Code hiện tại của bạn có lỗ hổng bảo mật liên quan đến **SQL Injection**, một trong những lỗ hổng phổ biến và nguy hiểm. Lỗ hổng này cho phép kẻ tấn công có thể thực thi các câu lệnh SQL tùy ý trên cơ sở dữ liệu của bạn thông qua việc nhập liệu không an toàn vào form tìm kiếm. Điều này có thể dẫn đến việc tiết lộ thông tin nhạy cảm, thay đổi hoặc xóa dữ liệu, hoặc thực hiện các hành vi độc hại khác.

**Sửa Lỗi và Giải Pháp**

Để khắc phục lỗ hổng SQL Injection, bạn cần sử dụng **Prepared Statements** với **mysqli**, cách này không chỉ đơn giản hóa việc viết mã mà còn cung cấp một lớp bảo vệ an toàn chống lại SQL injection. Dưới đây là phiên bản đã được sửa đổi của đoạn code PHP:

<?php

    if (!empty($\_POST["submit"])) {

        $search = $\_POST["content\_search"];

        // Kết nối đến cơ sở dữ liệu

        $dbh = new *mysqli*('localhost', 'root', 'hahien', 'sqlinjection');

        // Kiểm tra kết nối

        if ($dbh->connect\_error) {

            die("Lỗi kết nối: " . $dbh->connect\_error);

        }

        // Câu lệnh SQL sử dụng Prepared Statements

        $stmt = $dbh->prepare('SELECT \* FROM book WHERE BOOK = ?');

        $stmt->bind\_param('s', $search);

        // Thực thi truy vấn

        $stmt->execute();

        $result = $stmt->get\_result();

        // Hiển thị kết quả

        if ($result) {

            echo '<table border="1" width="100%" id="tab">

                    <thead>

                        <th>ID</th>

                        <th>BOOK</th>

                        <th>AUTHOR</th>

                        <th>DESCRIPTION</th>

                    </thead>

                    <tbody>';

            while ($row = $result->fetch\_assoc()) {

                echo '<tr>

                        <td>' . htmlspecialchars($row['ID']) . '</td>

                        <td>' . htmlspecialchars($row['BOOK']) . '</td>

                        <td>' . htmlspecialchars($row['AUTHOR']) . '</td>

                        <td>' . htmlspecialchars($row['DESCRIPTION']) . '</td>

                      </tr>';

            }

            echo '</tbody></table>';

        } else {

            echo "Lỗi truy vấn: " . $dbh->error;

        }

        // Đóng kết nối

        $stmt->close();

        $dbh->close();

    }

?>

**Lý do thay đổi:**

* **Sử dụng Prepared Statements**: Điều này bảo vệ bạn khỏi SQL injection bởi vì giá trị được gửi đến cơ sở dữ liệu không bao giờ được giải thích làm SQL. Mọi giá trị được xử lý như là dữ liệu thô, không có khả năng thực thi.
* **htmlspecialchars**: Hàm này được sử dụng khi hiển thị dữ liệu từ cơ sở dữ liệu ra trang web để ngăn chặn các cuộc tấn công Cross-Site Scripting (XSS), đảm bảo rằng dữ liệu được hiển thị sẽ không chứa các mã độc hại.

Dưới đây là các lỗ hổng bảo mật chính trong dự án "Broken\_Authentication" và cách khắc phục chúng.

**1. Lỗ hổng "Broken Authentication"**

**Mô tả lỗ hổng**

Ứng dụng không sử dụng phương pháp bảo mật mạnh để bảo vệ thông tin đăng nhập của người dùng. Mật khẩu được lưu trữ không mã hóa, dễ dàng bị tấn công.

**Lỗ hổng trong code**

* **broken\_auth.sql**:

INSERT INTO `user` VALUES (1,'admin','admin'),(2,'user','user');

Mật khẩu được lưu trữ ở dạng plaintext.

* **class/user.php**:

public function login($username, $password)

        {

*$this*->connect();

            $pdo = *$this*->pdo;

            $sql = $pdo->prepare('SELECT USER,PASS FROM user');

            $sql->execute();

            $user = $sql->fetchAll();

            foreach ($user as $key => $value) {

                if ($value[0] == $username and $value[1] == $password) {

                    return true;

                }

            }

            return false;

Kiểm tra đăng nhập bằng cách so sánh trực tiếp mật khẩu plaintext.

**Cách khắc phục**

* Sử dụng hàm hash mật khẩu khi lưu trữ và kiểm tra đăng nhập.

    public function login($username, $password) {

        $stmt = *$this*->conn->prepare("SELECT \* FROM user WHERE USER = ?");

        $stmt->bind\_param("s", $username);

        $stmt->execute();

        $user = $stmt->get\_result()->fetch\_assoc();

        if ($user && password\_verify($password, $user['PASS'])) {

            return $user;

        }

        return null;

    }

Cập nhật **broken\_auth.sql** để lưu trữ mật khẩu đã được hash.

INSERT INTO `user` VALUES (1,'admin','hashed\_admin\_password'),(2,'user','hashed\_user\_password');

**2. Credential Stuffing Attack**

**Mô tả lỗ hổng**

Ứng dụng dễ bị tấn công Credential Stuffing, nơi kẻ tấn công thử hàng loạt tên người dùng và mật khẩu để tìm kiếm cặp thông tin đúng, tận dụng việc nhiều người dùng sử dụng cùng mật khẩu trên nhiều trang web.

