**1. How the Attack Occurs (SQL Injection)**

SQL Injection (SQLi) occurs when an attacker exploits improperly sanitized user input in an SQL query. By injecting malicious SQL code into a query string (usually via input fields like username or search), the attacker can manipulate the structure of the original query.



**Attacker**  
The attacker targets a web application by entering malicious SQL code into input fields, such as the Username and Password fields shown in the image.  
Instead of submitting normal login credentials, the attacker might enter something like:

**Web Server**  
The web server receives this input and insecurely incorporates it into an SQL query, without proper input sanitization. The resulting query may look like:

**Database**  
The manipulated SQL query is then executed by the database. Because the logic has been altered by the attacker’s input, the database may:

- Return all user records,

- Allow unauthorized access,

- Or even expose, alter, or delete sensitive data.

**Result:**

This entire process allows an attacker to bypass authentication, steal data, or even take control of the system, depending on the severity of the vulnerability.

**Example Attack:**  
A normal login query:

SELECT \* FROM users WHERE username = 'admin' AND password = '1234';

An injected version:

SELECT \* FROM users WHERE username = 'admin' OR '1'='1' --' AND password = '1234';

The injected condition '1'='1' always evaluates to true, allowing unauthorized access. In IoT-based financial platforms, this can lead to unauthorized account control, data theft, or financial manipulation.

1. **How the Machine Learning Model Detects Abnormal Queries in Real-Time:**

The machine learning detection system works by learning patterns in labeled SQL queries7specifically the differences between benign and malicious query structures.

**How it operates in real-time:**

* **Input Layer:** A user-submitted SQL query is passed to the system (e.g., via a login form or API).
* **Preprocessing:** The query is cleaned—converted to lowercase, special characters removed, digits masked, and whitespace normalized.
* **Vectorization:** The cleaned query is converted into a numerical vector using a TF-IDF vectorizer, which captures token frequency and importance.
* **Prediction:** The vector is passed to a pre-trained Random Forest model, which classifies it as benign (0) or malicious (1).

**Real-Time Output:** The classification result is returned instantly and logged.

By learning structural anomalies like frequent use of keywords (OR, UNION, SELECT, DROP) in unnatural contexts the model identifies suspicious patterns without relying on hardcoded rules.

1. **Did You Modify the SQL Dataset? If Yes, How?**

**YES.** The original dataset from Kaggle was modified to create **Modified\_SQL\_Dataset.csv** to improve consistency, relevance, and training efficiency.

Modifications included:

Data Cleaning:

* Lowercased all SQL queries to reduce case variance.
* Replaced all numeric values with 0 (digit masking).
* Removed or normalized special characters (e.g., excessive punctuation).
* Stripped unnecessary whitespace and extra tokens.

Format Standardization:

* Ensured all queries followed a standard format to improve model learning.

Noise Reduction:

* Removed duplicate or excessively short queries that offered no useful learning pattern.

**4. What Is the Actual Pre-Test Score?**

The actual model evaluation scores from the pre-test phase are as follows:

|  |  |
| --- | --- |
| Metric | Score |
| Accuracy | 99.53% |
| Precision | 99.96% |
| Recall | 98.78% |
| F1-Score | 99.36% |

These metrics were generated using a trained Random Forest Classifier on an 80/20 train-test split of the dataset. The model showed extremely high precision, meaning it rarely misclassified benign queries, and strong recall, meaning it successfully identified most malicious ones.

**5. How Was the Data in the Pre-Test Analysis Generated?**

The analysis was based on predictions from the test set (20% of the cleaned dataset) evaluated against their true labels.

Step 1: The Modified\_SQL\_Dataset.csv was cleaned and transformed using TF-IDF vectorization.

Step 2: The data was split (80% training, 20% testing).

Step 3: The model predicted labels for the test set.

Step 4: A confusion matrix was generated:

* TP (True Positives): Malicious queries correctly identified
* FP (False Positives): Benign queries incorrectly flagged
* TN (True Negatives): Benign queries correctly identified
* FN (False Negatives): Malicious queries missed

Step 5: Metrics were computed from these counts:

* Accuracy = (TP + TN) / Total
* Precision = TP / (TP + FP)
* Recall = TP / (TP + FN)
* F1 = Harmonic mean of precision and recall

These results were saved to CSV files and visualized in Power BI and Streamlit.