An Emulation of SQLIA Detection, Prevention Using Pattern Based and Supervised Machine Learning Model

Mr.Bhushan R.Thakare[1], Prof.Sharad M.Rokade[2], Prof.Devidas S.Thosar [3]
Department of Computer Engineering
[1,2,3] Sir Visvesvaraya Institute of Technology, Nashik, India

Abstract- *Most of the web applications are associated with database as back-end so there are promises of SQL injection attacks (SQLIA) on it. Even SQLIA is among top ten attacks according to Open Web Application Security Project (OWASP) but still methods are not able to give appropriate solution to this problem. Numbers of methods are also discovered to overcome this attack, but which measure is more appropriate and can also provide fast access to application without conceding the security is also a major concern. Some existing approaches are good in security but they are not effective to handle large user's requests. In SQLIA, hacker can obtain the advantage of poor input authentication and weak coded web application. Due to the successful implementation of a SQLIA, integrity and confidentiality of data are lost which results in the degrading organization's market value. In proposed framework given query will pass over two phases. The two phases are pattern based and machine learning based. As well database firewall will be there for checking incoming SQL queries. If the given query passes all this phases, then only query is valid otherwise there is attack. The approach usage a patterns and machine learning models in which value is entered for every field is checked for SQLIA by parsing it. Proposed model also detected static and dynamic attacks.*

Keywords- SQLIA, OWASP, Machine learning, SQL Injection, web application, Static Attacks, Dynamic Attacks

I. INTRODUCTION

With the rapid growth of internet, web applications are becoming progressively popular and web application database gaining more and more value. Preventing attacks become an important for developers to protect database. Here, the main point is what was attacked or from where and find a way to prevent it. SQL injection attack is a code injection practise which is commonly used to attack the trusted data by injecting malicious SQL queries as input in entry field for execution. [2]

SQL injection attack (SQLIA) pose a serious security risk to the database driven web applications. SQLIAs are possible due insufficient input validation or improper creation of SQL statements in web applications. According to the OWASP and CWE/SANS, SQLIAs are the number one threat to web applications. This type of attack does not use any system resources but through nasty actions the hacker is able to retrieve/insert unauthorized data from/into the primary database.[6]

There are numerous types of SQLIA's and each has different approach, intent and significances. The different types of attacks are generally performed by the attacker; many of them are used together or successively, depending on the specific goals of the attacker. The most common types of SQLIAs include, tautology, illegal/logically incorrect queries, union, piggy-backed queries, and stored procedures.[5]

The main aim of this paper is to present a combined method which is a combination of static and dynamic methods, for the detection and prevention of SQLIA. The method is designed to decrease the vulnerability of the web applications.

II. LITERATURE SURVEY

A. Detection of SQL Injection Attacks by Removing the Parameter Values of SQL Query

In this paper method of query processing and parameter removal are collectively used to give a method which detects the difference between the original SQL query and the modified i.e. injected SQL query via the parameters. Evolution of methods from web application generation to web application security is making the web applications more secure and robust. Removing the parameters and comparing them with the original query structure will improve the performance of the system. [1]

B. A second-order SQL injection detection method

The first-order SQL injection load attack payload directly to the SQL query, but the attack method of second-order SQL injection is divided into 2 stages, namely the stage

of loading attack payload to database or file system and the stage of building the SQL query statement with the attack payload from database or file system.

For example, the common function of user registration and user information alteration in Web application can carry out an second-order SQL injection attack, the user registration function loads the attack payload to the database, and then the user information modification function extract the attack payload from database to construct a SQL database query, which will cause a second-order SQL injection.[3]

C. An Emulation of SQL Injection Disclosure &Deterrence

The system offers a methodology in which the input query is first divided using Heisenberg analysis and each block of the query is stored in the database. Heisenberg analysis [4] helps to divide the SQL query into each block and then compare the same with the constrained predefined query.

For example if a query like "SELECT * FROM USER WHERE USRNM='uname' AND PASS='pwd';" is partitioned using Heisenberg analysis then output occurs: SELECT","*","FROM","USER","WHERE","USRNM","=", "'uname'","AND","PASS", When a user try to login, while he enters the username and password the new query with entered values are divided into blocks and each block is confirmed with the old constrained query blocks. In order to perform verification, predefined query is divided, stored and matched with the constrained user query using Heisenberg analysis, so that those user queries following that matching properties are only granted access.

