SQL Injection: Classification and Prevention

Aditya Rai
Student, B.Tech CSE
Lovely Professional University
Phagwara, India
aditya.rai2424@gmail.com

MD. Mazharul Islam Miraz
Student, B.Tech CSE
Lovely Professional University
Phagwara, India
mazharul.miraz@outlook.com

Deshbandhu Das
Student, B.Tech CSE
Lovely Professional University
Phagwara, India
deshbandhudas25@gmail.com

Harpreet Kaur
Assistant Professor
Lovely Professional University
Phagwara, India
harpreet.23521@lpu.co.in

Swati
Assistant Professor
Lovely Professional University
Phagwara, India
rampalswati@gmail.com

2021 2nd International Conference on Intelligent Engineering and Management (ICIEM) | 978-1-6654-1450-0/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICIEM51511.2021.9445347

***Abstract*— With the world moving towards digitalization, more applications and servers are online hosted on the internet, more number of vulnerabilities came out which directly affects an individual and an organization financially and in terms of reputation too. Out of those many vulnerabilities such as Injection, Deserialization, Cross site scripting and more. Injection stand top as the most critical vulnerability found in the web application. Injection itself is a broad vulnerability as it further consists of SQL Injection, Command injection, LDAP Injection, No-SQL Injection etc. In this paper we have reviewed SQL Injection, different types of SQL injection attacks, their causes and remediation to comprehend this attack.**

Keywords—Injection, SQLi, OWASP Top, SQL Prevention, Database attack.

I. INTRODUCTION

As the technology is advancing, many new things have been evolved and same goes with the dark side of digital world i.e. hacking. As the technology is evolving, many new attacks have emerged. SQL Injection (SQLi) is one of those vulnerability which was known around 1990 and is still dominating other vulnerabilities in terms of impact. This vulnerability was first mentioned by "rain.forest.puppy" in December issue of Phrack Magazine. It said that "A Microsoft SQL server is possibly showing sensitive data through the input given by a normal user."[8][16] Information like name, address, phone number etc. OWASP (Open Web Application Security Project), a non-profit foundation that focuses on web application security and lists out major vulnerabilities. In every few year OWASP updates their list of vulnerability (commonly known as OWASP Top 10). In their every update Injection attacks has been ranked as top most critical vulnerability which an individual, company or organization should prevent in their application. Therefore, it is highly important for an individual or any company to be aware of types and preventions of this attack. [11][13]

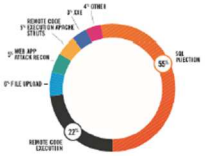


Figure-1

Alert Logic Cloud Security stated that "Four-Fifth of all the web application attacks we detected during 18 months period. SQL Injection alone was behind 55 percent of all observed attacks" as seen in Figure-1 [17]

II. SSQL INJECTION

A. What is SQL Injection SQL Injection is a vulnerability which targets the database. Any web application with database at its backend could be vulnerable to this SQL injection attack.[1][3][7] This vulnerability are more likely to be found in PHP (Hypertext Preprocessor) and ASP (Active Server Pages) applications. The main objective of attacker exploiting this vulnerability is to make database reveal its confidential information, which in-terms of web application can be username, password, secret keys etc. The basic or core reason behind this generally revolves around bad or poor coding practices. Other causes of SQL injection includes Invalid input, Generous privileges, Uncontrolled variable size, Error messages, Variable orphism, Dynamic SQL, Client side control, Multiple statements, Subselect, Into Outfile support etc.[7] Working of a web application now a days is as follow:

- User passes the request via web browser.
- Request contains parameters such as username, password etc. have to be sent database connected with the web application.
- Before sending request directly to Database, web application generates a dynamic SQL query in its backend and then sends to database.
- Database receives the query and executes it and send he response or output accordingly.

In Figure.2 we can understand the flow of requests from user to application via internet and then from server to database and then data being sent back to user via same route.

