A Survey on: SQL Injection assaults in Web Application

[1]Sabari Giri Murugan S,[2]Naman Rao,[3]Quraish Abbas Nagri
[1]Asst.Professor,[2]Student,[3]Student
[1]Department of CS & IT,
[1]Jain (Deemed-to-be University), Bangalore, India

Abstract: The world in recent times has improved normally within the discipline of innovation prompting further upgrade of the current software programs. There is big interest for Web Applications for each transportable and work location's a right away result of the expansion in clients of net. The essential targets are the Net Packages and the Web Servers as your login accreditations can be abused with the aid of the assailant making use of SQL Injection assault as they're the ongoing well-known risks even with the utilization of IDS and IPS. This paper speaks about the guideline notion at the back of a SQL attack, the vicinity techniques and ultimately the preventive measures.

Index Terms **- Web utility, Web Server, IDS, IPS.**

I. INTRODUCTION

Internet plays a very important role in our day-to-day life. Web applications are used by organizations to provide services like online banking, online shopping, social networking, etc. With the increasing usage of internet cyber crime is also increasing. Hence, security has become a major concern for the organizations. Web applications receive the requests from the user, interact with the backend database and return the relevant information to the users. The backend database contains the confidential and sensitive data that interests the attackers. [1]

Due to the more digitalization of world, usage of mobile phones, computers, tablets etc. is increasing very fast. With boosting of digitalization and internet the usage of web applications has also increased. Many of the web application has a three- tier construction i.e. Presentation tier, CGI tier, and Database tier. SQL injection attack is also known as SQL insertion attack. Due to advances in internet, most offline services have moved online. These online services use web applications and web services. Most web attacks target the vulnerabilities of web applications. The SQL Injection Attack (SQLIA) does not waste system resources as other attacks do. However, because of its ability to obtain/insert information from/to databases, it is a strong threat to servers like military or banking systems. The web application framework uses filtering methods for data inputted by user. By the development of the information technology, a massive amount of sensitive information is stored in the database. This information is most valuable for the organizations. Database intrusion attacks can occur for stealing the valuable and sensitive information. SQL injection is the most widespread security issue in the web applications. Code injection attacks consists of SQL injection attacks, in which SQL characters are inserted into the SQL statements using an un trusted access to change the logic or meaning of the intended query. When the SQL statements is constructed using the external input data, the threat of SQL injection is found. The attacker could able modify the query Statements by modifying or altering the input data. The SQL injection attack occurs due to lack of development time and training, lack of experience and knowledge of potential security issues, developers often misuse these methods which results in SQL injection vulnerabilities (SQLIVs).[2]

SQL injection attack could be approach through which attackers increase contact done back to end databases by adding malicious codes through front-end.SQL is Structured Query Language that is a computer language for supply maneuver & retrieving information stored in a relational database. The Relational Database Management Systems like My SQL, Sybase, Informix & MS Access, Oracle, SQL Server use SQL is a database language.

The SQL Injection weakness of security, it is providing right location in that and hacker could use it to bypass web applications verification & support mechanisms & take rear contents of entire database. The SQL Injection is work to include, adjust & erase information in main database, disturbing data veracity & extents like this, SQL Injection is also provides an attacker.[3]

SQL injection was an attack in which malicious code was embedded in strings that were later passed to database back end for parsing and execution. The malicious data produced database query results and acquired sensitive information, such as account credentials or internal business data. Through analyzing the principle of SQL injection attacks, prevention method was proposed to solve the double defense through the browser and server ends. [4]

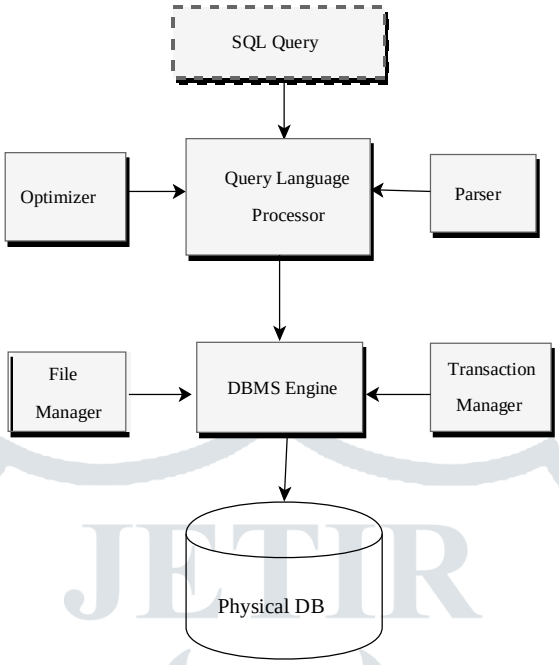

Fig 1. SQL Database

II. Types of SQL Injection Attacks

To counter this problem firstly we need to know the ways through which it can be implemented. SQL injections can be implemented in the following ways:

