Binghamton University, Watson School of Engineering

Project 2:
SQL Injection Detection

Name: Rahul Verma

Science of Cyber Security – CS 559-01

Prof. Guanhua Yan

October 24, 2022

Contents

SQL INJECTION DETECTION

1. Intro to SQLI detection:

As per the first project, the technique of SQL injection is used to insert harmful data like SQL query in the input as username and password to access the server. The fundamental working principle of the SQL injection approach is that the attacker may enter a statement that will always be true, giving him complete unauthorized control over the database and enable him to change or tamper the sensitive information stored. This is a very dangerous situation since it can result in significant loss for the person or an organization.

This detection method is machine learning based. A classifier will be used in this method to filter out the numerous dangerous spam keywords. The Naïve Bayes classifier is the one utilized in this project. We simply teach the algorithm to filter out the terms that we consider to be spam that might harm the application. The web application's source code isn't required to be changed in this method. Implementation is done separately and the source code of the web application is left as it is.

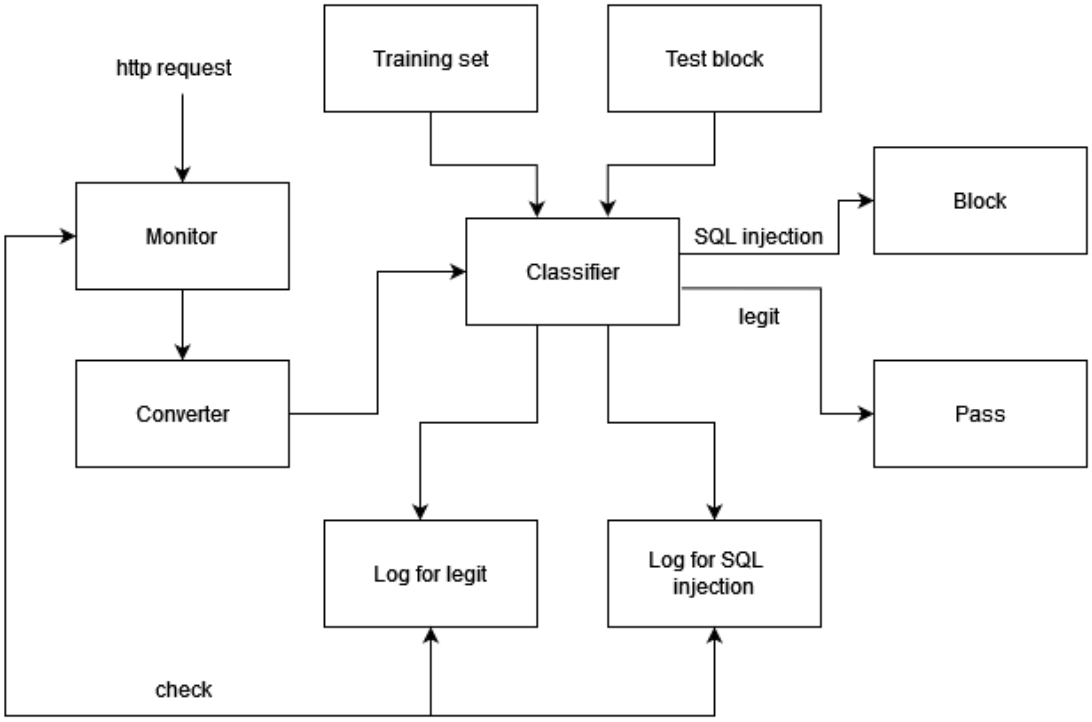
2. Type of detection method:

Detection using Naïve Bayes classifier comes under anomaly-based detection, as we are providing the training set to the classifier to categorise keywords into spam or legit category. Anomaly-based detection is basically a type of detection where the discovery of unexpected occurrences, observations, deviate considerably from the usual output, it is also known as outlier detection.

3. Naïve Bayes Classifier and its working:

The training set elements and the properties that we define here are crucial for identifying dangerous spam keywords. The monitor, classifier, and converter are the three main components utilized here. The fundamental operating principle is that we submit the HTTP request from the web app to the monitor first. It is sent to the converter via the monitor. The classifier then determines whether the data is legit or if it's an SQL injection attempt after receiving it from the converter. If it's a SQL injection attempt, it's detected, banned and access is denied or else the entry is allowed.

