Application Security

by Sophia Tutorial

WHAT'S COVERED

This tutorial explores security concerns with applications that connect to databases in two parts:

1. Confidentiality, Integrity, and Availability
2. SQL Injections

1. Confidentiality, Integrity, and Availability

Security is always a potential concern when it comes to information systems. Security concerns include data confidentiality, integrity, and availability.

Confidentiality focuses on ensuring that the data is protected against unauthorized access or disclosure of private information. Many organizations have to follow various laws and guidelines around data confidentiality, like the Health Insurance Portability and Accountability Act (HIPAA) in medicine, or the Sarbanes-Oxley Act (SOX) in the business world, as examples. As such, the data stored within databases needs to be classified as highly restricted, where very few individuals would have access (credit card information as an example); confidential, where certain groups would have information (pay information for employees as an example); or unrestricted, where anyone can have access.

Availability focuses on the accessibility of the data when authorized users wanted access to it.

Finally, integrity focuses on ensuring that the data is consistent and free from errors. The database itself plays a key role in data integrity, as we have seen in prior tutorials. Protecting sensitive data often involves using encryption. Through encryption, the underlying data is scrambled, using a key to a format so that it cannot be read as is. There are many forms of encryption with various algorithms. With weaker encryption, you may see attackers trying to decrypt the data by brute force, which basically means trying to guess what the decryption key is through iterative trial and error.

With applications, security vulnerabilities can occur due to many different factors. Individuals can sometimes take advantage of bugs within the application that connects to the database. When code is poorly developed or focused purely on functionality and not security, issues can occur. One example is session hijacking, when an individual takes over a web session from another individual. By doing so, the individual can get access to certain personal information from another user that they may not have been able to access otherwise.

2. SQL Injections

SQL injections are one of the most common web hacking techniques used with applications. It can damage your database, or provide complete unauthorized access to the database if things are not well protected. SQL injection typically works by attempting to add malicious code into SQL statements through the use of web page input.

This can occur if the page asks for input like a userid and concatenates the input to the SQL string. For example, from a program, we may have something that looks like:

myinput = readFromInput(“userid”);

mysql = “SELECT * FROM users where user_id =” + myinput;
If the user enters in 5 for the userid, this would create a SQL statement like:

SELECT *
FROM users
WHERE user_id = 5;
That would be the expected result. However, if the “hacker” entered in “5 or 1=1”, the following SQL statement would look like this:

SELECT *
FROM users
WHERE user_id = 5 or 1=1;
In we look at the statement, the 1=1 will always return true, which means that the query would return every single row within the table. Imagine if the table had usernames, passwords, and other user information. All of that would now be compromised in this successful SQL injection.

Or perhaps the input could look like this “5; DROP TABLE customer;”. The resulting query would look like this:

SELECT *
FROM users
WHERE user_id = 5; DROP TABLE customer;
If this SQL injection is successful, it could potentially drop the table customer, which is also quite problematic.

Different databases handle SQL injection issues slightly differently. To avoid these types of scenarios, what is important is that the application first ensures that the input data has been validated before sending the query to the database. We also want to filter input data to avoid the user bypassing our validation. By filtering the user input, we can ensure that we're checking for any special characters that should not be included. In many applications, the use of SQL parameters can help.

Your turn! Open the SQL tool by clicking on the LAUNCH DATABASE button below. Then enter in one of the examples above and see how it works. Next, try your own choices for which columns you want the query to provide.

SUMMARY

There are many different security concerns surrounding databases, including SQL injections.

Source: Authored by Vincent Tran