

LB Application Dev

by Lucy Boyt

Submission date: 11-Jan-2025 11:16AM (UTC+0000)

Submission ID: 248024238

File name: 49296_Lucy_Boyt_LB_Application_Dev_630034_1999997920.pdf (1.79M)

Word count: 3148

Character count: 18496



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Database Design and Programming

Databases and Application Development – ACCA5026

Lucy Boyt - 2308345

Contents

SQL Injection 2

Code Review..... 3

 References..... 28

 Appendix 28

 Image References..... 29

SQL Injection

Structured Query Language is a standard language used to store, manipulate and retrieve data within databases. SQL is a critical component used across various industries and technologies, for an example it is used within online banking, health care systems, social media platforms and E-commerce sites. [1]

When developers have not adequately validated user input before using it in a database query, there is a risk of SQL Injection. SQL injection is a vulnerability which can allow attackers to manipulate how an application interacts with its database by inserting harmful SQL commands into input fields such as search boxes and login forms. With the correct command, attackers can access, alter or delete information without authorisation. This could lead to attackers obtaining administrative control of databases, data breaches and compromised user privacy. [2]

In 2021, the Open Web Application Security Project (OWASP) listed Injection as the third most common security vulnerability in web applications.[3] In 2013, Yahoo experienced one of the largest data breaches in history. The SQL injection attack led to the exposure of 3 billion accounts, including security questions and answers, this raised concerns about potential identity theft. Fortnite reported a vulnerability which allowed attackers to access over 350 million user accounts; the issue was patched promptly. In 2018, a vulnerability in the Cisco Prime License manager was discovered; an attack could exploit POST requests and allow access to modify and delete data and gain shell access with privileges. The issue was patched on 28th November 2018. [4]

Attackers can manipulate user input to execute commands, for example, gaining admin or super user privileges, obtaining passwords, escalating privileges and executing OS-level commands. Such attacks exploit flaws such as direct interpolation of inputs, lack of validation and excessive database privileges.

OWASP provides the five best ways to mitigate injection vulnerability, such as HTML Encode all user input, using static analysis tools, using prepared statements (allowing the SQL parser to distinguish between code and data), using stored procedures and providing developer training on securing code. [5]

An example of SQL injection from User input would be if an attacker wanted to expose sensitive data such as usernames and passwords:

Original Script:

```
$keyword = $_GET['keyword'];  
$query = "SELECT name, price FROM products WHERE name LIKE '%$keyword%'";  
$result = odbc_exec($conn, $query);
```

An attacker could submit "UNION SELECT username, password FROM users; --"

Which would amend the query to: `SELECT name, price FROM products WHERE name LIKE '%'`
`UNION SELECT username, password FROM users;--'`;

This query allows the attacker to retrieve all usernames and passwords from the users table. Parameterised queries could have prevented this output from being provided.

For an example:

```
$keyword = $_GET['keyword'];  
  
$query = "SELECT name, price FROM products WHERE name LIKE ?;"  
$stmt = odbc_prepare($conn, $query);  
  
$param = "%$keyword%";  
odbc_execute($stmt, [$param]);
```

Within the above example, the user can still input their intended malicious query; however, it will be processed differently. The query is sent to the database with a placeholder '?' for the **\$param**, **\$stmt** represents a pre-compiled SQL query that has not been executed. **odbc_prepare** is used to separate the SQL code from the data input; this means any special characters that could be used maliciously are seen as plain data and not as part of the SQL command, which prevents attackers from altering the query logic. **odbc_execute** is then used to execute the prepared statement and parameter (\$stmt, \$param). As the query and the data are handled separately, malicious input will not alter the SQL syntax. The query has also been pre-compiled, which can improve performance if executed multiple times with different inputs. [6]

Code Review

The following table demonstrates SQL injection attempts across all pages to identify any weaknesses, the table will stipulate the type of injection attempted, provide a screenshot of the code and result from the website.

There are five types of injection attempted within these tests:

- 1) Bypass authentication: `'OR 1=1 --`, this injects a condition that is always true bypassing login authentication by terminating the query with `--` (commenting out the rest)
- 2) Extract database information: `' UNION SELECT null, database()` – this method uses a union statement to combine results from the query with the name of the current database.
- 3) Blind SQL injection `' AND 1 = 1 --` and `' AND 1=0 --` this tests if the website responds differently to a true or false condition.

- 4) Time based blind SQL injection ' OR SLEEP(5) -- delays the server response by 5 seconds if the query executes successfully which identifies vulnerabilities when no output is visible.
- 5) Exploit Information schema uses: ' UNION SELECT table_name, null FROM information_schema.tables -- retrieves table names from the database metadata which is useful for enumerating the database structure.

Website Name: landingpage.php (connects to) searchresults.php

Screenshot:

```

66      <?
67      $query = "SELECT Name FROM categories ORDER BY categoryID";
68      $result=mysqli_query($link,$query);
69
70      # while loop for dropdown
71      while ($r=mysqli_fetch_array($result))
72      {
73          echo '<option value="' . $r["Name"] . '">' . $r["Name"] . '</option>';
74      }
75      ?>
76      </select>
77

```

Figure 1 - landingpage.php query

Figure 2 - landingpage.php search function (minimal user input)

```

32
33 $level = $_POST['dl'];
34 $category = $_POST['cat'];
35

```

Figure 3 - variables set for searchresults.php

```

45 $query="SELECT
46     quests.QuestID,
47     quests.Title,
48     quests.Summary,
49     quests.Difficulty,
50     quests.Rewards,
51     categories.Name
52 FROM quests
53 INNER JOIN Categories on quests.CategoryID = categories.CategoryID
54 WHERE quests.difficulty = '$level' AND categories.Name LIKE '%$category%';
55
56 $result=mysqli_query($link,$query);
57 $numrows=mysqli_num_rows($result);
58

```

Figure 4 - query on searchresults.php

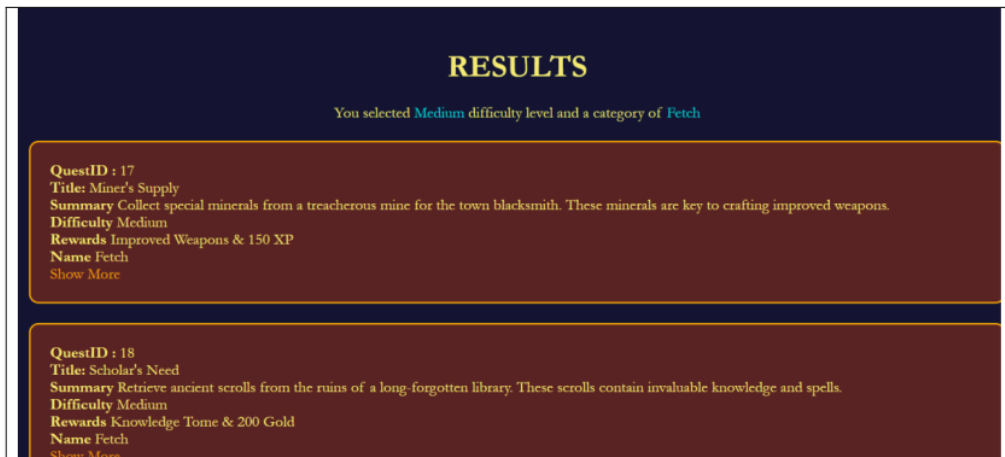


Figure 5 - Searchresults.php – outcome

SQL injection query: This query does not involve text input by the user. Users are provided options retrieved from the database to complete the query.

