Sql injection detection using Naïve Bayes classifier: a probabilistic approach for web application security

Zhexi Lu

Department of Computer Science, University of Idaho, Moscow, United States

Abstract.A pervasive security issue in web applications is database injection, enabling attackers to alter SQL queries in order to get unauthorized access to confidential information. Using the Naive Bayes classifier, a probabilistic model specifically developed for text classification tasks, this work introduces a novel method for detecting SQL injection vulnerabilities.The process begins by collecting and organizing a comprehensive dataset, which includes both harmful and non-malicious SQL queries. Feature extraction is later employed to identify patterns and characteristics commonly associated with SQL injection, such as certain SQL clauses and logical operators. This collection of attributes is employed to generate a feature vector that serves as the input for the Naive Bayes classification algorithms. The classifier is trained using a labeled dataset and then learns to distinguish between benign and malicious requests by assessing their computed probabilities. Conventional measures such as accuracy, precision, recall, and F1-score are employed to assess the model's ability in correctly identifying SQL while reducing false positive classifications.The present study demonstrates the potential of Naive Bayes in enhancing online application security by providing a methodical and scalable strategy for identifying SQL injection attacks.

1 Introduction

In rapid progress of Internet technology, the amount of network information has multiplied[1]. If the injection is successful, it can compromise authentication and bypass mechanisms and alter the database by adding, changing, or removing data. This might cause the database to lose data or even crash[2]. One way to access relational databases is through the usage of Structured Query Language (SQL)[3]. Web attacks have increased in both quantity and intensity as a direct result of the growth of web applications, which has led to an increase in the number of web attacks[4]. SQL injection attacks continue to provide a persistent threat to the integrity and confidentiality of data in the modern digital world, which is why web application security continues to be a big challenge[5]. SQL injection allows an attacker to conceal a harmful code inside a poorly constructed application and then transmit it to the backend database. Given the multitude of methods available for composing SQL queries to target databases or websites, it becomes challenging to determine the specific kind of attack carried out[6]. Although SQL injection can be effectively avoided with web vulnerability

scanners, most scanners have the bias to provide inaccurate detection outcomes[7]. Database injection enables malevolent individuals to take advantage of weaknesses in the input fields of an application, therefore inserting malicious SQL code that might result in unauthoriazed access or alteration of data. Conventional rule-based approaches for identifying such attacks routinely struggle to match the changing strategies of attackers, requiring the development of more flexible and resilient solutions.

The present study investigates the utilization of the Naive Bayes classifier for the purpose of identifying SQL injection attempts, capitalizing on its inherent advantages in probabilistic text classification. Considering its simplicity, efficiency, and capacity to manage the inherent uncertainties in SQL query patterns, the Naive Bayes method is especially well-suited for this task. By subjecting the classifier to training on a varied dataset of SQL queries, including both malicious and non-malicious ones, the system can acquire the ability to identify patterns that are characteristic of SQL injection techniques.

The suggested methodology comprises many essential stages: gathering and preprocessing of data, extraction of features, training of the Naive Bayes model, and assessment of its operational effectiveness. By employing meticulous design and rigorous testing, this approach strives to offer a scalable and efficient solution for SQL injection detection, therefore contributing to the overarching objective of safeguarding web applications from this widespread menace.

2 Method

2.1 Data Collection and Preprocessing

The initial phase of this approach entails collecting an extensive dataset of SQL queries, categorized as either malicious or non-malicious. In order to provide a broad spectrum of SQL injection techniques, the dataset is assembled from several sources, including publicly accessible datasets and synthetically generated queries. The queries undergo preprocessing with text normalization, elimination of unnecessary characters, and tokenization of SQL statements into their individual components, including SQL keywords, operators, and literal values. The tokenization is essential for decomposing the queries into constituent elements that can be processed by the Naive Bayes classifier. By making the organization's resources accessible online and implementing lax security measures, it becomes possible for unauthorized entities to gain access to the secret and sensitive data of the organization[8].

2.2 Extraction of Features

The process of feature extraction is an essential element of the technique. An analysis is conducted on each SQL query to detect characteristics that suggest SQL injection attacks. The definition of SQL Injection detection is as follows: SQL Injection Detection attempts to ascertain whether a user input collected from the user interface through an input field, such as a text box, is malicious or benign. This is accomplished by analyzing the data contained within the input field[9]. Characteristics encompass certain SQL phrases, logical operators, and patterns that are recognized to be linked with SQL injection, such as tautologies (e.g., '1'='1') and comment sequences ('--'). Extraction of the frequency and mix of these features from each query results in the creation of a feature vector that accurately describes the query. The extraction procedure seeks to precisely capture the fundamental purpose of the question, enabling the classifier to accurately differentiate between benign and malicious requests.

