CodeAlpha_Secure_Coding_Review

Name: Ahmed Elgendy

Student ID: CA/CC/12774

Secure_Coding_Review for ReciPHP Web-Application

Introduction

This document provides a comprehensive review of the secure coding practices implemented in an application chosen for an intern task at CodeAlpha. The task involved selecting an open-source application and conducting a manual security review process. The primary objective of this review is to identify security vulnerabilities within the codebase and offer recommendations for enhancing secure coding practices.

Task Requirements:

- Choose an open-source application to review.
- Conduct a thorough review of the codebase to identify security vulnerabilities.
- Provide detailed recommendations for implementing secure coding practices.
- Utilize manual code reviews to assess the codebase.

Application Description:

The application developed for this task is a web-based recipe management system known as "ReciPHP." It enables users to perform various actions such as adding, searching, and viewing recipes, as well as posting comments on recipes. The application is primarily written in PHP and interacts with a MySQL database for data storage and retrieval.

Security Review:

The security review process involved a meticulous examination of the codebase using manual code review techniques. Identified vulnerabilities were categorized, documented, and analyzed to provide actionable recommendations for implementing robust secure coding practices.

Methodology

Selection of ReciPHP:

The ReciPHP application was chosen for the secure coding review based on its availability as an open-source project and its suitability for assessing secure coding practices. ReciPHP is a web-based recipe management system developed in PHP and MySQL, making it a relevant choice for evaluating PHP application security.

Tool Selection:

To begin the security review process, the SourceForge website was utilized to download the ReciPHP application. SourceForge provides a platform for hosting open-source software projects, making it a reliable source for obtaining software for review.

Identification of Vulnerabilities:

The security review process involved the identification of potential vulnerabilities within the ReciPHP codebase. This was accomplished using various techniques, including:

- Utilizing the `grep` command to search for specific functions in PHP code that are commonly associated with security vulnerabilities, such as SQL injection (SQLi), cross-site scripting (XSS), and file inclusion.
- Conducting manual code reviews to inspect the files where potential vulnerabilities were identified through `grep` searches.
- Analyzing the sources (user inputs) and sinks (functions) within the codebase to validate if the identified vulnerabilities were indeed exploitable.

Remediation Process:

Upon identifying vulnerabilities, the remediation process involved:

- Creating new files with names prefixed by "new_" to indicate that they contain the fixed versions of the vulnerable files.
- Implementing appropriate fixes to address the identified vulnerabilities, such as using parameterized queries to prevent SQL injection, sanitizing user input to mitigate cross-site scripting, and securing sensitive information to prevent data leaks.
- Providing detailed recommendations for implementing secure coding practices to prevent similar vulnerabilities in the future.

Validation of Fixes:

After implementing the fixes, thorough testing and validation were performed to ensure that the remediation measures effectively addressed the identified vulnerabilities without introducing new security risks. This validation process involved testing the application under various scenarios to verify its resilience to common security threats.

Documentation:

Comprehensive documentation was created to record the identified vulnerabilities, the steps taken to remediate them, and the recommendations provided for implementing secure coding practices. This documentation serves as a valuable resource for developers, security professionals, and stakeholders involved in the development and maintenance of the ReciPHP application.

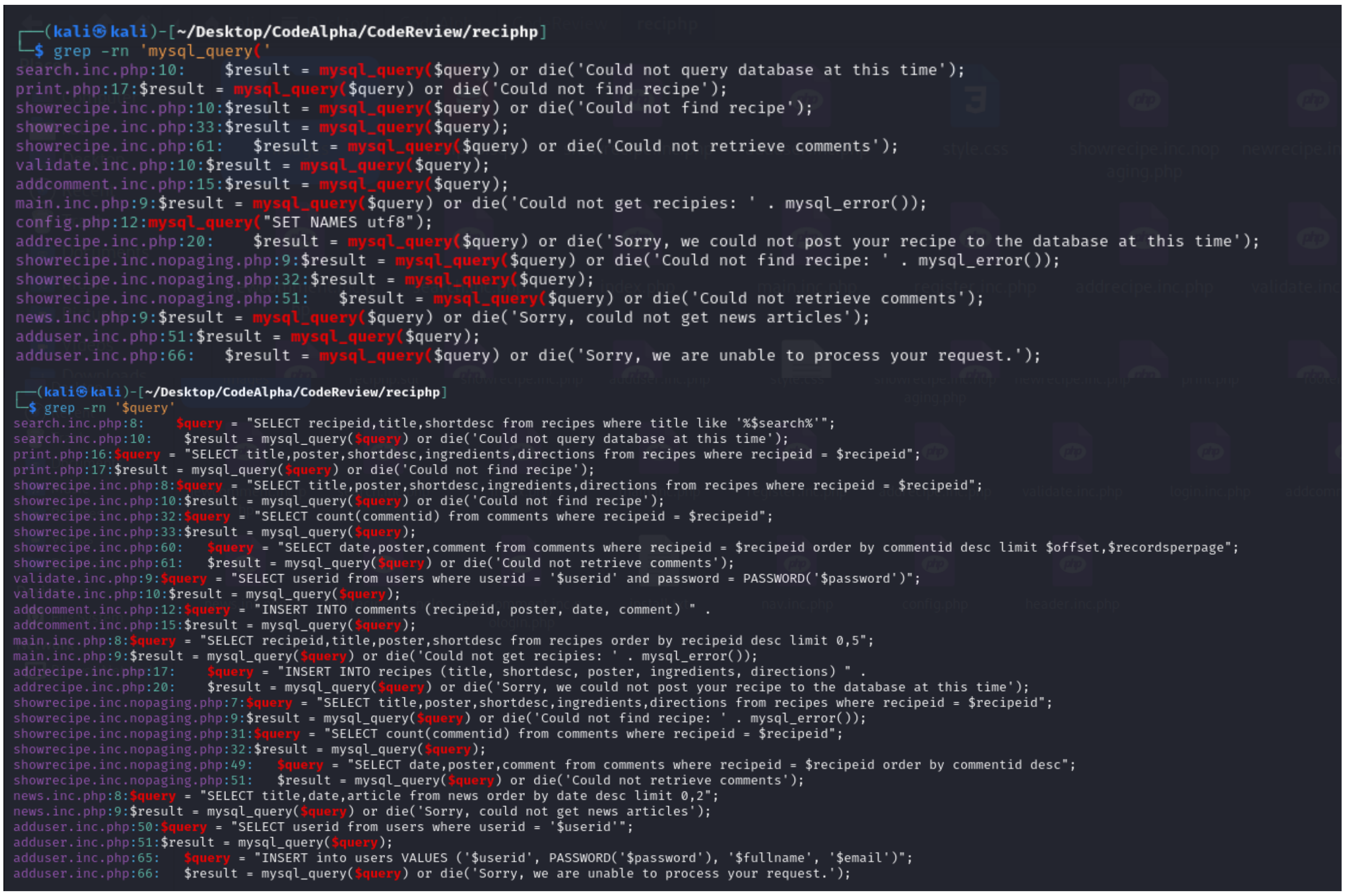
Identifying Vulnerabilities using grep

Vulnerabilities found in the source code:

SQL Injection (SQLi):

Description: SQL injection is a type of security vulnerability that occurs when an attacker is able to manipulate SQL queries executed by the application's database. This allows the attacker to modify the SQL queries in such a way that they can retrieve, modify, or delete data from the database, or even execute arbitrary SQL commands.

searching for SQLi:

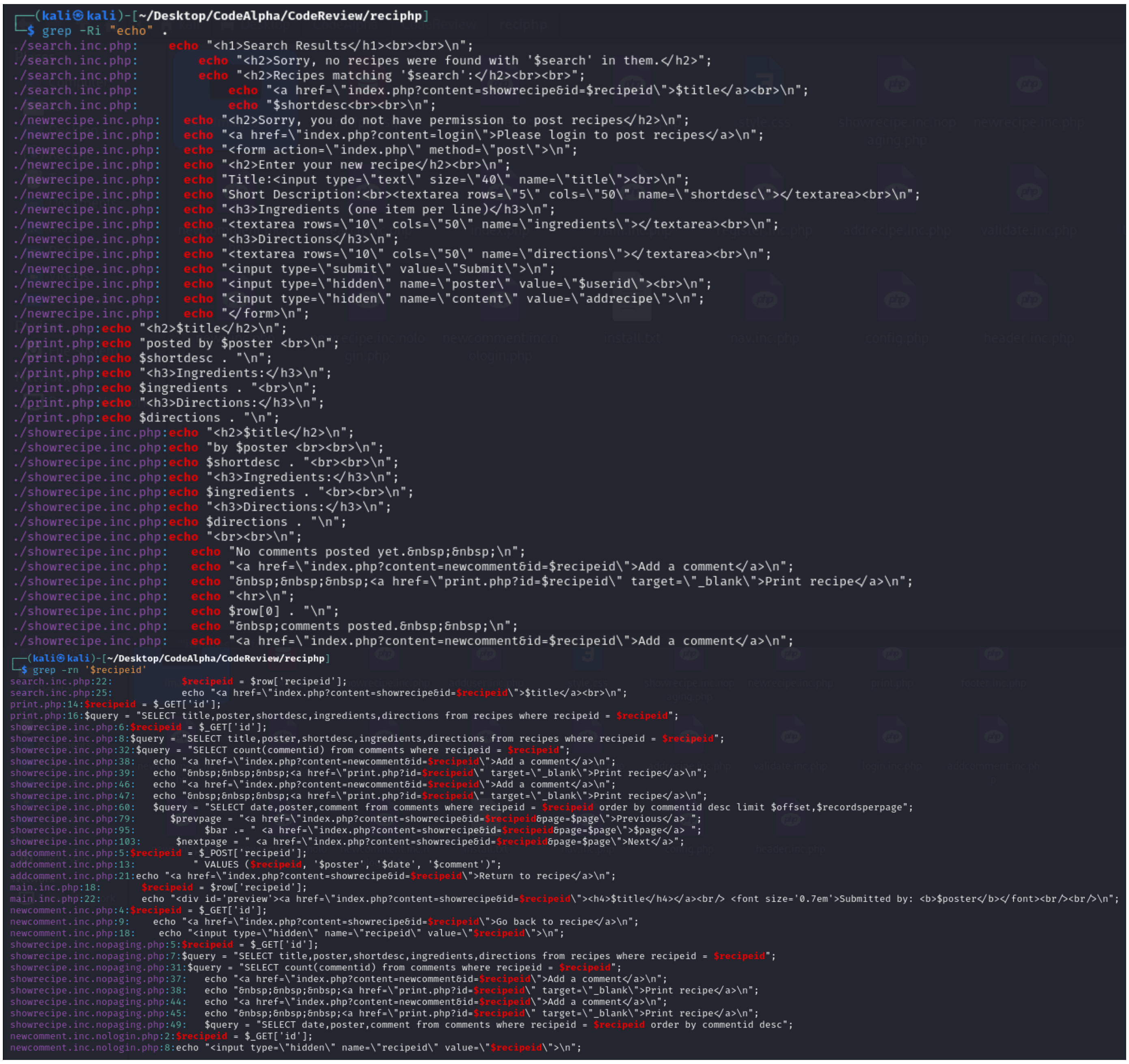


```
┌──(kali㉿kali)-[~/Desktop/CodeAlpha/CodeReview/reciphp]
└─$ grep -rn 'mysql_query('
search.inc.php:10:    $result = mysql_query($query) or die('Could not query database at this time');
print.php:17:$result = mysql_query($query) or die('Could not find recipe');
showrecipe.inc.php:10:$result = mysql_query($query) or die('Could not find recipe');
showrecipe.inc.php:33:$result = mysql_query($query);
showrecipe.inc.php:61:    $result = mysql_query($query) or die('Could not retrieve comments');
validate.inc.php:10:$result = mysql_query($query);
addcomment.inc.php:15:$result = mysql_query($query);
main.inc.php:9:$result = mysql_query($query) or die('Could not get recipies: ' . mysql_error());
config.php:12:mysql_query("SET NAMES utf8");
addrecipe.inc.php:20:    $result = mysql_query($query) or die('Sorry, we could not post your recipe to the database at this time');
showrecipe.inc.nopaging.php:9:$result = mysql_query($query) or die('Could not find recipe: ' . mysql_error());
showrecipe.inc.nopaging.php:32:$result = mysql_query($query);
showrecipe.inc.nopaging.php:51:    $result = mysql_query($query) or die('Could not retrieve comments');
news.inc.php:9:$result = mysql_query($query) or die('Sorry, could not get news articles');
adduser.inc.php:51:$result = mysql_query($query);
adduser.inc.php:66:    $result = mysql_query($query) or die('Sorry, we are unable to process your request.');

┌──(kali㉿kali)-[~/Desktop/CodeAlpha/CodeReview/reciphp]
└─$ grep -rn '$query'
search.inc.php:8:    $query = "SELECT recipeid,title,shortdesc from recipes where title like '%$search%'";
search.inc.php:10:    $result = mysql_query($query) or die('Could not query database at this time');
print.php:16:$query = "SELECT title,poster,shortdesc,ingredients,directions from recipes where recipeid = $recipeid";
print.php:17:$result = mysql_query($query) or die('Could not find recipe');
showrecipe.inc.php:8:$query = "SELECT title,poster,shortdesc,ingredients,directions from recipes where recipeid = $recipeid";
showrecipe.inc.php:10:$result = mysql_query($query) or die('Could not find recipe');
showrecipe.inc.php:32:$query = "SELECT count(commentid) from comments where recipeid = $recipeid";
showrecipe.inc.php:33:$result = mysql_query($query);
showrecipe.inc.php:60:    $query = "SELECT date,poster,comment from comments where recipeid = $recipeid order by commentid desc limit $offset,$recordsperpage";
showrecipe.inc.php:61:    $result = mysql_query($query) or die('Could not retrieve comments');
validate.inc.php:9:$query = "SELECT userid from users where userid = '$userid' and password = PASSWORD('$password')";
validate.inc.php:10:$result = mysql_query($query);
addcomment.inc.php:12:$query = "INSERT INTO comments (recipeid, poster, date, comment) " .
addcomment.inc.php:15:$result = mysql_query($query);
main.inc.php:8:$query = "SELECT recipeid,title,poster,shortdesc from recipes order by recipeid desc limit 0,5";
main.inc.php:9:$result = mysql_query($query) or die('Could not get recipies: ' . mysql_error());
addrecipe.inc.php:17:    $query = "INSERT INTO recipes (title, shortdesc, poster, ingredients, directions) " .
addrecipe.inc.php:20:    $result = mysql_query($query) or die('Sorry, we could not post your recipe to the database at this time');
showrecipe.inc.nopaging.php:7:$query = "SELECT title,poster,shortdesc,ingredients,directions from recipes where recipeid = $recipeid";
showrecipe.inc.nopaging.php:9:$result = mysql_query($query) or die('Could not find recipe: ' . mysql_error());
showrecipe.inc.nopaging.php:31:$query = "SELECT count(commentid) from comments where recipeid = $recipeid";
showrecipe.inc.nopaging.php:32:$result = mysql_query($query);
showrecipe.inc.nopaging.php:49:    $query = "SELECT date,poster,comment from comments where recipeid = $recipeid order by commentid desc";
showrecipe.inc.nopaging.php:51:    $result = mysql_query($query) or die('Could not retrieve comments');
news.inc.php:8:$query = "SELECT title,date,article from news order by date desc limit 0,2";
news.inc.php:9:$result = mysql_query($query) or die('Sorry, could not get news articles');
adduser.inc.php:50:$query = "SELECT userid from users where userid = '$userid'";
adduser.inc.php:51:$result = mysql_query($query);
adduser.inc.php:65:    $query = "INSERT into users VALUES ('$userid', PASSWORD('$password'), '$fullname', '$email')";
adduser.inc.php:66:    $result = mysql_query($query) or die('Sorry, we are unable to process your request.');
```

