EUROPEAN ACADEMIC RESEARCH
Vol. IV, Issue 11/ February 2017

Impact Factor: 3.4546 (UIF)
DRJI Value: 5.9 (B+)

ISSN 2286-4822
www.euacademic.org

SQL Injection Attacks and Bypass Filtration

ISLAM ABDALLA MOHAMED ABASS
Department of Computer Science
Al Jouf University
Qurayyat, Saudi Arabia

Abstract:

SQL injection attacks are a serious security threat to Web applications. They allow attackers to obtain the data stored in the database. To address this problem, i present an extensive review of the various types of SQL injection attacks known to date. For each type of attack, i provide descriptions of how attacks of that type could be performed and present a methodology to prevent SQL injection attacks. I also created a program to scan any website for SQL injection Vulnerability even if it the website use filtration to prevent SQL injection.

Key words: Web Application, PHP, Structured Query Language Injection (SQLI), Vulnerabilities.

1. INTRODUCTION:

Nowadays Companies can see the Internet as a business opportunity for advertising, communication, promotional, and E-commercial channel. Individuals can see the Internet as a mobile workplace, social activity, E-learning or even for fun. Because of that currently web applications are playing a magnificent role in providing vital information to users around the global. According to Tian et al. [1] web application software security becomes more and more important as a result

information access through web applications. Recent investigations show that web application vulnerabilities have become the largest security threat. The Web Sense Security Report shows that in the first half of year 2008, the most popular websites that have been utilized by various hackers to run malicious code were above 75%. Detecting and solving vulnerability is the effective way to enhance web security. Web application commonly has three tiers: presentation, logic, and storage. The most important tier is the storage, it contains all the sensitive data that web application used and because of that hacker usually focuses on it using different type of attack but the most devastating one in SQL injection attack.

2. THE DANGER OF SQLI ATTACKS [2, 16]

The truth is most flaws in application security can't be fully exploited without complementary flaws in the infrastructure. In November 2005, a teenage hacker broke into Information Security magazine using a SQL injection attack. Once in, he used his access to steal customer, member, and commercial information from the site, this isn't just an old example because SQLI is still at the top ten web application attacks now a day. According to IMPERVA web application attack report published in July 2013, retail applications suffered twice as many SQL injection attacks. Their analysis revealed that SQL injection attacks on retail applications were more intense, both in terms of number of attacks per incident and duration of an incident, so SQL injection attacks are becoming significantly more popular amongst hackers, according to recent data.

3. SQLI ATTACKS [3, 4]

SQL injection has probably existed since SQL databases were first connected to Web applications, its attack that consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit

can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. The attacker injected SQL commands into data-plane input in order to effect the execution of predefined SQL commands. This vulnerability is not just web related but can also occur in desktop applications that use SQL server backends. The detectability of these vulnerabilities depends on the complexity of the application in question.

For example, a typical form may ask for an ID and create a URL: http://www.somewebsite.com/?id=somedata. SQL Query behind this site could be " select text from news where id=$id". An attacker using SQL Injection may enter "some data or 1=1". If the web application does not properly validate or encode the user-supplied data and sends it directly to the database, the reply to the query will expose all ids in the database since the condition "1=1" is always true. This is a basic example, but it illustrates the importance of sanitizing user-supplied data before using it in a query or command.

4. SQL INJECTION TYPE

In this section we will discuss SALIA type. SQLIA is not new subject attackers can perform different type of attacks. These attacks my not performed in isolation depending on the goals of the attacker and the type of protection he face.

4.1 Tautologies [5, 6, 7]

This attack works by inserting an “always true” fragment into a WHERE clause of the SQL statement. This is often used in combination with the insertion of a double dash --, #, or try to balance the query. This will cause the remainder of a statement to be ignored, ensuring extraction of largest amount of data. An attacker can use this technique to bypass authentication pages

or even extract sensitive data. Example: In this example attack, an attacker submits single quote, OR1=1 followed by hyphen or double hash for the login input field (the input submitted for the other fields. The resulting query is: SELECT accounts FROM users WHERE login=" or 1=1 -- AND pass=" AND pin= .The code injected in the conditional (OR 1=1) transforms the entire WHERE clause into a tautology. The database uses the conditional as the basis for evaluating each row and deciding which ones to return to the application. Because the conditional is a tautology, the query evaluates to true for each row in the table and returns all of them. In our example, the returned set evaluates to a non-null value, which causes the application to conclude that the user authentication was successful.

