Urvashi Sanadhya / International Journal of Engineering Research and Applications (IJERA)
ISSN: 2248-9622 www.ijera.com Vol. 2, Issue4, July-august 2012, pp.2392-2398

Avoid SQL Injection Attacks

Urvashi Sanadhya
Department of Computer Science,
Mewar University, Chittorgarh
Tel: 07589397539

Abstract

An SQL injection attack targets web applications that are database-driven. The methods using for SQL injections are easy to learn and can cause major or significant damage to the system. To address this problem, we present the different types of SQL injection attacks known to date and will look at a selection of the methods available to a SQL injection attacker and how they are best defend against them. For each type of attack, we provide descriptions and examples of how attacks of that type could be performed. We also present and analyze existing detection and prevention techniques against SQL injection attacks.

Keywords-SQL injection, SQL injection attacks, Authentication attacks

1. Introduction

SQL Injection Attacks are one of the topmost threats for web application security,and SQL injections are one of the most serious vulnerability types. The SQL Injection attacks are easy to learn and exploitable, so this method of attack is easily used by attackers and hackers. Also many major and traditional security systems having different security layers like firewall, encryption, intrusion detection systems, Antivirus and anti malware are not able to detect this type of attack. Also database mechanism for authentication and authorization can be bypassed by tricky methods and using set of rules of that type of database.

SQL Injection is something related to web-hacking, but using some SQL knowledge and legal SQL commands to make it vulnerable. Its take the advantage of the fact over which a poorly secured web-application is developed. Also its takes the advantage of how data engines process the query/insecure code in database. Many SQL takes the advantage of errors/error message generated by systemon some query responses. SQL Injection attacks
employed by malicious users for different reasons, e. g. financial fraud, theft confidential data, deface website, sabotage, espionage, cyber terrorism, or simply for fun. Furthermore, SQL Injection attack techniques have become more common, more ambitious, easy to learn/implement, and increasingly sophisticated, so there is a need to find an effective and feasible solution for this problem in the computer security community.

2. Survey of SQL injection

SQL injection is a vulnerability that allows an attacker to alter backend SQL statements by manipulating the user input. An SQL injection occurs when web application accepts user input that is directly placed into SQL statements and does not property filter out dangerous characters.

(more) Advanced SQL Injection by Chris Anley [chris@ngssoftware.com] in 18/06/2002 suggested that the best defence against SQL injection is to apply comprehensive input validation, use a parameterised API, and never to compose query strings on an ad-hoc basis. In addition, a strong SQL Server lockdown is essential, incorporating strong passwords.

SQL Injection Signature Evasion Whitepaper (Imperva) concludes that reliance upon signature protections alone is not a practical defence against SQL injections attacks. A reasonably sized signature database does not provide reliable protection while a comprehensive signature database results in excessive management overhead, dramatic performance limitations, and false positives.

Lateral SQL Injection Revisited by David Litchfield suggest that an attacker needs the CREATE PUBLIC SYNONYM system privilege as a prerequisite to effect this attack, which helps to mitigate the risk. One should not place faith solely in this prerequisite to afford protection, as methods may be found that bypass the need for this privilege in the future. Instead, it is best practice to use variable binding in order to completely mitigate the risk this technique poses.

Lateral SQL Injection A new Class of Vulnerability in Oracle conclude that those functions and procedures that don't take user input can be exploited if SYSDATE is used. The lesson here is always, always validate and don't let this type of vulnerability get into your code. The second lesson is that no longer should DATE or NUMBER data types be considered as safe and not useful as injection vectors.

Analysis of SQL injection prevention using a filtering proxy server by David Rowe conclude that

- Independent of flaws in application coding and database privileges

- Can operate on a separate server with real time analysis
- Another layer of protection

Secure Query Processing By Blocking SQL injection by Dibyendu Aich descreibed that SQL injection is a common technique hackers employ to attack these web-based applications. These attacks reshape the SQL queries, thus altering the behavior of the program for the benefit of the hacker. Show a technique for detecting and preventing SQL Injection Attacks incidents. The technique abstracts the intended SQL query behaviour in an application in the form of an ordered sequence of tokens, as a one-time offline procedure using static analysis of the application code. This database is then validated against the entire different incoming SQL query at runtime to capture all malicious SQL queries, before they are sent to the database server for execution. To minimize searching time and response time it uses the modern processor architecture by perform the searching in a multi threaded way as well as it predict the possible correct list for an incoming query by introducing hit count calculation.

