Novel Approach to Detect SQL Injection Attacks

MSc Research Project

Chirag Chaudhary

x19213808

School of Computing

National College of Ireland

Supervisor: Prof. Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Chirag Chaudhary		
Student ID:	x19213808		
Programme:	MSc in Cyber Security	**Year:**	2020-21
Module:	MSc Research Project/Internship		
Supervisor:	Prof. Vikas Sahni		
Submission Due Date:	06/09/2021		
Project Title:	Novel Approach to Detect SQL Injection Attacks		
Word Count:	6625	**Page Count.**	20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

Signature: CHIRAG CHAUDHARY

Date: 01/09/2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	◻
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	◻
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	◻

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Novel Approach to Detect SQL Injection Attacks

Chirag Chaudhary
X19213808

Abstract

The concerns for the cyber security threats have increased drastically since the increase in the use of the online services which are web-based applications. To increase the customer base most of the organizations are providing with more and better services through the online platform and giving them access to their web applications. And since the online applications stores sensitive and personal data of the users, if any malicious individual are able to attain unauthorized access they can cause serious harm. These web applications use databases to store the data which can be operated by the use of the SQL language commands. The attackers use this language to gain access and change, delete or steal the data from the database. In this research machine learning algorithms are used to find a mechanism that can detect the SQL injection attacks. Throughout the research the experiments are performed using three major classification models which are – Gradient Boosting algorithm, Random Forests and Support Vector Regression. Further, critical analysis and comparison of each algorithm is implemented to determine the most optimal model that can be used to build the SQL Injection Detection System.

1. INTRODUCTION

All the online services we use on a daily basis are web-based applications. Most of the organizations attract its customer base through online platform by providing access to their web application. This has increased their customer base exponentially. On the other hand, this also gives window for the cyber security challenges. The security threat is a concern for SMEs and also for big organizations. All the online application we access stores our personal and sensitive data. Any unauthorized access to the data can cause harm at higher level.

The data we provide to the web applications are stored in a database. The operations and actions performed on this database can only be performed using Structured Query Language (SQL). It is the most common and user-friendly language used in any organization. With the ease, there are threats and vulnerability that comes along with this language. Attackers use this language to get the unauthorized access to the database to get the data access, modify data and even delete the whole data. This type of attack is known as SQL Injection attack. Many researches have been performed to mitigate this attack but unfortunately attackers were able to defend these techniques. This study propose to find the defensive mechanism against SQL Injection attack using machine learning algorithm.

In this study, Machine Learning algorithms are used to build the SQL Injection Detection system. Before finalizing this algorithm, multiple experiments are performed using Support Vector Machine, Random Forests and gradient Boosting classification model. Later, comparison have been made to determine the accuracy of all the three models.

The study begins with the in-depth research on previous methods used by the researchers to detect SQL Injection attack, scope of machine learning, SQL Injection attack, its type. In section 3 author have described the research method which include the solution architecture and workflow of the solution used. Further in section 4, definition of the design specification is explained where the author have explained each step involved in the implementation phase. Furthermore in section 5, implementation phase of the model is explained in detail that includes the dataset used and the implementation of all the three algorithm used including,

1. Support Vector Machine
2. Random Forests
3. Gradient Boosting

In section 6 which is evaluation phase, all the experiments are performed in this study including experiment 1, 2 and 3. Finally the last section includes the Conclusion and future work where the author has discussed the conclusion of the study and the future work that can be performed to improve this research.

2. LITERATURE REVIEW

SQL Injection acquire first rank in OWASP Top Ten (Inc., 2021). It is an attack where user can insert SQL type commands and can exploit the database of the web application. With the increase in the digitalization, this type of attack is increasing exponentially. As per the experts, it is the most serious vulnerability that has the capability to exploit the data confidentiality and the attacker can get the complete access of database. This can lead to fraud, data theft, loss of confidential information and personal data, and violation of data protection laws. In the following sections the author has researched about the SQL Injection, mechanism of SQL Injection, types of SQL Injection, the traditional approaches used to identify SQL Injection, Machine Learning, and various ML algorithms used by researchers in past studies.

2.1.SQL Injection

According to the study by Gartner Group over 300 internet sites, the web applications that are vulnerable to the SQL Injection Attack (SQLIA) are widespread (William G.J. Halfond, Jeremy Viegas, and Alessandro Orso, n.d.). There are well known attacks that occurred in the past on well-known and popular web sites. Most recently in early 2021, SQL injection attack happened on Gab website that resulted in the exfiltration of around 70 GB of data followed by the second attack in the next week. In 2015 year, the same attack was performed on one of the British telecommunications company TalkTalk's servers that resulted in the stealing of personal information of around 156,959 customers. Similarly, Barracuda Networks security was compromised in 2011 year and this attack led to the getting access to the username and email addresses of all the employees (Wikipedia, 2021).

To get the better understanding of the SQL injection attack. In the following sections, research about what is SQL Injection, its mechanism, types of SQL injection attacks and their consequences is performed.

SQL injection falls under the category of code-injection wherein the user while entering the data into the web application inserts the malicious code along with the data. This user data that includes malicious code is known as SQL Injection attack.

2.2.SQL Injection Mechanism

In the research paper (Swarnaprabha Patil, Prof. Nitin Agrawal, 2015) author discussed the various mechanisms that can be used to inject SQL Injection attack. Commonly known mechanisms are Injection through user input, Using Cookies for injection, injecting while using server variables, and second order injection.

Injection through user input: It is a type of an input mechanism where attacker uses commands or code and inserts that code using the form submission. Generally, the web application offers the feature of submitting form application. After user enters the details, these forms are then sent via HTTP using GET or POST requests to web application (Milic, 2019). This gives the attacker a window to enter malicious code. These codes are so accurate that can break the security of the web application.

Using cookies for Injection: This is the method that is used by the malicious client attackers. All the web application generates a file that stores all the information about the logs and activities performed by the application user (Dornseif, M., 2005). This file is known as Cookies. Wherever the cookie file used in order to produce the SQL queries, it file can be misused by the clients by corrupting the substance of the file. Attacker can insert a malicious code in the cookie file that further has the capability to perform SQL injection attack.

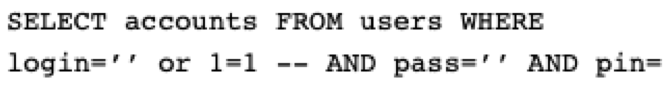
Injection while using server variables: There are certain characters such as HTTP, variables, network headers. When these characters are grouped together, they are known as server variables. In order to perform activities such as logging and identifying the length of browsing, Web application uses server variables. But these variables are required to be handled with appropriate measure as they can be used by the attacker to perform SQL injection. In any instance, if these variables are inserted into database without refining, the attacker can replace the original values of HTTP header with the malicious code and thus they can make the web application vulnerable to the SQL Injection attack (Principal, 2018).

Second-order injection: In this type of method, attacker initiates the vulnerable code in the database by triggering and enforcing SQLIA (Anley., 2002). Second-order injection uses a technique when the attacker has the knowledge of the input, when and where it will be used. Accordingly, the attacker defines its attacking technique.