Susceptible to injection:

As the query does not require user manual text input it is less likely to be vulnerable to SQL injection.

Website Name: Search.php (connects to) dosearch.php

Screenshots:

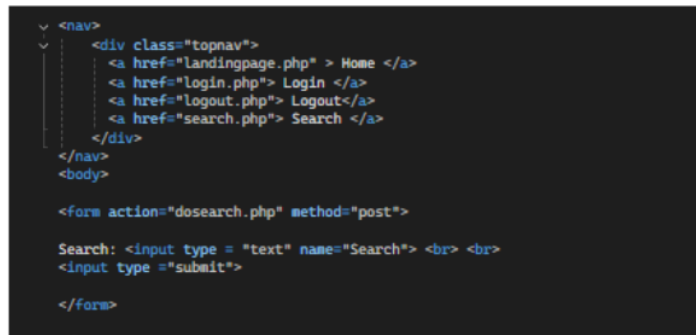


Figure 6 - search.php input options

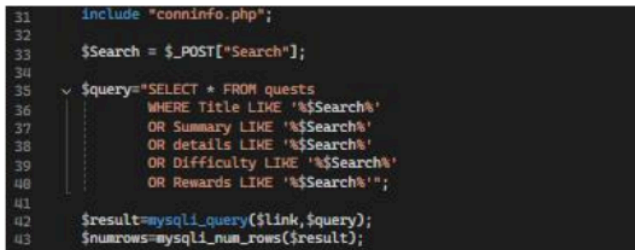


Figure 7 - dosearch.php query



Figure 8 - Search results following injection query

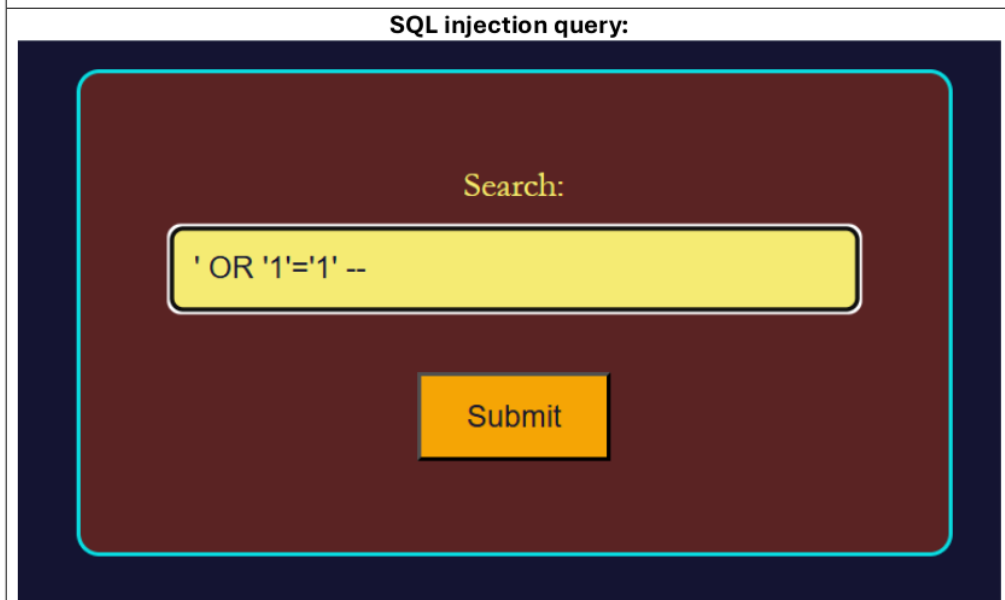


Figure 9 - Bypass authentication attempt

Susceptible to injection: As seen from the above screen shots an attacker can inject malicious SQL code bypassing filters to retrieve unintended data. This is due a lack of validation or sanitisation to the code. With this query the user can input \$Search into multiple fields without any safeguards. To improve this code using prepared statements to the following would prevent misinterpretation of the query.

```

$search = $_POST["Search"];

$query="SELECT * FROM quests
WHERE Title LIKE ?
OR Summary LIKE ?
OR details LIKE ?
OR Difficulty LIKE ?
OR Rewards LIKE ? ";

// add wildcards for the like operator
$search_param = "%" . $search . "%";

// Prepared statement
$stmt = mysqli_stmt_init($link);
mysqli_stmt_prepare($stmt, $query);
mysqli_stmt_bind_param($stmt, "sssss", $search_param, $search_param, $search_param, $search_param, $search_param);
mysqli_stmt_execute($stmt);

$result = mysqli_stmt_get_result($stmt);

$numrows=mysqli_num_rows($result);

```

Figure 10 - Amended prepared statement

Website Name: login.php (connects to) checklogin.php

Screenshot:

```

30      <form action="checklogin.php" method="POST">
31      <b>Please login:</b><br><br>
32      Username: <input type="text" name="un"><br>
33      Password: <input type="text" name="pw">
34      <br><br>
35      <input type="submit" value="Login">

```

Figure 11 - login.php input code

```

35      // submitted input
36      $un=$_POST["un"];
37      $pw=$_POST["pw"];
38
39      $query="SELECT * FROM users
40      WHERE Username = ?
41      AND Password = ? ";
42
43      $stmt = mysqli_stmt_init($link);
44      mysqli_stmt_prepare($stmt, $query);
45      mysqli_stmt_bind_param($stmt, "ss", $un, $pw);
46      mysqli_stmt_execute($stmt);
47
48      $result = mysqli_stmt_get_result($stmt);
49
50
51      $numrows = mysqli_num_rows ($result);

```

Figure 12 - checklogin.php prepared variables for query

Home Login Logout Search

Sorry, your log in details did not match our system [Back to home](#)

Figure 13 - result following attempted injection

SQL injection query:

Username:

' OR 1=1 --

Password:

admin

Login

Figure 14 - bypass attempt

Username:

' UNION SELECT null, database()

Password:

Figure 15 - Union injection attempt

Susceptible to injection: The script is not susceptible to bypass filtering or union injection. Further security would be beneficial when using passwords, within a live public database the passwords would be encrypted using the password_hash() function.