After confirmation if each query block matches, then the login is successfully done. Otherwise, error detection is done and access is denied. If the query does not match with any query block, then the login is blocked.

D. SQL Injection: - Study and Augmentation

There are various methods used to insert malicious code into database. Some of the common techniques used are:-

1) Through user input: In this, the attacker inserts nasty code by producing user input. The attacker simply inserts the code with typical SQL Query which goes unseen by the DBA resulting in illegal access of data.

2) Second order Injection: In this attack, the nasty codes are planted into the databases to indirectly trigger a SQLIA at some later time.

E. TYPES OF SQLIA

In the literature, SQLIAs have been categorized based on the way the attack is performed and the purpose of the attack. In this section, we summarize the most common types of attacks associated to our research. For each attack type, we present attack aims, details of the attack, an attack example, and references to research papers that discuss the attack technique in more detail.[2][5]. The attack can be static or dynamic.

Following queries present the static SQLIA attacks: Static or Embedded SQLIAs are SQL statements in an application that do not change at runtime.

a) Tautology

The major objective of such attacks is to insert nasty code into more than one conditional statement that they always authenticate to true. It is mostly used to bypass authentication. The aim of this attack is injection of more than one conditional statement so that their assessment is always true.

A generally used statement which always returns true is 1=1. This means by appending "or 1= 1" to the original query statement, the result of the query will be always true. For example, the below query is designed to return the record of the employee whose employee id is 9890. SELECT * from employee WHERE emp_id =9890;
After injecting "1=1", the above query will look like:
SELECT * from employee WHERE emp_id =9890 or 1=1;[7]

b) Illegal/Logically Incorrect Queries.

The intent of this attack is to detect injectable Parameters, perform database finger-printing, and extract trusted data. This type of attack is based on writing query statement that produces error messages. When a query is rejected, an error message is returned from the database including useful debugging information which helps the attacker. In the example below, attacker's goal is to cause a type conversion error that can reveal relevant data:[4][6]

Query: SELECT * FROM user WHERE id='110' AND password='1234' AND CONVERT(char, no) --';

c) Union Query

This type of attack is mostly used to bypass the Authentication process [7] and to extract data by inserting the union operator to the normal query. The following example in which the SQLIAs could taint the text "' UNION as:

SELECT * from accounts WHERE id='12'UNION select * from credit_card WHERE user='admin'--' and pass='pass'

In this example the second query is nasty because the text following '—'is disregarded as it becomes comment for the SQL Parser. However, if the query is executed, the attacker receives the credit card information.

d) Piggy-Backed Query

The purpose of this kind of attack is retrieval of information and DoS attack. This attack inserts nasty SQL queries into a normal SQL query.[2] It is possible because many SQL queries can be handled if the operator ''';'' is added after each query. As an example, consider the following query:

SELECT customer_info from accounts WHERE login_id = ''admin'' AND pass = '1223'; DELETE FROM accounts WHERE CustomerName ='Akshada';

After executing the first query the query interpreter sees the ';' and thus executes the second query with the first query. Since the second query is malicious, it will delete all the data of the customer 'Akshada'.

e) Stored Procedures

A stored procedure is a method in which a user can store his own function that can be used as needed. In this type of attack, the attacker tries to execute stored procedures present in the database with malicious inputs. An example is shown query:[7]

CREATE PROCEDURE DBO @userName varchar2, @pass varchar2,AS EXEC ("SELECT * FROM user WHERE id='" + @userName + "' and password='" + @password + "'); GO

The authorized/unauthorized use of stored procedure returns true/false. If the nesty input SHUTDOWN; --" for username or password on Stored Procedure. The Stored Procedure generates the following query statement which shut down the system.
SELECT userName FROM user_Table WHERE userName = user_1 AND password=' '; SHUTDOWN;

F. DYNAMIC ATTACKS

The attack done on dynamic query is stated as dynamic attacks.[7] Dynamic SQL is SQL statements that are built at runtime.You can use dynamic SQL for the purpose of execute dynamic queries, whose full text is not known till the runtime.

Consider the following query: SELECT * FROM Customers WHERE Country=$_POST['country'] AND City=$_POST['city'];

The $_POST variable is used to collect values from a form with method="post".
Information fixed from a form with the POST method is invisible to others and has no bounds on the amount of information to send.[7]

Developers often use inputs in SQL statements without any checks. This is the most common and serious programming fault. For example, the following PHP code represents such a dynamic SQL statement.

