International Journal of Engineering Research and Science & Technology

www.ijerst.org

ISSN : 2319-5991

Vol. 22 No. 3 (2026)

ijerst.editor@gmail.com

editor@ijerst.com

ISSN 2319-5991 www.ijerst.com

Peer Reviewed, Referred & Indexed Journal

Research Paper

Predictive Web Application Security Using Intelligent SQL Injection Detection

D. Aravind Reddy[1], G.Rajini[2]
[1]MCA Student , Dept of MCA , Ellenki College of Engineering and Technology , Hyderabad
[2]Assistant Professor , Dept of MCA , Ellenki College of Engineering and Technology , Hyderabad

***Abstract*— Web applications have become an essential part of modern businesses, making them a common target for cyberattacks. Among various security threats, SQL injection remains one of the most dangerous because it allows attackers to manipulate database queries and gain unauthorized access to sensitive information. Traditional detection techniques often rely on predefined rules or signatures, which are less effective against newly emerging attack patterns. This work presents an intelligent approach for improving web application security through the prediction and detection of SQL injection attacks using machine learning techniques. The proposed framework processes SQL queries by performing data cleaning, feature extraction, and text preprocessing before training an ensemble classification model. The trained model distinguishes normal queries from malicious ones with high accuracy, enabling early identification of potential attacks. Performance is evaluated using standard metrics such as accuracy, precision, recall, and F1-score to verify the effectiveness of the system. The developed framework also provides a simple interface for analyzing new SQL queries and predicting their security status. This approach supports proactive protection of web applications by enabling faster, more reliable, and automated detection of SQL injection vulnerabilities.**

***Keywords*— SQL Injection Detection, Web Application Security, Machine Learning, Ensemble Learning, Cybersecurity, Vulnerability Detection, SQL Query Analysis, Intrusion Detection, Feature Extraction, Predictive Security.**

I. INTRODUCTION

Web applications have become an essential part of modern businesses, education, healthcare, banking, and e-commerce, making web security more important than ever. As the amount of sensitive information stored in databases continues to grow, cyber attackers increasingly target web applications to gain unauthorized access to confidential data. SQL injection is one of the most common attacks because it exploits poorly validated user input to manipulate database queries and compromise application security [2]. Such attacks can lead to data theft, unauthorized modifications, financial losses, and disruption of online services. Detecting these vulnerabilities at an early stage is therefore necessary to reduce security risks and protect valuable information. Researchers have shown that software quality and secure coding practices play an important role in minimizing vulnerabilities during application development [3]. These challenges highlight the need for intelligent techniques that can accurately identify SQL injection attacks before they affect web applications.

Conventional security approaches, including manual code inspection and rule-based vulnerability scanners, have been widely used to detect security flaws in web applications. Although these techniques are useful for identifying known vulnerabilities, they often struggle to recognize newly emerging attack patterns and complex SQL injection attempts [19]. Static code analysis tools can detect programming errors before software deployment, but they may generate false alarms or overlook sophisticated vulnerabilities [6]. Modern web applications contain large volumes of source code and database interactions, making manual analysis difficult and time-consuming. Automated security testing techniques have therefore become an important part of secure software development, helping developers identify vulnerabilities more efficiently [7]. These limitations encourage the development of intelligent prediction systems capable of learning attack characteristics from real-world datasets.

Machine learning has emerged as a promising solution for improving web application security because it can automatically identify hidden patterns within SQL queries and classify malicious activities

with high accuracy. Instead of depending only on predefined signatures, learning-based models analyze extracted features to distinguish normal queries from injection attacks. Previous studies have shown that software metrics and code complexity information can be effectively used for predicting software defects and security vulnerabilities [18]. Feature extraction and classification techniques also improve the capability of machine learning models to identify complex attack behaviors from structured datasets [14]. By combining preprocessing, feature engineering, and intelligent learning algorithms, prediction models become more adaptable to evolving cyber threats and provide faster detection than conventional approaches.

The proposed framework applies intelligent prediction techniques to strengthen web application security against SQL injection attacks. Initially, SQL queries are collected and preprocessed by removing unnecessary information and extracting meaningful features for model training. The processed data are then used to train a machine learning classifier capable of distinguishing legitimate database queries from malicious ones. Earlier studies have demonstrated that analyzing source code characteristics and program structures can significantly improve vulnerability detection performance [13]. Research on vulnerability prediction has also shown that software change metrics and historical information contribute to more reliable security analysis [10]. By integrating these concepts into an automated prediction framework, the proposed system provides faster and more accurate identification of SQL injection attacks.