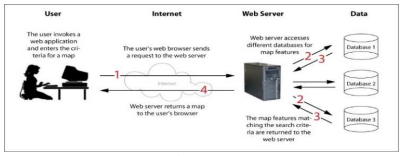


Figure-2

978-1-6654-1450-0/21/$31.00 ©2021 IEEE

For example, login functionality in a website asks user their user-id or email and their password to login for authentication. The request sent from the browser will contain two parameters possibly username and password. For instance let the values be: username = administrator and password = strongpassword. After parameters being sent to server, a SQL query is generated.

SELECT * FROM users WHERE
username='administrator' and
password='strongpassword' ;

When the query is sent to database, it is compared to the value from the data present in database. And on successful comparison, final result of query will be true and user will be logged in. When a SQL command or syntax can also be the part of parameter`s value sent to database. An unauthenticated user, with no knowledge of the password, can send a crafted input which can looks like this:

username='administrator' and password=aaa' OR '1'='1'

The respective SQL query will look like:

SELECT * FROM users WHERE username ='administrator' and password = 'aaa' OR '1' = '1' ;

Here if we focus on password part of query.

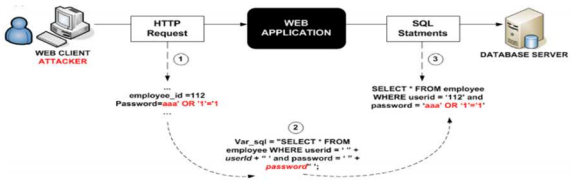
password = 'aaa' OR '1'= '1'

This statement will always result in TRUE because of OR Boolean operator and 1=1 is universal truth. Hence, one condition is true then whole logic will be true.

Input 1	Input 2	Output
True	True	True
True	False	True
False	True	True
False	False	False

OR Truth Table

And in this case unauthenticated user will be logged in but without having any password by tricking the SQL query and database. This , in general, is called SQL Injection.



Figure=3

B. *Impacts of SQL Injection*

This vulnerability affect the main aspects of security i.e. CIA triad i.e. Confidentiality, Integrity and Availability.[16]

- Confidentiality: SQL injection can be used to see the sensitive information which is stored inside the database such as usernames, passwords, secret keys and other information leading to potential loss of an individual and organization.
- Integrity: Using SQL injection data inside the database can be changed and altered using SQL 'UPDATE' and 'ALTER' query.
- Availability: Using SQL injection data from database can also be deleted using the 'DELETE' query.

C. *Detection of SQL Injection*

Detecting SQL vulnerability can be detected using some automated tools such as Burp Suite`s vulnerability scanner, Sqlmap, Sqlninja etc and manually it can also be detected using the following (but not limited to them) way or tests on the application.

1) Using a single quotation mark('), double quotation mark(") or backslash (/) inside input field and check for errors or changes.
2) Using different input rather than specified to check how the application responds when an unexpected type of input is received.
3) Using boolean statements such as OR, AND to look for different behavior of application.
4) Using specific payloads in input field to trigger time delays in application.
5) Using Out-of-band specific payloads to check for out-of-band network interaction.

D. *Types of SQL Injection*

SQL injection can be classified in many ways based on several factors. Generally, it can be classified in categories as mentioned below in the Figure.4.[16].

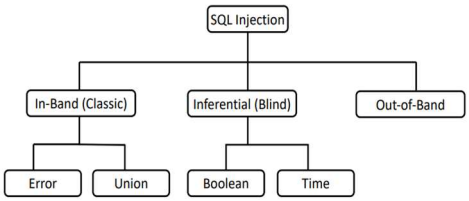


Figure 4{16}

1) *In-band SQL Injection:* This attack uses the same channel for launching as well as receiving the result of attack. For example, Web server can be used as same channel for sending the payload and receiving the output generated by that. This is the easiest of all other categories to exploit as output can be seen on the same channel.

   There are two subtypes of In-Band SQL specifically Error Based SQL as well as Union Based SQL.

   - *Error based SQL*
     In this type of SQL attack we use to send a unexpected input to generate some errors. And based on those error which have some info in it, we

will recreate our payload for getting towards successful payload.

