International Journal on Science and Technology (IJSAT)
E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

SQL Injection Attack Detection Using Logistic Regression and TF-IDF Vectorization

Mrs.Priyanka Pandarinath[1], Kothakapu Harini[2], Nallamasa Dilip[3], Mothikar Aditya[4]

[1]Assistant Professor, [2,3,4]Scholar Department of Computer Science and Engineering,
Nalla Narasimha Reddy Education Society's Group of Institutions, Hyderabad, India

ABSTRACT

SQL injection attacks pose a serious security risk to online applications because they provide hackers access to sensitive data and the ability to manipulate databases using malicious SQL commands. This project uses TF-IDF vectorization and logistic regression to detect SQL injection attacks using a machine learning method. To train the model, a dataset of legitimate and malicious SQL instructions is generated and preprocessed. An intuitive user interface for the realtime detection of SQL injection attempts is provided by the integration of the trained model into a Flask web application. Users can enter attempts at SQL injection into the program. Users can enter SQL instructions into the application to get immediate feedback on whether the command is malicious or genuine. By successfully identifying and mitigating possible SQL injection risks through machine learning, this technology improves the security posture of web applications.

KEYWORDS--- Cybersecurity, Flask, Web Application Security, Machine Learning, Logistic Regression, TF-IDF Vectorization, SQL Injection, Real-Time Detection, Data Preprocessing, and ModelTrainCybersecurity, Flask, Web Application Security, Machine Learning, Logistic Regression, TF-IDF Vectorization, SQL Injection, Real- Time Detection, Data Preprocessing, and ModelTrain

1. INTRODUCTION

SQL injection (SQLi) is one of the most common and deadly vulnerabilities that affects web applications. It is becoming more and more susceptible to many kinds of security threats. Attackers can corrupt or gain unauthorized access to sensitive data by manipulating databases using inadequately sanitized SQL queries.

Through the act of "injecting" or "plugging" unauthorized SQL code into inadequately sanitized online inputs, adversaries can get around authentication protocols, access confidential data, alter data, or even take full control of the database. When user data is concatenated straight into SQL queries without sufficient input validation, input fields like login forms, search boxes, or URL parameters are often the target of SQL injection attacks.

Conventional safeguards like parameterized queries and input validation can work, but sophisticated and developing SQLi techniques frequently get around them. A more resilient solution is needed as web applications become more complicated. The ability of traditional rule-based methods to identify complex SQL injection patterns is frequently restricted. In this research, a trained logistic regression model is used to present a machine learning method for identifying SQL injection assaults.

2. RELATED RESEARCH

Current Methods for Preventing and Identifying SQL Injection:

Web application security has long been threatened by SQL Injection (SQLi). Numerous methods have been devised to lessen and identify these assaults. Several of the most popular techniques are as follows:

Input validation: Makes ensuring that any information that users submit—through cookies, URL parameters, or forms— is accurate, safe, and clean. Malicious SQL instructions can be screened out by imposing stringent guidelines on permitted input types. But this approach alone is frequently inadequate since it can overlook intricate insertion efforts.

Prepared Statements with Parameterized Queries: This method inserts placeholders into SQL queries rather than incorporating user input directly. By ensuring that user inputs are handled as data rather than executable code, parameterized queries guard against malicious SQL commands being injected by attackers. Despite being incredibly successful, this method must be applied consistently throughout the application.

Stored Procedures: SQL commands that have been precompiled and are kept in the database as stored procedures. They increase security by isolating user input from SQL logic. Even storing stored procedures lessen the possibility of injection, they can still be dangerous if parameterization isn't used to adequately secure them.

Tools for Static Analysis: These programs examine an application's source code to find possible security holes, such as SQL injection. Risky coding techniques, including concatenating user input straight into SQL queries, can be found via static analysis. Static analysis tools, however, have the potential to produce false positives and may need manual inspection.

SQL injection attacks are always evolving, so even with these robust safeguards, skilled attackers might still be able to get past them. This has prompted research into increasingly sophisticated strategies, such as those based on machine learning.

Methods of Machine Learning for SQL Injection Detection:

Machine learning (ML) has become a viable method for identifying SQL injection threats in recent years. Machine learning models can detect anomalous activity that could be a sign of an attack by examining patterns in user behavior and SQL query patterns. In this field, several machine learning algorithms have been used:

Support Vector Machines (SVM): By identifying input queries as either legitimate or malicious, SVM, a supervised learning method, has been used to detect SQL injections.

