Classification of SQL Injection Attacks

Article · May 2010

CITATIONS
2

READS
2,270

1 author:

Khaleel Ahmad
Maulana Azad National Urdu University
71 PUBLICATIONS **43** CITATIONS

SEE PROFILE

Available ONLINE www.visualsoftindia.com/vsrd/vsrdindex.html

VSRD Technical & Non-Technical JOURNAL

A INTERNATIONAL JOURNAL

ISSN No. 0976-7967

VSRD-TNTJ, Vol. I (4), 2010, 235-242

RESEARCH COMMUNICATION

Classification of SQL Injection Attacks

[1]Khaleel Ahmad*, [2]Jayant Shekhar and [3]K.P. Yadav

ABSTRACT

Web-based applications constitute the worst threat of SQL injection that is SQL injection attack exploits the most web based applications. The attackers can trick the server to gain the authorization illegally and access the database by using SQL queries. This happens because the developers are not fully aware of attacks by SQL Injection and its causes. Many researchers have discussed about Sql Injection attack but no one can classify it. This paper is an attempt to categorize the SQL injection attack in order to comprehend the attacks more gently. This pattern permits different kind of vulnerabilities to lie under different classes of SQL injection attack that helps the developer or analyst to avoid the causes of occurrence of SQL injection attack.

Keywords: SQL Injection, SQL Injection Attack, Classification.

INTRODUCTION

SQL Injection is a method of exploiting the database of web application. It is done by injecting the SQL statements as an input string to gain an unauthorized access to a database. SQL injection is a serious vulnerability that leads to a high level of compromise - usually the ability to run any database query. It is a web-based application attack that connects to database back-ends and allows the bypass of firewall. The advantage of insecure code and bad input validation is clinched by the attacker to execute unauthorized SQL commands[4][5][6][7].

The objective of SQL Injection Attack (SQLIA) is to shaft the database system into running harmful code that can reveal confidential information. Hence we can say that Sql injection attack is an unauthorized access of database. In this paper we tried an attempt to classify the type of SQL Injection Attacks in order to enhance the security of database[4].

We build an attack repository, to evaluate the classification scheme, by collecting SQL injection attacks from white papers, technical reports, web advisories, hacker on-line communities, web sites, and mailing lists. This

[1]Senior Lecturer CSE/IT Department, Swami Vivekanand Subharti University, Meerut, UTTAR PRADESH, INDIA.
[2]Associate Professor, CSE/IT Department, Swami Vivekanand Subharti University, Meerut, UTTAR PRADESH, INDIA.
[3]Professor, CSE/IT Department, ACME College of Engineering, Ghaziabad, UTTAR PRADESH, INDIA.
*Correspondence : khaleelamna@yahoo.co.in

repository can help the developer to understand SQL injection attacks more thoroughly. It can also be used in the evaluation of defensive coding practices and intrusion detection systems.

CLASSIFICATION

Order Wise

First Order Injection Attack

First-order Injection 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 etc.

Second Order Injection Attack

Second Order injection Attack is the realization of malicious code injected into an application by an attacker, but not activated immediately by the application[1]. The malicious inputs are seeded by the attacker into a system or database. This is used to indirectly trigger an SQLIA which is used at later time. The attacker usually relies on where the input will be subsequently used and thus crafts his attack. Second order injection leaves a ticklish job of detection and prevention. This is because the point of injection is different from the point where the attack actually manifests it.