i) Tautology
ii) Illegal/logically incorrect queries
iii) End of line comment
iv) Timing attack
v) Union queries
vi) Blind SQL injection attacks

i) **Tautology**: This technique injects statements that are always true so that the queries always return results upon evaluation of WHERE condition.
Injected query: select name from user_details where username=’naman’ and password=’’or ‘1’=’1’.
ii) **Illegal/Logically incorrect queries**: This method is used by the threat agent to collect information about the database. Attacker intentionally injects SQL tokens or junk input in query to produce logical errors, type mismatches or syntax errors by purpose.
iii) **End of line comment**: In this technique the values are entered in the input field in such a way that rest of the query is treated as a comment.
iv) **Timing attack**: In this type of attack, the attacker guesses the information character by character, depending on the output form of true/false. In time based attacks, attacker introduces a delay by injecting an additional SLEEP (n) call into the query and then observing if the webpage was actually by n seconds.
v) **Union Query:** Injected query is concatenated with the original SQL query using the keyword UNION in order to get information related to other tables from the application.
vi) **Blind SQL injection attacks:** Attackers typically test for SQL injection vulnerabilities by sending the input that would cause the server to generate an invalid SQL query.
If the server then returns an error message to the client, the attacker will attempt to reverse-engineer portions of the original SQL query using information gained from these error messages. The typical administrative safeguard is simply to prohibit the display of database server error messages. Unfortunately, that’s not sufficient. If your application does not return error messages, it may still be susceptible to “blind” SQL injection attacks.

III. Related Work

There are lots of researches in area of SQL injection attack. Study of several researchers has been made. These studies discuss attacking mechanism along with solution of problem with their pros and cons. Following is the list of method and approaches used by existing researchers:

Table 1. Comparisons of previous approaches

Method Name	Advantages	Drawbacks
Novel based algorithm to prevent back end database[1]	Proposed model uses the concept of ASCII values with three phases that storing, checking and retrieving form database	Storage is an major concern because need double space for storing an ASCII values
Defeating SQL Injection-Removing Parameter values [2]	Proposed Model detects the injection attacks by using a combine static and dynamic analysis.	Runtime prevention approach requires dynamic monitoring systems but it could prevent all attacks.
Encountering SQL injection in Web application[3]	Using Anomaly based prediction query will be validated and if anamaly value reaches an threshold it will be rejected.	Validation of SQL query interms of case sensitivity it affect major concern of authenticity for an authorized users
SQL injection attack defense model[4]	The proposed defense model it tries to detect from network level legitimacy , Query length and checks the privilege	Mismatch interpretation from the SQL server it will not allow authorized user to access to that.
A novel method for SQL injection attack based on removing SQL query attribute values [5]	Proposed method can be used for modularization of detection programs.	It is independent of the DBMS when compared with other SQLIA detection methods.
SQL Injection: A demonstration and implications for accounting students[6]	Identifying and understanding the risks of SQL injection and its impacts on financial processes.	Identifying and understanding of risk related to SQL query it's not so easy to detect it.
Analysis of field data on web security vulnerabilities[7]	Application written in high level language have an less vulnerabilities and exploits	Vulnerabilities are not present in the source data analysis.

IV. SQL Injection Detection

Detect potential SQL injection vulnerabilities

The first step towards achieving a successful SQL injection attack is to detect vulnerabilities. Of course, some tools can automate the process, but it's better to understand how detection can be done manually. In addition, there are some situations where only manual testing will allow in-depth analysis.

Interfering with Query

The only way to know if an input source is potentially vulnerable is to trigger anomalies in the tested webpage or application. In order to do this, the attacker must submit input values that are likely to be incorrectly handled. This technique fuzzing will not yet confirm that an SQL injection flaw is present, but it will help finding what needs further testing.

