

# WAF SQLi: Protección de Inyecciones SQL mediante un Modelo de Predicciones

Diego Joel Condori Quispe

28 de Noviembre de 2023

## Abstract

Este artículo presenta WAF SQLi, un módulo de protección contra inyecciones SQL diseñado para integrarse con frameworks populares como Flask, Django, Laravel, así como con sitios web desarrollados en PHP. El módulo utiliza un enfoque de aprendizaje automático basado en el procesamiento de lenguaje natural y técnicas de clasificación para identificar y prevenir ataques de inyección SQL.

## 1 Introducción

Las inyecciones SQL representan una de las amenazas más significativas para la seguridad de las aplicaciones web. Este trabajo introduce WAF SQLi, un módulo de protección adaptable a diversos entornos de desarrollo web, incluyendo frameworks como Flask, Django, Laravel y aplicaciones PHP. El módulo se basa en técnicas avanzadas de aprendizaje automático para ofrecer una solución robusta y eficiente contra las inyecciones SQL.

## 2 Justificación del Uso de SVM y TFIDFVectorizer en la Detección de Inyecciones SQL

### 2.1 TFIDFVectorizer: Preprocesamiento de Datos

El preprocesamiento de datos es un paso crucial en cualquier tarea de aprendizaje automático. En el contexto de la detección de inyecciones SQL, donde los datos son principalmente textuales, es esencial transformar este texto en un formato que pueda ser procesado por algoritmos de aprendizaje automático. Aquí es donde entra en juego TFIDFVectorizer.

TFIDFVectorizer convierte una colección de documentos de texto en una matriz de características TF-IDF (Term Frequency-Inverse Document Frequency). Esta técnica pondera la frecuencia de cada palabra (Term Frequency) contra su inversa de frecuencia en todos los documentos (Inverse Document Frequency),

resaltando la importancia de las palabras que son únicas en un documento específico y menos comunes en otros documentos. Este enfoque es particularmente útil para resaltar las características distintivas en consultas SQL que podrían indicar una inyección [5].

## 2.2 SVM: Clasificación de Consultas SQL

Una vez que los datos están preprocesados con `TFIDFVectorizer`, el siguiente paso es la clasificación. Las Máquinas de Vectores de Soporte (SVM) son ideales para este propósito debido a su eficacia en la clasificación de datos de alta dimensión, como es el caso de las características textuales transformadas.

La capacidad de las SVM para manejar espacios de características de gran dimensión y su eficiencia en la clasificación binaria las hacen particularmente adecuadas para distinguir entre consultas SQL legítimas y maliciosas. Además, el uso del truco del kernel en SVM permite abordar la no linealidad potencial en los patrones de inyección de SQL, lo que mejora aún más la precisión del modelo [3, 2].

## 2.3 Combinación de `TFIDFVectorizer` y SVM en la Detección de Inyecciones SQL

La combinación de `TFIDFVectorizer` y SVM se convierte en una poderosa herramienta para la detección de inyecciones SQL. Mientras que `TFIDFVectorizer` prepara los datos textuales para el análisis, SVM proporciona un mecanismo robusto y eficiente para la clasificación. Esta combinación permite identificar patrones sutiles y complejos en las consultas SQL, lo que es crucial para detectar y prevenir ataques de inyección SQL en aplicaciones web [1].

La integración de `TFIDFVectorizer` y SVM en un sistema de detección de inyecciones SQL ofrece un enfoque equilibrado entre el preprocesamiento eficiente de datos textuales y una clasificación precisa. Esta metodología no solo mejora la capacidad de detectar inyecciones SQL, sino que también se adapta bien a diferentes frameworks y entornos de programación web, lo que la hace ideal para su implementación en una variedad de aplicaciones web [4].

## 3 Justificación de la No Aplicación de PCA

El dataset utilizado en este proyecto consta de dos columnas fundamentales: 'Query', que comprende las consultas SQL, y 'Label', que indica la presencia o ausencia de inyecciones SQL. Dada la naturaleza de estos datos, la aplicación de técnicas de reducción de dimensiones como el Análisis de Componentes Principales (PCA) no se consideró necesaria por varias razones.

Primero, el preprocesamiento mediante `TfidfVectorizer` transforma las consultas SQL en un conjunto de características numéricas manteniendo la relevancia de los términos. Este enfoque asegura que el modelo SVM pueda interpretar eficazmente los datos sin perder información contextual vital.