XSS (Cross-Site Scripting):

Description: Cross-Site Scripting (XSS) is a security vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users. This can lead to various forms of attack, such as stealing session cookies, redirecting users to malicious websites, or defacing web pages.

searching for XSS:

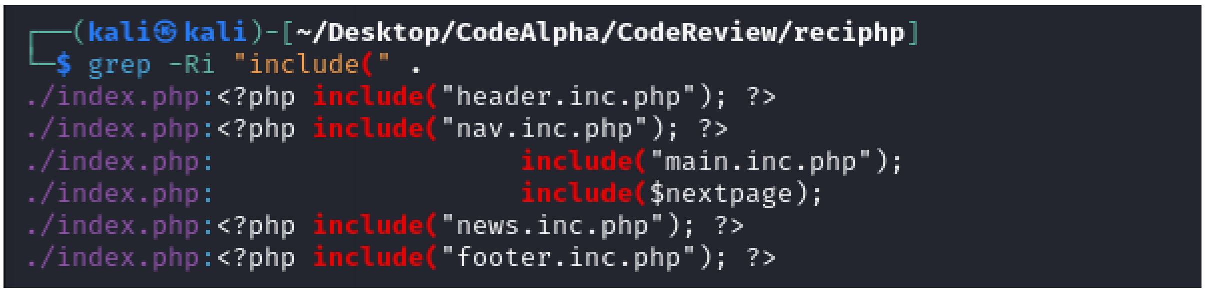


```
┌──(kali㉿kali)-[~/Desktop/CodeAlpha/CodeReview/reciphp]
└─$ grep -Ri "echo" .
./search.inc.php:    echo "<h1>Search Results</h1><br><br>\n";
./search.inc.php:        echo "<h2>Sorry, no recipes were found with '$search' in them.</h2>";
./search.inc.php:        echo "<h2>Recipes matching '$search':</h2><br><br>";
./search.inc.php:            echo "<a href=\"index.php?content=showrecipe&id=$recipeid\">$title</a><br>\n";
./search.inc.php:            echo "$shortdesc<br><br>\n";
./newrecipe.inc.php:   echo "<h2>Sorry, you do not have permission to post recipes</h2>\n";
./newrecipe.inc.php:   echo "<a href=\"index.php?content=login\">Please login to post recipes</a>\n";
./newrecipe.inc.php:   echo "<form action=\"index.php\" method=\"post\">\n";
./newrecipe.inc.php:   echo "<h2>Enter your new recipe</h2><br>\n";
./newrecipe.inc.php:   echo "Title:<input type=\"text\" size=\"40\" name=\"title\"><br>\n";
./newrecipe.inc.php:   echo "Short Description:<br><textarea rows=\"5\" cols=\"50\" name=\"shortdesc\"></textarea><br>\n";
./newrecipe.inc.php:   echo "<h3>Ingredients (one item per line)</h3>\n";
./newrecipe.inc.php:   echo "<textarea rows=\"10\" cols=\"50\" name=\"ingredients\"></textarea><br>\n";
./newrecipe.inc.php:   echo "<h3>Directions</h3>\n";
./newrecipe.inc.php:   echo "<textarea rows=\"10\" cols=\"50\" name=\"directions\"></textarea><br>\n";
./newrecipe.inc.php:   echo "<input type=\"submit\" value=\"Submit\">\n";
./newrecipe.inc.php:   echo "<input type=\"hidden\" name=\"poster\" value=\"$userid\"><br>\n";
./newrecipe.inc.php:   echo "<input type=\"hidden\" name=\"content\" value=\"addrecipe\">\n";
./newrecipe.inc.php:   echo "</form>\n";
./print.php:echo "<h2>$title</h2>\n";
./print.php:echo "posted by $poster <br>\n";
./print.php:echo $shortdesc . "\n";
./print.php:echo "<h3>Ingredients:</h3>\n";
./print.php:echo $ingredients . "<br>\n";
./print.php:echo "<h3>Directions:</h3>\n";
./print.php:echo $directions . "\n";
./showrecipe.inc.php:echo "<h2>$title</h2>\n";
./showrecipe.inc.php:echo "by $poster <br><br>\n";
./showrecipe.inc.php:echo $shortdesc . "<br><br>\n";
./showrecipe.inc.php:echo "<h3>Ingredients:</h3>\n";
./showrecipe.inc.php:echo $ingredients . "<br><br>\n";
./showrecipe.inc.php:echo "<h3>Directions:</h3>\n";
./showrecipe.inc.php:echo $directions . "\n";
./showrecipe.inc.php:echo "<br><br>\n";
./showrecipe.inc.php:    echo "No comments posted yet.&nbsp;&nbsp;\n";
./showrecipe.inc.php:    echo "<a href=\"index.php?content=newcomment&id=$recipeid\">Add a comment</a>\n";
./showrecipe.inc.php:    echo "&nbsp;&nbsp;&nbsp;<a href=\"print.php?id=$recipeid\" target=\"_blank\">Print recipe</a>\n";
./showrecipe.inc.php:    echo "<hr>\n";
./showrecipe.inc.php:    echo $row[0] . "\n";
./showrecipe.inc.php:    echo "&nbsp;comments posted.&nbsp;&nbsp;\n";
./showrecipe.inc.php:    echo "<a href=\"index.php?content=newcomment&id=$recipeid\">Add a comment</a>\n";
┌──(kali㉿kali)-[~/Desktop/CodeAlpha/CodeReview/reciphp]
└─$ grep -rn '$recipeid'
search.inc.php:22:            $recipeid = $row['recipeid'];
search.inc.php:25:            echo "<a href=\"index.php?content=showrecipe&id=$recipeid\">$title</a><br>\n";
print.php:14:$recipeid = $_GET['id'];
print.php:16:$query = "SELECT title,poster,shortdesc,ingredients,directions from recipes where recipeid = $recipeid";
showrecipe.inc.php:6:$recipeid = $_GET['id'];
showrecipe.inc.php:8:$query = "SELECT title,poster,shortdesc,ingredients,directions from recipes where recipeid = $recipeid";
showrecipe.inc.php:32:$query = "SELECT count(commentid) from comments where recipeid = $recipeid";
showrecipe.inc.php:38:   echo "<a href=\"index.php?content=newcomment&id=$recipeid\">Add a comment</a>\n";
showrecipe.inc.php:39:   echo "&nbsp;&nbsp;&nbsp;<a href=\"print.php?id=$recipeid\" target=\"_blank\">Print recipe</a>\n";
showrecipe.inc.php:46:   echo "<a href=\"index.php?content=newcomment&id=$recipeid\">Add a comment</a>\n";
showrecipe.inc.php:47:   echo "&nbsp;&nbsp;&nbsp;<a href=\"print.php?id=$recipeid\" target=\"_blank\">Print recipe</a>\n";
showrecipe.inc.php:60:   $query = "SELECT date,poster,comment from comments where recipeid = $recipeid order by commentid desc limit $offset,$recordsperpage";
showrecipe.inc.php:79:      $prevpage = "<a href=\"index.php?content=showrecipe&id=$recipeid&page=$page\">Previous</a> ";
showrecipe.inc.php:95:            $bar .= " <a href=\"index.php?content=showrecipe&id=$recipeid&page=$page\">$page</a> ";
showrecipe.inc.php:103:      $nextpage = " <a href=\"index.php?content=showrecipe&id=$recipeid&page=$page\">Next</a>";
addcomment.inc.php:5:$recipeid = $_POST['recipeid'];
addcomment.inc.php:13:              " VALUES ($recipeid, '$poster', '$date', '$comment')";
addcomment.inc.php:21:echo "<a href=\"index.php?content=showrecipe&id=$recipeid\">Return to recipe</a>\n";
main.inc.php:18:       $recipeid = $row['recipeid'];
main.inc.php:22:       echo "<div id='preview'><a href=\"index.php?content=showrecipe&id=$recipeid\"><h4>$title</h4></a><br/> <font size='0.7em'>Submitted by: <b>$poster</b></font><br/><br/>\n";
newcomment.inc.php:4:$recipeid = $_GET['id'];
newcomment.inc.php:9:    echo "<a href=\"index.php?content=showrecipe&id=$recipeid\">Go back to recipe</a>\n";
newcomment.inc.php:18:    echo "<input type=\"hidden\" name=\"recipeid\" value=\"$recipeid\">\n";
showrecipe.inc.nopaging.php:5:$recipeid = $_GET['id'];
showrecipe.inc.nopaging.php:7:$query = "SELECT title,poster,shortdesc,ingredients,directions from recipes where recipeid = $recipeid";
showrecipe.inc.nopaging.php:31:$query = "SELECT count(commentid) from comments where recipeid = $recipeid";
showrecipe.inc.nopaging.php:37:   echo "<a href=\"index.php?content=newcomment&id=$recipeid\">Add a comment</a>\n";
showrecipe.inc.nopaging.php:38:   echo "&nbsp;&nbsp;&nbsp;<a href=\"print.php?id=$recipeid\" target=\"_blank\">Print recipe</a>\n";
showrecipe.inc.nopaging.php:44:   echo "<a href=\"index.php?content=newcomment&id=$recipeid\">Add a comment</a>\n";
showrecipe.inc.nopaging.php:45:   echo "&nbsp;&nbsp;&nbsp;<a href=\"print.php?id=$recipeid\" target=\"_blank\">Print recipe</a>\n";
showrecipe.inc.nopaging.php:49:   $query = "SELECT date,poster,comment from comments where recipeid = $recipeid order by commentid desc";
newcomment.inc.nologin.php:2:$recipeid = $_GET['id'];
newcomment.inc.nologin.php:8:echo "<input type=\"hidden\" name=\"recipeid\" value=\"$recipeid\">\n";
```

File Inclusion:

Description: File inclusion vulnerabilities occur when an application allows user-controlled input to determine the files that are included or executed. Attackers can exploit this vulnerability to include malicious files from remote locations, leading to arbitrary code execution or unauthorized access to sensitive files.

searching for File Inclusion (RFI/LFI):

