ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 5, May 2016

Liferay: A Development Option for Preventing SQL-Injection

Rohan M. Bhagwat[1], Gauri M. Jodh[2], Sangeeta S. Adake[3], Rohini B. Bodawar[4], Reena Kharat[5]

B.E. Student, Department of Computer Engineering, PCCOE Nigdi, Pune, Maharashtra, India[1,2,3,4]

Professor, Department of Computer Engineering, PCCOE Nigdi, Pune, Maharashtra, India[5]

ABSTRACT: Web Applications are used by everyone in day to day life and we use database to store information. Relational database systems like MySQL, Oracle which are used by many web applications use the structured query language (SQL) for accessing the database and to perform operations on them. SQL is the most powerful and popular language and hence seeked the attention of many hackers. This made the SQL injection attack a well-known attack and took on the highest rank in security vulnerabilities. SQL injection attack is a popular attack in which a code injection technique is used to attack data-driven applications, in which malicious SQL queries are inserted into legitimate SQL query through a non-validated input from a user. (e.g. to dump the database contents to the attacker). With the successful execution of this query it leads to the SQL injection attack which reveals the confidential and sensitive data to the attacker. Thus the biggest challenge for any web application is protection of its database from the security vulnerabilities majorly THE SQL INJECTON ATTACK. To overcome this problem, LIFERAY, a new web development option can be used. Liferay is the finest solution for the SQL injection attack. Liferay is web portal development technology. Using Liferay technology user can develop websites and web portal with less efforts and more security. Liferay provides the secure login page. We do not write any code for database connectivity. At runtime SQL query is formed and get executed. Liferay provides the solution to the SQL injection attack. Liferay provide the secure coding which is the best practice to avoid the SQL injection attack.

KEYWORDS: SQL injection; Web application vulnerability; Information security; Liferay.

I. INTRODUCTION

Databases are the main accumulators of the confidential and sensitive data and hence have become the center of attraction for many hackers .SQL injection attack is one of the attacks that attackers use to snoop into the databases. Among all the security vulnerabilities, SQL injection is the most frequently used attack. As per the survey of Open Web Application Security Project (OWASP) SQL injection has 1st ranked attack among the top 10 list of web application attacks [1]. This paper provides information about the various possibilities of SQL injection attacks and how LIFERAY is used to prevent them.

SQL injection is a type of vulnerability found in security in which the attacker adds Structured Query Language (SQL) statements through non sanitized user parameters into the original SQL query and sends them to database server [2]. For example: On a Web form, for user authentication, when the users enter their name and password into the text boxes provided for them, those values are inserted into a SELECT query. If the values entered are found as expected, the user is allowed access; if they aren't found, access is denied. However, most Web forms have no mechanisms in place to block input other than names and passwords. Unless such precautions are taken, an attacker can use the input boxes to send their own request to the database, which could allow them to download the entire database or interact with it in other illicit ways. Attackers can insert the malicious query at the end of the URL seen in the address bar of the browser. The target of these attacks are not only limited in the web application but they also can hit desktop applications which have databases powered by SQL. Security vulnerability like SQL injection attack grants the permission to the attacker to access a web application's backend database and destroy confidentiality of data [8]. IDSs are helpful in detecting different types of malicious activities in the network, a combination of good configuration of web server and using parameterized queries in the coding phase can increase the protection against SQL injection attacks [3]. As the amount of SQL injection attacks were increasing it is necessary to prevent the financial losses

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 5, May 2016

occurring due to the attacks. Alhuzali stated that SQL injection attacks occurred when developers had combined hard-coded strings with user input queries [6]. SQL injection attack takes place because of the authentication issues. When user input is not validated then it leads to SQL injection attack [5]. Input validation is required to identify the client, server and URL [7].

Classification of SQL injection attacks:

- Tautologies,
- Illegal/Logically Incorrect Queries
- Union Query
- Inference
- Stored Procedures
- Piggy-Backed Queries
- Alternate Encodings

II. TYPES OF SQL INJECTION

A. Tautologies

The three main purposes of the TAUTOLOGY injection attacks are Identify injectable parameters, Bypass authentication and extract data.