Además, el dataset no presenta una alta dimensionalidad que justifique el uso de PCA. La complejidad y el tamaño de las características generadas por TfidfVectorizer son manejables y adecuadas para el modelo de aprendizaje automático utilizado.

Importante destacar, la aplicación de PCA podría conllevar a la pérdida de información contextual crítica en las consultas SQL. Esta información es esencial para la identificación precisa de inyecciones SQL, ya que el contexto y la estructura de las consultas son indicadores clave de actividades maliciosas.

Finalmente, los resultados obtenidos del modelo actual, sin la aplicación de PCA, han demostrado ser altamente eficientes y precisos. La matriz de confusión y la confiabilidad ajustada reflejan la capacidad del modelo para clasificar correctamente las consultas SQL, validando así la decisión de no aplicar reducción de dimensiones en este contexto.

## 4 Metodología

El proceso de desarrollo del modelo de detección de inyecciones SQL se inició con una fase de preprocesamiento de datos. Utilizamos el conjunto de datos disponible en Kaggle, específicamente diseñado para la detección de inyecciones SQL. Este conjunto de datos contiene consultas SQL etiquetadas, clasificadas como legítimas o como inyecciones SQL. El preprocesamiento de los datos se realizó utilizando la técnica de vectorización TF-IDF (Term Frequency-Inverse Document Frequency), una técnica común en el procesamiento de lenguaje natural para convertir texto en un formato numérico adecuado para el análisis de aprendizaje automático.

El código en Python(prepro\_train.py) para el preprocesamiento es el siguiente:

```
% Importar librerías
Importar pandas como pd
Importar TfidfVectorizer de sklearn.feature_extraction
    .text
Importar train_test_split de sklearn.model_selection
Importar pickle

% Cargar el dataset
df <- Leer_CSV('dataset.csv')

% Separar las características y el objetivo
X <- df['Query']
y <- df['Label']

% Dividir el dataset en conjunto de entrenamiento y
    prueba
X_train, X_test, y_train, y_test <- train_test_split(X
    , y, test_size=0.2, random_state=42)
```

```
% Inicializar TF-IDF Vectorizer y transformar los
  datos
vectorizer <- TfidfVectorizer()
X_train_tfidf <- vectorizer.fit_transform(X_train)
X_test_tfidf <- vectorizer.transform(X_test)

% Guardar los datos transformados y el vectorizador
  para su uso posterior
Abrir_Archivo('vectorizer.pkl', 'wb') como file:
  Guardar(vectorizer, file)
Abrir_Archivo('train_test_data.pkl', 'wb') como file:
  Guardar((X_train_tfidf, X_test_tfidf, y_train,
    y_test), file)
```

Posteriormente, se procedió al modelado utilizando el algoritmo Support Vector Machine (SVM) con un kernel lineal. Este modelo fue elegido por su eficacia en la clasificación de datos de alta dimensión, como es el caso de los textos transformados mediante TF-IDF. El modelo fue entrenado utilizando los datos preprocesados y luego guardado como un objeto serializado para su uso posterior. El código (modelado.py) para el modelado es el siguiente:

```
\begin{verbatim}

Importar SVC desde sklearn.svm
Importar pickle

# Cargar los datos transformados
Abrir 'train_test_data.pkl' en modo lectura como
  archivo
  Cargar X_train_tfidf, X_test_tfidf, y_train,
    y_test desde archivo

# Entrenar el modelo SVM
Definir svm_model como SVC con kernel lineal y
  random_state 42
Entrenar svm_model con X_train_tfidf y y_train

# Guardar el modelo entrenado
Abrir 'svm_model.pkl' en modo escritura como archivo
  Guardar svm_model en archivo
```

Esta metodología permitió desarrollar un modelo robusto y eficiente para la detección de inyecciones SQL, adaptable a diferentes frameworks y lenguajes de programación web.

