SQL injection Detection for Secure Atomic and Molecular Database node for India

Mr. Parth Patel[1], Dr. Bhalamurugan Sivaraman[2]

[1] *M.E., Network Security, GTU PG School, Gujarat, Ahmedabad, India*
[2] *Reader, Physical Research Laboratory (PRL), Gujarat, Ahmedabad, India*

ABSTRACT

We aim at making a unified database for the entire atomic and molecular physics community in India. The data producers from different theoreticians, experimentalists and their groups in India are to be unified in one database node in PRL. For which, we are going to produce a database platform for the groups scattered in India to input their knowledge. Such completely different databases are going to be unified under one node and PRL is going to be the node. We are using MySQL database and Yii framework of PHP for this task. For which we need securing the database from cyber-attack like SQL injection. SQL injection attack has been top most threat among security threats against web application. Now a days, attackers use automated tools for attacks on web application. So, we have an inclination to tend to get focused on SQL injection attack detection to secure primary unified atomic and molecular database of India in PRL.

Keyword:-*Database, Secure, Sql-injection, SQLIA detection*

1. INTRODUCTION

Web-based applications have its own importance in each sector like social, commercial, government, and academic communities. Massive volume of knowledge is hold on within the back-end databases, those knowledge square measure terribly helpful for any organization. So it can be target for attackers. Security of web information should be important for web application developers. However, most web applications are developed while not due attention on security. [1]
Web security has so continuing to be basically necessary and very difficult. One major security issue of web application is SQL-injection attacks. SQL Injection Attack (SQLIA) has been top most security threat in web applications for last fifteen years. Nowadays, attackers use refined tools to launch machine- controlled injection attacks. [1] Consistent with Trust-Wave, "98% of web applications have one or a lot of security vulnerabilities. Among various security threats against web applications, SQLIA has been predominantly used for nearly fifteen years."[2]

SQL injection is comparatively easy and well understood technique for attacker. There are many types of SQL injection. Using this techniques, Attacker used to exploit database using dynamic SQL queries without correct validation and get the access of database and steal data. The classic example is "SELECT * FROM Users WHERE User-id = 1 or 1=1" injection, referred to as a tautological attack, this type of attack is happened due to true condition using OR. There square measure but many alternative types of injection attacks looking on the technique used and attack intention. Nowadays, attackers use refined tools or Bot-nets to mechanically discover vulnerable sites and launch mass injection attacks. [2]

Most existing solutions for detection these attacks using log analysis, and using pattern matching or machine learning ways that. Pattern matching ways are effective and dynamic. They are cannot discover new forms of attacks. Supervised machine learning ways can discover new attacks, but they have to admit associate degree offline coaching part.

2. BACKGROUND

In this section, we briefly discuss the popular types of SQL injection attack. We try to provide a clear understanding for each type of SQL injection attacks and their existing solutions.

2.1 Tautologies

In the tautology attack, “The attacker try to use a conditional query statement. Attacker uses ‘WHERE’ clause to inject and turn the condition into a tautology which is always true. The classic example is ‘SELECT * FROM Users WHERE Userid = 1 or 1=1’ injection, referred to as a tautological attack.” The result would be all the data in accounts table because the condition of the WHERE clause is always true. You can always use parameterized query and also restrict user to input special character like “’” (single quote) and other which are used to build query. [8]

2.2 Piggy-backed query

Piggy-backed query SQLIA is one kind of attack. The purpose of this type of attack are “to extract data, modify data-set, and execute remote commands and Denial of Service Attack (DOS)”. In this attack, Hacker try to inject additional queries than original. So the first query is true executed normally and subsequent queries are injected. “Strictly verify user inputs on user facet and avoiding multiple statement executions on a database by scanning all queries for delimiter ‘;’”. [8]

2.3 Union query

Union query SQLIA is one kind of attack. The purpose of this attack is “bypassing authentication and extract data”. This type of attack can be done by inserting a UNION query into a vulnerable parameter. “Strictly verify user inputs on user facet and block multiple query executions at a same time on database side”. [8]

2.4 Stored procedures

Stored procedure is one kind of SQLIA. The purpose of this attack are “privilege escalation, Dos and execute remote commands”. The signature is delimiter “;” for this attack, stored procedure keywords such as: “SHUTDOWN, exec” etc. “Strictly verify user inputs on user facet, using a low privileged account to run, executing stored procedures with a safe interface and giving proper roles and privileges to stored procedures are some countermeasures”. [8]

2.5 Illegal/logically incorrect queries

Illegal/logically incorrect queries SQLIA is one kind of attack. The purpose of this attack is “to detect injectable parameters, identify database, and extract data”. Using this type of attack, “Attacker try to get the information regarding database type and structure”. “Strictly verifying user inputs on user facet and stopping generated error messages from a given database for preventing this attack”. [8]

2.6 Inference

Inference is one kind of SQLIA. The purpose of this attack are “detect injectable parameters, identify schema and extract data”. This type of attack is performed on well secured database. There are two popular types of inference SQL injection attack discussed as follows. [8]

• “Inference blind SQL injection”.

• “Inference timing SQL injection”.