$query = "SELECT info FROM userdata WHERE name = '$_GET["name"]' AND pwd = '$_GET["pwd"]'"; $_GET is the built in PHP super global array variable that is used to get values submitted via HTTP GET method.[7]

The array variable can be accessed from any script in the program; it has a global scope. This method shows the form values in the URL.

Database Injection

If the attacker has the ability to manipulate queries which are sent to the database, then he's able to inject a terminating character too [6]. Consider a following php script (database.php) which takes input from user and retrieves information from the database as follows:

```
<?php

$ID = $_GET['id'];
$connect = mysql_connect("localhost","raj","rt123");
if (!$connect)

{
die('Could not connect: ' . mysql_error());
}
mysql_select_db("my_db", $connect);
```

```
$resultq = mysql_query("SELECT fname FROM person WHERE id = $ID limit1;");

$row = mysql_fetch_array($resultq) or die(mysql_error());
echo $resultq;

mysql_close($connect);
?>
```

Let's assume that the attacker has sent via the GET method the following data stored in variable $ID, consider the above URL:http://localhost:8888/database.php?id= 1 or 1=1; # The effect of above GET request is that the interpretation of the query will be stopped at the terminating character. In the end the final query form is:SELECT fname FROM person WHERE id = 1 or 1=1; # limit 1; After the # character everything will be rejected by the database including "limit 1",so only the last column "fname" with all its records will be received as a query response.

III. PROPOSED METHODOLOGY

Before presenting the new method, let's appearance at the architectural layers of web applications as shown in Figure 1 below.

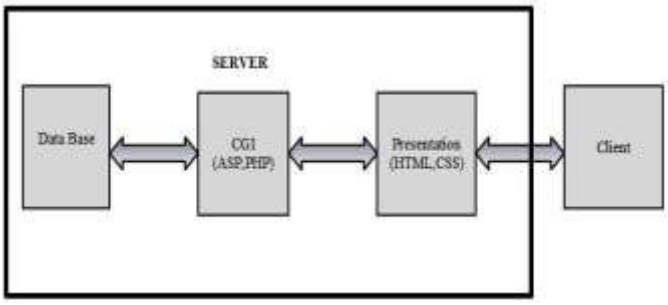

Fig.1. Different layers of Web Applications

The Client layer indicates the user's web browser which communicates with the server. The Presentation layer offers the graphical user interface (GUI) which shows the results of the user request. The CGI layer is responsible for management tasks at the server. The languages which are used in CGI layer are the server-based languages like ASP, PHP, etc.

At the Database layer, the data related to the web application are saved and in case a demand from CGI is received, the proper result will be sent to the client. [2]

As we discussed above, methods of detecting and preventing SQLIAs are either static which happens at the Database layer, or dynamic which happens at the CGI layer or combined method.

A. Architecture

Figure 2 shows proposed framework given query will pass over two phases. The two phases are pattern based and machine learning based. As well database firewall will be there for checking incoming SQL queries.

If the given query passes all this phases, then only query is valid otherwise there is attack.

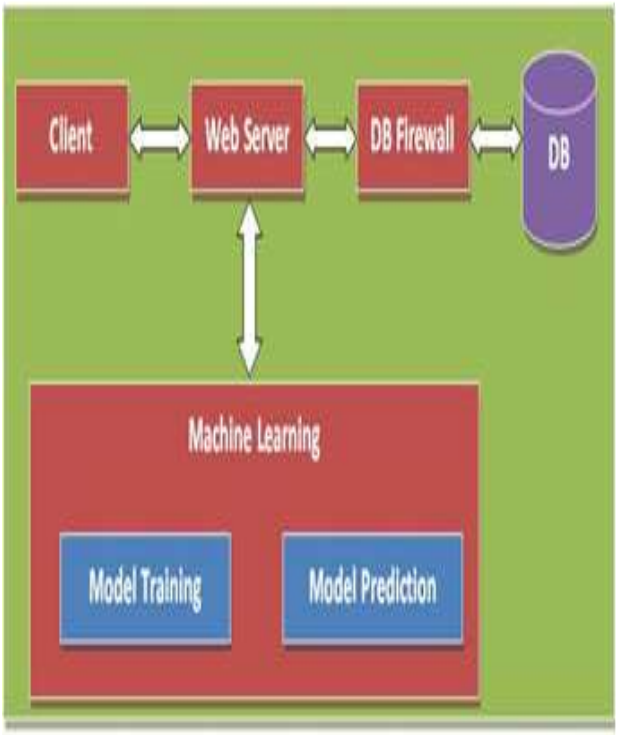

Fig.2. System Architecture

The Proposed framework is basically divided into following entities and modules:-

1) Client- The Client layer shows the user's web browser which communicates with the server.