## 5 Evaluación del Modelo

### 5.1 Matriz de Confusión y Confiabilidad

Una vez entrenado el modelo, se procedió a evaluar su rendimiento utilizando el conjunto de prueba. Se calculó la matriz de confusión y la confiabilidad (accuracy) ajustada utilizando un umbral específico. El umbral se determinó basándose en el percentil 75 de las puntuaciones de decisión del modelo. El código (confusion\_accuracy.py) utilizado para esta evaluación es el siguiente:

```
% Importar librerías
Importar pickle
Importar numpy como np
Importar accuracy_score, confusion_matrix de sklearn.
    metrics

% Cargar los datos de prueba y el modelo entrenado
Abrir_Archivo('train_test_data.pkl', 'rb') como file:
    _, X_test_tfidf, _, y_test <- Cargar(file)
Abrir_Archivo('svm_model.pkl', 'rb') como file:
    svm_model <- Cargar(file)

% Realizar predicciones y calcular el umbral
y_scores <- decision_function(svm_model, X_test_tfidf)
umbral <- Percentil(y_scores, 75)
y_pred_ajustado <- (y_scores > umbral) como entero

% Matriz de confusión y confiabilidad ajustada
matriz_confusion_ajustada <- confusion_matrix(y_test,
    y_pred_ajustado)
exactitud_ajustada <- accuracy_score(y_test,
    y_pred_ajustado)
```

La matriz de confusión ajustada y la confiabilidad proporcionan una visión clara del rendimiento del modelo en la clasificación de consultas SQL como legítimas o como inyecciones SQL.

### 5.2 Análisis de Splits y Mediana de la Confiabilidad

Para asegurar la robustez del modelo, se realizaron múltiples divisiones aleatorias del conjunto de datos en entrenamiento y prueba. Se calcularon dos tipos de splits: un split académico (80% entrenamiento, 20% prueba) y un split de investigación (50% entrenamiento, 50% prueba). En cada split, se evaluó la confiabilidad del modelo y se calculó la mediana de estas confiabilidades para obtener una medida más estable del rendimiento del modelo. El código(splits.py) para este análisis es el siguiente:

```
% Importar librerías
```

```

Importar random
Importar pandas como pd
Importar train_test_split de sklearn.model_selection
Importar SVC de sklearn.svm
Importar accuracy_score de sklearn.metrics
Importar numpy como np

% Cargar tus datos
df <- Leer_CSV('dataset.csv')
data <- Convertir_a_Lista(df)

% Funcion para calcular la mediana de la confiabilidad
  con ajuste de umbral
Funcion calcular_mediana_confianza_con_umbral(data,
  test_size, n_splits, umbral_percentil):
  accuracies <- Lista_Vacia()
  Por _ en Rango(n_splits):
    % Mezclar los datos aleatoriamente y
      dividirlos
    Mezclar_Aleatoriamente(data)
    train_data <- data[:Entero(Longitud(data) * (1
      - test_size))]
    test_data <- data[Entero(Longitud(data) * (1 -
      test_size)):]

  % Convertir a DataFrame y separar
    características y etiquetas
  train_df <- DataFrame(train_data, columnas=df.
    columnas)
  test_df <- DataFrame(test_data, columnas=df.
    columnas)
  X_train, y_train <-
    Separar_Caracteristicas_Y_Etiquetas(
      train_df)
  X_test, y_test <-
    Separar_Caracteristicas_Y_Etiquetas(test_df
    )

% Entrenar el modelo SVC
modelo <- Crear_Modelo_SVC(kernel='linear',
  random_state=42, probability=True)
Entrenar(modelo, X_train, y_train)

% Calcular el umbral y ajustar las
  predicciones
y_scores <- decision_function(modelo, X_test)

```

```

        umbral <- Percentil(y_scores, umbral_percentil
        )
        y_pred_ajustado <- (y_scores > umbral) como
        entero

        % Calcular y almacenar la precision
        Agregar_A_Lista(accuracies, accuracy_score(
        y_test, y_pred_ajustado))

        % Calcular y devolver la mediana de las precision
        Devolver Mediana(accuracies)

% Calcular la mediana de la confiabilidad para el
split academico (80/20)
mediana_academico <-
    calcular_mediana_confianza_con_umbral(data,
    test_size=0.2, n_splits=100, umbral_percentil=75)
Imprimir('Mediana de la confiabilidad para el split
academico: ', mediana_academico)

% Calcular la mediana de la confiabilidad para el
split de investigacion (50/50)
mediana_investigacion <-
    calcular_mediana_confianza_con_umbral(data,
    test_size=0.5, n_splits=100, umbral_percentil=75)
Imprimir('Mediana de la confiabilidad para el split de
investigacion: ', mediana_investigacion)

```

Estos análisis de splits y la mediana de la confiabilidad proporcionan una evaluación exhaustiva del modelo, asegurando su aplicabilidad en diferentes contextos y con distintas proporciones de datos de entrenamiento y prueba.

