Detection of SQL Injection Attacks: A Machine Learning Approach

Musaab Hasan
College of Engineering & Information Technology, University of Science & Technology of Fujariah
Fujairah, United Arab Emirates (UAE)
m.mohammad@ustf.ac.ae

Zayed Balbahaith
College of Technological Innovation Zayed University
Abu Dhabi, United Arab Emirates (UAE)
m80007225@zu.ac.ae

Mohammed Tarique
College of Engineering & Information Technology, University of Science & Technology of Fujariah
Fujairah, United Arab Emirates (UAE)
m.tarique@ustf.ac.ae

***Abstract*— With the rapid growth in online services, hacking (alternatively attacking) on online database applications has become a grave concern now. Attacks on online database application are being frequently reported. Among these attacks, the SQL injection attack is at the top of the list. The hackers alter the SQL query sent by the user and inject malicious code therein. Hence, they access the database and manipulate the data. It is reported in the literature that the traditional SQL injection detection algorithms fail to prevent this type of attack. In this paper, we propose a machine learning based heuristic algorithm to prevent the SQL injection attack. We use a dataset of 616 SQL statements to train and test 23 different machine learning classifiers. Among these classifiers, we select the best five classifiers based on their detection accuracy and develop a Graphical User Interface (GUI) application based on these five classifiers. We test our proposed algorithm and the results show that our algorithm is able to detect the SQL injection attack with a high accuracy (93.8%).**

Keywords*— *Database application, SQL injection, SQL detection, database Security, machine learning, classifiers.

I. Introduction

Global internet users have reached 4.2 billion in 2018. The International Telecommunication Union (ITU) is expecting that about 50% of world population will have Internet access by the end of year 2018 [1,2]. In harmony with this tremendous growth in Internet users, there has been a similar growth in the services offered over the Internet. Till now, numerous online services have been introduced including e-commerce, business-to-business support, and large data repositories. Many other innovative services are yet to come in a very near future. As the number of online services is increasing, so is the security threats, which have become a great concern now for both Internet users as well as online service providers. Having the organization's resources available online and adopting lenient security measures, unauthorized parties can access to confidential and sensitive data of an organization. Nowadays, database applications have become the main target of this kind of unauthorized access.

The security threats on database application are often associated with users' supplied data. Structured Query Language (SQL) is commonly used to query, operate, and administer database application. User supplied data are often used to generate the SQL statement that ultimately accesses database. The attackers can modify or manipulate the user-supplied data and hence can get access to the database. National Security Agency (NSA) [3] has reported numerous attacks on the database applications as shown in Fig. 1. Among these attacks, the most common attack is called cross site scripting attack (49.67%) and the next common attack is the SQL injection attack (18.01%). Open WEB Application Security Project (OWASP) has also listed the SQL injection attack as the top most threat on web based application in 2013 [4].

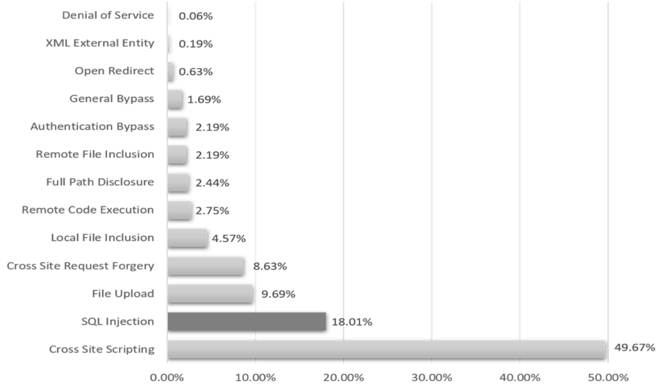


Fig. 1. Common attacks on Web-based applications [3]

By using SQL injection, the attackers can alter the SQL statement by replacing user's supplied data with their own data as shown in Fig. 2. Thus, the attackers can have direct access to an database server to retrieve confidential information. However, the impacts of the SQL injection attacks are far reaching and they vary depending on the database applications. Some of these impacts of a successful SQL injection attacks include authentication bypass, information disclosure, compromised data integrity, compromised availability of data, and remote command execution. Authentication bypass enables an attacker to access database application by using fake username and password. Information disclosure allows an attacker to obtain sensitive data from the database. Compromised data integrity assists an attacker to change some contents of the data in the database. Compromised availability data enables an attacker to delete some information from the database. Remote command execution allows an attacker to affect the host operating system.