This work presents a predictive framework for detecting SQL injection attacks in web applications using intelligent machine learning techniques. The proposed system focuses on improving prediction accuracy while reducing the effort required for manual security analysis. Experimental evaluation is performed using standard performance measures such as accuracy, precision, recall, and F1-score to verify the effectiveness of the developed model. Previous research has shown that automated vulnerability detection methods can significantly improve software security by identifying weaknesses before deployment [11]. Intelligent security prediction techniques continue to evolve by combining machine learning with efficient feature analysis to detect complex attack patterns [12]. The proposed framework contributes to secure web application development by providing an effective, scalable, and reliable solution for identifying SQL injection attacks in modern computing environments.

II. RELATED WORK

Basili et al., [1996] [3] presented a study on validating object-oriented design metrics as indicators of software quality. Their research investigated how different software design metrics could be used to estimate the quality and maintainability of software systems during development. The authors analyzed various object-oriented characteristics and examined their relationship with software defects and reliability. The study showed that well-designed software metrics help developers identify potential problem areas before deployment. They also emphasized that measuring software complexity at an early stage can reduce maintenance effort and improve software performance. Their findings demonstrated that software quality assessment should become an integral part of the development process rather than being performed only after implementation. This work provides valuable guidance for building secure and reliable applications by using software metrics to identify weaknesses early, thereby reducing the possibility of security vulnerabilities and improving overall software quality.

Evans and Larochelle, [2002] [6] proposed a lightweight static analysis approach for improving software security through automated vulnerability detection. Their work focused on identifying programming errors during software development without executing the source code. The authors demonstrated that static analysis techniques are capable of detecting many common security issues, including memory-related vulnerabilities, before software deployment. They explained that early vulnerability detection reduces development costs and minimizes the chances of security breaches in deployed applications. The study also highlighted that automated analysis tools can support developers by identifying coding mistakes that are difficult to detect manually. Their results showed that integrating static analysis into the software development process significantly improves application security and software reliability. This research provides an important foundation for developing intelligent vulnerability detection systems that assist developers in creating secure software with fewer coding errors.

Godefroid et al., [2012] [7] introduced the SAGE framework, an automated white-box fuzzing technique designed to discover software vulnerabilities through intelligent test case generation. Their research focused on systematically exploring different program execution paths to uncover hidden software defects and security weaknesses. Unlike traditional testing methods, the proposed approach generated new test inputs automatically, increasing the probability of finding complex vulnerabilities. The authors demonstrated

that white-box fuzzing can effectively identify critical software flaws before attackers exploit them. They also emphasized that automated security testing reduces manual effort while improving the coverage of software testing. The experimental results confirmed that the framework successfully detected numerous previously unknown vulnerabilities in real-world applications. Their work highlights the importance of intelligent testing techniques for strengthening software security and provides a valuable reference for automated vulnerability detection research.

Ma et al., [2016] [13] proposed a static analysis method for detecting buffer overflow vulnerabilities in C and C++ source code using Abstract Syntax Tree (AST) techniques. Their approach focused on analyzing program structure instead of executing the application, enabling security flaws to be identified during the development phase. The authors extracted structural information from source code and used it to recognize vulnerable programming patterns more accurately. Their experimental results showed that syntax tree analysis improved the detection of buffer overflow vulnerabilities while reducing unnecessary false warnings. The study emphasized that early identification of coding errors helps developers build safer and more reliable software systems. This research demonstrates the effectiveness of source code analysis for vulnerability detection and provides useful insights into designing intelligent software security tools capable of identifying programming weaknesses before software deployment.

Shin and Williams, [2008] [18] developed an empirical model for predicting software security vulnerabilities using code complexity metrics. Their study examined whether characteristics such as software complexity and design metrics could help estimate the likelihood of security weaknesses in software systems. The authors found that modules with higher complexity were generally more prone to vulnerabilities than simpler components. They demonstrated that complexity metrics could be used as effective indicators for identifying high-risk software modules before deployment. The research also suggested that predictive models can assist developers in prioritizing code inspection and security testing efforts. Their findings support the idea that software quality metrics play an important role in improving secure software development practices. This work serves as a strong foundation for applying intelligent prediction techniques to software vulnerability analysis and reducing security risks through early identification of vulnerable code.