- *Union based SQL*
  In UNION based SQL injection result of two queries are combined and displayed as single set of result. In this vulnerability two SQL query are sent to the data base via UNION command. [9] For example:

  www.randomwebsite.com/gallery.php?img=' UNION SELECT username, password FROM users--

2) *Inferential SQL Injection:* This attack have no actual data transfer via web application or database. There is not any output on the webpage in response of any request.. This type of attack generally takes more time to exploit. In such cases boolean type question or time based queries are useful which are another two subtypes of this attack.

   - *Boolean based Blind SQL:* When there is no output generated or showed by the database, we use true false condition to return a different result depending where the passed query returned true or false. The methodology here is to see how the webpage responses when a true or false query is executed. Inject a false payload and check the response then check the response of true payload. If application behaved differently in both cases then it is vulnerable to Boolean based SQL attack
     For example:
     True payload (at database side):
     SELECT id FROM user WHERE id=1 and 1=1;
     False payload:
     SELECT id FROM user WHERE id=1 and 1=2;
     If response of these two queries are different then the application is vulnerable.
     How it can be exploited: We can pass true false type queries and check the responses of each. Let say database have a user entry as
     Username Password Administrator password123
     Now we can ask the database about each character of password or username. Request:
     www.randomwebsite.com/gallery.php?id=1 and substring ((SELECT password FROM users WHERE username = 'Administrator'),1,1) = 'a'
     Here since the first character of password is 'p' and we checked if the first character is 'a' in our query. Database will response false. Looping through each character when we send the query like:
     www.randomwebsite.com/gallery.php?id=1 and substring ((SELECT password FROM users WHERE username = 'Administrator'),1,1) = 'p'

     This will response as true payload as the first character of our password is 'p' which we asked in our query. Same way we can loop through each index of password and at the end we will get our whole password which can lead to complete access of Administrator account. This can also be done easily by using automated tools like SQLmap or SQLninja.

   - *Time based SQL:* This attack depends on database pausing for a given amount of time, then returning the results indicating a successful SQL query execution. For example: If the first character of Administrator account`s password is 'p' then wait for 10 seconds. This can be done using SLEEP query. If the requested query is true the database will respond after 10 seconds or whatever is specified in query.

   - *Out of Band (OOB):* This vulnerability exists when there is no channel or medium for database to respond to web server. This consist of triggering out of band network connection to a system that we control. Thus, we force the database to send the response to that specific server we control. Using Out-of-band specific payloads to check for out-of-band network interaction.

     SQL Injection can be classified more exhaustively.[1][12]

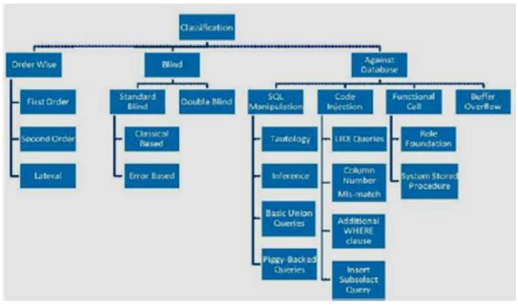


Figure 5[1]

1. *Order wise*
   - *First order Injection*: In this type of attack, response is received immediately. Either through web page or through some other mechanism.
   - *Second Order Injection*: In this type of attack, malicious code is injected into application but not exploited immediately or executed by application. This is used to trigger SQL attack later indirectly. This attack is generally tough to detect or prevent as the system are not exploited immediately.
   - *Lateral Injection Attack*: In this attack, Oracle databases are targeted. Common data types of Oracle such as DATE, NUMBER etc acts as attack vector. They do not accept any user input but using these database can be manipulated by attacker with some tricky coding. [10]

2. *Blind SQL Injection:* As discussed earlier, the application can be vulnerable to SQL Injection but the response from database might be hidden. Blind SQL allows user to generate query based on True False basis. Database responds differently to both True statement and False statement.

   - *Standard Blind*: It is based on the analysis of Boolean statements basically Yes/No based questions/queries. If the requested query is true, webpage will return different data as compared to false query.
   - Double Blind: In this, errors are prevented to be displayed on webpage as well as vulnerable SQL queries are also excluded. In such cases, exploitation depends on time (Time based Attack). If the request query is true database will show a delay of specific seconds or if its false it will show result immediately

3. *Against Database:* Among this, sub categorization is done as follows- SQL Manipulation, Code Injection, Functional Call. Buffer Overflow.