Even the web application programmer faltered the (=) character the attacker can use like statement, SELECT accounts FROM users WHERE login=" or 1 like 1 -- AND pass=" AND pin= .The attacker will succeed and get the same result.

4.2 Error base injection [1, 3, 4, 11, 12]

Before SQL injection was well understood, developers were advised to disable all verbose error messages in the mistaken belief that without error messages the attacker's data retrieval goal was next to impossible to achieve. In some cases developers would trap errors within the application and display generic error messages, whereas in other cases no errors would be shown to the user. So error based SQL injection takes advantage of poor error handling in web page processing. The idea behind this attack is to gather information about the database and the website behavior. First the attacker try to break the query by comment it, and this will make the application server generate error message. Most of these messages have Additional information, to help programmers debug their applications, further helps attackers gain information about the schema of the back-end database.as example let's assume that we have this web site:

http://localhost/ sqlia/Less-1/?id=1, to break the query we just have to add single quote and we will get this error message (You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '' LIMIT 0,1' at line 1).from the error message we know that we are dealing with MySQL DBMA. Note that the attacker can inject different text not just single quote. The next step is to balance the query by add another single quote or just add --+,# to comment the rest of the query .now the attacker could use order by command to get the number of column: http:// localhost/sqlia /Less-1/?id=1' order by 1 --+.The attacker keep entering number of column until he get this error message (Unknown column '4' in 'order clause). That mean the number of column is 3 (http://localhost/sqlia /Less-1/? id=1' order by 1, 2, 3 --+). Using this information, an attacker can then create further attacks that target specific pieces of information like database name, DBMA version or table content.

4.3 Union Query [4, 5, 8, 9, 11]

In union-query attacks, an attacker may use SQL injection to extract database user, version, name, table information etc. from another table by using UNION within the query. With this technique, an attacker can trick the application into returning data from a table different from the one that was intended by the developer .The attacker first injects into persistent storage such as a table row and collects some data and using this he will do further to collect more from the database. This statement treated as second query, so to make it work we can make the first statement fouls. The result of this attack is that the database returns a dataset that is the union of the results of the original first query and the results of the injected second query. This example will return the version of the DBMS: http://localhost/sqlia/Less-1/?id= -1' union select 1, version (), 3.The result of this injection will be 5.5.16.

4.4 DOUBLE QUERY [1, 13, 14]

This is a very handy technique to have in your arsenal as there will be many times Union injections just won't work and blind injections are very time consuming and hard to interpret sometimes. Double Query SQL Injection is a method for querying SQL databases by using two queries together combined in a single query statement. This basically ends up confusing the backend database and causing errors to be thrown. The errors received will contain the information we are trying to extract, just like previous error-based SQL injection.

Although is definitely faster than Blind & Time-Based injections, attackers will not have the ability to access anything using GROUP_CONCAT ( ) which means we will need to heavily rely on CONCAT( ) and the LIMIT feature to get all of the info from the database. as example: http://localhost/sqlia/Less-1/?id=1' and(select 1 FROM(select count(*), concat ((select (select concat(database( ))) FROM information_schema.tables LIMIT 0,1),floor(rand(0)*2))a FROM information_schema.tables GROUP BY a)b)--+

The technique used by attackers in double injection is very simple. They use both floor ( ) and rand ( ) to query information_schema. tables which are being nulled out in this request as floor (rand (0)*2) is null, which allows the rest of our request to be processed and return the current database name. The basic syntax will repeat itself so you will pick it up over time if it doesn't catch on right away. Now that we know it is vulnerable we can test for additional databases, as well as version info and user info. Once the basic system info is grasped we can move on to grabbing tables, columns, and finally extraction of data.