SVM is good in class distinction, but because of its computational complexity, it may have trouble with huge datasets or intricate injection patterns.

Decision Trees: Decision trees use a set of branching rules and conditions to categorize queries. They work well with structured datasets and are comparatively simple to interpret. Decision trees may, however, overfit the training set, which would reduce their ability to generalize to novel attacks.

Neural Networks: Because neural networks can recognize intricate patterns in data, they have demonstrated promise in identifying SQL injection. This is especially true of deep learning models. Neural networks can be computationally costly to train and implement, and they require a lot of labeled data to be trained.

When compared to conventional methods, machine learning techniques provide the following benefits:

Adaptive Learning: By retraining on updated datasets, machine learning models can gradually adjust to novel attack patterns.
Real-Time Detection: Machine learning algorithms are able to evaluate queries instantly, identifying potentially harmful activity as it happens.
Pattern Recognition: ML models, in contrast to static rules, are able to identify small, imperceptible patterns in SQL queries that may point to an attack.

Nevertheless, machine learning for SQL injection detection is not without its difficulties:

Data Availability: Large datasets containing both legitimate and malicious searches are necessary to build effective machine learning models, but they can be hard to come by. **False Positives:** ML models have the potential to generate false positives, which would mark valid queries as malicious, particularly in their early stages. **Model Complexity:** Security teams may find it more challenging to comprehend the reasons for a query's malicious flagging when dealing with algorithms that are difficult to grasp, such as neural networks. In order to detect SQL injection attacks, we present a machine learning-based method in this research that uses TF-IDF vectorization and logistic regression to provide a compromise between simplicity and efficacy.

3. Methodology Construction of Datasets

We created a dataset for this investigation that included both
legitimate and nefarious SQL queries. There are two categories within the dataset:

Common SQL commands: These were either manually crafted or taken from reliable, safe online resources. Every query is benign and adheres to typical SQL procedures.

Malicious SQL Commands: Union-based, boolean-based, error-based, and time-based injections are just a few of the popular SQL injection attack methods that were used to create these queries. The purpose of these instructions is to take advantage of holes in databases and applications.
The collection is made up of Y malicious queries (labeled as '1) and X normal searches (labeled as '0'). The machine learning model is trained using this labeled data in order to differentiate between instructions that are safe and those that are dangerous.

Preprocessing Text and Extracting Features

We utilized TF-IDF (Term Frequency-Inverse Document Frequency) vectorization to convert the SQL queries into a format suitable for machine learning. By quantifying a word's significance within a document (or query) in relation to the total dataset, TF-IDF generates numerical depictions of the text.

What makes TF-IDF?

TF-IDF was used because it penalizes popular terms that might not be important (such as SQL keywords like "SELECT" and "FROM") while providing a fair representation of word frequency. This means that, in contrast to straightforward methods like one-hot encoding, which might treat all words identically, the model might concentrate on phrases or patterns that are more likely to suggest malevolent conduct.
The process of feature extraction involves tokenizing each query and converting each word into a TF-IDF score.
Text is converted into numerical features using the TF-IDF (Term Frequency-Inverse Document

Frequency) vectorization, which is based on how frequently phrases occur in a document in comparison to a set of documents.

Below is an explanation of the formulas in question:

1. Frequency of Term (TF):

The frequency with which a term occurs in a document is measured by term frequency. A term's value increases with frequency of occurrence, however it is normalized to prevent bias toward longer documents.

TF (t,d) = The number of times the phrase t appears in the document d/Total terms in the document d Where:

t is a term.
and d is document.

2. Inverse Document Frequency(IDF):

A term's importance is determined by Inverse Document Frequency, which assesses how uncommon it is in all texts. A term's IDF score increases with its uniqueness.

IDF (t,D) = log (N/ df (t)) Where:
The corpus's total number of documents is N.
df(t) is the number of documents containing the term t.

3. Calculation of TF-IDF:

Each term's TF-IDF score is determined by multiplying its inverse document frequency (IDF) by its term frequency (TF):
TF-IDF(t,d,D)=TF(t,d)*IDF(t,D) Where
T is term.
D is collection of all documents. D is a single document.

Example The breakdown Assume:

In the 100-word document, document d, the term "t" appears three times.
Ten out of the 1000 papers in the corpus include the term "t"

1. TF Calculation:

TF (t, d) = 3/100 = 0.03

2. IDF Calculation:

IDF (t, D) = log (1000/1+10)
= log (1000/11) = 2.199(approx.)

