SQL INJECTION PREVENTION

By: Safiye Betül Kekilli

Alize Serra Acar

Advisor: Dr. Shadid Alam

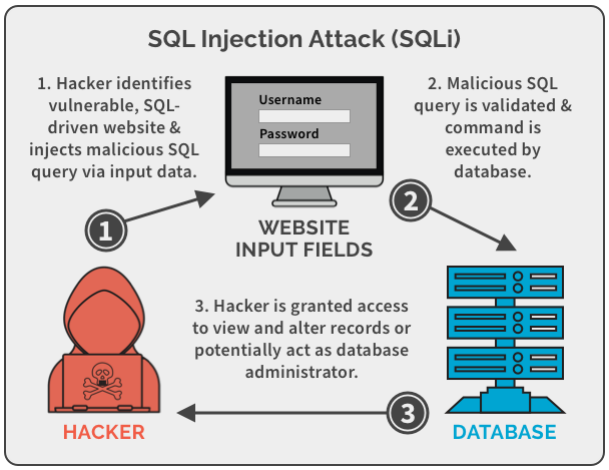
Department of Computer Science

Alparslan Türkeş Bilim ve Teknoloji Üniversitesi

TABLE OF CONTENTS

1.1 What is an SQL Injection(SQLi)?

SQL injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior



1.2 What is the impacts of SQLi?

- Sensitive data
- Passwords
- Credit card details

- Personal user information

1.3 PROJECTS REQUIREMENTS

- Implement a date range limit that ensures data is returned from a narrow date/time range
- Limit row counts processed or returned.This prevents returning too much data.
- Prevent blank searches
- Use one of SQLi prevention types

1.4Breaches Enabled by SQL Injection

- **GhostShell attack**—hackers from APT group Team GhostShell targeted 53 universities using SQL injection, stole and published 36,000 personal records belonging to students, faculty, and staff.
- **Turkish government**—another APT group, RedHack collective, used SQL injection to breach the Turkish government website and erase debt to government agencies.
- **Eleven breach**—a team of attackers used SQL injection to penetrate corporate systems at several companies, primarily the 7-Eleven retail chain, stealing 130 million credit card numbers.
- **HBGary breach**—hackers related to the Anonymous activist group used SQL Injection to take down the IT security company's website. The attack was a response to HBGary CEO publicizing that he had names of Anonymous organization members.

1.5 SQL INJECTION TYPES

- Retrieving hidden data, where you can modify an SQL query to return additional results.
- Subverting application logic, where you can change a query to interfere with the application's logic.
- UNION attacks, where you can retrieve data from different database tables.
- Examining the database, where you can extract information about the version and structure of the database.
- Blind SQL injection, where the results of a query you control are not returned in the application's responses.

1.5.1. Retrieving hidden data

An application that displays products in different categories. When user clicks the category, their browser requests the URL:

https://insecure-website.com/products?category=category

This causes the application to make an SQL query to retrieve details of the relevant products from database:

SELECT * FROM products WHERE category = 'Category' AND released = 1 This SQL query asks the database to return

- all details (*)
- from the products table
- where the category is Gifts
- and released is 1.safd

The restriction released = 1 is being used to hide products that are not released. For unreleased products, presumably released = 0. The application doesn't implement any defenses against SQL injection attacks, so an attacker can construct an attack like:

https://insecure-website.com/products?category=category'-- This results in the SQL query:

The modified query will return all items where either the category is category, or 1 is equal to

1. Since 1=1 is always true, the query will return all items.

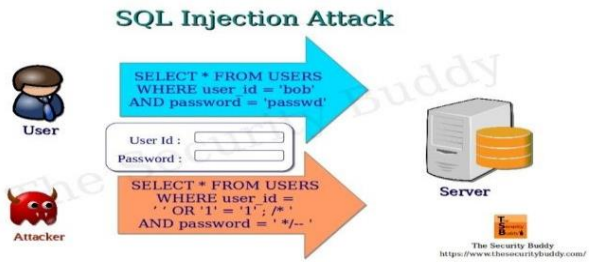
1.5.2. Subverting application logic

Consider an application that lets users log in with a username and password. If a user submits the username wiener and the password bluecheese, the application checks the credentials by performing the following SQL query:

SELECT * FROM users WHERE username = 'wiener' AND password = 'bluecheese'

If the query returns the details of a user, then the login is successful. Otherwise, it is rejected.Here, an attacker can log in as any user without a password simply by using the SQL comment sequence -- to remove the password check from the

WHERE clause of the query. For example, submitting the username administrator'-- and a blank password results in the following query:This query returns the user whose username is administrator and successfully logs the attacker in as that user.