   1. *SQL Manipulation*: This is the most common type of attack in which attacker attempts to change or alter SQL queries.
   - *Tautology*: This attack aims to insert multiple conditional statement in the query trying to make it universally true. Generally, WHERE function of SQL is exploited by adding the conditional statement such as AND , OR etc.[2][3][4][12]

     For example: In a poorly configured Login page, an attacker can add " ' or 1=1 -- " in the username field. This results in query as:

     SELECT * FROM users WHERE username='' or 1=1 – AND password= '';

     This results in database ignoring the password field as (--) acts as comment in SQL. Whereas (or 1=1) is universally true, which leads in successful login or bypass login.
   - *Inference* : In this type of attack, database is not providing any error message by which we can modify our query or get any sensitive information. Therefore, we use to ask questions to database in terms of true false regarding the data values inside the table of database.[12]
   - *Basic Union Queries* : In this type of attack, two queries are sent using the UNION query of SQL. This combines the two query executes them and output the result as single set which consists data of first query and the data of next query which was injected.[2][3] [12].
     For example:
     SELECT product FROM products WHERE id=1 UNION SELECT username, password FROM users;

     Here, injected query will show usernames and passwords from users table into the webpage as single set result.
   - Piggy Bagged Queries: In this type of attack, instead of fetching any data from the database, attacker try to add new and different queries into the original one. Now instead of passing queries to bypass or fetch data attacker have more command over the database. New tables can be created, old tables can be destroyed and much more which can be very harmful.[2][3]
     For Example:
     SELECT accounts FROM users WHERE id= 'admin' AND pass= ''; DROP TABLE users –'

     Here (;) is delimiter which tells SQL that a command has end and post that is a new command.

   2. *Code Injection:* This attack tries to add additional SQL queries or commands at the end of SQL statements in any application. It can also be further classified into subcategories.

      1. *Like Queries:* It is a bit similar to pattern matching. The user input incorporated into LIKE query can sabotage the whole query and can do much more out of which is slowing down the response time of the query. When these LIKE queries used all along multiple times can work as Denial Of Service attack by overloading the Database.
      2. *Column Number Mismatch:* When UNION query is used to fetch data via additional query we might get error as either Column number do not match or their datatypes do not match. In such cases it tells us exact number of columns present in a table.
         SELECT product FROM all products WHERE name LIKE ''UNION ALL SELECT 9,9,9' Text', 9 FROM other table WHERE '' = '';

   3. *Additional WHERE Clause:* This attack can be seen multiple time in other attacks also. Where we can add additional condition to SQL

statement that gets appended after the original query.

4. *Insert Subselect Query:* There is a function found in SQL, INSERT, which is used to insert data into the database. Its advanced usage can lead an attacker to access all the database.
For example:
INSERT INTO users values (‘ “ ‘ +(SELECT password FROM users) + ’ ‘, ‘randomwebsite@randomwebsite.com’, ‘password’)

3. *Functional Call Injection:* It is the injection of database functions into a SQL statement which can be used to manipulate or access data in the database. In this there are two subcategories namely Role foundation and System stored procedure.

1. *Role Foundation:* When we access any data from the webpage the request would be like:
www.randomwebsite.com/images.php?id=2

Now on adding ‘AND 1=1’ in the URL so that it would look like this:
www.randomwebsite.com/images.php? id=2 and 1=1

Now if the same webpage is received then it is vulnerable to SQL attack.

2. *System Stored Procedures:* This attack involves attacker to use and execute stored procedure which is already present in the database. The only condition over here is that the running database should be known whether it is Oracle, MSSQL, MYSQL etc [3][10] [12]

3. *Buffer Overflow:* Some databases are sometimes vulnerable to buffer overflow attacks. Such database is ORACLE in which buffer overflow was found. Usually, the connection with the database is lost or the database stops working properly which lead to termination of client connections which makes this another Denial of service vector.[10]

