INTERNATIONAL JOURNAL OF PURE AND APPLIED RESEARCH IN ENGINEERING AND TECHNOLOGY

A PATH FOR HORIZING YOUR INNOVATIVE WORK

A DATABASE SECURITY BY PREVENTING SQLINJECTION ATTACKS (SQLIA)

GAJANAN P. BABHULKAR, KEDAR G. PATHARE

M.E. Scholar, Information Technology, Prof. Ram Meghe Institute of Technology and Research, Amravati, India.

Accepted Date: 15/03/2016; Published Date: 01/05/2016

Abstract: The paper focuses on security issues that are associated with the database system that are often used by many firms in their operations. Data is the most valuable asset in today's world as it is used in day –to –day life from a single individual to large organizations. To make the retrieval and maintenance of data easy and efficient it is stored in a database. Databases are a favorite target for attackers because of the data these are containing and also because of their volume. When an internet user interacts in web environment by surfing the Net, sending electronic mail messages and participating in online forums lot of data is generated which may have user's private information. There are many ways a database can be compromised. In this paper issues related to information leakage through SQL injection attacks are presented as well as detection and protection mechanisms are also discussed.

Keywords: Security, SQL Injection, database security, security techniques, integrity.

Corresponding Author: MR. GAJANAN P. BABHULKAR

Access Online On:

www.ijpret.com

How to Cite This Article:

Gajanan P. Babhulkar, IJPRET, 2016; Volume 4 (9): 539-546

PAPER-QR CODE

INTRODUCTION

Database security issues have been more complex due to widespread use and use of distributed client/server architecture as opposed to mainframes system. Databases are a firm main resource and therefore, policies and procedure must be put into place to safeguard its security and the integrity of the data it contains. Besides, access to the database has been become more rampant due to the internet and intranets therefore, increasing the risks of unauthorized access. The objective of database security is to protect database from accidental or intentional loss.

Authorization can be one of the techniques that can be used for granting rights of access of a subject into a system. Another method that is effective is the view. This is a virtual table that can be produced at the time of request of data access. What happens is that view has to have access in the tables other than the base tables in such a way those restrictions are made on the user. This provides appropriate security to crucial data. Back up is the process of taking to an offline storage facility, data and log file. To keep track of transaction involving the database, it is necessary for one to have journal file on all updates of the database.

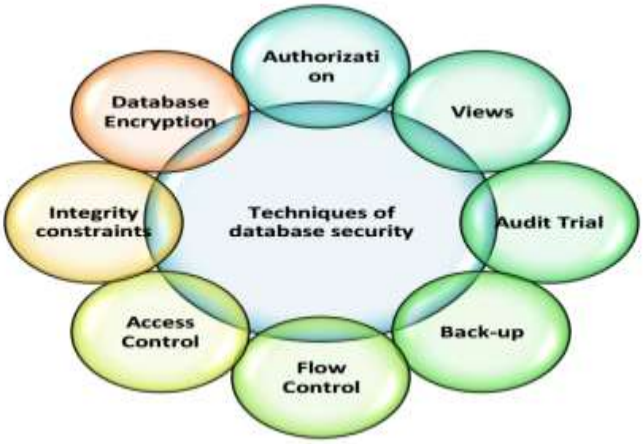

Fig 1.Mechanism of Database Security

In event of failure of the database system, the log file and the database are then used to restore the database to normal functioning position. Integrity constraint is used to contribute to avoid cases of data becoming invalid and hence giving misleading information. The ultimate goal of the constraints is to maintain integrity of the data and hence its consistency. Database

can be secured through encryption. This is encoding of the system using special algorithm that is only accessible when decryption key is provided. This is especial useful when sending sensitive information over communication lines. Audit trial is another method that can help in the database security. Audit trial need to be carried to found the history of operations on the database. It is necessary to restore information lost as well as discover abuse of privileges by any users. Another technique that can be used to secure database is the use of access control. This is the where the access to the system is only given after verifying the credentials of the user and only after such verification is done, the access is given. Use of steganography is rampant in the era of information technology. This technique is used to hide information from unauthorized access.

