SQL Injection Detection using Machine Learning

Anamika Joshi
Dept. of Information Technology
National Institute of Technology, Karnataka
Surathkal, India
anamika.iit89@gmail.com

Geetha V
Dept. of Information Technology
National Institute of Technology, Karnataka
Surathkal, India
geethav.nitk@gmail.com

***Abstract*— In the present world, the web is the firmest and most common medium of communication and business interchange. Every day, millions of data are loaded through various channels on the web by users and user input can be malicious. Therefore, security becomes a very important aspect of web applications. Since they are easily accessible, they are prone to many vulnerabilities which if neglected can cause harm. The attackers make use of these loopholes to gain unauthorized access by performing various illegal activities. SQL Injection is one such attack which is easy to perform but difficult to detect because of its varied types and channel. This may result in theft, leak of personal data or loss of property. In this paper we have analyzed the existing solutions to the problems such as AMNESIA [1] and SQLrand [3] and their limitations. We have devised a classifier for detection of SQL Injection attacks. The proposed classifier uses combination of Naïve Bayes machine learning algorithm and Role Based Access Control mechanism for detection. The proposed model is tested based on the test cases derived from the three SQLIA attacks: comments, union and tautology.**

Keywords—Web Security; Machine Learning; Web Application.

I. INTRODUCTION

A. SQL Injection Attack

Many organizations in the world use the web as a medium for their everyday transactions and businesses. The backbone of every web based application is Relational Data base which are operated by statements written in a special language called as Structured Query Language (SQL). Structured Query Language injection is an injection technique used to attack sites in which the attacker inserts SQL characters or keywords into a SQL statement via unrestricted user input parameters to modify the intended query's logic. Whenever a request is generated from the user end, a query is generated. The query contains user input which can be malicious. It is important to make our web applications learn that the user input comes from an external source and it can be malicious, so we need to process it before it actually gets executed. It is the responsibility of the programmer to write intelligent code which prevents any such intrusion. But due to negligence or ignorance, the user input is kept unprocessed which provides a room for the attacker to intrude in to the system. There are different types of SQL injection attacks, based on the type of user input channel, the response received from server, how server responses to the malicious input from the user, impact point, etc. Tautology attack, union based attack, error based attack, and blind SQL injection are some of the basic injection types. In the tautology type, the SQL query is modified such that the conditions always result to TRUE. In the union based attack, the queries are appended with the use of UNION statement such that one of the queries performs malicious task. The blind SQL injections function by asking the database a true or false question and checking whether valid page is returned or not or by using the time it took for the valid page to return as the answer to the question. There is another category of SQL injection called second order injection which is more complex and difficult to detect. But this category of injection attack is not considered in this paper to reduce the complexity.

Unauthorized login is an example of SQLIA. This task of login is accomplished by the use of tautology class of SQL injection. The SQL statement is modified in such a way that no matter what password is provided, login is made possible for the attacker. The WHERE statement is a constituent of the SQL statement which is practiced to describe conditions for pulling information as described in statement (1)

```
SELECT * FROM user_details WHERE id='user1' AND
Password = 'password1'; (1)
```

The intended purpose of statement (1) is to fetch data from user_details table whose id and password matches with the id and password provided as input by the user in the query. If at least one of the conditions of the WHERE statement is TRUE, it extracts all data in the table. After injection, the new query which is formed will always result in tautology. Hence, it becomes usable to access all data in a table through the use of malicious SQL statements that can change the structure of WHERE statements with the help or input given in statement (2)

```
'OR 1=1; --                                    (2)
```

When the malicious SQL statement (2) is injected to the non-malicious SQL statement (1) the query formed is shown in statement (3)

```
SELECT * FROM user_details WHERE id= ' OR 1 =1' -- '
AND password='password1';                      (3)
```

After the injection of statement (2) in the query the query becomes malicious. The WHERE statement in the SQL statement (3) always turns TRUE, because 1=1 is a tautology. So, the modified SQL statement (3) successfully extracts all

978-1-4799-4190-2/14/$31.00 ©2014 IEEE

data from user_details table despite id and password does not match. All the statements after the comment line ('AND password='pass';) is ignored because they are treated as comments.