1.5.3 Union attacks

When an application is vulnerable to SQL injection and the results of the query are returned within the application's responses, the UNION keyword can be used to retrieve data from other tables within the database. This results in an SQL injection UNION attack.The UNION keyword lets you execute one or more additional SELECT queries and append the results to the original query. For example:

SELECT a, b FROM table1 UNION SELECT c, d FROM table2

This SQL query will return a single result set with two columns, containing values from columns a and b in table1 and columns c and d in table2.

For a UNION query to work, two key requirements must be met:

- The individual queries must return the same number of columns.
- The data types in each column must be compatible between the individual queries.

The UNION attacks most important requirements are:

- How many columns are being returned from the original query?
- Which columns returned from the original query are of a suitable data type to hold the results from the injected query?

1.5.4 Blind SQL Attacks

in this attack, no error messages are received from the database; We extract the data by submitting queries to the database. Blind SQL injections can be divided into boolean-based SQL Injection and time-based SQL Injection

SQLi attacks can also be classified by the method they use to inject data:

- SQLi based on user input
- SQLi based on cookies
- SQLi based on HTTP headers
- Second order SQL injection

1.6 SO HOW CAN WE PREVENT SQL INJECTİON?

1.4.1 Prepared Statementi
1.4.2 Validate User Inputs
1.4.3 Enforce Least Privilige
1.4.4 Penetration Testing
1.4.5 Use a WAF

1.6.1 Prepared Statements with Parameterized Queries

Prepared statements guarantee that user inputs passed to SQL statements are safe. The idea is that a user's input cannot directly influence the SQL commands that control and access an SQL database. The database will treat all inputs (including malicious SQL statements) as ordinary inputted data and not as commands.

In a prepared statement, the variables in a query are always separated from the rest of the query. In other words, when a developer defines the code for SQL queries (prepared statements) the user input and code are separated.

1.6.2 Validate User Inputs

Another way to defend a website's front line against SQL injections is to validate user inputs. With this method, whatever a user inputs to the web application, there is always a validation (filtering) process allowing/disallowing inputs.

To validate user inputs, identify and define the essential SQL statements. **Establish an SQL statement whitelist that helps filter out all unvalidated statements and accepts valid inputs.**

1.6.3 Enforce Least Privilege

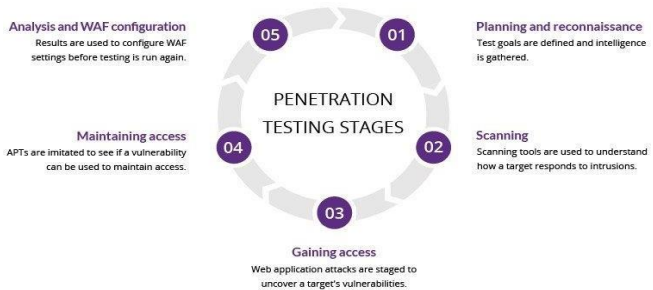
As a second line of defense, the level of access and user's privileges to a database should be limited. It is important to restrict access rights to the specific users, accounts, or computer processes that need to perform a specific activity on the database. For example, if a website or application only needs to use SELECT statements for the database, there should not be any user privileges to use INSERT or DELETE.

1.6.4 Perform Penetration Testing

It is SQL injection is to know the website's flaws and to perform regular assessments of the website's defenses by auditing suspicious activities, user privileges, and using pen-testing tools.

SQL injection hackers and even script kiddies are beginning to use automation for time- consuming and repetitive tasks. Examples are Havij, an automated SQL Injection tool, that helps penetration testers (or bad actors) to find and exploit SQL Injection vulnerabilities.

Another is SQLMap, an open-source penetration tool used to automatically scan, detect, and perform SQL injections and take control of databases.