III. DATASET DETAILS

The dataset used in this project contains different types of web requests and query statements collected for detecting vulnerabilities in web applications. It includes both normal requests and malicious inputs representing attacks such as SQL Injection (SQLi), Cross-Site Scripting (XSS), and Remote File Inclusion (RFI). Each record consists of a query or request along with its corresponding class label, making the dataset suitable for supervised machine learning. The data is organized in a structured tabular format, allowing efficient processing and analysis. Before model training, the dataset is examined to remove duplicate records, handle missing values, and ensure that each class is properly represented. Since the data consists of textual queries, Natural Language Processing (NLP) techniques are applied to extract meaningful information from the input. This dataset provides the foundation for training the machine learning model to accurately identify different categories of web application vulnerabilities.

To prepare the dataset for classification, several preprocessing techniques are performed to improve data quality and model performance. Initially, text cleaning is carried out by removing stop words, unnecessary symbols, and irrelevant characters from each query. The cleaned text is then converted into numerical feature vectors using the TF-IDF (Term Frequency–Inverse Document Frequency) technique, enabling machine learning algorithms to process textual data effectively. The dataset is randomly shuffled to avoid bias and to ensure balanced learning across all vulnerability classes. After preprocessing, the data is divided into training and testing sets, where approximately 90% of the records are used for training and the remaining 10% are reserved for testing. This data split allows the ensemble classifier to learn important attack patterns while evaluating its performance on previously unseen samples, ensuring accurate and reliable vulnerability prediction.

IV. PROPOSED METHODOLOGY

The proposed system follows a structured approach to detect and predict web application vulnerabilities using an ensemble machine learning model. Initially, a dataset containing normal web requests and malicious queries, including SQL Injection (SQLi), Cross-Site Scripting (XSS), and Remote File Inclusion (RFI), is collected and analyzed. The dataset is then preprocessed to improve its quality before model training. This process includes removing duplicate records, eliminating stop words through Natural Language Processing (NLP), and converting the textual queries into numerical feature vectors using the TF-IDF technique. The processed

dataset is randomly shuffled to ensure an unbiased distribution of all vulnerability classes. After preprocessing, the dataset is divided into training and testing sets, where the training data is used to build the prediction model and the testing data is used to evaluate its performance on unseen samples.

Once the dataset preparation is completed, an ensemble machine learning model combining Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Naïve Bayes is trained to classify different types of web vulnerabilities. The trained model is tested using the test dataset, and its performance is measured through evaluation metrics such as accuracy, precision, recall, and F1-score. A confusion matrix is generated to visualize the classification results and identify correctly and incorrectly predicted instances. The ensemble approach improves prediction accuracy by combining the strengths of multiple classifiers. Finally, the trained model is used to analyze new web queries uploaded by users and predict whether they represent normal traffic, SQL Injection, Cross-Site Scripting, or Remote File Inclusion attacks, providing an effective solution for intelligent web application security.

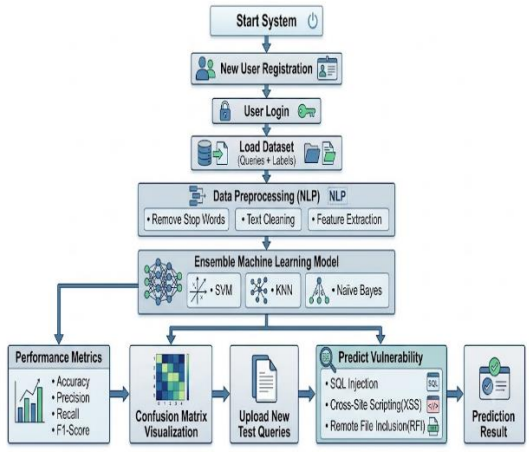

Figure [1]: Predictive Web Application Security Framework

Figure [1] presents the workflow of the proposed web application security framework. The system begins with user registration and login, followed by dataset upload and NLP-based preprocessing, including stop word removal, text cleaning, and feature extraction. The processed data is used to train an ensemble machine learning model. Model performance is evaluated using accuracy, precision, recall, F1-score, and a confusion matrix. Finally, the trained model predicts vulnerabilities such as SQL Injection (SQLi), Cross-Site Scripting (XSS), and Remote File Inclusion (RFI) from new test queries.

