Volume 6, Issue 2, February 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

SQL Injection& XSS Attacks: An In-Depth Explanation Method of Exploiting Attack and Prevention Technique in Asp.Net Web Application

Shweta Sharma*
M.Tech student, Dept. of IT
AIET, Lko, India

Madhulika Sharma
HOD of Dept. of CSE & IT
AIET, Lko, India

Anil Kumar Tripathi
Project Incharge in Rajdeep Electrical Pvt. Ltd., India

Abstract— During the last few decades use of web and database applications extend to a large extent. With the increase in use of the web based applications there was also increase in use of online database applications. When there is growth in one technology the associated problems also arises. Out of many problems and threats to the database applications one potential problem is of SQL injection. It is a code injection technique, used to hack data-driven applications, in which malicious SQL codes are inserted into an entry field for execution. It is one of the many web attack mechanisms used by attackers to steal data from organizations. SQLIAs is a type of attack that takes advantage of improper coding of your web applications that allows attacker to inject SQL statements into say a login form to allow them to gain access to the data held within your database. This type of attacks arises because the fields available for user input allow SQL statements to pass through and query the database directly. This type of attack can compromise confidentiality and integrity of information in databases. Actually, an attacker pokes to the web application database and consequently, access to data. For stopping SQLIAs different approaches have been proposed by researchers but they are not enough because that implemented approaches cannot stop all type of attacks. Another attack which is also very harmful for web based application i.e. XSS (Cross Site Scripting) attack. Cross Site Scripting attacks occur when a hacker uses a web application to send malicious code in the form of a browser side script, to a different end user. A hacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser will execute the script like a trusted script because there is no way to know that the script should not be trusted. The reason is that an end user thinks the script came from a trusted source; the malicious script can access session tokens, cookies, or any sensitive information retained by the browser. Cross Site Scripts can even rewrite the content of the HTML page. In this work I have studied all about SQLIAs and XSS attacks. All type of different techniques which can defect or prevent them is presented. Many solutions were also proposed with passage of time. The proposed solution which I have given in this work can prevent from XSS and SQL Injection attacks. This is very helpful for developers those are working in an organization. I have proposed a layered architecture technique which is not exactly 3 layer architecture but this is much enough to prevent from XSS and SQLIAs.

Keywords— SQLIA Prevention, XSS, SQLIA Detection, SQL Attacks, Vulnerabilities .

I. INTRODUCTION

Web applications enable online business including banking, university admissions, email, social networking, shopping, and various government activities. The users only need a web browser and internet connection to gain any information from web applications. When users achieve dynamic information from web applications or make transactions online, web application contain such as credit card numbers and product inventory data, typically has most valuable for both the users and to the service providers.

Web based applications can contain sensitive and confidential information which is stored in database. These applications accept the data from the users input. This data is fetched from the database through the queries. SQLIA is still one of the most popular attack used in system hacking. With the use of SQL Injection Attack attacker can gain information or can have unauthorized access to the system by injecting malicious sql queries to user input. By this way attacker gains control over web application maximum harm or damage is caused.

To insert, update, delete, and retrieve the data from the database SQL language is used. When User enters data in the input fields it becomes part of the SQL query which is written at the backend. For Example, If user want to Login into their Admin Panel, user enter loginid and Password.

For gaining access of Admin Panel user login his/her loginid and password. This loginid and password form the part of the internal SQL query. After passing loginid and password, SQL query is executed on the database to check whether the user is authenticated or not means loginid and password is existing in database or not. The attackers who want to gain information from the admin panel, provides malicious query instead of correct login credentials in the input fields of the web application. This injected code can change the structure of the original sql query, allows the attacker to gain access to the information it was not authorized for. This type of method which allows the attacker to change or alter the original SQL query by adding the injected malicious SQL query in the input fields is recognized as SQL injection Attack [1].

In this type of attack, attacker tries to alter SQL query by inserting new guessing SQL keywords. Attacker can modify the original SQL query by inserting new query through user input field. When injected query concatenated with sql command, this will formed syntactically correct. By this way any information within the database will be altered, dropped or extracted.