The classifier, which has a preset training set, determines whether the input is legit or if it's an SQL injection attempt. This collection includes the test samples that the developer has specified.



- **Monitor:**

The monitor is essentially the first stage that the data must go through. HTTP POST/HTTP GET methods are used to send the data. In this case, URL decoding is needed. What occurs in URL decoding is that it only permits a particular set of characters to pass when the user enters his credentials, which is significant to the server and authentication. In this instance, we ensure that all the letters are transformed to upper case after decoding. By doing this, the characters are all in the same format, making it impossible for an attacker to access the database by utilizing mixed-case characters.

- **Converter:**

HTTP parameters are the input the converter receives from the monitor. The parameters must be transformed into a numeric format so that the classifier can understand them. It is the classifier who determines if the data is legit or SQL injection. The pattern attributes and keyword set are the two primary attributes. The two most important factors in pattern characteristics seem to be length and the number of keywords. When it comes to SQL injection, the usage of special characters should be avoided since, when done correctly, they can make the query statement a true or false Boolean value, evading the security check and giving the attacker unauthorized access to all the information. Dollar signs($), single quotes(‘’), equal to signs(=), hashes(#), and other symbols are frequently used as characters. Additionally, terms like SELECT, FROM, DROP and spaces are utilized and are treated like characters, as well as there is a space requirement for every query. Consider the following example: SELECT * FROM users WHERE username = “xyz” etc. Here, there are three keywords i.e., SELECT, FROM, and WHERE and 43 characters.

- **Classifier:**

The converter provides the classifier with its input. The classifier determines whether a keyword qualifies as spam or not. The Naïve Bayes classifier is the one being utilized, where the Bayesian decision theory is used to build the classifier. Using the Bayes equation, the probability of classes is computed. The keyword with the highest value is determined to be a spam keyword based on probability.

In this case, X stands for the provided attributes. If there are 'n' patterns, then X is represented by:

X = {(X1, t1), (X2, t2) ... (Xn, tn)}

The collection of attributes for all patterns, Xn, is denoted by the symbol Xi = (x1, x2, ...)T

Ti is the set of all w’s such that: Ti = {w1, w2, ..., wm}

It is determined that attributes from the samples represent the functions' parameters. The probabilities based on the Bayes equation are calculated by this function alone and are written as follows: P(X|wi) P(wi).

- **Algorithm:**

The Naïve Bayes classifier's algorithm is as follows:

Input: X

Output: None

- Begin
- g1(X) = P(X|w1) (w1);
- g2(X) = P(X|w2) (w2);
- Set k as the maximum value between g1(X) and g2(X);
- If (wk == Injection pattern);
- Declare “SQL Injection”;
- Insert into SQL Injection Log;
- Execute Block Process;
- Else;
- Declare “Legit”;
- Insert into Legit Log;
- Execute Pass block;
- End.

Two probabilities are computed here as part of the algorithm's operation in this classifier. They are as follows: one for the SQL injection string and one for the legit string. The subsequent tasks are carried out based on the probability outcome. If there is a higher probability that the string is legit, it is run and a log entry is made in the legit log. The process is halted, the SQL injection string is executed, and logs are recorded in the SQL injection log if the probability of the SQL injection string is higher. This prevents access to the database.

4. Working of training set:

To carry out either one of the two options, the classifier must first be trained. For the algorithm to understand which strings are legit and which are not, predetermined sample sets are specified in the code. Here, we have two options: either manually enter a variety of strings that we believe to be spam, or just make a string pattern generator. For instance: String like (1’ or ‘1’ = ‘1) is a SQL injection string that an attacker can use as the username and password input to create a SQL query that will always return true. The algorithm may be trained to filter out such prohibited characters.

5. Performance of detection method:

As machine learning is based on the training set and requires some time to learn from the real-world inputs, it is impossible to get accurate results all the time, there is a chance for it to get false alarms, but as time progresses, we can make the system more reliable by letting it collect the input information from the usage and keep increasing its accuracy. As for the early usage of this system, it will show accurate detection results most of the time. After entering 5 different SQL injection queries, the classifier detected all of them.