Another case, in which an injection can occur, is in the INSERT query. Let's suppose that a web application has two fields, an integer ID and the comment string. So the INSERT query would be framed as shown in statement (4)

INSERT INTO COMMENT_TABLE VALUES (02,'I love to dance'); (4)

Consider an attacker enters the following malicious comment:

'); DELETE FROM user_detail; -- (5)

If no security check is done, then statement can result our single INSERT statement in to performing something which is not intended by the developer, causing exposure of important data. Now the inserted query looks as shown in statement (6)

INSERT INTO COMMENT_TABLE VALUES (02,'); DELETE FROM user_detail; -- '); (6)

If the INSERT statement is framed using the input given in statement, it would delete everything from your `users` table. This operation is not allowed to any other user except the administrator. Even a small negligence can cause serious trouble to the database.

II. REALTED WORK

The pattern match is one of the earliest and most common methods for detecting SQL Injection attack. AMNESIA [1] is one such tool which uses pattern match mechanism. In order to classify malicious query, the user input is matched with the registered patterns. If a mismatch is found, a possible attack is detected. It combines static analysis and run time monitoring. Dynamic Tainting [2] is also similar method based on pattern match. The user input is processed before being executed. Dynamic Tainting treats all user input as tainted data. When tainted data is used for any operation of critical importance such as retrieving password or accessing file, Dynamic Tainting checks it against the already registered patterns. A web code id detected malicious if tainted data don't match up with the registered patterns.

Malicious web code is also detected by Parsing Approach. This method parses user input along SQL grammar. SQL Check is one of Parsing Approach. In this method, each user input is taken as an argument query and analyzes the syntaxes of SQL queries by the SQL parser. If argument queries don't match up with registered data type, SQL Check detects it as malicious query. SQLrand [3] is also a parsing method for preventing SQL injection. It uses intermediate proxy that converts the SQL to its standard language. The queries injected by the attacker is caught and terminated by the database parser. SQL Guard [4] is another parsing method which is similar to SQL Check. It performs static checking by extracting some positions which send SQL queries like AMNESIA, and parses non-malicious SQL queries of each point. When SQL queries are generated, it is sent to the security aspect where it dynamically exchanges. SQL Guard detects a code malicious if a dynamically analyzed syntax doesn't match corresponding statically analyzed syntax.

Among other methods, one method is a hybrid of pattern matching and Role Based Access Control mechanism [5]. It is a 2- layered approach and for a query to be executed by the security aspect, it has to pass through both the layers of security. In the first layer, the query has to match with the static registered queries. For a match to occur, the query should be in compliance with the data type, position and type. If in any case, the first layer fails, RBAC provides a second layer of defense. RBAC is based on the fact that access is provided to the users is based on their role. If a particular operation or activity is not allowed for a particular user, then that operation will not be executed. Only if the dynamic query passes both the layers, the query is executed otherwise not.

Another method uses machine learning algorithms [6] for the detection of SQLIAs. In the learning phase, the features are extracted and rules are defined. The training set consists of malicious and non-malicious code. From the training set, the features are extracted by any feature extraction method like TF-IDF method. The classifier classifies the given code as malicious or non-malicious based on the feature vectors. The most crucial factor in this method is the selection of features. It is important to extract features efficiently in order to attain higher efficiency. One advantage by using machine learning algorithms is that, it gives leverage to broader range of SQL queries. Also, the accuracy of detection is raised and the rate false positives are decreased.

III. METHODOLOGY

In this section, we propose a method for detecting SQL injection.

Our approach detects the SQLIAs using the Naïve Bayes machine learning Algorithm. Machine learning methods provide highly accurate predictions on test data. They also give leverage to larger data sets which is a crucial factor in our case, because there are many different kinds of attack for which a particular pattern cannot be dug.

