

Databases and Application Dev

by Callum Bailey

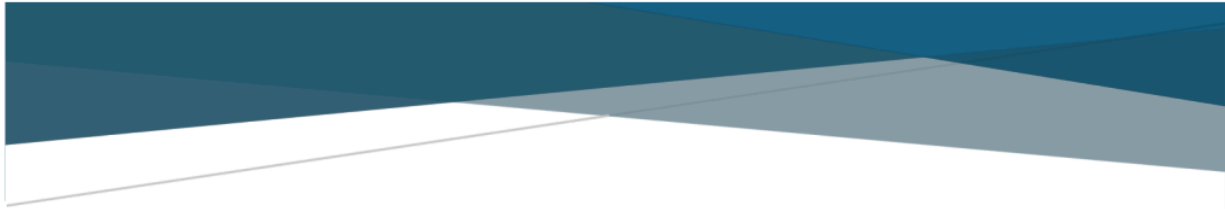
Submission date: 01-Jan-2025 09:36PM (UTC+0000)

Submission ID: 246992219

File name: 49684_Callum_Bailey_Databases_and_Application_Dev_630034_1606015026.docx (300.33K)

Word count: 626

Character count: 3508



DATABASES AND APPLICATION DEVELOPMENT

Callum Bailey (2307997)

Contents

Introduction 2

Implementation of SQLI..... 2

Conclusion 5

Reference List..... 5

Introduction

SQL Injection is a security vulnerability for websites that exploits weaknesses with SQL statements when interacting with databases. These can be exploited either by malicious attackers who know how to manipulate SQL queries by injecting input into the fields such as when entering data into a form or by regular users who mistakenly input 'special' characters which results in the remainder of the statement to be ignore and sometimes retrieving data. To prevent this issue arising, user data inputs should be sanitised or adapted to not allow injection to occur.

SQL Injection is a significant issue for websites as can cause issues with:

- **Data confidentiality:** Attackers can extract sensitive data, such as user credentials, financial records, or other private information.
- **Data Corruption:** Users using SQL injection can manipulate, update, and delete data, which can cause massive destruction to organisations that rely on data to function.
- **Disrupt services:** In extreme cases, attackers can delete entire databases or execute commands that render a website unusable.

To mitigate these problems, the first defence is to restrict what can be entered into forms on a website. We can protect a large proportion of the data by using prepared statements, this is accomplished by the statements separating the SQL code from the data, this ensures that the user inputs are treated as data rather than code.[1]
There are few ways in which attackers can use SQL injection these can include:

- **Error Based SQL injection:** This is where the attacker uses their input to cause the system to create an error and reveal sensitive information.
 - **Boolean-based SQLi:** 'The attacker sends SQL queries that force the application to return TRUE or FALSE, allowing data retrieval one bit at a time.'
- [2]

Implementation of SQLI

In my code I have decided to protect against SQL injection by using prepared statements and therefore separating the code from the input data as described above. Below are screenshots of where I have incorporated prepared statements:

```
10 <?php
11 include 'db.php';
12
13 if (isset($_GET['search'])) {
14     $search = '%' . $_GET['search'] . '%';
15     $query = "SELECT * FROM Cars WHERE make LIKE ? OR model LIKE ?";
16     $stmt = mysqli_stmt_init($conn);
17     mysqli_stmt_prepare($stmt, $query);
18     mysqli_stmt_bind_param($stmt, "ss", $search, $search);
19     mysqli_stmt_execute($stmt);
20     $result = mysqli_stmt_get_result($stmt);
21
22     echo "<h1>Search Results</h1>";
23     if (mysqli_num_rows($result) > 0) {
24         while ($car = mysqli_fetch_assoc($result)) {
25             echo "<div>";
26             echo "<h2>" . $car['make'] . " " . $car['model'] . "</h2>";
27             echo "<p>Year: " . $car['year'] . "</p>";
28             echo "<p>Price: $" . $car['rental_price_per_day'] . " per day</p>";
29             echo "<h1>" . "<img src='" . $car['image'] . "' style='width:400px;height:300";
30             echo "</div>";
31         }
32     } else {
33         echo "<p>No cars match your search.</p>";
34     }
35 }
```

This screenshot is of my file search_cars.php, the prepared statement covers all user input on this page. I then tried to enter ' OR 1=1 – which is used by attackers to make a statement return as 'true' which has the potential to retrieve information from a database. The entry and result are shown in the below screenshots, and I believe the show that my prepared statements have been effective in preventing SQL injection.