Testing Strings

To create anomalies in the tested script you need to submit input values prone to generate an invalid SQL syntax. Some key testing strings will allow you to achieve this. Those include special characters that should have been filtered by the application. Here are a few examples from the SQL injection testing strings article.

TEST STRINGS TO DETECT SQL INJECTION (1 PER LINE).
xyz'
xyz')
1 OR
1)

Detection Example

Before jumping in the core of the subject let's take a look at a simple example.

THE QUERY IS BUILT WITHOUT SANITIZING PARAMETERS.
$query = "SELECT title, content FROM posts WHERE page='".$_GET['page']."'";
TEST STRING SUBMITTED BY THE ATTACKER (VALUE OF PAGE PARAMETER).
xyz')
QUERY GENERATED. NOTICE THAT THE INJECTED QUOTE CLOSES THE STRING (COLOR CAN BE CONFUSING HERE)!
SELECT title, content FROM posts WHERE page='xyz')'

As expected, the page returned an error.

ERROR RETURNED AFTER INJECTION ATTEMPT.
1064 - 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 ')" at line 1.

In this example it is pretty obvious that there is a security flaw. However, in some cases it is much more difficult to tell. When generic errors or HTTP errors are returned, it becomes hard to know if the suspicious data was intercepted by server-side validation or if the code handled an error generated by the query. For this reason, it is necessary to push tests a bit further.

Confirm Detection

Inference testing is a good technique to confirm that a potential flaw is really vulnerable. Simply put, testing SQL injection by inference will allow the attacker to reconstruct information by analysing the application's response to different requests. In this case, the tester will craft special SQL segments to make sure he really has control over the query. Two different tests will be required.

V. SQL Prevention Methods

Parameterized Statements

Programming languages talk to SQL databases using **database drivers.** A driver allows an application to construct and run SQL statements against a database, extracting and manipulating data as needed. **Parameterized statements** make sure that the parameters (i.e. inputs) passed into SQL statements are treated in a safe manner.

```
String sql = "SELECT * FROM users WHERE email = ?";
// Run the query, passing the 'email' parameter value...
ResultSet results = stmt.executeQuery(sql, email);
while (results.next()) {
// ...do something with the data returned.}
```

The code here portrays that the utilization of a particular parameter empowers in protecting the code and the product from the assault inside.

Contrast this to explicit construction of the SQL string, which is **very dangerous**

```
// Bad, bad news! Don't construct the query with string concatenation.
String sql = "SELECT * FROM users WHERE email = '" + email + "'";
// I have a bad feeling about this...
ResultSet results = stmt.executeQuery(sql);
while(results.next()) {
// ...oh look, we got hacked.
}
```

This code has the real imperfection of utilizing the string link that causes the assailant to utilize the holes in utilizing the strings to transfer the code into the framework.

The key difference is the data being passed to the execute Query (...) method.

In the first case, the parameterized string and the parameters are passed to the database separately, which allows the driver to correctly interpret them. In the second case, the full SQL statement is constructed before the driver is invoked, meaning we are vulnerable to maliciously crafted parameters.

NOTE: Use parameterized statements where available, they are your number one protection against SQL injection.

Object Relational Mapping

Many development teams prefer to use **Object Relational Mapping (ORM)** frameworks to make the translation of SQL result sets into code objects more seamless. ORM tools often mean developers will rarely have to write SQL statements in their code – and these tools thankfully use parameterized statements under the hood.

```
Def current_user(email)
  User.find_by_email(email)
end
```

Code like this is protected from SQL Injection assaults as they utilize the in-manufactured directions rather than the essential directions that are powerless against the assault.

Using an ORM does not automatically make you immune to SQL injection, however. Many ORM frameworks allow you to construct SQL statements, or fragments of SQL statements, when more complex operations need to be performed on the database. For example, the following Ruby code *is* vulnerable to injection attacks:

```
Def current_user(email)
User. where ("email = '" + email + "'")
end
```

Escaping Inputs

If you are unable to use parameterized statements or a library that writes SQL for you, the next best approach is to ensure proper escaping of special string characters in input parameters.

Injection attacks often rely on the attacker being able to craft an input that will prematurely close the argument string in which they appear in the SQL statement. (This is why you will often see ' or " characters in attempted SQL injection attacks.)