3. TF-IDF CALCULATION:

TF-IDF (t, d, D) = 0.03*2.199 = 0.06597.
This indicates the phrase t's ultimate weight in document d. The phrase is more relevant for that text the higher its TFIDF score.

Dataset: A set of SQL queries for SQL injection detection that have been classified as malicious (1) or normal (0). **Prior to processing:**

Tokenization is the process of dividing SQL queries into discrete tokens.
Vectorization: TF-IDF vectorization is used to translate SQL queries into numerical model characteristics.

2. Layer of Machine Learning Models

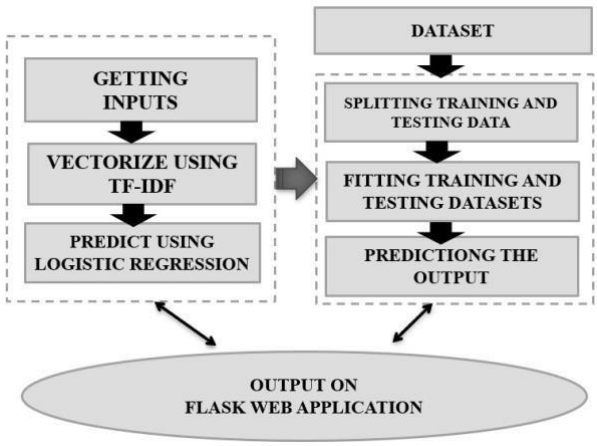
The fundamental logic for utilizing the taught machine learning model to identify SQL injection threats is contained in this layer.

Model Selection: In this instance, classification is accomplished by logistic regression.

Using the TF-IDF vectorized SQL query, it makes a prediction as to whether the query is an injection attack or not.
Instruction Procedure:

Preprocessed SQL queries and labels (0 for benign, 1 for malevolent) are the input.
A trained logistic regression model with the ability to determine if a query is an injection or not is the output**.** A trained logistic regression model with the ability to determine if a query is an injection or not is the output. **Metrics of Performance:**
The model's accuracy is the frequency of true predictions. Precision: The percentage of positive predictions that came true.
Recall that The percentage of real positives that the model detected.
The harmonic mean of recall and precision is the F1-Score.

3.ARCHITECTURE



We will use a common Machine Learning web application structure to design the architecture for your SQL injection detection project, which consists of a Flask web application, a trained model, and the ability to detect SQL injection assaults.

1. Layer of Data Collection
This layer is in charge of compiling and getting ready the dataset, which contains SQL injection attacks as well as standard queries.

3. **Model Persistence Layer Pickle:** Pickle is used to serialize the TF-IDF vectorizer and trained model so that the web application can load them later.
model.pkl is the model file. tfidf_vectorizer.pkl is the vectorizer file.

4. Flask, the Web Application Layer
The machine learning model and users are interfaced with by this layer.

A simple web framework called Flask is used to communicate with the model. The essential elements consist of:
A webpage where users can submit SQL queries to test for injection is called an input form.
API Destinations:
POST /predict: This function takes user-supplied SQL queries via an API or web form, and produces a prediction (either regular or SQL injection).

Prediction Process: Takes in the user's input query. uses the previously saved TF-IDF vectorizer to preprocess the query. Calculates the likelihood that a query is an injection or normal using the trained Logistic Regression model.

5. Interface Layer
This is how your program communicates with users on the front end.

4. EVALUATION

The assessment of the trained model's effectiveness in identifying SQL injection attacks is a component of the SQL injection detection project evaluation. Here's how to use common classification metrics and validation procedures to assess the performance of Logistic Regression model:

Metrics for Evaluation
Common metrics you might use for binary classification model (regular query vs. SQL injection) are as follows:

Precision:
The overall accuracy of the model's predictions is measured by accuracy. It is determined by dividing the total number of forecasts by the ratio of accurate predictions (including genuine positives and true negatives).

ACCURACY = (True Positives + True Negatives)/Total samples.

Precision:
Precision indicates the proportion of true positive predictions (SQL injection) made by the model. When false positives are expensive, it is helpful.

PRECISION = True Positives/True Positives + False Positives.