2.3.Types of SQL Injection Attack

There are numerous studies that have been done in order to specify the types of SQL Injection attacks. In the study (Rubidha Devi.D*, 2R.Venkatesan, 3Raghuraman.K , Dec-2016), the authors have described the types of SQL Injection attacks into multiple categories. The study states that there are attacks that can be done independently and few attacks are injected in combination. The categories defined by them are,

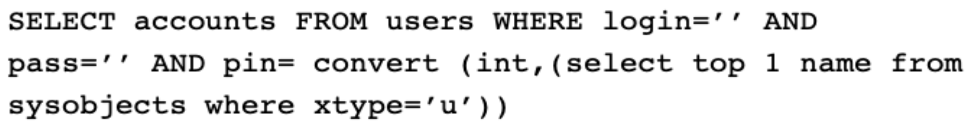
Tautology based attack: In this attack, the malicious code is injected in conditional statements. The objective is to enforce the conditional statement outcome to be true. The intent of this attack is to bypass authentication, to extract the data from the database, and to get the details of parameters that can be used for performing injection. Example query of Tautology attack is shown in the below figure.



```
SELECT accounts FROM users WHERE
login='' or 1=1 -- AND pass='' AND pin=
```

Figure 1: Tautology based attack

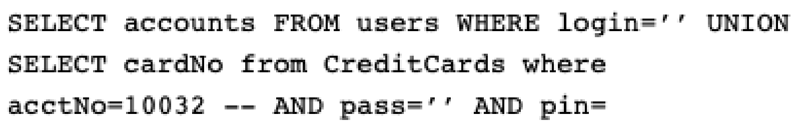
Incorrect Queries: In this type of attack, the information of the database is changed by the attacker. Here, attacker focuses on getting access to the sensitive and critical information of database including its structure, datatype and more. Example of the query of Incorrect Queries attack type is given in the figure below.



```
SELECT accounts FROM users WHERE login='' AND
pass='' AND pin= convert (int,(select top 1 name from
sysobjects where xtype='u'))
```

Figure 2:Incorrect queries attack

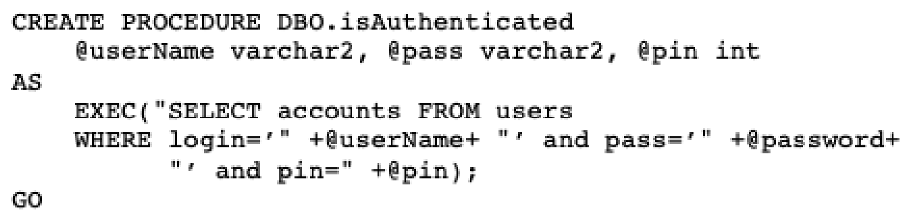
Union Queries: In this type of attack, attacker targets the database queries. Any SQL query performed by the developer is changed by the attacker by adding its own query with the developer query. This results into getting the output which attacker wants in addition to the output expected by the developer. Generally, this type of attack is implemented by injecting code in the web application in a certain form of statements such as, UNION, SELECT or more. Examples of the query of Union queries attack type is shown below.



```
SELECT accounts FROM users WHERE login='' UNION
SELECT cardNo from CreditCards where
acctNo=10032 -- AND pass='' AND pin=
```

Figure 3: Union Queries attack

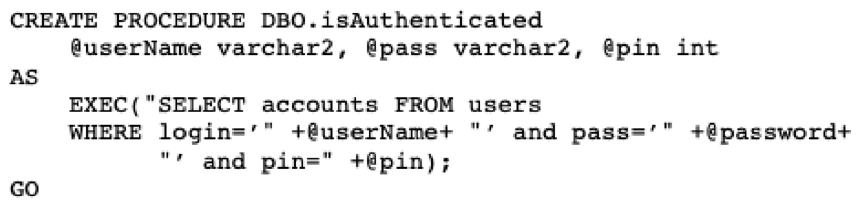
Stored Procedures: Here author explains how stored procedures can be used by an attacker for SQLIA. According to this study (Rubidha Devi.D*, 2R.Venkatesan, 3Raghuraman.K , Dec-2016), Stored procedures are those features of database that are provided by the database owners to improve the database functionality. There procedures are used to perform interaction between operating system and database. If incase, attacker gets to know the type of database used in the web application backend, it will be easier for an attacker to perform SQL injection using stored procedures. Example query of such type of attack is shown below.



```
CREATE PROCEDURE DBO.isAuthenticated
    @userName varchar2, @pass varchar2, @pin int
AS
    EXEC("SELECT accounts FROM users
    WHERE login='" +@userName+ "' and pass='" +@password+
          "' and pin=" +@pin);
GO
```

Figure 4: Stored Procedure attack

Inference: This type of attack is also known as Piggy-back queries. This type of attack is different from all the attacks as in this case attacker instead of making changes in the existing queries, it adds its own query in addition to the existing the query. This results in a situation where database starts receiving more than one query subsequently. The first query generally is the original and followed by the query of attacker injected along with the original. This leads to the SQL Injection attack and makes the web application vulnerable. Example of query of this type of attack is given below.



```
CREATE PROCEDURE DBO.isAuthenticated
    @userName varchar2, @pass varchar2, @pin int
AS
    EXEC("SELECT accounts FROM users
    WHERE login='" +@userName+ "' and pass='" +@password+
          "' and pin=" +@pin);
GO
```

Figure 5: Inference attack

Alternate Encodings: This type of attack is more of prevention method rather than an attack. In this type of attack the query is written with an objective to prevent the injected code detection by the automated tools and codes. Here, attacker writes a query and attack it with the malicious code to execute the SQL injection. In this way, the attackers can eliminate the detection of the SQL attack tools.

After getting a good knowledge and understanding of SQL Injection, its mechanism, and types. The study further researched about the methods to detect the SQL Injection attacks. It came across multiple methods that are used by the researchers and were able to identify SQL Injection attacks to a certain level. In the below sections it has discussed few studies and have outlined their conclusion.

In a study (Jemal, 2020), author has discussed and proposed various SQL injection attack detection technique. In this paper, the solution of detecting the attack is defined based on the classification of the SQL injection attacks. Majorly author focused on summarizing the SQL injection attacks, their sources, types and goals. The contribution of the paper includes, summary of the SQL attack, its source, types, goals were explained. Further, detection and prevention technique of SQL injection attack based on their classes are described. SQL Injection attack were classified as Query-model based SQLI, Obfuscation based, Monitoring and Auditing based, Entropy based, Ontology based SQL attacks. Moreover, the solutions proposed are, SQLStor detection method that depends on the comparison of semantics that is by comparing the original structure of syntax tree with the structure generated by the benign query. Obfuscation/DE Obfuscation technique for detecting SQL injection attack that uses encryption method in order to prevent unauthorized access to the database. PSIQOP known as Preventing AQL Injection attack by Query optimization process. Automation testing approach to detect entropy-based SQL injection attacks.