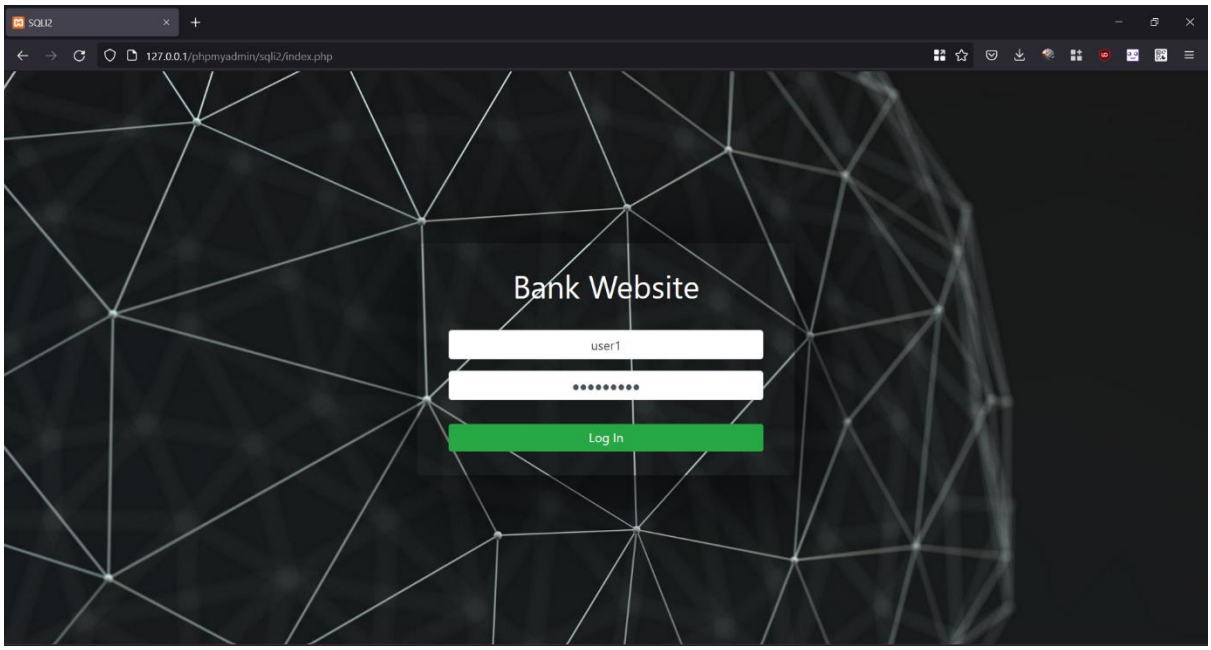
6. Screenshots:

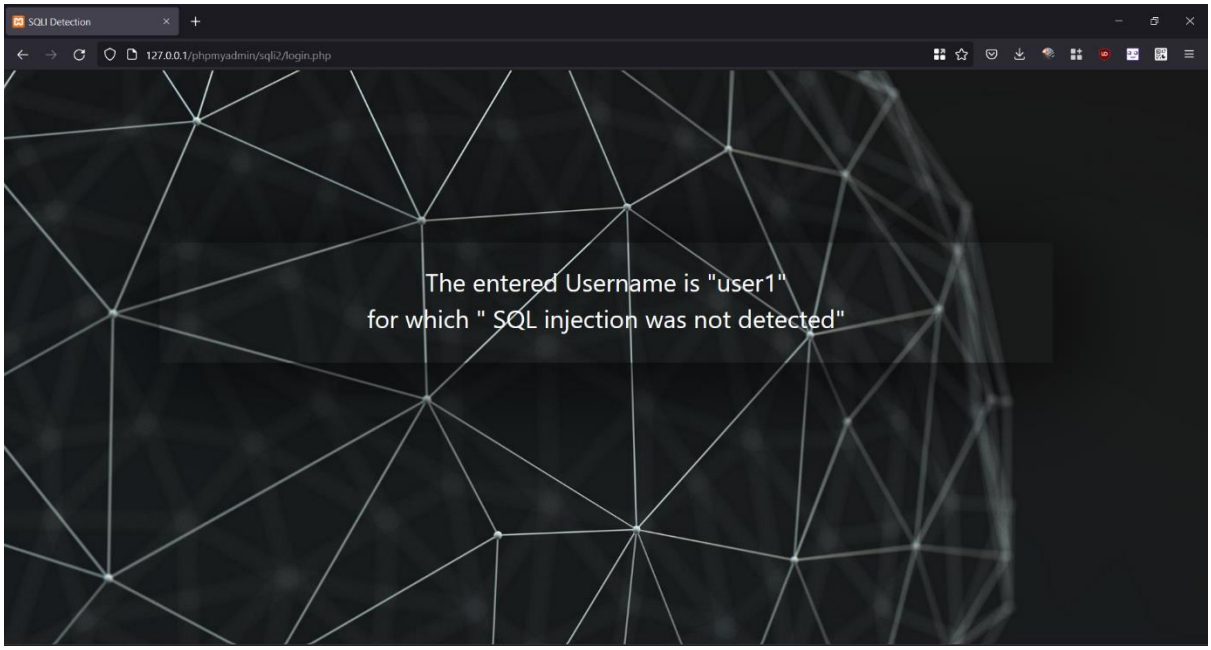
```
$classifier->train("$$, dollars", $spam);
$classifier->train("The possibility of it being an injection query increases with the length of the string", $spam);
$classifier->train("105; DROP TABLE Users", $spam);
$classifier->train("this is a spam username and password", $spam);
$classifier->train("drop * <>", $spam);
$classifier->train("drop * from database", $spam);
$classifier->train("drop * from database", $spam);
$classifier->train("select * from database", $spam);
$classifier->train("1 OR 1=1 and drop", $spam);
$classifier->train("1 or 1 = 1", $spam);
$classifier->train("1 or 1 = 1 and user <> rahul", $spam);
$classifier->train("select * FROM db", $spam);
$classifier->train("105; DROP TABLE Users", $spam);
$classifier->train("delete * from database", $spam);
$classifier->train("105; INSERT into TABLE Users", $spam);
$classifier->train("1 or 1 = 1 and user <> john", $spam);
```