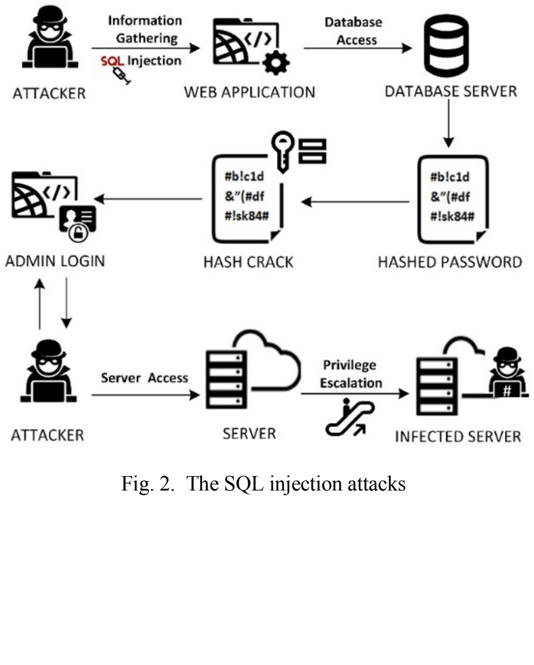


Fig. 2. The SQL injection attacks

Preventing the SQL attack requires accurate configuration and database programming. In addition, database applications must be associated with enough security measures [5]. The query coming from the user side could be in the form of (a) inputs in text box [6], (b) cookies saved in the user machine [7], and (c) server variables placed in the URL [8,9].

Researchers have recommended various techniques to prevent the SQL injection attack. The most popular techniques can be loosely classified as static, dynamic, and hybrid (both static and dynamic) [11]. The static analysis technique checks the correctness of the generated SQL queries and examines the mismatch in the queries [3]. The dynamic analysis technique is considered as a more advanced technique, which allows the system to identify the SQL injection in the queries that are valid [10]. Hybrid technique encapsulates the advantages of both static and dynamic analysis. First, it uses the static analysis to build and train the detection models. Then, it uses dynamic analysis to investigate these models to make the correct decision [11]. Both dynamic and hybrid analysis techniques depend mainly on machine learning approach.

Many solutions have been suggested based on the above mentioned techniques. Among these solutions machine learning approach is considered very effective in detecting the SQL injection attack. It has been proved that the machine learning approach is not only suitable to prevent existing known attacks, but also suitable to prevent unknown attacks. However, the results may include many 'false' positives and 'false' negatives depending on the classifier used [9]. The SQL injected statement can be easily detected provided the proper classifier is used and up-to-date data is used to train the same [10].

In this paper, we propose a heuristic approach to detect the SQL injection attack. Our approach combines the advantages of both static and dynamic analysis through machine learning algorithm. We use MATLAB to develop the system and we also consider a well-studied data set that covers all possible SQL statements. The data set are used to train and test 23 different machine learning classifiers. Then, we select the best five classifiers based on their performances in terms of the true positive and true negative rates. We also present a Graphical User Interface (GUI) based SQL injection detection system called the SQLi detection system in this paper. We test the proposed system and the results show that our system is able to detect the SQL injection attacks with a high accuracy (i.e., 93.8%). The proposed system is placed between the database application and database to make decision whether to pass the query or not to access the database.

The rest of this paper is organized as follows. Some recent related works are presented in section II. The proposed system is presented in section III showing the methods for extracting and selecting the features of the SQL statements and the process for collecting the dataset. The training and testing of the machine learning classifiers are introduced in the same section with the process for selecting the classification algorithms is explained in section IV. The results and analysis are presented in section V. Finally, conclusions and suggested future works are presented in section VI.

II. Related Works

Many researches have been carried out for detecting the SQL injection attack. In this section, we present a critical review of some recently published related works. Minhas J. and Kumar, R. adopt a combined (static and dynamic) technique to detect the SQL injection attacks in [11]. The proposed technique is very efficient, and time saving to detect and block the SQL injection attacks. The proposed technique extracts attribute values and compares the static SQL queries with the dynamic SQL queries. The objective of their study is to minimize the acknowledgment time for the incoming queries. The authors consider two cases namely character by character, and queries having the same number of tokens. It is shown in the work that a combination of both static and dynamic detections reduces the possibility of false detection of the SQL injected statement.

