference Latency is strictly monitored to ensure that every query is analyzed in milliseconds, maintaining a seamless user experience.

IV. SYSTEM ARCHITECTURE

This framework is highly flexible and can be integrated into the program logic or as a database proxy. As a result, all queries are vetted before they are sent to the DBMS through four important stages:

- Interception Layer: This layer works as the primary gatekeeper, where all incoming SQL requests are caught in real-time before any unverified query reaches the data layer.
- Extraction Engine: This layer processes all the queries caught by the Interception Layer and calculates a high-dimensional feature vector, converting the raw syntax of the query into machine-readable data.
- Inference Engine: Using a pre-trained ML model, this layer evaluates the extracted vector and assigns a probability value that classifies the query as Benign or Malicious.
- Action Module: This module carries out defensive action based on the risk value. If the value crosses a certain threshold, the module blocks the query and generates a detailed alert for further analysis.

V. EVALUATION METRICS

To assess the efficacy of the proposed framework, the accuracy of the proposed model is measured in terms of the confusion matrix. The accuracy of the proposed system is ensured by the four fundamental metrics:

- Accuracy: This represents the overall percentage of correct predictions, including both benign and malicious SQL queries.
- •Precision: This represents the reliability of the proposed system. Precision is measured by calculating the ratio of all the true malicious queries to the total number of queries detected as malicious.
- Recall (Sensitivity): This represents the efficacy of the proposed system in detecting all the threats in the dataset. High recall is important to avoid the possibility of Zero-Day injection attacks.
- F1-Score: This represents the reliability of the proposed system by calculating the harmonic mean of Precision and Recall. This is the most important score that needs to be maximized to ensure the highest security without compromising the usability of the system.

VI. ANALYSIS OF ADVANTAGES & CHALLENGES

- Predictive Power: It can detect new "Zero-Day" exploits by using structural logic instead of string matching.
- Operational Automation: It can remove the necessity for manual regex maintenance by learning to automatically differentiate between malicious and benign traffic.
- System Scalability: It can efficiently monitor huge volumes of traffic in cloud-native, distributed database environments.
- Data Dependency: It relies on diverse data sets for reliability, as limited data can lead to false positives.
- Latency Constraints: It needs to be highly optimized to avoid compromising database response times with security checks.
- Adversarial Risks: It is exposed to the threat of specific queries designed to evade detection by mimicking legitimate queries.

VII. CONCLUSION & FUTURE DIRECTIONS

This study also establishes that Machine Learning (ML) is an effective, intelligent solution in place of traditional SQL injection (SQLi) detection techniques. Prioritizing query behaviour analysis in place of traditional string matching is key in further enhancing database security.

The following is where we believe this defense system is headed in the future:

Future Research Directions

- Advanced Neural Architectures: Utilizing Deep Learning techniques, specifically Recurrent Neural Networks (RNNs) or Transformers, in order to develop a more sophisticated state of knowledge with regards to complex query situations and natural language.
- Hybrid Defense Systems: Creating a multi-faceted security system that combines Machine Learning-based detection with traditional Query Sanitization in order to develop a more comprehensive system that is not limited by any single solution.
- Edge-Side Detection: Moving the detection system to the Network Edge. By doing so, we may be able to eliminate threats even before they reach the main infrastructure.

VIII. TECHNICAL STACK

- Language: Python 3.x
- Libraries: Scikit-learn (Classification), Pandas (Data handling), NLTK (Tokenization).
- Database: PostgreSQL / MySQL for simulation.
- Environment: Jupyter Notebooks for R&D; Flask for API-based detection services.

REFERENCES

[1] Augustine, N., Sultan, A. B. M., Osman, M. H., & Sharif, K. Y. (2024). Application of artificial intelligence in detecting SQL injection attacks. JOIV: International Journal on Informatics Visualization, 8(4), 2131-2138.

[2] Casmiry, E., Mduma, N., & Sinde, R. (2025). Enhanced SQL injection detection using chi-square feature selection and machine learning

classifiers. Frontiers in Big Data, 8, 1686479.

[3] Farooq, M. S., et al. (2023). Ensemble machine learning approaches for detection of SQL injection attack. Proceedings of the International Conference on Emerging Technologies in Computing.

[4] Ghozali, I., Asy'ari, M., & Rahayu, S. (2022). A novel SQL injection detection using Bi-LSTM and TF-IDF. 2022 7th International Conference on Information Technology (ICIT), 112-117.

[5] Hacham, A., & Uçan, O. N. (2023). Detection of malicious SQL injections using SVM and KNN algorithms. 2023 7th International Symposium on Innovative Approaches in Smart Technologies (ISAS), 1-6.

[6] Ibrahim, A., et al. (2025). An efficient SQL injection detection with a hybrid CNN & random forest approach. Journal of Information Systems Engineering and Management, 10(1), 2979.

[7] Oudah, M. A. M., & Marhusin, M. F. (2024). SQL injection detection using machine learning: A review. Malaysian Journal of Science Health & Technology, 10(1), 39-49.

[8] Rosca, C. M., Stancu, A., & Popescu, C. (2024). Machine learning models for SQL injection detection. Electronics, 13(17), 3420.

[9] Triloka, J., Hartono, H., & Sutedi, S. (2022). Detection of SQL injection attack using machine learning based on natural language processing. International Journal of Artificial Intelligence Research, 6(2), 1-10.

[10] Zulu, J., Han, B., Alsmadi, I., & Liang, G. (2024). Enhancing machine learning based SQL injection detection using contextualized word embedding. Proceedings of the 2024 ACM Southeast Conference, 215-221.