Using a Web Server Test Bed to Analyze the Limitations of Web Application Vulnerability Scanners by David A. Shelly suggested a method to analyze the flaws and limitations of several of the most popular commercial and free/open-source web application scanners by using a secure and insecure version of a custom-built web application. Using this described method, key improvements that should be made to web application scanner techniques to reduce the number of false-positive and false-negative results are proposed.

Techniques and Tools for Engineering Secure Web Applications By Gary Michael this dissertation describes the first formal, realistic characterization of SQL injection and the analyses can detect and block real attacks and uncover unknown vulnerabilities in real world code.

3. Related Work and Observations of SQL Injection Attack

There are four main categories of SQL Injection attacks against databases:

1. SQL Manipulation: manipulation is process of modifying the SQL statements by using various operations such as UNION. Another way for implementing SQL Injection using SQL Manipulation method is by changing the where clause of the SQL statement to get different results.
2. Code Injection: Code injection is process of inserting new SQL statements or database commands into the vulnerable SQL statement. One of the code injection attacks is to append a SQL Server EXECUTE command to the vulnerable SQL statement. This type of attack is only possible when multiple SQL statements per database request are supported.
3. Function Call Injection: Function call injection is process of inserting various database function calls into a vulnerable SQL statement. These function calls could be making operating system calls or manipulate data in the database.
4. Buffer Overflows: Buffer overflow is caused by using function call injection. For most of the commercial and open source databases, patches are available. This type of attack is possible when the server is un-patched.

Mostly web-application developing technologies are susceptible to this attack:
They are JSP, XML, XSL, ASP, JavaScript etc. which can access database.

Detection of SQL Injection vulnerability

Detection of SQL injection vulnerability in a system is very tough task, as SQL Injection is nothing but simple logical game of valid SQLs. So this can be doing by enter each and every possible way the attacker can input the query.

To detect SQL Injection we must have to know about how SQL Injection is possible in an application and what different types of SQL Injection attacks are.
Mainly it can categorize in two stages:
There are two main types of attacks. First-order attacks are when the attacker receives the desired result immediately, either by direct response from the application they are interacting with or some other response mechanism, such as email. Second-order attacks are when the attacker injects some data that will reside in the database, but the payload will not be immediately activated.
Furthermore the classification is also based on commonly two types of attacks:

1. **Login authentication attack:**

Many web-sites or web-application which deals with transaction/view of user related data must have login panel or login page. They must have mainly two field :UserName or UserID and :password and an login/sign in button to login into database. There is also an ‘forget password link’ which sends password to the user who make a request by clicking and input desired fields.
For our example of SQL injection, we will use a hypothetical form which many people have probably dealt with before: the “email me my password” form, which many websites have in case one of their users forgets their password.

Figure 1- Login authentication attack

The way a typical "email me my password" form/link works is this: it takes the email address as an input from the user, and then the application does a search in the database for that email address. If the application does not find anything in the database for that particular email address, then it simply does not send out an email with a new password to anyone. However, if the application does successfully find that email address in its database, then it will send out an email to that email address with a new password, or whatever information is required to reset the password.

(Here below the dark background and with red font is always user input.)

i) First test we do here is to check what error it give on inserting a single quote.

The query become SELECT <column> FROM <table> WHERE <emailfield>=' <desiredEmail >'.

ii) Here an attacker does not know the mail-ID so he made his mind to manipulate the query which can give some result. He use tautology query, in which the where condition is always true. i.e.

SELECT <column> FROM <table> WHERE <emailfield>='abc' and '1'='1';

But unlike the actual query which should return only a single value, this query will return every values of the column since query's where condition is always true. But the actual record taken for operational purpose is the first record returned by the query, or a value taken at random.