Joshi, A and Geetha, V [12] present a method for detecting the malicious SQL queries based on a combination of two classifiers namely Naïve Bayes and RBC control mechanism. The tested dataset consists of malicious and non-malicious codes, which are classified based on the feature vectors. Their novel contribution of the work is to split the query into significant elements called tokens. Then, they use the index of tokens for further processing and hence they have termed their method as tokenization method. In the classification phase, the proposed system reads the dataset from the text files and sends each data to the classifiers. The results show that the proposed classifier can achieve an accuracy of 93.3%. The major limitation of the work is that a small test dataset is used. Another limitation is that a limited dataset features have been used in the work.

Kamtuo, K and Soomlek, C present a framework in [13] to extort the SQL commands from the dataset and mark them as input variables. These variables are then sent to the machine learning model for the prediction of the SQL injection attacks. The proposed approach prevents the SQL injection attack on the illegal or logically incorrect queries, union queries, and piggybacked queries on server-side scripting by applying compiler platform and machine learning. The authors use 1,100 datasets of vulnerabilities to train four different classifiers namely SVM, Boosted Decision Tree, Artificial Neural Network, and Decision Jungle. The results show that the Decision Tree is the best model in terms of processing and accuracy in prediction. The major limitation of the work is that the authors do not consider client-side actions in the proposed work.

Kumar et al. [14] propose a novel runtime technique to prevent SQL query injection attack based on a combination of static and dynamic analysis. The technique relies on obtaining the value of the SQL query attribute of the web pages when the users submit the parameters. The technique then matches them with a predetermined template query. The proposed system validates user input to reduce the runtime overhead. The authors also present a comparison of detection and prevention methods of the SQL injection attacks namely AMNESIA, SQLCheck, SQLrand, JDBC-Checker, SQLGuard, and Tautology Checker. They use six different criteria namely Tautologies, Illegal/Incorrect Queries, Union Queries, Piggy-Backed Queries, Stored procedures, and Queries structure bypassing in the work. The limitation of the work is that it does not consider a real-life environment to test the accuracy and effectiveness of their approach.

Uwagbole, S and Lu Fan, W [15] propose a machine learning based classification system for detection and prevention of the SQL injection attacks through monitoring the collected dataset from known attacks reported in the literature. The proposed classification system then tests the dataset by checking the signatures list for proving and verifying the outcome result of the token-based phase. The classification system deploys the test on support vector machine (SVM) algorithm by blocking malicious web requests from entering the target back-end database. The proposed work has the advantage of implementing it on a big data set; however, it lacks the implementation on multiple classifiers of machine learning. In addition, the authors did not present any model to test their system.

Flexible design of layers and efficient formulas [26] made very popular deep learning-based methods in different scientific area, especially in automated disease diagnosis, which requires precise operations [27,28]. However, it has been observed recently that there are still some open issues affecting feature extraction in deep learning [29] also parameters should be chosen carefully [30]. Therefore, in this work, a deep neural network has not been preferred.

III. The Proposed Approach

This section provides an overview of the proposed approach and explains the steps to detect the SQL injections. Fig. 3 shows the detail procedure of the proposed system. First, the injected and non-injected SQL statements are collected and recorded in a spreadsheet. Each statement in the spreadsheet is labelled with "No" for non-injected statement and "Yes" for the injected ones. Then the pre-processed spreadsheet file is imported into the proposed features extractor. The feature extractor analyses the statements and generates a feature array for each. The spreadsheet file, containing the extracted features arrays, is split up into 5 fold/parts for the cross validation. When one-fold is used as a testing set, the rest four are used as the training set. This process is repeated five times to ensure that all the dataset are used for both training and testing and to let the classifier classify the results correctly and reduce the false positives and negatives in later stages.

```
1.   Load the dataset for injected SQL statements
2.   Load the dataset for the non-injected SQL statements
3.   Create a spreadsheet file combining the injected and non-injected datasets
4.   Extract the features from the spreadsheet file by applying the developed
       "Features Extractor" MATLAB code.
5.   For Each of the 23 machine learning classifiers
6.       Split the extracted features from the dataset into five folds.
7.       For Each fold of the five folds
8.           Use the current fold as testing set
9.           Use the remaining four folds as training set
10.      End For
11.      Analyse the classifier and identify its accuracy
12.  End For
13.  Select the most accurate five classifiers
14.  Implement the selected classifiers in a user-friendly interface
       for checking new SQL statements.
```

Fig. 3. The proposed system procedure

The classification algorithms, available with MATLAB, are used to train the dataset through 23 different machine learning classifiers. After the classification learners are trained, the accuracy of each classifier is studied and the most accurate five classifiers are selected to achieve an accuracy of 93.8%. At the end, a graphical user interface program is developed in MATLAB to let the users test their own SQL statements through the best five classifiers and identify whether the submitted SQL statements are injected with malicious commands or not.

