How to Prevent SQL Injection in PHP

SQL injection is one of the most common vulnerabilities in applications on the web today. This article will show you how to 100% prevent SQL injection on your website using Prepared Statements in PHP.

Method 1 What is SQL Injection?

SQL Injection is a type of vulnerability in applications that use an SQL database. The vulnerability arises when a user input is used in a SQL Statement.

Like Below:

```
$name = $_GET['username'];
$query = "SELECT password FROM tbl_user WHERE name = '$name' ";
```

As you can see the value the user enters into the URL variable *username* will get assigned to the variable *$name* and then placed directly into the SQL statement. This means that is possible for the user to edit the SQL statement.

```
$name = "admin' OR 1=1 -- ";
$query = "SELECT password FROM tbl_user WHERE name = '$name' ";
```

The SQL database will then receive the SQL statement as the following:

```
SELECT password FROM tbl_users WHERE name = 'admin' OR 1=1 -- '
```

Which is valid SQL, and instead of returning one password for the user, the statement would return all the passwords in the table *tbl_user*. This is not something anyone wants in their web applications. This article will show you how to prevent this type of vulnerability.

Method 2 Use Prepared Statements

To prevent SQL injections we will have to use something called prepared statements which uses bound parameters. Prepared Statements do not combine variables with SQL strings, so it is not possible for an attacker to modify the SQL statement. Prepared Statements combine the variable with the compiled SQL statement, this means that the SQL and the variables are sent separately and the variables are just interpreted as strings, not part of the SQL statement.

Method 3

Prepared Statements with mySQLi.

Using the methods in the steps below, you will not need to use any other SQL injection filtering techniques such as mysql_real_escape_string(). This is because with prepared statements it is not possible to do conventional SQL injection.

1 **mySQLi SELECT Query.**
The below script is how to SELECT data from a table using mySQLi Prepared Statements.

```
$name = $_GET['username'];

if ($stmt = $mysqli->prepare("SELECT password FROM tbl_users WHERE name=?")) {

    // Bind a variable to the parameter as a string.
    $stmt->bind_param("s", $name);

    // Execute the statement.
    $stmt->execute();

    // Get the variables from the query.
    $stmt->bind_result($pass);

    // Fetch the data.
    $stmt->fetch();

    // Display the data.
    printf("Password for user %s is %s\n", $name, $pass);

    // Close the prepared statement.
    $stmt->close();

}
```

Note: The variable $mysqli is the mySQLi Connection Object.

2 **mySQLi INSERT Query.**
The below script is how to INSERT data into a table using mySQLi Prepared Statements.

```
$name = $_GET['username'];
$password = $_GET['password'];

if ($stmt = $mysqli->prepare("INSERT INTO tbl_users (name, password) VALUES (?, ?)")) {

    // Bind the variables to the parameter as strings.
    $stmt->bind_param("ss", $name, $password);

    // Execute the statement.
    $stmt->execute();

    // Close the prepared statement.
    $stmt->close();

}
```

Note: The variable $mysqli is the mySQLi Connection Object.

3 **mySQLi UPDATE Query.**
The below script is how to UPDATE data in a table using mySQLi Prepared Statements.

```
$name = $_GET['username'];
$password = $_GET['password'];

if ($stmt = $mysqli->prepare("UPDATE tbl_users SET password = ? WHERE name = ?")) {

    // Bind the variables to the parameter as strings.
    $stmt->bind_param("ss", $password, $name);

    // Execute the statement.
    $stmt->execute();

    // Close the prepared statement.
    $stmt->close();

}
```

Note: The variable $mysqli is the mySQLi Connection Object.

4 **mySQLi DELETE Query.**
The below script is how to DELETE data from a table using mySQLi Prepared Statements.

```
$name = $_GET['username'];
$password = $_GET['password'];

if ($stmt = $mysqli->prepare("DELETE FROM tbl_users WHERE name = ?")) {

    // Bind the variable to the parameter as a string.
    $stmt->bind_param("s", $name);

    // Execute the statement.
    $stmt->execute();

    // Close the prepared statement.
    $stmt->close();

}
```

Note: The variable $mysqli is the mySQLi Connection Object.

Community Q&A

Why use the $_GET method instead of the $_POST method?

wikiHow Contributor

$_GET is easier to use when developing or when no user-sensitive data is being sent. Examples could include info for the generation of the webpage, or a position where the user scrolled last on the page, etc.

Flag as duplicate

Not Helpful 0 Helpful 0

Can you answer these readers' questions?

Refresh

On **How to Contact Livingsocial**, a reader asks:

How can I get a deal bucks credit for $72.98 moved to my credit card?

Your answer...

Reply

On **How to Check Child Support Payments Online**, a reader asks:

Why hasn't my new child support card been sent?

Your answer...

Reply

On **How to Apply Advantix for Dogs**, a reader asks:

Is it okay to apply Advantix to my dog in wet weather?

Your answer...

Reply

Further Reading

- Info on SQL Injection