Always there are mostly three responses for various input :

- 'Your password has been send to <desiredEmail>'.
- 'The entered Email is incorrect'.
- Some server error.

The first and second responses are sure about that there is a valid SQL run. Or there is no error in the query passed, while the third one is a bad SQL since it will return a server/SQL error.

iii) Guessing column name:

Here in the mind of attacker is sure that there must be email-ID and password in query along with other user login information.

Let he guess a field name as 'email' and try in the query and find out if the SQL is valid or not.

SELECT <column> FROM <table> WHERE field ='x' and email is null; --

He don't care about matching the email address (which is why use a dummy 'x'), and the -- marks the start of an SQL comment. This is an effective way to "consume" the final quote provided by application and not worry about matching them.

It gives response. If it is a server error, that's means SQl is invalid and syntax error can be thrown. So there must be wrong field name guess, try it for other as 'email_address', 'emailID' etc.

If the response is valid, it makes surety that the guessing field is a valid one.

Here 'AND' is use in query to ensure that on valid response there should not be generated response like 'here is your password' and emails from the application to the random user. So to avoid this suspicious activity 'and' is used in query which will always sure that mail is never generated to any user while getting a valid response.

Similarly other fields/columns are also detected through hit and trial method. Say email, password, e_id, name.

iv) Finding the table name:

Table name can be retrieve through several approaches. To accomplish this a 'hit and trial' method is use with using SQL functionality of accessing the fields.

For example, here 'email' is known field and guessed table is 'emp_master'. So the executable query be like :

SELECT <column> FROM <table> WHERE field ='x'andemp_master.email is null; --

If the response of this query is valid or returned as 'Unknown email' then SQL was well formed and table name is properly guessed.

v) Finding some users

Till now table name, column name is guessed. For getting clues about some user, the first place to start, of course, is the company's website to find who is who: the "About us" or "Contact" pages often list who's running the place. Many of these contain email addresses. So the 'LIKE' keyword of SQL helps most to get username. The targeted SQL is build some as :

SELECT <column> FROM <table> WHERE field ='x'ORname like '%ram%'; --

So gradually by refining name a good guess of user name can be achieve.

2. URL query based attack:

a) Finding vulnerable/target web-site:

The vulnerable website have URL ending with queried field like 'id=' or 'fieldno=' etc. We take an example: www.garo.cc

b) Checking the Vulnerability:

In order to check the vulnerability, add the single quotes(') at the end of the url and hit enter. (No space between the number and single quotes).
For eg. http://www.garo.cc/text.php?pageid=16'
If the page remains in same page or showing that page not found or showing some other webpage, then it is not vulnerable.
BUT if it showing any errors which is related to SQL query, then it is vulnerable.

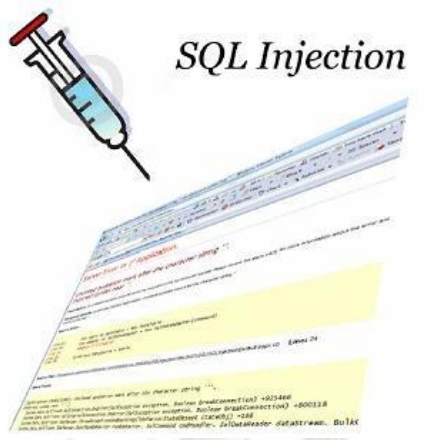

Figure 2-URL query based attack

c) Finding numbers of columns:

With the help of simple and basic commands of SQL we can exploit furthermore.
Put 'order by n' at the end of the URL string. Where 'n' is the number from 1,2,3,4,5, ... and so on.
Change the numbers until we get the error like 'Unknown column'. The number on which you get the error, you make sure the number of column in the table which is used in that query is previous one which give no error.
For eg: http://www.garo.cc/text.php?pageid=1 order by 1 -- (no error)
http://www.garo.cc/text.php?pageid=1 order by 2 -- (no error)
http://www.garo.cc/text.php?pageid=1 order by 3 -- (no error)
http://www.garo.cc/text.php?pageid=1 order by 4 -- (no error)
http://www.garo.cc/text.php?pageid=1 order by 5 -- (no error)
http://www.garo.cc/text.php?pageid=1 order by 6 -- (unknow error)
The error may occur when we put 'order by 6', then we must say that there are 5 columns in that table.