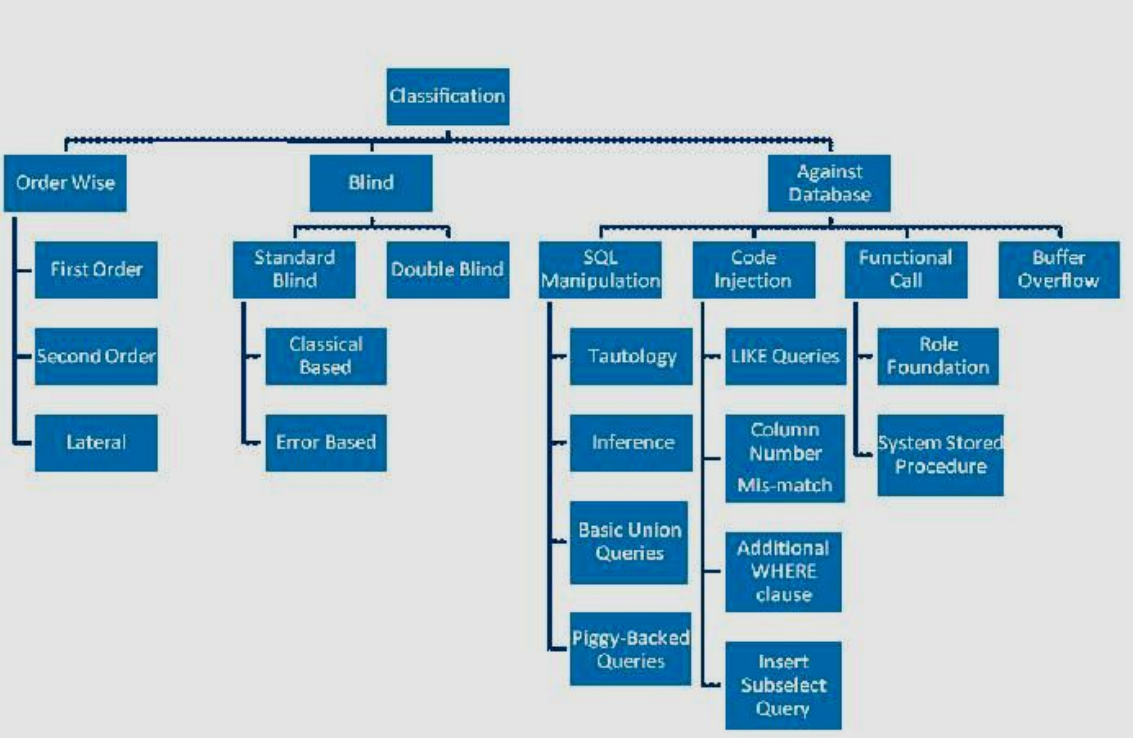

Fig. : Classification of SQL Injection Attacks

Lateral Injection Attack

This is a technique that is used to compromise Oracle databases remotely. The attack exploits some common data types, including DATE and NUMBER, which do not take any input from the user and so are not normally

considered to be exploitable. The database can be manipulated by an attacker just by using a bit of creative coding[2].

Classical SQL injection Attack

The classical exploitation of SQL injection vulnerabilities provides an opportunity to merge two SQL queries. This redeems the purpose to gain additional data from a certain table. To access the information from the Database Management System would become easier with the help of classical SQL injection[3].

Blind SQL Injection Attack

When web application is vulnerable to an SQL injection but the results of the injection are hidden to the attacker, blind SQLA is used. The displayed page may be different from the original one. This depends on the results of a logical statement injected into the legitimate SQL statement called for that page. The blind SQLIA allows the threat agent to infer the construct of the database through evaluating expressions that are coupled with statements that always evaluate to true and statements that always evaluate to false[5][14]. The Blind SQL injection can further be classified as:

Standard Blind

Sometimes it is impossible to exploit an SQL Injection vulnerability using classical technique that involves application of <union> operator and separator<";">. In this situation, Standard Blind SQLIA is implemented[3]. It allows one to write and read files and get data from tables provided only the entries are read symbol-by-symbol. This kind of attack can further be classified as:

Classical Based Blind SQL Injection

This attack is based on analysis of true/ false logical expression. If the expression is true, then the web application will return certain content, and if it is false, the application will return content[3].

Error-Based Blind SQL Injection

The error based blind SQL Injection is the quickest technique of SQL Injection exploitation. The august of this method is that the valuable information of various DBMSs can be stored into the error messages in case of receiving illegal SQL expression. This technique can be used if any error of SQL expression processing occurred in the DBMS is returned back by the vulnerable application[3].

Double Blind SQL Injection

Double blind SQL Injection is a technique in which all error messages and vulnerable queries are excluded from the page returned by the web application and the request results do not influence the returned page. Exploitation of this kind of vulnerability uses only time delays under SQL query processing; i.e., it is false if an SQL query is executed immediately, but if it is executed with an N-second delay, then it is true[3].

Against Database

On the basis of attack against the database management system, SQL injection attack can be classified as:

SQL Manipulation

The most common type of SQL Injection attack is SQL manipulation. The attacker attempts to modify the existing SQL statement[13]. This SQL manipulation attack can be classified as:

Tautology

The aim of a tautology-based attack is to inject code in one or more conditional statements such that the evaluation is always true. In this type of SQLIA, an attacker exploits an injectable field that is used in a query's WHERE conditional i.e. the queries always return results upon evaluation of a WHERE conditional parameter. All the rows in the database table targeted by the query are returned while transforming the conditional into a tautology. *Example*: In this example, an attacker submits " ' or 1=1 - -" for the *login* input field (the input submitted for the other fields is irrelevant). 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[6][7] [8][9][10].