Training a machine learning classifier requires an array of features that describes each SQL statement. For this purpose, a MATLAB program is developed to investigate each statement and analyse it to produce an array of features. These features are considered the predictors for the machine learning classifier and an extra record is added to them to indicate whether the SQL statement is injected or not. Selecting the features extracted from the statements is an essential part of any machine learning based system. Based on the existing references, we experiment our proposed system initially with nine features. Three out of the initial nine features are excluded from the system as they are found to be producing a misleading effect on the classification system and influencing the overall accuracy of the detection. The selected features are described as follows.

A. Any comment character is present

Having a single or multiple lines comment character in a statement generated from an application is not a common practice by the programmers. The reason is that when a user executes her/his query in the applications, she/he never includes a comment with it. On the other hand, the attackers use this approach to include a comment character after their malicious command to let the system neglect the rest of the query maintained by the application.

B. Number of semicolons

The SQL statements are terminated with one semicolon at the end of it. When an attacker injects her/his malicious commands in the middle of the statement, she/he places a semicolon and then comments the rest of the statement, which result in having multiple semicolons. As we are dealing with it as a string and not command, we can identify if multiple semicolons are present or not.

C. Presence of always true conditions

Always true conditions rarely can be found in the benign SQL statements while they are the most common terms used by attackers in their injection to make the condition always satisfied after the OR operator. Researchers have considered this approach; however, they only have considered the checking for number1=number1, variable1=variable1, and 'string'='string'. In our approach, we add the checking for number1!=number2, variable1!=variable2, and 'string1'!='string2' so that it will also always return a true value.

D. The number of commands per statement

Regular queries generated by applications usually consist of one request only such as SELECT. When an attacker injects her/his code in the original SQL statement, it results in having one statement with more than one requests.

E. Presence of abnormal commands

When an application reads the input from a user and inserts it into a statement, it usually uses SELECT, INSERT, or DELETE requests. Having statement coming from the application with DROP, CREATE, COMMIT, ROLLBACK, GRANT, REVOKE, and DECLARE, requests may indicate that a malicious action is happening.

F. Presence of special keywords

Having a statement coming from an application, which is operated by a user requesting for the version of the SQL server, could indicate a malicious action. A set of similar keywords are used to be checked in the statement through the developed features identifier code. If any of the keywords is present, the program will detect it.

The features that we select for our features extraction system are well experimented and studied in the literature. We implement our system through MATLAB code. The step-by-step pseudocode for the developed features extraction method is given by the algorithm shown in Fig. 4.

```
1.  procedure Features Extractor
2.  Input the name of the spreadsheet file to be imported
3.  Load the spreadsheet file
4.  Save the spreadsheet file content to STRING ARRAY
5.  Initialize Total Array to zeros
6.  Initialize Statements number counter to one
7.  While Statements number counter less than the total number of
      statements
8.      Get the row having the statement with number that is equal to the
        counter
9.      Calculate the features
10.     Append all the calculated values to Total Array
11. end While
12. Create Excel file and save the array to it
13. end procedure
```

Fig. 4. Features Extractor Algorithm

Obtaining a good quality dataset for the SQL statements is a challenging task as no datasets are available on the internet. In our case, the dataset must include two types of instances; those related to non-injected SQL statements and injected statements. Fig. 5 shows the process followed in preparing the dataset for the non-injected SQL statements, where all SQL statements are collected from the examples provided in [21]. We select 105 SQL statements in the work. Note that the statements that are not generated by the application, but generated by the users (i.e., dropping tables or granting privileges) are not considered for this list. The syntax for the collected 105 SQL statements are verified manually and the statements are kept in a spreadsheet file as a preparation for the next step.

```
1.  Gather all the SQL statements examples from the w3eschools SQL
      tutorials.
2.  Eliminate the non-related and repeated statements.
3.  Create a spreadsheet file containing all the SQL statements.
4.  Verify the SQL statements to produce SQL_Statements spreadsheet file.
```

Fig. 5. Dataset Collection procedure for non-injected SQL Statements.