In one of the studies (Muhammad Amirulluqman Azman, 2021), the author proposed to use Machine learning to detect SQL Injection. This study was close to our solution but vary in terms of the research approach and the result obtained. In this paper, author proposed Signature based detector and it described its solution to be distributed majorly in three phases. First phase includes Extraction of the data. For this they have used the log files of one of the web applications and a tool is used that extracts the URL. Further the log file data is converted into a set of k signatures using k-cross fold validation. The second stage includes building classifier model. Author classified the URL into two classes, benign and malicious web requests using Boyer's Moore classification algorithm. The model performs string matching function. It compares each URL feature with the set of malicious features. This solution was showing high accuracy but there was a limitation of this solution. In case of real-time data, this solution failed to address the problem.

In another study (S. O. Uwagbole, W. J. Buchanan and L. Fan,, 2017), Machine learning predictive analytics approach is used to detect SQL Injection attack. The dataset used by the author contained symbols and SQL tokens. The model built was further deployed as a web service so that model can learn in the live environment and can increase its knowledge base. The major technique used in this study are, dataset extraction, text pre-processing, Feature hashing, filter-based feature selection technique, splitting training and test data, training the model and testing the results. The drawback of this solution was that it only predicts whether the web requests contain SQL injection attack or not. When it comes to multi-classification, the model has the limitation. So, in this paper the objective is to propose am advance solution to eliminate this limitation.

Another study that proposed a solution for detecting SQL injection is (K. Kamtuo and C. Soomlek, 2017). In this paper, author built a Machine learning model to detect the SQL Injection attack. More than one model is developed in this study including Support Vector Machine, Artificial Neural Networks (ANN), Boosted Decision tree, and Decision Jungle. The author further did a comparative analysis of the results of all the four models. The efficiency of the models built by the author were evaluated using parameters such as precision, processing time and, probability of false alarm. The results shows that the logically incorrect queries were best detected by Support Vector Machine model, Union queries and piggybacked queries were best predicted by Decision jungle model. The solution proposed in this study was efficient, but the limitation was the author failed to build a model that can work for all types of vulnerabilities of SQL Injection. So, in order to eliminate this limitation, this study was conducted to build a model that is efficient for all the types of queries.

Apart from machine learning there were other tools and technologies to detect SQL injection attack. In a study (Inyong Lee, 2012), author proposed a solution that was using static and dynamic analysis of the web application. The static analysis was defined as the method where the sentences of SQL query are analyzed in order to detect and prevent SQL injection. The objective of the static method is to evaluate the user input and replace the query whenever necessary to prevent any malicious query to be entered. For example, JDBC checker uses JSA (Java String Analysis) (Inyong Lee, 2012) that dynamically checks the user input and eliminate SQL injection. Similarly, dynamic analysis is a method that analyze and evaluates the response received from the web application post scanning method. The scanning is process where we send each input to the target and the target sends a response. This response id analyzed in the dynamic analysis. This study proposes a novel approach which is a combination of both static and dynamic analysis. The method is designed in such a way that it eliminates the attribute values of all the SQL queries at real time thus performing dynamic analysis and later it performs a comparison between the static analyzed SQL queries thus, performing static analysis. The proposed method was unique, but the limitation is it cannot be implemented on a Web Application connected to the database.

In another study (Hou, 2016), the solution for detecting SQL injection attacks is proposed. The major focus of this study to detect the SQL injection in the cloud environment. The method proposed in the combination of dynamic analysis and the input filtering. Further the solution proposed in implemented in the cloud environment to detect the SQL injection attack on the cloud. The solution was implemented in three stages. First phase includes analyzing the SQL statements by collecting the SQL keywords using lexical regulation method. Second phase includes development of rule tree by analyzing the collected SQL keywords and the last stage includes detecting the SQL injection by traversing the rule tree based on the model that was developed by SQL syntax regulations.

In a study (L. Xiao, 2016), the problem of detecting SQL injection attack is resolved. The author proposed a novel approach of high precision. The solution was named as Expectation-based Detection method (L. Xiao, 2016). After doing an intense study, author says that there were many solution proposed for detecting SQL injection that predicts the past experienced attacks but the unknown attacks were not discovered and detected. This method solves a problem of detecting unknown SQL injection attacks. The expectation - based detection model was designed in two stages. The first stage is known as Expectation. In this stage the calculation of probability of SQL injection attack keywords is performed from a dataset using the formula shown below.

$$\mathrm{E(X)} = x_1P_1 + x_2P_2 + \cdots + x_nP_n$$

Figure 6: Formula for calculation Expectation value

The second stage was termed as Preparation. In this stage the model detects the SQL injection attack on the basis of the Expectation calculation. The model built by the author was efficient but the experiments resulted in the calculations of prediction of special keywords that were not of high-precision and needs to be improved by doing more appropriate calculation.

After doing intense research of previous studies and work performed by multiple researchers this study concluded that apart from the automatic tools there are no accurate and efficient solution proposed for detecting SQL injection attacks. So, we decided to propose an advance solution for detecting SQL injection that can be used at industrial level to eliminate the vulnerabilities of SQL injection. This study proposes to build advance machine learning models to detect SQL injection and to classify those attack. It further proposes to do a comparative analysis of all the models built and to find out the best and efficient mode suitable for solving our problem.

In the following sections discusses the method used in the proposed solution. The deep understanding of each model used in our study is discussed to get the better understanding of algorithm behind the working of the models.

3. RESEARCH METHOD

This section contains the methodology that is utilized to solve the issue. Herein several methods are used to experiment along with the previous studies done by the researchers in the past using the traditional methods.

The study offers a technique to detect SQL Injection utilizing the Machine Learning upon conducting considerable research and evaluating past work in detecting SQL Injection attacks (R. Sathya, 2013). It also proposes performing a comparative analysis of several machine learning models in order to identify the most suitable model for handling this challenge.

The two types of Machine Learning algorithms are:-
a) Supervised Learning Algorithm and
b) Unsupervised Learning Algorithm.

Supervised Learning: The supervised learning algorithm depends on training a sample data from a source of data that has previously been classified correctly (Cannady, 2017). In feed forward or Multilayer Perceptron (MLP) models, such strategies are used. There are three different properties of this MLP (Cannady, 2017).

- One or more layers of hidden neurons that are not part of the input or output layers of the network that enable the network to learn and solve any complex problems
- The nonlinearity reflected in the neuronal activity is differentiable and,
- The interconnection model of the network exhibits a high degree of connectivity these characteristics along with learning through training solve difficult and diverse problems.

The error back propagation algorithm is a method of learning in a supervised model via training. The samples of input and output are used to train the network in an error correction-learning algorithm and determine the error signals, which is the difference between the expected output and the intended output and the synaptic weights of the neurons are modified which is equal to the error signal's product and the synaptic weight's input instance.

