SQL - Injection

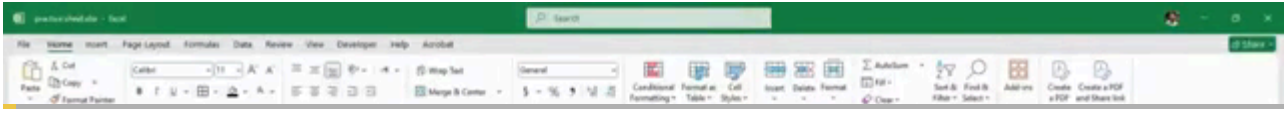
What is SQL Injection?

SQL Injection is a way for hackers to trick a website into giving them access to data they are not supposed to see. Websites use databases to store things like usernames, passwords, and other information.

Normally, when you type into a form (like a login box), the website sends your input to the database. If the website doesn't check your input properly, a hacker can type in special commands instead of normal text, and the database will follow those commands.

- This is dangerous because it can let attackers see private data.
- It can allow them to log in without a password.
- They can change or delete information.
- In some cases, attackers can even take control of the system.

In simple words, SQL Injection happens when a website trusts user input too much and doesn't clean or check it before using it in the database.



How SQL Injection Works

SQL Injection occurs when a website directly uses what a user types into a form inside a database query, without checking it first. This means the database might accidentally treat the user's input as a command instead of just text.

For example, imagine a login form where the website takes the username and password and puts them straight into a query. If a hacker types in special symbols or commands instead of normal text, the database can be tricked into giving access without the real password. consider the following PHP code snippet that checks user login:

```php
<?php
// Vulnerable code
$username = $_POST['username'];
$password = $_POST['password'];

$query = "SELECT * FROM USERS WHERE USERNAME = '$username' AND PASSWORD = '$password'";
$result = mysqli_query($conn, $query);
?>
```

If a malicious user enters **' OR '1'='1** as the password, the query becomes:

```
SELECT * FROM USERS WHERE USERNAME = 'admin' AND PASSWORD = '' OR '1'='1';
```

This condition is always true because **'1'='1'** evaluates to true, allowing the attacker to log in without knowing the actual password.

Types of SQL Injection

Following are different types of SQL injection:

- **Classic SQL Injection:** The hacker directly adds bad commands into a database query to make it do things it shouldn't.
- **Union-Based SQL Injection:** The hacker uses the **UNION** command to get information from other tables in the database.
- **Error-Based SQL Injection:** The hacker looks at error messages from the database to learn secret information about it.
- **Blind SQL Injection:** The hacker can't see error messages, so they guess information by watching how the website behaves (like true/false responses, time delays, etc.).

Example: Error-Based SQL Injection

Let's say we have a table called **CUSTOMERS** that stores customer details like name, age, address, and salary. We can create it using this query:

```
CREATE TABLE CUSTOMERS (
   ID INT NOT NULL,
   NAME VARCHAR (20) NOT NULL,
   AGE INT NOT NULL,
   ADDRESS CHAR (25),
   SALARY DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

Now, let us insert few records into this table using the INSERT statement as follows:

```
INSERT INTO CUSTOMERS VALUES
(1, 'Ramesh', 32, 'Ahmedabad', 2000.00 ),
(2, 'Khilan', 25, 'Delhi', 1500.00 ),
(3, 'Kaushik', 23, 'Kota', 2000.00 ),
(4, 'Chaitali', 25, 'Mumbai', 6500.00 ),
(5, 'Hardik', 27, 'Bhopal', 8500.00 ),
(6, 'Komal', 22, 'Hyderabad', 4500.00 ),
(7, 'Muffy', 24, 'Indore', 10000.00 );
```

The table will be created as follows:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	Hyderabad	4500.00
7	Muffy	24	Indore	10000.00

Imagine a website uses this query to get a customer's name and salary by ID:

```
SELECT NAME, SALARY FROM CUSTOMERS WHERE ID = '1';
```

If a hacker types **1 OR 1=1** instead of a normal ID, the query becomes:

```
SELECT NAME, SALARY FROM CUSTOMERS WHERE ID = 1 OR 1=1;
```

Because **1=1** is always true, the database returns **all rows** instead of just one record, exposing everyone's information:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	Hyderabad	4500.00
7	Muffy	24	Indore	10000.00

How to Prevent SQL Injection

Following are the best ways to prevent SQL Injection:

- **Use Prepared Statements (Parameterized Queries):** Make sure user input is treated only as data, not as part of the database commands.
- **Use Stored Procedures:** Keep SQL code inside the database instead of building queries with user input.
- **Validate and Clean Input:** Always check and clean what users type before using it in queries.
- **Give Minimum Permissions:** Give database accounts only the access they really need.
- **Use ORM Libraries:** Tools like Hibernate or Entity Framework handle SQL safely for you.
- **Turn Off Detailed Error Messages:** Don't show database errors to users, so hackers can't learn about the database.

Example of a Safe Query (Prepared Statement)

In this example, the website uses a **prepared statement**. The question marks **?** act as placeholders for user input. Then, the **bind_param** function safely attaches the username and password to the query.

This ensures that no matter what the user types, it is treated only as data and **cannot change the structure of the SQL query**. As a result, SQL Injection attacks are prevented, making the application much safer.

```php
<?php
// Safe code using prepared statements
$stmt = $conn->prepare("SELECT * FROM USERS WHERE USERNAME = ? AND PASSWORD = ?");
$stmt->bind_param("ss", $username, $password);
$stmt->execute();
$result = $stmt->get_result();
?>
```

Here, user inputs are safely bound to the query, eliminating the risk of injection.

Conclusion

SQL Injection is a very common and serious security problem that can let attackers access or change your data. Developers can protect their websites by following simple safety steps like using prepared statements, checking and cleaning user input, and giving database accounts only the access they need. By doing these things, applications become much safer from SQL Injection attacks.