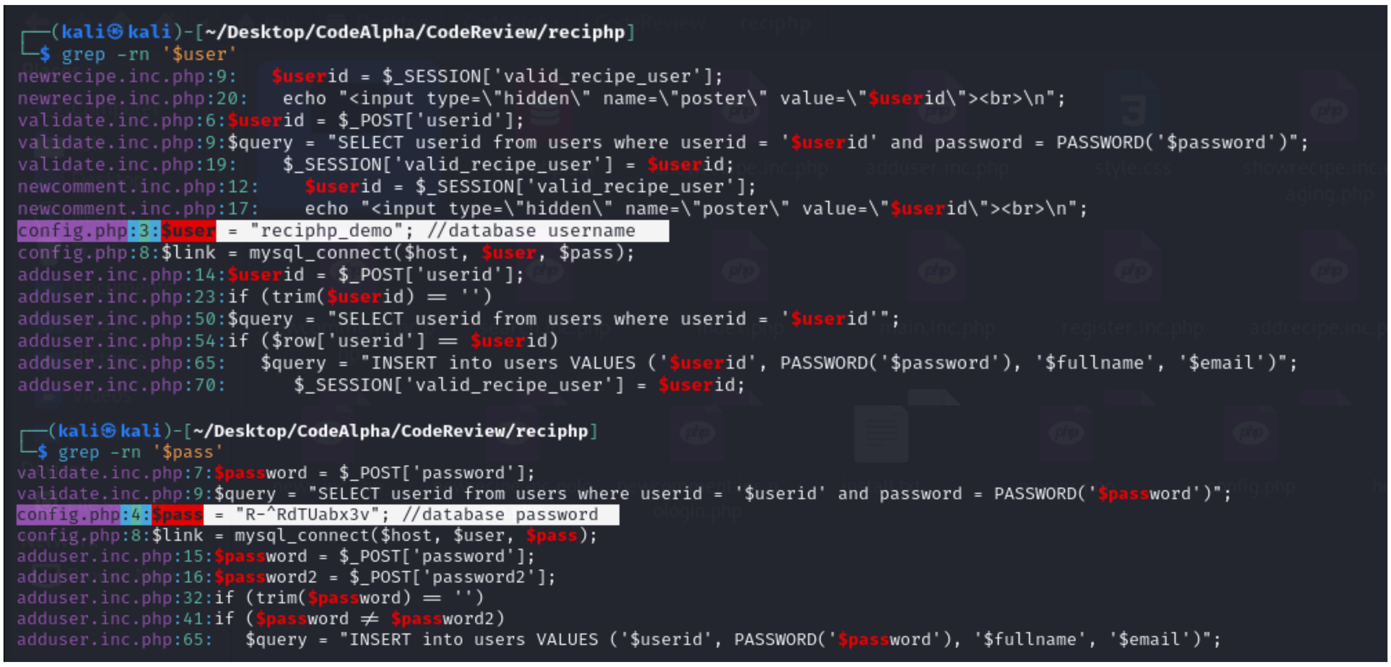


```
┌──(kali㉿kali)-[~/Desktop/CodeAlpha/CodeReview/reciphp]
└─$ grep -Ri "include(" .
./index.php:<?php include("header.inc.php"); ?>
./index.php:<?php include("nav.inc.php"); ?>
./index.php:                    include("main.inc.php");
./index.php:                    include($nextpage);
./index.php:<?php include("news.inc.php"); ?>
./index.php:<?php include("footer.inc.php"); ?>
```

Sensitive Data Exposure:

Description: Storing credentials or sensitive information such as passwords in clear text within the source code is a security risk. If an attacker gains access to the source code, they can easily retrieve these credentials and gain unauthorized access to the system.

searching for sensitive data:



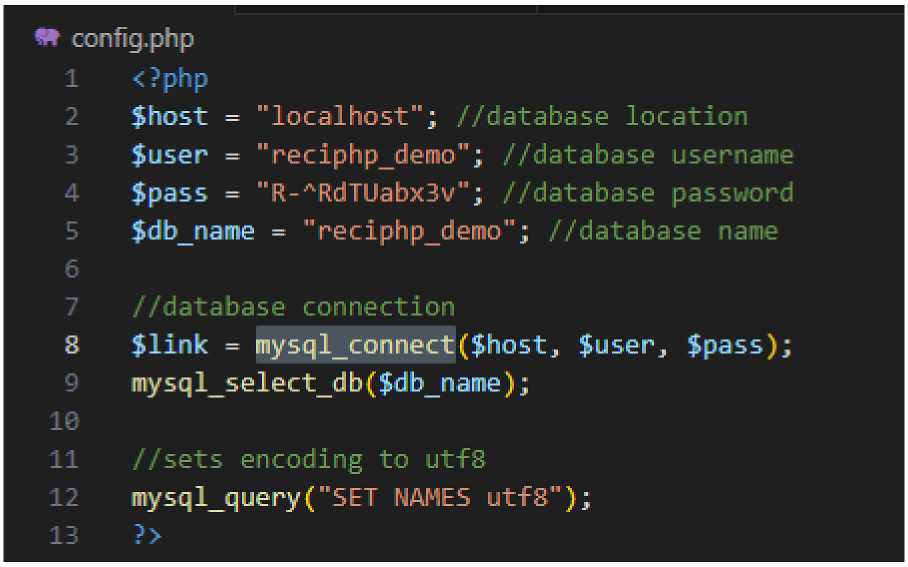
```
┌──(kali㉿kali)-[~/Desktop/CodeAlpha/CodeReview/reciphp]
└─$ grep -rn '$user'
newrecipe.inc.php:9:    $userid = $_SESSION['valid_recipe_user'];
newrecipe.inc.php:20:    echo "<input type=\"hidden\" name=\"poster\" value=\"$userid\"><br>\n";
validate.inc.php:6:$userid = $_POST['userid'];
validate.inc.php:9:$query = "SELECT userid from users where userid = '$userid' and password = PASSWORD('$password')";
validate.inc.php:19:    $_SESSION['valid_recipe_user'] = $userid;
newcomment.inc.php:12:    $userid = $_SESSION['valid_recipe_user'];
newcomment.inc.php:17:    echo "<input type=\"hidden\" name=\"poster\" value=\"$userid\"><br>\n";
config.php:3:$user = "reciphp_demo"; //database username
config.php:8:$link = mysql_connect($host, $user, $pass);
adduser.inc.php:14:$userid = $_POST['userid'];
adduser.inc.php:23:if (trim($userid) == '')
adduser.inc.php:50:$query = "SELECT userid from users where userid = '$userid'";
adduser.inc.php:54:if ($row['userid'] == $userid)
adduser.inc.php:65:   $query = "INSERT into users VALUES ('$userid', PASSWORD('$password'), '$fullname', '$email')";
adduser.inc.php:70:      $_SESSION['valid_recipe_user'] = $userid;

┌──(kali㉿kali)-[~/Desktop/CodeAlpha/CodeReview/reciphp]
└─$ grep -rn '$pass'
validate.inc.php:7:$password = $_POST['password'];
validate.inc.php:9:$query = "SELECT userid from users where userid = '$userid' and password = PASSWORD('$password')";
config.php:4:$pass = "R-^RdTUabx3v"; //database password
config.php:8:$link = mysql_connect($host, $user, $pass);
adduser.inc.php:15:$password = $_POST['password'];
adduser.inc.php:16:$password2 = $_POST['password2'];
adduser.inc.php:32:if (trim($password) == '')
adduser.inc.php:41:if ($password ≠ $password2)
adduser.inc.php:65:   $query = "INSERT into users VALUES ('$userid', PASSWORD('$password'), '$fullname', '$email')";
```

Vulnerabilities and Fixes

File: config.php

- Vulnerabilities: Hard-coded credentials and use of deprecated functions (mysql_connect)
- Risk Severity: Medium
- Likelihood of Exploitation: Low
- Ease of Exploitation: Low
- Remediation Approach: Remove hard-coded credentials and switch to PDO(PHP Data Objects) for database connection. Use environment variables for storing credentials.

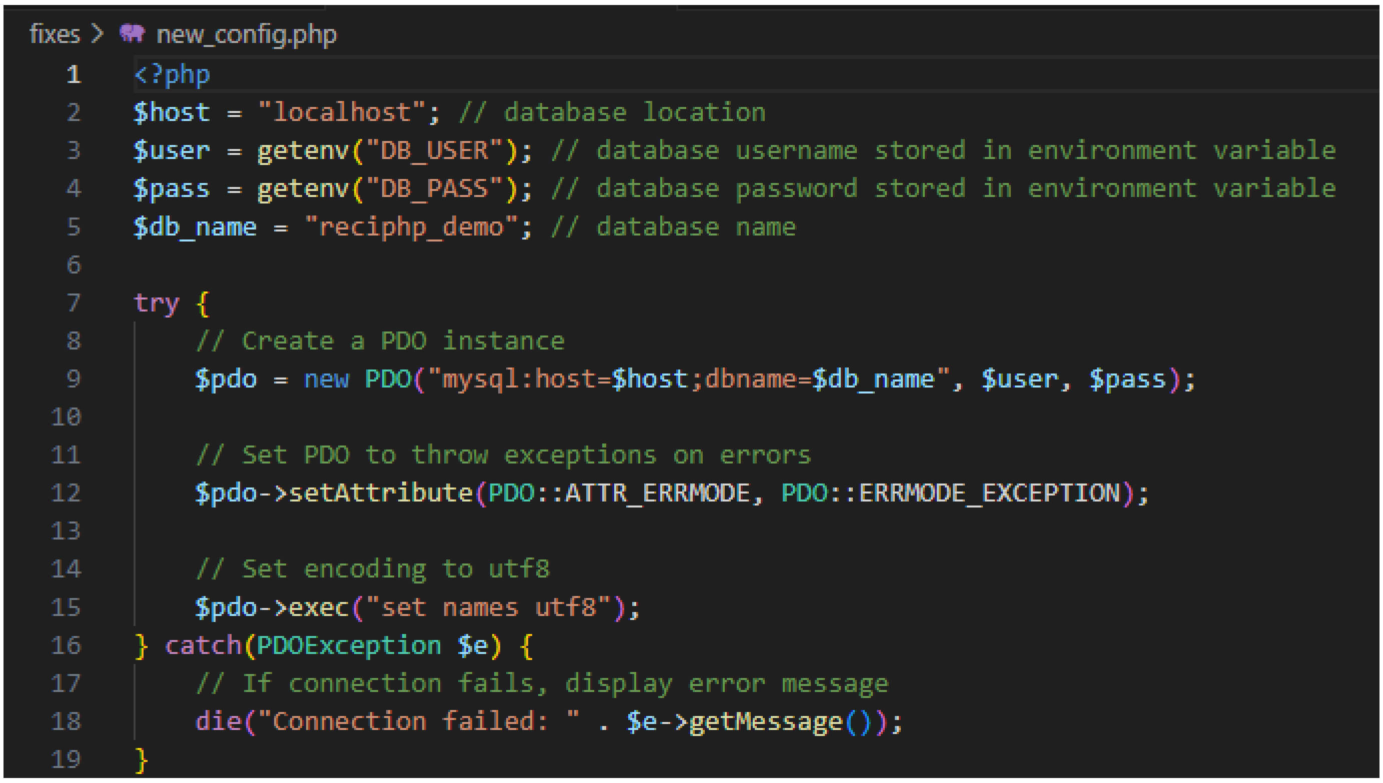
Vulnerable Code:



config.php

```php
<?php
$host = "localhost"; //database location
$user = "reciphp_demo"; //database username
$pass = "R-^RdTUabx3v"; //database password
$db_name = "reciphp_demo"; //database name

//database connection
$link = mysql_connect($host, $user, $pass);
mysql_select_db($db_name);

//sets encoding to utf8
mysql_query("SET NAMES utf8");
?>
```

Fixed Code:



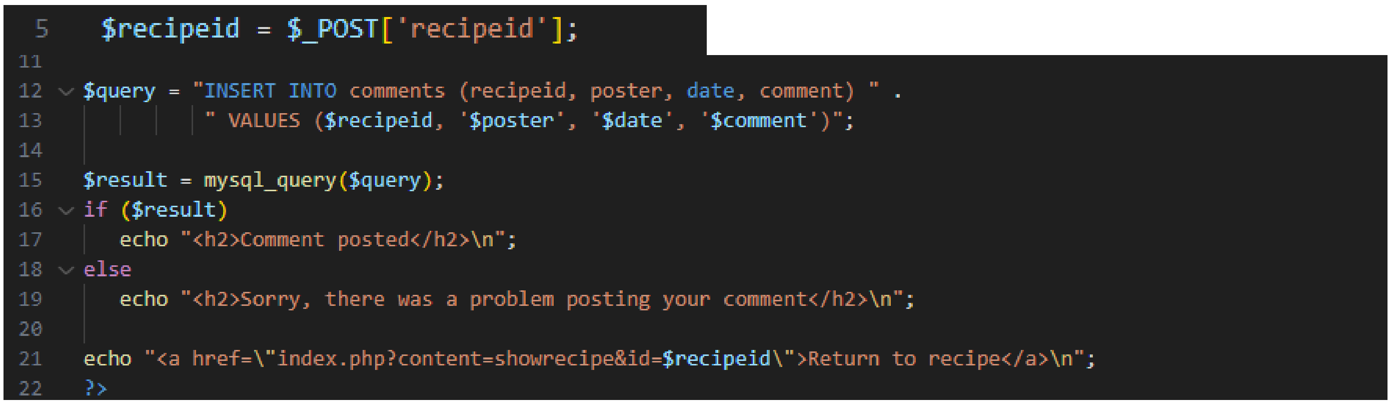
fixes > new_config.php

```php
<?php
$host = "localhost"; // database location
$user = getenv("DB_USER"); // database username stored in environment variable
$pass = getenv("DB_PASS"); // database password stored in environment variable
$db_name = "reciphp_demo"; // database name

try {
    // Create a PDO instance
    $pdo = new PDO("mysql:host=$host;dbname=$db_name", $user, $pass);

    // Set PDO to throw exceptions on errors
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // Set encoding to utf8
    $pdo->exec("set names utf8");
} catch(PDOException $e) {
    // If connection fails, display error message
    die("Connection failed: " . $e->getMessage());
}
```