When we use web applications there is an increase in number of various types of attacks that target them. To secure a web application against hacking is a big challenge today. Cross-Site Scripting (XSS) technique to hack a web application is very harmful. It is one of the most common types of hacking technique to hack the web application.

Cross-Site Scripting vulnerabilities are being exploited by the hackers to steal web browser's resources such as credentials, cookies etc. by injecting the malicious JavaScript code on the victim's web applications.

In addition, Injection techniques have become more ambitious, more frequent, and increasingly sophisticated, so still there is a need to find a feasible and effective solution for this problem in the web application security community.

Prevention or detection of SQL injection attack is an active topic of research in industry and academia. Various automatic tools and security systems have been implemented to achieve security issues, but none of them are accurate or complete enough to give guarantee and absolute level of security on web applications. So we felt that there should be such type of method which will be easily deployable, does not need source code modification as well as provide a good performance. To achieve this, our research work is driven to the way of implementing a new modified SQL injection prevention technique. This technique is also focused on the XSS attack prevention which is also very harmful for web application.

II. CONSEQUENCES OF SQL INJECTION ATTACKS & XSS ATTACKS

The typical intentions of the SQLIA are:

1. In order to perform database finger printing. Fingerprint information are the information about type and version of database. Every successful SQLIA attack must know about certain information in the form of version of SQL language used. Different data base management system use different version of SQL language. Few example Oracle use PL/SQL and MS SQL Server use T-SQL.
2. Run a command to get all the information about user and password
3. In order to determine the database schema. A database schema is the structure of the database with name of fields, tables and logical relationships. In order to load a successful subsequent attack these information are very crucial.
4. In order to extract and modify data. This is the core aim of the SQLIA and most abundant and common types of attacks.

XSS attacks can be used to conduct a number of browser-based attacks. Some of them as follows:-

1. Other malicious activities
2. Pseudo defacement of the application
3. Capturing sensitive information viewed by application users
4. Directed delivery of browser-based exploits
5. Port scanning of internal hosts ("internal" in relation to the users of the web application)
6. Hijacking another user's browser

Most of the attacks (about 90%) are related to financial loss but in the long run most damage is on the repute of the organizations. According to a survey about 60% of customer stop using services after a security breach into the organization.

III. MOST COMMON VULNARABILITIES FOR SQLIA & XSS ATTACKS

The most common type of vulnerabilities for SQLIA attacks are "insufficient input validations", existence of "privileged accounts" and "extra functionalities". Input validation is the technique to filter user input to prevent malicious attempt from users. Improper or weak authentication mechanism of user and password matching on the log in page is the most common initiation of SQLIA. When user input is allowed without proper verification the attacker is free to execute code for his/her malicious intents. Attacker usually exploits the weak or improper input authentication to induct malicious code to get desired results for malicious intents.

By XSS attack hacker can send a malicious script to as unsuspending user. Both SQLIA & XSS attacks are very harmful for importance information.

IV. VARIOUS TYPE OF SQL INJECTION & XSS ATTACKS TECHNIQUES

SQL Injection Techniques:-

(i) Tautology Attacks:- Tautology attacks aim to insert malicious code with the conditional statements to order to make their evaluation always to true. The consequences of this attack vary to certain extent and totally depend upon the intention of the attacker. The most common intent is to bypass the authentication process and to extract data for the intended data. The inject-able field is exploited that is used in WHERE clause of the query. The transformation of all the conditions into tautology, results into return of all the rows of the intended table. In order to be successful for the

tautology based attack. Attacker must have some know how about inject able or vulnerable parameters and up to some extent the coding constructs that evaluate the query and return results. Tautology based attacks considered successful when all the results are displayed by the code or at least one record that allow some other actions. The example explained in the SQLIA process the query generated response to malicious user input is:

Select * from tbl users Where username='SHWETA' OR '1'='1'- -'AND password='whatever

Malicious user submits the **[1=1 --]** with the user name**. –** causes the impact of remaining query ignored. This code change **WHERE** clause into a tautology results into true for each row and return the entire row of table.

(ii) Union Attack:- In this technique safe query is joined with injected query with the keyword UNION in order to get data from other tables. In this sort of technique a vulnerable parameters is used to return the result from other tables.