Naive Bayes classifier is a probabilistic model which assumes that the value of a particular feature is unrelated to the presence or absence of any other feature. We have built a classifier for classification of malicious and non-malicious query. The data set consists of malicious and non-malicious queries. The training set is a large collection of the given data set and the test set is a smaller set of such queries. Naïve Bayes algorithm detects an attack with the help of two probabilities, one is prior probability and other is posterior probability. The prior probability is calculated from the given training set. The count of malicious and non-malicious queries is calculated to calculate the prior probability. This is a very prior and rough estimation of the classifier.

978-1-4799-4190-2/14/$31.00 ©2014 IEEE

Next, we calculate the likelihood of each of malicious and non malicious queries separately. The likelihood of a query being malicious is calculated from number of features matching the given test case and the total no of malicious queries. The likelihood of a query being non malicious is calculated from number of features matching the given test case and the total no of non malicious queries.

The architecture of the system is shown in figure 1. The data set consists of malicious and non-malicious queries. From the data sets, the features are extracted and these features are used to generate rules for classification. The role of the user is given as input to the system it acts as one of the features for classification. Along with this, the test case which is the query to be classified is given as input. The tokenizer converts queries in to tokens of features and labels them as malicious or non-malicious. This input is sent to the classifier, which then finally classifies queries as malicious or non malicious query.

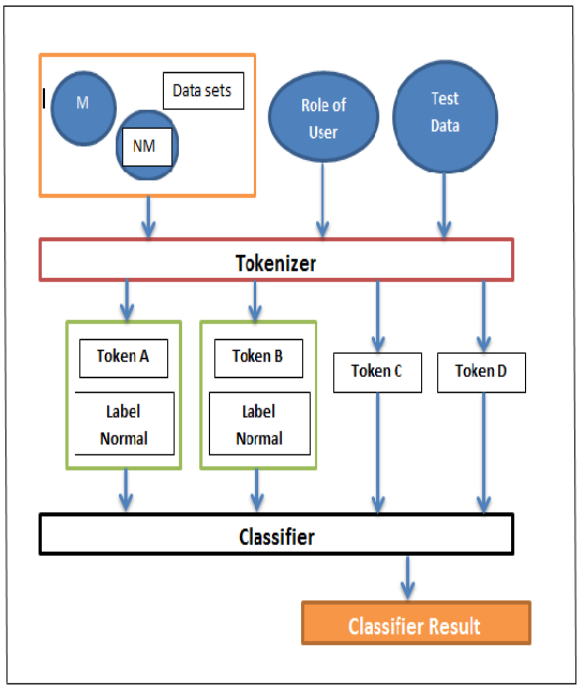

Fig. 1. Architecture of the proposed model

A. Feature Extraction

Feature extraction is the process of transforming the input data into the set of features in order to perform the desired task using this reduced representation instead of the full size input is called feature extraction. Each term is extracted from datasets by any extraction method. The most common method is the Blank Separation Method. The blank separation method extracts terms with respect to each blank space. Number of occurrence of each term is also calculated. For example, when the following document is separated:

'OR 'a'='a'--;

The output of the Blank Separation method is shown in Table I.

TABLE I. BLANK SEPARATION METHOD EXAMPLE

Name	*No of Terms*
'	5
OR	1
a	2
=	1
;	1
--	1

B. Tokenization Method

Tokenization is the process of breaking the query into meaningful elements called tokens. The list of tokens becomes input for further processing which is classification. Some of the tokens of SQLIAs are shown in the table II.

TABLE II. TOKENS OF SQLIA

Name	*Corresponding Token*
Single Line Comment	--
Multi line Comment	/**/
Operator	< > <= >= = == != << >> | & - + % ^
Logical Operator	NOT AND OR && ||
Punctuation	[] () , ; ' "
Literal	"string" 'string'
Keywords	SELECT UPDATE INSERT CREATE DROP ALTER RENAME
Numerals	Includes +, -, e and R
Role	Admin, Guest, Registered User

Now, when we separate the document, the Tokenizing Method gives the output as shown in Table III.