Website Name: checklogin (links to) account

Screenshot:

```
31 include "conninfo.php";
32
33 // Check if the user is logged in
34
35 if ($_SESSION['Username'] == "") {
36     echo "You are not logged in <br><br>";
37     ?>
38     <a href="login.php"> Login Page </a>
39     <?
40     exit();
41 } else {
42     echo "Logged in as: " . $_SESSION['Username'] . "<br>";
43 }
44
45 // Fetch details from database
46
47 $un = $_SESSION['Username'];
48 $query = "SELECT * FROM users WHERE Username = '$un'";
49 $result = mysqli_query($link, $query);
50
51 // Check if the query was successful
52
53 if (!$result) {
54     echo "Database query failed: " . mysqli_error($link);
55     exit();
56 }
57
58 // Fetch user details
59
60 $row = mysqli_fetch_assoc($result);
61 if (!$row) {
62     echo "No user found with username: " . $un;
63     exit();
64 }
65
66 ?>
67
```

Figure 16 - account.php script uses the username linked to the session to verify if the correct user is logged in.

```
79 if ($row['user_type'] == 'Admin') {
80     ?>
81 }
82 </div>
83 Admin Function
84 <div class="admin-link">
85     <a href="crud.operations/erasequests.php"></a> <!-- Delete quests -->
86 </div>
87 <? }?>
```

Figure 17 - Admin Functions

SQL injection query:

The connection between these two pages does not require any user input. The user will have been provided with an account screen that uses the user login details input on login.php. The page verifies that the session is still live with the previously input login details. As there is no user input required for these pages there Malicious SQL injection should not be possible.

Susceptible to injection: Should a malicious web user attempt to access this page they will be met with an error from the checklogin page prior to reaching this page therefore this page should be protected against malevolent use. Further to this the page leads only as a directory to other functions. The code does check if a user is an Admin granting them further functionality with CRUD functions.

Website Name: account.php (connects to) amenduser.php (connects to)
update_user.php

Screenshot:

```
<?
session_start();
include "../conninfo.php";

// LOGGED IN??

if ($_SESSION['Username'] == "") {
    echo "You are not logged in <br><br>";
    ?>
    <a href="../login.php"> Login Page </a>
    <?php
}

// FETCH USER DETAILS
$un = $_SESSION['Username'];
$query = "SELECT * FROM users WHERE Username = '$un'";
$result = mysqli_query($link, $query);
$row = mysqli_fetch_assoc($result);
?>

<!-- Update User Form -->
<h3> User Details </h3>

<?php if ($row['user_type'] == 'Admin') { ?>
```

Figure 18 - Check user is logged in to access the page and user type to limit admin privileges

```

<form action="update_user.php" method="POST">
    <label for="Username"> Username: </label>
    <input type="text" name="Username" value="<?php echo $row['Username']; ?>"> <br>

    <label for="password"> <br> Password: </label>
    <input type="text" name="Password">

    <label for="Email"> <br> Email: </label>
    <input type="text" name="Email" value="<?php echo $row['Email']; ?>">

    <input type="submit" name="update" value="Update">

    <input type="submit" name="delete" value="Delete Account">

    <input type="submit" name="Admin" value="Add Admin Privileges">

    <!-- Array ( [Username] => Tricell [Password] => [Email] => admin@no.com [admin] => Add Admin Privileges ) -->

</form>
<!-- add admin -->

else { ?>

    <form action="update_user.php" method="POST">
        <label for="Username"> Username: </label>
        <input type="text" name="Username" value="<?php echo $row['Username']; ?>" readonly><br>

        <label for="password"> <br> Password: </label>
        <input type="text" name="Password">

        <label for="Email"> <br> Email: </label>
        <input type="text" name="Email" value="<?php echo $row['Email']; ?>">

        <input type="submit" name="update" value="Update">

    </form>

```

Figure 19 - User form provided - inputs user details as read only for standard users

A screenshot of a web form for updating user information. The form has a dark red background with rounded corners and a thin blue border. It contains three input fields and one button. The first field is labeled 'Username:' and contains the text 'test123'. The second field is labeled 'Password:' and is empty. The third field is labeled 'Email:' and contains the text 'test@email.com'. Below these fields is an orange button with the text 'Update'.

Figure 20 - Basic user view - amenduser.php

```
Fatal error: Uncaught mysqli_sql_exception: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'UNION SELECT null, database()' at line 3 in C:\xampp\htdocs\assignment\crud.operations\update_user.php:45 Stack trace: #0 C:\xampp\htdocs\assignment\crud.operations\update_user.php(45): mysqli_query(Object(mysqli), 'UPDATE users SE...') #1 {main} thrown in C:\xampp\htdocs\assignment\crud.operations\update_user.php on line 45
```

Figure 21 - error from injection attempt

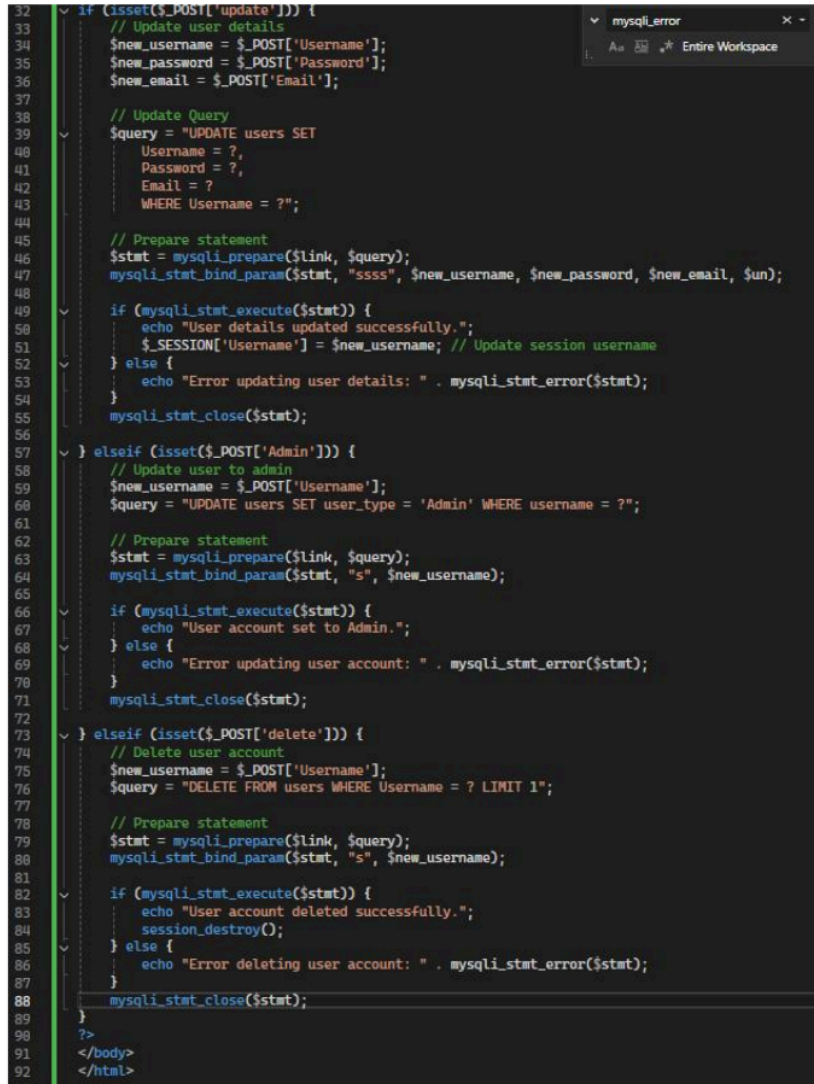
SQL injection query:

A screenshot of a web form for updating user information. The form has a dark red background with rounded corners and a thin blue border. It contains one input field labeled 'Password:'. The field contains the text `' UNION SELECT null, database()`.