Inference

In this attack, the query is being modified into the form of an action which is executed based on the answer to a true/-false question about data values in the database. In this type of injection, attacker try to attack a site that is enough secured not to provide acceptable feedback via database error messages when an injection has succeeded. The attacker must use a different method to obtain the response from the database since database error messages are unavailable to him. In this situation, the attacker injects commands into the site and then observes how the function/response of the website changes. By carefully observing the changing behavior of the site , attacker can extrapolate not only vulnerable parameters, but also additional information about the values in the database. Researchers have reported that with these techniques they have been able to achieve a data extraction rate of 1B/s .

Example: Consider two possible injections into the *login* field. The first being "legalUser' and 1=0 - -" and the second, "legalUser' and 1=1 - ".These injections result in the following two queries:

SELECT accounts FROM users WHERE login='legalUser'

and 1=0 -- ' AND pass='' AND pin=0

SELECT accounts FROM users WHERE login='legalUser'

and 1=1 -- ' AND pass='' AND pin=0

In the first scenario, the application is secure and the input for *login* is validated correctly. In this case, both injections would return login error messages, and the attacker would know that the *login* parameter is not vulnerable. In the second scenario, application is insecure and the *login* parameter is vulnerable to injection. The attacker submits the first injection and, gets a login error message, because it always evaluates to false. However, the attacker does not know if this is because the application validated the input correctly and blocked the attack attempt or because the attack itself caused the login error. The attacker then submits the second query, which always evaluates to true. If in this case there is no login error message, then the attacker knows that the attack went through and that the *login* parameter is vulnerable to injection[5][7][12].

Basic Union Queries

With this technique malicious user tricks the server to return data that were not intended to be returned by the developers. In this attack, an attacker exploits a vulnerable parameter to change the data set returned for a given query. Attackers do this by injecting a statement of the form: UNION SELECT <rest of injected query>.Since the attacker control the second/injected query completely, they can use it to retrieve information from a specified table. The result of this attack causes the database to returns dataset that is the union of the results of the original first query and the results of the injected second query[5][6][7][8][10].

Piggy-Backed Queries

In this SQLIA, attackers do not aim to modify the query instead; they try to include new and distinct queries into the original query. This result database to receive multiple SQL queries and can be proved extremely harmful. *Example*: If the attacker inputs "'; drop table users - -" into the *pass* field, the application generates the query:

SELECT accounts FROM users WHERE login='doe' AND

pass='''; drop table users -- ' AND pin=123

After execution of first query, the database would recognize the query delimiter (";") and proceed for the injected second query. The execution of second query would lead to drop table 'users', which would likely damage valuable information[5][6][7][8][10].

Code Injection

Code injection attacks attempt to add additional SQL statements or commands to the existing SQL statement. This type of attack is frequently used against Microsoft SQL Server applications, but seldom works with an Oracle database[13]. This attack can be classified as:

LIKE Queries

The attacker attempts to manipulate the SQL statement using like queries. Consequently, user input incorporated into a LIKE query parameter can subvert the query, complicate the LIKE match, and in many cases, prevent the use of indices, which slows a query substantially. With a few iterations, a compromised LIKE query could launch a Denial of Service attack by overloading the database[4][11].

Example: $sub = mysql_real_escape_string ("%something"); // still %something

mysql_query ("SELECT * FROM messages WHERE subject LIKE '{$sub}%'")

Column Number Mismatch

To accumulate the knowledge of this type of injection, consider a random example. Let the original query is:
SELECT product_name FROM all_products WHERE product_name like '&*Chairs*&'

Then the attack would be:

SELECT product_name FROM all_products WHERE product_name like '' *UNION ALL SELECT 9,9 FROM SysObjects WHERE '' = ''*

Above query would give errors that indicate that there is mismatch in the number of columns and their data type in the union of SysObjects table and the columns that are specified using 9. The error "Operand type mis-match" is mainly because the data type mis-match in the Union clause caused by the injected string. Another error we might see is "All queries in an SQL Statement containing a UNION operator must have an equal number of expressions in their target list" is because the number of columns is not matching.

After multiple trial and errors a statement like this may succeed.

SELECT product_name FROM all_products WHERE product_name like '' *UNION ALL SELECT 9,9, 9,' Text', 9 FROM SysObjects WHERE '' = ''*

Result set of the above query will show all the rows in the SysObjects table and will also show constant row values for each row in the SysObjects table defined in the query[4].

Additional WHERE clause

The case arises where there may be additional WHERE condition in the SL statement that gets added after the injected string.

SELECT firstName, LastName, Title from Employees

WHERE City = ' "& strCity &" ' AND Country = 'INDIA' "

On the first attempt after injecting the string the resulting query would look like:

SELECT firstName, LastName, Title from Employees WHERE City = 'NoSuchCity' UNION ALL Select OtherField from OtherTable WHERE 1 = 1 AND Country = 'USA'

Above query will result in a Error Message like this: Invalid column Name 'Country' because 'OtherTable' does not have a column called 'Country'. In case the backend is MS SQL Server, the problem can be solved using ";--" to get rid of the rest of the query string[4].