Unsupervised Learning: An unsupervised learning algorithm utilizes the self-organizing networks of neural to recognize the concealed order within an unlabelled input data. The capability to retain and sort out information with no signals for error to analyze the prospective resolution. In unsupervised learning, the lack of strategy for the learning algorithm might be useful since it enables the algorithms to explore retrospectively for patterns that were not initially considered. The following are the primary characteristics of Self-Organizing Maps (SOM) (Handa, 2019;):

- An adaptive transformation is done, wherein random dimension pattern of the incoming signals is converted into one- or two-dimensional map.
- Feed forward structure is represented by the network along with sole computational layer which is made up of rows and columns of neurons.
- Each input signal is retained in its right context at every step of representation, and
- Neurons that engage with highly associated segment of data are near together and interact via synaptic connections. Since the neurons challenge each other in the layer to become active therefore this computational layer is also known as the competitive algorithm.

The study has utilized supervised learning algorithm to resolve the issue. The Supervised Learning includes two types of machine learning models which are regression model and classification model and it has used the classification model since the main goal of this study is to determine the whether the query comprises of any SQL Injection attack or not.

One of the most commonly occurring decision-making activities involves in human activity is classification. When an object needs to be allocated to a predetermined class or class influenced by a number of recorded attributes, a classification challenge arises. Many industrial issues have been labeled as classification issues. Weather forecasting, character recognition, bankruptcy forecasting, medical diagnosis, Stock market estimation, and speech recognition are just a few examples. Many industrial issues have been labeled as classification issues. Weather forecasting, character recognition, bankruptcy forecasting, medical diagnosis, Stock market estimation, and speech recognition are just a few examples. These classification difficulties can be handled in a few ways, by using both mathematically and nonlinearly (Siraj, n.d.).

So as to develop a machine learning model, the study particularly uses the Gradient Boosting technique which is an Ensemble learning strategy for reducing inaccuracies and delivering more correct estimates. To generate results, the Gradient Boosting technique uses basic classifiers, primarily decision trees, in a sequential way. Gradient Boosting with Multiple Decision Tree Classifiers The approach begins by classifying the data with a basic classifier while using several decision tree classifiers for the Gradient Boosting. The figure [7] shows the structure of Gradient Boosting algorithm.

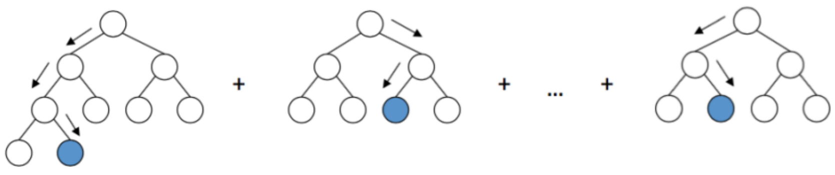


Figure 7: Gradient Boosting Algorithm structural design

The results are then used to determine the defects or data points that the basic classifier could not comfortably integrate. In the next round, the algorithm concentrates on those data points and tries to integrate them as well. By following these steps, the inaccuracies are decreased and also considered the outlying data points. However, in an attempt of overdoing this renovation might also result in over fitting. Therefore, the key aspect of considering this strategy is learning to terminate remodeling at an appropriate precision and mistake rate as a result.

4. DESIGN SPECIFICATION

In this section description of the architecture of our solution is described. The solution proposed in this study is to build a machine learning model to detect SQL injection attacks. The stages involved in implementing SQL injection detection system from scratch is discussed and described. There is total six stages involved in implementing our model. The structure of the model is shown below.

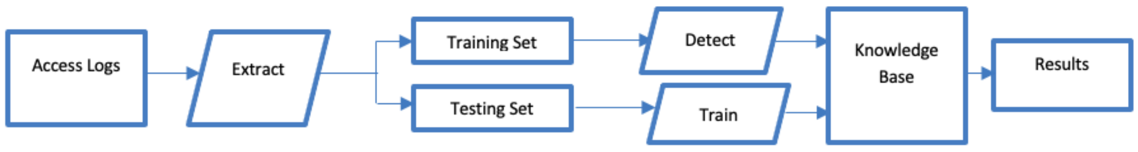


Figure 8: Architecture of SQL Detection system

Access Logs: Logs represents the collection of plain-text data and SQL injection data. To access the logs generally automatic tools are used. For example, Sematext logs, Paesseler PRTG Network monitor etc.

Extract: The logs accessed from the web application are then extracted in a text format in .csv format. The dataset obtained goes through data preparation, cleaning and Feature extraction phase.

Training Set and Testing Set: In this stage to train and test our model, split the dataset into two parts training and test data. It is always advisable to split the dataset in ratio of 3:2 or 7:3. Major part of dataset to be used as a training data and rest for testing.

Train: In this phase, training of the model is performed by training dataset which means increasing the knowledge base of the model. The study has used 70% of our dataset for training.

Test: Once your model is trained, now your machine can be tested using test dataset. For testing the model, 30% of the dataset is used which is unknown to our model in order to test the efficiency of our model.

Knowledge base: The knowledge base is defined as the training dataset which the machine has learnt. The more data provided to the model, your model performs more accurate and the efficiency of the model gets increased.

Results: To determine the results, there are multiple techniques that can be used such as, accuracy score, f1 score, confusion matrix. The result is to determine the accuracy and efficiency of the model. The more the accuracy % the better is the prediction.

5. IMPLEMENTATION

5.1. Dataset

The dataset consists of the following two parts: Their descriptions are given under respective sections.

5.1.1. Plain-Text Dataset

This dataset consists of plain-text sentences and has around four thousand rows. The plaintext dataset has been created with payloads received from html forms. The dataset consists of a combination of URL's, special characters, textual data and numerical data.

Following features of this dataset make it a good choice for this problem.

5.1.2. Diversity

The dataset not only contains just the textual data, but it is comprised of special characters and numbers. This is helpful while training the model to identify SQL Injections with better accuracy and avoid false positives.

5.1.3. Source

The dataset is created by collecting user inputs from a form in a web application. Because of the source of the dataset, there is more probability of wide range of scenarios being covered for training the model efficiently.

5.2. SQL Injection Dataset

It was difficult to compile a dataset for this topic because no datasets with open access to genuine SQL Injection assaults were accessible. A software tool called Libinjection was used to construct the dataset for SQL Injections. Libinjection is a free software tool for web application vulnerability assessments. To check the web applications for vulnerabilities of SQL Injection attacks, it sends SQL Injections as payload. All the payloads created by libinjection were collected using this software for a specific instance, and along with a dataset all these payloads were utilized as that of the Structured Query Language Injection dataset (Bentéjac, 2021). Around 6,000 SQL Injections including all three categories, Blind SQL Injection, Error Based, and Union Based, are included in this collection. An example of SQL Injection dataset is shown in Figure