d) Displaying the displayable/vulnerable columns:

For finding columns which is used to display its values on web-page, we have to use 'union select <column 1> ... '
The column which is displayable on the web-page is automatically displays its sequence number in its column field.
For eg. http://www.garo.cc/text.php?pageid=1 union select 1,2,3,4,5 --
here from previous method we have found that there are 5 numbers of columns used, so we take column sequence upto 5 with the union in the URL. The '--' sign denote the comment part from which the database engine can not read further coded query. On carefully observing the front face of the web-page, we can find one or more numbers as a column sequence which are given and executed by us through union query. Say for example we get number 3 on the web-page, then we concluded that the column number 3rd of the used table is the one which is used for displaying data of that table. So that column is vulnerable. An attacker can get many more information from the help of that column.

e) Finding version, database, user:

Let say if in step 'b' the error appeared is related to SQL query and have mention the error message with database as MySQL, then according to MySQL : database(), version()/@@version, user() can be used for getting database, its version and currently login user name.
For example:
http://www.garo.cc/text.php?pageid=1 and union select 1,2,database(),4,5 –
will give the database name on the front face of that web-page where '3' number is displayed.
Similarly for getting version we have to write URL as :
http://www.garo.cc/text.php?pageid=1 and union select 1,2,database(),4,5 --
and for user information:
http://www.garo.cc/text.php?pageid=1 and union select 1,2,user(),4,5 –

f) Finding the table name:

This is the very dangerous situation when for simply using MySQL's SQL commands, we can get the tables used in that schema which is using in the web-site.
For example:
http://www.garo.cc/text.php?pageid=1 and union select 1,2,group_concat(table_name),4,5 from information_schema.tables where table_schema = database() --
Say output of query is admin,garo_news, garo_categories, etc.
Here attacker may take interest in 'admin' table.

Here an attacker has used various MySQL feature as group_concat(table_name), it will concat all the tables name in a string, information_schema.tables, it has stored all information of tables which is used for the particular schema.

table_schema=database(), it is a where clause which takes only currently held schema.

g) Finding the column name:

Here also an attacker can take full use of SQL commands of MySQL. In order to get column name it gives URL as : http://www.garo.cc/text.php?pageid=1 and union select group_concat(column_name) from information_schema.columns WHERE table_name=<The table name which we get from previous step> —

Here also the output of the above URL execution would give all the column name in that particular table.From here the attacker may peek into the data of that table, If he take 'admin' table then probably he can get login information(username,passwordetc).

4. Prevention of SQL Injection

From the previous section, we have seen that there are various methods and types that are used by attacker to get modified SQL query executed. They put some new idea and put it as a valid SQL they use full valid functionality of SQL database used and flaws of developers.

The prevention methods for the above type of attack are discussed as follows.

Prevention for Login Authentication attack and URL based attack:

a. **Reject BAD Input**:

First thing is to sanitize the input before it goes into the application for further query execution. If the input is check before entering in application, then the major part of prevention is done. So in general, the BAD input must be restricted.

b. **Input Datatype**:

In the further series of sanitize input data, we have to check for the datatype of user ID – input variable. Many web-application/database dependent application requires user_ID as a numeric field. Then at those application must implement the user_ID field to be always inputted as numeric values by setting the datatype as number type. It should restrict the input for the characters other than numbers only. For eg. SELECT <columns> from emp_master where userid=:user_id; Here :user_id is a numeric type variable which always allows only numeric type input data.