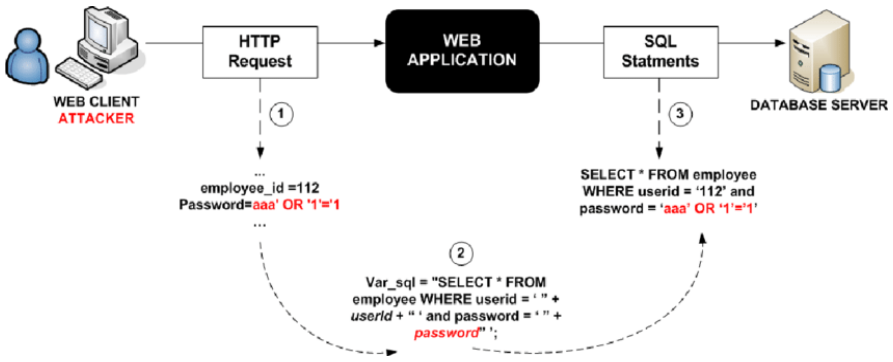


1.6.5 USE A WAF

A web application firewall, or WAF, is a security tool for monitoring, filtering and blocking incoming and outgoing data packets from a web application or website. WAFs can be host- based, network-based or cloud-based and are typically deployed through reverse proxies and placed in front of an application or website (or multiple apps and sites).

1.6.6 Dataset and Testing

we are going to use different machine learning algorithms to detect SQL Injection attacks. The dataset used for this research consisted of both vulnerable and safe SQL code. The datasets considered as input must be preprocessed and a set of features must be extracted from this dataset through a process called feature extraction.



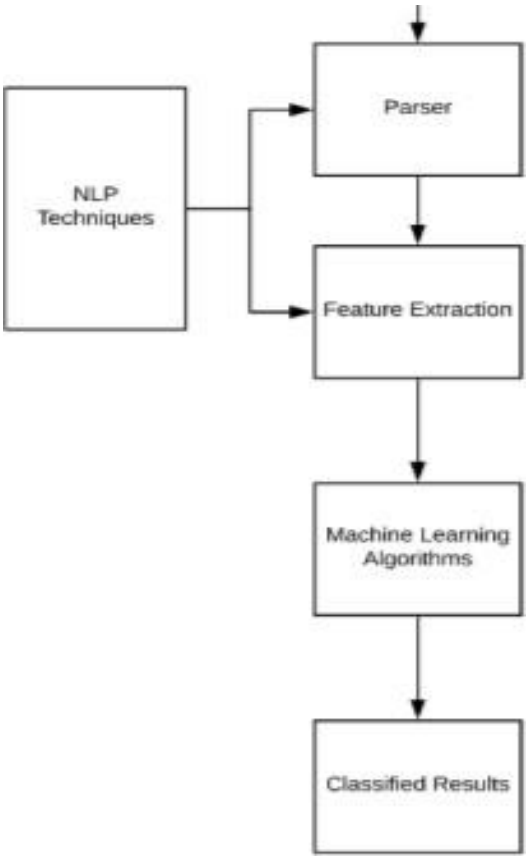
Dataset

we are going to use different machine learning algorithms to detect SQL Injection attacks. The datasets considered as input must be preprocessed and a set of features must be extracted from this dataset through a process called feature extraction. Our dataset includes 30920 querys and this

querys 30873 times are unique. We'll clean the dataset at first we'll clean our dataset. After that

we'll seperate the data test and train dataset. In our datasets one primary key for detect, it is label 0 has no injection and label 1 has injection for our feature dataset

Testing Set: Here, once the model is obtained, you can predict using the model obtained on the training set.



Proposed model to Detect SQL Injection Attacks using Machine Learning Algorithms

2.0 Technologies

We choose machine learning method for detecting. We'll use for this Python with Spyder and Weka tool for machine learning. In Python we should get some libraries, these are

- sklearn,
- numpy,
- panda
- nlp libraries.

2.1 Proposed Model

In this section, we want a unique model using machine learning to detect SQLi attacks. Instead of tackling the injection on the injection attacks on the server side by guarding the database, this model is designed to act as a filter on the client side. As a result, most of our data are text-based.

2.1.1 Data Prepartion

For the purpose of reducing data noise and improving precision, unnecessary spaces and escape sequences will eliminate and all the queries will convert into lowercase form.

2.1.2 Training and Test Dataset

The training and testing sets are taken randomly from the dataset with a conventional ratio of 80-20 (80% for training and 20% for testing) usingtrain_test_splitfunction built into sklearn library:
train_test_split(trainDF['text'], trainDF['label'], test_size = 0.2)

We could change the test_size . For example, if test_size =0.4 %60 for training and %40 for testing.