2.7 Alternate encoding

Alternate encoding SQL injection attack is one kind of SQLIA. The purpose of this attack is “evade detection”. In this attack, the attacker “injected encoded text to bypass defensive coding practices”. The possible signatures for this attack are: “exec(), Char(), ASCII(), BIN(), HEX(), UNHEX(), BASE64(), DEC(), ROT13(), etc.” “Strictly verify user inputs on user facet, for instance prohibiting any usage of meta-characters like Char (), and treating all meta-characters as traditional characters on information aspect will forestall the alternate encoding SQL injection attack”. [8]

3. RELATED WORK

SQL injection can be detect using machine learning algorithm and pattern matching algorithm. I found some research gap on existing papers.

In paper [1], Authors presented a “novel method to detect malicious queries using a twin Hidden Markov Model”. HMM is one of the machine learning method which is used by Author. In this method, there are two distinct phase “training phase and run time phase”. “The HMMs are first trained using known samples of genuine and injected queries.” “The model generated is then used at run-time for detecting injection attacks”. The scope is “System designed to work at the database firewall layer so it used for protecting multiple web applications hosted on a shared server”. The Research gap is “Author focused on ‘where’ clause query and Piggybacked query could not detect using this system”. [1]

In paper [2], “Melody Moh, Santhosh Pininti, Sindhusha Doddapaneni and Teng-Sheng Moh were proposed multistage log analysis system. This system used both pattern matching and machine learning methods”. It used logs generated by the application during attacks to effectively detect attacks. Author used “Kibana for pattern matching and Bayes Net model for machine learning”. Experiment results shown that “the two-stage system is able to detect significantly more SQL injections that a single-stage system”. The research gap is “Authors used Bayes Net model which is supervised learning method which is also rely on labeled data”.

In paper [3], Bhakti Maheshwarkar and Nidhi Maheshwarkar were proposed SQL Injection Union Query Attacks Prevention Using Tokenization Technique (SIUQAPTT) model. This model is useful for detect UNION type malicious query which is very harmful to the web application. Proposed model used to check SQL injection and if injection presents it directly terminate the query. The research gap is “Authors focused only UNION type of query”.

In paper [4], Nency Patel and Narendra Shekokar were proposed a technique that uses “static anomaly detection using modified AhoCorasick Pattern matching algorithm”. They are used two techniques, one is “SQLMAP tool and another is AIIDA-SQL (An Adaptive Intelligent Intrusion Detector Agent-SQL)”, which based on “neural network”. After detection they applied “the search algorithm to check whether the query is in the list or not”. The research gap is “SQLMAP is take 10-15 minutes to run one query. So, it is very time consuming and AIIDA-SQL tool cannot give accurate result. False positive rate is high”.

In paper [5], Anamika Joshi and Geetha V were proposed a method for detection of SQL injection attack based on “Naive Bayes Machine Learning Algorithm combined with Role Based Access control mechanism”. The approach detects malicious queries with the help of classifier. “The addition of another parameter for RBAC has been increased the accuracy of detection and also reduced number of false positives. The proposed classifier classifies the test set with 93.3% accuracy”. Author used three attacks comments, union and tautology. The research gap is “Future attack is not detecting using this method”. The Naive byes is rely on label data.

In this paper, Author presented most occurred SQL injection attacks in the web application like “Tautology, Piggy-back, Union, and Banner Grabbing”. Author given two methods to detect that type of SQL Injection attack. First method is “Brute-Force String Matching” used for checking the given input and matching with predefined string. And second method is “Longest Common Sub-sequence Method”. If the query contains undesirable actions then “LCS” method is return “the sub-sequence characters”. The research gap is, this two methods are useful for given attack only and this methods are not useful for new future SQL injection attacks. [10]

4. MOTIVATION

New security vulnerabilities square measure discovered daily in today's system, networking, web application and mobile application. However web application became primary targets of cyber-attacks. Analysis of the “National

Vulnerability database (NVD)" maintained by the "National Institute of standard and Technology (NIST)" shows "the fast increase of vulnerabilities that occur principally in web primarily based application (SQL injection) as percent of the whole vulnerabilities". [17]

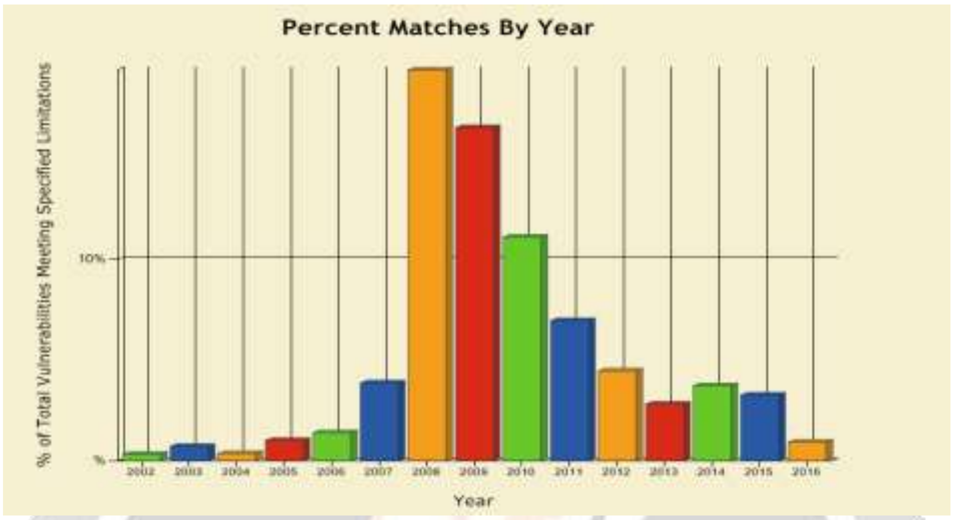


Fig-1 PROPOSED SYSTEM NIST Report [17]

"Open Web Application Security Project (OWASP)" is an organization that provides "unbiased and practical, cost-effective information" about computer and Web applications. As per last two report of OWASP which have been published in 2010 and 2013, SQL injection have been top most vulnerabilities in the cyber world. [18]

There are some existing solution available for detect and prevent SQL injection vulnerabilities. They mostly used pattern matching algorithm and machine learning method. Pattern matching algorithm is only for the available pattern. They did not work for finding new vulnerabilities. Thus still now they did not work 100%. False positive rate is high. So we will try to decrease the false positive rate and try some different solutions.