So no one can write or inject SQL code in those fields.

c. **Input length**

It is wiser to set the input length of input field/variable's length. It always restrict the input parameters/variables to of fixed length and restrict the attacker too for injecting unusual SQL. For eg. User_ID can be put to maximum 20 characters. Also the Password field's length must also restricted to 15 characters only. Definitely these restrictions may not allow any attacker to put long injected SQL in LOGIN fields.

d. **Data-type conversion must be explicit**: Always if required the datatype conversion must be of explicitly done before query execution. For example: If the username is of numeric type in database. But requirement of field(username) on the LOGIN form is of string type. Then just before query execution the username is explicitly converted into numeric type then pass into query variables for further execution.

This method just filters out any unwanted injected SQL.

e. **Exception handling**: The major flaws in developing the code is that it should not properly handled for all type of errors. Each and every code must be properly handled for any type of exception occurs.

Major SQL Injection Attackers rely on errors occurs. They are always waiting for responses occur when any SQL or related query is injected. With the help of responses and errors they obtain much important information as we have seen in previous chapters.

f. **Avoid 'LIKE' in query structure**: Developer should avoid using 'LIKE' in SQL code. As it gives attacker an ease to guess values and data. For Eg: SELECT <column_name> FROM emp_master WHERE name like '%ADMIN%'. The attacker attempts to manipulate the SQL statement to execute as – SELECT <column_name> FROM emp_master WHERE name like '%'. Above query will substitute the input string ADMIN to the query and will search for all the records that have input string anywhere in the name values. If the attacker injects the string then he can get all the sensitivedata.

g. **System monitoring**: A full time DBA can monitor the suspicious query execution and transaction in the system. He might be monitoring or auditing sp_tracexxx files time to time.

h. **Only necessary grant and access are made for application account in database**: If the application running with database administrator's account then it has potential for an attacker to perform crucial commands with database. He can then able to inject many operating system level commands to explore hard disk of server. Similarly if possible where ever only SELECT rights are granted would be only granted. This will greatly helpful to restrict injected transaction or other SQL in system.

i. **Update Server by applying time-to-time patches**: It will avoid buffer overflow. Many

attackers if access systems call then using system calls they make buffer overflow condition. These can be avoided by applying patches and keep server update.

j. **Using Parameterised Queries**: Many other people may suggest it to prevent SQL Injection attack while LOGIN. In this type, the query is passed in prepared statements where it is executed using procedure call. This is standard procedure call which is much trusted.

5. Algorithm of Proposed Solution

a. **Escaping keywords, quotes and comments**: In most of the SQL injection attacks, attacker uses the SQL keywords, quotes and comments in the SQL query and makes the final query as infected. The SQL keywords, quotes and comments are legal and are unsuspected, so the attacker make use of them.
So before query execution the input variables must be sanitize by an procedure which can detect SQL keywords, quotes and comments.
For eg:username field is given as ' ; DROP TABLE emp_master; --

The legitimate query becomes:
SELECT <column_name> FROM emp_master WHERE name='' ;DROP TABLEemp_master; -- ' and password='';
The above query is too much dangerous, since it can delete the LOGIN table as well as user_information table. So before this variable is set into the query for further execution. An algorithm of a procedure which can detect SQL keywords, comments, is developed as follows:

```
procedureis_validate_field(v_input_fieldvarchar)
begin
        declare
        v_checkvarchar;
        begin
                if (v_input_field in ('SELECT','INSERT','DELETE','UPDATE','MERGE','SHUTDOWN','DROP','ALTER','CREATE','WHERE','AND','OR','EXEC','ORDER BY','UNION','GROUP BY','HAVINH','/','--')) then
                        v_check='TRUE';
                else
                        v_check='FALSE';
                end if;
        if v_check='TRUE' then
                message('Retry entries.');
(terminate the process/action and clear the login form and go to first field.);
        endif;
        end;
end procedure is_validate_field;
```

b. **Escaping all encoding in query execution**: If SQL keywords are escaped before query application, then the attackers put the encoding method for injecting SQL in legitimate query. Since simple keywords can be caught then the attackers make their ascii code and put the encoding along with as injected SQL.