5.2.1. Categories

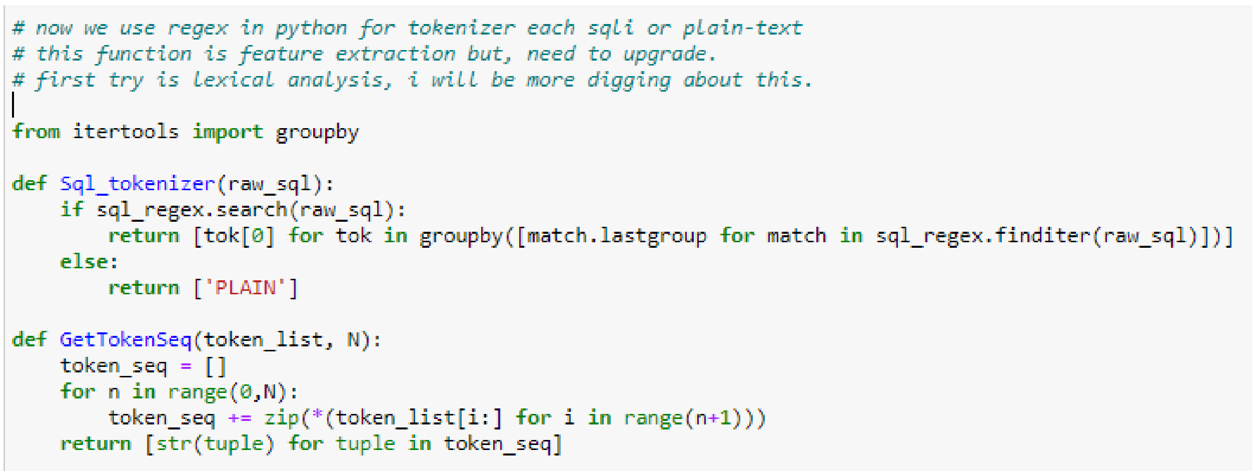
SQL Injections of all categories are included which will help during the model training, which will result in the ability to correctly recognize all the categories of the SQL injections.

5.2.2. Size

The dataset is large enough for this model to be suitably trained.

5.2. Tokenization

Tokenization is generally the first and most essential stage in data preprocessing in machine learning analyses that use datasets based on text. Tokenization is the process of splitting a string of characters into little parts known as tokens. In other cases, tokenization often entails the removal of specific characters (Natekin Alexey, 2013). This is a common technique used in learning based on words. The picture below shows a typical tokenization instance in NLP (Natural Language Processing). Each component of the statement is tokenized at each stage, as can be observed.



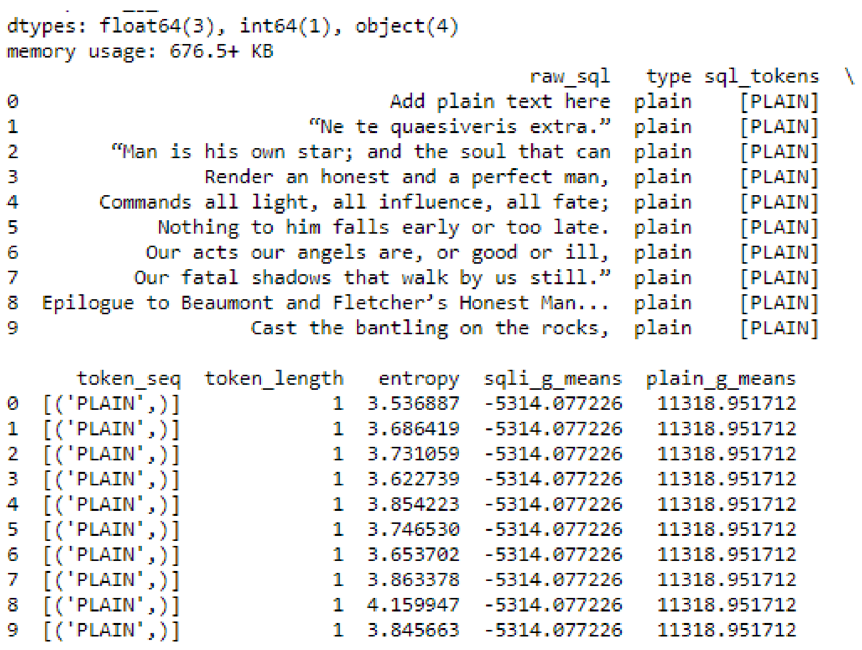
```python
# now we use regex in python for tokenizer each sqli or plain-text
# this function is feature extraction but, need to upgrade.
# first try is lexical analysis, i will be more digging about this.

from itertools import groupby

def Sql_tokenizer(raw_sql):
    if sql_regex.search(raw_sql):
        return [tok[0] for tok in groupby([match.lastgroup for match in sql_regex.finditer(raw_sql)])]
    else:
        return ['PLAIN']

def GetTokenSeq(token_list, N):
    token_seq = []
    for n in range(0,N):
        token_seq += zip(*(token_list[i:] for i in range(n+1)))
    return [str(tuple) for tuple in token_seq]
```

Figure 9: Tokenization Implementation



```
dtypes: float64(3), int64(1), object(4)
memory usage: 676.5+ KB
                                              raw_sql   type sql_tokens  \
0                                 Add plain text here  plain    [PLAIN]
1                           "Ne te quaesiveris extra."  plain    [PLAIN]
2                "Man is his own star; and the soul that can  plain    [PLAIN]
3                       Render an honest and a perfect man,  plain    [PLAIN]
4              Commands all light, all influence, all fate;  plain    [PLAIN]
5                    Nothing to him falls early or too late.  plain    [PLAIN]
6                   Our acts our angels are, or good or ill,  plain    [PLAIN]
7                  Our fatal shadows that walk by us still."  plain    [PLAIN]
8   Epilogue to Beaumont and Fletcher's Honest Man...  plain    [PLAIN]
9                        Cast the bantling on the rocks,  plain    [PLAIN]

      token_seq  token_length   entropy  sqli_g_means  plain_g_means
0  [('PLAIN',)]             1  3.536887  -5314.077226   11318.951712
1  [('PLAIN',)]             1  3.686419  -5314.077226   11318.951712
2  [('PLAIN',)]             1  3.731059  -5314.077226   11318.951712
3  [('PLAIN',)]             1  3.622739  -5314.077226   11318.951712
4  [('PLAIN',)]             1  3.854223  -5314.077226   11318.951712
5  [('PLAIN',)]             1  3.746530  -5314.077226   11318.951712
6  [('PLAIN',)]             1  3.653702  -5314.077226   11318.951712
7  [('PLAIN',)]             1  3.863378  -5314.077226   11318.951712
8  [('PLAIN',)]             1  4.159947  -5314.077226   11318.951712
9  [('PLAIN',)]             1  3.845663  -5314.077226   11318.951712
```

Figure 10: Output of Tokenization

Furthermore, because we're looking for SQL Injections, each letter in both datasets is kept, and regular expressions is utilized to construct tokens are constructed rather than tokenizing words in this scenario. In this method, a series of characters are combined with each other for tokenization.

5.2.1. Regular Expression

Plain-text as well as SQL Injection datasets both have regex employed to tokenize every input. They specify a string structure for a series of characters. The matching of the pattern are widely utilizes the Regular expressions (Chen, n.d.). Regular expression to be compiled into a regular expression object re.compile() technique is utilized in this research.

Several SQL queries along with SQL words that are reserved are utilized to generate the regular expression. Lexical evaluation employing regular expressions in Python is used to accomplish tokenization.