Fig. 6 shows the process followed in obtaining the dataset for the SQL injected statements. This Fig. also shows that the pre-processing of the data and making them ready for the next stage. The most updated and authentic lists are found in the OWASP [22]. The lists show generic as well as vendor specific SQL statements. All proper statements are collected, and their syntaxes are verified manually. Then, the collected injection lists are classified according to their syntaxes that are inserted after the numeric data, string data, numeric data in a bracket, and string data in a bracket. A set of benign statements are selected that includes almost all possibilities. Then the list of injections is injected into these statements to generate full the SQL injected statements. The full SQL statements list is double-checked, and the syntax is verified manually resulting in a 513 SQL injected statements that are collected in one spreadsheet file to be processed for the next step.

```
1.  Load all the SQL injection statements list from OWASP SecLists Project.
2.  For Each of the SQL statements
3.      If statement syntax is in numeric data with bracket
4.          Complete the SQL injected statement by appending
          it with bracket and the closing character.
5.      Else If statement syntax is in string data with bracket
6.          Complete the SQL injected statement by appending
          it with quotation symbol, bracket, and the closing
          character.
7.      Else If statement syntax is in numeric data
8.          Complete the SQL injected statement by the
          numeric closing character.
9.      Else If statement syntax is in string data
10.         Complete the SQL injected statement by the string
          closing character.
11.     End IF
12. End For
13. Generate full list of injected SQL statements in a spreadsheet file.
14. Verify the statements and produce the SQLi_Statements spreadsheet file.
```

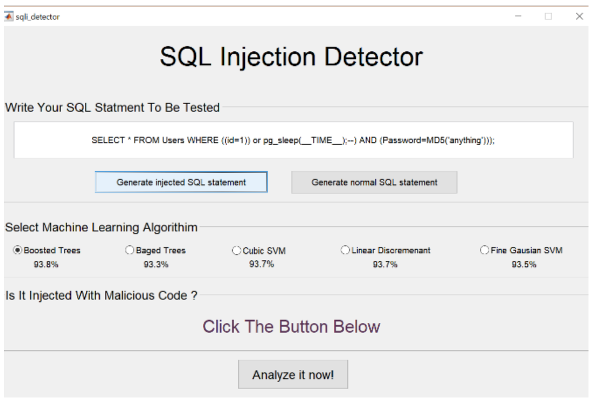
Fig. 6. Dataset Collection procedure for SQL injected Statements.

IV. Classifications

The MATLAB classification learners are used to train and test the dataset with 23 different machine learning algorithms including Coarse KNN, Bagged Trees, Linear SVM, Fine KNN, Medium KNN, RUS Boosted Trees, Subspace Discriminant, Boosted Trees, Weighted KNN, Cubic KNN, Linear Discriminant, Medium Tree, Subspace KNN, Simple Tree, Quadratic Discriminant, Cubic SVM, Fine Gaussian SVM, Cosine KNN, Complex Tree, Logistic Regression, Coarse Gaussian SVM, Medium Gaussian, and SVM. Based on the accuracy of the results we select the best five classifiers to develop our proposed SQL injection detection system.

We discover that the most accurate algorithm is Ensemble Boosted Tress that achieves an accuracy of 93.8%. This algorithm generates a prediction model in the pattern of an ensemble of weak prediction models based on tree regression learning by permitting optimization of an autocratic differentiable loss function [16]. The second algorithm that achieves an accuracy of 93.8% was Ensemble Bagged Trees. This algorithm is based on Random Forest [23], REPTree [24], and J48 [25]. The algorithm considers the samples of a training set using a method called sampling with replacement. Then, it employs the bootstrap sample to train a different component of the base classifier [17]. The Linear Discriminant algorithm achieves a 93.7% detection accuracy. This algorithm is based on the analysis of variance (ANOVA) and pattern regression to find a linear combination of features from the provided data [18]. The Cubic SVM also achieves an accuracy of 93.7%. This algorithm uses the cubic distance metric, classification by obtaining the hyper-plane that distinguishes the two classes of data [19]. The Fine Gaussian SVM is the last algorithm selected for our system as it achieves an accuracy of 93.5%. This algorithm depends on the selection of Kernel, and it uses cross validation on few collections of parameter in successions [20].

