2013 International Conference on Informatics and Creative Multimedia

A taxonomy of SQL Injection Attacks

Amirmohammad Sadeghian, Mazdak Zamani, Shahidan M. Abdullah
Advanced Informatics School
Universiti Teknologi Malaysia
Kuala Lumpur, Malaysia
i@root25.com , mazdak@utm.my , mshahidan@ic.utm.my

***Abstract*— Nowadays web applications play an important role in online business including social networks, online services, banking, shopping, classes, email and etc. Ease of use and access to web application make them more popular in offering online services instead of in person services. a simple user just need a computer and an internet connection to access web application and use online services provided by that application. There is one core in common between all dynamic web application and that is their need to use a database to store information inside that and retrieve that information upon the user request or add, edit and delete them. Among all database types, rational databases are very popular. Most of relational database management systems such as MySQL, Oracle, MS SQL Server, MS Access, Postgres use SQL as their language. Flexibility of SQL makes it a powerful language. It allows the user to ask what information he wants without having any knowledge about how the information will be fetch. However vast use of SQL based databases make it the center of attention of hackers. SQL injection attack is a well-known security threat to database driven web applications. A successful SQL injection attack reveals critical confidential information to the hacker. In this paper first we provided background information on this vulnerability. Next we present a comprehensive review of different types of SQL injection attack. For each attack we provide an example that shows how the attack launches. Finally we propose the best solution at development phase to defeat SQL injection and conclusion.**

Keywords- SQL Injection; SQLIA; Web Application Vulnerability; Information Security;

I. INTRODUCTION

Currently all businesses prefer to move on from the "in person services" to "online services", and this is due to cost of labor and possible mistakes that might happen during the work. A well-established web application can easily satisfy needs of companies that want to serve their customers online. Web applications that use database to store and retrieve data are named “Database Driven Web Applications”. One of the most common types of databases is relational database. Structured Query Language (SQL) is a type of programming language which created for handling and controlling data in the relational database management systems (RDBMS).

Flexibility of SQL makes it a powerful language. It allows the user asks what information he wants without having any knowledge about how the information will be retrieved. This will help users without knowledge of programming to run a query. However, SQL includes powerful functions that allow professional users with programming skills to make more complex queries and execute them to achieve higher performance. Like other programming languages SQL let the user to use inline comments in the code, even between statements. SQL also give the ability of concatenation and combining to character or values.

Databases are the main storage of potential confidential information online and this is a good motivation for attackers to target them. SQL injection is one of attacks that attackers use to breach the database security. Among many types of web application vulnerabilities, SQL Injection is the most dangerous one [1]. The main motivation of this paper is to inform web application developer and security researcher about possibilities of SQL injection attack. Having a good understanding of how different types of SQL injection attacks works can be helpful in writing a more defect free code.

The rest of this paper is organized as follows. Section 2 explained the background of SQL injection problem. Section 3 explained different types of SQL injection attack following by an example. Section 4 proposed the best solution at coding phase to defeat SQL injection attack. And section 5 is the conclusion.

II. SQL INJECTION PHENOMENON

SQL injection is a type of attack which attacker inserts a malicious SQL query into the web application by appending it to the input parameters [2]. SQL injection occurs when the developer used dynamic queries which are concatenated with variables from the user side. In lack of strong input validation the malicious SQL query will insert into the web application and instead of variables, concatenate itself with the legitimate query. And send to the database management system for execution. In result the malicious query will be executed.

For instance we assume that we have a webpage that receive an integer variable as news id and show the related news to that Id.

```
http://www.domain.com/news.php?nid=170
```

Attacker append the “‘ OR ‘1’= ‘1 “ to the end of the URL:

```
http://www.domain.com/news.php?nid=170’OR‘1’=
’1
```

In result of opening this address the PHP web page will return all the news without considering the id of news. This is because the "1=1" is a true condition and "OR" will make the whole condition true, no matter which statement is true.

III. SQL Injection Classification

A. Tautologies

This type of SQL injection attack works by making the "WHERE" clause always true, And this will result in bypassing the condition inside the SQL statement. Attackers mostly use tautology SQL injection to bypass the authentication. They also add inline comment signs to ignore the remaining part of the statement to achieve to the highest amount of the result in return with the lowest range of conditions [3]. Mostly SQL tautologies are comes handy when the attacker try to force a SQL statement to return all records, by ignoring all WHERE conditions. The most common tautology is "or 1=1". It will put another condition by concatenating the "OR" and the "1=1" criteria that always is true so the result of the whole condition will be true [4].