Also for an example:
'; INSERT INTOemp_masterVALUES(101,char(0x68)+char(0x61)+char(0x82)+char(0x64)+char(0x70)); --can insert an attacker's oriented user.
So if char,hex,ascii etc encoding are detected in input and then the good input is allow to pass in query for execution, then the effort of attacker got waste. For this an algorithm is developed and coded in a procedure. In which different encodings are detected and prevent the infected data for further execution in query.
The procedure which can detect different encoding is as follows:

```
procedureis_encoded_field(v_input_fieldvarchar)
begin
        declare
        v_checkvarchar;
        begin
if (v_input_field in ('CHAR','ASCII','HEX','NUMBER','(',')')) then
                        v_check='TRUE';
                else
                        v_check='FALSE';
                end if;
                if v_check='TRUE' then
                        message('Retry entries.');
(terminate the process/action and clear the login form and go to first field.);
                endif;
        end;
end procedure is_encoded_field;
```

c. **Applying Encrypted data technique**:
For LOGIN form, there are two fields: 1. Username 2. Password.
But for LOGIN/sign in one should be a registered member. Here in the technique when user register himself then server receives request from user and register as a new user. This is maintain in a user information table. For eg.we take that table as 'emp_master'. The table contains three fields, 1. Username 2.Password 3.Encrypted key.
Here the 'Encrypted key' is generated by system and must be unique for all registered users. This 'Encrypted key' is generated and saved in table at the time of user registration and use the 'username and password' field in its forming, its formation is initiated through calling a function:
For example: At the time of user registration, the following insert query is execute for inserting new

user.

```
INSERT INTO emp_masterVALUES('RAM','ram_password',function_en_key('RAM','ram_password'));
```

Here 'function_en_key' is a function which generates 'Encrypted key'. The function's algorithm is as follows:

```
char function_en_key(:username, :password)
begin declarev_enc_keyvarchar; begin v_enc_key := any_encryption_technique(:username||:password);
        RETURN(v_enc_key);
    end;
end function_en_key;
```

At the time of user login or sign in, username and password should be matched with the username and password in table stored in server, along with the 'Encrypted key' which is also resides in the table. If comparison is successful then the user is allow to Login into the application otherwise make him retry. The comparison procedure is as follows:

```
-- :username and :password are user supplied fields.
procedureis_encoded_field(:username, :password)
begin declare v_checkvarchar; begin
        SELECT 'x' INTO v_check
FROM emp_master
        WHERE
encrypted_key=function_en_key(:username, :password) and name=:username and password=:password;
    (exception handling)
        if (v_check = 'x') then
            message('Successful login');
        else
            message('Try again.');
(terminate the process/action and clear the login form and go to first field.);
end if;
    end;
end procedure is_encoded_field;
```

6. Conclusion and Future work

In this paper, we have described a brief study of SQL injection as well as a solutionfor preventing SQLInjection Attacks. To perform this assessment firstly identified the detection of SQL Injection vulnerability. SQLInjection Attacks can be introduced into anapplication and identified which method was able to hold which mechanism. Lot of the techniques have trouble handling attacks that acquire advantage of poorlycodedstored procedures and SQL queries cannot handleattacks. This variation could be clarified by the detailthat focused on Prevention of sql injection.

Future work should focus on optimized and evaluating thetechniques correctness and usefulness in practice. Practicalestimation will be performing which permitcomparing the performance of the different techniqueswhen they are subjected to real world attacks andvalidinputs.

REFERENCES

[1] Advanced SQL Injection in SQL Server Applications
[2] SQL Injection Attack and Defense PacketSource Security White Papers
[3] Useful stuff_ SQL-Injection Attacks on the example
[4] SQL Injection Cheat Sheet
[5] An Authentication Mechanism to prevent SQL Injection Attacks
[6] Lateral SQL Injection Revisited Final (1)
[7] SQL Injection Signature Evasion Whitepaper
[8] Data-Mining with SQL Injection and Inference
[9] Steve Fried's Unixwiz.net Tech Tips
[10] Prevent SQL Injection in Asp.net
[11] Prevent SQL Injection in Asp.net
[12] Robert J. Hansen MeredithL.Patterson
[13] SQL injection Attacks and Defense
[14] An Authentication Mechanism against SQL Injection