TABLE III. OUTPUT OF TOKENIZATION METHOD

Name	*No of Terms*
Punctuation	5
Logical Operator	1
Numeral	2
Operator	1
Single Line Comment	1

C. Role Based Access Control

There are different users of a web application and the privileges also differ depending on the type of user. Each user is bound to access only some part of the application where he/she is authorized to access.

In the classifier, the role of the user is taken as a parameter for evaluation. If a particular operation is not

allowed for a given user then a potential threat to the system is detected. The role and the operations allowed to the given user is stated in table IV.

TABLE IV. ROLE ACTIVITY TABLE

S No	*Role*	*Allowed Operations/ Activities*
1	Admin	View, Update, Delete all records
2	Registered User	View, Update, Modify only personal records.
3	Guest User	Only View

D. Proposed Algorithm

The algorithm for the detection is explained below. The symbols used in the algorithm are explained below:

Q_{test}= Dynamic query generated by the user which is to be classified as malicious or non malicious.

N_m= No of malicious Queries.

N_{nm}= No of non malicious Queries.

T_q= Total no of queries in the data set.

$P_{prior}(M)$ = Prior probability of malicious queries.

$P_{prior}(NM)$ = Prior probability of non malicious queries.

L_{qm}= Likelihood of Qtest given Malicious Queries.

L_{qnm}= Likelihood of Qtest given non malicious Queries

F_{match}= Number of features matching the test case.

$P_{post}(M)$= Posterior probability of Qtest being malicious query.

$P_{post}(NM)$= Posterior probability of Qtest being a non malicious query.

1) Pre-processing:

a) In the pre-processing step, the queries are broken down in to tokens as explained in the previous section.

b) They are then separated in to different categories according to the class it belongs.

2) Classification:

a) In the classification process, the dynamic query is classified as malicious or non malicious according to its features. The classification is done using the Naïve Bayes algorithm.

b) The number of malicious and non malicious queries is calculated and from this count the prior probability is calculated.

$$P_{prior}(M) = N_m / T_q$$
$$P_{prior}(NM) = N_{nm} / T_q$$

c) Having formulated our prior probability, we are now ready to classify a new query Qtest. For this likelihood has to be calculated, which is given by:

$$L_{qm} = F_{match} / N_m$$

$$L_{qnm} = F_{match} / N_{nm}$$

d) The final classification is produced by combining both sources of information, i.e., the prior and the likelihood, to form a posterior probability using the so-called Bayes' rule.

$$P_{post}(M) = P_{prior}(M) * L_{qm}$$
$$P_{post}(NM) = P_{prior}(NM) * L_{qnm}$$

e) Finally we classify Qtest as malicious or non-malicious depending on both prior probability and likelihood.

f) At last, the precision and recall is calculated along with accuracy of detection.

On the basis of this result, the given dynamic query Q_{test} is classified as either being malicious or non-malicious.

IV. RESULTS AND DISCUSSION

We implemented and evaluated the classifier for SQLIAs detection using the machine learning Naive-Bayes Algorithm. In the learning process, the training dataset is read by the application from text files and puts each data to the learning method of the classifier. The classifier generates feature vectors from received data by blank separation and tokenizing method and learns it by machine learning method. The feature vector also includes role of the user which is used for classification based on Role Based Access Control mechanism. In the classification process, the application reads the test dataset from text files and puts each data to the classification method of the classifier. From the generated feature vector classification is done. The classification results of the Naïve Bayes machine learning approach is analyzed by Precision and Recall in the application. We have used several dataset for evaluation as shown in table V.

TABLE V. DATA SETS FOR SQLIAS EVALUATION

	Training Data	*Test Data*
Normal Code	63	38
Malicious Code	51	26

The evaluation results of Naïve Bayes classification is described by Recall-Precision graphs as shown in figure2.