Below is a SQL query for fetching all columns that their username are equal to "Administrator" and their pass is equal to "root".

```
SELECT * FROM TABLE_USERS WHERE USERNAME = 
'ADMINISTRATOR' and PASS = 'ROOT'
```

Below is a variable value that is injected as a SQL Injection tautology attack.

```
' OR 1=1 --
```

And below is the final SQL query in result of the concatenation of the outside variable and the main SQL statement. Running this query will return all rows from the "TABLE_USERS" that their username is "ADMINISTRATOR" without considering their password.

```
SELECT * FROM TABLE_USERS WHERE USERNAME = 
'ADMINISTRATOR' and PASS = '' OR 1=1 -'
```

B. Illegal/Logically Incorrect Queries

In this technique the aim of attack is to collect important information about the schematic and structure of tables and fields inside the database. Attacker later will use the collected data for launching another attack in more details [5]. Basically the mechanism of this attack works by injecting wrong or incorrect SQL query into the web application and web application will return some information about the database in form of error message. Inappropriate handling of errors can leads to showing internal database error messages to the attacker. These kinds of messages will reveal critical information about the database structure and attacker will use them to conduct another attack with higher impact on the website. Even in some cases if the web application limit the output errors or the way which they appearing to avoid showing the structure of the database, the fact that an error show or don't show can be enough informing for the attacker.

For instance in the following attack, the attacker insert "ABCD" " in the username field and because the "(Double Quote) will break the structure of the SQL statement in result Database will return an error message that reveal another column name inside the query string. Now attacker know that there is a column exist in database named "Password".

```
Input (username): 'ABCD"
Input (password): 'TEST"

Sql: SELECT * FROM USERS WHERE username = 
'ABCD"' AND password = 'TEST'

Result: "Incorrect syntax near 'ABCD'. 
Unclosed quotation mark after the character 
string '' AND Password='TEST''."
```

In result of Incorrect Queries attack two type of error might return, logical and syntax. Logical errors use to fetch the name of the columns or tables. But syntax errors show which parameters are open to injection attack.

For securing the web application against this attack the error reporting system should be disabled or properly configured to show only limited errors that don't reveal critical information about the database structure.

C. UNION Query

The UNION operator in SQL language is used to join two independent queries together. In union query, attacker uses "UNION" to extract data from other tables by injecting another select query and unioning that with original SQL statement. By using this method attacker force the database to return result from extra tables other than the one defined in the legitimate SQL query [6].

For extracting data from the database by using union, attacker need to have the structure of the database such as table names and field names that later can build the secondary query based on these information and join it with the original query string. In the following example attacker insert a union SQL query at the end of the URL in the address bar of the browser.

```
Original URL:http://www.example.com/news.php?
newsid=340

Manipulated URL: http://www.example.com/news
.php?newsid=340 UNION SELECT CreditcardNo, 
PinNo FROM CreditCardTable
```

In result of the SQL injection the query will look alike below:

```
SELECT NewsTitle, NewsBody FROM News
WHERE NewsID = '340' UNION SELECT 
CreditcardNo, PinNo FROM CreditCardTable;
```

Consequently database will return two columns. The content of these columns are join of the results from the first query and the second query. In this example the first row is fetch from news table and second and third rows are fetching from the credit card table

TABLE I. An Example Dataset Result of Union Query Attack.

News Title 340	This is the example news body….
100 340 955 888 333	398
230 110 715 701 325	102

A successful Union Query attack can reveal the whole structure of the database and all data inside that. Below is a flow of a full union query attack and it shows that an attacker with zero background knowledge about the current database can find Table, Column and field names.

In the following example in the four first attempt attacker try to understand, how many columns are selected in the legitimate query. This is due to the fact that for a successful union query, attacker needs to know how many columns are selecting in the original query. And when he wants to build a new query and join it with the original query he should select the same amount of columns from the secondary table.

```
URL: http://example.com/news.php?id=10 UNION
SELECT ALL 1--
ERROR: All queries in an SQL statement
containing a UNION operator must have an
equal number of expressions in their target
lists.

URL:http://example.com/news.php?id=10  UNION
SELECT ALL 1,2--
ERROR:All queries in an SQL statement
containing a UNION operator must have an
equal number of expressions in their target
lists.

URL:http://example.com/news.php?id=10  UNION
SELECT ALL 1,2,3--
ERROR:All queries in an SQL statement
containing a UNION operator must have an
equal number of expressions in their target
lists.

URL:http://example.com/news.php?id=10  UNION
SELECT ALL 1,2,3,4-
NO ERROR
```

In the fourth attempt when he didn't received any error he can understand that there are four columns selected in the original query. From now on he has to select his desire information in form of four columns. For instance following attack will retrieve the database name:

```
http://example.com/news.php?id=10       UNION
SELECT ALL 1,DB_NAME,3,4--
```

D. *Piggy-Backed Queries*

In this type of attack, the attacker will inject an independent query and in result of a successful attack the second query will run after the first original query that already ran. The different of this attack with UNION attack is that the queries will not join each other but they are completely independent. This attack named piggy back because the secondary query will be sent to database under the cover of the first query [6].