Using syntax UNION SELECT <injected query> with a legitimate query helps to retrieve results from other tables in contrast to table specified by the developer. With this sort of query the result returned from the first query and result from the second part of the query is returned to the attacker. As the illegitimate part of the query is in complete control of the attacker he or she can access the data from any table. For example if the user insert the fallowing query into the login field ;

[' UNION SELECT pwd FROM user-info WHERE id='abc' AND pwd='']

It would join with the original query and the resultant query generated in response to this query is shown in the small table.

SELECT * FROM users WHERE id='' UNION SELECT pwd FROM user_info WHERE id='abc' - - AND pwd=''

In this query there would no result returned from the first query but the second query return the password from the user_info table for the abc user.

(iii) Logically Incorrect query Attack:- This type of attack used to get information about the structure of database. This sort of attack help to get basic type of information that can be used for subsequent attacks. Information returned by the server in response to an error is used for making other attacks. The error page returned by the server contains information that can be exploited by inject able parameters. In this technique attacker tries to generate syntax, logical or type conversion information. Syntax errors are used to gain information about injects able parameters. Type conversions errors help to gain information about data type and logical errors help to gain information about the tables and fields.

(iv) Alternate Encoding:- Illegitimate queries injected by changing the coding schema in the form of Unicode, ASCII or hexadecimal format. Using encoding for the queries the injected code can bypass the filter or scanner used to test the legitimacy of the query. For example attacker can use char (44) which is the code for quote that may be the bad character for the scanner.

(v) Inference:- In this types of technique the attacker try to change the behaviour of the database. Build injection and timing attack are based on inference principle. In blind injection the attacker uses series of true and false questions from the database. When developers hide the errors details the SQLIA become difficult but not impossible. Blind injection techniques are used in this sort of scenarios. In timing attack response delays from the database observed in order to inject queries.

XSS Attacks Techniques:-

Basically there are three types of XSS attacks. These are as follows:-

a. Stored XSS (AKA Persistent or Type I)
b. Reflected XSS (AKA Non-Persistent or Type II)
c. DOM Based XSS (AKA Type-0)

a. Stored XSS (AKA Persistent or Type I):- In this type of attack injected script is permanently stored on the target servers, such as in a database, in a message forum, comment field, visitor log etc .Victim then retrieves malicious script from the server when it requests the stored information. Sometimes this stored XSS is referred to as Persistent or Type-1 XSS.

b. Reflected XSS (AKA Non-Persistent or Type II):- In Reflected XSS attack , which is also known as second type of Cross – Site Scripting attack, an attacker sends the injected script to the site which is vulnerable so that it will be immediately returned back to the user.

c. DOM Based XSS (AKA Type-0):- DOM Based XSS attack which is also known as third Cross- Site Scripting attack. This type of attack occurs entirely in the web browser. An attacker performs the injection of malicious script into the web page; in the other types, the web server performs the injection.

V. PROBLEM FORMULATION

This chapter describes the basics of SQLIA, the basic techniques and defence techniques along with the proposed and existing solutions to the issue.

Status of SQLIA defence techniques:-

As we mentioned above the SQL injection has gained the attentions of researcher and different defence techniques emerged over the last few years and it his rate emerged to a large extent. The next figure depicts the story of increasing defence techniques emerged over the last few years and a rising trend is very obvious in the figure.

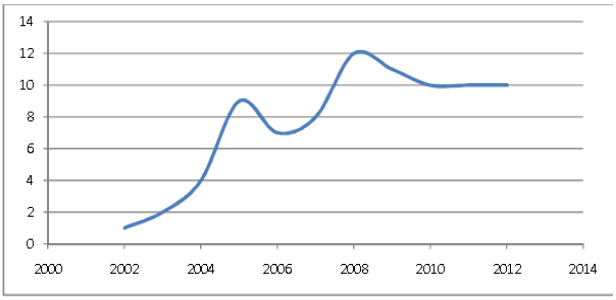

Fig 1: No of defence techniques emerged over the time

These techniques use different detection principles. Out of the emerged techniques over that last few years thirteen percent use tainted data flow, thirteen percent anomaly based, twenty five use signature based and forty nine percent use grammar based technique.