Figure 22 - Injection attempt

Susceptible to injection:

The error presented indicates that the user input directly embeds into the SQL query without validation or cleaning of the statement. Allowing an attacker to inject malicious SQL syntax. To resolve this the variables within the query, need to be prepared as follows:



```
32 if (isset($_POST['update'])) {
33     // Update user details
34     $new_username = $_POST['Username'];
35     $new_password = $_POST['Password'];
36     $new_email = $_POST['Email'];
37
38     // Update Query
39     $query = "UPDATE users SET
40         Username = ?,
41         Password = ?,
42         Email = ?
43         WHERE Username = ?";
44
45     // Prepare statement
46     $stmt = mysqli_prepare($link, $query);
47     mysqli_stmt_bind_param($stmt, "ssss", $new_username, $new_password, $new_email, $new_email);
48
49     if (mysqli_stmt_execute($stmt)) {
50         echo "User details updated successfully.";
51         $_SESSION['Username'] = $new_username; // Update session username
52     } else {
53         echo "Error updating user details: " . mysqli_stmt_error($stmt);
54     }
55     mysqli_stmt_close($stmt);
56 }
57 elseif (isset($_POST['Admin'])) {
58     // Update user to admin
59     $new_username = $_POST['Username'];
60     $query = "UPDATE users SET user_type = 'Admin' WHERE username = ?";
61
62     // Prepare statement
63     $stmt = mysqli_prepare($link, $query);
64     mysqli_stmt_bind_param($stmt, "s", $new_username);
65
66     if (mysqli_stmt_execute($stmt)) {
67         echo "User account set to Admin.";
68     } else {
69         echo "Error updating user account: " . mysqli_stmt_error($stmt);
70     }
71     mysqli_stmt_close($stmt);
72 }
73 elseif (isset($_POST['delete'])) {
74     // Delete user account
75     $new_username = $_POST['Username'];
76     $query = "DELETE FROM users WHERE Username = ? LIMIT 1";
77
78     // Prepare statement
79     $stmt = mysqli_prepare($link, $query);
80     mysqli_stmt_bind_param($stmt, "s", $new_username);
81
82     if (mysqli_stmt_execute($stmt)) {
83         echo "User account deleted successfully.";
84         session_destroy();
85     } else {
86         echo "Error deleting user account: " . mysqli_stmt_error($stmt);
87     }
88     mysqli_stmt_close($stmt);
89 }
90 ?>
91 </body>
92 </html>
```

Figure 23 - Amended SQL Prepared statements

Website Name: createquest.php (connects to) savequest.php

Screenshot:

```
<form action="savequest.php" method="POST" enctype="multipart/form-data">

<label for="Title"> Title: </label>
<input type="text" name="Title" required >

<label for="Summary"> Summary: </label>
<input type="text" name="Summary" required >

<label for="Details"> Detail: </label>
<input type="text" name="Details">

<label>Select a difficulty level:</label>
<div class="radio-group">
  <input type="radio" id="level" name="difficulty" value="Easy">
  <label for "level"> Easy </label>
  <input type="radio" id="level" name="difficulty" value="Medium">
  <label for "level"> Medium </label>
  <input type="radio" id="level" name="difficulty" value="Hard">
  <label for "level"> Hard </label>
  <br><br>
</div>

<label for="Rewards"> Rewards: </label>
<input type="text" name="Rewards" required >

<label for="category"> Select a category: </label>
<select name="cat" id="category">

  <?
  $query = "SELECT Name FROM categories ORDER BY categoryID";
  $result=mysqli_query($link,$query);

  # while loop for dropdown
  while ($r=mysqli_fetch_array($result))
  {
    echo '<option value="' . $r["Name"] . '>' . ($r["Name"]) . '</option>';
  }
  ?>

Upload Image: <input type="file" Name="imgfile"> <br>

<input type="submit" value="Submit">
```

Figure 24 - createquest.php user input form

```
// Check if the user is logged in
if (empty($_SESSION['Username'])) {
  echo "You are not logged in. <a href='../login.php'>Login Page</a>";
  exit();
}
```

Figure 25 - savequest.php checks that the user is logged in before running the form

```

58
59 // Get the CategoryID based on the category name
60 $categoryQuery = "SELECT CategoryID FROM categories WHERE Name = ?";
61 $stmt = mysqli_prepare($link, $categoryQuery);
62 if (!$stmt) {
63     die("SQL preparation error: " . mysqli_error($link));
64 }
65
66 mysqli_stmt_bind_param($stmt, "s", $category);
67 mysqli_stmt_execute($stmt);
68 $result = mysqli_stmt_get_result($stmt);
69 $categoryData = mysqli_fetch_assoc($result);
70
71 if (!$categoryData) {
72     die("Category not found.");
73 }
74 $categoryID = $categoryData['CategoryID'];
75 mysqli_stmt_close($stmt);
76
77 // Insert the new quest into the database
78 $insertQuery = "INSERT INTO Quests (Title, Summary, details, Difficulty, Rewards, CategoryID, image)
79                 VALUES (?, ?, ?, ?, ?, ?, ?)";
80 $stmt = mysqli_prepare($link, $insertQuery);
81 if (!$stmt) {
82     die("SQL preparation error: " . mysqli_error($link));
83 }
84
85 mysqli_stmt_bind_param($stmt, "sssssis", $title, $summary, $details, $difficulty, $rewards, $categoryID, $imagePath);
86 if (mysqli_stmt_execute($stmt)) {
87     echo "Quest created successfully!";
88 } else {
89     echo "Error creating quest: " . mysqli_error($link);
90 }
91
92 mysqli_stmt_close($stmt);
93 mysqli_close($link);
94
95 // Redirect to quest management or display success message
96 echo "<br><a href='../account.php'>Return to account Management</a>";
97 exit();
98 }
99
100 // Redirect if accessed directly without submitting the form
101 header('Location: createquest.php');
102 exit();
103 ?>
104 </body>
105 </html>

```

Figure 26 - The form data is inserted into the database using a prepared statement preventing SQL injection

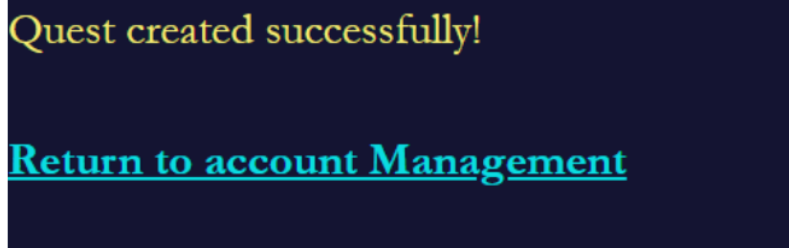


Figure 27 - Results from injection attempt

SQL injection query:

Create your adventure

Title:

' UNION SELECT table_name, null FROM information_schema.tables --

Summary:

Figure 28 - Exploit information_schema tables attempt.

Susceptible to injection:

Whilst the injection attempt has not been successful further validation of the fields to limit input would benefit the database from unnecessary records being added.