Implementing this attack is only possible if the database configured in a way that give this permission to the user to run multiple queries in the same line. This type of attack can be very dangerous because it give the ability to the attacker to add any kind of SQL command he want and run it in the database, which can causes a high impact incident.

Semicolon (;) is playing an important role in this type of attack because attacker use it as a delimiter for the end of the first query and the start of new query. But in some database management systems, the existence of delimiter is not necessary.

In the following example we can see a query which will fetch news from the news table based on 3 conditions of year, author and type.

```
SELECT * FROM news WHERE year='.$year.' AND
author='.$author.' AND type='.$type.'
```

Assume attacker inset the following inputs:

```
Input Year: 2013
Input Author: ; drop table users --
Input Type: public
```

In result of inserting above inputs the following query will be made, this query will select all news that are from year 2013 and semicolon finish the first SQL query and second query will delete the table of user's information and "--" will ignore the rest of the query.

```
SELECT * FROM news WHERE year='2013' AND
author=''; drop table users -- '    AND
type='public'
```

E. *Stored Procedures*

Stored procedures are premade portion of SQL queries that are designed to do a specific task. Some of the database systems have their own pre-defined stored procedures for working with operating system. Poor written store procedures are also vulnerable to SQL injection attack and attacker can execute them to achieve his malicious goals. If the attacker can execute database predefined stored procedures, he also will be able to run commands on operating system of the server machine (Privilege escalation).

Currently a lot of developers wrongly believe that using of stored procedures is a good method to avoid SQL injection but this is not true in general. Basically stored procedures can be helpful in avoiding SQL injection by limiting the types of statements that can be passed to SQL parameters. This limitation cannot completely protect the application against SQL injection because still there are some ways to bypass these limitations.

In the following example there is a stored procedure that receive category variable from the outside world.

```
ALTER   PROCEDURE   get_news   (@category
NVARCHAR(50))         AS
BEGIN
    DECLARE @sqlcmd NVARCHAR(MAX);
    SET @sqlcmd = N'SELECT * FROM news WHERE
news_cat = ''' + @category + '''';
    EXECUTE(@sqlcmd)
END
```

Assume attacker insert the following input:

```
sport'; SHUTDOWN; --
```

In result of running this query all news from the news table with category of sport will be selected and after that semicolon will end the first query and second query will shut down the SQL server.

```
SELECT * FROM news WHERE news_cat = 'sport';
SHUTDOWN; --
```

F. *Inference*

In this type of attack, attackers inject the SQL and observe the differences in return from the web application. Basically attack launched by asking questions. For example if the answer is "A" do "M" or if the answer is "B" do "N"[7]. Usually this attack take place when the web application is harden in aspect of error handling and attacker cannot use the error messages.

There are two main attack technique categorized as Inference attacks, "Timing Attacks" and "Blind Injections".

- Timing Attacks:

  In timing attack, SQL injection will let the attacker to understand the answer to his question by the time it takes to load the result page. This type of attack are very likely to works in secure web application because they are relies on the delay that happen in the running of the injected SQL and not the web application output [8].

In the following example attacker ask from database, if database version contains number 4 (like 4) have a 10second delay before you replay and load the page.

```
http://www.MyWebsite.com/news.php?id=12
0   AND   IF(version()   like   '4%',
sleep(10), 'false'))--
```

Above example is in MySQL. The delay command is different based on the vendor of the database management system. Microsoft SQL server uses “WAITFOR” command. Oracle and MySQL use “SLEEP” command for making the delay.

- Blind Injections:

  Blind SQL injection is another technique of inference injection. In this type of attack the attacker will asks a true / false question and he observe the answer based on behavior of web application in response (Also known as content-based). This situation makes the attack process harder for the attacker but cannot avoid the attack. For example we assume the web application is secured enough to avoid showing error messages that contains database structure. But if the application is still vulnerable to SQL injection, the attacker will ask from the web application if first letter of username of database is "a" show the page or vice versa. In the worst case these attempts continue for 26 times until he can guess the first letter of the database name, and consequently he has to repeat the same procedure to retrieve the other letters of the database username.

  In the following example attacker try to guess the table names which exists inside the database. In this case he had two attempts, at the first attempt he asks from the database “Select first row as 1 from the table admin”. Table “admin” is not exists in the database, so the result is false and this result is part of AND condition. Consequently the page will not load because the condition is not satisfied.

```
http://example.com/news.php?id=132   AND
(select 1 from admin limit 0,1)=1
```

  We assume a table named "users" exists in the database, so in the second attempt query can select the first row of the table. And this satisfies the condition and the page will load in result, now the attacker knows that there is a table named "users" exists in the database.

```
http://example.com/news.php?id=132   AND
(select 1 from users limit 0,1)=1
```

G. *Alternate Encodings*

Alternate encoding is not an independent type of attack but it’s a technique that mostly used next to other SQL injection techniques to avoid security system of that web application or network infrastructure from detecting of the attack. In other words, it only used as a cover for other attacks to evade from Intrusion detection systems (IDS) [9].