SQLIA detection and prevention tools and techniques analysis:-

(i) Abstracting Application-Level Web Security

In this technique authors proposed structural techniques for the web developers to implement security policies for web applications. Authors believe that these techniques help to protect several types of web attacks. Their proposed techniques based on three components including the Security policy description language (SPDL), policy compilers, and security gateways. Application validations rules are defined in SPDL where transformation rule decide what have to do when a malicious attack performed on the attack. Policy compilers transform the rule into code. Whereas security gateways between the client and server acts as firewall to implement the specific security policy. [2]

(ii) Web Application Security Assessment by Fault Injection and Behaviour Monitoring

This technique works well for SQLIA as well as for XSS attacks. It detects web vulnerabilities with the help of web crawlers. It also uses different types of attacks technique as use machine learning approach to defend against these attacks. It uses machine learning approach therefore considered as best than other techniques. It is implemented into WAVES (Wave Application Vulnerability and Error Scanner) tool by the author. [3]

(iii) An Automated Universal Server Level Solution for SQL Injection Security Flaw

This technique is used to safeguard the web server against the SQL injection techniques. Author claimed that this technique can be used to protect any type of web server from SQLIA. AUSELSQI intercept the HTTP messages for any malicious or suspicious contents and rejects the malicious contents. It is implemented this technique on an ISAPI filter on IIS web server where it can also act as firewall. [4]

(iv) SQLRand: Preventing SQL Injection Attacks

In this technique the author applies the randomization of instructions set of SQL language. SQL statements are tried to change with the help of a random number that make it very difficult for the attacker to predict the syntax of SQL. This technique known as SQLRand implemented using a proxy that sit between the server and client that translate back the randomized SQL statements to actual code. When a malicious user make an attempt the proxy would reject the queries as it would not be recognized by him. [5]

(v) WASP: Protecting Web Applications Using Positive Tainting and Syntax- Aware Evaluation

This technique safeguards the web application against the SQLIA using the positive tainting. It tracks the data for being trusted or not. At the next phase it use dynamic tainting for trusted only trusted data and only pass the trusted data for further processing. This technique also performs static analysis of the syntax of the query for possible vulnerabilities. It is implemented in in WASP (Web Application SQL-injection Preventer). [6]

VI. ROPOSED SOLUTION

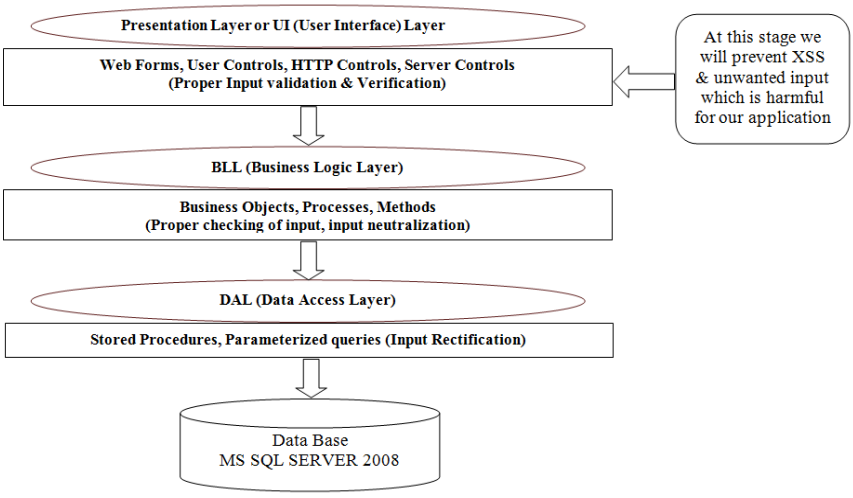

Fig 2 Proposed architecture to prevent SQLIA & XSS attack

We proposed architecture in ASP.Net which works for both Windows as well as Web applications. In this architecture we use concepts of OOPs (Object Oriented programming Systems). It has certain advantages over the previous developed techniques and model. We have just focused on code reusability and security purpose for avoiding SQLIAs.
We have proposed an architecture which is like three layer architecture but we do not need to code again and again on three layers. We need to code on 2 layers only third layer which we have developed once and can use again and again for reducing reusability of code.