[Back to home](#)
Search by Make/Model:

Search

[Back to home](#)
Search by Make/Model:

Search

Search Results
No cars match your search.

```

<?php
include 'db.php';

if ($_SERVER['REQUEST_METHOD'] === 'POST') {

    $user_id = isset($_POST['user_id']) ? intval($_POST['user_id']) : null;
    $car_id = isset($_POST['car_id']) ? intval($_POST['car_id']) : null;
    $start_date = isset($_POST['start_date']) ? $_POST['start_date'] : null;
    $end_date = isset($_POST['end_date']) ? $_POST['end_date'] : null;
    $Email = isset($_POST['Email']) ? strtolower($_POST['Email']) : null;

    $query = "INSERT INTO bookings (user_id, car_id, start_date, end_date, Email) VALUES
    $stmt = mysqli_stmt_init($conn);
    mysqli_stmt_prepare($stmt, $query);
    mysqli_stmt_bind_param($stmt, "iiss", $user_id, $car_id, $start_date, $end_date, $Em

    if (mysqli_stmt_execute($stmt)) {
        echo "Booking successfully added!";
    } else {
        echo "Error: " . mysqli_stmt_error($stmt);
    }
}
?>

```

Here is a screenshot of my booking.php file, where users can make a booking for their desired vehicle, similarly this covers all the user inputs on this page.

These implementations should be enough to prevent SQL injection. There is an area where SQL injection may occur this is on my update_car.php file, which is displayed below, I haven't included SQL injection here, but a prepared statement could also be used to prevent this. The statement would need to cover ID and Price inputs.

```

<link rel="stylesheet" href="stylesheet.css">
<a href="landingpage.php">

Back to home</a>
<form action="update_car.php" method="POST">
    <label for="id">Car ID:</label>
    <input type="number" name="id" id="id" required><br>
    <label for="price">New Rental Price:</label>
    <input type="number" name="price" id="price" step="0.01" required><br>
    <button type="submit">Update Price</button>
</form>

<?php
include 'db.php';

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $id = intval($_POST['id']);
    $price = floatval($_POST['price']);

    $query = "UPDATE Cars SET rental_price_per_day = $price WHERE id = $id";
    if (mysqli_query($conn, $query)) {
        echo "Car updated successfully!";
    } else {
        echo "Error: " . mysqli_error($conn);
    }
}
?>

```

Conclusion

Overall, it is evident that SQL injection if not controlled can be at a detriment to an organisation, prepared statements are also essential in preventing injection and should be used wherever possible and will protect from malicious individuals trying to make attacks on databases. By ensuring SQL injection does not occur data protection is preserved, and the database can continue being used as intended.

Reference List

[1]

etd_admin, "How to Prevent SQL Injection in PHP While Using Prepared Statements with User Inputs - Explain to Dev," *Explain to Dev - Empowering developers with the knowledge to build, create, and innovate in the software world.*, Nov. 23, 2024.
<https://explaintodev.com/how-to-prevent-sql-injection-in-php-while-using-prepared-statements-with-user-inputs/> (accessed Jan. 01, 2025).

[2]

TechMindXperts, "The Ultimate Beginner's Guide to SQL Injection: Understanding, Examples, and Prevention," *Medium*, Apr. 30, 2023.
<https://medium.com/@techmindxperts/the-ultimate-beginners-guide-to-sql-injection-understanding-examples-and-prevention-a05f93529aa2>

Databases and Application Dev

GRADEMARK REPORT

FINAL GRADE

63/100

GENERAL COMMENTS

Database Structure and Design - Quite good with the linking tables, but there are a lot of repeating groups of text in make and model which could be normalised out 7/10

Range of back-end coding - Make sure when developing websites that your homepage is either home.php or index.php so that it loads automatically.

I'm unclear why you have a log in, yet add, edit and delete are available even if you're not logged in. Login doesn't seem to work anyway

Delete is there with an id, this would be better with a button next to a car - it's hard to remember a car id and easy to get it wrong. Same comment for edit

Almost impossible to use the booking from a user point of view because it required you to remember User id and car id - a pulldown menu or similar would be better here. Some validation provided however.

Search is OK, you can search for Make/Model, might be nice if was more extensive eg search for year as well.

Image was uploaded

Overall integration of website - Very sparsely styled. Navigation is OK but a bit basic. Would be better ot have navigation bar at the top. 5/10

SQL Injection overview - A reasonable summary, but you need some code examples to support your words, even if it's just the 1=1 example from class and show how it causes a problem. 9/15

SQL Code implementation/proposal - Good, you have added prepared statements where appropriate - there are a couple of other places that you could have mentioned too 12/15

General comments:

You must avoid the use of "I", "my" and so on in academic reports - particular at Levels 5 and 6.

Overall mark for PHP assignment: 63%

Overall mark for module (including Divya's assignment) - 56%

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6