Insert – Subselect Query

The use of advanced insert query can help the attacker to access all the records of the databse.

Example: INSERT INTO tableName values (‘ “ ‘ + (SELECT TOP 1 Fieldname FROM tableName) + ’ ‘ , ‘xyz@xyz.com’ , ‘240402364’)**.**

Where ever this information displayed to the user typically places like pages where the users are allowed to edit user information. In the above attack the first value in the FieldName will be displayed in place of the user name. If TOP 1 is not used , there will be an error message “subselect returned too many rows”. Attacker can go through all the records using NOT in clause[4].

Functional Call Injection

Function call injection is the insertion of database functions into a vulnerable SQL statement. These function calls can be used to make operating system calls or manipulate data in the database[13]. This can be classified as:

Role Foundation

Press Releases are provided through their portal in many companies. Typically the user requests for a press release would look like:

http://www.somecompany.com/PressRelase.jsp?PressRealeaseID=5

The corresponding SQL statement used by the application would look like:

Select title, description, releaseDate, body from pressRelease WHERE pressRelaseID=5

All the information requested corresponding to the 5th press release is returned by the database server. This information is formatted by the application in an HTML page and provided to the user.

If injected string is 5 AND 1 = 1 and application still returns the same document then application is susceptible to SQL injection attack[4].

System Stored Procedures

In this type of SQLIAs the attacker tries to execute stored procedures present in the database. If the running backend server is known, the attacker perpetrates his attack to exploit the stored procedure. If he is able to inject SQL string successfully then stored procedures are no safer. The access to the system store procedures depends on the access privileges of the application user on the database. Most of the time, output may be absent on the screen as would in case of a normal SQL statement when a stored procedure is executed successfully[4][5][6].

Buffer Overflow

In several databases Buffer overflows have been identified. Some database functions are susceptible to buffer overflows that can be exploited through a SQL injection attack in an un-patched database. Most application and

web servers are unable to handle the loss of a database connection due to a buffer overflow. Usually, the web process hangs until the connection to the client is terminated, thus making this a very effective denial of service attack[13].

CONCLUSION

We have classified SQL Injection Attack that can be proved fruitful against the act of malicious intruders. This can help in fixing or atleast reducing the possibility of occurrence of vulnerability that can damage the database security. A detailed study of each attack is being done in order to categorize the SQL Injection Attack. With the help of these categories, industries expert and developers can bestow more security to databases of web applications.

REFERENCES

[1] Gunter Ollmann (2004): second – order code injection attacks, Advance code injection techniques and testing procedure. NGS Software Insight Security Research.

[2] David Litchfield: Lateral SQL injection: A new class of vulnerability in Oracle.

[3] Dmitry Evteev: Methods of Quick exploitation of blind SQL injection.

[4] Sagar Joshi (2005): SQL injection attack and defense: Web Application and SQL injection. http://www.securitydocs.com/library/3587

[5] William G.J. Halfond, Jeremy Viegas, and Alessandro Orso (2006): A Classification of SQL Injection Attacks and Countermeasures. IEEE Conference.

[6] San-Tsai Sun, Ting Han Wei, Stephen Liu, and Sheung Lau: Classification of SQL Injection Attacks. Electrical and Computer Engineering, University of British Columbia

[7] C. Anley (2002): Advanced SQL Injection in SQL Server Applications. White paper, Next Generation Security Software Ltd.

[8] S. McDonald (2002): SQL Injection: Modes of attack, defense, and why it matters. White paper, GovernmentSecurity.org.

[9] M. Howard and D. LeBlanc (2003): *Writing Secure Code*. Microsoft Press, Redmond, Washington, second edition.

[10] SQL Injection (2002). White paper, S. Labs. SPI Dynamics, Inc. http://www.spidynamics.com/assets/documents/WhitepaperSQLInjection.pdf.

[11] http://dev.mysql.com/tech-resources/articles/guide-to-php-security-ch3.pdf

[12] K. Spett (2003): Blind Sql injection. White paper, SPI Dynamics, Inc. http://www.spidynamics.com/whitepapers/Blind SQLInjection.pdf.

[13] Stephen Kost (2004): An introduction to SQL injection Attacks for oracle developers.

[14] Justin Clarke (2008): Using SQLBrute to brute force data from a blind SQL injection point. http://www.justinclarke.com/archives/2006/03/sqlbrute.html. Retrieved October 18, 2008.

❑❑