V. RESULT AND DISCUSSION

The experimental evaluation demonstrates that the proposed ensemble machine learning model provides reliable detection of web application vulnerabilities. After loading the dataset, Natural Language Processing techniques were applied to remove unnecessary words, clean the query text, and extract meaningful features for model training. The processed dataset was then divided into training and testing sets to ensure fair performance evaluation. The ensemble model, which combines Support Vector Machine, K-Nearest Neighbors, and Naïve Bayes classifiers, achieved an overall prediction accuracy of **95%** with high precision, recall, and F1-score values. The confusion matrix indicated that the majority of SQL Injection, Cross-Site Scripting, and Remote File Inclusion queries were correctly classified, while only a few instances were misidentified. The prediction module also successfully analyzed newly uploaded test queries and identified their corresponding vulnerability types. These results demonstrate that the proposed framework is capable of providing accurate, efficient, and consistent vulnerability detection, making it suitable for strengthening the security of modern web applications.

Figure [2]: Ensemble Classifier Performance

Figure [2] shows the performance of the proposed Ensemble Classifier after training on the vulnerability dataset. The model achieved an overall accuracy of 95.5% with high precision, recall, and F1-score values, indicating its effectiveness in detecting web application vulnerabilities. These results demonstrate that the ensemble learning approach provides reliable and accurate classification of malicious web queries, making it suitable for SQL Injection vulnerability detection.

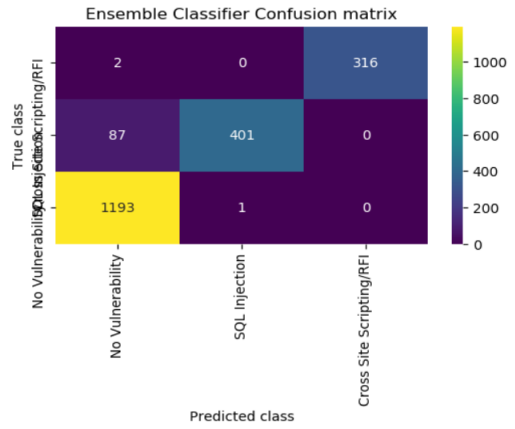

Figure [3]: Ensemble Classifier Confusion Matrix

Figure [3] presents the confusion matrix of the proposed Ensemble Classifier. The graph compares the actual and predicted vulnerability classes, including No Vulnerability, SQL Injection, and Cross-Site Scripting (XSS). Most predictions are correctly classified along the diagonal, while only a few instances are misclassified. This visualization confirms the strong classification performance of the proposed model in detecting web application vulnerabilities.

Figure [4]: Vulnerability Prediction Results

Figure [4] shows the prediction results generated by the proposed web application security system. The uploaded test queries are analyzed by the trained ensemble machine learning model, and each query is classified into its corresponding category, such as SQL Injection, Cross-Site Scripting (XSS), Remote File Inclusion (RFI), or No Vulnerability. This demonstrates the model's ability to accurately identify different types of web application vulnerabilities from new input data.

DISCUSSION

The results obtained from this study indicate that the proposed ensemble learning approach is effective for detecting different types of web application vulnerabilities. Combining multiple classifiers improves prediction performance by utilizing the strengths of each algorithm instead of relying on a single model. The preprocessing stage, including text cleaning, stop word removal, and feature extraction through Natural Language Processing, significantly enhanced the quality of the input data and contributed to better classification accuracy. The confusion matrix confirmed that only a limited number of queries were incorrectly classified, demonstrating the robustness of the developed model. In addition, the system accurately identified SQL Injection, Cross-Site Scripting, and Remote File Inclusion attacks from previously unseen test queries, proving its practical applicability. The user-friendly prediction module further simplifies vulnerability analysis for developers and security professionals. Overall, the proposed framework offers an intelligent and dependable solution for improving web application security while reducing the time and effort required for manual vulnerability detection.

