Using Non-Standard Techniques to Approach the Detection of SQL Injection Attacks

Sydney Lipar (*Author*)
Texas A&M University
Department of Computer Science & Engineering
College Station, TX 77840
sydneylipar@tamu.edu

Shyam Ramachandran (*Author*)
Texas A&M University
Department of Computer Science & Engineering
College Station, TX 77840
Janshyam03@tamu.edu

***Abstract*—The recent growth of backend related information technology has led to the rise of various SQL injection attack tools, often causing problems in the field of network security. Methods to detect and predict SQL injection attacks with Machine Learning using feature extraction and language processing contain inefficiencies that we plan to address using our novel approach. It is important to address these issues effectively because attacks can exploit vulnerabilities and potentially give attackers unauthorized access to sensitive information. This paper, along with our included code and video, proposes the use of TF-IDF combined with n-grams to capture the unique components and sequential structure of SQL payloads. The experimental results show that our model has a higher accuracy, precision, recall rate, and F-1 score compared to other models trained on the dataset—validating our effectiveness in detecting malicious payloads.**

Keywords—SQL injection; intrusion detection; TF-IDF; text vectorization

I. Introduction

With how prevalent backend internet services have become, security problems are of increasing concern because of the vast number and complexity of attack payloads. Research conducted by OWASP has shown that SQL injection attacks rank 3rd in web application security risk, with over 94% of applications being tested for some form of injection [6]. These attacks can have severe consequences, including unauthorized access to sensitive data, such as personal information or financial records. Attackers can manipulate or delete critical data, often bypassing authentication mechanisms and leading to service outages.

Malicious users carry out SQL injection attacks by modifying their communications with database servers anytime they need to retrieve or store user data. SQL statements created by the attacker can be executed while the web server is fetching content from the application, and access systems in that way. Many defense methods and detection strategies have been researched in recent years, but constant change in attacker methods have led most of these strategies to have shortcomings. Automated tools like SQLMap can simulate attacks and scan for system vulnerabilities, but require extensive resources and information about the attacks.

The rise of Machine Learning has led to novel approaches in the field, with their primary advantage being their capability to detect new attacks. These techniques do have their drawbacks, with a possibility of increasing processing time and varying results depending on the algorithm used. The strategy we are proposing utilizes text-based vectorization and classification using ensemble learning to best recognize malicious SQL payloads and classify them effectively.

II. Related Works

Many detection methods have been proposed over recent years due to the increasing danger associated with SQL injections in network and backend payloads. Traditional approaches are normally carried out through methods like parsing, payload extraction, and others. Initial research in the field was centered along feature extraction methods, mainly due to the reasoning that SQL injection attack payloads differ from that of ordinary natural language. This proved ineffective because of problems with inaccurate feature segmentation and large payload vocabulary.

Work done in paper [2] looks to overcome some of these shortcomings, proposing a new “Feature Ratio

Method" for feature extraction. Performed on a dataset of over 20,000 URLs, the research aimed to compare their newly proposed methods with previous techniques. By plotting the average evaluation scores of "Method A" (The old method) and their new method on seven different ML classifying algorithms, they were able to determine an average F1 score of 96.29% for the algorithm, better than methods of previous studies in both accuracy and precision.

As research in the field advanced, new methods to improve the accuracy of detection models arose, using techniques like language processing and vectorization. One such method was introduced in research conducted by IIT Allahabad [4], using Natural Language Processing (NLP) and feature extraction to adapt to different SQL injection variants and optimize for false negative and false positive rates. For contextual learning, their model used methods known as masking and next-sentence prediction to predict words based on others in the context. Their proposed algorithm had F-1 scores as high as 97% in their large dataset.

To improve upon this, work done by Luo et al. through the Communications University of China [1] utilized a Convolutional Neural Network to classify their data, which can bypass some of the inefficiencies caused by evolving SQL injection techniques. CNNs use a convolutional layer system, assigning weights to each layer where filters allow the network to learn features by sharing these weights across the input. Through backpropagation, the CNN is better able to identify patterns in the SQL data, and make estimates based on its findings.