Tautology means a formula which becomes true under all possible interpretations. Similarly in tautology based SQL injection attack the conditional OR operator is used for the injection of the code so that the query always evaluates to TRUE. These attacks are usually done to bypass user authentication [9]. They extract data by inserting a tautology in the WHERE clause of a SQL query. This modified tautology query then transforms the original query into a tautology (that is evaluated to TRUE) thus providing all the rows in the database table to the unauthorized user. A SQL tautology is of the form “or <comparison expression>”, where the comparison expression uses one or more relational operators to compare operands and generate an always TRUE condition.

Consider following Example:

If an unauthorized user id as abcd and password as anything’ or ‘x’=’x then the resulting query will be

Select * from user_details where user_id=‘abcd’ and password = ‘anything’ or ‘x’ =’x’

B. Illegal/Logically Incorrect Queries

The purposes of the logically incorrect queries are to identify injectable parameters, Identify database and extract data.

In this type of injection attacker tries to gather information about the type and structure of the back-end database of a web application. The attack is considered as the preliminary step for further attacks. The attacker fires an incorrect query to the database. The application server returns a default message and the attacker takes advantage of this message. They inject code in vulnerable or injectable parameters which creates syntax, type conversion, or logical error. This output provides with some clue about the database.

Consider following example:

1) Through typing error one can identify the data types of certain columns.
2) Also Logical error often exposes the names of the tables and columns.

C. UNION Query

The purpose of the union query is to bypass authentication and extract data.

This type of attack is done by inserting the union query into a vulnerable parameter which returns a dataset that is union of the original valid query and the results of the injected query. The SQL UNION operator combines the results of two or more queries and makes a result set which includes fetched rows from the partitioning queries in the UNION.

Basic rules for combining two or more queries using UNION:

1) Number of columns and order of columns of all queries must be same.
2) The data types of the columns of involving table in each query should be same or compatible.
3) Usually returned column names are taken from the first query.

By default the union behaves like UNION [DISTINCT] that is it eliminates the duplicate rows but by using the keyword “ALL” with UNION returns all rows using duplicates.

D. Piggy-Backed Queries

The purpose of this attack is to extract data, modify dataset and execute remote commands. This type of attack is different than the other attacks because here the attacker injects additional queries to the original query, as a result the

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 5, May 2016

database receives multiple SQL queries. The first query is the original query which is valid and executed normally. The subsequent queries are the injected queries which are executed in additional to the valid query. These subsequent queries are injected by using a delimiter (";") after the valid query. Due to the misconfiguration a system is vulnerable to the piggy-backed queries and allows multiple statements in one query.

Consider following example:

If the attacker inputs abcd as user-id and ''; drop table xyz – as password in the login form then the application will generate the following query:

Select * from user_table where userid = 'abcd' and password = ''; drop table xyz – '

After completing the first query the database recognizes the query delimiter (";") and executes the injected second query. The second query drops table xyz, which would destroy valuable information.

E. Stored Procedures

The purpose of this attack is privilege escalation, denial of service and executing remote commands.

A stored procedure is actually stored in the data dictionary of the database. The stored procedure includes data validation or access control mechanisms. It can also consolidate and centralize logic that was originally implemented in applications. When execution and processing becomes complex and requires many sql queries then they are moved to the stored procedures and all the applications call the procedures.

Stored procedure type of injection attack try to execute the stored procedures of the database. Most of the databases have procedures apart from the user defined procedures that extend the functionality of the database and allows interaction with the operating system. Initially the attacker will try to find the type of database used through other SQL injection attacks like illegal queries SQL injection attack and further will try to execute various procedures through injected code. As the stored procedures are written by the developers and are vulnerable to execute remote commands, privilege escalation, buffer overflows, and even provide administrative access to the operating system.

For example:

If the attacker injects ';SHUTDOWN;-- into either the user ID or password fields then it will generate the following SQL code:

Select* from user_table where userid = 'abcd' and password = ''; SHUTDOWN; -- '

The above command will cause database to shutdown.

F. Inference

The purpose of this SQL injection to identify injectable parameters, identify schema and extract data.

This type of attack is normally created in the style of true false statement. It is normally applied on well secured databases which do not return any valuable feedback or give error messages. When the attacker finds a vulnerable parameter he injects various conditions by firing various queries and carefully observes the situation to know whether they are true or false. If statement evaluates to TRUE, the page functions normally and if it return false the page behaves significantly different from the normal functioning. This type of injection is called as blind injection.

Time attack is also a type of inference attack. In this attack, the attacker designs a conditional statement and injects it through the vulnerable parameter and gather information based on the time delay in response of the database.