SQL injection is currently the most common form of web site attack in that web forms are very common, often they are not coded properly and the hacking tools used to find weaknesses and take advantage of them are commonly available online. This kind of exploit is easy enough to accomplish that even inexperienced hackers can accomplish mischief. However, in the hands of the very skilled hacker, a web code weakness can reveal root level access of web servers and from there attacks on other networked servers can be accomplished.

Structured Query Language (SQL) is the nearly universal language of databases that allows the storage, manipulation, and retrieval of data. Databases that use SQL include MS SQL Server, MySQL, Oracle, Access and File maker Pro and these databases are equally subject to SQL injection attack.

Web based forms must allow some access to your database to allow entry of data and a response, so this kind of attack bypasses firewalls and endpoint defenses. Any web form, even a simple logon form or search box, might provide access to your data by means of SQL injection if coded incorrectly.

In a SQL injection attack, an attacker typically inserts (or “injects”) unauthorized SQL statements into a vulnerable SQL data channel. Typically targeted data channels include stored procedures and Web application input parameters. These injected statements are then passed to the database where they are executed. For example in a web application the user inserts a query instead of his name. Using SQL injection, attackers may gain unrestricted access to an entire database [1]. SQL Injection Attacks (SQLIAs) have known as one of the most common threats to the security of database-driven applications. So there is not enough assurance for confidentiality and integrity of this information. SQLIA is a class of code injection attacks that take advantage of lack of user input validation. In fact, attackers can shape their illegitimate

input as parts of final query string which operate by databases. Financial web applications or secret information systems could be the victims of this vulnerability because attackers by abusing this vulnerability can threat their authority, integrity and confidentiality.

Causes of SQL injection

When accessing the application over internet vulnerabilities are the main causes of any kind of attack [2]. In this section, vulnerabilities that might exist naturally in web applications and can be exploited by SQL injection attacks will be presented

Invalidated input: This is almost the most common vulnerability on performing a SQLIA. There are some there is no any checking for them so can be abused in SQL injection attacks. These parameters may contain SQL keywords, e.g. INSERT, UPDATE or SQL control characters such as quotation marks and semicolons.

Generous privileges: Normally in database the privileges are defined as the rules to state which database
subject has access to which object and what operation are associated with user to be allowed to perform on the objects. Typical privileges include allowing execution of actions, e.g. SELECT, INSERT, UPDATE, DELETE, DROP, on certain objects. Web applications open database connections using the specific account for accessing the database. An attacker who bypasses authentication gains privileges equal to the accounts. The number of available attack methods and affected objects increases when more privileges are given to the account. the worst case happen If an account can connect to system that is associated with the system administrator because normally has all privileges.

Uncontrolled variable size: If variables allow storage of data be larger than expected consequently allow attackers to enter modified or faked SQL statements. Scripts that do not control variable length may even open the way for attacks, such as buffer overflow.

Dynamic SQL: SQL queries dynamically built by scripts or programs into a query string. Typically, one or more scripts and programs contribute and finally by combining user input such as name and password, make the WHERE clauses of the query statement. The problem is that query building components can also receive SQL keywords and control characters. It means attacker can make a completely different query than what was intended.

Client-side only control: If input validation is implemented in client-side scripts only, then security functions of those scripts can be overridden using cross-site scripting. Therefore, attackers can bypass input validation and send invalidated input to the server-side.

Into Outfile support: Some of RDBMS benefit from the INTO OUTFILE clause. In this condition an attacker can manipulate SQL queries then they produce a text file containing query results. If attackers can later gain access to this file, they can abuse the same information, for example, bypass authentication.

Multiple statements: If the database supports UNION so, attacker has more chance because there are more attack methods for SQL injection. For instance, an additional INSERT statement could be added after a SELECT statement, causing two different queries to be executed. If this is performed in a login form, the attacker may add him or herself to the table of users.

Types of SQL Injection attacks

1. ***SQL Manipulation***

SQL manipulation usually involves modifying the SQL query through altering the WHERE clause. In this class of attack, amend the WHERE clause of the statement so the WHERE clause constantly results in TRUE. For example,

"SELECT * FROM Student WHERE userid = '118' and password ='aaa' OR '1'='1'" As the statement (1=1) has been added to the query statement so it is always true.