5. PROBLEM STATEMENT

Existing system were used pattern matching and machine learning methods to detect SQL injection attack. Pattern matching algorithm can detect only known patterns but they did not work for new SQL injection attacks means future attacks. [4]

In Machine learning, existing system used supervised learning method. This method is also rely on training data means labeled data. This method would not be detect future attacks. [1][5]

Existing system is apply tokenization technique but they are only focused on UNION type queries. So, this method is also not detecting future attacks if any without UNION query. [3]

All existing system were used one method either pattern matching or machine learning so accuracy did not maintain in all system. [5]

We got focused on how to detect future attacks of SQL injection and find best optimum solution for that with accuracy.

6. PROPOSED SYSTEM

In this chapter, I provide system work-flow and phases of the systems.

6.1 System Work flow

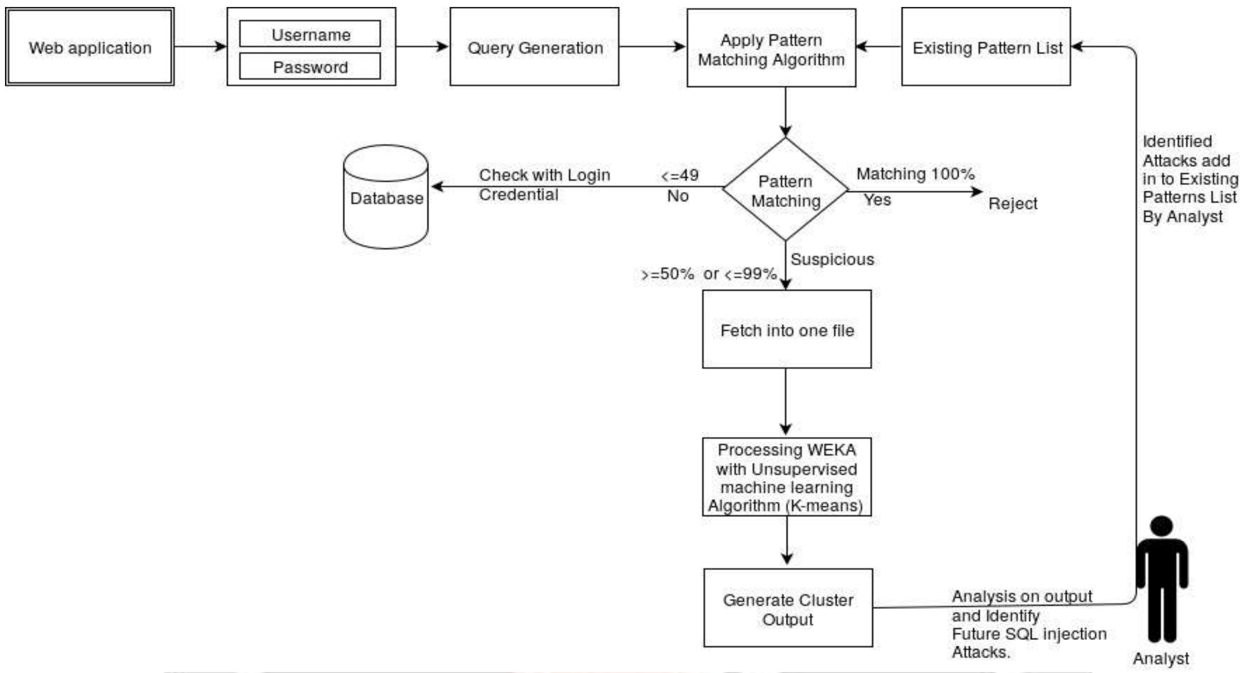

Web application
Username
Password
Query Generation
Apply Pattern Matching Algorithm
Existing Pattern List
Identified Attacks add in to Existing Patterns List By Analyst
Database
Check with Login Credential
<=49
No
Pattern Matching
Matching 100%
Yes
Reject
Suspicious
>=50% or <=99%
Fetch into one file
Processing WEKA with Unsupervised machine learning Algorithm (K-means)
Generate Cluster Output
Analysis on output and Identify Future SQL injection Attacks.
Analyst

Fig-2 System Work flow

6.2 Phases of the System

Query Generation Phase: In this phase, I fetched the user's "login request" and stored into session variable user-name and password. Using this variable we made query. This query is being input for pattern matching algorithm.

Pattern Matching Algorithm Phase: In this phase, we apply Pattern Matching Algorithm for detection of SQL injection. Generated Queries used for "input" in this algorithm.