In this type of attack, encoding techniques such as Base64, ASCII, HEX or Unicode might be used to trick the IDS/IPS by changing the look of the SQL injection query.

Assume we have an intrusion detection system with following signature. In this signature, IDS will look for “ ' or 1=1 – ” , if IDS successfully found the pattern will drop the connection and show an error.

```
alert   tcp   any   any   ->   $HTTP_SERVERS
$HTTP_PORTS (msg: "SQL Injection attempt";
flow: to_server, established; content: "' or
1=1 --"; nocase; sid: 1; rev:1;)
```

In the following examples, we can see the encoding of the same SQL injection string for evasion from IDS.

HEX encoding of ' or 1=1 -- for use in URL:

```
%31%20%4F%52%20%31%3D%31
```

HEX encoding of ' or 1=1 --:

```
&#x31;&#x20;&#x4F;&#x52;&#x20;&#x31;&#x3D;&#x
31;
```

Decimal encoding of ' or 1=1 --:

```
&#49&#32&#79&#82&#32&#49&#61&#49
```

Base64 encoding of ' or 1=1 --:

```
MSBPUiAxPTE=
```

Using comments in the attack is also very common technique of evasion. Comments can change the usual appearance of the query to avoid the detection by IDSs and IPSs [10].

Following example use “/* */” comment to evade. In this example whatever is in between /* and */ will not considered to be execute in MySQL server but it can change the look of the attack in result the IDS cannot detect it.

```
DROP/*comments will goes here*/users =
```

In the following example we can see that using of comment even in the middle of name of the function is also possible and it will not be considered for execution in MySQL. In this example DR/**/OP will act like DROP function, so this flexibility of SQL language will give too much choices to attacker to change the appearance of his attack to evade the detection algorithms.

```
DR/**/OP users
```

IV. PROPOSED SOLUTION

Security researchers proposed wide ranges of solutions to combat with SQL injection. These solutions can be categorized in three main groups of Best code practices, SQL injection detection and SQL injection runtime prevention. Each of them has their own advantages and disadvantages which due to the space limitation we cannot cover them in this study. But the best solution can be the avoidance through writing secure codes. Writing secure code is a part of best coding practices. Among all best code practices, using Parameterized Queries is the most secure and efficient technique.

Parameterized Queries also known as prepared statements. In this technique there are some placeholders in the SQL query for the user variables. Database management system first will compile the SQL statement without considering the placeholders and store the result. Next it will add the variables and compile the statement for the second time. Consequently even if the attacker inserts a malicious query, the database will treat it like an ordinary string.

Parameterized queries make sure that attacker cannot change the SQL query string even in case of using dynamic queries.

V. Conclusion

SQL injection is a dangerous attacking method which can be very sophisticated. In this paper we only cover a simple example of each attack. However each of these attacks can be launch in a more complicated way. A good understanding of SQL injection techniques can help developers to make their applications and the network more secure against this vulnerability. There are many types of defense techniques exist against this attack, but we believe avoidance is the best solution. We strongly suggest to developers to use Parameterized queries for making dynamic queries to avoid SQL injection.

References

[1] (OWASP), “O.W.A.S.P. Top 10 Vulnerabilities.”; Available from: https://www.owasp.org/index.php/Top_10 2013.

[2] Shar, L.K. and T. Hee Beng Kuan, Defeating SQL Injection. Computer, 2013. 46(3): p. 69-77.

[3] Dharam, R. and S.G. Shiva. Runtime monitors for tautology based SQL injection attacks. in Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), 2012 International Conference on. 2012.

[4] Halfond, W., J. Viegas, and A. Orso. A classification of SQL-injection attacks and countermeasures. in Proceedings of the IEEE International Symposium on Secure Software Engineering, Arlington, VA, USA. 2006.

[5] Jie, W., et al. Augmented attack tree modeling of SQL injection attacks. in Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on. 2010.

[6] Medhane, M.H.A.S., Efficient Solution for SQL Injection Attack Detection and Prevention. International Journal of Soft Computing and Engineering (IJSCE), 2013.

[7] Anley, C., Advanced SQL injection in SQL server applications. White paper, Next Generation Security Software Ltd, 2002.

[8] Halfond, W.G.J., A. Orso, and P. Manolios, WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation. Software Engineering, IEEE Transactions on, 2008. 34(1): p. 65-81.

[9] Lori Mac Vittie, "SQL Injection Evasion Detection", White Paper-F5 Networks , September 2007 .

[10] Maor, Ofer, and Amichai Shulman. "SQL injection signatures evasion." Imperva, Inc., Apr (2004).