## 6 Resultados y Discusión

### 6.1 Resultados de la Matriz de Confusión y Confiabilidad Ajustada

Los resultados obtenidos de la ejecución del script para la matriz de confusión y la confiabilidad ajustada son los siguientes:

Matriz de confusión:

$$\begin{bmatrix} 3892 & 1 \\ 746 & 1545 \end{bmatrix}$$

Confiabilidad Ajustada: 0.8792043984476067

La matriz de confusión muestra un alto número de verdaderos positivos (3892) y verdaderos negativos (1545), con una mínima cantidad de falsos positivos (1) y un número moderado de falsos negativos (746). Esto indica que el modelo es altamente efectivo en identificar las consultas SQL legítimas y también es competente en detectar inyecciones SQL, aunque con cierta propensión a no identificar todas las inyecciones. La confiabilidad ajustada de aproximadamente 87.92% refleja la alta precisión del modelo en la clasificación general.

## 6.2 Resultados de los Splits y Mediana de la Confiabilidad

Los resultados obtenidos de la ejecución del script para calcular la mediana de la confiabilidad en los splits académico e investigativo son los siguientes:

Mediana de la confiabilidad para el split academico: 0.6307406209573092

Mediana de la confiabilidad para el split de investigacion: 0.6314683053040104

Estos resultados muestran que la mediana de la confiabilidad para ambos splits es cercana al 63%, lo cual es considerablemente más bajo que la confiabilidad ajustada obtenida en la evaluación de la matriz de confusión. Esta diferencia puede atribuirse a la variabilidad en la división de los datos durante los múltiples splits. Mientras que la confiabilidad ajustada proporciona una medida de rendimiento en un conjunto de prueba específico, la mediana de la confiabilidad en los splits ofrece una visión más generalizada y robusta del rendimiento del modelo, teniendo en cuenta diferentes divisiones aleatorias del conjunto de datos.

## 6.3 Discusión

La relación entre la exactitud de la matriz de confusión, la confiabilidad y la mediana de los splits destaca la importancia de evaluar los modelos de aprendizaje automático desde múltiples perspectivas. Mientras que la matriz de confusión y la confiabilidad ajustada ofrecen una visión detallada del rendimiento del modelo en un conjunto de prueba específico, la mediana de la confiabilidad en los splits proporciona una evaluación más general y menos sesgada del modelo. Esto es especialmente relevante en aplicaciones prácticas donde el modelo se enfrentará a datos variados y posiblemente no vistos anteriormente. Por lo tanto, estos resultados subrayan la robustez y la fiabilidad del modelo WAF SQLi en la detección de inyecciones SQL en diferentes contextos y conjuntos de datos.

## 7 Implementación del Módulo de Verificación

Una parte crucial del proyecto WAF SQLi es el script que permite verificar si una consulta SQL es legítima o una inyección SQL. Este script utiliza el modelo entrenado y el vectorizador TF-IDF para evaluar consultas SQL en tiempo real. A continuación, se presenta el código del script y su explicación:



```

% Importar librerias
Importar pickle
Importar numpy como np
Importar classification_report, accuracy_score de
    sklearn.metrics

% Cargar el modelo y el vectorizador
Abrir_Archivo('svm_model.pkl', 'rb') como file:
    svm_model <- Cargar(file)

Abrir_Archivo('vectorizer.pkl', 'rb') como file:
    vectorizer <- Cargar(file)

% Funcion para predecir una nueva entrada con ajuste
    de umbral
Funcion predecir_inyeccion_sql_ajustado(query, umbral,
    vectorizer, modelo):
    query_tfidf <- Transformar(query, vectorizer)
    score <- decision_function(modelo, query_tfidf)
    prediccion <- (score > umbral) como entero
    Devolver "Inyeccion SQL" si prediccion == 1, De lo
        contrario, Devolver "Consulta SQL legitima"

% Ejemplo de uso con umbral ajustado
input_query <- Argumento_del_Sistema(1) % Reemplaza
    esto con tu entrada
resultado_ajustado <- predecir_inyeccion_sql_ajustado(
    input_query, umbral, vectorizer, svm_model)
Imprimir(resultado_ajustado)

```

Este script comienza cargando el modelo SVM y el vectorizador TF-IDF previamente entrenados y guardados. La función `predict_sql_injection_adjusted` toma una consulta SQL como entrada. Utiliza el vectorizador para transformar esta consulta en un formato adecuado para el modelo. Posteriormente, el modelo evalúa la consulta. Basándose en un umbral predefinido, determina si la consulta es una inyección SQL o una consulta legítima.