Based on the above mentioned five machine learning algorithms, we develop a GUI MATLAB program as shown in Fig. 7. The GUI allows a user to input her/his SQL statement and get it classified by our proposed system. The pseudocode of the developed GUI program is shown in Fig. 8.



sqli_detector

SQL Injection Detector

Write Your SQL Statment To Be Tested

```
SELECT * FROM Users WHERE ((id=1)) or pg_sleep(__TIME__);--) AND (Password=MD5('anything')));
```

Generate injected SQL statement | Generate normal SQL statement

Select Machine Learning Algorithim

[✓ Boosted Trees 93.8%] Baged Trees 93.3% Cubic SVM 93.7% Linear Discremenant 93.7% Fine Gausian SVM 93.5%

Is It Injected With Malicious Code ?

Click The Button Below

Analyze it now!

Fig. 7. SQL injection GUI program screenshot

```
1.  procedure sqli_detector GUI
2.  Select the classifier
3.  Load the classifier data structure
4.  Input the statement or generate random one automatically
5.  Save the statement into STRING
6.  Calculate the features
7.  Create Total array and assign all calculated features to it
8.  Call predict function of the classifier structure and pass Total
      array to it
9.  Display the result of prediction
10. end procedure
```

Fig. 8. The SQL Detector GUI Algorithm

V. The Results And Analysis

The accuracy of the results of any machine learning based system is measured based on True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN) values that are generated after the testing phase. In our case, the TP and TN represent the number of SQL statements that are classified correctly while FP and FN represent the SQL statements that are classified incorrectly. The values of TP, FP, TN, and FN can be combined in a matrix called confusion matrix as shown in Table 1. The higher the values of TP and TN the more accurate the classification is.

Table 1. Guide to read confusion matrix

True Negative Correctly classified as non-injected SQL Statement	**False Negative** Incorrectly classified as injected SQL Statement
False Positive Incorrectly classified as non-injected SQL Statement	**True Positive** Correctly classified as injected Statement

Table 2. Confusion matrix, TP, FP and Accuracies for top 5 classifiers.

Classifier	Confusion Matrix		TP Rate (%)	TN Rate (%)	Accuracy (%)
Ensemble Boosted Trees	66	37	>99	64	93.8
	1	512			
Ensemble Bagged Trees	66	37	>99	64	93.8
	1	512			
Linear Discriminant	64	39	100	62	93.7
	0	513			
Cubic SVM	65	38	>99	63	93.7
	1	512			
Fine Gaussian SVM	63	40	100	61	93.5
	0	513			

Table 2 presents the results for the five most accurate classifiers. We can conclude from Table 2 that our system achieves more than 99% accuracy in classifying the injected SQL statements in all the five classifiers. The best rate for classifying non-injected SQL statements as non-injected is 64%, which is the reason for reducing the overall accuracy of our system to 93.8% in the top two classifiers. The achieved accuracy is relatively high and it can be further improved by adding more instances in the dataset for the non-injected SQL statements as our dataset was having a small number of instances for the non-injected SQL statements compared to the injected ones. However, our system ensures that almost no SQL-injected statement is incorrectly classified as benign.

The results for the Ensemble Boosted and Bagged Trees are found to be identical. Having 616 SQL statements, 578 out them are classified correctly; while 38 only are classified incorrectly. Both classifiers present a high capability in detecting the SQL-injected statements in more than 99% of the tested data. Fig. 9 shows the receiver operating characteristic (ROC) curve for Ensemble Boosted and Bagged Trees classifiers. Having a ROC curve with an area under the curve (AUC) higher than 0.9 indicates that the system is efficient in classifying injected from non-injected SQL statements correctly.