The steps of the algorithm are following

```
1: Procedure SPMA (Query, SPL [])
INPUT: Query <—User Generated Query
SPL [] <—Existing Pattern List with m Anomaly Pattern
2: For j=1 to m do
3: Compare the Query with all Existing Patterns of list
4: Find, Anomaly score = Matching score * 100 / SPL length
5: IF (Anomaly score == 100%)
Then Query is rejected
Break;
6: IF (Anomaly score >= 50 AND Anomaly score <= 99)
Then
7: Query is Suspicious and fetched into one file
```

```
Break;

8: END FOR LOOP

9: IF (Anomaly score <= 49)

10: then, Query is Genuine and checked with database if it is true then logging otherwise give message for try with true username and password

END PROCEDURE
```

Processing for WEKA Phase: In this phase, we are using “generated suspicious query log” for “WEKA process” using k-means algorithm. K-means is “unsupervised machine learning algorithm”. Unsupervised machine algorithm have no labeled data so, it is useful for detecting new attacks or future attacks using k clusters. We used CSV file for “input”. So first we need to fill CSV file named sqllog.csv manually. We used SQL keywords as attribute. We used simple k-means for detecting SQL injection. If Analyst will find any attack using k-means, then it will be added this new pattern in to Existing Pattern Matching List.

7. IMPLEMENTATION

Implementation has been done in three modules.

• Implementation of Website and Database for Atomic and Molecular Physics Community

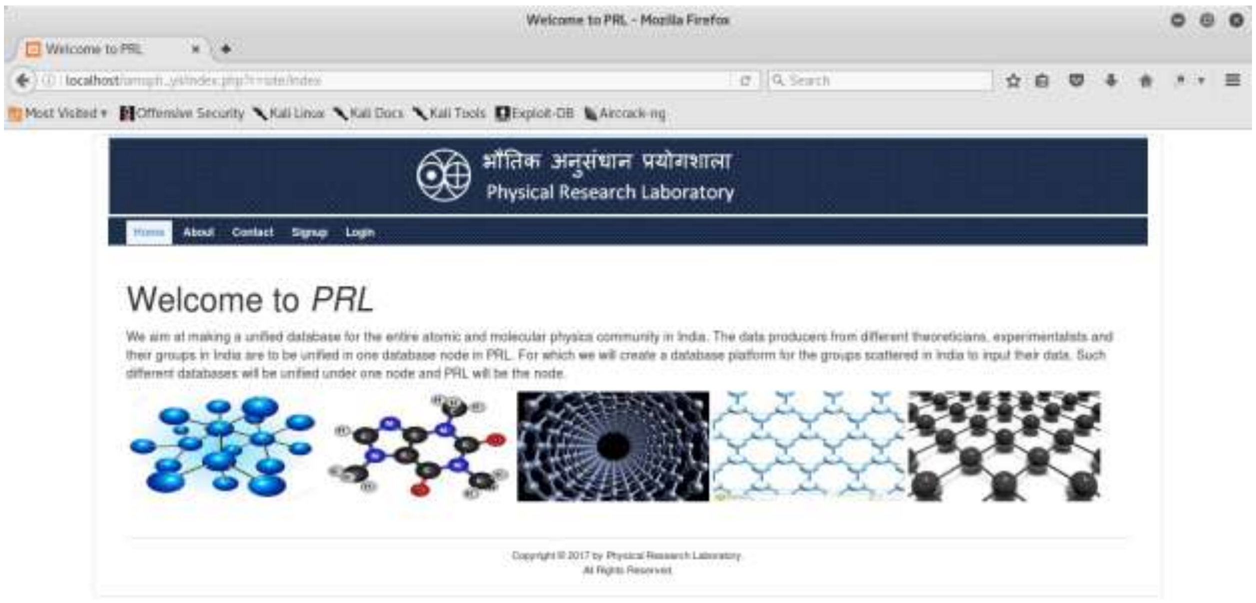
• Implementation of Pattern Matching Algorithm

• WEKA- Implementation of K-means Algorithm

7.1 Implementation of Website and Database for Atomic and Molecular Physics Community

I have been implemented a website and database for Atomic and Molecular Physics Community. Web implementation done using PHP language and Yii framework. I used MySQL Database.

IJARIIE



Welcome to *PRL*

We aim at making a unified database for the entire atomic and molecular physics community in India. The data producers from different theoreticians, experimentalists and their groups in India are to be unified in one database node in PRL. For which we will create a database platform for the groups scattered in India to input their data. Such different databases will be unified under one node and PRL will be the node.

Copyright © 2017 by Physical Research Laboratory
All Rights Reserved.