III. Methodology

To work around some of the issues brought up in the feature ratio [2] and NLP [4] papers, we proposed a SQL injection detection method that improves upon many of the commonly-used techniques. Our model utilizes TF-IDF to vectorize and prepare the data for classification via an RFC. Detailed below is the methodology and steps we used in creating our non-standard SQL injection detection pipeline.

A. Data Collection

For this research, our initial focus was on acquiring labeled data to enable supervised learning. Labeled data is crucial as it reduces ambiguity, enhances model evaluation, and allows for performance validation by comparing model predictions against true labels. The data used in this study was acquired via research conducted by Luo [1] and supplemented with additional datasets and techniques to ensure robustness.

It was also of the utmost importance that our labeled data was of the highest quality. High quality data is crucial to the success of our research as it directly impacts the accuracy, reliability, and generalizability of the models we created. To confirm the dataset created by Luo [1] was of sufficiently high quality, we analyzed its composition. Luo's dataset uses data gathered from widely recognized datasets in the field of information security, such as UNSW-NB15, KDD99, and HTTP CSIC 2010. Furthermore, the dataset includes labeled payload data extracted from public SQL injection cases, ensuring that it not only covers diverse scenarios but also provides accurate labels critical for supervised learning.

Table I. Experiment Dataset Structure

	Normal Payloads	**SQL Injection Payloads**
Data	35,574	36,472

As shown above in Table I. the dataset is composed of the following: 35,574 lines of normal payloads and 36,472 lines of injection payloads. To prepare the data for our machine learning algorithm, we manually added labels to the data. Payloads considered normal were labeled as zero, while SQL injection payloads—representing fraud—were labeled as one.

All of the data was then combined into a single dataset of 72,046 labeled lines and stored in a large Pandas DataFrame. This labeled dataset was subsequently shuffled and split into new training and testing sets, allowing us to implement k-fold cross-validation and ensure our model would perform well across different subsets of the data.

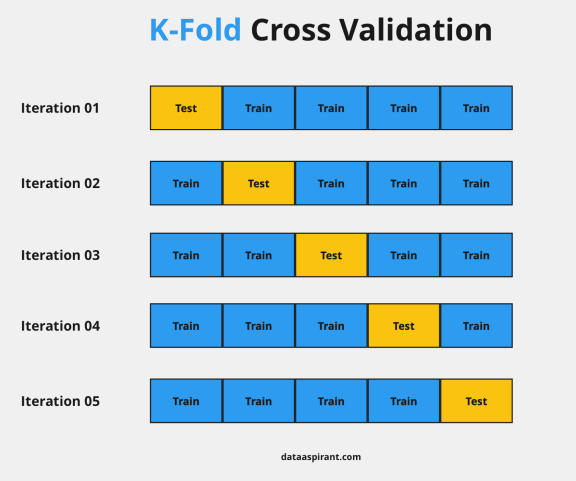
The large size of the dataset provided several advantages. With over 72,000 lines, the increased diversity within our data helped mitigate overfitting when training our model. In addition to benefiting the training of our model, the size of our data set also increased the model's ability to generalize to situations outside only our research. Finally, the size of our dataset is crucial to our implementation of machine learning models using both feature extraction and TF-IDF. For feature extraction, it allows the model to identify complex relationships between features within a single payload, while also identifying a larger variety of features across different payloads. For TF-IDF, a larger dataset improves our TF-IDF vectorizer's ability to

capture a wider vocabulary, while also reducing noise in the data–avoiding skewed term frequencies. Overall, the collected dataset served as a strong foundation for a robust analysis of various SQL injection detection methods.