Website Name: managequests.php (connects to) editquest.php

Screenshots:

```
// Check if logged in
if ($_SESSION['Username'] == '') {
    echo "You are not logged in <br> <br>" ;
    ?>
    <a href= "../login.php"> Login Page </a>
    <?>
}

// pull quest id from previous page.
if (empty($_GET['quest_id']) || !ctype_digit($_GET['quest_id'])) {
    die("Invalid quest ID.");
}

$quest_id = intval($_GET['quest_id']);
$user_id = $_SESSION['UserID'];
```

Figure 29 - editquest.php retrieves the quest_id from the previous page.

```
20 // Fetch Quest Details
21 $query =
22 "SELECT * FROM Quests
23 JOIN user_quests ON Quests.QuestID = user_quests.QuestID
24 JOIN categories ON Quests.CategoryID = Categories.CategoryID
25 WHERE user_quests.UserID = ?";
26
27 $stmt = mysqli_stmt_init($link);
28 mysqli_stmt_prepare($stmt, $query);
29 mysqli_stmt_bind_param($stmt, "s", $user_id);
30 mysqli_stmt_execute($stmt);
31
32 $result = mysqli_stmt_get_result($stmt);
33 if ($result->num_rows == 0) {
34     die("You do not have permission to edit this quest!");
35 }
36
37 if (!$result) {
38     die("Database Query Error: " . mysqli_error($link));
39 }
40
41 $quest = mysqli_fetch_assoc($result);
42 if (!$quest) {
43     die("Quest not found: ");
44 }
45
46 // Checks Request Method is POST to run if block... escaping SQL Injection
47 if ($_SERVER['REQUEST_METHOD'] == "POST") {
48     $title = mysqli_real_escape_string($link, $_POST['Title']);
49     $summary = mysqli_real_escape_string($link, $_POST['Summary']);
50     $details = mysqli_real_escape_string($link, $_POST['Details']);
51     $difficulty = mysqli_real_escape_string($link, $_POST['Difficulty']);
52     $rewards = mysqli_real_escape_string($link, $_POST['Rewards']);
53
54     // convert Cat into int
55     $category_id = intval($_POST['CategoryID']);
56
57     $update_quest_query = "UPDATE Quests SET
58         Title = '$title',
59         Summary = '$summary',
60         Details = '$details',
61         Difficulty = '$difficulty',
62         Rewards = '$rewards',
63         CategoryID = '$category_id'
64         WHERE QuestID = '$quest_id'";
65
66     $update_quest_result = mysqli_query($link, $update_quest_query);
67
68     if ($update_quest_result) {
69         echo "<p> Quest updated successfully! </p>";
70         echo "<a href = \"managequests.php\">Back to Manage Quests </a>";
71     }
72     else {
73         echo "<p> Error updating quest: " . mysqli_error($link) . "</p>";
74     }
75 }
76 ?>
```

Figure 30 - editquest.php statements prepared user_id to ensure correct user is editing form.

Quest updated successfully!

Figure 31 - Results from SQL Injection attempt

SQL injection query:

The screenshot shows a web interface titled "Edit Quest Details". It contains two input fields: "Title:" and "Summary:". The "Title:" field contains the text "' OR 1=1 --", which is a common SQL injection payload. The "Summary:" field is empty.

Figure 32 - SQL injection attempt

Susceptible to injection: The query within editquest.php has not been victim to the SQL injection however the quest title has updated within our database to the attempt phrase:

Title	Summary	Details	Difficulty	Rewards	Category	Actions
' OR 1=1 --	Win the grand tournament in the colosseum. Battle against skilled gladiators to earn the champion's trophy and gain fame.	Step into the grand coliseum and prove your mettle in the Arena Challenge, a gladiatorial contest of strength, skill, and endurance. Compete against a series of formidable opponents, each more powerful than the last, in front of a roaring crowd. The battles range from one-on-one duels to chaotic free-for-alls. Victory requires mastery of your abilities and keen tactical thinking to outwit your adversaries. The final round pits you against a mysterious champion whose prowess in combat is unparalleled. Emerging victorious not only crowns you as the arena's champion but also rewards you with the Champion's Trophy, a symbol of your unmatched prowess.	Medium	0	Combat	Edit

Figure 33 - Editquest.php following query submission. Infiltration failed however the fields within the record have changed to the implemented query code.

Further validation rules could be set to limit special characters from being included within the database fields.

Edit Quest Details

Title:

' OR SLEEP(5)

Summary:

Set sail to a mysterious island rumored to be home to hidden treasures and ancient secrets. Overcome the island's challenge.

Figure 34 - Further test following amendments

Title:

' OR SLEEP(5)

Summary:

Set sail to a mysterious island rumored to be home to hidden treasures and ancient secrets. Overcome the island's challenge.

Please match the requested format.
Title is required and must not exceed 20 characters or hold any special characters.

Website Name: Erasequests.php (connects to) Updatequests.php

Screenshots:

```
<?
session_start();
include "../conninfo.php";

// Logged in?
if ($_SESSION['Username'] == "") {
    echo "You are not logged in <br> <br>" ;
    ?>
    <a href="../login.php"> Login Page </a>
    <?
}
```

Figure 35 - erasequests.php passes the logged in user details in the session to confirm if the user is logged in. This statement output of "Username" was cleansed at the point of userinput on the login.php page.

```
if ($row['user_type'] == 'Admin') {
```

Figure 36 - from the same login.php input the script will validate if the user has admin user_type to grant access to additional permissions

```
// Retrieve all quests
$query =
"SELECT Quests.QuestID,
    Quests.Title,
    Quests.Summary,
    Quests.details,
    Quests.Difficulty,
    Quests.Rewards,
    Categories.Name
FROM Quests
JOIN categories ON Quests.CategoryID = Categories.CategoryID";

$result = mysqli_query($link, $query);
if (!$result) {
    die ("Database Query Error: " . mysqli_error($link));
}??>
```

Figure 37 - erasequests.php will provide a full list of the quests within the database having validated Admin user type

```

<a href="updatequest.php?quest_id=<? echo $quest['QuestID'];?>">
<button>Edit</button></a><br>

<button onclick="showDeleteModal('<?php echo $quest['QuestID'];
?>', '<?php echo addslashes($quest['Title']); ?>')">Delete</button>

```

Figure 38 - The user is provided with two buttons, one which deletes the quest the other allows them to update the database.

```

17
18 // Fetch Quest Details
19 $query = "SELECT * FROM Quests
20         JOIN user_requests ON quests.QuestID = user_requests.QuestID
21         JOIN categories ON quests.CategoryID = categories.CategoryID
22         WHERE quests.QuestID = ?";
23 $stmt = mysqli_prepare($link, $query);
24 mysqli_stmt_bind_param($stmt, 'i', $quest_id);
25 mysqli_stmt_execute($stmt);
26 $result = mysqli_stmt_get_result($stmt);
27
28 if (!$result) {
29     die("Database Query Error: " . mysqli_error($link));
30 }
31
32 $quest = mysqli_fetch_assoc($result);
33
34 if (!$quest) {
35     die("Quest not found.");
36 }
37