A. Stages of Proposed Mechanism

The proposed technique consists of four-stage security mechanism, which are as follows:-

1) Proper Input Validations and Verification stage: This stage works on client (user) side "web page at browser"; it detects the use of special characters with the help of regular expressions. This technique is called validations. With this mechanism, we are able to protect the web page controls like (text boxes) from any kind of special character, which is meaningful to the SQL. The only drawback of this stage is that, it cannot handle the attacks from the query strings, cookies etc. At this stage we also secure our data by XSS attacks.

2) Data filtration Stage: This stage basically works on the code written behind the design page. This stage detects the malicious SQL code, inject by various other techniques like query strings, URL's. The data filtration stage analyzes the data before passing on to the database.

3) *Data Cleaning Stage (Sanitizer)* **:** Parameterized query is parameterized database access API provided by development platform such as Prepare Statement in Java or SQL Parameter .NET. Instead of composing SQL by concatenating string, each parameter in a SQL query is declared using placeholder and input is provided separately. The sanitizer classify the inputs as follows-

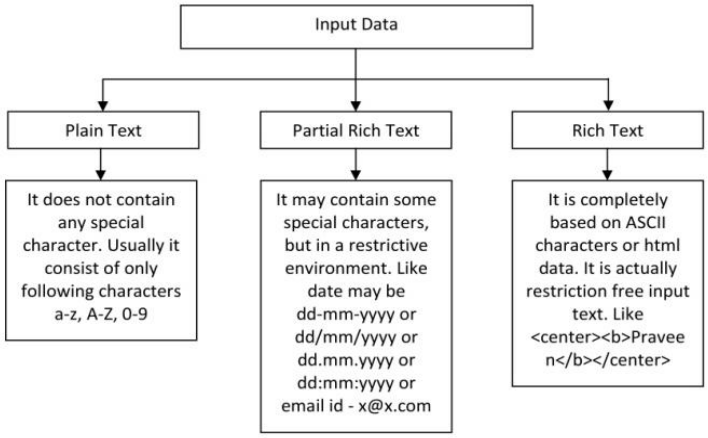

Fig 3 Input Classification based on their nature and types

4) Parameterized Check Stage: Parameterized queries keep the query and the data separate through the use of placeholders known as "bound" parameters. This helps in preventing SQLIA by not allowing the structure of the query to be altered; rather it merely "fills in" the input parameters into their positions and keeps the rest of the query structure intact. Since a majority of the SQLIA techniques relies on altering the query structure for injection attacks, this serves as a very effective combat technique.

B. Communication Flow of Our Reduced Architecture

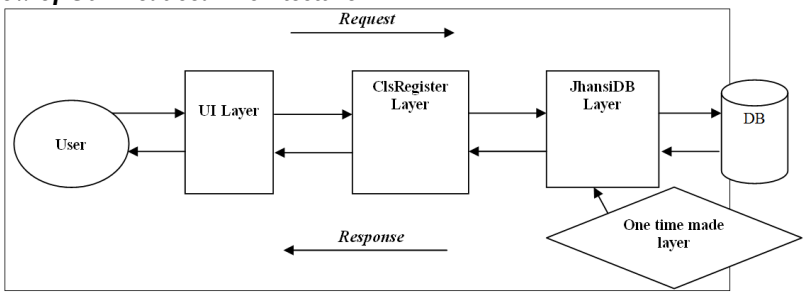

Fig 4 Requests and Responses in the proposed Architecture

Each layer requires different sets of skills:

I. UI Layer is also called Presentation Layer. The Presentation layer requires skills such as HTML, CSS and possibly JavaScript, plus UI design.
II. ClsRegister Layer is also called Business Logic Layer. The Business layer requires skills in a programming language so that business rules can be processed by a computer.
III. JhansiDB Layer is also called Data Access Layer. The Data Access layer requires SQL skills in the form of Data Definition Language (DDL) and Data Manipulation Language (DML), plus database design.