File: addcomment.inc.php

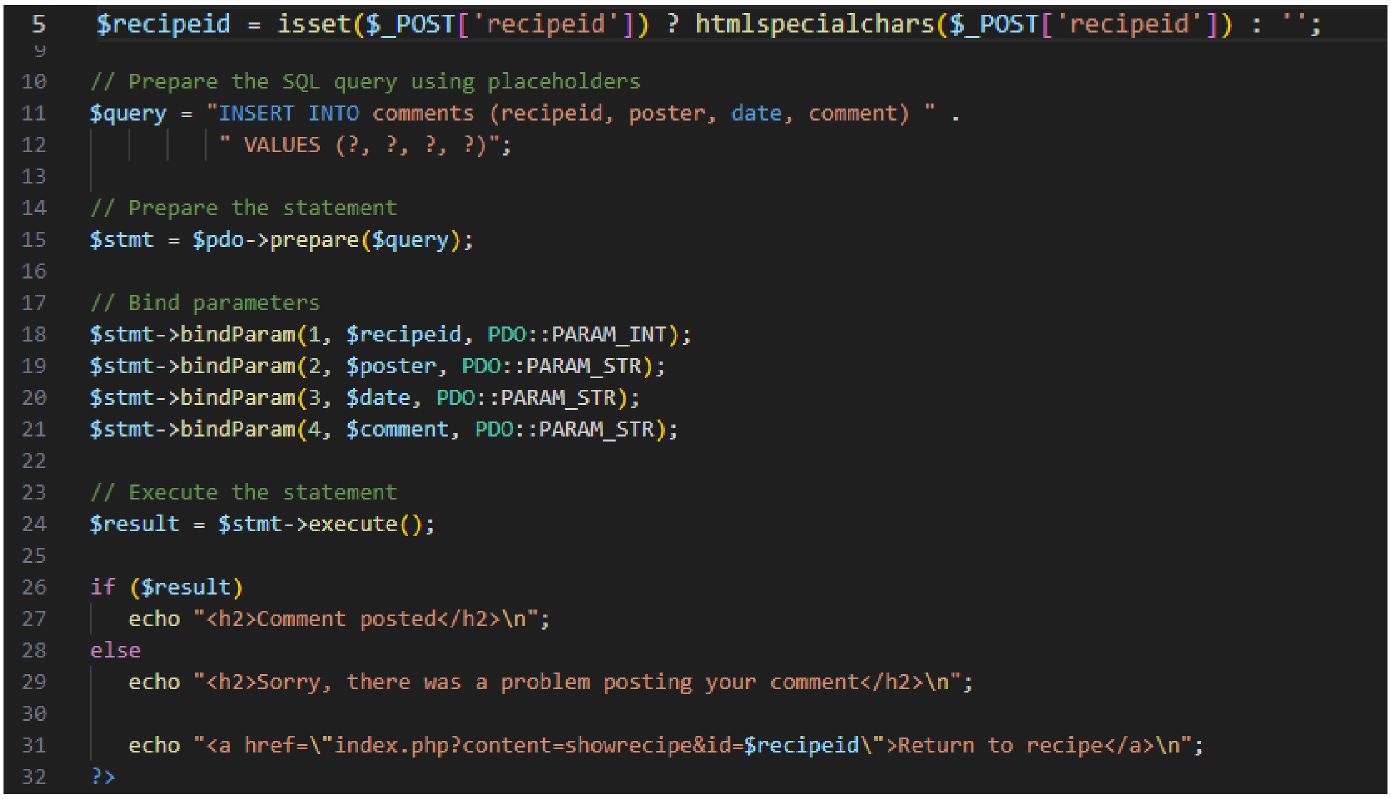
- Vulnerabilities: SQL injection (SQLi) and cross-site scripting (XSS)
- Risk Severity: High
- Likelihood of Exploitation: Moderate
- Ease of Exploitation: Moderate
- Remediation Approach: Implement parameterized queries or prepared statements using PDO(PHP Data Objects) to prevent SQL injection. Sanitize user input using htmlspecialchars to prevent XSS attacks.

Vulnerable Code:



```
$recipeid = $_POST['recipeid'];

$query = "INSERT INTO comments (recipeid, poster, date, comment) " .
            " VALUES ($recipeid, '$poster', '$date', '$comment')";

$result = mysql_query($query);
if ($result)
    echo "<h2>Comment posted</h2>\n";
else
    echo "<h2>Sorry, there was a problem posting your comment</h2>\n";

echo "<a href=\"index.php?content=showrecipe&id=$recipeid\">Return to recipe</a>\n";
?>
```

Fixed Code:

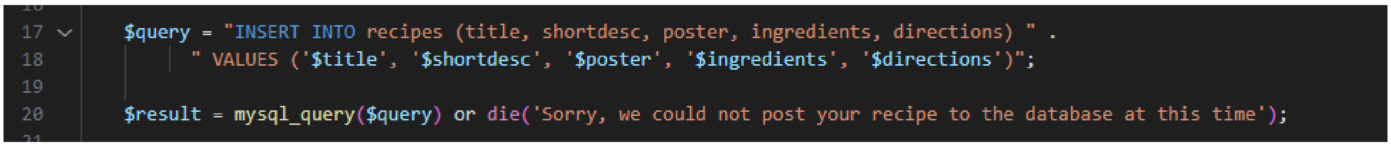


```
$recipeid = isset($_POST['recipeid']) ? htmlspecialchars($_POST['recipeid']) : '';

// Prepare the SQL query using placeholders
$query = "INSERT INTO comments (recipeid, poster, date, comment) " .
            " VALUES (?, ?, ?, ?)";

// Prepare the statement
$stmt = $pdo->prepare($query);

// Bind parameters
$stmt->bindParam(1, $recipeid, PDO::PARAM_INT);
$stmt->bindParam(2, $poster, PDO::PARAM_STR);
$stmt->bindParam(3, $date, PDO::PARAM_STR);
$stmt->bindParam(4, $comment, PDO::PARAM_STR);

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

if ($result)
    echo "<h2>Comment posted</h2>\n";
else
    echo "<h2>Sorry, there was a problem posting your comment</h2>\n";

    echo "<a href=\"index.php?content=showrecipe&id=$recipeid\">Return to recipe</a>\n";
?>
```

File: addrecipe.inc.php

- Vulnerabilities: SQL injection (SQLi)
- Risk Severity: High
- Likelihood of Exploitation: High
- Ease of Exploitation: Moderate
- Remediation Approach: Implement parameterized queries or prepared statements using PDO(PHP Data Objects) to prevent SQL injection.

Vulnerable Code:



```
    $query = "INSERT INTO recipes (title, shortdesc, poster, ingredients, directions) " .
        " VALUES ('$title', '$shortdesc', '$poster', '$ingredients', '$directions')";

    $result = mysql_query($query) or die('Sorry, we could not post your recipe to the database at this time');
```

Fixed Code:

```
// Prepare the SQL query using placeholders
$query = "INSERT INTO recipes (title, shortdesc, poster, ingredients, directions) " .
         " VALUES (?, ?, ?, ?, ?)";

// Prepare the statement
$stmt = $pdo->prepare($query);

// Bind parameters
$stmt->bindParam(1, $title, PDO::PARAM_STR);
$stmt->bindParam(2, $shortdesc, PDO::PARAM_STR);
$stmt->bindParam(3, $poster, PDO::PARAM_STR);
$stmt->bindParam(4, $ingredients, PDO::PARAM_STR);
$stmt->bindParam(5, $directions, PDO::PARAM_STR);

// Execute the statement
$result = $stmt->execute();
```

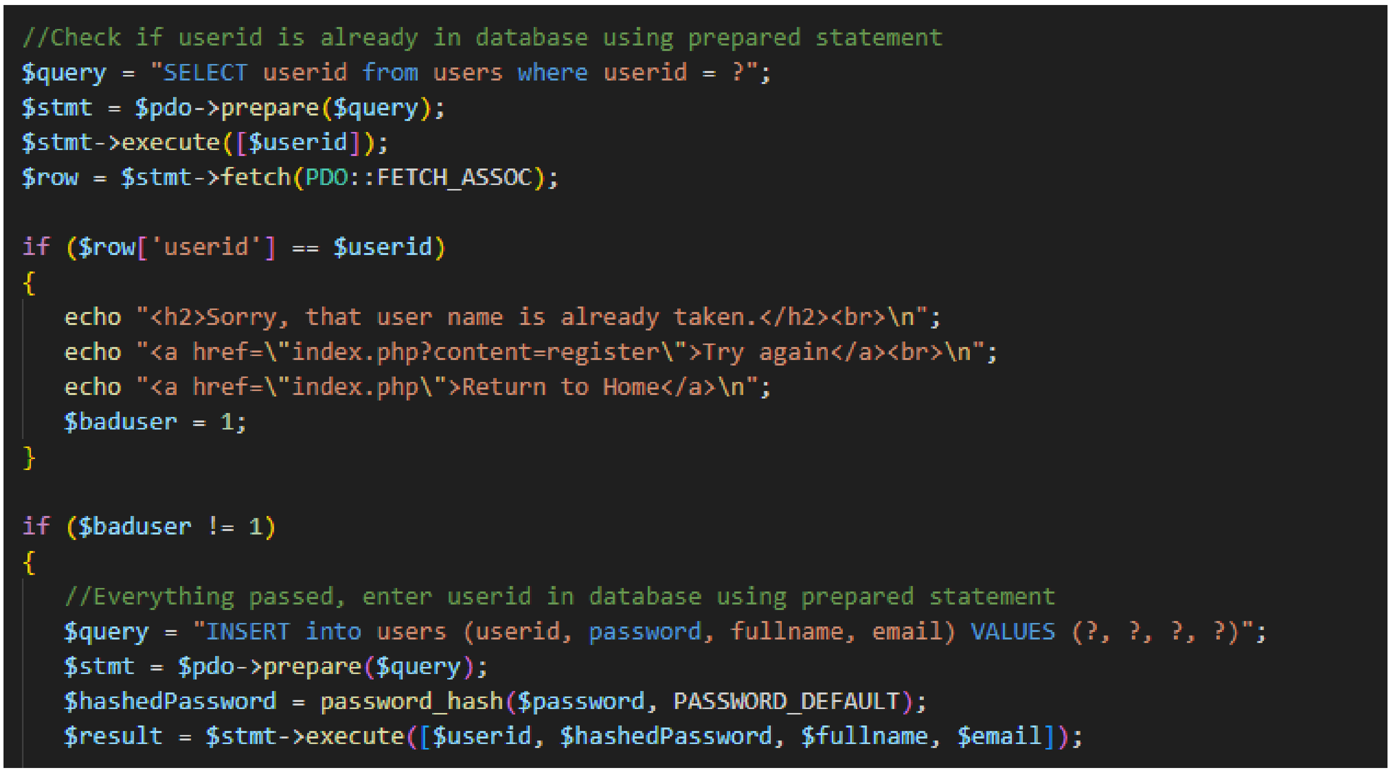
File: adduser.inc.php

- Vulnerabilities: SQL injection (SQLi)
- Risk Severity: High
- Likelihood of Exploitation: High
- Ease of Exploitation: Moderate
- Remediation Approach: Implement parameterized queries or prepared statements using PDO(PHP Data Objects) to prevent SQL injection.

Vulnerable Code:

```
//Check if userid is already in database
$query = "SELECT userid from users where userid = '$userid'";
$result = mysql_query($query);
$row = mysql_fetch_array($result, MYSQL_ASSOC);

if ($row['userid'] == $userid)
{
   echo "<h2>Sorry, that user name is already taken.</h2><br>\n";
   echo "<a href=\"index.php?content=register\">Try again</a><br>\n";
   echo "<a href=\"index.php\">Return to Home</a>\n";
   $baduser = 1;
}

if ($baduser != 1)
{
   //Everything passed, enter userid in database
   $query = "INSERT into users VALUES ('$userid', PASSWORD('$password'), '$fullname', '$email')";
   $result = mysql_query($query) or die('Sorry, we are unable to process your request.');
```

Fixed Code:

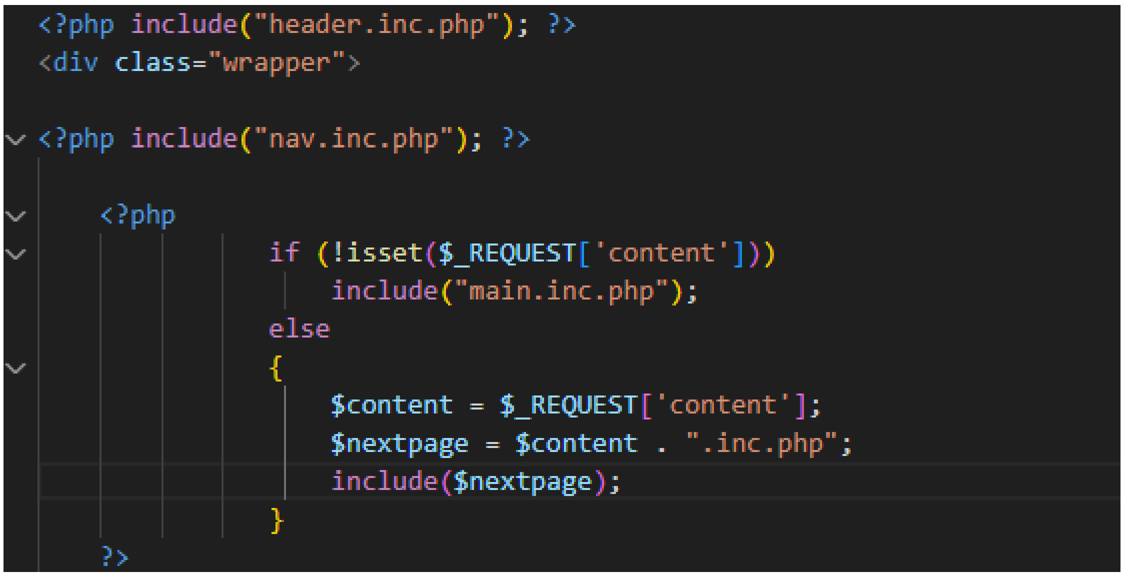


```php
//Check if userid is already in database using prepared statement
$query = "SELECT userid from users where userid = ?";
$stmt = $pdo->prepare($query);
$stmt->execute([$userid]);
$row = $stmt->fetch(PDO::FETCH_ASSOC);

if ($row['userid'] == $userid)
{
   echo "<h2>Sorry, that user name is already taken.</h2><br>\n";
   echo "<a href=\"index.php?content=register\">Try again</a><br>\n";
   echo "<a href=\"index.php\">Return to Home</a>\n";
   $baduser = 1;
}

if ($baduser != 1)
{
   //Everything passed, enter userid in database using prepared statement
   $query = "INSERT into users (userid, password, fullname, email) VALUES (?, ?, ?, ?)";
   $stmt = $pdo->prepare($query);
   $hashedPassword = password_hash($password, PASSWORD_DEFAULT);
   $result = $stmt->execute([$userid, $hashedPassword, $fullname, $email]);
```

