Vaishika Agrwal

Roll No. :2201AI42

**SQL Injection Detector Using Reinforcement Learning**

**SQL Injection Detector Using Reinforcement Learning**

The system aims to generate new SQL queries by implementing Reinforcement Learning and predicting the values of newly generated queries. The goal is to train an agent to learn optimal actions.

There are two agents:

1. DQN Agent
2. Simple Agent

Basic workflow of the system:

1. **Initialization**:

* The environment is initialized with a set of predefined queries and a security model.
* The RL agent starts with no knowledge of effective injection techniques.

1. **Action and Reward Loop**:

* The agent selects an action to modify a query.
* The environment evaluates the modified query using the security model.
* A reward is assigned based on the query's ability to bypass the model.

1. **Learning**:

* The agent learns over multiple episodes to maximize rewards, refining its strategy for crafting SQL injection payloads.

The project comprises of the three parts. Let’s dive into the detailed implementation of the code.

**A) Training the ML model**

**1. Data Preparation**

* **Load Data**: The dataset sqli\_dataset.csv is loaded using pandas. It contains:
  + **query**: The input SQL query strings.
  + **label**: The labels indicating whether the query is an SQLi (1) or a normal query (0).
* **Regex-Based SQLi Detection**:
  + The function check\_regex\_sql\_patterns uses regular expressions to detect potential SQLi patterns in the queries, such as suspicious keywords (SELECT, UNION, OR 1=1), comments (--, #), and WHERE clauses.
  + The detection adds a binary flag regex\_sqli\_flag to indicate SQLi presence.

**2. Splitting Dataset**

* The dataset is split into training and testing sets:
  + **Training Set (80%)**: Used for training the machine learning model.
  + **Testing Set (20%)**: Used to evaluate the trained model.

**3. Pipeline Creation**

* A **Pipeline** is created with two steps:
  1. **TfidfVectorizer**:
     + Converts textual SQL queries into numerical features using TF-IDF, capturing the importance of terms.
  2. **Multinomial Naive Bayes Classifier**

**4. Model Training**

* Model is trained using Naive Bayes Classifier.

**5. Model Evaluation**

* Printed confusion matrix and classification report

**6. Saving the Model**

* The trained pipeline is saved to sqli\_model.pkl for later use.

**B) IMPLEMENTING REINFORCEMENT LEARNING**

The implementation comprises of three parts:

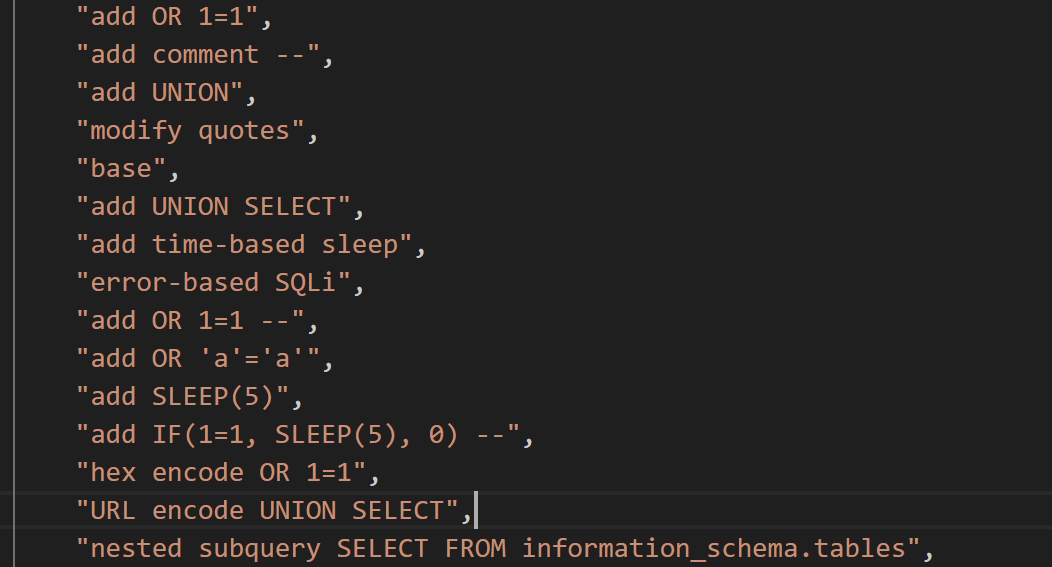
1. SQL Injection Environment
2. DQN Agent
3. Simple Q-Learning Agent.

**1. SQL Injection Environment**

The **environment** simulates the interaction between the agent and the SQLi detection model.

**Key Features**

* **Queries**: A list of SQL queries is provided to the environment (queries). The queries are from the dataset used to train our ML model. These are used as the starting points for modifications.
* **Actions**: The agent can select from a predefined list of **modifications** to apply to a query (e.g., adding OR 1=1, changing quotes, or injecting a UNION SELECT statement).