VI. CONCLUSION

The proposed web application security framework demonstrates that machine learning can effectively detect different types of web application vulnerabilities, including SQL Injection, Cross-Site Scripting (XSS), and Remote File Inclusion (RFI). By combining Natural Language Processing techniques with an ensemble machine learning model, the system successfully analyzes web queries and classifies them with high prediction accuracy. The preprocessing stage, which includes text cleaning, stop word removal, and feature extraction, plays an important role in improving the quality of the training data and enhancing the overall performance of the model. Experimental results show that the ensemble approach provides reliable classification with strong accuracy, precision, recall, and F1-score values, making it suitable for practical web security applications. The developed prediction module also enables users to analyze new queries and identify potential vulnerabilities automatically. In the future, the framework can be enhanced by incorporating deep learning techniques, larger and more diverse datasets, real-time traffic monitoring, and explainable artificial intelligence methods. These improvements can further increase prediction accuracy, scalability, and the ability to detect newly emerging web application attacks in dynamic online environments.

REFERENCES

[1] Enron email dataset. https://www.cs.cmu.edu/~enron/. Accessed: 2017-07-01.

[2] National vulnerability database. https://nvd.nist.gov. Accessed: 2017-07-01.

[3] V. R. Basili, L. C. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. IEEE Transactions on software engineering, 22(10):751–761, 1996.

[4] D. Brumley, T.-c. Chiueh, R. Johnson, H. Lin, and D. Song. Rich: Automatically protecting against integer-based vulnerabilities. Department of Electrical and Computing Engineering, page 28, 2007.

[5] N. Dor, M. Rodeh, and M. Sagiv. Cssv: Towards a realistic tool for statically detecting all buffer overflows in c. In ACM Sigplan Notices, volume 38, pages 155–167. ACM, 2003.

[6] D. Evans and D. Larochelle. Improving security using extensible lightweight static analysis. IEEE software, 19(1):42–51, 2002.

[7] P. Godefroid, M. Y. Levin, and D. Molnar. Sage: whitebox fuzzing for security testing. Queue, 10(1):20, 2012.

[8] I. Haller, A. Slowinska, M. Neugschwandtner, and H. Bos. Dowsing for overflows: A guided fuzzer to find buffer boundary violations. In USENIX Security Symposium, pages 49–64, 2013.

[9] A. E. Hassan. Predicting faults using the complexity of code changes. In Proceedings of the 31st International Conference on Software Engineering, pages 78–88. IEEE Computer Society, 2009.

[10] S. Kim, T. Zimmermann, E. J. Whitehead Jr, and A. Zeller. Predicting faults from cached history. In Proceedings of the 29th international conference on Software Engineering, pages 489–498. IEEE Computer Society, 2007.

[11] D. Larochelle, D. Evans, et al. Statically detecting likely buffer overflow vulnerabilities. In USENIX Security Symposium, volume 32. Washington DC, 2001.

[12] P. Lathar, R. Shah, and K. Srinivasa. Stacy-static code analysis for enhanced vulnerability detection. Cogent Engineering, 4(1):1335470, 2017.

[13] R. Ma, Y. Yan, L. Wang, C. Hu, and J. Xue. Static buffer overflow detection for c/c++ source code based on abstract syntax tree. Journal of Residuals Science & Technology, 13(6), 2016.

[14] R. Moser, W. Pedrycz, and G. Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In Proceedings of the 30th international conference on Software engineering, pages 181–190. ACM, 2008.

[15] R. M. Pampapathi, B. G. Mirkin, and M. Levene. A suffix tree approach to antispam email filtering. Machine Learning, 65(1):309–338, 2006.

[16] E. Penttilä et al. Improving c++ software quality with static code analysis. N/A, 2014.

[17] H. Shacham. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS '07, pages 552–561, New York, NY, USA, 2007. ACM.

[18] Y. Shin and L. Williams. An empirical model to predict security vulnerabilities using code complexity metrics. In Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, pages 315–317. ACM, 2008.

[19] J. Viega, J.-T. Bloch, Y. Kohno, and G. McGraw. Its4: A static vulnerability scanner for c and c++ code. In Computer Security Applications, 2000. ACSAC'00. 16th Annual Conference, pages 257–267. IEEE, 2000.

[20] D. Wagner, J. S. Foster, E. A. Brewer, and A. Aiken. A first step towards automated detection of buffer overrun vulnerabilities. In NDSS, pages 2000–02, 2000.

[21] Shashank, A. (2025). Self-Healing Data Pipelines for Enhanced Reliability: A Paradigm Shift in Enterprise Data Management. Journal of Computer Science and Technology Studies, 7(8), 1097-1104.