File: index.php

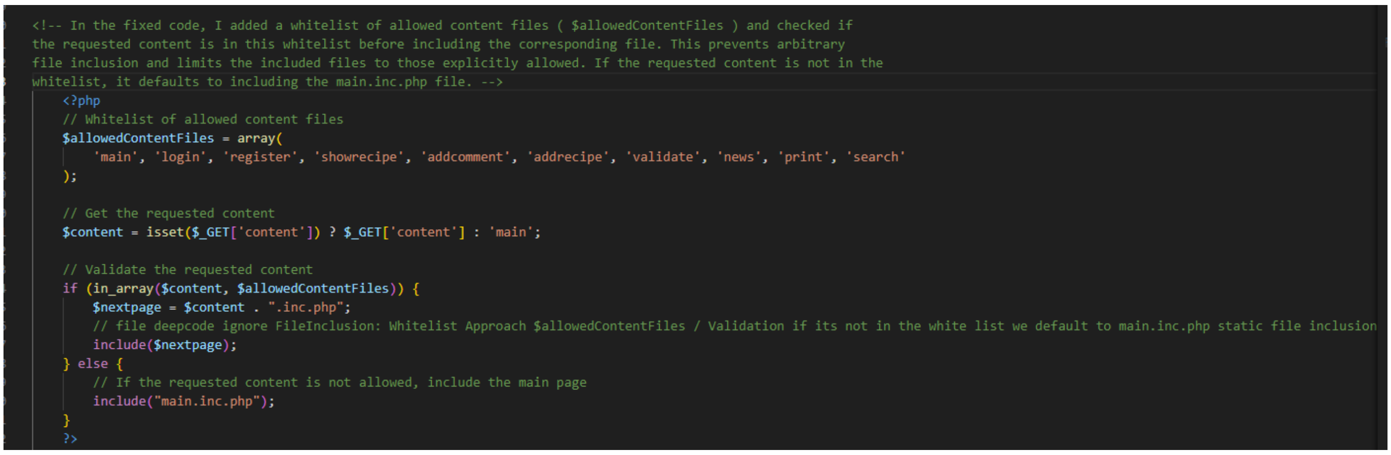
- Vulnerabilities: Remote file inclusion
- Risk Severity: Medium
- Likelihood of Exploitation: Low
- Ease of Exploitation: Low
- Remediation Approach: Avoid using user-controlled input in file inclusion functions. Use whitelisting or input validation to restrict file paths.

Vulnerable Code:



```php
<?php include("header.inc.php"); ?>
<div class="wrapper">

<?php include("nav.inc.php"); ?>

    <?php
            if (!isset($_REQUEST['content']))
                include("main.inc.php");
            else
            {
                $content = $_REQUEST['content'];
                $nextpage = $content . ".inc.php";
                include($nextpage);
            }
    ?>
```

Fixed Code:

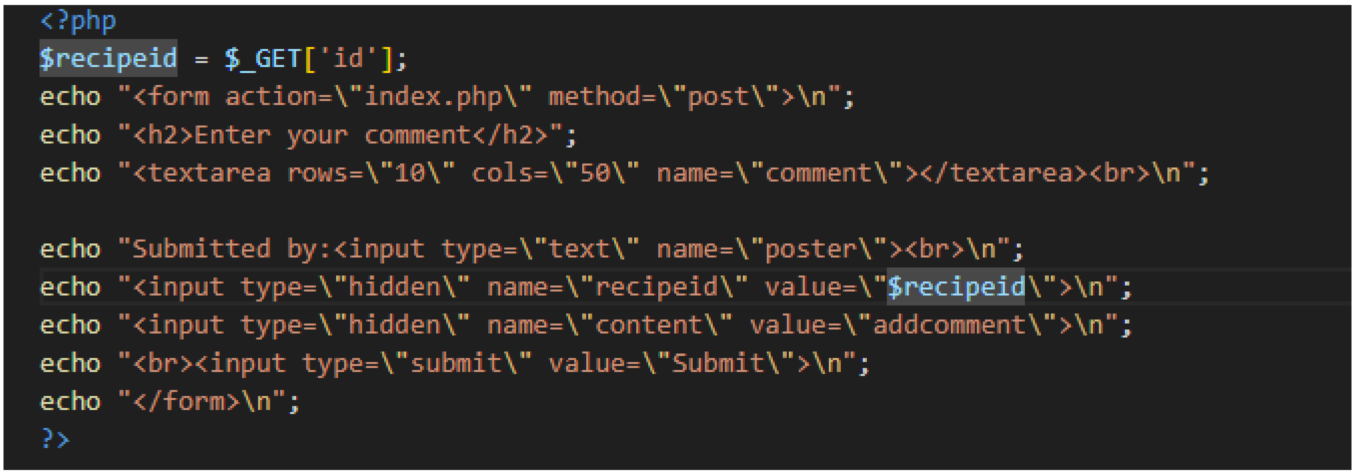


```php
<!-- In the fixed code, I added a whitelist of allowed content files ( $allowedContentFiles ) and checked if
the requested content is in this whitelist before including the corresponding file. This prevents arbitrary
file inclusion and limits the included files to those explicitly allowed. If the requested content is not in the
whitelist, it defaults to including the main.inc.php file. -->
    <?php
    // Whitelist of allowed content files
    $allowedContentFiles = array(
        'main', 'login', 'register', 'showrecipe', 'addcomment', 'addrecipe', 'validate', 'news', 'print', 'search'
    );

    // Get the requested content
    $content = isset($_GET['content']) ? $_GET['content'] : 'main';

    // Validate the requested content
    if (in_array($content, $allowedContentFiles)) {
        $nextpage = $content . ".inc.php";
        // file deepcode ignore FileInclusion: Whitelist Approach $allowedContentFiles / Validation if its not in the white list we default to main.inc.php static file inclusion
        include($nextpage);
    } else {
        // If the requested content is not allowed, include the main page
        include("main.inc.php");
    }
    ?>
```

File: newcomment.inc.nologin.php

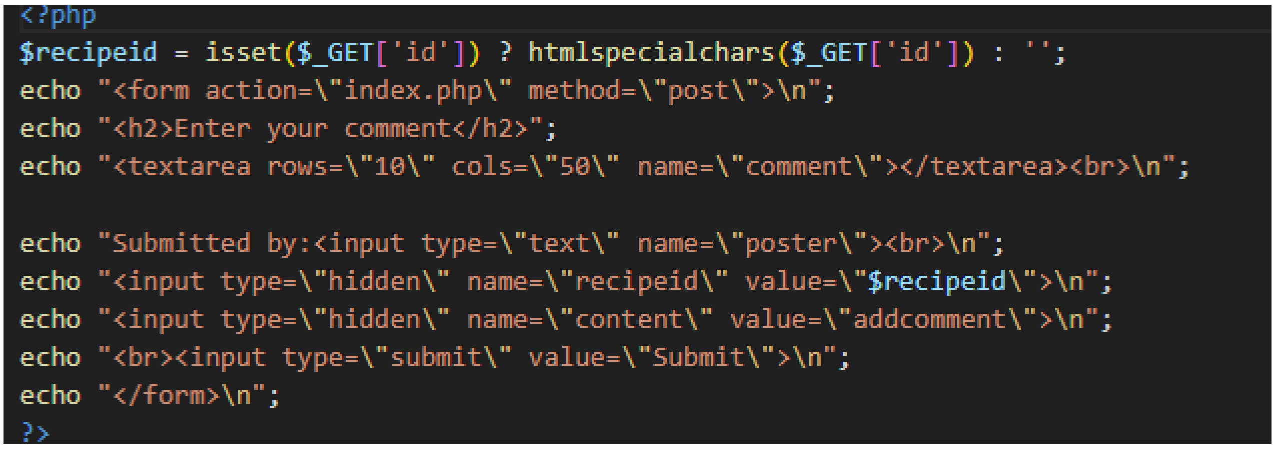
- Vulnerabilities: Cross-site scripting (XSS)
- Risk Severity: Medium
- Likelihood of Exploitation: Moderate
- Ease of Exploitation: Moderate
- Remediation Approach: Sanitize user input using htmlspecialchars to prevent XSS attacks.

Vulnerable Code:



```php
<?php
$recipeid = $_GET['id'];
echo "<form action=\"index.php\" method=\"post\">\n";
echo "<h2>Enter your comment</h2>";
echo "<textarea rows=\"10\" cols=\"50\" name=\"comment\"></textarea><br>\n";

echo "Submitted by:<input type=\"text\" name=\"poster\"><br>\n";
echo "<input type=\"hidden\" name=\"recipeid\" value=\"$recipeid\">\n";
echo "<input type=\"hidden\" name=\"content\" value=\"addcomment\">\n";
echo "<br><input type=\"submit\" value=\"Submit\">\n";
echo "</form>\n";
?>
```

Fixed Code:

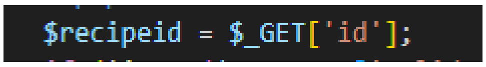


```php
<?php
$recipeid = isset($_GET['id']) ? htmlspecialchars($_GET['id']) : '';
echo "<form action=\"index.php\" method=\"post\">\n";
echo "<h2>Enter your comment</h2>";
echo "<textarea rows=\"10\" cols=\"50\" name=\"comment\"></textarea><br>\n";

echo "Submitted by:<input type=\"text\" name=\"poster\"><br>\n";
echo "<input type=\"hidden\" name=\"recipeid\" value=\"$recipeid\">\n";
echo "<input type=\"hidden\" name=\"content\" value=\"addcomment\">\n";
echo "<br><input type=\"submit\" value=\"Submit\">\n";
echo "</form>\n";
?>
```

File: newcomment.inc.php

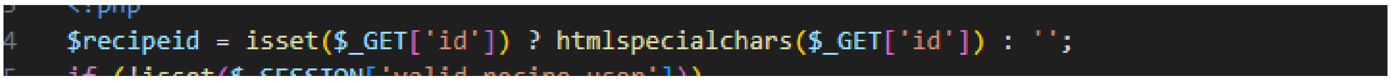
- Vulnerabilities: Cross-site scripting (XSS)
- Risk Severity: Medium
- Likelihood of Exploitation: Moderate
- Ease of Exploitation: Moderate
- Remediation Approach: Sanitize user input using htmlspecialchars to prevent XSS attacks.

Vulnerable Code:



```php
$recipeid = $_GET['id'];
```

Fixed Code:

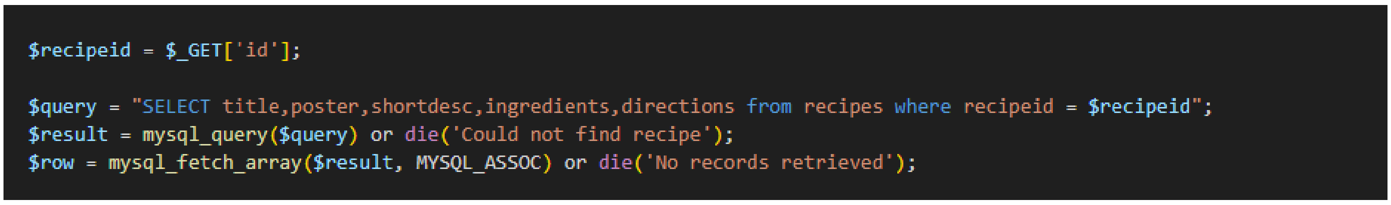


```php
<?php
$recipeid = isset($_GET['id']) ? htmlspecialchars($_GET['id']) : '';
```

File: print.php

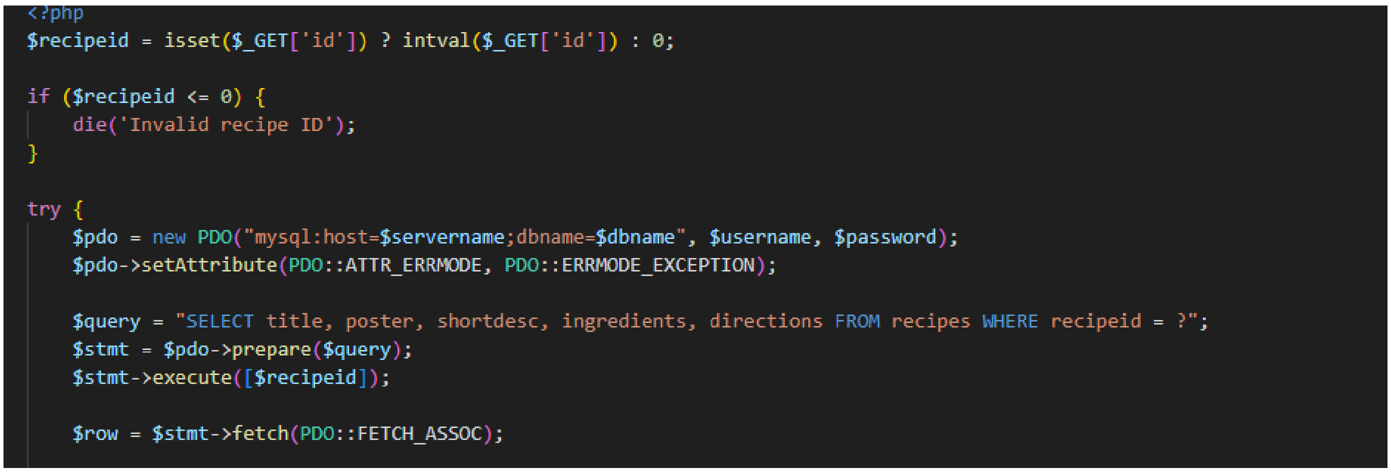
- Vulnerabilities: SQL injection (SQLi)
- Risk Severity: High
- Likelihood of Exploitation: High
- Ease of Exploitation: Moderate
- Remediation Approach: Implement parameterized queries or prepared statements using PDO(PHP Data Objects) to prevent SQL injection.