E. Manual Command

Some common commands used by attack while exploiting SQL vulnerability to get sensitive information or harm the system. [6]

a. *Login Injection Command*
Logging in as any user-
SELECT * FROM user WHERE name=‘--------’ AND password=‘--------’;
Payload : username= ‘ OR ‘x’=‘x’—
Logging in as specific user-
Payload: username= ‘x’ OR name LIKE ‘%admin %’ –

b. *Insert Injection Command*
INSERT INTO users (name,password) VALUES (‘attacker’, ‘password123’);
This will add a new entry in the database.

c. *Update Injection Command*
UPDATE user SET name= ‘attacker’ WHERE email = ‘admin@admin.com’;
This will update name of entry having email as admin@admin.com

d. *Delete Injection Command*
DELETE FROM user WHERE username= ‘admin’;
This will delete admin user entry from the database.

e. *Drop Command*
DROP table user;
This will remove user table from the database

F. Automated Exploitation

Even-though above mentioned attack techniques or categories may seem to be a tidious task in its own but due to some open source available tools it becomes easy to exploit these vulnerabilities. Tools like SQLmap and SQLninja are magnificent when it comes to automated SQL exploitation. Out of which SQLMap is an impressive tool which contains many payloads for exploiting SQL vulnerabilities. It can exploit Boolean based, time based, error based, UNION based etc. There can be very rare chance that an SQL vulnerability is not exploited even after tweaking with it or such tools.[5][14]

III. COUNTER MEASURES

Even though there are so many attack types in SQL. Luckily, defence is comparatively simple as most of the solution involves better handling of user provided data. Some techniques can have a bit of drawback but can be proved fruitful for defensing. Below are mentioned some defensing techniques against SQL Attacks.

1. *Whitelisting/Blacklisting*: Whitelist refers to allowing only those word/character in input field which are predefined by the developer. For example username generally do not have special characters so restricting special characters in username field. It won’t be accepted. Any other entry or input will be rejected. Reverse of this is Blacklisting, which refers to not allow those characters or words which are defined in blacklist by developer. Any input which is mentioned in blacklist will automatically be removed or will generate error. SQL syntax such as UNION, AND, OR etc can be put into blacklist which will not be used by attacker to access data. Whitelisting is more effective than blacklist as an attacker an use complicated encoding schemes that may not be present in blacklist.[4][5]

2. *Prepared Statement/Parameterized Query*: This technique was originally made for efficient usage of SQL but it also provided better security mechanism. it consist of predefined structure of SQL syntax which have placeholders in it. Parameters which are given by user to application are not directly sent to database. They are first placed in the predefined query and if the query

generates no error then the SQL query is sent to database for execution. This prevents execution of injected SQL query directly in database [5]. For example: SELECT * FROM phonebook WHERE username = ? AND password = ? Where (?) is placeholder? This query will be completed by server and then sent to database. [3]

3. *Stored Procedure*: They are very similar to parameterized query but with stored procedures, the SQL code is kept within SQL database.[5]
4. *Defensive Coding Practice*: This consist many techniques such as forbid the use of meta-characters, identification of all input fields. [5][7][12]
5. *Taint based approach*: This approach requires alteration of PHP interpreter to track taint information of the input. Later, it should intelligently detect and reject queries if any malicious code in injected. [2][5]
6. *Proxy Filters*: This refers to proxy filtering system that uses input validation rules on the data that is sent to application. The only drawback of this is it requires human interaction and developer should have an idea of what data should be removed by pattern and filter.[5][12]
7. *mysql_real_escape_string()*: This is a function already present in SQL used to escape special characters in a string like \x00, \r, \n, \, ‘, “, and \x1a. [4][6][9]
8. *Instruction Set Randomization*: SQLrand is the technique which allows developers to create randomized query instead of normal SQL query. Which will lead to syntax error whenever attacker inject the malicious input.
9. *Low Privileges*: Not everyone should be allowed to execute SELECT, INSERT, UPDATE, DELETE, DROP etc. Therefore, database subject should have less or minimum privileges.[5]
10. *Output Escaping*: Since server side of application mostly interacts with the frontend of website i.e. webpage, there some functions can be used to handle user data and convert some pre-defined characters (malicious). One such function in PHP (server side language) is htmlspecialchars (). [9]