To separate the objects into tokens, the Groupby() technique is utilized. Using the tokens that are generated from the dataset, extraction of feature is conducted on the dataset. There are three factors in the token object.

5.2.2. Token_Count

The token count argument keeps track of how many times certain token appears in the overall dataset.

5.2.3. Token_Value

The token value attribute holds the true figures of newly generated tokens.

5.2.4. Token_Type

The plain or sqli are the two categories in which the token_type criterion the token falls wherein the dataset of SQL injection creates sqli token types and the dataset based on plain text creates the plain token type. Further, function groupby() is utilized to assembled the tokens simultaneously depending on the series that it frequently happen concurrently.

5.3.Feature Extraction

The initial stage in extraction of features is to compute the G-test Scores including all token items in the dataset, which is done once the dataset has been tokenized. A dataframe is built using Python's Pandas package before computing the G-test results. The following columns make up this dataframe, which serves as the fresh dataset.

- Token_Count

- Token_Value

The fresh dataset is calculated using the G-test.

5.3.1. Calculating G-test score

The likelihood ratio is another name for the G-test score. It is regarded as a viable substitute of the Chi-square Test. When there is only one categorical variable and 2 classes to categorize, the G-test score is typically utilized. Because the classification is to categorize data into 2 different classes, plain-text and SQL injection, this aspect makes G-test score ideal for usage in our technique. G-test scores are used to measure how far a prediction deviates from the optimal prediction. They are used with ordinal attributes. The data must be pre-processed before the G-test scores can be computed. The numbers in the data are transformed to float type numerical numbers. In this scenario, two sorts of G-test scores are computed.

- Observed G-test Score
- Expected G-Test Score

The overall tokens, the number of tokens in each row, and the categories of tokens are used to determine the expected G-test score. If the data had been distributed properly then the expected G-test score would be obtained. The real score of the data event is the recorded G-test score.

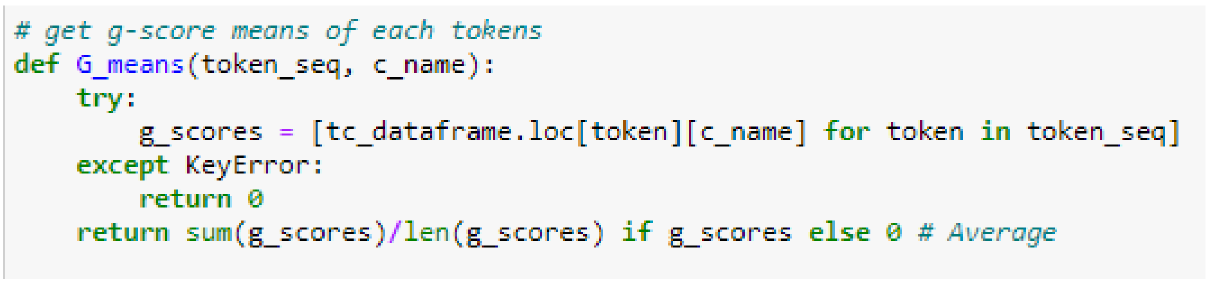
5.3.2. Calculate Entropy

The computation of the entropy is the next stage in the extraction of the feature, the computation is done for every row that is in the dataset. The way it is easier to estimate the indiscrimination of the data. The entropy of the dataset depends on the level of similarity of the data, wherein the dataset entropy will be less if the data is extremely alike and vice versa. So as to split the divide the data entropy is used by the decision trees (Farooq, n.d.).

A decision tree's objective is to divide data in a manner that related data is clustered collectively. As a result, the decision trees use entropy to authenticate their split. They proceed with the separation if the entropy drops, and if the entropy grows, they try to divide at a different place.

5.3.3. Calculate G-test Score Mean

The approximate value of G-test scores for every token inside the dataset is obtained in this stage. Utilizing pandas library in Python, a fresh dataset is created and maintained in a Data frame. Utilizing the Gradient Boosting Classifier, the model is trained using the dataset.



```
# get g-score means of each tokens
def G_means(token_seq, c_name):
    try:
        g_scores = [tc_dataframe.loc[token][c_name] for token in token_seq]
    except KeyError:
        return 0
    return sum(g_scores)/len(g_scores) if g_scores else 0 # Average
```

Figure 11: Formula for G-test mean score

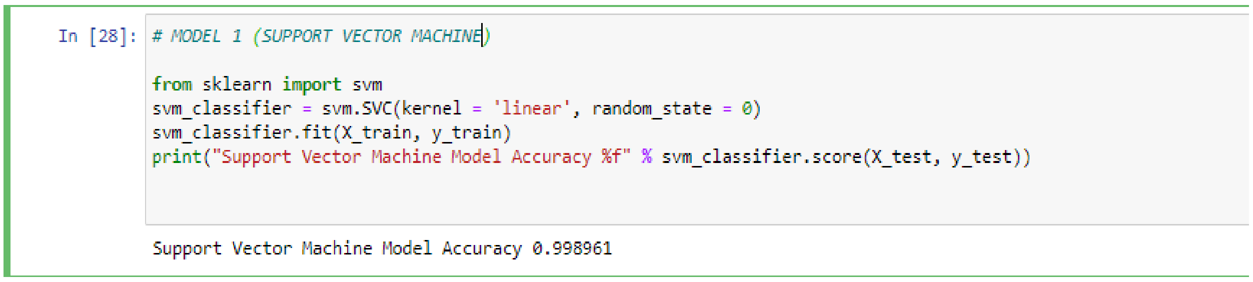
6. EVALUATION

6.1. Experiment 1

The first experiment performed was using Support Vector Machine algorithm. Support Vector Machine classification model was built to detect SQL injection attack. The parameters considered for building the model are,

Kernel = Linear, random_state = 0

In the next step, models are trained by fitting 'X_train' and 'Y_train' data. Then, the testing of the model was performed by predicting 'X_test' data which was compared to the 'Y_test'. Finally, to evaluate the efficiency of our model, this study has used Accuracy score method. The model fitting and prediction is shown in figure [9]



```
In [28]: # MODEL 1 (SUPPORT VECTOR MACHINE)

         from sklearn import svm
         svm_classifier = svm.SVC(kernel = 'linear', random_state = 0)
         svm_classifier.fit(X_train, y_train)
         print("Support Vector Machine Model Accuracy %f" % svm_classifier.score(X_test, y_test))

         Support Vector Machine Model Accuracy 0.998961
```

Figure 12: Experiment 1 - Support Vector Machine Model

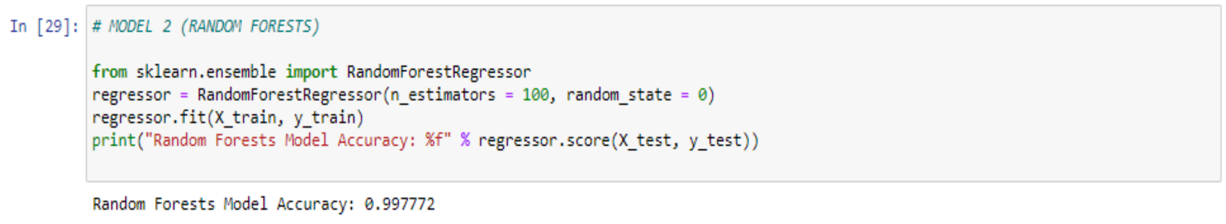
Results: The accuracy of the Support Vector Machine model is 99.8%