B. K-Folds Cross Validation

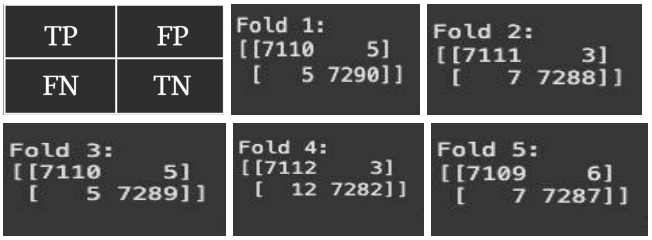
The dataset consists of a total of 72,046 lines, with each k-fold iteration involving approximately 14,000 test lines and 58,000 training lines. In our 5-fold cross-validation process, the data is split into five subsets, with each subset serving as the test set once while the remaining four subsets are used for training. This approach, both in theory and implementation, helps ensure that the model is evaluated on all portions of the data, providing a robust measure of its generalization performance. We successfully implemented 5-fold cross-validation on our dataset, confirming that the dataset is well-suited for developing our machine learning model.

Figure I. K-Fold Validation Visualization



Choosing a common k-value—like 5—provides a good balance between bias and variance in our performance estimate. With too little folds, the results gathered could not be conclusive of the dataset as a whole. On the contrary, with too many folds, each fold might become too small, leading to a high variance in the performance estimate. Even if the accuracy doesn't change significantly after a certain number of folds, a standardized process ensures that the final performance metric is reliable and indicative of our dataset as a whole.

Figure II. K-Fold Validation Matrices



When looking at the validation results (as shown in Figure II above), it can be seen that for each iteration, the number of false positives ranged between 3 and 6, while the number of false negatives varied from 5 to 12. Additionally, the standard deviation of the number of false positives detected between each iteration was 1.2. To account for the size of our dataset, we analyzed the rate at which false positives were detected. We divided 1.2 false positives per 14,410 test lines, giving the rate at which the machine learning algorithm detects false positives at 0.00008328. This extremely low rate of false positives highlights the model's accuracy and reliability, demonstrating its strong performance in distinguishing between normal and malicious payloads.

A. Feature Extraction with SVM

The first machine learning technique we used to classify our SQL injection payloads involved the implementation of feature extraction, a critical step in the process of transforming complex, high-dimensional data into a simpler and more relevant set of features. Feature extraction helps our model focus on important patterns and relationships within the data by selecting or constructing the most informative attributes while reducing unnecessary noise. This step is crucial because it not only simplifies the data but also improves the efficiency and accuracy of the model. In our implementation, this process allowed us to transform the raw input data into a set of key features, which were used to train our machine learning model for SQL injection detection.

Our code implementation enables feature extraction for detecting SQL injection attacks using a Support Vector Machine (SVM). An SVM is a classifier which works by finding the optimal boundary that separates the classes in the data. By maximizing the distance between support vectors (points on the boundary), the algorithm works to improve the model's ability to generalize to new, unseen data. In our implementation, the SVM is trained using a linear kernel, where the classifier finds a linear hyperplane that

best separates the classes, identifying patterns and relationships between features and their labels.

To prepare the data for the SVM, a feature extraction function was applied to the data, in which we extracted several key features: the length of each input, the count of query parameters, the number of suspicious characters (such as quotes and semicolons), the occurrence of common SQL keywords (like SELECT and UNION), and the presence of specific SQL injection patterns identified through regular expressions. The dataset was then split into training and testing sets using an 80-20 split, where we put 80% of the data into our training set and 20% of the data in our testing set, before training an SVM model with a linear kernel on the extracted features. After training, the model made predictions on the test set, and its performance was evaluated using a classification report, which provides precision, recall, and F1 scores. A confusion matrix was also displayed to show the counts of true positives, false positives, true negatives, and false negatives, providing further insights into the model's ability to correctly detect malicious payloads and avoid misclassifying normal inputs.

While the model showed promising results overall, this initial implementation showed signs of concern with respect to our false positive rate. The model was flagging more normal payloads as malicious than desired, which can be problematic in real-world applications where minimizing false alarms is critical.