```

Figure 39 - Updatequest.php pulls the quest details from the selected quest on the previous page and checks if the quest is valid having used a prepared statement to ensure the quest id is correct.

```

48 // Update Quest
49 $update_quest_query = "UPDATE quests SET
50     Title = ?,
51     Summary = ?,
52     Details = ?,
53     Difficulty = ?,
54     Rewards = ?,
55     CategoryID = ?
56     WHERE QuestID = ?";
57 $stmt = mysqli_stmt_init($link);
58 mysqli_stmt_prepare($stmt, $update_quest_query);
59 mysqli_stmt_bind_param($stmt, "sssssi", $title, $summary, $details, $difficulty, $rewards, $category_id, $quest_id);
60
61 if (mysqli_stmt_execute($stmt)) {
62     echo "<p>Quest updated successfully!</p>";
63     echo '<a href="erasequests.php">Back to Manage Quests</a>';
64 } else {
65     echo "<p>Error updating quest: " . mysqli_stmt_error($stmt) . "</p>";
66 }
67 mysqli_stmt_close($stmt);
68
69

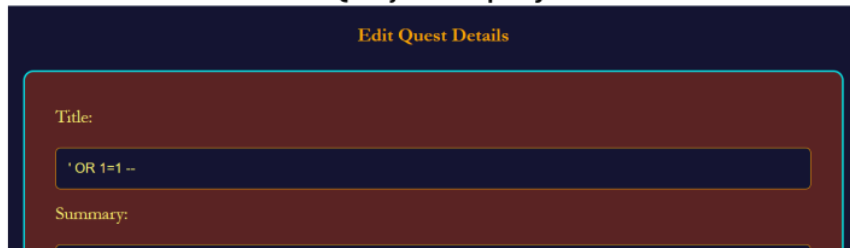
```

Figure 40 - To update the record the Update query is passed through a prepared statement detailing the type of datatype before confirming the request has been successful or not.

Title	Summary	Details	Difficulty	Rewards	Category	Actions
OR 1-1	Slay the dragon that has been ravaging the kingdom. Face the dragon in a fierce battle, utilizing your combat skills to claim dragon scale armor.	Embark on the ultimate test of courage and skill—slaying the dragon that has been ravaging the kingdom. This fearsome creature resides in a volcanic lair filled with rivers of molten lava and clouds of toxic fumes. The path to the dragon's den is fraught with danger, from fire-breathing minions to collapsing volcanic tunnels. The dragon itself is a majestic yet terrifying adversary, with scales as hard as steel and flames capable of incinerating anything in their path. To defeat it, you'll need to strategize, utilizing both your combat skills and the environment to your advantage. Successfully completing this quest earns you the coveted Dragon Scale Armor and 500 XP, marking you as a true hero of legend.	Hard	500 Gold Pieces and exclusive Dragon Gloves	Combat	Edit Delete

Figure 41 - Result following attempt to inject using the edit form. There are visible issues with the input here caused by insufficient validation. Restriction of symbols would benefit the records from being updated with erroneous records.

SQL injection query:



The screenshot shows a web form titled "Edit Quest Details". It has two main sections: "Title:" and "Summary:". The "Title:" section contains a text input field with the value "' OR 1=1 --". The "Summary:" section is empty.

Figure 42 - SQL injection on Admin Edit form

Susceptible to injection:

As previously mentioned, the SQL injection is unsuccessful; however, the title name has been changed to the phrase used by malicious users. Further validation rules could be set to prevent such inputs being added to the record.

```
// VALIDATIONS
$errors = [];
if (empty($title) || strlen($title) > 20 || !preg_match('/^[A-Za-z0-9 ]+$/ ', $title)) {
    $errors[] = "Title is required and must not exceed 20 characters or hold any special characters.";
}

if (empty($summary) || strlen($summary) > 255 || !preg_match('/^[A-Za-z0-9 ]+$/ ', $summary)) {
    $errors[] = "Summary is required and must not exceed 255 characters or hold any special characters.";
}

if (!empty($details) && strlen($details) > 1000 || !preg_match('/^[A-Za-z0-9 ]+$/ ', $details)) {
    $errors[] = "Details must not exceed 1000 characters or hold any special characters.";
}

if (empty($rewards) || strlen($rewards) > 100 || !preg_match('/^[A-Za-z0-9 ]+$/ ', $rewards)) {
    $errors[] = "Rewards must not exceed 100 characters and can only contain letters, numbers, and spaces.";
}
```

Figure 43 - Example of additional validation rules included to prevent special characters from being included within the record.

Website Name: Erasequests.php (connects to) Removequests.php

Screenshot:

```
<form method="post" id="deleteForm" action="removequest.php">
    <input type="hidden" name="quest_id" id="questId">
    <button type="submit" name="confirm" value="true"> Confirm</button>
    <button type="button" class="close"> Cancel</button>
```

Figure 44 - input method for erasequests

```

5
6 // Logged in?
7 if ($_SESSION['Username'] == "") {
8     echo "You are not logged in <br> <br>" ;
9     ?>
10     <a href="../login.php"> Login Page </a>
11     <? }
12
13
14 // Fetch details from database
15
16     $un = $_SESSION['Username'];
17     $query = "SELECT * FROM users WHERE Username = '$un'";
18     $result = mysqli_query($link, $query);
19
20     // Check if the query was successful
21
22     if (!$result) {
23         echo "Database query failed: " . mysqli_error($link);
24         exit();
25     }
26
27     // Fetch user details
28
29     $row = mysqli_fetch_assoc($result);
30     if (!$row || $row['user_type'] !== 'Admin') {
31         echo "You do not have permission to access this page.";
32         exit();
33     }
34
35

```

Figure 45 - removequest.php validates the session user (as entered on the login.php page)

```

?>
if ($_SERVER['REQUEST_METHOD'] === 'POST' && isset($_POST['confirm'])) {

    $quest_id = $_POST['quest_id'];

    $delete_query = "DELETE FROM Quests WHERE QuestID = ?";
    $stmt = mysqli_prepare($link, $delete_query);

    if (!$stmt) {
        die("SQL preparation error: " . mysqli_error($link));
    }

    mysqli_stmt_bind_param($stmt, "i", $quest_id);
    if (mysqli_stmt_execute($stmt)) {
        echo "Quest deleted successfully. <br><br>";
        header('Location: erasequests.php');
    } else {
        echo "Error deleting quest: " . mysqli_error($link);
    }

    mysqli_stmt_close($stmt);
    exit;

    header('Location: erasequests.php');
    exit();

}

?>

```

Figure 46 - removequest.php query to delete questid

SQL injection query:

The request method confirms that a POST form has been submitted to access this page. The page then subsequently checks the quest_id received from the form and prepares the id prior to running the query. As there is no option for SQL Injection for this form to access this query other than via the Login.php which has been tested above.

Susceptible to injection:

The delete query bases the quest_id on the previously selected quest_id using POST. The use of mysqli_prepare and mysqli_stmt_bind_param protects against SQL injection for the deletion operation.