2. Invalid **Queries (Code Injection)**

When a query is rejected, an error message is returned from the database including useful debugging information. This error messages help attacker to find vulnerable parameters in the application and consequently database of the application. In fact attacker injects junk input or SQL tokens in query to produce syntax error, type mismatches, or logical errors by purpose. In this example attacker makes a type mismatch error by injecting the following text into the *pin* input field:

i) Original URL: http://www.arch.polimini.it/eventi/?id_nav=8868
ii) SQL Injection: http://www.arch.polimini.it/eventi/?id_nav=88648'
iii) Error message showed: SELECT name FROM Student WHERE id =8868\' from the message error we can find out name of table and fields: name; Student; id. By the gained information attacker can organize more strict attacks.

3. Function Call Injection

Function call injection is the addition of database functions or user defined functions into a vulnerable SQL queries. These function calls can be used to make internal calls or modify data in the database that can be harmful to the users.

4. Buffer Overflows

SQL injection of buffer overflows is a subset of function call injection. In several commercial and open-source databases, vulnerabilities exist in a few database functions that may result in a buffer overflow.

How SQL Injection Works

Any customers, employees and business partners may all have the right to store or retrieve information from your database. Your site probably allows any site visitor to submit and retrieve data. Legitimate access for visitors includes site search, sign up forms, contact forms, logon forms and all of these provide windows into your database. These various points of access are quite possibly incorporated in 'off-the-shelf' applications or may be custom applications set up just for your site. These forms and their supporting code have likely come from many sources, were acquired at different times and possibly installed by different people.

SQL injection is the use of these publicly available fields to gain entry to your database. This is done by entering SQL commands into your form fields instead of the expected data. Improperly coded forms will allow a hacker to use them as an entry point to your database at which point the data in the database may become visible and access to other databases on the same server or other servers in the network may be possible[4].

Web site features such as contact forms, logon pages, support requests, search functions, feedback fields, shopping carts and even the functions that deliver dynamic web page content, are all susceptible to SQL injection attack because the very fields presented for visitor use MUST allow at least some SQL commands to pass through directly to the database.

Prevention Strategy

i) All queries should be parameterized.

All data access techniques provide some means for escaping SQL meta-characters automatically. The following sections detail how to perform input validation and meta-character escaping using popular data access technologies[3].

ii) Prepared Statements

A Prepared Statement represents a precompiled SQL statement that can be executed multiple times without having to recompile for every execution. Variables passed as arguments to prepared statements will automatically be escaped by the JDBC driver.
Example: 1

```
String selectStatement = "SELECT * FROM Student WHERE userId = ? ";

PreparedStatement prepStmt = con.prepareStatement(selectStatement);

prepStmt.setString(1, userId);

ResultSet rs = prepStmt.executeQuery();
```

Although Prepared Statements helps in defending against SQL Injection, there are possibilities of SQL Injection attacks through inappropriate usage of Prepared Statements. The example below explains such a scenario where the input variables are passed directly into the Prepared Statement and thereby paving way for SQL Injection attacks.

Example: 2

```
String strUserName = request.getParameter("Txt_UserName");

PreparedStatement prepStmt = con.prepareStatement("SELECT * FROM user WHERE userId =
'+strUserName+'");
```

iii) String concatenation should never be used to create dynamic SQL. The important thing to remember is to never construct SQL statements using string concatenation of unchecked input values. Creating of dynamic queries via the java.sql.Statement class leads to SQL Injection.

REFERENCES

1. Mr. Saurabh Kulkarni, Dr. Siddhaling Urolagin, "Review of Attacks on Databases and Database Security Techniques", International Journal of Emerging Technology and Advanced Engineering, ISSN 2250-2459, Volume 2, Issue 11, November 2012.
2. T. Pietraszek and C. V. Berghe. Defending against Injection Attacks through Context-Sensitive String evaluation. Recent Advances in Intrusion Detection, Volume: 3858, Pp: 124- 145, 2006.
3. S. Singh, Database systems: Concepts, Design and applications New Delhi: Pearson Education India, 2009.K. Elissa, "Title of paper if known," unpublished.
4. S. Sumanthi, Fundamentals of relational database management systems Berlin: Springer, 2007.