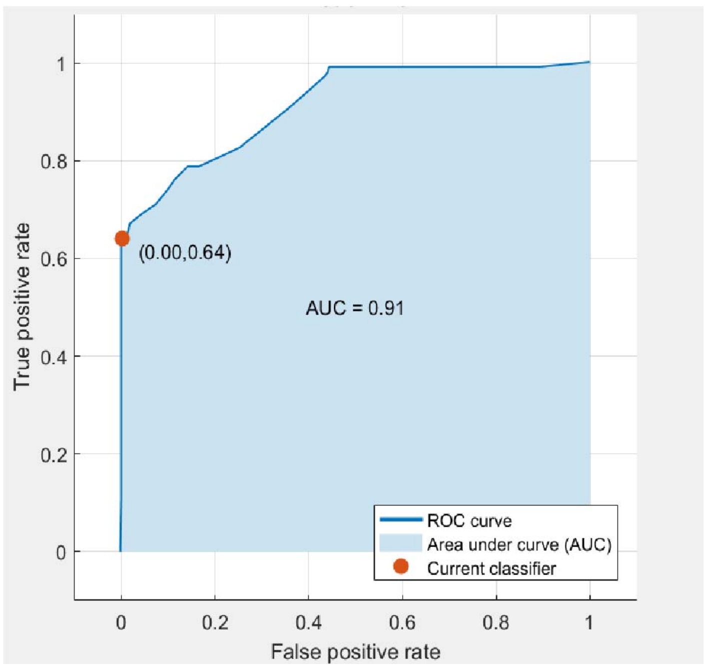


Fig. 9. The ROC curve for Ensemble Boosted and Bagged Trees classifiers.

VI. Conclusion

This research deals with extracting a set of features from the SQL statements and analysing them to detect whether they are injected with malicious commands or not. The proposed system resides between the database application and the database management system to examine the flow of queries and decides whether the queries are allowed to pass or not to the database. The performance of 23 different classifiers are evaluated in the work. Among these classifiers, the best five classifiers are selected (based on accuracy) to implement our proposed system. The proposed system is extensively tested, and the results show that both Ensemble Boosted and Bagged Trees classifiers provides the highest classification accuracy (93.8%). The area under the receiver operating characteristic curves for both are found to be above 0.9, which indicates that the system efficiently performs the detection. To make the proposed system more accessible for users, a GUI has been developed. The GUI provides a user-friendly and efficient interface for the users to enter the SQL statements and get them classified efficiently by our system. To enhance the efficiency of the system, more non-injected SQL statements need to be considered in the dataset and more features need to be studied and experimented in future. Our system is also scalable is a sense that any enhancement can be easily implemented with minor modification.

REFERENCES

[1] Internet World Stats: Usage and Population Statistics available at https://www.internetworldstats.com/stats.htm accessed on Juny 03,2019.

[2] ICT Facts and Fig. available at https://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFig.s2015 accessed on June 03,2019.

[3] National Security Agency, “Defending Against the Exploitation of SQL Vulnerabilities to Compromise a Network”, available at https://www.iad.gov/iad/library/ia-guidance/tech-briefs/defending-againsttheexploitation- of-sql-vulnerabilities-to.cfm acccessed on June 03,2019.

[4] Mark Churphy, “The Open WEB Application Security Project”, available at https://www.owasp.org/SQL_Injection accessed on June 03,2019.

[5] Gupta, M. K., Govil, Singh., "Static analysis approaches to detect SQL injection and cross site scripting vulnerabilities in WEB applications: A survey." *In Proceedings of the IEEE Conference on Recent Advances and Innovations in Engineering (ICRAIE)*, May 9-11, 2004, Jaipur, India, pp. 1-5.

[6] S.W.Boyd and AD.Keromyt , "SQLrand: Prevent ing SQL Inject ion At tacks," *In Procedings of the 2nd Applied Cryptography and Network Security (ACNS) Conference*, Yellow Mountain, Jun 8-11, 2004, pp. 292-302

[7] Kemalis, K. and T. Tzouramanis, “SQL-IDS: A Specification-based Approach for SQL injection Detection”, *Proceedings of the ACM symposium on Applied computing (SAC)*, Fortaleza, Ceará, Brazil , March 16-20, 2008, pp. 2153 2158.

[8] William G.J. Hal fond and Alessandro Orso, “A Classification of SQL injection attacks and countermeasures”, In Proceedimgs of the IEEE International Symposeum on Secure Software Engineering, March 2006 available at https://pdfs.semanticscholar.org/81a5/02b52485e52713ccab6d260f15871c2acdcb.pdf

[9] Aldwairi, M., Hasan, M., & Balbahaith, Z. (2017). Detection of drive-by download attacks using machine learning approach. *International Journal of Information Security and Privacy (IJISP)*, *11*(4), 16-28.

[10] C. Bockermann, M. Apel, and M. Meier, “Learning SQL for database intrusion detection using context-sensitive modelling (extended abstract),” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009, vol. 5587 LNCS, pp. 196–205.