B. Feature Extraction with RFC

To address this issue, we aimed to improve the precision of the model by switching to a Random Forest Classifier (RFC). The RFC was expected to enhance precision by better handling the complexities in the data and reducing the tendency to misclassify normal inputs as malicious. By leveraging the ensemble nature of RFC, we anticipated a reduction in false positives and a more balanced trade-off between precision and recall, ultimately leading to a more reliable model for SQL injection detection.

We continued our research by applying the same feature extraction process as in the previous SVM model, which involved analyzing metrics like input length, SQL keyword counts, etc. The data was then shuffled before being split into training and testing sets using an 80-20 split. Differing from our previous implementation, we implemented a Random Forest Classifier (RFC), an ensemble learning technique that builds multiple decision trees using bootstrapping and random feature selection. The process begins with bootstrapping, where multiple random samples are drawn, with replacement, from the original dataset. Each decision tree is then trained on one of these bootstrapped samples. After the data samples are selected, the next step is random feature selection: at each split in the tree, only a random subset of features is considered, rather than all features. This introduces diversity among the decision trees, preventing them from becoming too similar to one another. Finally, once all trees are trained, they each make predictions on the test data, and their results are combined via voting to determine the final prediction. This approach allows the Random Forest to leverage the collective decision-making power of all the trees. After training our RFC on the extracted features explained above, we evaluated its performance by predicting results on the test set and examining the results through both a classification report and confusion matrix.

C. Non Standard Technique: TF-IDF

Where traditional feature extraction was focused on quantifiable aspects of the input, such as length, counts of specific characters, or predefined SQL keywords, TF-IDF takes a more text-centric approach by analyzing the relative importance of terms in the payloads. TF-IDF assigns a weight to each term based on how frequently it appears in a specific payload (term frequency) relative to how often it appears across the entire dataset (inverse document frequency). This allows the model to highlight rare and potentially significant terms, such as specific SQL keywords or unusual patterns, while disregarding common or less relevant terms that may appear in both benign and malicious payloads.

We utilized the scikit-learn library, a widely used Python library for machine learning, to implement our method. The process began by vectorizing the data with a TF-IDF vectorizer, which converts text data into numerical representations. The vectorizer represents text by assigning weights to terms based on their importance within a document relative to their importance across the dataset. Then, using the fit_transform method, the TF-IDF vectorizer learns these weights and transforms the input text into a sparse matrix format containing the computed TF-IDF values for all terms. This transformed data was then split into training and testing sets, with 80% of the data used to train the Random Forest Classifier (RFC) and 20% reserved for evaluating the model's performance on unseen data. The RFC was trained on the TF-IDF features and their corresponding labels, enabling it to learn how to effectively distinguish between normal and malicious payloads. The RFC was chosen because of its improved performance over the SVM when using the data’s extracted features to create

our models. To enable effective comparisons between the RFC using feature extraction and the RFC using TF-IDF, we again computed both the appropriate classification report and confusion matrix.

D. Non Standard Technique: TF-IDF with n-grams

To improve upon solely using a TF-IDF vectorizer, we added n-grams to the vectorizer for the final implementation of our approach. While a TF-IDF approach alone captures the unique components of the SQL queries, an n-grams approach lets the model recognize context within the queries, preserving the order of terms and therefore the intent of the payload. By combining these two methods, we aimed to leverage the strengths of both approaches to achieve a more accurate and nuanced SQL injection detection.

Our approach utilized a TF-IDF vectorizer with n-grams to transform the tokenized queries in our dataset into a numerical format. This captures not only individual words (unigrams) but also consecutive word pairs (bigrams) and triplets (trigrams), which are particularly effective in detecting the contextual patterns often present in SQL injection attacks. By leveraging this technique, the vectorizer identifies multi-word sequences typical in malicious payloads but rare in benign queries, enabling the model to better differentiate between the two. For example, SQL injection patterns often include sequences of tokens, such as “SELECT *” or “UNION SELECT.” The resulting vectors were then fed into a Random Forest Classifier (RFC), which was trained to classify each query as either malicious or benign.