Although it is possible for a single person to have all of the above skills, such people are quite rare. In large organizations with large software applications this splitting of an application into separate layers makes it possible for each layer to be developed and maintained by different teams with the relevant specialist skills. In this architecture we made JhansiDB Layer one time and can use in many projects. So we are working at only 2 layers UI Layer and ClsRegister Layer.

VII. IMPLEMENTATION

The complete implementation of the proposed work carried out using the Microsoft .Net framework 3.5 with Visual Studio 2008, SQL Server 2008, IIS Server. The demonstrated website works in two modes at every instance. Whether its login or search, inserting or updating, etc. Following are the two modes:

Secure Mode In this mode the website works under the guidance of our proposed architecture which restricts malicious queries and XSS attacks also.

In this mode, the website also works in different tiers. Each tier contains a level of attack detection. For example, the client levels identified the input type and according to that apply the regular expression as the input validation.

At server side, the website analyses the input data and sends to the database stored procedure, where execution of SQL commands performs and the desired data retrieved.

At data access layer, the website received the data from the internal web pages, and then it uses the sanitization process to make decision on input data before sending to the database program.

The most important feature of our proposed plan for detecting & preventing the SQL injections is that it never overhead the data in the database. It can be applied to any existing web model without performing any alteration in database schema.

A. UI layer

(a) Validations stage This stage works on client side "web page at browser"; it detects the use of special characters with the help of regular expressions. This technique is called validations. With this mechanism, we are able to protect the web page controls like (text boxes) from any kind of special character, which is meaningful to the SQL. The only drawback of this stage is that, it cannot handle the attacks from the query strings, cookies etc. The other problem at this layer is Xss attack . To resolve this type of error we have to code something which is explained below:-

Example of this type of error:-

By default, ASP.Net will not allow any HTML tag in any input controls. In this case, you will receive the below error if you input html tags and submit the page.

A potentially dangerous Request.Form value was detected from the client

For this problem , we only need to set Validation Request="false" in those web pages where we want to allow HTML input.

The normal code of .aspx page to show value of textbox is as below:-

```
<asp: Textbox ID="txtMsg" runat="server" Text Mode="MultiLine"></asp:TextBox>
<asp:Button ID="btnSubmit" runat="server" Onclick="btnSubmit_Click" Text="Submit"/>
```

Code Behind Code :-

```
Protected void btnSubmit_Click(object sender, EventArgs e)
{
Response.Write(txtMsg.Text);
}
```

Execute the above page and type the below string as input in the textbox and click submit.

```
<script>alert('You are hacked!')</script>
```

To solve this problem we need to add the ValidateRequest="false" in @Page line of web page like this in Web Forms :-

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default"
ValidateRequest="false" %>
```

Otherwise we can set it globally to work for all the pages without having this error we need to add ValidateRequest="false" in web.config file under system.web section like this:-

Web.Config:-

```
<system.web>
--------------------------
<httpRunTime requestValidationMode="2.0"/>
<pages ValidateRequest="false">
</pages>
----------------------
</system.web>
```

(b) Data filtration Stage: This stage basically works on the code written behind the design page. This stage detects the malicious SQL code, inject by various other techniques like query strings, URL's. The data filtration stage analyzes the data before passing on to the database.

(1) Using Parameterize query: If we want to use inline SQL,then to stop SQL injection we can pass the parameter in sql query and can add SQL Parameter in the query .this way we can prevent sql injection .there is nothing wrong in using sql statement inline.

Parameterized query tell to the SQL server that data passed into any parameter will remain in the data channel thats how

sql server prevent the SQL injection. But still its not the best way as we are writing business logic inline .every time i have to write again this query.

(2) Using stored procedure: The best way to prevent from SQL injection is use stored procedure. As business logic are hidden , Its provide better performance , reusability . Now you will have to just protect the table and stored procedure by using permissions.

Simple Login Query:-

SELECT username, password FROM tbl_userinfo WHERE name = 'praveen' AND pass='praveen'

Simple Login Query- Stored Procedure