[11] Jaskanwal Minhas,Raman Kumar,"Blocking of SQL Injection Attacks by Comparing Static and Dynamic Queries", *International Journal of Communication Networks and Information Security*, Vol.5, No.2, 2013, pp.1-9

[12] A. Joshi, and V. Geetha. , “SQL injection detection using machine learning”, *In Proceedings of the International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, Kanyakumari, India, July 10-11, 2014, pp. 1111-1115.

[13] K. Kamtuo, and C. Soomlek, “Machine learning for SQL injection prevention on server-side scripting”, *In Proceedings of the International Computer Science and Engineering Conference (ICSEC),* Chiang Mai, Thailand, December 14-17, 2016, pp. 1-6.

[14] Kumar, K., Jena, D., & Kumar, R. , “A novel approach to detect SQL injection in WEB applications”, *International Journal of Application or Innovation in Engineering & Management (IJAIEM)*, Vol. 2, No. 6, June 2013, pp. 37-48

[15] Uwagbole, S., J. Buchanan, and Lu Fan, “Applied machine learning predictive analytics to SQL injection attack detection and prevention”, *Procceding of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, Lisbob, Portugal, 8-12 May, 2017, pp.1087-1090

[16] Elith, J. Leathwick, J. R. & Hastie, T., “A working guide to boosted regression trees”, *Journal of Animal Ecology*, Vol. 77, No. 4, July 2008, pp. 802-813.

[17] Shah, S. A. A., Aziz, W., Arif, M. & Nadeem, M. S. A. , “Decision trees based classification of cardiotocograms using bagging approach”, *In Proceedings of the 2015 13th International Conference on Frontiers of Information Technology (FIT), Islamabad, Pakistan, December 14-16, 2015,* pp. 12-17

[18] Kok, J. N., Koronacki, J., de Mantaras, R. L., Matwin, S., and Mladenic, D., “Machine learning: ECML 2007”, In Proceedings of the 18th European Conference on Machine Learning, Warsaw, Poland, September 17-21, 2007

[19] Feraru, M. & Zbancioc, M. , “Speech emotion recognition for SROL database using weighted KNN algorithm”, *In Proceedings of the 2013 International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, Pitesti, Romania, *June 27-29, 2013.*

[20] Quinlan, J. R., “Induction of decision trees”, *Machine Learning*, Vol.1, No. 1, March 1986, pp. 81-106

[21] SQL Tutorial available at https://www.w3schools.com/sql/ accessed on June 4, 2019

[22] OWASP SecLists Project available at https://www.owasp.org/index.php/OWASP_SecLists_Project accessed on June 04,2019

[23] Dimitriadis, S. I., Liparas, D., and Tsolaki, M. N., “Random forest feature selection, fusion and ensemble strategy: Combining multiple morphological MRI measures to discriminate among healhy elderly, MCI, cMCI and alzheimer’s disease patients: From the alzheimer’s disease neuroimaging initiative (ADNI) database”. *Journal of neuroscience methods*, Vol. 302, May 2018, pp.14-23.

[24] Subasi, A., Alzahrani, S., Aljuhani, A., & Aljedani, M. , “Comparison of Decision Tree Algorithms for Spam E-mail Filtering”, *In Proceedings of the International Conference on Computer Applications & Information Security* (ICCAIS), Riyadh, Saudi Arabia, April 4-6, 2018, pp. 1-5

[25] Kalyankar, N. V. , “Effect of Training Set Size in Decision Tree Construction by using GATree and J48 Algorithm”, *In Proceedings of the World Congress on Engineering,* Vol. I, July 4-6, 2018, London available at http://www.iaeng.org/publication/WCE2018/WCE2018_pp193-196.pdf

[26] Goceri, E. (2018). Formulas behind deep learning success. In international conference on applied analysis and mathematical modeling. Istanbul, Turkey.

[27] Shen, D., Wu, G., & Suk, H. I. (2017). Deep learning in medical image analysis. *Annual review of biomedical engineering*, *19*, 221-248.

[28] Goceri, E. (2019). Diagnosis of Alzheimer's Disease with Sobolev Gradient Based Optimization and 3D Convolutional Neural Network. *International journal for numerical methods in biomedical engineering*, e3225.

[29] Goceri, E. (2019). Challenges and Recent Solutions for Image Segmentation in the Era of Deep Learning. *The 9th Int.Conf. on Image Processing Theory, Tools and Applications.*

[30] Goceri, E., & Gooya, A. (2018). On the importance of batch size for deep learning. In *international conference on mathematics. Istanbul, Turkey.*