SQL Injection Cheat Sheet

What Are SQL Injection Attacks?

SQL injection attacks occur when a malicious user includes SQL elements in the input in a tricky way that the application calls the stored procedure and passes variables to it. The query is prepared and stored in the database engine, so that SQL commands and user input (parameters) are passed separately. The prepared statements library replaces these parameters with values supplied by the user, so that SQL commands and user input (parameters) are passed separately.

SQL Injection Types

TYPE 1: INJECTION BASED SQL INJECTION

The attacker sends a request designed to cause an error in the database server. The server returns an error message to the attacker, which allows them to draw conclusions about the database structure or other sensitive information when the web application

TYPE 2: INJECTION BASED SQL INJECTION: UNION BASED SQL INJECTION

The SQL engine combines sensitive information with legitimate information that the web application

TYPE 3: INSTRUCTIONAL SQL INJECTION

The attacker uses a UNION statement to guess the content of a file names, and more.

TYPE 4: SQL INJECTION: EXECUTE INJECTION

The attacker includes a special database command in the payload – this command causes a request to an external resource (controlled by the attacker)

TYPE 5: OUT-OF-BAND SQL INJECTION

This type of SQL injection is often used to access sensitive information from the database server, which delays the whole response. The attacker draws conclusions from the length of response delays and repeats the same for boolean-based payloads, the attacker sends a payload that includes a time delay SLEEP function. If there is a request coming once the payload is executed, this confirms that the command was executed and the attacker can send a long request to create more delays, which delays the whole response.

What Makes SQL Injection Vulnerable?

SQL injection vulnerabilities occur when the application does not properly validate user input or when it passes user-generated content directly into SQL queries. The application should not accept user input as part of an SQL statement without proper validation. This includes checking the length, format, and content of the input to ensure it is suitable for use in an SQL statement. The application should also use parameterized queries or prepared statements to ensure that user input is properly sanitized before it is used in an SQL query. This helps to prevent SQL injection attacks by ensuring that user input is treated as data, not as part of an SQL statement.

Stored Procedures

A stored procedure is a block of code that can be stored in a database and executed repeatedly. They are useful for complex logic, calculations, and data manipulation. Stored procedures can be executed by calling them from the application, passing parameters that are stored in the database. Stored procedures are often used to implement complex logic, such as validation, cleanup, or data manipulation. They can also be used to implement business logic, such as calculating the total cost of an order or updating the status of a transaction. Stored procedures are a powerful feature of database systems and can be used to improve performance and security.

Basic SQL Injection Example

Create the procedure:

```sql
CREATE PROCEDURE example(IN suppliedId VARCHAR(8))
BEGIN
EXEC database.dbo.example 1;
SELECT column_name FROM table_name WHERE id = @id;
END;
```

This procedure takes a single parameter, `suppliedId`, which is a string. The procedure first executes the command `EXEC database.dbo.example 1;`, which will be used to perform an SQL operation. This command will be executed as an SQL statement, and the result will be returned to the application. The procedure then selects a column from the `table_name` table, where the `id` column is equal to the value of the `suppliedId` parameter.

Calling the procedure with id = 1:

```sql
CALL example("0;DELETE FROM users WHERE 1");
```

The application calls the stored procedure and passes variables to it. The query is prepared and stored in the database engine, and user input (parameters) are passed separately. The prepared statements library replaces these parameters with values supplied by the user, so that SQL commands and user input (parameters) are passed separately. The command `EXEC database.dbo.example 1;` will be executed as an SQL statement, and the result will be returned to the application. The procedure then selects a column from the `table_name` table, where the `id` column is equal to the value of the `suppliedId` parameter.

The SQL injection payload will not work:

```sql
CALL example("0;DELETE FROM users WHERE 1");
```

No, WAFs only make it more difficult for the attacker to send SQL injection payloads.

You have limited web application security knowledge

You are a developer or you know programming

How to fix your code that has SQL injection vulnerabilities

1. Avoid using SQL statements that are constructed from user input. Instead, use parameterized queries or prepared statements, which pass the parameters separately from the query string. This ensures that user input is properly sanitized before it is used in an SQL statement.

2. Use parameterized queries or prepared statements to ensure that user input is properly sanitized before it is used in an SQL statement. This helps to prevent SQL injection attacks by ensuring that user input is treated as data, not as part of an SQL statement.

3. Implement input validation to check the length, format, and content of the input to ensure it is suitable for use in an SQL statement. This helps to prevent SQL injection attacks by ensuring that user input is properly sanitized before it is used in an SQL statement.

4. Use an external script to call the stored procedure, instead of passing parameters directly as input. This helps to prevent SQL injection attacks by ensuring that user input is properly sanitized before it is used in an SQL statement.

5. Use an external script to call the stored procedure, instead of passing parameters directly as input. This helps to prevent SQL injection attacks by ensuring that user input is properly sanitized before it is used in an SQL statement.

Using a Predefined Stored Procedure

CALL example("0;DELETE FROM users WHERE 1");

The application calls the stored procedure and passes variables to it. The query is prepared and stored in the database engine, and user input (parameters) are passed separately. The prepared statements library replaces these parameters with values supplied by the user, so that SQL commands and user input (parameters) are passed separately. The command `EXEC database.dbo.example 1;` will be executed as an SQL statement, and the result will be returned to the application. The procedure then selects a column from the `table_name` table, where the `id` column is equal to the value of the `suppliedId` parameter.

The SQL injection payload will not work:

```
CALL example("0;DELETE FROM users WHERE 1");
```

No, WAFs only make it more difficult for the attacker to send SQL injection payloads.

Using a Predefined Stored Procedure

CALL example("0;DELETE FROM users WHERE 1");

The application calls the stored procedure and passes variables to it. The query is prepared and stored in the database engine, and user input (parameters) are passed separately. The prepared statements library replaces these parameters with values supplied by the user, so that SQL commands and user input (parameters) are passed separately. The command `EXEC database.dbo.example 1;` will be executed as an SQL statement, and the result will be returned to the application. The procedure then selects a column from the `table_name` table, where the `id` column is equal to the value of the `suppliedId` parameter.