2) Web Server- The languages which are used in this layer are the server-based languages like ASP, PHP, and so on. etc. At the Database layer, the data linked to the web application are saved and in case a request from CGI is received, the proper result will be sent to the client.

3) DB Firewall- For increasing security of database, the firewall will be installed and used to avoid the SQLIA. It will helpful for detecting illegal queries.

4) Machine Learning- Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs.

B. Algorithm

N: Total number of fixed SQL Queries in web application

FQi: i'th Fixed Query in web application

DQi: Different types of queries from FQi FQ_array={FQ1, FQ2...FQn}, FDQ={FDQ1....FDQn}

//Dynamic Analysis of Queries

1. For each i=1 to N in FQ_array
2. Get FQi
3. FDQi=f(testArray(FQi))
4. if FDQi_total>0 then
5. print Attack Detected
6.else
7.print Normal Query
8.End if
9.End {For}

IV. RESULT AND DISCUSSIONS

The mechanism is executed for real web application to detect and prevent the SQL injection attacks. It detects all SQL injection attack types. Figure 3 shows that the static attack detection page and figure 4 shows that the dynamic attack detection page.

Fig. 3. Static Attack Detection Page

Fig. 4. Dynamic Attack Detection page

The evaluation for results is done on the basis of Dataset, Response time, Computing time and Detection Rate.

- **Dataset**:- Figure 5 consists of normal as well as nasty (Malicious) queries.

- **Response time**:- Figure 6 shows the response time of the system in handling the raised errors. The time to detect the SQL injection is considered for particular attack type.

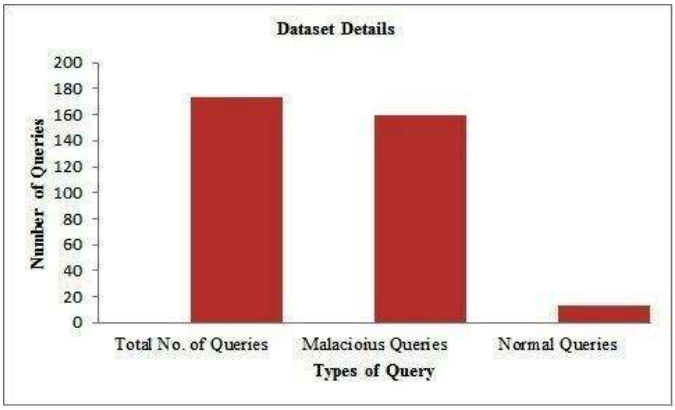

Fig. 5. Dataset for normal query and injected query For Static Attacks

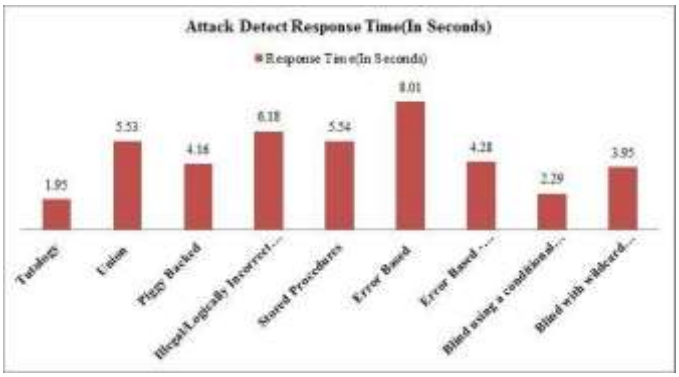

Fig.6. Response time of various Static attack types

• Computing time: Figure7 shows the time required to compute the result of the normal query with respect to static injected query.