Fig- 3 Home page of Website

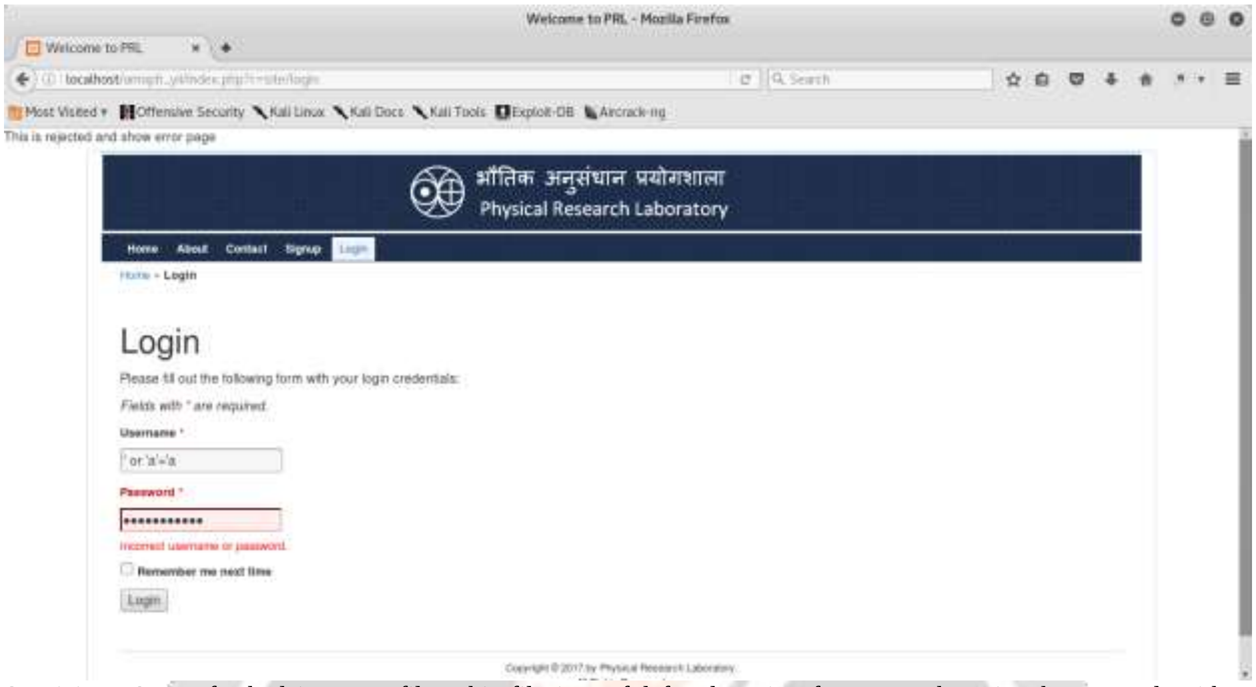
7.2 Implementation of Pattern Matching Algorithm

In Proposed System, I have been given algorithm for Pattern Matching Algorithm. Using this Algorithm, I found three types of query.

• Rejected Query (Matching 100% with Existing Patterns List)

• Suspicious Query (Matching more than 50% and less than 99% with Existing Patterns List)

• Accepted Query (Matching less than 49% with Existing Patterns List)


This is rejected and show error page

भौतिक अनुसंधान प्रयोगशाला
Physical Research Laboratory

Home » Login

Login

Please fill out the following form with your login credentials:

*Fields with * are required.*

Username *: ' or 'a'='a

Password *: ••••••••••••

Incorrect username or password.

☐ Remember me next time

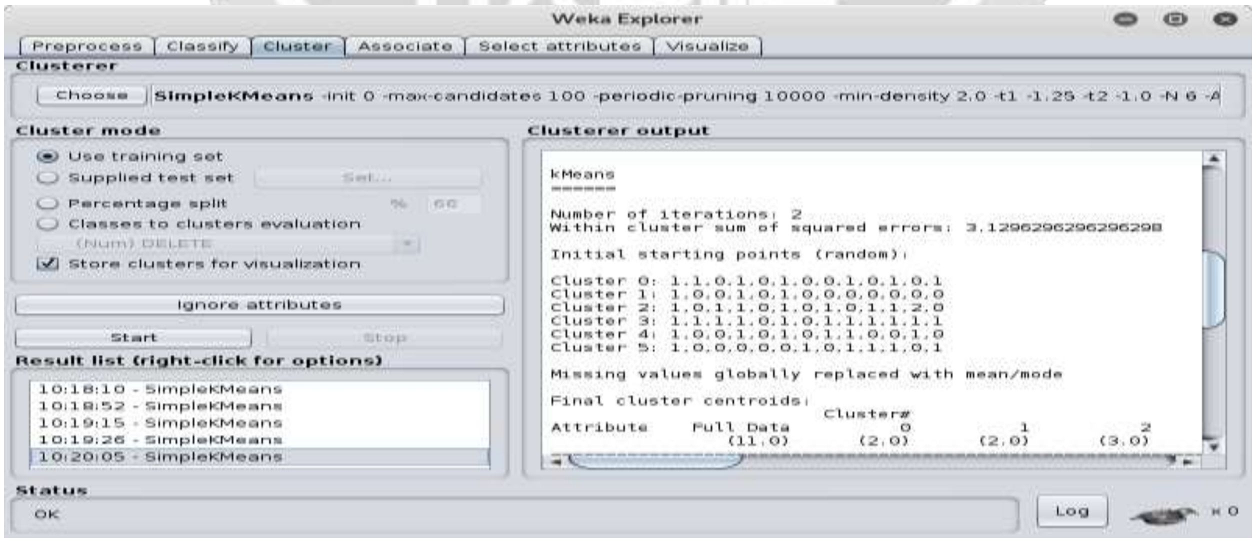
Copyright © 2017 by Physical Research Laboratory

Suspicious Query fetched into one file. This file is useful for detecting future attacks using k-means algorithm. Accepted Query checked with Database login credential. If it is true then user can logging otherwise they will be get error “Username or Password is wrong”.