RECALL:

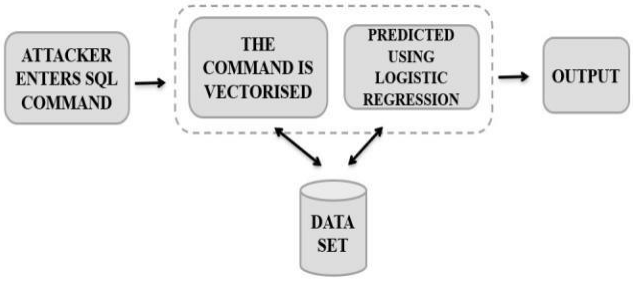
The model's recall quantifies how well it recognizes real positive cases, or SQL injections. Since it is expensive to overlook positive situations, it is crucial.

RECALL = True Positives/True Positives + False Positives.

F1-Score:

The F1-Score, which is particularly helpful when there is an imbalance between classes, is the harmonic mean of precision and recall. It offers a balanced measure between the two.

F1-Score = 2*(Precision*Recall)/(Precision + Recall).



5. RESULT

The project's goal was to use a dataset of both malicious and normal SQL queries to develop a machine learning-based system for identifying SQL injection attacks. Logistic regression was used to train and test the model, and the accuracy, precision, recall, and F1-score are popular classification metrics used to assess the model's performance.
The following are some of the project's major findings:
The model's high overall accuracy suggests that, for the most part, it can distinguish between benign and malevolent SQL queries.
The precision of a model in identifying SQL injection attacks refers to its ability to distinguish between genuine attacks and false positives, which are simply routine queries that have been mistakenly marked.

All things considered, the methodology has a lot of potential to improve database security through effective malicious SQL query detection. Still, there are a few areas that may use improvement, like lowering the false positive rate—the

percentage of normal queries that are mistakenly categorized as attacks—by adjusting the model or adding more sophisticated methods like ensemble learning, neural networks, or anomaly detection

systems.

A scalable, automated solution to SQL injection detection is provided by this machine learning technique, which may be tailored to a variety of applications needing strong database security.

Recall measures how well the model detects real SQL injection attempts, notwithstanding the occasional false positive. Given the potentially significant cost of a missed attack (false negative), excellent recall is necessary to guarantee that no attack is overlooked. The model's overall efficacy in balancing the trade-off between accuracy and memory is shown by the F1-score, which is a harmonic mean of precision and recall.

6. CONCLUSION

The project aims to tackle the crucial problem of SQL injection attacks, which are a common and potentially hazardous security hazard in web applications. The study successfully illustrates how to recognize and mitigate SQL injection issues by utilizing sophisticated detection techniques and preventive measures. The creation of a reliable SQL injection detection system that makes use of complex text-processing methods like TF-IDF and machine learning algorithms like Logistic Regression are important results. These techniques have been included into a web application to instantly detect and notify users of any SQL injection risks. The project demonstrates the significance of protecting web applications from SQL injection attacks and guarantees that the system in place can correctly identify and handle such threats, protecting user security and data integrity.

REFERENCES

1. Shehu Magawata Shagari, shagari1978@gmail.com, Department of Computer Science, Kebbi State University of Science and Technology, Aliero, Nigeria.
2. Danlami Gabi, Department of Computer Science, Kebbi State University of Science and Technology, Aliero, Nigeria.
3. Nasiru Muhammad Dankolo, Department of Computer Science, Kebbi State University of Scienc and Technology, Aliero Nigeria.
4. Noah Ndakotsu Gana, Department of Cyber Security Science, Federal University of Technology, Minna, Nigeria. [5]Z. Chen & M.Guo, “Research on SQL injection detection technology based on SVM”, Internatioal Conference on Smart Materials, Intelligent Manufacturing and Automation(2018)
5. [6] S. O. Uwagbole, W. J. Buchanan & L. Fan, “Applied
6. Machine learning predictive analytics to SQL injection attack detection and prevention”, IFIP/IEEE Symposium on Integrated Network an Service Management (IM) (2017) 1087.
7. [7]R. Chandradhekhar, M. Mardithaya, S. Thilagam & D. Saha, “SQL injection attack mechanisms and prevention techniques”, International Conference and Advanced Computing, Networking and Security(2011) 524
8. A. Dasgupta, V. Narasayya & M. Syamala, “A static analysis framework for database applications”, IEEE 25th International Conference on Data Engineering (2009) 1403.
9. C. S. Kumar, J. Seetha, S. R. Vinotha, “Security implications of distributed database management system models”, International Journal of Soft Computing and Software Engineering 2 (2012) 20.
10. C. Anley. “Advanced SQL injection in SQL server applications,” https://crypto.stanford.edu/cs155old/cs155 spring09/papers/sql injection.pdf. Accessed 14 December,

2021.