Training the classifier for spam keywords

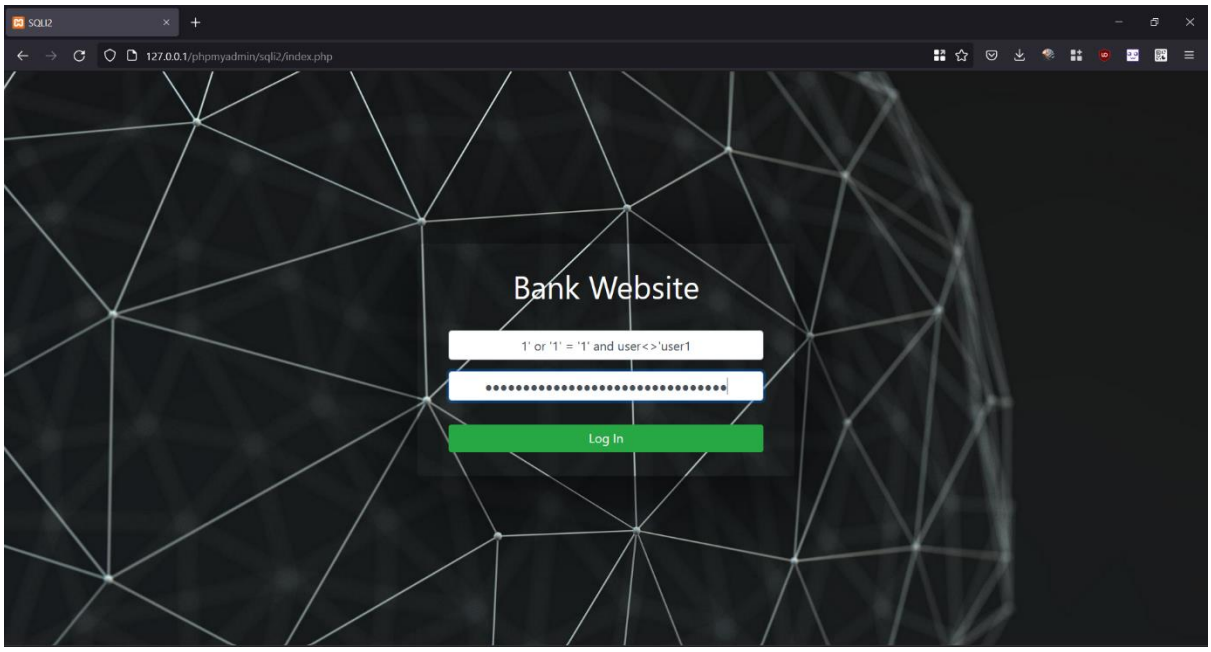
```
$classifier->train("rahul", $ham);
$classifier->train("john", $ham);
$classifier->train("rverma4", $ham);
$classifier->train("erenjaeger", $ham);
$classifier->train("killua@hxh", $ham);
$classifier->train("natusvincere", $ham);
$classifier->train("milesmorales", $ham);
$classifier->train("teamliquid.fc", $ham);
$classifier->train("johanliebert", $ham);
$classifier->train("Gojo666", $ham);
$classifier->train("Gon@hxh", $ham);
$classifier->train("Killua999", $ham);
$classifier->train("Goku111", $ham);
$classifier->train("freiza222", $ham);
$classifier->train("astralis.fc", $ham);
$classifier->train("hinatashoyo", $ham);
$classifier->train("kennys@cs", $ham);
$classifier->train("chainsawman", $ham);
$classifier->train("hyperbeast", $ham);
$classifier->train("doritojoe", $ham);
```