Fig. 4. Demo of SQL injection Detection using Pattern Matching (Rejected Query)

7.3 WEKA-Implementation of K-means Algorithm

WEKA stands for “Waikato Environment for Knowledge Analysis”. WEKA is collection of machine learning algorithm. I used Simple k-means algorithm for finding future attacks of SQL injection using some SQL keywords. For that, I made sqllog.csv and manually filled value according to suspicious query which fetched into one file. After analysis, I will find future attacks using this algorithm.



Clusterer: SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 6 -A

Cluster mode: (●) Use training set; ○ Supplied test set; ○ Percentage split % 66; ○ Classes to clusters evaluation (Num) DELETE; ☑ Store clusters for visualization

Result list (right-click for options):

```
10:18:10 - SimpleKMeans
10:18:52 - SimpleKMeans
10:19:15 - SimpleKMeans
10:19:26 - SimpleKMeans
10:20:05 - SimpleKMeans
```

Clusterer output:

```
kMeans
======

Number of iterations: 2
Within cluster sum of squared errors: 3.1296296296296298

Initial starting points (random):

Cluster 0: 1.1.0.1.0.1.0.0.1.0.1.0.1
Cluster 1: 1.0.0.1.0.1.0.0.0.0.0.0.0
Cluster 2: 1.0.1.1.0.1.0.1.0.1.1.2.0
Cluster 3: 1.1.1.1.0.1.0.1.1.1.1.1.1
Cluster 4: 1.0.0.1.0.1.0.1.1.0.0.1.0
Cluster 5: 1.0.0.0.0.0.1.0.1.1.1.0.1

Missing values globally replaced with mean/mode

Final cluster centroids:
                                 Cluster#
Attribute     Full Data          0            1            2
                (11.0)       (2.0)        (2.0)        (3.0)
```

Status: OK

Fig-5 Clustered output

8. RESULT ANALYSIS

In this chapter, we discuss performance obtained by combining pattern matching and unsupervised machine learning k-means methods.

8.1 Experimental Results

We try to do SQL injection attack on our web application and detect using our implemented model. We used two methods pattern matching and unsupervised machine learning k-means method.

Confusion Matrix Table				
		PREDICTED		
		Genuine	Injected	Total
ACTUAL	Genuine	490(TN)	6(FP)	496(N)
	Injected	10(FN)	470(TP)	480(P)
Total		976		

Table 1- CONFUSION MATRIX TABLE

Experimental Evaluation		
Function	Formula	Result
Precision	TP/(TP+FP)	98.74 %
Recall	TP/(TP+FN)	97.92 %
Sensitivity(TPR)	TP/(TP+FN)	97.92 %
Specificity(TNR)	TN/(FP+TN)	98.79 %
1-Specificity(FPR)	FP/(FP+TN)	1.21 %
Accuracy	(TP+TN)/(P+N)	98.36 %
F1-Score	2TP/(2TP+FP+FN)	98.32 %