```
CREATE PROCEDURE [dbo].[splogin]
@username nvarchar(20),
@password nvarchar(20)
AS
BEGIN
SET NOCOUNT ON;
SELECT username, password FROM tbl_user WHERE username = @username AND password=@password;
END
```

The above stated query can be easily hacked by the attackers. As there is no mechanism for trouble shooting.

Proposed Login Query: Stored procedure

```
CREATE PROCEDURE [dbo].[spsecurelogin]
@username nvarchar(20),
@password nvarchar(20)
AS
BEGIN
SET NOCOUNT ON;
SELECT
username, password
FROM
tbl_user
WHERE
hashbytes('SHA',username) = hashbytes('SHA',@username)
AND
hashbytes('SHA',password) = hashbytes('SHA',@password);
END
```

The above stated query is free from any malicious code, in-fact if user somehow transmits malicious data to the SQL procedure; our proposed SQL query will compute the Hash values with "SHA" (128 Bits) then it will be compared with database. Therefore, due to applying operation of hash values the input values are sanitizes completely and guarantees you solution in legal way.

Similarly, with the handling of query strings:

The QueryString collection is used to retrieve the variable values in the HTTP query string. The HTTP query string is specified by the values following the question mark (?), like this:

Response.Redirect

("admin.aspx?adm_name=prav")

The line above generates a variable named adm_name with the value "prav". The Query strings are also generated by form submission, or by a user typing a query into the address bar of the browser.

While at the redirected page, this query string handled by the code for further use like:

```
Dim admin_name as String
admin_name = Request.QueryString("adm_name")
```

hence, it is the duty of the application programmer to check the validity of this query string value and make sure that, there is n malicious characters or code in the value.

IsValid(admin_name) 'Boolean type function

Or

PT_Sanitizer(admin_name) 'Boolean Type Func

The above function is uses by our proposed system, which sanitize the admin name and help us to decide whether to send the data to database or not.

The IsValid Code looks like:

```
Public Function IsValid(ByVal str As String) As Boolean
        Dim rx As New Regex
("[0-9a-zA-Z$%^&amp;amp;*()_+!]{3,20}")
 Dim bool As Boolean = rx.IsMatch(str)
        Return bool
    End Function
```

While, the plain text sanitizer simply checks the presence of any trouble making characters.

VIII. RESULT

WEB FORM AFTER USING PROPOSED MECHANISM:-

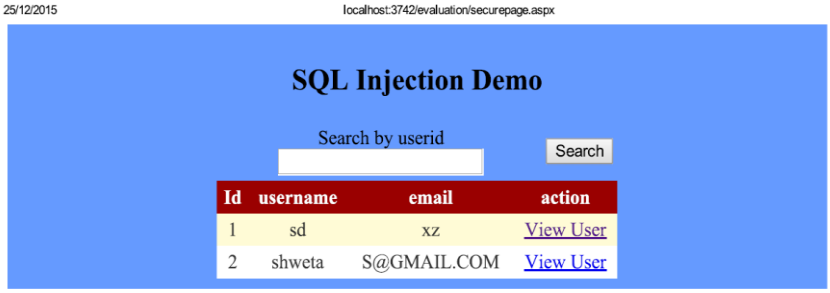

Fig 5 Web form after using proposed mechanism

AFTER APPLYING MALICIOUS SQL QUERY

WE HAVE WRITTEN THIS MALICIOUS SQL QUERY TO FECTCH IMPORTANT RECORDS

1 UNION SELECT USERID,NAME,PASSWORD FROM USER_INFO

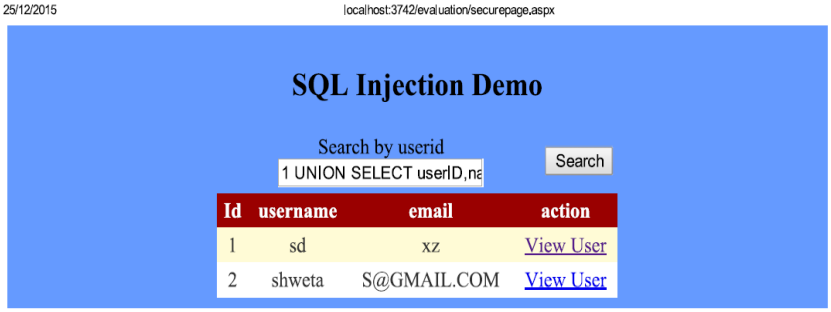

Fig 6 Output Screen After Applying SQL Injection Query

NOW WE ARE NOT ABLE TO HACK DATABASE BY USING SQLI ATTACK.

NOW CHECK 1=1 TAUTOLOGY.

FIG 7 OUTPUT SCREEN AFTER CHECKING TAUTOLOGY ATTACK

We can't access without Authorize Details of Username & Password.

IX. CONCLUSION AND FUTURE WORK

In this paper, we have concentrated on the specific area of SQL injection. Though this is a narrow subject, we believe that this area is in need of further investigation, mainly because of two reasons: first, we cannot be certain that we have compiled a definite list of all components that could be taken into consideration. Secondly, SQL injection attacks are most likely to evolve and new vulnerabilities will be found, together with new countermeasures to deal with them. Since many hacking sites are available on the web, and since attack methods are well described and distributed between hackers, we believe that information about new attack methods should continuously be surveyed and new countermeasures should be developed.

According to OWASP's Ten Most Critical Web Application Security vulnerabilities [8], many SQL injection-related issues are among the most harmful threats to web applications. Since we have in this thesis only covered SQL injection aspects, we would like to suggest that further studies should be

made on other threats to related security issues, especially such that relate to application security. The reason is that several authors have mentioned that organizations spend most security resources on operating system and

Network level security [7, 8, 9], and not enough on application layer security. If further studies will be made on application layer security issues, and particularly on web application, it would be possible to compile results from all of these into general security guidelines, which could be used in developing more secure web applications.

One of our goals in this paper was to increase the level of security awareness among organizations regarding web applications, especially towards SQL injection threats. We hope that further surveys in this area and in related web application subjects will help achieving that goal, so that hopefully security standards will be implemented and countermeasures built into applications during development. Ultimately, organizations will use a proactive approach towards application layer security, which will then be an indispensable part of web applications.

REFERENCES

[1] Neha Singh,Ravindra Kumar Purwar,SQL Injection –A HazardTo web applications, *International Journal of Advanced Research in computer Science and Software Engineering*,vol.2,Issue 6,June 2012,pp. 42-46.

[2] Abstracting application-level web security. David Scott, Richard Sharp. New York : s.n., 2002. Proceedings Of the 11th international conference on World Wide Web . pp. 396-407 .

[3] Web application security assessment by fault injection and behavior monitoring. Yao-Wen Huang, Shih-Kun Huang , Tsung-Po Lin , Chung-Hung Tsai. New York : s.n., 2003. Proceedings of the 12th international conference on World Wide Web . pp. 148-159 .

[4] An automated universal server level solution for SQL injection security flaw. Alfantookh, A.A. 2004. Electrical, Electronic and Computer Engineering, 2004. ICEEC '04. 2004 International Conference . pp. 131-135.

[5] SQLrand: Preventing SQL Injection Attacks. S W Boyd, A D Keromytis. 2004. Proc. Second Int'l Conf. Applied Cryptography and Network Security.

[6] Wasp: Protecting web applications using positive tainting and syntax-aware evaluation. W. G.J. Halfond, A. Orso, P. Manolios. 2008. IEEE Trans. Softw. Eng. p. 34:65.

[7] Using parse tree validation to prevent SQL injection attacks. Gregory Buehrer, Bruce W. Weide , Paolo A. G. Sivilotti. New York : s.n., 2005. SEM '05 Proceedings of the 5th international workshop on Software engineering and middleware . pp. 106-113 .

[8] M.S., Sam. SQL Injection Protection by Variable Normalization of SQL Statement. http://www.securitydocs.com/library/3388. [Online] june 16, 2005.http://www.securitydocs.com/link.php?action=detail&id=3388&headerfooter=no.

[9] Dynamic taint propagation for Java. Haldar, V., Chandra, D. and Franz, M. Tucson, AZ : s.n., 2005. 21st Annual Computer Security Applications Conference. pp. 9 pp. - 311 .