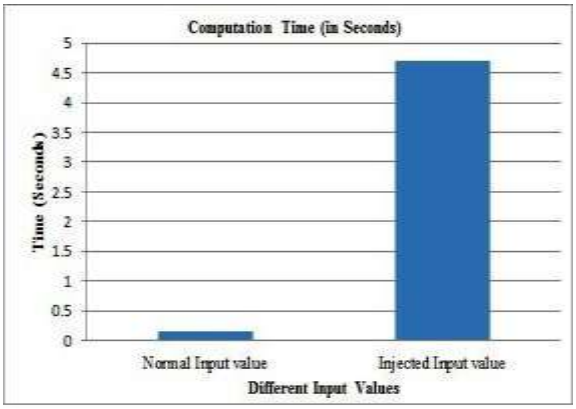

Fig. 7. Computation time for normal query & Static injected query

•**Detection Rate**: - Table-I gives statistics of detection rate. A total of 173 queries are issued for each SQLIA page out of which 160 are malicious and 13 are normal. Figure 8 shows the graph of static SQLIA detection rate and normal input (normal query) detection rate.

TABLE I. Analysis of Static SQLIA and Normal query Detection rate

Web page	Analysis of detection rate		
	Attacks Detected / Malicious Input	***Attacks Detected /Normal input***	***Detection Rate (%)***
SQLIA page	160/160	13/13	100
Detect Page	160/160	13/13	100
Prevent page	160/160	13/13	100

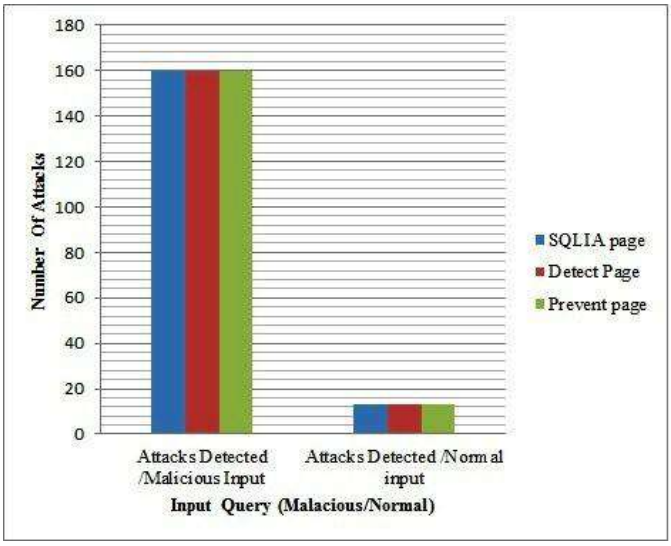

Fig.8. Analysis of Static SQLIA Detection Rate and Normal query Detection Rate.

• **Dataset**:- Figure 9 consists of dynamic query-(Malicious query)

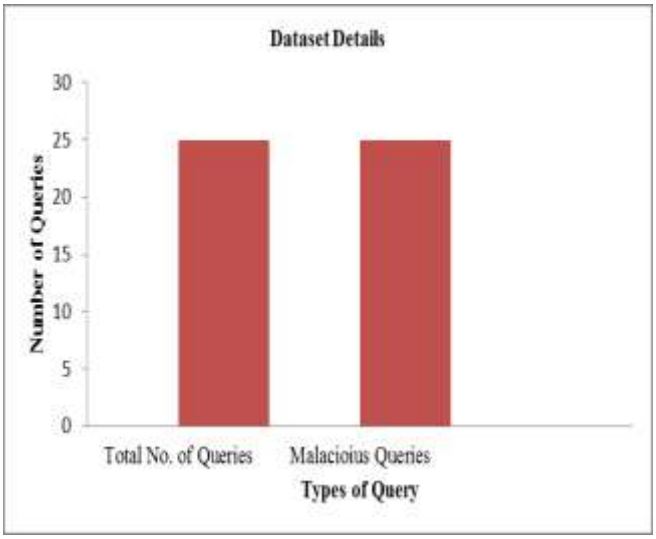

Fig. 9. Dataset for Dynamic Attacks.

• **Computing time**: Figure 10 shows the time required to compute the result of the dynamic injected query.