Vulnerable Code:



```php
$recipeid = $_GET['id'];

$query = "SELECT title,poster,shortdesc,ingredients,directions from recipes where recipeid = $recipeid";
$result = mysql_query($query) or die('Could not find recipe');
$row = mysql_fetch_array($result, MYSQL_ASSOC) or die('No records retrieved');
```

Fixed Code:

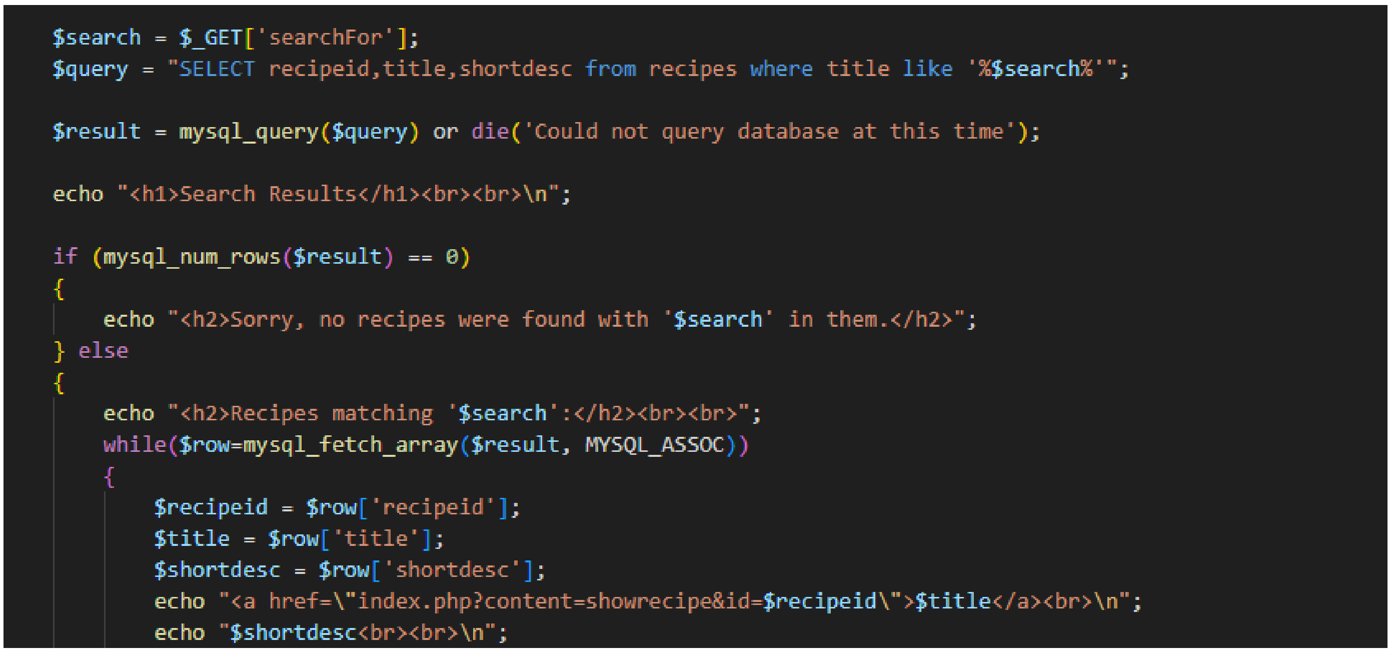


```php
<?php
$recipeid = isset($_GET['id']) ? intval($_GET['id']) : 0;

if ($recipeid <= 0) {
    die('Invalid recipe ID');
}

try {
    $pdo = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $query = "SELECT title, poster, shortdesc, ingredients, directions FROM recipes WHERE recipeid = ?";
    $stmt = $pdo->prepare($query);
    $stmt->execute([$recipeid]);

    $row = $stmt->fetch(PDO::FETCH_ASSOC);
```

File: search.inc.php

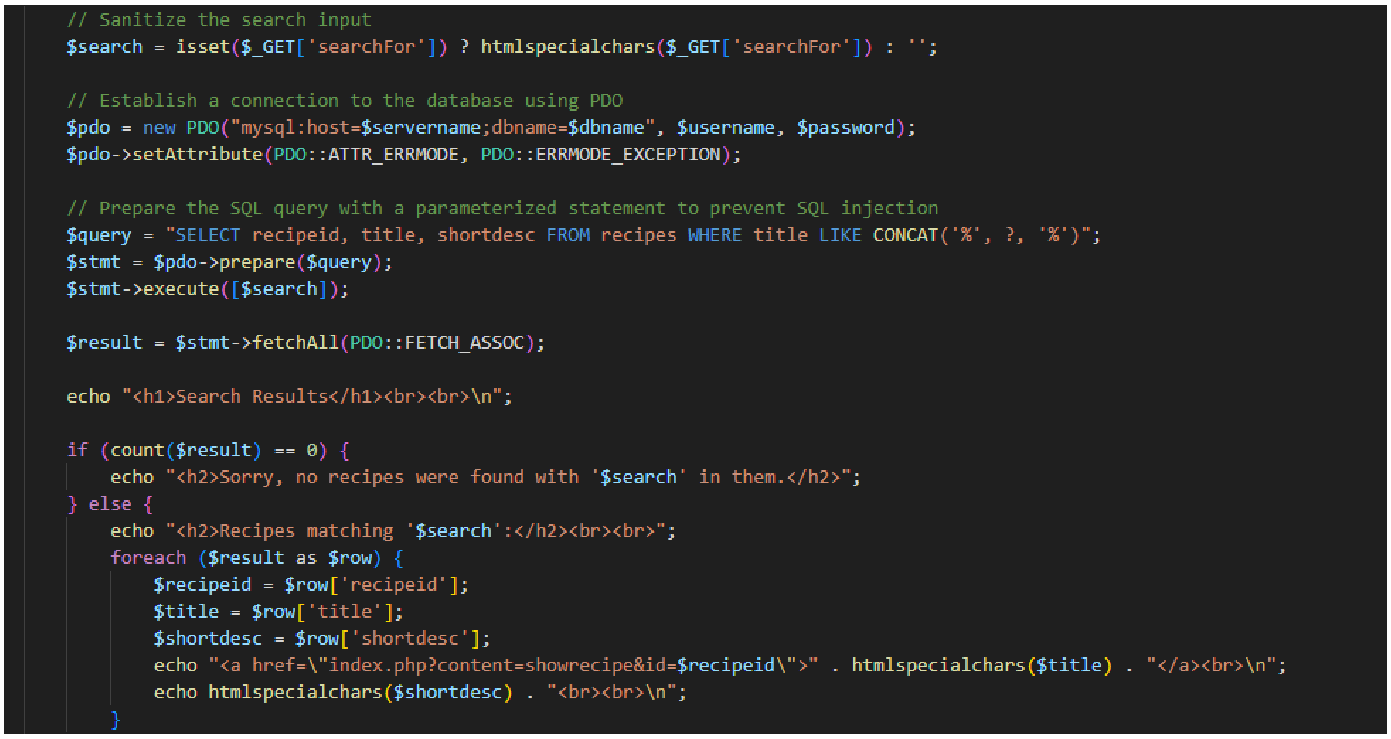
- Vulnerabilities: SQL injection (SQLi) and cross-site scripting (XSS)
- Risk Severity: High
- Likelihood of Exploitation: High
- Ease of Exploitation: Moderate
- Remediation Approach: Implement parameterized queries or prepared statements using PDO(PHP Data Objects) to prevent SQL injection. Sanitize user input using htmlspecialchars to prevent XSS attacks.

Vulnerable Code:



```php
$search = $_GET['searchFor'];
$query = "SELECT recipeid,title,shortdesc from recipes where title like '%$search%'";

$result = mysql_query($query) or die('Could not query database at this time');

echo "<h1>Search Results</h1><br><br>\n";

if (mysql_num_rows($result) == 0)
{
    echo "<h2>Sorry, no recipes were found with '$search' in them.</h2>";
} else
{
    echo "<h2>Recipes matching '$search':</h2><br><br>";
    while($row=mysql_fetch_array($result, MYSQL_ASSOC))
    {
        $recipeid = $row['recipeid'];
        $title = $row['title'];
        $shortdesc = $row['shortdesc'];
        echo "<a href=\"index.php?content=showrecipe&id=$recipeid\">$title</a><br>\n";
        echo "$shortdesc<br><br>\n";
```

Fixed Code:

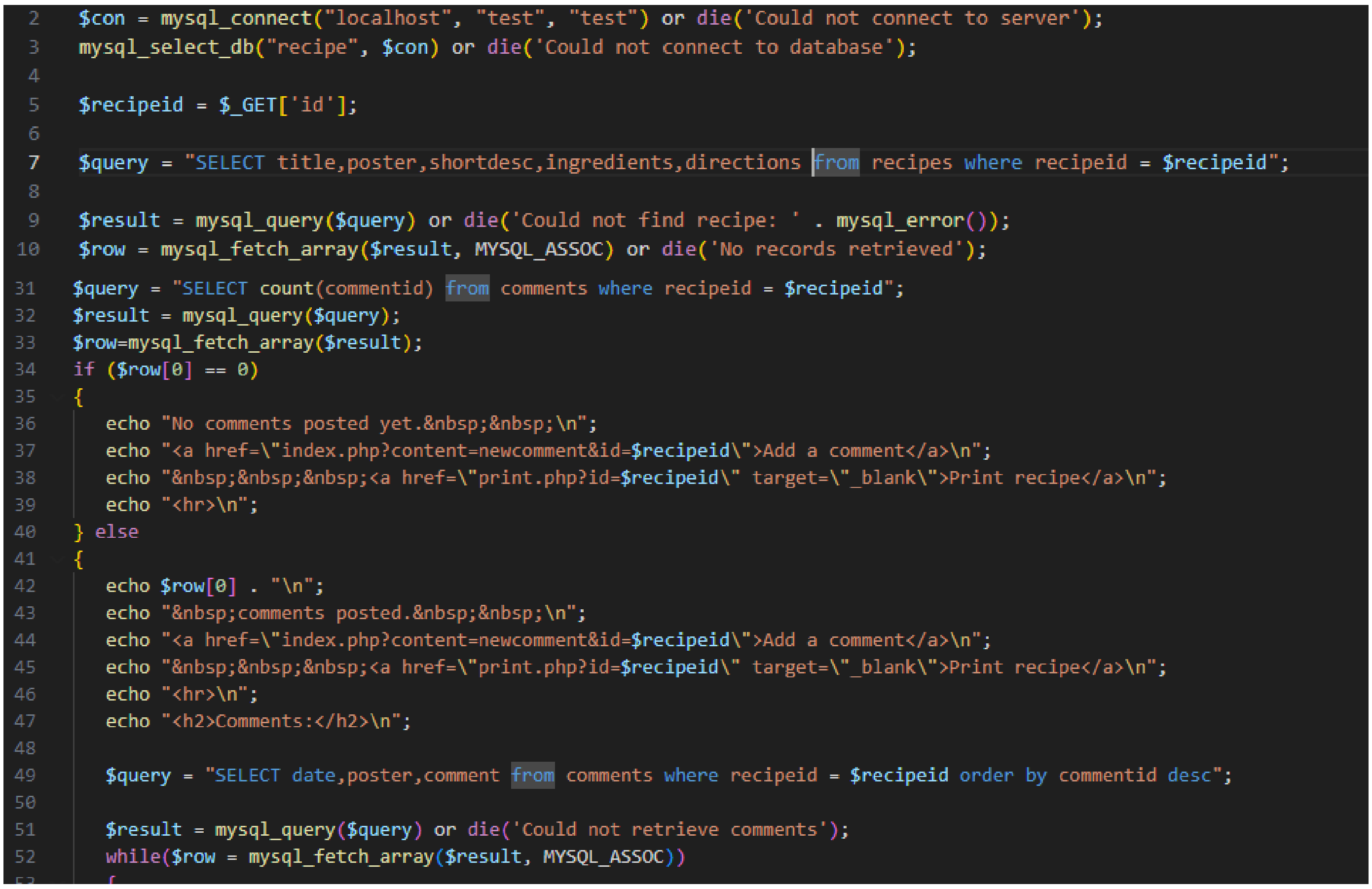


```php
// Sanitize the search input
$search = isset($_GET['searchFor']) ? htmlspecialchars($_GET['searchFor']) : '';

// Establish a connection to the database using PDO
$pdo = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

// Prepare the SQL query with a parameterized statement to prevent SQL injection
$query = "SELECT recipeid, title, shortdesc FROM recipes WHERE title LIKE CONCAT('%', ?, '%')";
$stmt = $pdo->prepare($query);
$stmt->execute([$search]);

$result = $stmt->fetchAll(PDO::FETCH_ASSOC);

echo "<h1>Search Results</h1><br><br>\n";

if (count($result) == 0) {
    echo "<h2>Sorry, no recipes were found with '$search' in them.</h2>";
} else {
    echo "<h2>Recipes matching '$search':</h2><br><br>";
    foreach ($result as $row) {
        $recipeid = $row['recipeid'];
        $title = $row['title'];
        $shortdesc = $row['shortdesc'];
        echo "<a href=\"index.php?content=showrecipe&id=$recipeid\">" . htmlspecialchars($title) . "</a><br>\n";
        echo htmlspecialchars($shortdesc) . "<br><br>\n";
    }
```