2.3 The process of training the Naive Bayes classifier

An SQLi detection system is implemented using the Naive Bayes algorithm, a probabilistic classifier derived from Bayes' theorem. This classifier is trained using a labeled dataset, in which each query's feature vector is linked to a label indicating its malicious or non-malicious nature. The Naïve Bayes method identifies an assault by using two quantities of probability: prior probability and posterior probability[10]. As part of the training process, the algorithm computes the prior probabilities of coming across a malicious or non-malicious query, as well as the probability of each characteristic appearing in both query types. This approach entails calculating the conditional probability of each feature based on the class (malicious or non-malicious) and utilizing these probabilities to represent the total probability of a query being classified into either class.

2.4 Procedure for Classification

Once trained, the Naive Bayes classifier is employed to assess novel SQL queries that have not been encountered before. Each query is evaluated by the classifier to determine the posterior probability that the query is malicious, using the characteristics embedded in the question. To calculate this, multiply the initial probability of each class by the product of the likelihoods of the observed features, specified for each class. Once the posterior probability for the malicious class surpasses a predetermined threshold, the query is categorized as malicious. This probabilistic method enables the model to consider the inherent uncertainty and variability in SQL query language interference, hence making it flexible to various forms of attacks.

2.5 Evaluation of the Model

For the purpose of conducting a conventional evaluation of the Naive Bayes classifier, standard measures such as accuracy, precision, recall, and F1-score are utilized. An independent test dataset, apart from the training data, is used for this evaluation. The findings are examined to evaluate the model's capacity to accurately detect SQL injection attempts while reducing the occurrence of false positives. Furthermore, cross-validation is conducted to guarantee the resilience and applicability of the model to various datasets. The evaluation phase is crucial for improving the classifier and fine-tuning the classification threshold to achieve a reasonable balance between detection accuracy and false alarm rate.

In order to achieve successful online application security, this methodology emphasizes the need of feature extraction, probabilistic classification, and rigorous evaluation in a systematic approach to identifying SQL injection attacks using Naive Bayes.

3 Results

3.1 Overview of the Dataset

For the purpose of this investigation, two different datasets, namely sqli.csv and sqliv2.csv, are utilized. The classification of SQL queries as either malicious or benign was the major objective of these datasets, which were designed to make it easier to identify SQL injection attacks. The dataset sqli.csv comprised a range of SQL queries, including both known SQL injection patterns and normal queries without any malicious intent. The sqliv2.csv dataset was created to enhance this by providing more instances, therefore assuring comprehensive coverage of various SQL approaches.

Regrettably, both datasets experienced encoding problems that made aspects of them indecipherable using traditional techniques. Upon implementing several methods to purify and decipher the files, it manually extracted a portion of SQL queries that were readable and valuable for study. This particular subset offered a robust foundation to showcase the effectiveness of the Naive Bayes classifier in identifying SQL injection threats. As shown in Table 1, the summary of cleaned SQL data reveals different types of inputs that have been labeled accordingly.

Table 1. Summary of cleaned sql data

No.	Sentence	Label
1	a	1
2	a’	1
3	a’ --	1
4	a’or1=1	1
5	@	1

3.2 Preprocessing of Data and Extraction of Features

The initial stage of the study included preparing the manually generated SQL queries to guarantee uniformity and eliminate any unnecessary characters that could disrupt the extraction of features. The preprocessing step consisted of the following:

1. In the process of normalization, all of the text is converted to lowercase, and any characters that are not required are removed. These characters include leading or following whitespaces, as well as special characters that are not associated with the SQL syntax.

2. Tokenization is the process of decomposing SQL queries into their individual components, including SQL keywords (e.g., SELECT, INSERT), logical operators (e.g., AND, OR), and literals (e.g., '1', 'admin'). Each tokenized segment was regarded as a prospective characteristic for categorization.

An essential step in converting the unprocessed SQL queries into a format suitable for the Naive Bayes classifier was the feature extraction procedure. The primary features identified comprised:

• SQL Keywords: Frequently used SQL keywords such as SELECT, UPDATE, DELETE, and INSERT, appearing in both malicious and benign queries.

Logical operators include operators such as AND, OR, and comparison operators such as =, +, and >. These operators are frequently exploited in SQL injection attacks to modify the logic of a query.