4.5 Blind injection [1, 4, 12, 15, 21]

In normal SQL injection hackers rely on error messages returned from the database in order to give them some clues on how to proceed with their SQL injection attack. But with blind SQL injection the hacker does not need to see any error

messages in order to run his/her attack on the database – and that is exactly why it is called blind SQL injection. So, even if the database error messages are turned off a hacker can still run a blind SQL injection attack. In this situation, the attacker injects commands into the site and then observes how the function/response of the website changes. By carefully noting when the site behaves the same and when its behavior changes, the attacker can deduce not only whether certain parameters are vulnerable, but also additional information about the values in the database.

4.6 Boolean Blind Injection

In this technique, the information must be inferred from the behavior of the page by asking the server true/false questions. If the injected statement evaluates to true, the site continues to function normally. If the statement evaluates to false, although there is no descriptive error message, the page differs significantly from the normally-functioning page.

This technique is very slow because you must take in consideration all the possibility.as example this will ask the if the database first letter is (e) : http://localhost/sqli /Less-5/?id=1' and (ascii(substr((select table_name from information_schema_tables),1,1)))=101 --+ .

We can speed the technique by use binary search. The attacker is asking if the ASCII value of the character is greater-than or less-than-or-equal-to the value of X. If the value is greater, the attacker knows this by observing an additional 5 second delay in the response of the database. The attacker can then use a binary search by varying the value of X to identify the value of the first character. So as example we can ask the database first name in ascii is larger than 112 which equal to the letter(r), by doing this we can get to the first letter faster.

4.7 Timing blind Injection Attacks

A timing attack allows an attacker to gain information from a database by observing timing delays in the response of the database. This attack is very similar to Boolean blind Injection, but uses a different method of inference. It relies on the database pausing for a specified amount of time, then returning the results, indicating successful SQL query executing. Using this method, an attacker enumerates each letter of the desired piece of data using the following logic:

If the first letter of the first database's name is an 'A', wait for 10 seconds. If the first letter of the first database's name is a 'B', wait for 10 seconds etc. There are 2 functions in MySQL which gives significant delay

1. Sleep ( ).
2. Benchmark ( ).

Sleep ( ) gives you strictly fixed amount of delay while benchmark ( ) varies amount of delay depending on load on the server. The detection criteria for this type of the injection are going to be the delayed response of the website after injection.

More than detecting SQL injection using delay operation, it is more used to dig information out of database. Confirmation of the information exists in database is indicated by the delay in response given by database after injection. It is more of the ex-filtration methodology than discovery. Let us consider an example how to take out information from database using delay. Let us inject something interesting using sleep ( ) in query.

http://localhost/sqli /Less-5/? Id=1' UNION select if (SYSTEM_USER='root', sleep (100), 1);-- This time our injection string is bit lengthy but very much effective and damaging. We have injected " 1' UNION select if (SYSTEM_USER='root', sleep (100), 1); -- " as an injection. Union clause will combine the following query input and give combined result. If will check if logged in system user is 'root' or not. If the user is root then it will execute sleep for give amount of time else it will just give '1' to union clause.

4.8 Common Blind SQL Injection Scenarios

Here are three common scenarios in which blind SQL injection is useful:

1. When submitting an exploit that renders the SQL query invalid a generic error page is returned, while submitting correct SQL returns a page whose content is controllable to some degree. This is commonly seen in pages where information is displayed based on the user's selection; for example, a user clicks a link containing an id parameter that uniquely identifies a product in the database, or the user submits a search request. In both cases, the user can control the output provided by the page in the sense that either a valid or an invalid identifier could be submitted, which would affect what was retrieved and displayed.
2. A generic error page is returned when submitting an exploit that renders the SQL query invalid, while submitting correct SQL returns a page whose content is not controllable. You might encounter this when a page has multiple SQL queries but only the first query is vulnerable and it does not produce output. A second common instance of this scenario is SQL injection in UPDATE or INSERT statements, where submitted information is written into the database and does not produce output, but could produce generic errors.
3. Submitting broken or incorrect SQL does not produce an error page or influence the output of the page in any way. Because errors are not returned in this category of blind SQL injection scenarios time-based exploits or exploits that produce out-of-band side effects are the most successful at identifying vulnerable parameters.