File: showrecipe.inc.nopaging.php

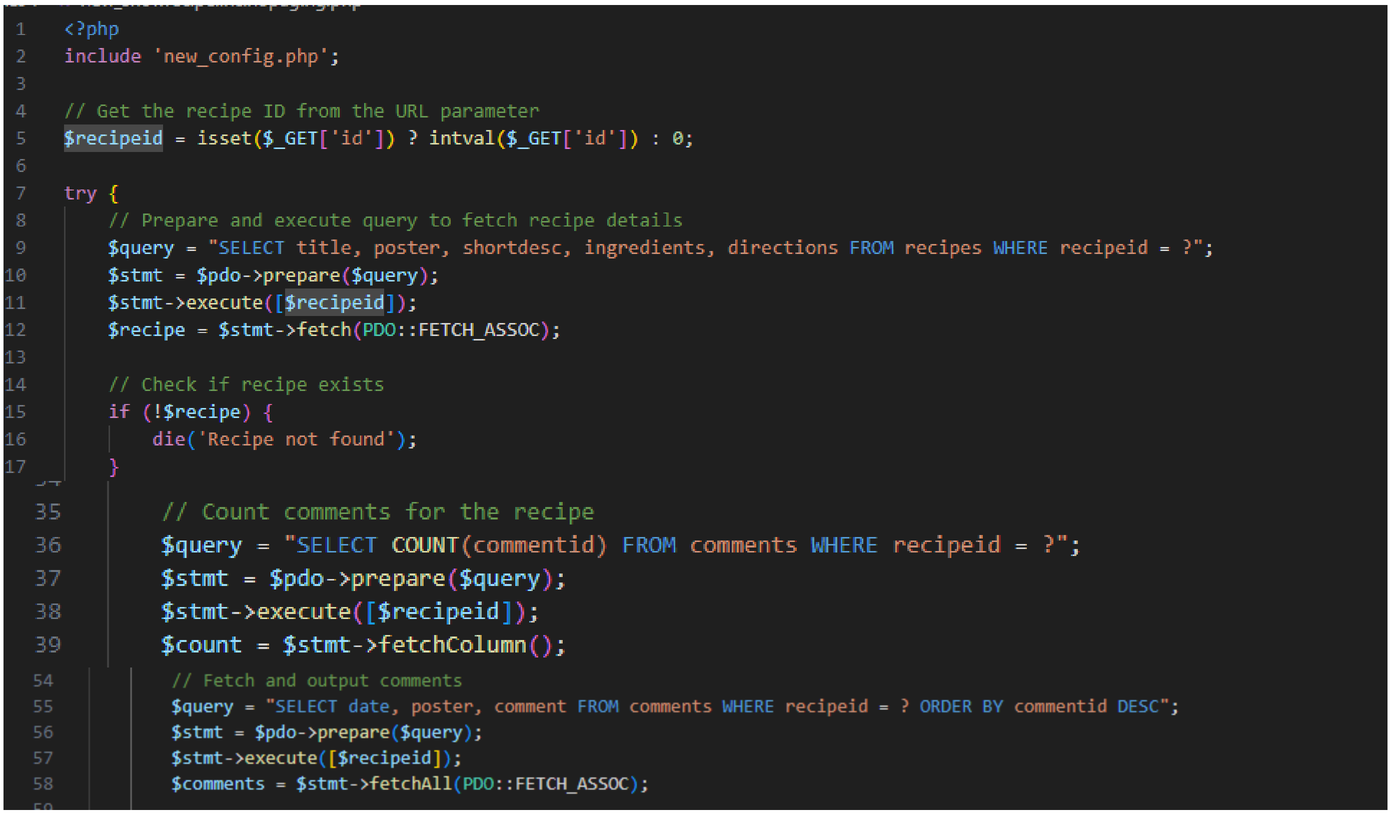
- Vulnerabilities: SQL injection (SQLi) and cross-site scripting (XSS), clear text credentials
- Risk Severity: High
- Likelihood of Exploitation: High
- Ease of Exploitation: Moderate
- Remediation Approach: Implement parameterized queries or prepared statements using PDO(PHP Data Objects) to prevent SQL injection. Sanitize user input using htmlspecialchars to prevent XSS attacks. Avoid storing credentials in clear text.

Vulnerable Code:



```php
$con = mysql_connect("localhost", "test", "test") or die('Could not connect to server');
mysql_select_db("recipe", $con) or die('Could not connect to database');

$recipeid = $_GET['id'];

$query = "SELECT title,poster,shortdesc,ingredients,directions from recipes where recipeid = $recipeid";

$result = mysql_query($query) or die('Could not find recipe: ' . mysql_error());
$row = mysql_fetch_array($result, MYSQL_ASSOC) or die('No records retrieved');
$query = "SELECT count(commentid) from comments where recipeid = $recipeid";
$result = mysql_query($query);
$row=mysql_fetch_array($result);
if ($row[0] == 0)
{
    echo "No comments posted yet.&nbsp;&nbsp;\n";
    echo "<a href=\"index.php?content=newcomment&id=$recipeid\">Add a comment</a>\n";
    echo "&nbsp;&nbsp;&nbsp;<a href=\"print.php?id=$recipeid\" target=\"_blank\">Print recipe</a>\n";
    echo "<hr>\n";
} else
{
    echo $row[0] . "\n";
    echo "&nbsp;comments posted.&nbsp;&nbsp;\n";
    echo "<a href=\"index.php?content=newcomment&id=$recipeid\">Add a comment</a>\n";
    echo "&nbsp;&nbsp;&nbsp;<a href=\"print.php?id=$recipeid\" target=\"_blank\">Print recipe</a>\n";
    echo "<hr>\n";
    echo "<h2>Comments:</h2>\n";

    $query = "SELECT date,poster,comment from comments where recipeid = $recipeid order by commentid desc";

    $result = mysql_query($query) or die('Could not retrieve comments');
    while($row = mysql_fetch_array($result, MYSQL_ASSOC))
```

Fixed Code:



```php
<?php
include 'new_config.php';

// Get the recipe ID from the URL parameter
$recipeid = isset($_GET['id']) ? intval($_GET['id']) : 0;

try {
    // Prepare and execute query to fetch recipe details
    $query = "SELECT title, poster, shortdesc, ingredients, directions FROM recipes WHERE recipeid = ?";
    $stmt = $pdo->prepare($query);
    $stmt->execute([$recipeid]);
    $recipe = $stmt->fetch(PDO::FETCH_ASSOC);

    // Check if recipe exists
    if (!$recipe) {
        die('Recipe not found');
    }
```

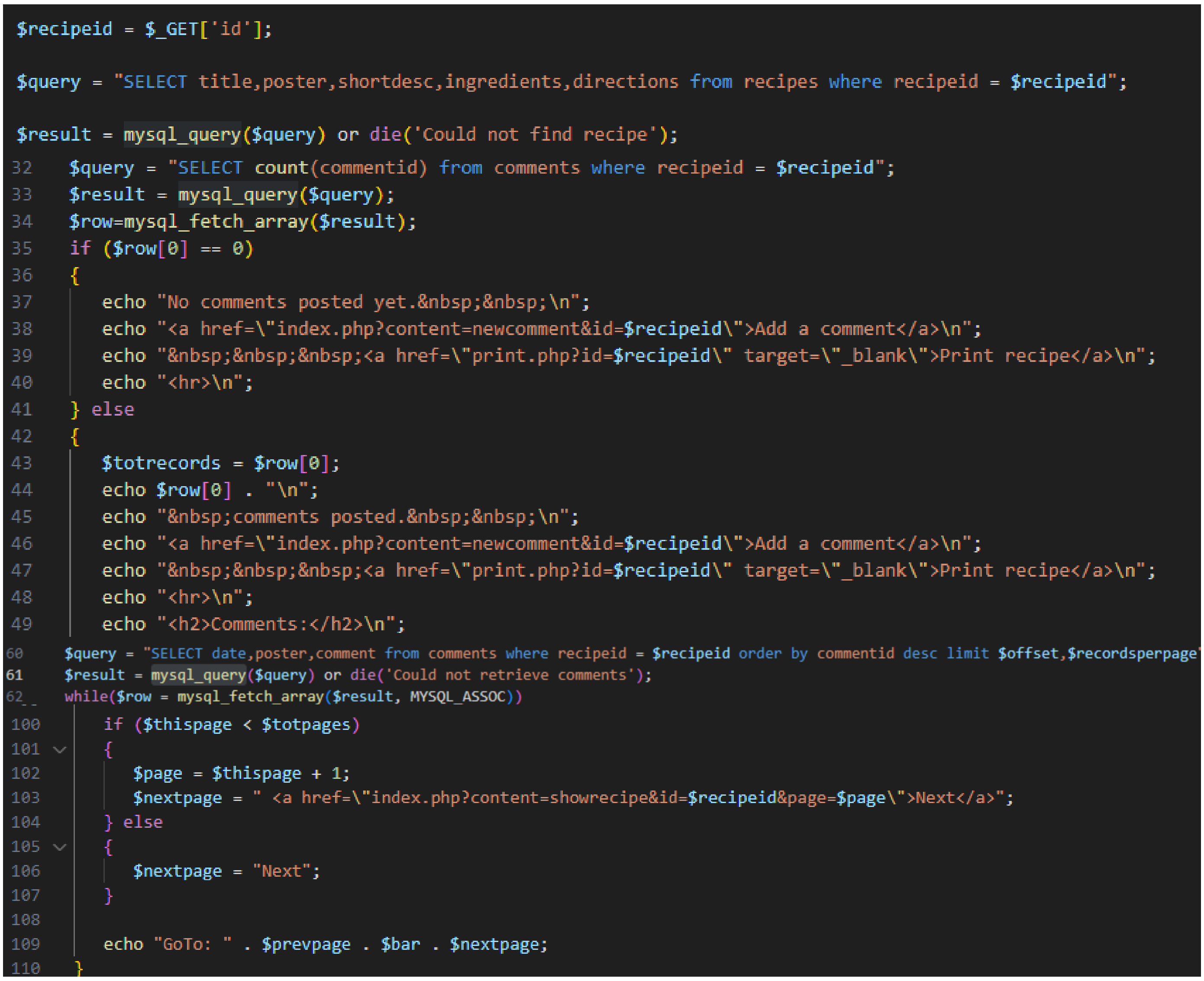
```php
        // Count comments for the recipe
        $query = "SELECT COUNT(commentid) FROM comments WHERE recipeid = ?";
        $stmt = $pdo->prepare($query);
        $stmt->execute([$recipeid]);
        $count = $stmt->fetchColumn();
```

```php
            // Fetch and output comments
            $query = "SELECT date, poster, comment FROM comments WHERE recipeid = ? ORDER BY commentid DESC";
            $stmt = $pdo->prepare($query);
            $stmt->execute([$recipeid]);
            $comments = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

File: showrecipe.inc.php

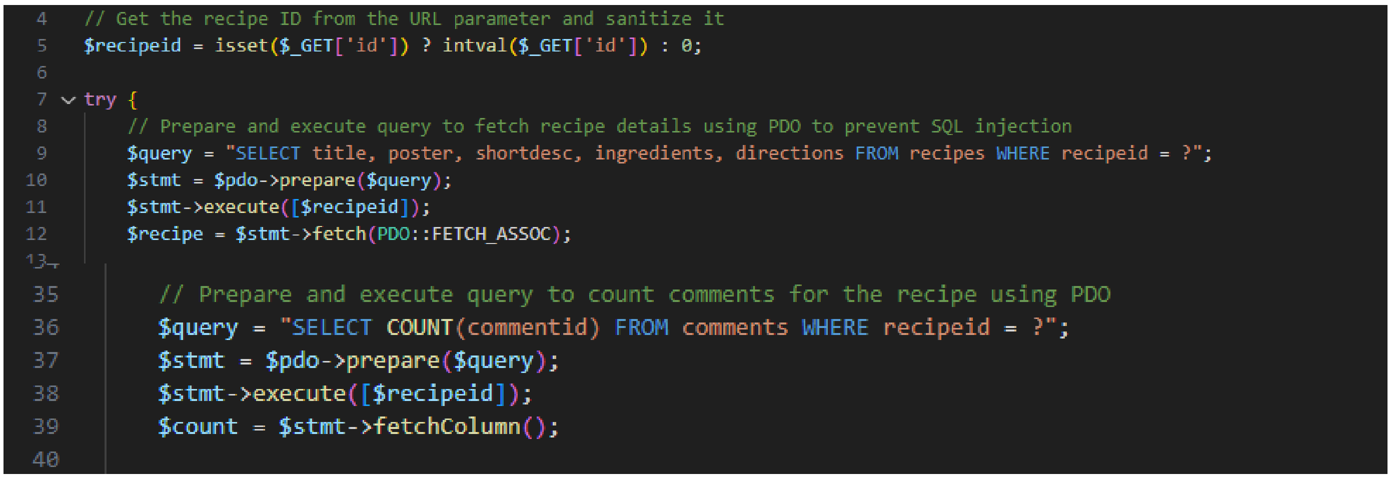
- Vulnerabilities: SQL injection (SQLi) and cross-site scripting (XSS)
- Risk Severity: High
- Likelihood of Exploitation: High
- Ease of Exploitation: Moderate
- Remediation Approach: Implement parameterized queries or prepared statements using PDO(PHP Data Objects) to prevent SQL injection. Sanitize user input using htmlspecialchars to prevent XSS attacks. Avoid storing credentials in clear text.

Vulnerable Code:

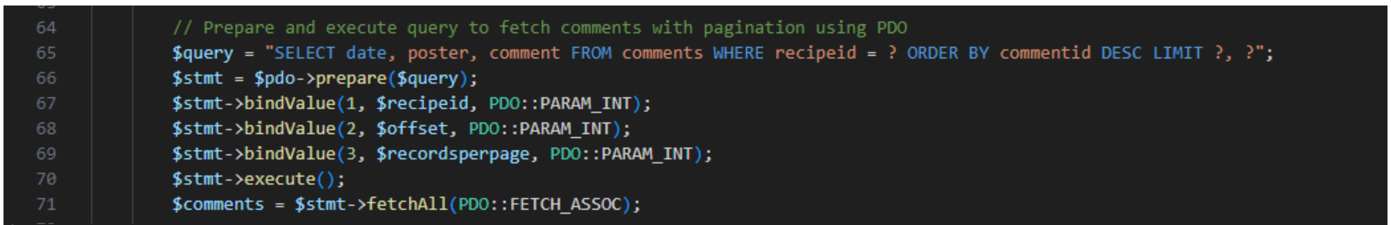