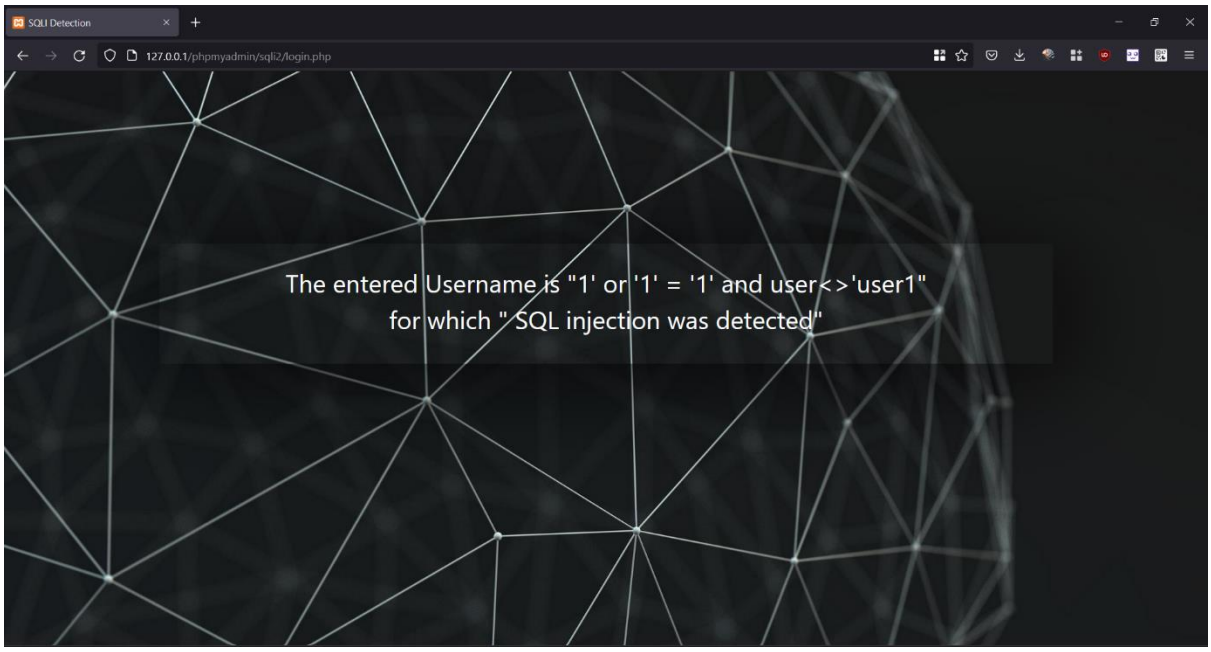
Training the classifier for legit keywords





Because it is a short string without any special characters and its probability is determined by the database count, the username “user1” that was entered above is secure from SQL injection. So, SQL injection was not detected.





The username “ 1’ or ‘1’ = ‘1’ and user<>‘user1 ” is an attempt at SQL injection with special characters that was used as an input to test the probability calculation of the classifier. Although it is not explicitly defined in the training set in this case, the classifier classifies it as a vulnerable string. So, SQL injection is detected here.

7. Bibliography:

- https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c
- https://www.geeksforgeeks.org/naive-bayes-classifiers/
- https://en.wikipedia.org/wiki/Naive_Bayes_classifier
- https://www.sqlshack.com/sql-injection-detection-and-prevention/