5. Defending Against SQL Injection Attacks [1, 4, 17]

The good news is that there actually is a lot that web site owners can do to defend against SQL injection attacks. Although there is no such thing as a 100 percent guarantee in

network security, Formidable obstacles can be placed in the path of SQL injection attempts.

1. **Comprehensive data sanitization**: Web sites must filter all user input. Ideally, user data should be filtered for context. For example, e-mail addresses should be filtered to allow only the characters allowed in an e-mail address, phone numbers should be filtered to allow only the characters allowed in a phone number, and so on.
2. **Use a web application firewall**: A popular example is the free, open source module ModSecurity which is available for Apache, and Microsoft IIS web servers. ModSecurity provides a sophisticated and ever-evolving set of rules to filter potentially dangerous web requests. Its SQL injection defenses can catch most attempts to sneak SQL through web channels.
3. **Limit database privileges by context**: Create multiple database user accounts with the minimum levels of privilege for their usage environment. For example, the code behind a login page should query the database using an account limited only to the relevant credentials table. This way, a breach through this channel cannot be leveraged to compromise the entire database.
4. **Avoid constructing SQL queries with user input**: Even data sanitization routines can be flawed. Ideally, using SQL variable binding with prepared statements or stored procedures is much safer than constructing full queries.
5. **Craft error messages carefully**: Hackers can and will use your own error messages against you to better dial in future attacks. That's why both the development team and DBAs need to think about the error messages they're returning when users input something unexpected.
6. **Stored Procedures Protect Against SQL Injection**: The usefulness of a stored procedure as a

protective measure has everything to do with how the stored procedure is written. Write a stored procedure one way, and you can prevent SQL Injection. Write it another way, and you are still vulnerable.

The wrong way

Suppose the verifyUser stored procedure was created by dynamically building a SQL string within the stored procedure, like this:

```
CREATE PROCEDURE verifyUser

@username varchar(50),
@password varchar(50)
AS
BEGIN
  DECLARE @sql nvarchar(500);
  SET @sql = 'SELECT * FROM UserTable WHERE UserName
= ''' + @username + '''
  AND Password = ''' + @password + ''' ';

  EXEC(@sql);
END
GO
```

Now, when I execute my PHP script with this input…

…the SQL that the stored procedure executes is this:

```
SELECT * FROM UserTable WHERE UserName = 'Brian' --'
AND Password = 'any password'
```

The last half of the query is commented out! As long as my user name matches some user name in the database, I'm in.

By building the SQL query as a string in the stored procedure and concatenating parameter values in that string, I run the same risks that are inherent in concatenating parameter values in application code – I'm vulnerable to SQL injection. I admit that building a query dynamically as shown in the stored procedure above is somewhat contrived, but it is meant to show what is possible (and what NOT to do). Fortunately, avoiding the problem above is easy…

The right way

Now suppose the verifyUser stored procedure was created like this:

```
CREATE PROCEDURE verifyUser

@username varchar(50),
@password varchar(50)
AS
BEGIN
  SELECT * FROM UserTable WHERE UserName = @username AND Password = @password;
END
GO
```

Now, an execution plan for the SELECT query exists on the server before the query is executed. The plan only allows our original query to be executed. Parameter values (even if they are injected SQL) won't be executed because they are not part of the plan. So, if I submit a username like I did in the example above (Brian' --), it will be treated as user input, not SQL code. In other words, the query will look for a user with this password instead of executing unexpected SQL code.

6. AUTOMATIC SQL INJECTION [18, 19]

SQLMAP: it's an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections.

Havij: it's an automated SQL Injection tool that helps penetration testers to find and exploit SQL Injection vulnerabilities on a web page.

It can take advantage of a vulnerable web application. By using this software, user can perform back-end database fingerprinting, retrieve DBMS login names and password hashes, dump tables and columns, fetch data from the database, execute SQL statements against the server, and even access the underlying file system and execute operating system shell commands.

The distinctive power of HAVIJ that differentiates it from similar tools lies in its unique methods of injection. The success rate of attack on vulnerable targets using HAVIJ is above 95%. Netsparker: is a false positive free web application security scanner that can be used to identify web application vulnerabilities such as SQL Injection and Cross-site scripting in your web applications and websites.