```php
$recipeid = $_GET['id'];

$query = "SELECT title,poster,shortdesc,ingredients,directions from recipes where recipeid = $recipeid";

$result = mysql_query($query) or die('Could not find recipe');
    $query = "SELECT count(commentid) from comments where recipeid = $recipeid";
    $result = mysql_query($query);
    $row=mysql_fetch_array($result);
    if ($row[0] == 0)
    {
        echo "No comments posted yet.&nbsp;&nbsp;\n";
        echo "<a href=\"index.php?content=newcomment&id=$recipeid\">Add a comment</a>\n";
        echo "&nbsp;&nbsp;&nbsp;<a href=\"print.php?id=$recipeid\" target=\"_blank\">Print recipe</a>\n";
        echo "<hr>\n";
    } else
    {
        $totrecords = $row[0];
        echo $row[0] . "\n";
        echo "&nbsp;comments posted.&nbsp;&nbsp;\n";
        echo "<a href=\"index.php?content=newcomment&id=$recipeid\">Add a comment</a>\n";
        echo "&nbsp;&nbsp;&nbsp;<a href=\"print.php?id=$recipeid\" target=\"_blank\">Print recipe</a>\n";
        echo "<hr>\n";
        echo "<h2>Comments:</h2>\n";
    $query = "SELECT date,poster,comment from comments where recipeid = $recipeid order by commentid desc limit $offset,$recordsperpage
    $result = mysql_query($query) or die('Could not retrieve comments');
    while($row = mysql_fetch_array($result, MYSQL_ASSOC))
        if ($thispage < $totpages)
        {
            $page = $thispage + 1;
            $nextpage = " <a href=\"index.php?content=showrecipe&id=$recipeid&page=$page\">Next</a>";
        } else
        {
            $nextpage = "Next";
        }

        echo "GoTo: " . $prevpage . $bar . $nextpage;
    }
```

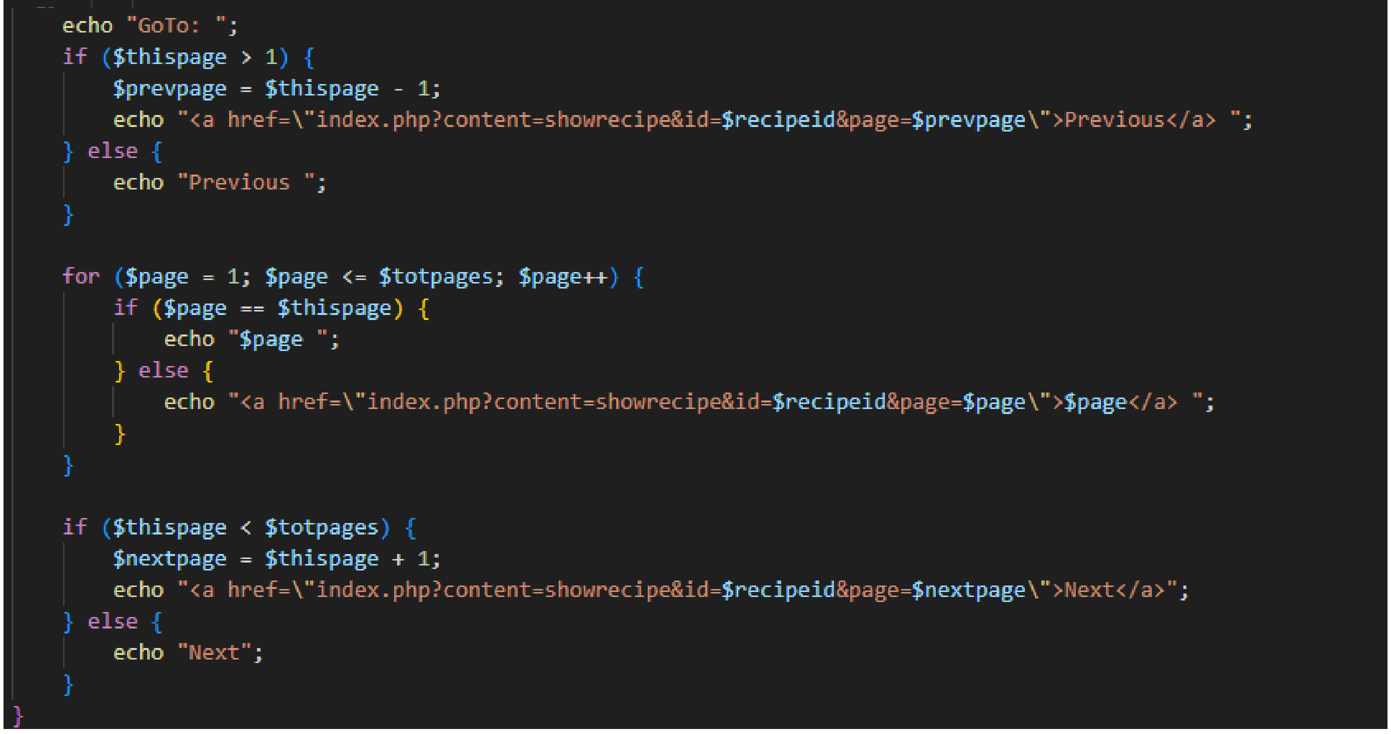
Fixed Code:



```php
// Get the recipe ID from the URL parameter and sanitize it
$recipeid = isset($_GET['id']) ? intval($_GET['id']) : 0;

try {
    // Prepare and execute query to fetch recipe details using PDO to prevent SQL injection
    $query = "SELECT title, poster, shortdesc, ingredients, directions FROM recipes WHERE recipeid = ?";
    $stmt = $pdo->prepare($query);
    $stmt->execute([$recipeid]);
    $recipe = $stmt->fetch(PDO::FETCH_ASSOC);

        // Prepare and execute query to count comments for the recipe using PDO
        $query = "SELECT COUNT(commentid) FROM comments WHERE recipeid = ?";
        $stmt = $pdo->prepare($query);
        $stmt->execute([$recipeid]);
        $count = $stmt->fetchColumn();
```



```
        // Prepare and execute query to fetch comments with pagination using PDO
        $query = "SELECT date, poster, comment FROM comments WHERE recipeid = ? ORDER BY commentid DESC LIMIT ?, ?";
        $stmt = $pdo->prepare($query);
        $stmt->bindValue(1, $recipeid, PDO::PARAM_INT);
        $stmt->bindValue(2, $offset, PDO::PARAM_INT);
        $stmt->bindValue(3, $recordsperpage, PDO::PARAM_INT);
        $stmt->execute();
        $comments = $stmt->fetchAll(PDO::FETCH_ASSOC);
```



```
    echo "GoTo: ";
    if ($thispage > 1) {
        $prevpage = $thispage - 1;
        echo "<a href=\"index.php?content=showrecipe&id=$recipeid&page=$prevpage\">Previous</a> ";
    } else {
        echo "Previous ";
    }

    for ($page = 1; $page <= $totpages; $page++) {
        if ($page == $thispage) {
            echo "$page ";
        } else {
            echo "<a href=\"index.php?content=showrecipe&id=$recipeid&page=$page\">$page</a> ";
        }
    }

    if ($thispage < $totpages) {
        $nextpage = $thispage + 1;
        echo "<a href=\"index.php?content=showrecipe&id=$recipeid&page=$nextpage\">Next</a>";
    } else {
        echo "Next";
    }
}
```

File: validate.inc.php

- Vulnerabilities: SQL injection (SQLi)
- Risk Severity: High
- Likelihood of Exploitation: High
- Ease of Exploitation: Moderate
- Remediation Approach: Implement parameterized queries or prepared statements using PDO(PHP Data Objects) to prevent SQL injection.

Vulnerable Code:

```
$userid = $_POST['userid'];
$password = $_POST['password'];

$query = "SELECT userid from users where userid = '$userid' and password = PASSWORD('$password')";
$result = mysql_query($query);

if (mysql_num_rows($result) == 0)
{
    echo "<h2>Sorry, your user account was not validated.</h2><br>\n";
    echo "<a href=\"index.php?content=login\">Try again</a><br>\n";
    echo "<a href=\"index.php\">Return to Home</a>\n";
} else
{
    $_SESSION['valid_recipe_user'] = $userid;
    echo "<h2>Your user account has been validated, you can now post recipes and comments</h2><br>\n";
    echo "<a href=\"index.php\">Return to Home</a>\n";
}
?>
</div></div>
```

Fixed Code:

```
try {
    // Prepare and execute query using PDO to prevent SQL injection
    $query = "SELECT userid FROM users WHERE userid = :userid AND password = PASSWORD(:password)";
    $stmt = $pdo->prepare($query);
    $stmt->bindParam(':userid', $userid, PDO::PARAM_STR);
    $stmt->bindParam(':password', $password, PDO::PARAM_STR);
    $stmt->execute();

    // Check if user exists
    if ($stmt->rowCount() == 0) {
        echo "<h2>Sorry, your user account was not validated.</h2><br>\n";
        echo "<a href=\"index.php?content=login\">Try again</a><br>\n";
        echo "<a href=\"index.php\">Return to Home</a>\n";
    } else {
        // Start session and set user as validated
        session_start();
        $_SESSION['valid_recipe_user'] = $userid;
        echo "<h2>Your user account has been validated, you can now post recipes and comments</h2><br>\n";
        echo "<a href=\"index.php\">Return to Home</a>\n";
    }
} catch (PDOException $e) {
    // If an error occurs, display error message
    die("Error: " . $e->getMessage());
}
```

Recommendations

1. **Implement Parameterized Queries:** Use parameterized queries or prepared statements with PDO to prevent SQL injection attacks. This approach helps separate SQL logic from user input, reducing the risk of injection vulnerabilities.
2. **Sanitize User Input:** Sanitize user input before displaying it on web pages to prevent cross-site scripting (XSS) attacks. Utilize functions like `htmlspecialchars()` to encode special characters and prevent script injection.
3. **Avoid Hard-Coded Credentials:** Remove hard-coded credentials and sensitive information from the source code. Instead, use environment variables or secure configuration files to store credentials securely.

4. **Update Deprecated Functions:** Replace deprecated functions like `mysql_connect()` with more secure alternatives such as PDO for database connectivity. Deprecated functions may have security vulnerabilities and lack support in newer PHP versions.
5. **Prevent File Inclusion Vulnerabilities:** Avoid using user-controlled input in file inclusion functions to prevent remote file inclusion (RFI) and local file inclusion (LFI) attacks. Implement whitelisting or input validation to restrict file paths and prevent unauthorized access to sensitive files.
6. **Regular Security Updates:** Stay informed about security updates and patches for PHP, MySQL, and other dependencies used in the application. Regularly update the application and its components to mitigate security risks associated with known vulnerabilities.

Conclusion

The security review of ReciPHP identified several vulnerabilities, including SQL injection, cross-site scripting, hard-coded credentials, and file inclusion. By implementing the recommended secure coding practices outlined above, the application can significantly improve its security posture and better protect user data against potential threats. Continuous monitoring and proactive security measures are essential to maintaining the security and integrity of the ReciPHP application.

References:

- ReciPHP Source-Code
- TryHackMe SAST Room
- Pentesterlab Code Review
- secure-code-review-checklist
- PHP-vulnerability-audit-cheatsheet
- php-pdo-prepared-statements-to-prevent-sql-injection
- OWASP TOP TEN
- OWASP Security Code Review 101
- OWASP SQL Injection Prevention Cheat Sheet
- OWASP XSS (Cross Site Scripting) Prevention Cheat Sheet
- OWASP PHP Configuration Cheat Sheet