6.2. Experiment 2

The second experiment performed was using Random Forests Classifier. Random Forests classification model was built to detect SQL injection attack. The parameters considered for building the model are,

n_estimators = 100, random_state = 0

In the next step, models are trained by fitting 'X_train' and 'Y_train' data. Then, the testing of the model was performed by predicting 'X_test' data which was compared to the 'Y_test'. Finally, to evaluate the efficiency of our model, this study has used Accuracy score method. The model fitting and prediction is shown in figure [10].



```
In [29]: # MODEL 2 (RANDOM FORESTS)

         from sklearn.ensemble import RandomForestRegressor
         regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)
         regressor.fit(X_train, y_train)
         print("Random Forests Model Accuracy: %f" % regressor.score(X_test, y_test))

         Random Forests Model Accuracy: 0.997772
```

Figure 13: Experiment 2 - Random Forests Classification Model

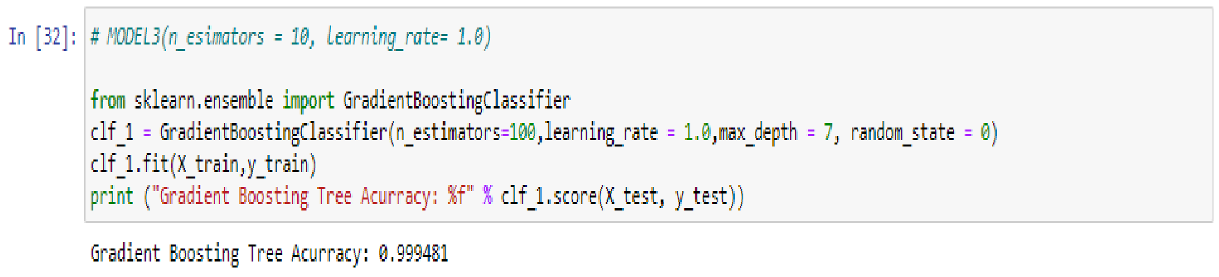
Results: The accuracy of the Support Vector Machine model is 99.7%

6.3. Experiment 3

The third experiment performed was using Gradient Boosting Classifier algorithm. Gradient Boosting classification model was built to detect SQL injection attack. The parameters considered for building the model were,

n_estimators = 100, learning_rate = 0, max_depth = 7, random_state = 0

In the next step, models are trained by fitting 'X_train' and 'Y_train' data. Then, the testing of the model was performed by predicting 'X_test' data which was compared to the 'Y_test'. Finally, to evaluate the efficiency of our model, this study has used Accuracy score method. The model fitting and prediction is shown in figure [11].



```
In [32]: # MODEL3(n_esimators = 10, learning_rate= 1.0)

         from sklearn.ensemble import GradientBoostingClassifier
         clf_1 = GradientBoostingClassifier(n_estimators=100,learning_rate = 1.0,max_depth = 7, random_state = 0)
         clf_1.fit(X_train,y_train)
         print ("Gradient Boosting Tree Acurracy: %f" % clf_1.score(X_test, y_test))

         Gradient Boosting Tree Acurracy: 0.999481
```

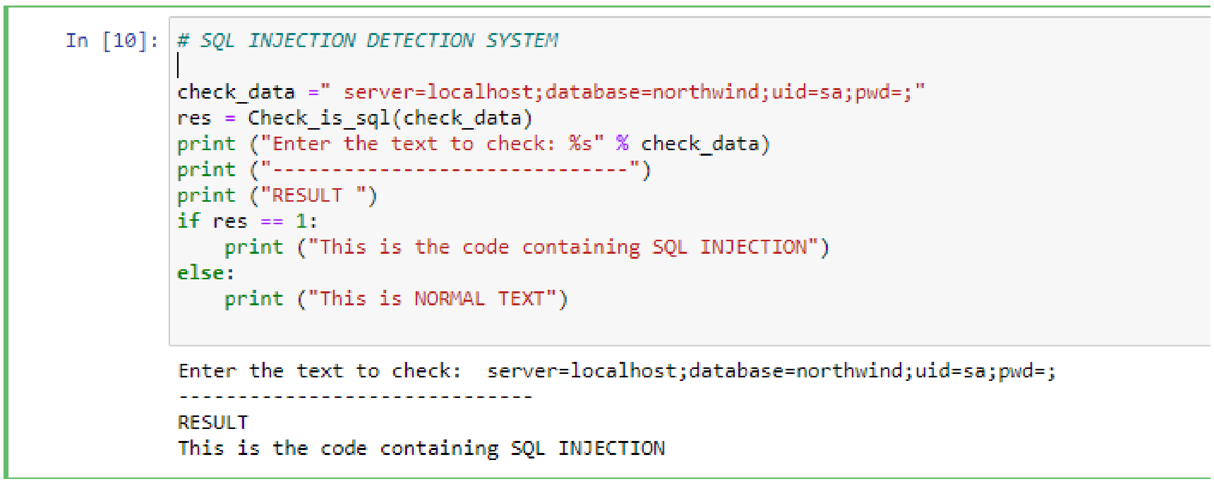
Figure 14: Experiment 3 - Gradient Boosting Classification Model

Results: The accuracy of the Gradient Boosting Machine Learning model is 99.9%

6.4. Discussion

As you can see that the third experiment gives the maximum accuracy of 99.9%. Thus, in order to solve the problem statement, the Gradient Boosting classifier model outperforms and was used to predict the SQL Injection attacks.

After finding out the most optimal classification model, Gradient Boosting model is been used to build our SQL Injection Detection system that will predict any SQL query. Our SQL Detection system takes the SQL statement as a user input and detects if the provided input contains the SQL injection or not. See the below figure [12] that shows the example of multiple user inputs. The variable "check_data" stores the user input value.



```
In [10]: # SQL INJECTION DETECTION SYSTEM
         |
         check_data =" server=localhost;database=northwind;uid=sa;pwd=;"
         res = Check_is_sql(check_data)
         print ("Enter the text to check: %s" % check_data)
         print ("------------------------------")
         print ("RESULT ")
         if res == 1:
             print ("This is the code containing SQL INJECTION")
         else:
             print ("This is NORMAL TEXT")

         Enter the text to check:  server=localhost;database=northwind;uid=sa;pwd=;
         ------------------------------
         RESULT
         This is the code containing SQL INJECTION
```

Figure 15: SQL Injection Detection System Model

The figure [13] below shows the few examples of user inputs and the results obtained by our SQL Injection Detection System.