2.1.3 Parsing

Our model cpme across a common adversity during the data processing phase where traditional machine learning model explicitly takes in structured tabular numeric data, but our collected data are entirely non-structured texts. This is where parser for text comes into play. Text parsing is the process of transforming given series of text into desired smaller components based on some specific The latter divides the text into tokens, where each token can be a character, a word or a phrase. In the case of SQLi attacks, the regular expressions do not determine the malice of a query, the appropriate text parsing method for this model is tokenization. Queries are split into tokens of words. For example: Parsing “or 1=1 -- 1” into “or”, “1=1”, “--”, “1”

2.1.4 Natural Language Processing

(NLP) techniques and feature extraction For NLP, there are many featured engineering techniques, but the one that proves to be the most useful for this SQLi attacks detecting model is Word Level TF-IDF Vectors. TF-IDF stands for Term Frequency and Inverse Document Frequency, which is an important index for term searching and figuring out the relevancy of specific terms in a document.

Term Frequency specifically measures how often a word occurs in a document, where Document Frequency determines how often a word occurs in an entire set of documents. The formula to calculate the relevancy of a specific word is as follows:

- Term Frequency
- Document Frequency

OR

- Term Frequency
- Inverse Document Frequency

2.1.5 Machine Learning Algorithms

- Logistic Regression
- Random Forest
- Support Vector Machines
- Naive bayes
- Decision Tree

2.2 Metrics

The foremost possibility is then assess with the test dataset, and the factors of the algorithms that we were used to compare. This result is obtained from calculating True Positives (T P), True Negatives (T N), False Positives (F P), and False Negatives (F N). From this values, the system calculates accuracy, precision, and recall at the last of the valuation procedure. This may be represented in a matrix form and known as confusion matrix. By checking the FN and FP values we decide which parameter is used to evaluate.

2.3 Result of Theorem

We will use this algorithms and compare their accuracy values. We will choose the maximal accuracy value for reliable result and we will detect the injection with that algorithm. For the algorithm if the label <0.5 the dataset has no injection but it is higher than 0.5 it has injection. So we'll decide it with machine learning and nlp method for find the injection.

3. IMPLEMENTATION

In our dataset we have 4201 lines. These lines have 2 columns; a sentence column or label column. In Sentence Coloumn we have different type of SQL attacks: How? For example:

- A SQL Query
- A Zip File
- A pdf file
- A username; password; website kind an attack
- Or senteces

In Label coloumn we have 0 or 1 values.

If LABEL==0:

This sentence has no SQL Injection Attack

Else LABEL == 1:

This sentence has SQL Injection Attack.

```
import glob
import time
import pandas as pd
import keras
from keras.models import load_model
import pickle
import tensorflow as tf
from nltk import ngrams
from nltk.tokenize import sent_tokenize
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
from nltk.stem import PorterStemmer
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
```

```
from sklearn.metrics import recall_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn import tree
from sklearn.metrics import f1_score
from keras.models import Sequential
from keras import layers
from keras.preprocessing.text import Tokenizer
from keras.wrappers.scikit_learn import KerasClassifier
```

These are the libraries that we used.

Feature selection is used to reduce the dimensionality of data for performance reasons. This is a technique to remove redundant features, which are providing duplicate information of another feature, and irrelevant features, which are providing little or no useful information to help with normalization. We used an information gain technique for feature selection and are using Python, Batch Normalization with a genetic search algorithm, both of which are discussed below.