7. BYPASS FILTRATION [4, 20]

Functions and keywords filtering prevents web applications from being attacked by using a functions and keywords black list. If an attacker submits an injection code containing a keyword or SQL function in the black list, the injection will be unsuccessful.

However, if the attacker is able to manipulate the injection by using another keyword or function, the black list will fail to prevent the attack. In order to prevent attacks, a number of keywords and functions has to be put into the black list. However, this affects users when the users want to submit input with a word in the black list. They will be unable to submit the input because it is being filtered by the black list. The following scenarios show cases of using functions and keywords filtering and bypassing techniques. We will take PHP language as example of filtration by using preg_match function or preg_replace to filtrate the input.

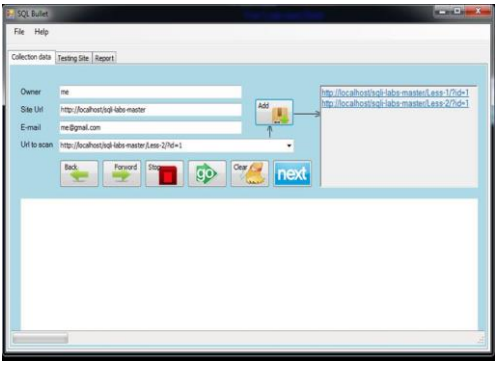
Key word filter	**PHP filter code**	**Filtered injection**	**Bypassed injection**
and	preg_match('/(and)/i', $id)	1 and 1 = 1	1 && 1 = 1
Or	preg_match('/(or)/i', $id)	1 or 1 = 1	1 || 1 = 1
and, or, union	preg_match('/(and|or|union)/i', $id)	union select user, password from users	1 || (select user from users where user_id = 1) = 'admin'
and, or, union, where	preg_match('/(and|or|union|where)/i', $id)	1 || (select user from users where user_id = 1) = 'admin'	1 || (select user from users limit 1) = 'admin'
and, or, union, where, limit, group by	preg_match('/(and|or|union|where|limit|group by)/i', $id)	1 || (select user from users group by user_id having user_id = 1) = 'admin'	1 || (select substr(gruop_concat(user_id),1,1) user from users) = 1
Whitespace	preg_replace('/[\s]/',"", $id)	union select user, password from users	Union%a0select%a0 user,%a0password %a0from%a0users

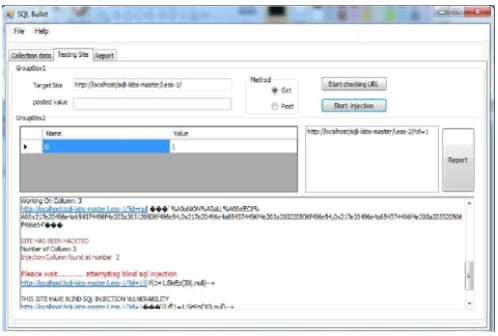
From the above table we can see it's very easy to bypass filtration. Attackers can use manual SQL injection to bypass filtration by using the technique in the above table or others. So filtration is not enough to stop SQL injection. But the problem is not bypass filters because if you detected SQL vulnerabilities by using Automatic SQL injection you can fix it. So we must check first the capability of the software that do Automatic SQL injection to determine if it can bypass filtration.

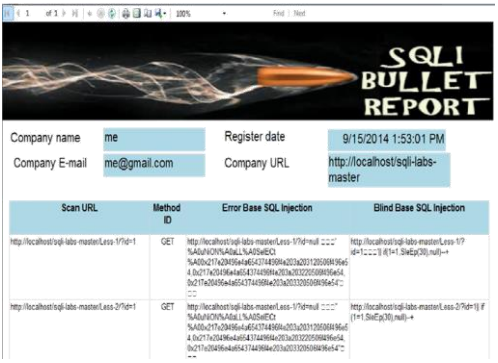
8. PROPOSED SOLUTION:

Most of the software used to scan for SQL vulnerability doesn't by bass filtration even if it's weak. At the same time depending on filtration to protect against SQLI is easy to bypass it, because of that I create software than can bypass most filtration by changing the input parameter to ASCII code, and inject it in the website. I have called it SQL Bullet. It can use Get, or Post in this process. SQL Bullet use based error injection first and then tries Blind SQL injection. SQL injection can scan for SQL vulnerability for more than one page, even if the pages using post, or get. This program can be used to check for SQL vulnerability by using automatic SQLI. The program has three steps.