Website Name: register.php (connects to) saveaccount.php

Screenshots:

```
32 <form action="saveaccount.php" method="POST">
33
34 <label for="Username"> Username: </label>
35 <input type="text" name ="Username" required >
36 <label for="Password"> Password: </label>
37 <input type="text" name="Password" pattern="(?!.*\d)(?!.*[a-z])(?!.*[A-Z]).{8,}"
38 title="Must contain at least one number and one uppercase and lowercase letter, and at least 8 or more characters">
39 <label for="Email"> Email Address: </label>
40 <input type="text" name = "Email" required>
41
42 <input type = "submit" value ="Submit">
43
44 </form>
45
46 </body>
47 <html>
```

Figure 47 - register.php holds a form which leads the inputs to saveaccount.php

```
30 include "../conninfo.php";
31
32 // Function to handle form submission and database insertion
33 function handleFormSubmission($link) {
34     $requiredFields = ['Username', 'Password', 'Email'];
35     $formData = array_map(fn($field) => $_POST[$field] ?? null, $requiredFields);
36
37     // Check if all form fields are filled
38     if (in_array(null, $formData, true)) {
39         echo "Please fill in all fields.";
40         return;
41     }
42
43     [$Username, $Password, $Email] = $formData;
44     $query = "INSERT INTO users (Username, Password, Email) VALUES (?, ?, ?)";
45     $stmt = mysqli_stmt_init($link);
46
47     if (mysqli_stmt_prepare($stmt, $query)) {
48         mysqli_stmt_bind_param($stmt, "sss", $Username, $Password, $Email);
49         mysqli_stmt_execute($stmt);
50         echo "<strong>Record Added!</strong><br><br>";
51         echo "<a href='../login.php'>Login</a>";
52     } else {
53         echo "Failed to prepare the SQL statement.";
54     }
55 }
56
57 // Ensure the form has been submitted via POST method
58 if ($_SERVER["REQUEST_METHOD"] == "POST") {
59     handleFormSubmission($link);
60 } else {
61     echo "Form was not submitted.";
62 }
63 ?>
```

Figure 48 - Function and Query within saveaccount.php which prepares the statement from the previous page.

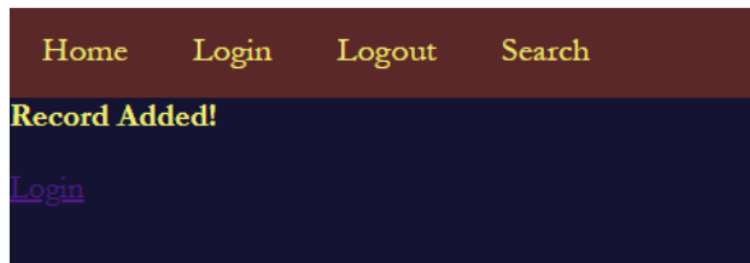


Figure 49 - Result of SQL Injection

☐	Edit	Copy	Delete	42 DBwizard	Password123 ' OR 1=1 --	User
--------------------------	------	------	--------	-------------	-------------------------	------

Figure 50 - phpmyadmin view of added record.

SQL injection query:

Register an Account

Username:

DBwizard

Password:

Password123

Email Address:

' OR 1=1 --

Submit

Figure 51 - SQL injection input

Register an Account

Username:

JapaneseWarrior

Password:

.....

Email Address:

' AND 1 = 1 --

Please include an '@' in the email address. " AND 1 = 1 --" is missing an '@'.

Submit

Figure 52 - Blind SQL injection attempt

Susceptible to injection: The query to insert users into the database uses prepared statements and `mysqli_stmt_bind_param` to mitigate SQL injection.

There is a potential vulnerability within the database as passwords are inserted directly without hashing. Using `password_hash` before storing them in the database would reduce the risk of data being exposed in plaintext.

Further validations for inputs would be beneficial to reduce errors on records such as invalid email addresses.

```

// Validation
$errors = [];

// Username
if (!preg_match('/^[A-Za-z0-9]{4,20}$/', $Username)) {
    $errors[] = "Username must be 5 - 20 characters long and contain only letter and numbers.";
}

// Password
if (!preg_match('/(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,}/', $Password)) {
    $errors[] = "Password must contain at least one number and one uppercase and lowercase letter, and at least 8 or more characters.";
}

//Email Address
if(!filter_var($Email, FILTER_VALIDATE_EMAIL)) {
    $errors[] = "Invalid email address.";
}

if (empty($errors)) {
    foreach ($errors as $error) {
        echo "<p style= 'color:red; '>$error</p>";
    }
    return;
}

```

Figure 53 - Example additional validations applied to the PHP side

Following a review of the web pages above it is clear that prepared statements are beneficial in the prevention of SQL Injection, however the use of Validation rules on the front and back end of forms to restrict records being amended with erroneous inputs.

Future improvements to be made to the script include:

Front end form improvement using the <email> and <password> tag, max and min length to stipulate the length of a use input along with set rules for password input such as Upper case, lowercase and a numeric value. Restricting the fields that allow special characters would likely remove the input of the SQL injection queries.

On the back-end validation and error reporting for the inputs which are not stopped by the front-end form. Using if statements to print the relevant error message to the front-end page.

References

- [1] M. E. Whitman and H. J. Mattord, *Principles of information security*, 6th ed. Boston, Mass.: Cengage Learning, 2022.
- [2] M. Horner and T. Hyslip, "SQL Injection: The Longest Running Sequel in Programming History," *The Journal of Digital Forensics, Security and Law*, 2017, doi: <https://doi.org/10.15394/jdfsl.2017.1475>.
- [3] "Introduction - OWASP Top 10:2021," [owasp.org](https://owasp.org/Top10/A00_2021_Introduction/). https://owasp.org/Top10/A00_2021_Introduction/
- [4] "Cisco Prime License Manager SQL Injection Vulnerability," *Cisco*, Feb. 2024. <https://www.cisco.com/c/en/us/support/docs/csa/cisco-sa-20181128-plm-sql-inject.html>
- [5] L. Conklin and G. Robinson, "CODE REVIEW GUIDE RELEASE Creative Commons (CC) Attribution." Available: https://owasp.org/www-project-code-review-guide/assets/OWASP_Code_Review_Guide_v2.pdf
- [6] "PHP: Documentation," *Php.net*, 2019. <https://www.php.net/docs.php>

Appendix

I acknowledge the use of ChatGPT (<https://chatgpt.com/>) on 16 December 2024 to support my research using the prompt:

- 1) Please provided me with materials on SQL Injection for research. Include academic sources from the past 3 years.
- 2) Please provided me with materials on high profile data breaches that were as a result of SQL Injection. Give details and sources to the information including what data was obtained

I confirm that I have not presented any AI generated text as my own work. I confirm I have copies of my drafts, notes, and other resources I used before, and after the use of AI tools which I may be asked to provide in evidence.