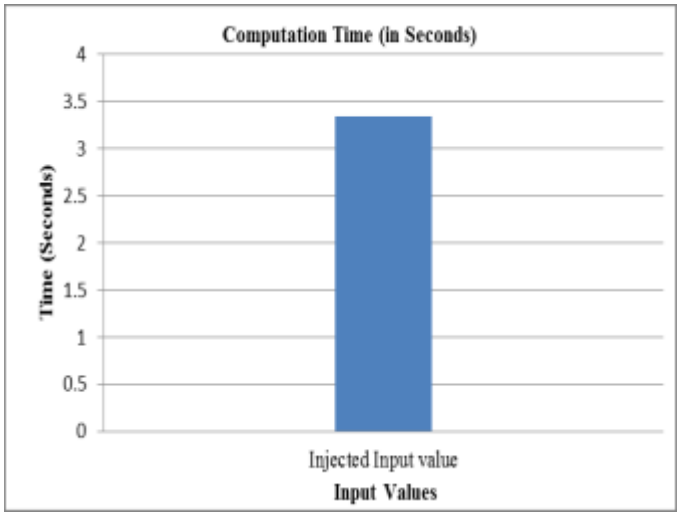

Fig.10. Computation time for Dynamic injected query

- **Detection Rate**: - Table-II gives statistics of detection rate. A total of 25 queries are issued for each SQLIA page.

Figure 11 shows the graph of dynamic SQLIA detection rate.

TABLE II. Analysis of Dynamic SQLIA detection rate

Web page	Analysis of detection rate	
	Attacks Detected / Malicious Input	***Detection Rate (%)***
SQLIA page	25/25	100
Detect Page	25/25	100
Prevent page	25/25	100

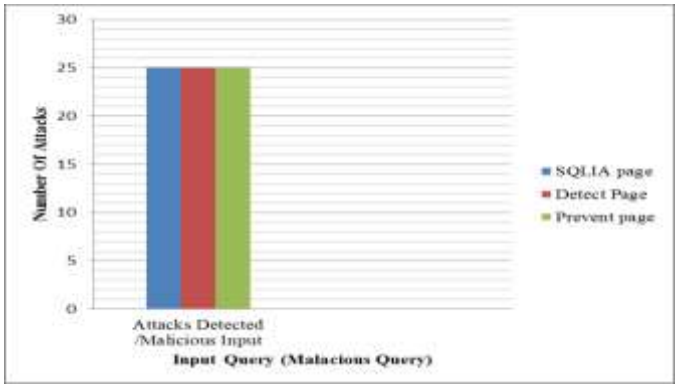

Fig. 11. Analysis of Dynamic SQLIA Detection Rate

V. CONCLUSION

In this paper we proposed three stage frameworks for detection and prevention of SQLIA attacks. As the popularity of web application increasing, the security of web application is major problem. SQL Injection attacks are the exorbitant and dangerous attacks on web applications: it is a code injection technique that allows attackers to take unrestricted access to the databases and potentially slight information like usernames, passwords, email ids, credit card details present in them. It covers the wide-ranging introduction to SQL injection attacks and their types. As well the literature review present the various ideas proposed by various researchers in the field of SQL injection. Then the various purposes of SQL injection attack are explained. The proposed approach usage a patterns and machine learning models in which value is entered for every field is checked for SQLIA by parsing it and static and dynamic attacks are covered.

ACKNOWLEDGEMENT

I am colossally gratifying to Dr. K. T. V. Reddy, Principal, Sir Visvesvaraya Institute of Technology (SVIT), Nashik for inspiring me towards this and for implausible backing and leadership too. As well we prolong obligations towards Prof. Shedge K. N. (Asst.Professor), HOD M. Tech (CSE) of Computer Engineering Department, Prof. Thosar D. S., Assistant Professor and M.E Coordinator and Staff Members for their appreciated Assistance and Provision.

REFERENCES

[1] Rajashree A. Katole, Dr. Swati S. Sherekar, Dr. Vilas M.Thakare, SGBAU, Amravati, Maharashtra, India. Detection of SQL Injection Attacks by Removing the Parameter Values of SQL Query 978-1-5386-0807-4/18/c 2018 IEEE

[2] Mr. B.R.Thakare, Prof. D.S.Thosar- A Novel 3 Stage Hybrid Framework for SQLIA Detection, Prevention Using Pattern Based and Supervised Machine Learning Model-www.ijariie.com

[3] Chen Ping-A second-order SQL injection detection method 978-1-5090-6414-4/17/2017 IEEE

[4] SaiLekshmi A S,Devipriya V S- An Emulation of SQL Injection Disclosure and Deterrence,978-1-5090-6590-5/17/2017 IEEE

[5] Pankajdeep Kaur,Kanwal Preet Kaur, SQL Injection: Study and Augmentation-978-1-4799-8436-7/15/2015 IEEE

[6] https://www.owasp.org

[7] Dafydd Stuttard and Marcus Pinto. The Web Application Hackers Handbook: Discovering and Exploiting Security a.Flaws