STUDENT NAME:  **OJI VICTOR KALU**

REGISTRATION NUMBER**:** **20201246182**

PROJECT TOPIC: **IOT SECURITY IN FINANCIAL ECOSYSTEMS: ENSURING COMPLIANCE WHILE MITIGATING CYBER RISKS**

**Attack Identified:** SQL INJECTION

**Attack pattern: design on how the attack occurs (description of your design below the pattern)**

Description of How the Attack Occurs:

1. Entry Point: The attacker identifies input fields (e.g., login forms, search bars) in an IoT-enabled financial system that directly interact with the database.
2. Payload Injection: The attacker injects malicious SQL code such as **' OR '1'='1** into the input field.
3. Execution: The backend system, lacking proper input sanitization, executes the SQL query with the injected code.
4. Impact: This may result in unauthorized data access, account takeover, or complete database dumps.

Example: **SELECT \* FROM users WHERE username = 'admin' AND password = '' OR '1'='1';**

**Mitigation Pattern: Design**

Design on How the Attack Could Be Mitigated:

1. Input Validation & Sanitization: All user inputs must be validated against a whitelist of acceptable characters.
2. Use of Prepared Statements: Use parameterized queries to prevent SQL commands from being interpreted as data.
3. Database Permissions: Least privilege should be enforced to limit what the application can do.
4. Web Application Firewall (WAF): Deploy WAFs to filter and block malicious traffic.

Example: **cursor.execute("SELECT \* FROM users WHERE username = %s AND password = %s", (username, password))**

**1. Methodology**

Machine Learning-Based Methodology

**2.Data collection**: state the dataset collected and source

* state the dataset collected :**Kaggle - SQL Injection Detection Dataset**
* source :**Kaggle**

1. **Instrument Used for Pre-Testing Existing Attacks**

|  |  |
| --- | --- |
| Component | Role |
| TF-IDF Vectorizer | Converts raw SQL queries into numerical vectors based on term frequency |
| Random Forest Model | Learns patterns that differentiate between SQLi and normal queries |

**4.Analyze the pre-test results to figure out the actual problem in the existing system ( explain how the analysis is done mathematically PowerBi**

* Use the Confusion Matrix (Mathematical Basis)

You already evaluated the model using:

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

Behind the scenes, these are calculated from a confusion matrix:

|  | Predicted: SQLi (1) | Predicted: Normal (0) |
| --- | --- | --- |
| Actual: SQLi (1) | TP = 1234 | FN = 15 |
| Actual: Normal (0) | FP = 2 | TN = 1468 |

Use the formulas:

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Precision = TP / (TP + FP)

Recall = TP / (TP + FN)

F1-Score = 2 × (Precision × Recall) / (Precision + Recall)

Problem Identification:

False Negatives (FN = 15): The model missed 15 SQL injection attempts.

This is a security risk — undetected attacks can be dangerous in financial systems.

False Positives (FP = 2): 2 normal queries were wrongly flagged.

This affects usability but is less risky than FN.

2. Visualize in Power BI (Step-by-Step)

📁 A. Load Confusion Matrix Data into Power BI

Create a table with:

| Category | Count |
| --- | --- |
| True Positives | 1234 |
| False Positives | 2 |
| False Negatives | 15 |
| True Negatives | 1468 |

Recommended Power BI Visuals

Stacked Column Chart

X-Axis: Category

Y-Axis: Count

Use colors to highlight FP and FN

KPI Cards

Show Precision, Recall, F1-Score, Accuracy

You can create custom measures in Power BI:

Accuracy = DIVIDE([TP] + [TN], [TP] + [TN] + [FP] + [FN])

Precision = DIVIDE([TP], [TP] + [FP])

Recall = DIVIDE([TP], [TP] + [FN])

F1Score = 2 \* ([Precision] \* [Recall]) / ([Precision] + [Recall])

Conclusion from Pre-Test Analysis

| Observation | Impact | Action |
| --- | --- | --- |
| FN > 0 (some SQLi not detected) | High security risk → attackers can bypass detection | Add more features, boost recall |
| FP exists (normal flagged as SQLi) | Low usability issue → safe queries rejected | May need tuning or threshold adjustment |
| High precision + low FN | Model is very accurate but not perfect | Possibly acceptable for now, but log FNs |

1. **Explain how your data was cleaned and reason for cleaning the data**

**What Was Cleaned:**

**The raw dataset (Modified\_SQL\_Dataset.csv) contained a mix of normal and SQL injection queries. These queries often had:**

**How the Data Was Cleaned**

The dataset contained various SQL queries, including both normal and malicious ones. These queries were cleaned using a series of preprocessing steps to make them suitable for machine learning. Here's what was done:

**Lowercasing:**  
All queries were converted to lowercase so that the model wouldn’t treat SELECT and select as different words.

**Replacing Numbers:**  
All numeric values were replaced with the digit 0. This helps the model focus on the structure of the query rather than specific values, which can vary widely but mean the same thing (e.g., id = 1 vs. id = 100).

**Removing Special Characters**:  
Most non-alphanumeric characters (except important ones like ', ", =, and \_) were removed. This reduced noise from irrelevant symbols and helped standardize the input.

**Trimming Extra Spaces:**  
Extra whitespace was removed to make tokenization more accurate.

**Handling Missing Data:**  
Any rows with missing queries or labels were removed to ensure the model only trained on complete data.

**Why Cleaning Was Done**

Improve Accuracy: Clean, consistent data helps the model learn meaningful patterns instead of memorizing irrelevant details.

Reduce Noise: Removing unnecessary characters and normalizing inputs avoids confusing the model.

Standardize Input: Ensures that the same query written in different ways is treated the same.

Prevent Overfitting: Generalizing numbers and symbols helps the model learn logic, not exact values.

Better Feature Extraction: Clean text is easier to tokenize and vectorize (especially for TF-IDF).

**6.Classify your data for both training and testing**