IV. Results

E. Evaluation Metrics

In this experiment, we evaluated our models using four key metrics: accuracy, precision, recall, and F1-score. Accuracy measures the proportion of all instances that were classified correctly, providing an overall sense of the model's performance. Precision answers the question: of all the instances flagged as SQL injections, how many were actually malicious? Recall evaluates the model's ability to detect SQL injections, calculating the proportion of actual malicious queries that were correctly identified. Finally, F1-score serves as the harmonic mean of precision and recall, balancing the trade-off between these two metrics to provide a single, comprehensive measure of the model’s effectiveness. Below are the definitions of the four metrics mathematically.

Figure III. Evaluation Metrics

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN} \quad (1)$$

$$Precision = \frac{TP}{TP+FP} \quad (2)$$

$$Recall = \frac{TP}{TP+FN} \quad (3)$$

$$F1 = \frac{2TP}{2TP+FP+FN} \quad (4)$$

F. Experimental Results

For our first two implementations of SQL injection detection, we focused on feature extraction but varied the classification models, comparing the performance of a Support Vector Machine (SVM) with a Random Forest Classifier (RFC). The SVM did not perform as well as the RFC, likely due to the dataset's lack of clear separable boundaries, which are essential for the SVM's hyperplane-based approach. In contrast, the RFC handled the dataset's complexity more effectively, leveraging its ensemble of decision trees to capture non-linear relationships in the features.

The results clearly demonstrated this improvement. While the SVM achieved an overall accuracy of 96%, the RFC achieved an overall accuracy of 99%. Additionally, the SVM performed with a precision of 0.94 and recall of 0.98 for normal (label 0) instances, and a precision of 0.97 and recall of 0.94 for malicious (label 1) instances. The RFC improved precision to 0.98 and recall to 1.0 for normal instances, while improving precision to 1.0 and recall to 0.98 for malicious instances.

Although the RFC demonstrated significant improvement over the SVM in handling feature extraction, the false positive rate remained a concern for both models. The SVM model incorrectly classified 178 instances as malicious, while the RFC reduced this number slightly to 172 false positives. Overall, the total number of misclassifications dropped from 614 with the SVM to 206 with the RFC. However, these results highlight that both models still flagged too many normal payloads as malicious, which poses a challenge in real-world applications where minimizing false alarms is critical.

By switching from feature extraction to TF-IDF, we were able to drastically reduce the total number of misclassifications. We brought the number of false positives and false negatives down from around 206 to

just five—demonstrating a significant improvement in model performance.

Overall, TF-IDF proved to deliver the highest accuracy among all approaches, with and without the addition of n-grams. Implementation C, which used TF-IDF alone, resulted in only 2 false positives out of 14,410 queries, while implementation D, which incorporated n-grams, produced 3 false positives—a difference that is not statistically significant. Our novel approach of using TF-IDF, as opposed to the more traditional feature extraction methods, achieved superior accuracy and precision. However, our prediction that adding n-grams would enhance performance was not supported, as the baseline TF-IDF implementation was already highly optimized, leaving little room for further improvement.

Table II. Results Summary

Model	**Results**			
	Accuracy	*Precision*	*Recall*	*F-1*
A	95.73%	97.46%	94.00%	95.70%
B	98.57%	99.52%	97.63%	98.57%
C	99.96%	99.97%	99.96%	99.97%
D	99.96%	99.96%	99.97%	99.97%

Iteration A: Feature Extraction with SVM
Iteration B: Feature Extraction with RFC
Iteration C: TF-IDF
Iteration D: TF-IDF with n-grams

G. Discussion

Our results from models A and B demonstrate a significant improvement in accuracy when transitioning from an SVM to an RFC with feature extraction, as accuracy increased from 95.73% to 98.57%. This aligns with findings in published research. Zhang [2], reported a similar trend using their feature ratio method to detect SQL injection attacks, where accuracy rose from 91.63% with an SVM to 99.64% with an RFC. These consistent results highlight the superior performance of an RFC in handling complex patterns and feature interactions.