Other prevention techniques that can make attacks less successful. For example keeping table names complicated making it difficult for attacker to guess it. There are a few drawbacks of Defensive coding such as it is difficult to implement, it only prevents a subset of attack types, increases cost and complexity of altering or writing new code [2][7]. Static analysis, which is another method of detection, can sometimes generate high false positives. Some vulnerability can exist at run time which cannot be detected statically. Moreover, static analysis is time consuming. [2]

IV. CONCLUSION

We have seen that how SQL injection can be used by an attacker to access sensitive information from the database by using different techniques as discussed above. Being the Top most critical vulnerability for a long time it is an important thing to keep note of while creating web applications to keep check of every possible way to prevent this attack. Also, the detailed categorization security researchers and developers can put more focus on database security. Moreover, one should try to minimize taking user input in their web application, as taking user input creates risk of SQL Injection. If user input has to be taken, then the application should be well tested by using the different types of attack mentioned, manually and by using automated tools. Different test cases should be used to test application regressively. If any vulnerability is found then it can be fixed using the above mentioned counter measures. Different counter measure works for different types of SQL Attack.

REFERENCES

[1] Ahmad, Khaleel. (2010). Classification of SQL Injection Attacks. VSRD Technical & Non-Technical Journal. 1. 236-242.

[2] Alattar, Munqath & Medhane,. (2013). Efficient Solution for SQL Injection Attack Detection and Prevention. International Journal of Soft Computing and Engineering (IJSCE). 3. 2231-2307.

[3] Manmadhan, Sruthy & T, Manesh. (2012). A Method of Detecting Sql Injection Attack to Secure Web Applications. International Journal of Distributed and Parallel systems. 3. 1-8. 10.5121/ijdps.2012.3601.

[4] Gopal, Girdhar. (2016). CASE STUDY OF SQL INJECTION ATTACKS. 10.5281/zenodo.56935.

[5] Mavromoustakos, Stephanos & Patel, Aakash & Chaudhary, Kinjal & Chokshi, Parth & Patel, Shaili. (2016). Causes and Prevention of SQL Injection Attacks in Web Applications. 55-59. 10.1145/3026724 .3026742.

[6] Kolhe, Abhay & Adhikari, Pratik. (2014). Injection, Detection, Prevention of SQL Injection Attacks. International Journal of Computer Applications. 87. 10.5120/15224-3739.

[7] Tajpour, Atefeh & Ibrahim, Suhaimi & Sharifi, Mohammad. (2012). Web Application Security by SQL Injection DetectionTools. International Journal of Computer Science Issues. 9.

[8] Hyslip, Thomas. (2017). SQL Injection: The Longest Running Sequel in Programming History. The Journal of Digital Forensics, Security and Law. 10.15394/jdfsl.2017.1475.

[9] Voitovych, Olesya & Yuvkovetskyi, O. & Kupershtein, Leonid. (2016). SQL injection prevention system. 1-4. 10.1109/UkrMiCo.2016 .7739642.

[10] Stephen Kost (2004): An introduction to SQL injection Attacks for Oracle developers

[11] “Top 10 Web Application Security Risks”- https://owasp.org/www-project-top-ten/

[12] Halfond, William & Viegas, Jeremy & Orso, Alessandro. (2006). A Classification of SQL Injection Attacks and Countermeasures.

[13] Alghamdi, A., Ahmad, B., & Imran, M. (November, 2015). SQL injection attack, still an unaddressed issue with dynamic web applications. International Journal of Computer Science Engineering, 4(6)

[14] Gunawan, Teddy & Lim, Muhammad & Kartiwi, Mira & Malik, Noreha & Ismail, Nanang. (2018). Penetration testing using Kali linux: SQL injection, XSS, wordpres, and WPA2 attacks. Indonesian Journal of Electrical Engineering and Computer Science. 12. 729-737. 10.11591/ijeecs.v12.i2.pp729-737.

[15] https://github.com/rkhal101/Web-Security-Academy-Series/tree/main/sql-injection/theory

[16] http://phrack.org/issues/54/8.html

[17] https://www.alertlogic.com/blog/tried-and-true-sql-injection-still-a-leading-method-of-cyber-attack-d58/