1 Introduction

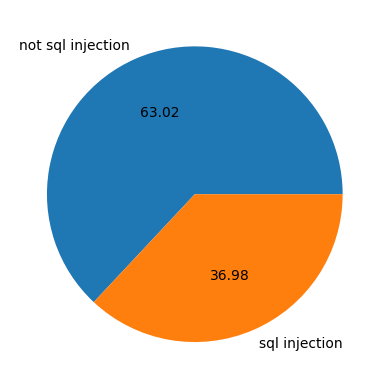
This project focuses on detecting SQL Injection attacks using various machine learning and deep learning models. We implemented five models: **SVM**,**Logistic Regression**, **MLP**, **RNN**, and **LSTM**. These models were trained on a labeled dataset to distinguish between malicious and normal SQL queries.

The goal is to compare their performance and establish a baseline before evaluating a more advanced model based on **BERT**. This report summarizes the implementation and results of the six models for validation.

2 **Dataset Description**

This study uses two separate datasets for training and testing purposes:

* **Training Dataset (SQLIV3\_cleaned2.csv)**:
  + Contains **30,614 SQL queries**.
  + Each entry is composed of a Sentence (the SQL query) and a Label (0 for benign, 1 for SQL Injection).
  + Label distribution:
    - **Normal (Label = 0)**: 19,268 queries
    - **Malicious (Label = 1)**: 11,346 queries



* **Testing Dataset (sqliv2\_utf8.csv)**:
  + Contains **33,760 SQL queries**.
  + Structured in the same format with Sentence and Label columns.
  + Label distribution:
    - **Normal (Label = 0)**: 22,305 queries
    - **Malicious (Label = 1)**: 11,455 queries

**(SQLIV3\_cleaned2.csv)**

Before training, both datasets were cleaned by removing duplicate queries. These queries were then vectorized using appropriate text processing techniques such as **TF-IDF** or **token embeddings**, depending on the model.

3.1 Support Vector Machine (SVM)

We used a Support Vector Machine (SVM) classifier with a linear kernel and a regularization parameter of C=1. The SQL queries were first transformed into numerical features prior to training via TF-IDF vectorization. This allowed the model to learn how much weight each token carried in recognizing wicked patterns.

The model was trained on the cleaned-up training set and validated against the independent test set.

Results:

•Accuracy: **98.78%**

The SVM model performed equally well, with an extremely high precision that indicates its strong ability to detect SQL Injection attempts properly and prevent false positives. These results create a good traditional machine learning benchmark for comparison against deep learning models and transformer-based models.

**3.2 Regression logestic**

We utilized the scikit-learn library to implement a logistic regression model that detects SQL injection attacks from a labeled text sample dataset. The data was first preprocessed by removing the duplicates and unwanted characters, then converted into numerical vectors using the TF-IDF method, which maintains word importance in the dataset. The logistic regression algorithm was trained to predict inputs as SQL injection or not, and we tuned parameters and performed model optimization by verifying performance on training and validation sets.

After being preprocessed, the data contained 11,424 samples with labels of SQL injections (label = 1). For the training data, the logistic regression model performed an overall accuracy of 98.12% and precision of 99.73%, reflecting a very high generalization capability and a very low rate of false positives. These results reflect the model's strength in text classification of cybersecurity data. It identified 10,892 malicious queries correctly with a detection accuracy of 95.34% when evaluated on the test set, thus confirming its position as a trusted baseline method for SQL injection attack detection from natural language processing.

**3.3Multilayer Perceptron (MLP)**

To contrast the performance of neural networks in classifying SQL injection attacks, two Multilayer Perceptron (MLP) architectures were utilized utilizing the Keras Sequential API. The purpose was to compare the effect of hidden layers on classification performance.

This model consists of three hidden layers with diminishing neuron dimensions and ReLU activations and a sigmoid output layer for binary classification.

Architecture:

* • **Input dimension**: 3000 (TF-IDF vector size)
* Dense(512), activation = ReLU
* Dense(256), activation = ReLU
* Dense(128), activation = ReLU
* Dense(1), activation = Sigmoid

**Training Configuration**:

* Optimizer: SGD (learning rate = 0.01)
* Loss function: Binary Crossentropy
* Epochs: 27
* Batch size: 32
* Validation split: 20%

Results:

• Final training loss: 0.0304

• Final training accuracy: 99.40%

• Test accuracy (on SQLi samples): **98.78%**

• Number of SQL injection queries detected**: 11,314** out of **11,455**

3.4 simple Neural Network

A baseline model with fewer layers was built, using just an input layer and an output layer (no hidden layers). The intention was to serve as a point of comparison when quantifying the impact of having hidden layers.

Architecture:

•(Dense(1), input\_dim = 3000, activation = Sigmoid)

Training Configuration:

• Optimizer: SGD (learning rate = 0.01)

• Loss function: Binary Crossentropy

• Epochs: 27

• Batch size: 32

• Validation split: 20%

Results:

• Training loss: 0.3927

• Training accuracy: 92.00%

• Test accuracy (on SQLi samples): **83.10**

• Total SQL injection queries detected: 9,518 out of 11,455

3.5 RNN

We used a deep learning model with **SimpleRNN** to detect SQL injection attacks. Training data was deduplicated, and text was lightly preprocessed to preserve SQL-specific characters. Sentences were tokenized without filtering symbols, and sequences were padded based on the 95th percentile of sentence lengths. The model was built using embedding and two stacked SimpleRNN layers with following dense layers for enhanced learning capacity.

The model achieved an extremely high test accuracy of 99.33%, with good detection ability for malicious SQL queries. Specifically, it properly identified **11,353** out of 11,424 SQL injection samples in the test set, with a detection rate of **99.43**%. These statistics demonstrate the model's ability for identifying SQL injection attack patterns.

3.6 LSTM

We employed an LSTM-based deep learning model for the detection of SQL injection attacks. Our dataset was preprocessed by removing duplicates and minimal text preprocessing. After tokenization and padding, the model was built with embedding layers, LSTM layers, and dropout layers to prevent overfitting. It was trained with the Adam optimizer and binary cross-entropy loss.

The model has a testing accuracy of 99.06%, with **99.51%** detection of SQL injection attacks, correctly identifying **11,362** out of 11,424 spam queries. This supports the efficacy of the model in SQL injection vulnerability detection and prevention