Image References

- [1] José García Lázaro, "900 year old jungle temple - Ta Promh, Cambodia," *Pinterest*, Aug. 13, 2013. <https://mx.pinterest.com/pin/40813940346315591/> (accessed Jan. 07, 2025).
- [2] Vie Studio, "Coming Soon on White Surface · Free Stock Photo," *Pexels*, Sep. 21, 2020. <https://www.pexels.com/photo/coming-soon-on-white-surface-4439444/> (accessed Jan. 07, 2025).
- [3] W. Alejandro, "Person Posing in Cave · Free Stock Photo," *Pexels*, Jun. 07, 2023. <https://www.pexels.com/photo/person-posing-in-cave-16653748/> (accessed Jan. 07, 2025).
- [4] "Mystery Skull Island Wallpapers | HD Wallpapers | ID #27263," *Hdwallpapers.in*, 2019. https://www.hdwallpapers.in/mystery_skull_island-wallpapers.html (accessed Jan. 07, 2025).
- [5] "Oasis Desert - 'what once was...' by LindezaBlue on DeviantArt," *Deviantart.com*, Jun. 13, 2023. <https://www.deviantart.com/lindezablue/art/Oasis-Desert-what-once-was-967034234> (accessed Jan. 07, 2025).
- [6] B. for, "Ilustración de Stock DnD Battlemat Battlemat for Frozen Tundra. | Adobe Stock," *Adobe Stock*, 2025. <https://stock.adobe.com/es/images/dnd-battlemat-battlemat-for-frozen-tundra/801223843> (accessed Jan. 07, 2025).
- [7] radiiadmin, "Abandoned bandit camp along the river - NPC Beta Abandoned Areas, On the road, River Areas, Side Quests NPC and map assets for homebrew RPG," *NPC Beta - Free to use NPC and map assets for your homebrew RPG gaming sessions*, Jun. 19, 2023. <https://npcbeta.com/abandoned-bandit-camp-along-the-river/> (accessed Jan. 07, 2025).
- [8] "dragon_hunt by bananasjuice on DeviantArt," *Deviantart.com*, Mar. 23, 2008. <https://www.deviantart.com/bananasjuice/art/dragon-hunt-80771431> (accessed Jan. 07, 2025).
- [9] Terk, "Orc knight , BoissB - Blanca.J," *Pinterest*, Mar. 16, 2019. <https://uk.pinterest.com/pin/824018063049760674/> (accessed Jan. 07, 2025).
- [10] "Old arena, Sergey Lameyko," *ArtStation*, 2025. <https://www.artstation.com/artwork/5EK2A> (accessed Jan. 07, 2025).
- [11] M. Gandhi, "White Wolves in a Forest · Free Stock Photo," *Pexels*, Feb. 20, 2024. <https://www.pexels.com/photo/white-wolves-in-a-forest-20260773/> (accessed Jan. 07, 2025).

- [12] Critical Role wiki, "Kamaljiori - Critical Role," *Critical Role wiki*, Oct. 19, 2024. <https://criticalrole.miraheze.org/wiki/Kamaljiori> (accessed Jan. 07, 2025).
- [13] "Labyrinth of Time Surreal Digital Scene | Premium AI-generated image," *Freepik*, 2024. https://www.freepik.com/premium-ai-image/labyrinth-time-surreal-digital-scene_184967375.htm (accessed Jan. 07, 2025).
- [14] S. King, "Magic Circle 5e D&D Guide," *Tabletop Den*, Mar. 09, 2023. <https://tabletopden.com/dnd/magic-circle-5e-guide/> (accessed Jan. 07, 2025).
- [15] "AIVIK Studio," *Artstation.com*, 2025. <https://www.artstation.com/artwork/8won8Q> (accessed Jan. 07, 2025).
- [16] A. Boyt, *Purple Forest*. 2025.
- [17] "10 HERBALISM SCHOOLS for the Herb Curious," *Anima Mundi Herbals*, 2025. <https://animamundiherbals.com/blogs/blog/9-herbalism-schools-for-the-herb-curious> (accessed Jan. 07, 2025).
- [18] "Explore the best of Pinterest," *Pinterest*, 2022. <https://uk.pinterest.com/ideas/>
- [19] "11 D&D Chef ideas | dnd characters, character art, character inspiration," *Pinterest*, Nov. 05, 2019. <https://uk.pinterest.com/ianrousculp/dd-chef/> (accessed Jan. 07, 2025).
- [20] "World's Most Haunted Forests," *Travel + Leisure*. <https://www.travelandleisure.com/trip-ideas/nature-travel/worlds-most-haunted-forests>
- [21] Z. Ebner, "merchant city art," *Pinterest*, Aug. 14, 2016. <https://uk.pinterest.com/pin/86905467788629950/> (accessed Jan. 07, 2025).
- [22] Ergtoth Grifton, "The Worst Kinds of Quests - SUPERJUMP - Medium," *Medium*, Sep. 14, 2020. <https://medium.com/super-jump/the-worst-kinds-of-quests-dc498d6c6de3> (accessed Jan. 07, 2025).
- [23] R. Lach, "A Kid in Yellow Jacket Sitting under a Tree in the Woods · Free Stock Photo," *Pexels*, Nov. 2021, doi: <https://doi.org/10366582/pexels-photo-10366582>.
- [24] R. Mitchell, "Halls of Ancient Wisdom: 7 Remarkable Ancient Libraries," *Ancient Origins Reconstructing the story of humanity's past*, Jul. 16, 2023. <https://www.ancient-origins.net/history-important-events/ancient-libraries-0018859> (accessed Jan. 07, 2025).
- [25] Crate Entertainment, "Ultimate Hardcore Deceiver- Plague Doctor (Support Healer)," *Crate Entertainment Forum*, Nov. 24, 2022. <https://forums.crateentertainment.com/t/ultimate-hardcore-deceiver-plague-doctor-support-healer/123628> (accessed Jan. 07, 2025).

[26] "A beautiful mysterious island hidden from the world | Premium AI-generated image," *Freepik*, 2023. https://www.freepik.com/premium-ai-image/beautiful-mysterious-island-hidden-from-world_37677532.htm (accessed Jan. 07, 2025).

[27] "The Lord of the Rings LCG : Bifur," *Blogspot.com*, 2025. <https://magali-villeneuve.blogspot.com/2011/12/lord-of-rings-lcg-bifur.html> (accessed Jan. 07, 2025).

FINAL GRADE

84/100

GENERAL COMMENTS

Database Structure and Design - Good - lots of tables appropriately normalise. You might consider rather than repeating "Pending", you could use 1 2 3 etc for states and then just sort out the text for the front end display in code - this could mean fewer instances of repeating text in your database. 9/10

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.

Be careful that you select a radio button my default or an error is shows.

Search results are good with a following page also.

Be careful to validate the input - by not entering anything in the search bar, I was able to see the whole database - probably not what you want, and as your site grows the results page would be huge!

Login is good - it would be better to filter "Login" out of the top bar once you're logged in to make it more user friendly.

Add is good with some validation - I would make image non-essential though as this throws an error if you don't put one in. Appreciate the use of the populated pull down menu

Good delete with confirmation.

Login is well managed with URL attacks accounted for. Well done.

A good range shown overall, just a bit more attention to detail with the validation/search 40/50

Overall integration of website - Quite well laid out but the yellow on maroon is a little hard on the eyes! I lost my way a bit on the admin side of things, make sure you have a link back to the admin home from all pages.

7/10

SQL Injection overview - Good summary and the example are clearly shown. Happy with this. Could have mentioned the danger of single quote and why 13/15

SQL Code implementation/proposal - Very thorough. Well done. 15/15

Overall mark for PHP assignment: 84%

Overall mark for module (including Divya's assignment) - 88.5%

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