Programming languages have standard ways to describe strings containing quotes within them – SQL is no different in this respect. Typically, doubling up the quote character – replacing ' with " – means **"**treat this quote as part of the string, not the end of the string**".**

Escaping symbol characters is a simple way to protect against most SQL injection attacks, and many languages have standard functions to achieve this. There are a couple of drawbacks to this approach, however:

- You need to be very careful to escape characters everywhere in your codebase where an SQL statement is constructed.
- Not all injection attacks rely on abuse of quote characters.

For example, when an numeric ID is expected in a SQL statement, quote characters are not required. The following code is still vulnerable to injection attacks, no matter how much you play around with quote characters

```
defcurrent_user(id)
  User.where("id = " + id)
end
```

How much ever you attempt to alter the code and attempt to adjust the substance the vulnerabilities in the essential codes can't be changed.

Sanitizing Inputs

Sanitizing inputs is a good practice for all applications. In our example hack, the user supplied a password as ' or 1=1--, which looks pretty suspicious as a password choice.

Developers should always make an effort to reject inputs that look suspicious out of hand, while taking care not to accidentally punish legitimate users. For instance, your application may clean parameters supplied in GET and POST requests in the following ways:

- Check that supplied fields like email addresses match a regular expression.
- Ensure that numeric or alphanumeric fields do not contain symbol characters.
- Reject (or strip) out whitespace and new line characters where they are not appropriate.

VI. Proposed Model

This paper focused on how SQL injection attack possesses serious threats to web users and web application services majorly caused by input validation and similar other reasons. Rather than utilizing a similar old systems like the code based, concrete assault and the tainted based vulnerability, we can ensure that the rudiments are being securely utilizing the inbuilt keyword rather than the essential and basic ones, which make it much secure and will help in diminishing the affected codes that are internally code based and to prevent them from spreading to further influence the other active and live codes. Therefore we can utilize the counteractive action procedures referenced in the preventive measures to further protect the codes from assaults of a similar kind. This paper also highlights introduction to various types of SQL injection attacks their detection and prevention techniques.

VII. Conclusion

SQL injection attack is a serious security and integrity problem in Web application and all those actions performed by user with the help of web. Most of these attacks are successful due to lack of valid input, rules, policies and standards by the user. The attackers make good use of these vulnerabilities and carry out these attacks. Hence the prevention and knowledge about these attacks is very important in order to keep data and databases safe. In addition to SQL injection attacks, we discussed about various detection and prevention techniques in this paper.

References

[1] Mahima Srivastava,"Algorithm to prevent back end database against SQL injection attacks,"IEEE International conference on computing for sustainable global development, 2014, pp.754-757.

[2] Rajashree.katole, Dr.Swati S.Shrekar, Dr.Vilas M. Thakare,"Detection of SQL injection attacks by removing the parameter values of SQL Query."IEEE proceedings of the conference on Inventive systems and Control,2018,pp.736-741

[3] Joshi padma N, Dr.N.Ravishankar, Dr.M.B. Raju and N.CH.Ravi,"Encountering SQL injection in Web applications." IEEE proceedings of the conference on computing methodologies and communication,2018,pp.251-261

[4] Li Qian,Zhenyuan Zhu,Jun Hu,"Resaerch of SQL injection attack and prevention technology."IEEE proceedings of the conference in detection and information fusion,2015,pp.303-307

[5] Indrani B., E. Ramaraj (2012) "An Efficient Technique for Detection & Prevention of SQL Injection Attack using ASCII Based String Matching" International Conference on Communication Technology & System Design

[6] P. Sonam, "Protection of Web Application against Sql Injection Attacks", International Journal of Modern Engineering Research Vol.3, Issue.1, Jan-Feb. 2013 pp-166-168

[7] Lwin Khin Shar & Hee Beng Kuan (2013)Tan Mining SQL Injection & Cross Site Scripting Vulnerabilities using Hybrid Program Analysis

[8] S. Pankaj Sharma, "Integrated approach to prevent SQL injection attack & reflected cross site scripting attack," International Journal on Recent & Innovation Trends in Computing & Communication Volume: 1 Issue: 4,2014

[9] Amirmohammad Sadeghian 2014 SQL Injection Vulnerability General Patch Using Header Sanitization

[10] Nabeel Salih Ali Protection Web Applications using Real-Time Technique to Detect Structured Query Language Injection Attacks 2016

[11] Kanchan Choudhary, Anuj Kumar Singh (2016) "A Modified Scheme for Preventing web Application against SQL Injection Attack", International Journal of Computer Applications (0975 – 8887) Volume 141 – No.10, May 2016