Step 1: Entering website data
In this step user fill application data about the site and it have more than one page to scan. The program has its own web browser to check if the web site is valid.
Step 2: attempt to hack the site
In this step the program tries to hack the site and give information about the process.
Step 3: report
In this step the program give report about the hack attempt and the code used to hack the site.







9. CONCLUSION AND FUTURE WORK

In this paper, i have presented current techniques of SQL injection as well as a solution methodology for preventing SQLIAs. To perform this evaluation, I first identified the various types of SQLIAs Known to date. To show how dangers SQLIAs can be I studied the different mechanisms used for SQLI and Common Blind SQL Injection Scenarios. Also discuss the mechanisms used in protection. The program I designed can scan any website for SQLI vulnerability and give report about it. It can bypass most type of filtration by convert the injected parameter to ASCII code. Future work focuses on bypass weak firewall used to prevent SQLI.

10. REFERENCES

[1] Tian wei , Yang Ju – Feng, Xu Jing, S I Guan – Nan, "Attack Model Based Penetration Test for SQL Injection Vulnerability", Computer and software conference workshops (COMPSACW), 2012 IEEE 36th .

[2] Teenage hacker facing court case for data theft." Taipei Times, January 22, 2006

[3] OWASP – Open Web Application Secuirty Project. Top ten most web application vulnerabilities. https://www.owasp.org/index.php/Testing_for_SQL_Injection_%28OWASP-DV-005%29#out_of_band exploitaion_technique,April 2014

[4] SQL Injection Attacks and Defense. Justen clarke, Dave hartely, Joseph Hemler, Hroon , (2009).

[5] W. G. J. Halfond, A. Orso and P. Manolios,"Using Positive Tainting and Syntax-aware Evaluation to Counter SQL Injection Attacks", Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2006.

[6] A. Ciampa, C.A. Visaggio and M.D. Penta, " A Heuristic-based Approach for Detecting SQL-Injection Vulnerabilities in Web Applications," in ACM Proceedings of the ICSE Workshop on Software Engineering for Secure Systems, pp. 43-49, 2010.

[7] http://blog.didierstevens.com/2010/02/02/quickpost-quasi-tautologies-sql-injection/

[8] C Anley. Advanced SQL Injection in SQL Server Applications. White Paper Next Generation Security Software Ltd., 2002. http://www.nextgenss.com/papers/advanced sql injection.pdf.

[9] S. McDoland, SQL Injection. Modes of Attack, defence and why it matters. White paper, GovernmentSecurity.org, April 2002

[10] Chris Anley. "(More) Advanced SQL Injection". Chris Anley. NGS Software URL: http://www.nextgenss.com/ pagers/ more_advanced_sql_injection.pdf

[11] Debasish Das, Utpal Sharma , D.K. Bhattacharyya, An Approach to Detection of SQL Injection Attack based on Dynamic Query Matching, paper 2010
[12] Wisdom Kwawu Torgby, Nana Yaw Asabere," Structured Query Language Injection (SQLI) Attacks:Detection and Prevention Techniques in Web, International Journal of Computer Applications 2013.
[13] http://kaoticcreations.blogspot.com/p/double-query-based-sql-injection.html.
[14]http://www.madleets.com/Thread-SQL-INJECTION-Double-Query-Error-Based.
[15] http://flagdefenders.blogspot.com/2012/12/sql-injection-part-3-time-based.html.
[16] Imperva, SQL Injection Signature Evasion Whitepaper the Wrong Solution to the Right Problem, 2004
[17] http://www.esecurityplanet.com/hackers/how-to-prevent-sql-injection-attacks.html.
[18] http://sqlmap.org/.
[19] https://www.netsparker.com/sql-injection/.
[20] http://www.exploit-db.com/papers/17934/.
[21] Martin G. Nystrom, "SQL Injection Defenses", O'Reilly Media, 2007.