Additionally, our implementation utilized n-grams of up to three, but it would be valuable to explore the performance impact of limiting the number of n-grams to only one or two. Research by Guo [3] suggests that both precision and recall can improve incrementally when adapting our model to account for various types of n-grams (n = 1, 2, or 3). While unigrams, bigrams, and trigrams have shown incremental improvements, it could also prove effective to test our model using higher values of n. However, their impact might be minimal, as the nature of SQL injection statements often relies on shorter, specific patterns rather than extended sequences of terms.

V. Conclusion

The outcome of our research demonstrates that using TF-IDF, both with and without n-grams, provides highly accurate results on our dataset, achieving accuracy and precision rates exceeding 99%. With only 5 misclassifications out of 14,410 URL instances, our approach proved effective in detecting SQL injection attacks. This is significant, as SQL injection detection for web URLs is critical in protecting against vulnerabilities that could compromise web application security, user data, and organizational integrity.

In the future, we would like to test our model's approach on different datasets to ensure our results are repeatable outside of our testing environment. We propose using datasets that are both larger and include more complex SQL statements, so we can better assess our model's generalizability and scalability. Additionally, it would be beneficial to analyze how we could feed the vectorized data from TF-IDF and n-grams into various different classifiers. We suggest exploring classifiers other than an RFC to potentially enhance detection capabilities and adapt to evolving attack patterns.

References

[1] A. Luo, W. Huang and W. Fan, "A CNN-based Approach to the Detection of SQL Injection Attacks," 2019 IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS), Beijing, China, 2019, pp. 320-324, doi: 10.1109/ICIS46139.2019.8940196. keywords: {Feature extraction; SQL injection; Payloads; Data models; Data mining; Training; Convolution; CNN; SQL Injection; ModSecurity},

[2] S. Zhang, Y. Li and Q. Jiang, "Feature Ratio Method: A Payload Feature Extraction and Detection Approach for SQL Injection Attacks," 2023 3rd Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS), Shenyang, China, 2023, pp. 172-175, doi: 10.1109/ACCTCS58815.2023.00019. keywords: {Vocabulary;Machine learning algorithms;Text categorization;Semantics;Symbols;SQL injection;Syntactics;feature ratio method;machine learning;SQLIA;neural networks},

[3] Z. Guo et al., "SQL Injection Detection Method Based on N-Gram and TFIDF," 2023 International Seminar on Computer Science and Engineering Technology (SCSET), New York, NY, USA, 2023, pp. 204-207, doi: 10.1109/SCSET58950.2023.00053. keywords: {Support vector machines;Seminars;Computer science;Heuristic algorithms;Intrusion detection;SQL injection;Network security;Network security;SQL injection;intrusion detection;N-Gram;text vectorization},

[4] S. Lakhani, A. Yadav and V. Singh, "Detecting SQL Injection Attack using Natural Language Processing," 2022 IEEE 9th Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON), Prayagraj, India, 2022, pp. 1-5, doi: 10.1109/UPCON56432.2022.9986458. keywords: {Structured Query Language;Codes;Filtering;Bit error rate;Machine learning;SQL injection;Feature extraction;Structured Query Language Injection;Web Application Security;Machine Learning;Deep Learning;Natural Language Processing},

[5] A. Arif, “Cross Validation In Machine Learning,” *Dataaspirant*, Dec. 03, 2020.

[6] OWASP, “A03 Injection - OWASP Top 10:2021,” *owasp.org*, 2021.

[7] Y. Fang, J. Peng, L. Liu, and C. Huang. Wovsqli: Detection of sql injection behaviors using word vector and lstm. ACM International Conference Proceeding Series (ICPS), pages 170–174, 2018.