# SQL Injection: An Overview

## What is SQL Injection?

SQL Injection is a security vulnerability that allows attackers to interfere with the queries that an application makes to its database. It typically occurs when user input is improperly validated or sanitized and gets included directly in SQL statements. This can lead to unauthorized access, data breaches, or even complete control over the database.

## How Does SQL Injection Work?

SQL Injection exploits vulnerabilities in an application's input handling. Attackers craft malicious inputs to manipulate SQL queries and execute unintended operations. For example:

Vulnerable Code Example (C#):

string query = "SELECT \* FROM Users WHERE Username = '" + username + "' AND Password = '" + password + "'";  
SqlCommand command = new SqlCommand(query, connection);

In this example, an attacker can input something like the following to bypass authentication:

' OR '1'='1

This would result in the following query being executed:

SELECT \* FROM Users WHERE Username = '' OR '1'='1' AND Password = '';

## What Kind of Things Are Possible Through SQL Injection?

Through SQL Injection, attackers can perform a variety of malicious actions, including:

- Authentication Bypass: Accessing restricted areas without valid credentials.

- Data Theft: Extracting sensitive information such as user credentials.

- Data Manipulation: Deleting, updating, or inserting malicious data.

- Privilege Escalation: Gaining administrative access to the database.

- Database Destruction: Dropping tables or entire databases.

- Remote Code Execution: Leveraging database features to execute commands on the server.

## Measures to Prevent SQL Injection (ASP.NET Focus)

To protect your ASP.NET applications from SQL Injection, you can implement the following measures:

1. Use Parameterized Queries (Prepared Statements):

Always use parameterized queries to separate SQL logic from user input. For example:

string query = "SELECT \* FROM Users WHERE Username = @Username AND Password = @Password";  
SqlCommand command = new SqlCommand(query, connection);  
command.Parameters.AddWithValue("@Username", username);  
command.Parameters.AddWithValue("@Password", password);

2. Use Stored Procedures:

Stored procedures execute predefined SQL statements and limit dynamic query construction.

3. Input Validation:

Validate and sanitize all user inputs to ensure they conform to expected formats.

4. Least Privilege Principle:

Use database accounts with the minimum permissions required for the application.

5. Use ORM (Object-Relational Mapping) Frameworks:

Frameworks like Entity Framework help to abstract SQL queries and prevent injection.

6. Enable Parameterized Logging:

Avoid concatenating raw inputs into logs, as this can expose additional vulnerabilities.

7. Use Web Application Firewalls (WAF):

Deploy a WAF to monitor and block malicious SQL Injection attempts.