Table 2- RESULT ANALYSIS TABLE

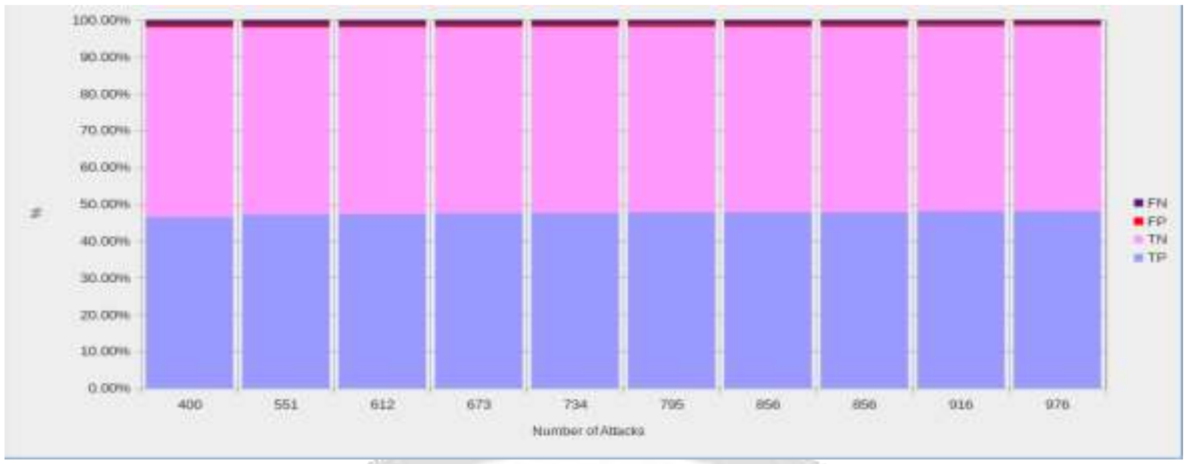


Fig-6 Ratio of TP, TN, FP, FN vs. Number of Attacks

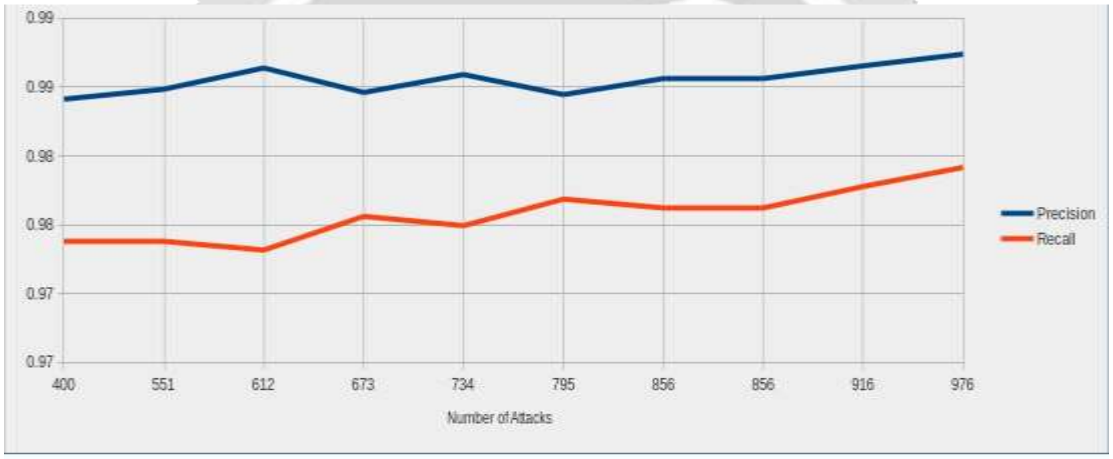


Fig-7 Ratio of Precision, Recall vs. Number of Attacks

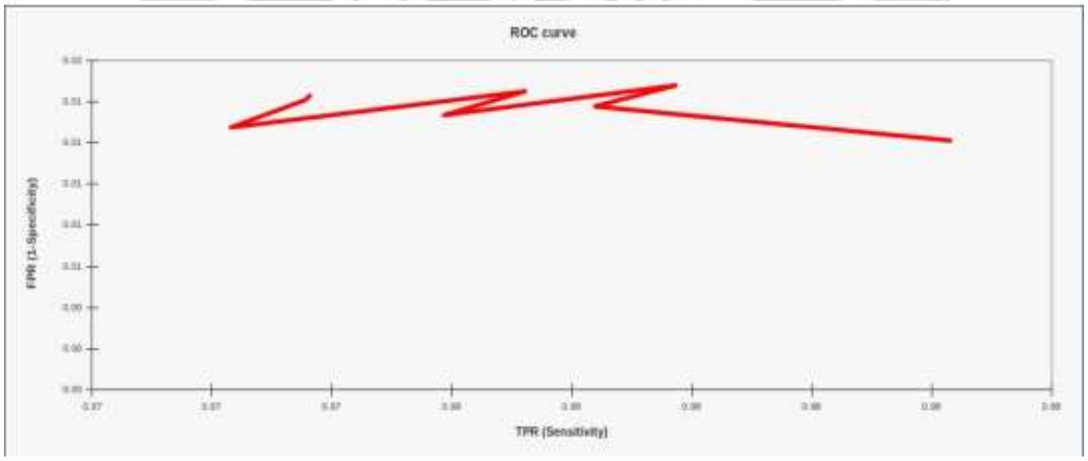


Fig-8 ROC curve

8.2 Processing Overhead

- Processing Overhead means total time taken by system to run one query and get result. “Processing Overhead added by the system at run-time consist of query generation and pattern matching evaluation and decision”.

- We cannot consider k-means algorithm in processing overhead because it used manually by us and not relevant to end-user activity to perform task on our web application, therefore ignored.

- The average processing time for “query generation” and “pattern matching evaluation and decision” were measured and shown in Table-III

Processing Time per Query		
Component	Genuine	Injected
Query Generation	1.1123 ms	2.1424 ms
Pattern Matching Evaluation and Decision	4.3245 ms	6.4378 ms
Total	5.4368 ms	8.5802 ms

Table-3 PROCESSING OVERHEAD TABLE

8.3 Comparison with Existing System

In this section, we compare our system based on Accuracy and Sensitivity with existing system which is also two stage system means they also used pattern matching as well as machine learning but they used supervised machine learning method.