Also, in Table VI, the evaluation results described by Precision and Recall are shown. The maximum accuracy of Naïve Bayes Algorithm reaches to 93.3 percent. Additionally, both precision and recall are high. These results lead that the classifier by Naïve Bayes using Role Based Access Control mechanism as one of the parameters improves the detection rate and reduces the false positive rates.

TABLE VI. EVALUATION RESULTS OF SQLIAS

	Accuracy	*Precision*	*Recall*
Naïve-Bayes Algorithm	93.3	1.0	0.89

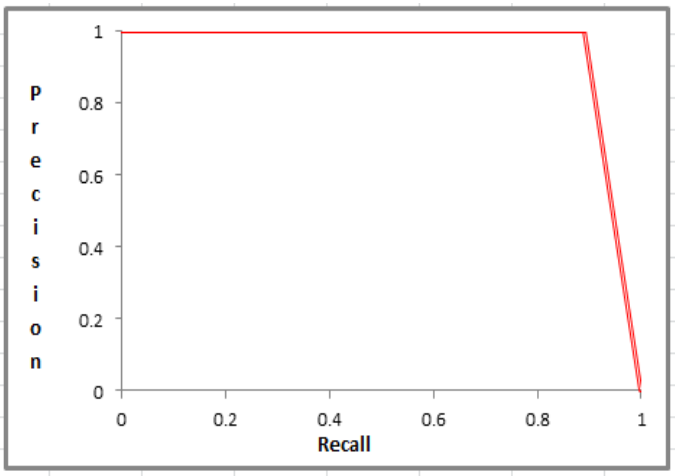

Fig. 2. The Precision and Recall Graph of Naïve Bayes Algorithm

V. CONCLUSION

In this paper, we have proposed a method for detection of SQL injection attack based on Naïve Bayes Machine Learning Algorithm combined with Role Based Access control mechanism. Our approach detects malicious queries with the help of classifier. The addition of another parameter for RBAC has increased the accuracy of detection and also reduced number of false positives. The proposed classifier classifies the test set with 93.3% accuracy. We think that our method should be tested against larger datasets to evaluate the efficiency. The proposed method can be enhanced for detection of other types of SQL injection attacks also by extracting features appropriately.

REFERENCES

[1] W.G.Halfond and Aorso, "AMNESIA Analysis and Monitoring for Neutralizing SQL-Injection Attacks," Proc. IEEE and ACM International Conference on Automatic Software Engineering (ASE 2005), Long Beach, CA, USA, Nov 2005.

[2] V.Haldar, D.Chandra, and M.Franz, "Dynamic Taint Propagation for Java," Proc. 2 1st Annual Computer Security Applications Conference, Dec 2005.

[3] S.W.Boyd and AD.Keromytis, "SQLrand: Preventing SQL Injection Attacks," Proc. the 2nd Applied Cryptography and Network Security (ACNS) Conference, pp. 292-302, Jun 2004.

[4] G.T.Buehrer, RW.Weide, and P.AG.Sivilotti, "Using Parse Tree Validation to Prevent SQL Injection Attacks," International Workshop on Software Engineering and Middleware (SEM), 2005.

[5] Evans Dogbe, Richard Millham, Prenitha Singh "A Combined Approach to Prevent SQL Injection Attacks," Science and Information Conference 2013 October 7-9, 2013, London, UK.

[6] Ryohei Komiya, Incheon Paik, Masayuki Hisada, "Classification of Malicious Web Code by Machine Learning," Awareness Science and Technology (iCAST), 2011 3rd International Conference on , vol., no., pp.406,411, 27-30 Sept. 2011.

[7] Z.Su and G.Wassermann "The Essence of Command Injection Attacks in Web Applications," The 33rd Annual Symposium on Principles of Programming Language (POPL 2006), Jan 2006.

[8] The Open Web Application Security Project (OWASP). The Ten Most Critical Web Application Security Risks 20 10. https://www.owasp.org/index.php/Top_10_2013.