Consider following example:

http;//www.example.com/product.php?producy_id=100 AND if(version) like '5%', sleep(15), 'false'))—

Here the attacker checks whither the system is using a MySQL version 5.x or not, making the server to delay the answer in 15 seconds (the attacker can increase the delay's time).

G. Alternate Encodings

The purpose of the Alternate encoding SQL injection attack is to evade detection.

In this type of attack the attacker injects encoded text to bypass defensive coding practices.

They arrange alternate methods of encoding through their injected strings such as using hexadecimal, ASCII, and Unicode character encoding. Scanning and detection techniques are not fully effective against alternate encodings.

Consider following example:

Select * FROM users WHERE login= 'AND pass ='; exec(char()*7363876d676387))

In the above example char function and ASCII hexadecimal encoding are used.

The char function returns the actual characters(s) of hexadecimal encoding of characters(s).

The encoded string is translated into the shutdown command by database when it is executed.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 5, May 2016

III. PROPOSED SOLUTION

A Liferay is a free and open source portlet technology. Liferay is enterprise portal software product. Liferay Portal is a web platform. Liferay is used for the web portal and websites development. Liferay is available in both community and enterprise edition. Liferay Community edition is free and open source used for web portal development.

Users can develop web portals and portlets using Liferay with less efforts as Liferay contains in built web content management system. Liferay is also known as web application framework. Liferay is purely written in Java. Liferay comes with HSQL (Hypersonic) as default database to the backend. User can configure Liferay with different databases like MySQL, Oracle, and Sybase etc. as a backend at the time of Liferay installation and configuration. Liferay is majorly available in Liferay 6.2.x CE & EE, Liferay 6.1.x CE & EE and Liferay 5.2.x CE & EE versions [4]. As Liferay is a web application, users have to just download and deploy it on system. Liferay comes with sites, organization and user groups, user can select any one according to requirements. Liferay has social networking portlets also using which social networking web sites can be developed. Liferay has very good document management system which is used for managing documents. This document management system is implemented by Java Content Repository (JSR). Liferay supports Service Oriented Architecture (SOA). To integrate Liferay portal with other services we can use this feature of Liferay.

In Liferay we do not write any code for database connectivity. Liferay at backend uses hibernate for database connection. We provide the connection details only during the installation. These details are stored in xml file for further connectivity.

A. Liferay Secure Login Page

1) Understanding SQL injection attacks against Login form:

Login bypass is clearly one of the most popular SQL injection techniques.

Consider Login page with username and password verification. When a user enters a user name and password, SQL query is created and executed to search on the database to verify them.

E. g: username = admin, password=admin.

SQL Query: SELECT * FROM users WHERE name='admin' and password='admin';

The above query searches for the user in the table whose name is admin and password is admin. If corresponding matching entry is found, the user is authenticated.

In order to divert this security mechanism, SQL code has to be injected on the input fields. The SQL code has to be injected in such a manner that the valid result should be produced after SQL statement is executed. If the executed SQL query contains errors in the syntax, it will not fetch a valid result. So injecting random SQL commands and submitting the form will not always produce successful authentication. Let us consider the next example,

E. g: username = admin, password=' 'or '1=1'

SQL Query: SELECT * FROM users WHERE name='admin' and password="or'1=1';

If the username is already known, the only thing to be bypassed is the password verification. So, the SQL commands should be constructed in the similar way.

The **password='' or '1=1'** condition is always true, so the password verification never happens. It can also be said that the above statement is more or less equal to,

SELECT * FROM users WHERE name='admin'

That is just one of the possibility. The real exploit is limited only by the imagination of the tester.

2) Liferay Sign-in Portlet:

Liferay provides a sign-in portlet out of the box. This sign in portlet has its input fields initialized. This is, this input field are not vulnerable to SQL injection. The best way to avoid or prevent SQL injection is secure coding. As using the default sign in portlet we avoid redundent code and also get a secure login portlet. If the developer does not want the default sign-in portlet he can create a custom portlet.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 5, May 2016

3) CRUD Operations using Liferay:

The CRUD operation cycle is at the center of most dynamic websites. CRUD means Create, Read, Update and Delete. (Retrieve may occasionally be substituted for Read.) The CRUD functions are the user interfaces to databases, as they permit users to create, view, modify and alter data. CRUD works on records in databases and manipulates these entities.