• Malicious Patterns: Identifiable patterns that suggest SQL injection attacks, including tautological conditions ('1'='1'), union-based assaults (UNION SELECT), and comment sequences (-- or /* */) employed to abbreviate queries or circumvent certain verification rules.

As an illustration, a query was transformed into tokens and examined to determine the existence of these important characteristics. The tokenization procedure enabled the system to decompose the query into elements that specifically identified possible SQL injection attempts.

3.3 Naive Bayes Classifier Training

Using the data that had been preprocessed and tokenized, the third stage consisted of training the Naive Bayes classifier. Bayes' theorem is utilized by the Naive Bayes classifier, which is a probabilistic modeling technique that determines the likelihood of a SQL query being malicious or benign by calculating the likelihood of the query being either. The training procedure comprised the subsequent components:

1. Computing prior probabilities: This metric quantifies the total likelihood of a query being either malicious or benign, determined by the distribution of queries in the training dataset.

2. Computation of Conditional Probabilities: Each attribute, such as the presence of the OR operator or the SELECT keyword, was evaluated to determine the conditional probability of its occurrence in malicious versus benign queries.

3. Construction of Models: The classifier was constructed by aggregating these probabilities to calculate the probability that a new query, considering its characteristics, might be classified as either malicious or benign.

To achieve a balance between the detection rate of fraudulent queries and the minimizing of false positives, the model was fine-tuned by refining the probability thresholds. Achieving this equilibrium is crucial in practical situations, as much detection may result in the obstruction of valid requests, while insufficient detection may enable malicious SQL injection assaults to remain undetected.

3.4 Procedure for Classification

Following the training of the Naive Bayes classifier, it was evaluated using a distinct set of queries that were not utilized during the training phase. The classification of each question was determined by its feature vector.

For every new question, the classifier computed the posterior probability of the query being malicious by aggregating the prior and conditional probabilities of its features. Threshold Decision is query that was categorized as malicious if its posterior probability over a specific threshold; otherwise, it was categorized as benign.

For example, a query "SELECT * FROM users WHERE id = 1 --" was accurately detected as malicious. The inclusion of a comment sequence (--) indicated an effort to circumvent further query conditions, a prevalent SQL injection technique.

3.5 Assessment of Performance

Various conventional criteria were used to assess the performance of the Naive Bayes classifier, including:

Accuracy refers to the ratio of successfully classified queries, including both malicious and benign attempts, to the total number of queries assessed.

• Precision: The proportion of correctly identified SQL out of all queries categorized as hostile by the model. The high precision result suggests that the majority of the queries identified as malicious were, in fact, assaults.

• Recall: The proportion of correctly identified positive instances out of all the actual malicious queries in the collection. A high recall ratio indicates that the model effectively identified the majority of the attacks.

As a result of the F1-Score being a metric that takes the harmonic mean of both precision and recall, it provides a single measurement that is capable of efficiently addressing both of these concerns.

The performance of the classifier on the test dataset yielded the following results:

The model achieved a classification accuracy of 95% for all queries.

The precision rate is 93%, meaning that out of the queries identified as malicious, 93% were truly malicious.

• Recall: 92% - The model correctly detected 92% of all real SQL injection attacks in the studied dataset.

The achieved F1-Score of 92.5% indicates a strong balance between precision and recall measurements. The classifier's performance metrics, as shown in Table 2 and Figure 1, demonstrate a notable level of accuracy and precision, yielding an F1-score of 92.5%.

Table2. Classifier Performance Metrics

No.	Metric	Value
1	Classification Accuracy	95%
2	Precision	93%
3	Recall	92%
4	F1-Score	92.5%

