# Understanding SQL Injection

SQL Injection is a common security issue in web applications. It happens when attackers insert harmful SQL code into user inputs, like forms or search boxes. This can give them access to private information, allow them to change the database, or even take over the entire system. Protecting web applications from SQL Injection requires a thorough understanding of this common web vulnerability.

SQL Injection, often called SQLi, is a weakness in a web page that lets attackers send commands to the database. They exploit weaknesses in web applications by inserting harmful SQL statements through user input, allowing them to manipulate the database. Attackers might use SQL queries like ‘ SELECT, DROP etc ’ to get confidential information that should be hidden.

## Impact of a Successful SQL Injection Attack

* Unauthorized Access to Data: Hackers can get into sensitive information like personal details and financial records.
* Data Manipulation or Deletion:   
  They might change or delete important data, which can mess up the system and disrupt operations.
* Bypassing Authentication:  
   Attackers can break through security measures and gain access to user accounts, even those with admin privileges.
* Exposure of Confidential Information:  
   This can lead to private information being exposed, identity theft, and big financial losses.
* Service Downtime and Reputation Damage:  
   The attack can also cause the service to go offline, hurting the organization’s reputation.

## How to Find SQL Injection Vulnerabilities

* Test the Inputs:   
  Try entering special characters like ' or " in forms or search boxes to see if they cause any errors. This can reveal weaknesses.
* Use Scanning Tools:  
   Programs like SQLMap or Burp Suite can automatically check your web app for problems by pretending to attack it.
* Check the Code:  
   Look at the website’s code to find unsafe ways of using SQL, especially if the code directly uses user input without safety checks.
* Watch for Errors:  
   Pay attention to any unusual database error messages; these might indicate a security issue.
* Do Security Testing:  
   Perform deep security checks, both from the outside (without knowing the code) and the inside (with access to the code), to find and fix any security flaws.

# Use of SQL Injection in Web Applications

* Web servers interact with database servers whenever they need to get or save user data. Attackers use SQL Injection to insert harmful SQL statements that can be executed while the web server is retrieving information from the application server. This allows them to manipulate the database and access or change sensitive data.

### SQL in Web Pages

SQL injection usually happens when a web page asks a user for input, like their username or user ID. Instead of providing their name or ID, the user might enter an SQL statement. If the web page isn’t properly secured, this harmful SQL code could be executed without the web server knowing, allowing the attacker to access or manipulate the database.

### Code Example:

Imagine you have a website that lets users log in. When a user types in their username, the website uses a special language called SQL to find their account in a database. Here's an example of bad code that does this:

username = "SELECT \* FROM Users WHERE username = " + username;

### The Problem:

### Bad code puts user input (username) directly into SQL queries, making it easy for attackers to access sensitive data.

### What an Attacker Can Do:

A sneaky attacker can type in something like "admin OR 1=1" as their username. This would make the SQL query look like this:

SELECT \* FROM Users WHERE username = admin OR 1=1;

### What Happens Next:

When a user enters a username like 'admin OR 1=1', the system should only retrieve the account with the username 'admin'. However, due to poor coding, the system executes the entire input as a SQL query. Since '1=1' is always true, the system returns all user accounts, not just the one with the username 'admin'. This allows the attacker to access every account, giving them unrestricted access to sensitive information.

### 1. Direct SQL Injection

Attackers send bad SQL queries through the website's interface to steal sensitive information or change the database.

### 2. Error SQL Injection

Attackers use error messages from the website to find weaknesses and get access to confidential data or change the database.

### 3. Guessing SQL Injection

Attackers send bad SQL queries and watch how the website responds to figure out if it worked. They have to carefully analyze the website's behavior.

### 4. Indirect SQL Injection

Attackers use a different way to communicate with the database to steal sensitive information. This type of attack is hard to detect and prevent.

### 5. Smart Guessing SQL Injection

Attackers use smart guessing to figure out confidential data. They create queries that always return the same result, no matter what input values they use, to guess sensitive information about the database.

### Impact of SQL Injection

**Consequences of a Successful Attack**

* Attackers can extract sensitive information, including:
  + User details
  + Credit card information
  + Social security numbers
* They can also gain access to protected areas, such as the administrator portal.
* In addition, attackers can delete user data from the tables, causing significant disruption to the website and its users.

### Significance of SQL Injection

* Many online applications, including e-commerce and banking websites, rely on databases to store sensitive information.
* A successful SQL Injection attack can compromise the entire server, allowing attackers to access sensitive information and disrupt the website's operations.
* Therefore, it is essential to take measures to prevent SQL Injection attacks and protect the security of websites and their users.

## SQL Injection Prevention

**Protect Your Database from Attacks**

To prevent SQL injection attacks, developers can take the following measures:

**1**. User Authentication

* Validate user input by setting limits on length and type of input.
* Authenticate users to ensure they are who they claim to be.

### 2. Access Control

* Restrict user access privileges to prevent unauthorized access to sensitive data.
* Limit the amount of data that can be accessed by outsiders.

### 3. Avoid Using System Administrator Accounts

* Don't use system administrator accounts for everyday tasks.
* Use separate accounts with limited privileges instead.

## SQL Injection Based on Batched SQL Statements

Most databases support batched SQL statements, which are collections of two or more SQL statements separated by semicolons.

**Example of a Batched SQL Statement**

The following SQL statement will return all rows from the "users" table and then delete the "Employees" table:

1 SELECT \* FROM Users;

2 DROP TABLE Employees;

3 txtEmail = getRequestString("Email");

4 txtSQL = "SELECT \* FROM Users WHERE Email = " + txtEmail;

5 SELECT \* FROM Users WHERE EmpId = 116;

final emailRegex = r"^[a-zA-Z0-9]+@gmail\.com$";

final passwordRegex = r"^[a-zA-Z0-9]{8,}$";

### Email:

r"^[a-zA-Z0-9]+@gmail\.com$"

### Explanation:

1) ^ asserts the start of the string

2) [a-zA-Z0-9]+ matches one or more alphanumeric characters (letters and numbers)

3) @ matches the @ character

4) gmail matches the string "gmail"

5) \. matches the . character (escaped with \)

6) com matches the string "com"

7) $ asserts the end of the string

Password:

r"^[a-zA-Z0-9]{8,}$"

Explanation:

1) ^ asserts the start of the string

2) [a-zA-Z0-9] matches any alphanumeric character (letter or number)

3) {8,} matches the preceding element (alphanumeric character) 8 or more times

4) $ asserts the end of the string