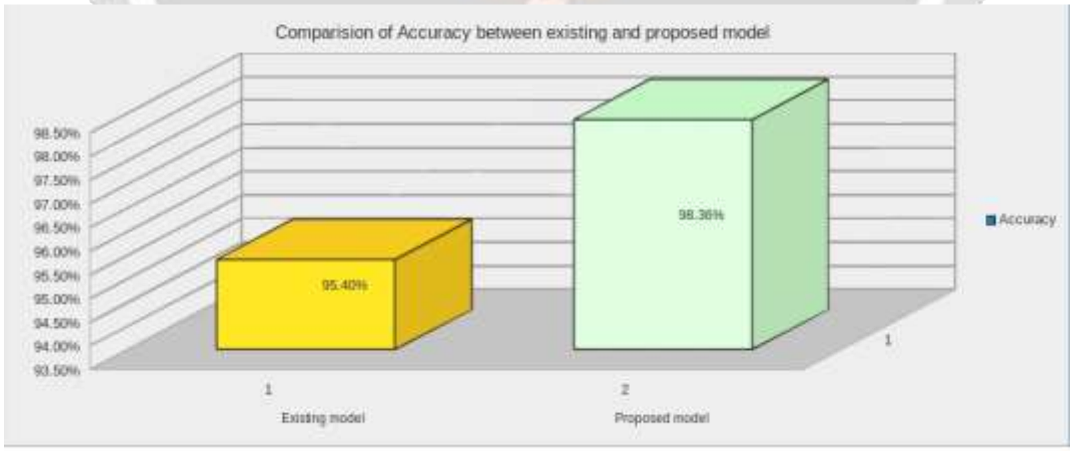


Fig-9 Comparison of Accuracy between Existing model and proposed model

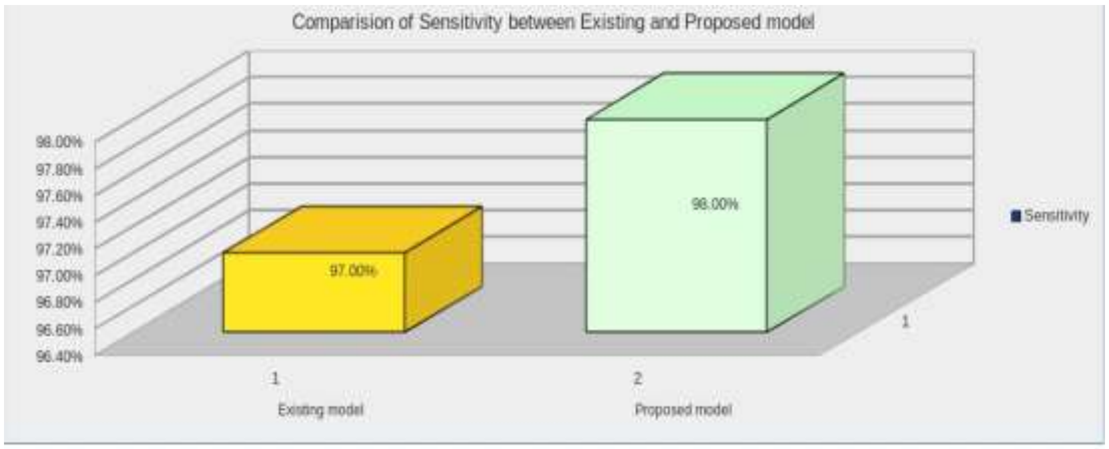


Fig-10 Comparison of Sensitivity between Existing model and proposed model

9. CONCLUSION AND FUTURE WORK

9.1 Conclusion

We provides brief introduction for detection of SQL injection with a review study, several works have been proposed and all of them have same goal to detect SQL injection attack. This survey shows that SQL injection attack is major threat in last 10-15 years. Using SQL injection attack, Attacker got major benefits of useful data. So there is need to employing an efficient method for detection of future SQL injection attacks.

We used "Pattern matching algorithm" for detecting existing SQL injection attacks and "unsupervised machine learning kmeans algorithm" for detecting future SQL injection attacks. We covered both existing and future SQL injection attacks. So, we got higher accuracy as well as sensitivity for detecting SQL injection attack than existing system.

9.2 Future Work

For future work, we need to increase the attributes for detecting future SQL injection attacks in k-means method.

REFERENCES

[1] A. S. Debabrata Kar, Khushboo Agarwal and S. Panigrahi, "Detection of sql injection attacks using hidden markov model," *IEEE*, 2016.

[2] S. D. Melody Moh, Santhosh Pininti and T.-S. Moh, "Detecting web attacks using multi-stage log analysis," *IEEE*, 2016.

[3] B. Maheshwarkar and N. Maheshwarkar, "Sql injection union query attacks prevention using tokenization technique," *ACM*, 2016.

[4] N. Patel and N. Shekokar, "Implementation of pattern matching algorithm to defend sqlia," *ELSEVIER*, vol. 45, pp. 453–459, 2015.

[5] A. Joshi and G. V, "Sql injection detection using machine learning," *IEEE*, 2014.

[6] N. M. Sheykhkanloo, "Employing neural networks for the detection of sql injection attack," *ACM*, 2014.

[7] T. Oosawa and T. Matsuda, "Sql injection attack detection method using the approximation function of zeta distribution," *IEEE*, 2014.

[8] N. M. Sheykhkanloo, "Sql-ids: Evaluation of sqli attack detection and classification based on machine learning techniques," *ACM*, 2015.

[9] M. R. I. Hussein Alnabulsi and Q. Mamun, "Detecting sql injection attacks using snort ids," *IEEE*, 2014.

[10] R. M. Anitha .V, Supha Lakshmi .A and S.K, "Detecting various sql injection vulnerabilities using string matching and lcs method," *IEEE*, 2014.

[11] M. A. Abd Jalil, Fakariah Hani and T. F. A. Rahman, "A method for web application vulnerabilities detection by using boyer-moore string matching algorithm," *ELSEVIER*, vol. 72, pp. 112–121, 2015.

[12] S. Y. Inyong Lee, Soonki Jeong and J. Moon, "A novel method for sql injection attack detection based on removing sql query attribute values," *ELSEVIER*, vol. 55, pp. 58–68, 2011.

[13] M. S. Aliero, "A component based sql injection vulnerability detection tool," *IEEE*, 2015.

[14] M. Ceccato, "Sofia: An automated security oracle for black-box testing of sql-injection vulnerabilities," *ACM*, 2016.

[15] C. D. N. Dennis Appelt and L. C. Briand, "Automated testing for sql injection vulnerabilities: Aninput mutation approach," *ACM*, 2014.

[16] N. Kaur and P. kaur, "Mitigation of sql injection attacks using threat modeling," *ACM*, vol. 39.

[17] ""National vulnerabilities database"." https://web.nvd.nist.gov/view/vuln/search. Accessed: 2016-10-10.

[18] "Top 10 2013-top 10." https://www.owasp.org/index.php/Top$_1$0$_2$013$-$ $Top_1 0.Accessed{:}$ $2016-10-10$.