3.1 SYSTEM DESIGN

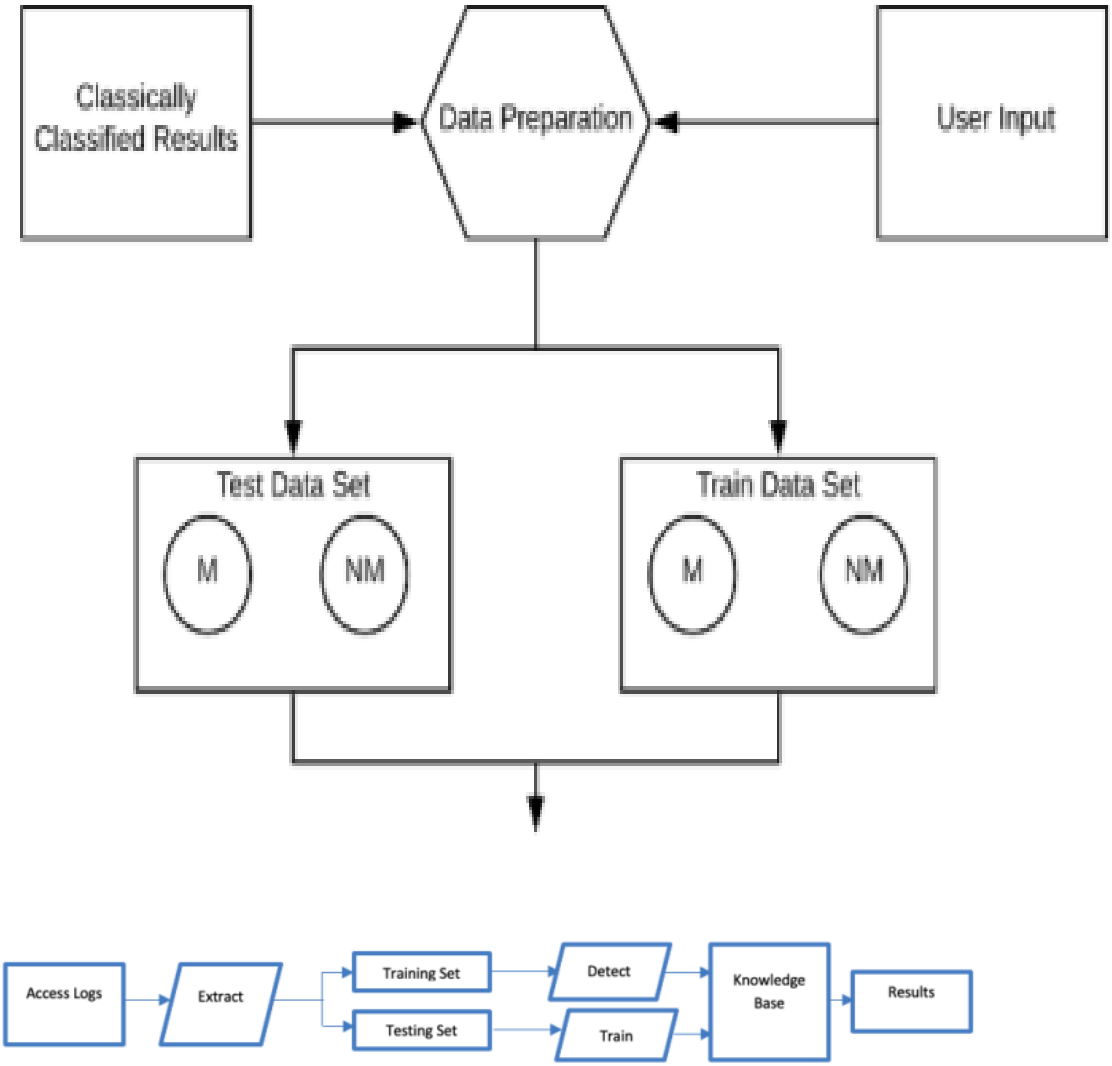
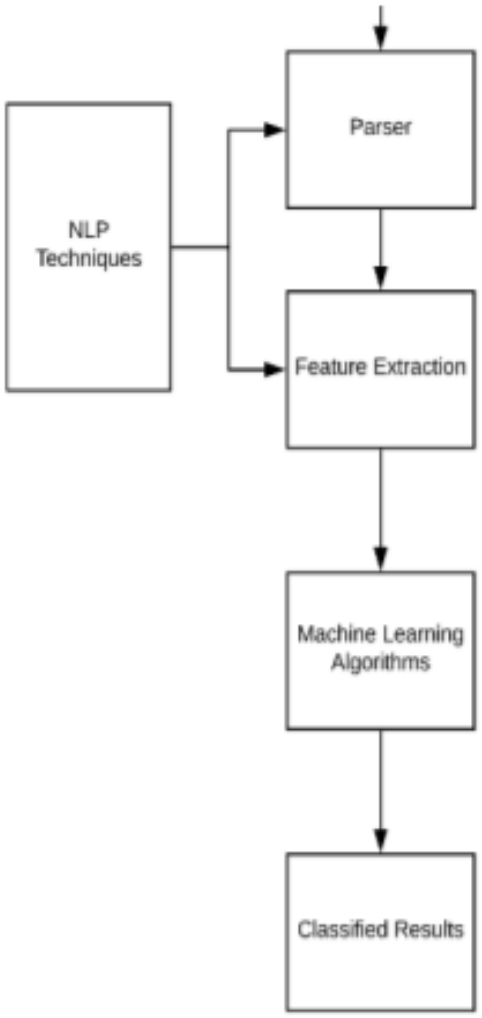