* **State Representation**:
  + The state is represented as a one-hot vector of length equal to the number of queries (e.g., if there are 5 queries, the state could be [0, 1, 0, 0, 0]).
* **Rewards**:
  + If the query successfully bypasses the detection model (check\_bypass), the agent receives a reward of 1.
  + A diversity bonus of 0.1 is added for generating unique queries.
* **Methods**:
  + **modify\_query(query, action)**: Modifies the query based on the selected action.
  + **reset()**: Randomly initializes the environment state.
  + **step(action)**:
    - Executes the chosen action.
    - Determines the reward based on the model's prediction (check\_bypass).
    - Transitions to a new query state.
    - Signals whether the episode is complete.

**Objective:**

Agents aim to learn which actions (modifications) will most likely bypass the SQLi detection system.

**2. Deep Q-Network (DQN) Agent**

The **DQN Agent** uses reinforcement learning with neural networks to learn optimal actions.

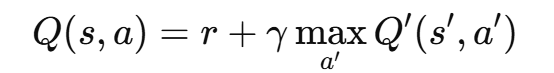
**Key Features**

1. **Neural Networks**:
   1. **Q-Network**

* **Purpose**: This network approximates the Q-function Q(s,a;θ), which predicts the expected cumulative reward for taking a specific action a in a given state s.
* **Structure**:
  + **Input**: Represents the current state s, typically as a vector.
  + **Hidden Layers**: Process the input through a series of fully connected (dense) layers with nonlinear activation ReLU.
  + **Output**: Produces Q(s,a;θ) for all possible actions. The number of outputs equals the size of the action space.

**1.2 Target Q-Network**

* **Purpose**: Provides stable Q-value targets during training, avoiding oscillations or divergence in the learning process.
* **Structure**: Identical to the Q-network (same layers and architecture), but the weights are updated less frequently.
* **Usage**:
  + During training, the target Q-network calculates the target Q-values using the Bellman equation.



**r:** The immediate reward received after taking action aaa in state sss.

**γ:** The discount factor controls the importance of future rewards.

* + Periodically, the weights of the Q-network are copied to the target Q-network using update\_target\_network.

1. **Experience Replay**

* **Purpose**: Stores past experiences (state, action, reward, next state, done) in a buffer to break the correlation between consecutive samples and make learning more efficient.
* **Process**:
  + Experiences are sampled randomly in minibatches during training, ensuring that updates are based on a diverse set of past experiences rather than a single trajectory.

1. **Loss Function**

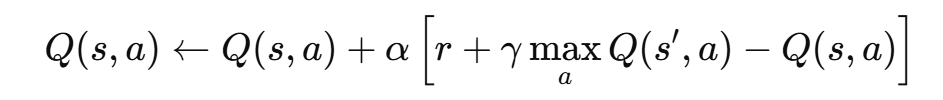
* The agent minimizes the mean squared error (MSE) loss between the predicted Q-values and the target Q-values

**3. Simple Q-Learning Agent**

The **Simple Agent** uses a classic tabular Q-learning approach.

**Key Features**

1. **Q-Table**:
   * A matrix where rows represent states and columns represent actions.
   * Each entry, Q(s,a), represents the expected future reward for taking action a in state s.
2. **Learning Rule**:
   * Updates the Q-value using the Bellman equation:



1. **Exploration vs. Exploitation**:
   * **Epsilon-Greedy Policy**:
     + Chooses a random action with probability epsilon (exploration).
     + Chooses the best-known action with 1 - epsilon (exploitation).
     + Epsilon decays over time, gradually shifting from exploration to exploitation.
2. **Training Process**:
   * **Loss Function**:
     + The loss is the mean squared error (MSE) between the predicted Q-values and the target Q-values.
     + The Q-network learns to predict these target Q-values.
   * **Optimizer**:
     + Uses the Adam optimizer to update the network's weights.

5. **Main Program**

**Data and Environment Initialization**:

* + Loads a pre-trained SQLi detection model (sqli\_model.pkl).
  + Reads a dataset of queries (sqli\_dataset.csv).
  + Initializes the SQLInjectionEnv with queries and actions.

**Agent Training**:

* + For **DQNAgent**:
    - Runs multiple episodes.
  + For **Simple Agent**:
    - Runs multiple episodes.
    - Updates the Q-table after each action using the Q-learning rule.
    - Gradually reduces epsilon for better exploitation.

**Query Generation**:

* + Both agents generate modified queries during training.
  + Saves these queries to separate CSV files for further analysis.

**User Input and Detection**:

* + Accepts a user-provided SQL query.
  + Uses the SQLi detection model to classify the query as malicious or benign.

**C) Predicting newly generated queries.**

* The newly generated queries the agents generate are passed to our ML model.