S.NO	USER INPUT (Check_data)	RESULTS
1	server=localhost;database=northwind;uid=sa;pwd=;	`This is the code containing SQL INJECTION`
2	SELECT * FROM users WHERE username = '' OR 1=1-- ' AND password = 'foo'	`Enter the text to check:  SELECT * FROM users WHERE username = '' OR 1=1-- ' AND password = 'foo'` `------------------------------` `RESULT` `This is the code containing SQL INJECTION`
3	SELECT /*!32302 1/0, */ 1 FROM tablename	`Enter the text to check:  SELECT /*!32302 1/0, */ 1 FROM tablename` `------------------------------` `RESULT` `This is NORMAL TEXT`
4	SELECT ID, Username, Email FROM [User]WHERE ID = 1 AND ISNULL(ASCII(SUBSTRING((SELECT TOP 1 name FROM sysObjects WHERE xtYpe=0x55	`Enter the text to check:  SELECT ID, Username, Email FROM [User]WHERE ID = 1 AND ISNULL(ASCII(SUBSTRING((SELECT TOP 1 name FROM sysObjects WHERE xtYpe=0x55 AND name NOT IN(SELECT TOP 0 name FROM sysObjects WHERE xtYpe=0x55)),1,1)),0)>80--` `------------------------------` `RESULT` `This is the code containing SQL INJECTION`

	AND name NOT IN(SELECT TOP 0 name FROM sysObjects WHERE xtYpe=0x55)),1,1)),0)>80-- "	

5	hi my name is chirag chaudhary	`Enter the text to check:  hi my name is chirag chaudhary`
		`-----------------------------`
		`-`
		`RESULT`
		`This is NORMAL TEXT`

Figure 16: SQL Injection Detection system Results Table

7. CONCLUSION AND FUTURE WORK

After doing an in-depth research about SQL Injection attacks and their effects in an industry, it is really important to mitigate this attack and to provide a reliable solution against it. This vulnerability is listed in top 10 OWASP and is still considered as one of the main attack that can harm the personal and sensitive data. There were many mechanism that were applied in order to solve this problem but majority of them only focused on the previous attacks and the unknown attacks were not registered. In this study, machine learning algorithm has been used to register this problem and build a SQL Injection detection system that can predict previous and unknown queries and cam classify whether the given query contains a SQL injection attack or not. This study experimented with three major classification models, Support Vector Regression, Random Forests and Gradient Boosting algorithm. In our case, Gradient Boosting algorithm out performs and gives an accuracy of 99.9% . To achieve this efficiency hyper-parameter tuning was performed. To conclude, this study was able to build a SQL Injection Detection system using Gradient Boosting model and our system is able to classify unknown queries as a SQL Injection attack and plain text.

In the future this study could be further enhanced to detect other important attacks. Further the approach used in this model can also be modified and can be combined with other approaches such as dynamic and static code analysis. Alternatively to enhance the security and performance of existing approach one can increase the size of the dataset which will improve the knowledge base of the model. Since the scope of machine learning is wide, also one can further experiment with neural networks as they are relatively strong models.

8. REFERENCES

Anley., C., 2002. Advanced SQL Injection In SQL Server Applications. *White paper, Next Generation Security Software Ltd.* .

Bentéjac, C. C. A. &. M.-M. G., 2021. A comparative analysis of gradient boosting algorithms.. *Artif Intell Rev 54,* , Issue https://doi.org/10.1007/s10462-020-09896-5, p. 1937–1967 .

Cannady, J. B. F. a. J., 2017. *The promise of machine learning in cybersecurity,.* s.l., IEEE, pp. pp. 1-6.

Chen, T. C. G., n.d. *XGBoost: A Scalable Tree Boosting System,* Washington: IEEE.

Dornseif, M., 2005. *Common Failures in Internet Applications,* s.l.: s.n.

Farooq, U., n.d. *Ensemble Machine Learning Approaches for Detection of SQL Injection Attack,* s.l.: Preliminary communication.

Handa, A. S. A. S. S., 2019;. Machine learning in cybersecurity: A review.. *WIREs Data Mining Knowl Discov.* , Issue https://doi.org/10.1002/widm.1306.

Hou, K. W. a. Y., 2016. *Detection method of SQL injection attack in cloud computing environment.* s.l., IEEE, pp. pp. 487-493,.

Inc., O. F., 2021. *OWASP Top Ten.* [Online] Available at: https://owasp.org/www-project-top-ten/ [Accessed 03 09 2021].

Inyong Lee, S. J. S. Y. J. M., 2012. A novel method for SQL injection attack detection based on removing SQL query attribute values. *Mathematical and Computer Modelling ScienceDirect,* Volume 55(Issues 1–2), pp. Pages 58-68,.

Jemal, I. &. C. O. &. H. H. &. M. A., 2020. SQL Injection Attack Detection and Prevention Techniques Using Machine Learning.. *International Journal of Applied Engineering Research,* pp. 569-580.

K. Kamtuo and C. Soomlek, 2017. *Machine Learning for SQL injection prevention on server-side scripting.* Chiang Mai, Thailand, IEEE, pp. pp. 1-6.

L. Xiao, S. M. T. I. a. K. S., 2016. SQL Injection Attack Detection Method Using Expectation Criterion. *2016 Fourth International Symposium on Computing and Networking (CANDAR),* Issue doi: 10.1109/CANDAR.2016.0116., pp. pp. 649-654.

Milic, T., 2019. *SQL Injection Attacks and Defenses,* Melobourne: Swinburne University of technology.

Muhammad Amirulluqman Azman, M. F. M. a. R. S., 2021. Machine Learning-Based Technique to Detect SQL Injection Attack. *Journal of Computer Science,* 05 February.

Natekin Alexey, K. A., 2013. Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics,* 7(https://www.frontiersin.org/article/10.3389/fnbot.2013.00021), p. 21.

Principal, A. C., 2018. SQL Injection Attacks and Prevention: An Overview. *IJCRT,* 2 April.6(2).

R. Sathya, A. A., 2013. Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification. *(IJARAI) International Journal of Advanced Research in Artificial Intelligence,* Volume 2.

Rubidha Devi.D*, 2R.Venkatesan, 3Raghuraman.K , Dec-2016 . A STUDY ON SQL INJECTION TECHNIQUES. *nternational Journal of Pharmacy & Technology IJPT,* Vol. 8(Issue No.4), pp. 22405-22415 .

S. O. Uwagbole, W. J. Buchanan and L. Fan,, 2017. *Applied Machine Learning predictive analytics to SQL Injection Attack detection and prevention.* Lisbon, Portugal, IEEE, pp. pp. 1087-1090.

Siraj, V. F. a. A., n.d. *Applications of Machine Learning in Cyber Security,* USA: IEEE.

Swarnaprabha Patil, Prof. Nitin Agrawal, 2015. Web Security Attacks and Injection- A Survey. *International Journal of Advancements in Research & Technology,* February.4(2).

Wikipedia, 2021. *SQL injection.* [Online] Available at: https://en.wikipedia.org/wiki/SQL_injection [Accessed 23 08 2021].

William G.J. Halfond, Jeremy Viegas, and Alessandro Orso, n.d. *A Classification of SQL Injection Attacks and Countermeasures,* s.l.: Gatec.edu.