Figure 8: Architecture of SQL Detection system



3.2 HOW THE CLASSIFICATION WORKS IN DATA?

Parameter For Classification	Categories
Attack Sources	User input Cookies Server variables Second order injection
Attack Goals	Database finger printing Analysing schema Extracting data Amending data Executing dos Equivocating detection Bypassing authentication Remote control Privilege intensification
Attack Types	Tautology Illegal/logically incorrect queries Union query Piggyback query Stored procedure Inference Alternate encoding

4. HOW THE CODE DOES WORK?

We wrote the code: in python, Pycharm IDE. For attacks, we have a data file from first meeting, as we discuss. First we read our sqli.csv data file and preprocessed it. We preprocessed it with CountVectorizer method and stop words library and method. In preprocessing; we are converting the sentence coloumn to arrays. After the preprocessing; we have 4717 features.

	0	1	2	3	4	5	...	4711	4712	4713	4714	4715	4716
0	0	0	0	0	0	0	...	0	0	0	0	0	0
1	0	0	0	0	0	0	...	0	0	0	0	0	0
2	0	0	0	0	0	0	...	0	0	0	0	0	0
3	0	0	0	0	0	0	...	0	0	0	0	0	0
4	0	0	0	0	0	0	...	0	0	0	0	0	0

This is our metrices. We have train and test split: We are using train_test_split function to seperating. We can seperate it depending on weights. For example in our code

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Our test size is equal to 0.2 so this gives a value of 1/5.

After that, our preprocessing continues with batch normalization. What is batch normalization? Batch normalization is a way of accelerating training and many studies have found it to be important to use to obtain state-of-the-art results on benchmark problems.

$$\widehat{h}_j \leftarrow \gamma_j \frac{h_j - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}} + \beta_j.$$

Figure: Batch Normalization Formula

After the batch normalization our new values:

```
Total params: 110,955
Trainable params: 108,907
Non-trainable params: 2,048
```

4.1 EVALUTION METRICS

After the normalization process we have different algorithms for detecting. And we compared them with each others. We used for the comparing a confusion matrix, accuracy, precision, recall and the F1 values.

Figure: Confusion Matrix

	Predicted as Normal Request	Predicted as Malicious Request
Normal Request	TN	FN
Malicious Request	FP	TP

FORMULAS

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

After that we are predicting with 5 different algorithms into the confusion matrix. Then we are deciding the which algorithm is the most consistent? Our Testing Values are like here:

4.2 TESTING VALUES

Algorithm	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.9285714285714286	0.90	1.0	0.85
Naive Bayes	0.9773809523809524	0.92	1.0	0.95
Random Forest	0.7642857142857142	0.86	1.0	0.87
Support Vector Machines	0.9071428571428571	1.0	0.21	0.36
Decision Tree	0.875	0.70	1.0	0.96

For our values the most accuracy value has naive bayes algorithm. Why naive bayes? Naive Bayes's results is always between 0-1. Because of Our label coloumn's values are 0 or 1.

After the confusion matrix process, we are cleaning data with clean data function. This function gives a parameter of features. It cleans the special wrong and unnecessary variables in file. After that the prediction function comes. That function predicts to input sentences SQL attack or not. How it do this? It cleanes the data with recalling clean_data, after that it converts the file to a model. Our sentences in the model file. Then you input the sentence.

After all this it checks the repetation and for the sentence if the label <0.5 your sentence has no injection but it is higher than 0.5 it has an injection.

```
if repeat == True:

    if result > 0.5:
        print("ALERT!!!! SQL injection Detected")


    elif result <= 0.5:
        print("It is normal")
```

5. Result

SQL injection attacks, and web-based attacks in general, continue to be a major issue in the security of financial, health, and other critical data, and this problem only increases in importance as more societal processes become more dependent on the internet. In this project we have proposed a multi-source data analysis system for increased accuracy in detection of SQL injection attacks, and have established that the algorithms we have experimented with such as rule-based and naive bayes algorithms have in our experiments achieved accuracy close to that of Neural Networks and are much better in terms of time necessary to build models and execution time when classifying testing data. Future works include collection of additional data such as traffic outbound from the web application to the browser, collection of larger datasets to see if this improves performance, analysis of additional machine learning techniques for both accuracy and performance, and adapting this system to detect other types of web-based attack.