Liferay uses MVC architecture by default. MVC is a great design pattern for web applications, but it contains only three components: a model for data, a view for displaying that data, and a controller for handling page flows. The MVC design pattern doesn't have any facility for saving, or *persisting* an application's data model so that it can be retrieved later and be used for further process. For that, we need more layers in our application: a *persistence* layer and a *service* layer.

The persistence layer is mainly responsible for saving and retrieving our model data. The service layer is like a buffer zone between our application and persistence layer. Service layer gives us the freedom in the future to swap out the persistence layer for a different implementation without modifying anything but the calls in the service layer. This kind of loose coupling is robust application design, and Liferay supports that in its frameworks. It does that by providing Service Builder, which is a framework for generating the model, service, and persistence layers of our application.

Service Builder makes use of the hibernate and spring integration DAO (Data Access Objects) mechanism [4].

Main Components in Service Layer

a) **DAO or Persistence classes**
b) **POJO or Model**
c) **Service Classes**

a) DAO or Persistence classes:

DAO or Persistence classes will give access to direct database interaction functionality like insert, delete and update. Business logic for these will be written in these classes.

Each table will have one persistence class thst will provide interaction functionality with database. In this persistence class actual database interaction will be present. Hibernate/jdbc is needed to interact with the database. Persistence/DAO classes will be used to get the session factory object and hence session will be opened. This session object will be used to do all database interactions, and after all the tasks are completed session will be closed.

b) POJO(Plain Old Java Object) or Model:

The POJO or model classes will act as mediator between application layer and database. Data will be passes to database using these classes. Each record in the table is considered as one POJO/Model class object. This object will be used in sessions to pass data into table and to get data from the table into POJO/Model class. During database interaction such objects are called Persistence Objects. These Persistence objects are nothing but java beans. When a session is opened, java bean will bind with table's rows to respective table.

c) Service Classes:

Application layer and Database interaction layer will be separated by service classes. These classes will be used to write business logic if required before inserting data.

To interact with database, DAO classes are used after business logic is performed. There may be many DAO classes to interact with tables in databases.

Liferay service builder takes an input as *service.xml* file and generates set of classes and interfaces to perform CRUD operation on Database.

In *service.xml*, we need to define our business entities and their relation. Each entity in *serivce.xml* represents Database table and attributes of entity are mapped to table column. We can define multiple entities in same *service.xml* file. We can also take the references of entities defined by *service.xml* in other portlets.

Liferay expects us to place the *service.xml* on specific path. It should be placed under **/WEB-INF** folder.
Let us take an example,

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 5, May 2016

Table: users
Columns: f_name, l_name, email, password

Creating tables Without Liferay:
Create table users(id int, f_name varchar(45), l_name varchar(45), email varchar(45), password varchar(45));

Creating tables With Liferay:
In service.xml we write the following code,

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE service-builder PUBLIC "-//Liferay//DTD
Service Builder 6.0.0//EN"
"http://www.liferay.com/dtd/liferay-service-
builder_6_0_0.dtd">
<service-builder package-path="com.test">
   <author>user</author>
   <namespace>test</namespace>
   <entity name="users" local-service="true">
      <column name="userId" primary="true"
type="int"></column>
      <column name="f_name" type="String"></column>
      <column name="l_name" type="String"></column>
      <column name="email" type="String"></column>
      <column name="password" type="String"></column>
   </entity>
</service-builder>
```

After building the service.xml with ant, few classes and interfaces are generated. To perform CRUD operation we require the *Util* classes. The following are main *Util* classes [4]:

- XXXServiceUtil
- XXXLocalServiceUtil
- XXXUtil

XYZ is entity name mentioned in service.xml

- UsersServiceUtil
- UsersLocalServiceUtil
- UsersUtil

I) Inserting records in the table:
i) Without Liferay:

insert into users values(1, 'Joe', 'Smith', 'test@liferay.com', 'I haven't decided yet');

ii) With Liferay:

```
//create primary key
int userId = CounterLocalServiceUtil.increment();

//create student persistance object
Users user = null;
user = UsersLocalServiceUtil.createUsers(userId);
```

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 5, May 2016

```
//add the data in persistance object
user.setFirstName (firstName);
user.setLastName (lastName);
user.setEmail (email);
user.setPassword(password);


//Add user persistence object to database student table
User = UsersLocalServiceUtil.addUsers(user);
```

II) Updating Records in the table:

i) Without Liferay:

'UPDATE users SET password="qwerty" WHERE userId = 1';

ii) With Liferay:

```
Users user = UsersLocalServiceUtil.getUsers(userId);

//fill update information
User.setPassword(password)

//password = qwerty
User = UsersLocalServiceUtil.updateUsers(user);
```

III) Deleting Records from table:

i) Without Liferay:

'DELETE FROM users WHERE userId=1';

ii) With Liferay:

```
Users user =UsersLocalServiceUtil.deleteUsers(userId);
```

IV) Searching Records in the table:

i) Without Liferay:

'SELECT * FROM users where userId=1';

ii) With Liferay:

```
Users user=UsersLocalServiceUtil.getUsers(userId);
```

All the above are just basic operations, if you are not satisfied with these queries you can write your custom query, dynamic query or finders.

From the above we can observe that the developer does not have to write any connection or basic database queries thereby allowing him to concentrate on business logic. The developer can concentrate on validating user input to avoid any SQL exception. As the developer does not have to write the sql query the chance of having any error in coding is reduced, therefore making the code secure.

IV. SIMULATION RESULTS

Liferay is more secure as SQL injection cannot be done on Liferay. Service.xml is build using ant and generate predefined classes. Business logic is written in the serviceUtil classes. As shown in the Fig. 1, from 2004 to 2014 there is no record found of SQL injection attack on Liferay.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 5, May 2016

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
2004	1						1								
2005	1						1								
2007	2						2								
2008	6						4						2		
2009	1						1								
2010	1						1								
2011	5		1				2				2				
2014	2						2								
Total	19		1				14				2		2		
% Of All		0.0	5.3	0.0	0.0	0.0	73.7	0.0	0.0	0.0	10.5	0.0	10.5	0.0	

Fig. 1. Vulnerability Trends over Time

V. CONCLUSION AND FUTURE SCOPE

SQL injection attack is first ranked among the most dangerous attacks. To provide security to web application is important task while developing any web application. SQL injection attack takes the advantage of lack of user input validation and destroys the database confidentiality. There are different techniques to perform SQL injection attacks hence every time using legitimate security solutions, we cannot provide complete security to the web application. Liferay provides secure coding which is the best practice to avoid SQL injection attack. Liferay provides the database connectivity using hibernate and spring. Queries are generated during run time. We don't have to write the queries. Hence SQL injection attack is difficult to occur in Liferay.

REFERENCES

1. (OWASP), "O.W.A.S.P. Top 10 Vulnerabilities."; Available from: https://www.owasp.org/index.php/Top_10.
2. Antunes, N. and M. Vieira, "Defending against Web Application Vulnerabilities." Computer, 2012. 45(2): p. 66-72.
3. Amirmohammad Sadeghian, Mazdak Zamani, Suhaimi Ibrahim, "SQL Injection is Still Alive:A Study on SQL Injection Signature Evasion Techniques", International Conference on Informatics and Creative Multimedia. 2013.
4. "Introduction to Liferay Portal"; Available from: http://www.liferaysavvy.com/2014/01/introduction-to-liferay-portal.html.
5. A. Tajpour, S. Ibrahim, M. Sharif "Web application security by SQL injection detection tools" [J]Int J Comp Sci Issues, 2012.9: 332-339.
6. A. Alhuzali, H. Tora, Q. Nguyen, "Analysis of SQL injection methods and its prevention", Final project of ISA 564 Security Lab, 2012.
7. Li Qian, Jiahua Wan, Lu Chen, Xiuming Chen, "Complete Web Security Testing Methods and Recommendations", the 1st International Conference on Computer Sciences and Applications, CSA2013: 86-89.
8. Atefeh Tajpour, Maslin Massrum, "Comparison of SQL Injection Detection and Prevention Techniques", The 2nd International Conforence on Education Technology and Computer, ICETC 2010: 174-179.
9. "SQL injection methods."; Available from: www.w3resource.com/sql/sql-injection/sql-injction.php.
10. Amirmohammad Sadeghian, Mazdak Zamani, Shahidan M. Abdullah, " A taxonomy of SQL Injection Attacks ", International Conference on Informatics and Creative Multimedia, 2013.
11. Amirmohammad Sadeghian, Mazdak Zamani, Azizah Abd. Manaf, "A Taxonomy of SQL Injection Detection and Prevention Techniques", Internation Conference on Informatics and Creative Multimedia, 2013.