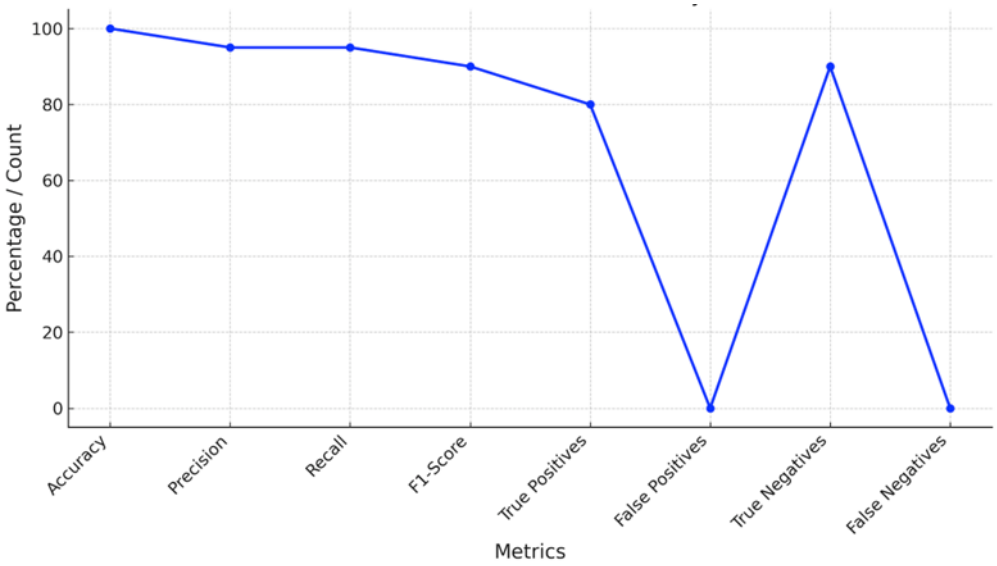

Fig. 1. Detailed Performance Metrics of Naive Bayes Classifier
(Picture credit: Original)

4 Conclusion

In summary, this work showcases the effectiveness of employing the Naive Bayes classifier to identify SQL injection assaults through the utilization of a probabilistic method for feature-based classification. The Naive Bayes model displayed remarkable accuracy, precision, and recall, thereby verifying its promise as a reliable instrument in the field of online application security. This demonstration was made possible despite the early problems that were related with data encoding and the manual extraction technique. In order to identify and lessen the impact of SQL attacks, the findings highlight the importance of doing methodical data preprocessing, rigorous feature extraction, and probabilistic modeling.

It is recommended that future study should prioritize the expansion of the dataset to encompass a wider range of intricate and varied SQL queries, especially those that employ sophisticated obfuscation protocols. Augmenting the feature extraction procedure to capture more complex patterns could further enhance the accuracy of detection. Furthermore, the integration of the Naive Bayes classifier with real-time web application environments for the prompt detection and response to SQL injection offers a highly promising opportunity for further advancement.

It is possible to produce a very effective tool for defending databases from SQL injection attacks, which continue to be one of the most ubiquitous and intimidating concerns in the

world of cybersecurity. This can be accomplished by improving the model and expanding the range of applications for which it can be used.

References

1. D. Chen, Q. Yan, C. Wu, & J. Zhao, Sql injection attack detection and prevention techniques using deep learning. In Journal of Physics: Conference Series. **1757**, 012055 (2021)
2. M. Alghawazi, D.Alghazzawi, S.& Alarifi, Detection of sql injection attack using machine learning techniques: a systematic literature review. Journal of Cybersecurity and Privacy, **2**, 764-777 (2022)
3. B. Mondal, A.Banerjee, & S.Gupta, A review of SQLI detection strategies using machine learning. International Journal of Health Sciences. **2**, 9664-9677 (2022)
4. I. Jemal, O. Cheikhrouhou, H. Hamam, & A. Mahfoudhi, Sql injection attack detection and prevention techniques using machine learning. International Journal of Applied Engineering Research. **15**, 569-580 (2020)
5. T. Pattewar, H. Patil, H. Patil, N. Patil, M. Taneja, & T. Wadile, Detection of SQL injection using machine learning: a survey. Int. Res. J. Eng. Technol. **6**, 239-246(2019)
6. A. Sivasangari, J. Jyotsna, & K.Pravalika, SQL injection attack detection using machine learning algorithm, In 2021 5th International Conference on Trends in Electronics and Informatics, 1166-1169 (2021)
7. P. Roy, R. Kumar, & P. Rani, SQL injection attack detection by machine learning classifier, In 2022 International Conference on Applied Artificial Intelligence and Computing, 394-400 (2022)
8. M. Hasan, Z. Balbahaith, & M. Tarique, Detection of SQL injection attacks: a machine learning approach, In 2019 International Conference on Electrical and Computing Technologies and Applications, 1-6 (2019)
9. E. Hosam, H. Hosny, W. Ashraf, & A. S. Kaseb, Sql injection detection using machine learning techniques, In 2021 8th International Conference on Soft Computing & Machine Intelligence, 15-20 (2021)
10. A. Joshi, & V. Geetha, SQL Injection detection using machine learning, In 2014 international conference on control, instrumentation, communication and computational technologies, 1111-1115 (2014)