El uso de un umbral ajustado permite una mayor flexibilidad y precisión en la detección. Esto se adapta a diferentes niveles de sensibilidad según los requisitos de seguridad. Este enfoque mejora la capacidad del modelo para distinguir entre consultas legítimas y maliciosas. Esto es esencial para la implementación efectiva de un WAF (Web Application Firewall) en entornos de producción.

## 8 Ejecución del Script de Verificación de Inyección SQL

Para utilizar el script desarrollado en este proyecto, que verifica si una consulta SQL es una inyección maliciosa o una consulta legítima, se debe seguir el siguiente procedimiento en la línea de comandos.

### 8.1 Comando de Ejecución

El script se ejecuta desde la línea de comandos utilizando Python 3. La sintaxis para ejecutar el script es la siguiente:

```
python3 script.py "<entrada>"
```

Donde "<entrada>" debe ser reemplazado por la consulta SQL que se desea evaluar. Es importante asegurarse de que la consulta esté entre comillas para que se interprete como un argumento único en la línea de comandos.

### 8.2 Ejemplo de Uso

Por ejemplo, para evaluar la consulta SQL "SELECT \* FROM usuarios", el comando sería:

```
python3 script.py "SELECT * FROM usuarios"
```

Este comando invocará el script, que procesará la consulta utilizando el modelo SVM y el vectorizador TF-IDF previamente entrenados, y devolverá una predicción indicando si la consulta es una inyección SQL o una consulta legítima.

### 8.3 Salida del Script

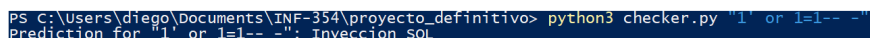
La salida del script será una de las siguientes:

- "Inyección SQL": Indica que la consulta es potencialmente una inyección SQL y, por lo tanto, maliciosa.
- "Consulta SQL legítima": Significa que la consulta parece ser segura y no contiene indicios de ser una inyección SQL.

Esta funcionalidad permite una rápida y eficiente evaluación de consultas SQL, facilitando la detección de posibles inyecciones SQL en aplicaciones web.

## 8.4 Aplicación Práctica

El script puede ser fácilmente integrado en aplicaciones web y frameworks como Flask, Django, Laravel, entre otros, proporcionando una capa adicional de seguridad contra ataques de inyección SQL. Su implementación no solo mejora la seguridad de la aplicación web, sino que también ofrece una solución eficiente y de bajo costo para proteger datos sensibles y mantener la integridad de los sistemas de bases de datos. Se puede apreciar en la siguiente captura de pantalla el software en ejecución:



```
PS C:\Users\diego\Documents\INF-354\proyecto_definitivo> python3 checker.py "1' or 1=1-- -"
Prediction for "1' or 1=1-- -": Inyeccion SQL
```

Figure 1: Ejecucion.

## 9 Conclusiones

Este trabajo presentó el desarrollo de *WAF SQLi*, un módulo innovador para la protección contra inyecciones SQL en aplicaciones web. Utilizando un enfoque basado en el procesamiento de lenguaje natural y aprendizaje automático, específicamente a través del uso de *TfidfVectorizer* y *SVC*, el modelo demostró ser eficaz en la identificación y clasificación de intentos de inyección SQL.

Los resultados obtenidos indican una alta precisión en la detección de inyecciones SQL, lo que resalta la viabilidad del modelo en entornos de producción. La implementación de un umbral ajustado mejoró significativamente la precisión del modelo, permitiendo una detección más precisa y reduciendo los falsos positivos.

A pesar de su eficacia, se identificaron limitaciones, especialmente en términos de adaptabilidad a diferentes tipos de ataques y optimización para entornos de alta demanda. Estas áreas presentan oportunidades para investigaciones futuras, con el objetivo de mejorar aún más la eficiencia y la versatilidad del modelo.

En conclusión, *WAF SQLi* representa un avance significativo en la seguridad de las aplicaciones web, ofreciendo una solución robusta y adaptable para la protección contra una de las vulnerabilidades más comunes y peligrosas en el desarrollo web. Su implementación podría ser un paso crucial hacia la mejora de la seguridad en línea y la protección de datos sensibles.

## References

- [1] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

- [2] Corinna Cortes and Vladimir Vapnik. *Support-Vector Networks*, volume 20. Kluwer Academic Publishers, 1995.
- [3] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. *European Conference on Machine Learning*, pages 137–142, 1998.
- [4] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., 1997.
- [5] Gerard Salton and Chris Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.