**Cách khắc phục**

* **Giới hạn số lần đăng nhập thất bại:** Thiết lập giới hạn số lần đăng nhập thất bại trong một khoảng thời gian nhất định.

    session\_start();

    if (!isset($\_SESSION['login\_attempts'])) {

        $\_SESSION['login\_attempts'] = 0;

    }

    if ($\_SESSION['login\_attempts'] > 5) {

        echo "Too many login attempts. Please try again later.";

        exit;

    }

    if ($user = *$this*->login($\_POST['username'], $\_POST['password'])) {

        $\_SESSION['login\_attempts'] = 0;

        // Đăng nhập thành công

    } else {

        $\_SESSION['login\_attempts']++;

        echo "Invalid username or password.";

    }

**Sử dụng CAPTCHA:** Thêm CAPTCHA để ngăn chặn tấn công brute-force.

    <form action="#" method="POST" enctype="multipart/form-data" id="main-form">

    <p>Select image to upload</p>

    <input type="file" name="file\_upload">

    <input type="submit" name="submit" value="Upload">

    <!-- Thêm CAPTCHA ở đây -->

    <div *class*="g-recaptcha" data-sitekey="your\_site\_key"></div>

    </form>

* **Áp dụng xác thực đa yếu tố (MFA):** Yêu cầu người dùng nhập mã xác thực từ điện thoại hoặc email sau khi nhập mật khẩu.

**Tổng kết**

* **Broken Authentication:**
  + Lưu trữ mật khẩu dưới dạng hash (bcrypt).
  + So sánh mật khẩu nhập vào với mật khẩu hash trong cơ sở dữ liệu.
* **Credential Stuffing:**
  + Giới hạn số lần đăng nhập thất bại.
  + Sử dụng CAPTCHA.
  + Áp dụng xác thực đa yếu tố (MFA).

Áp dụng những biện pháp này sẽ giúp cải thiện bảo mật cho ứng dụng của bạn và giảm thiểu rủi ro bị tấn công.

Tương tự như khắc phục lỗ hổng SQL Injection, với lỗ hổng trong code tải lên tập tin của bạn, bạn cũng cần áp dụng những cải tiến để đảm bảo an toàn. Code hiện tại có lỗ hổng trong việc kiểm tra loại tập tin tải lên chỉ dựa vào MIME type, điều này không đủ để bảo đảm rằng tập tin được tải lên là an toàn và không chứa mã độc. Ngoài ra, code không kiểm tra kích thước tập tin và không xử lý các trường hợp tên tập tin có thể chứa các ký tự đặc biệt có thể được sử dụng để thực hiện các cuộc tấn công như Path Traversal.

Dưới đây là phiên bản cải tiến của đoạn code cho phần tải lên tập tin, áp dụng các biện pháp bảo mật tốt hơn:

<!DOCTYPE *html*>

<html *lang*="en">

<head>

    <link *rel*="stylesheet" *type*="text/css" *href*="../css/style\_fileUpload.css">

    <meta *charset*="UTF-8">

    <title>PHP\_File-Upload</title>

</head>

<body>

    <div *class*="menu">

        <ul>

            <li><a *href*="?action=image">Upload file image</a></li>

            <li><a *href*="?action=zip">Upload file zip</a></li>

            <li><a *href*="upload/">Upload</a></li>

        </ul>

    </div>

    <div *class*="content">

        <form *action*="#" *method*="POST" *enctype*="multipart/form-data" *id*="main-form">

            <p>Select image to upload</p>

            <input *type*="file" *name*="file\_upload" *accept*="image/jpeg, image/png, image/jpg">

            <input *type*="submit" *name*="submit" *value*="Upload">

        </form>

        <?php

            if (isset($\_POST["submit"])) {

                $target\_dir = "upload/";

                $file\_name = basename($\_FILES["file\_upload"]["name"]);

                $target\_file = $target\_dir . $file\_name;

                $upload\_size = $\_FILES["file\_upload"]["size"];

                $upload\_type = $\_FILES["file\_upload"]["type"];

                // Validate file extension

                $allowed\_types = array('jpg', 'jpeg', 'png');

                $file\_extension = pathinfo($file\_name, PATHINFO\_EXTENSION);

                if (in\_array(strtolower($file\_extension), $allowed\_types)) {

                    if ($upload\_size < 5000000) { // Limit the file size to 5MB

                        if (move\_uploaded\_file($\_FILES["file\_upload"]["tmp\_name"], $target\_file)) {

                            echo htmlspecialchars($target\_file) . " successfully uploaded!";

                        } else {

                            echo "File Error: Could not move the file.";

                        }

                    } else {

                        echo "File Error: File size is too large.";

                    }

                } else {

                    echo "File Error: Only JPEG, PNG and JPG files are allowed.";

                }

            }

        ?>

    </div>

</body>

</html>

**Các Thay Đổi và Lý Do:**

* **Kiểm tra Phần Mở Rộng Tập Tin ($file\_extension)**: Điều này giúp đảm bảo chỉ những tập tin có định dạng được cho phép mới được xử lý, bảo vệ hệ thống chống lại việc tải lên tập tin nguy hiểm.
* **Kiểm Tra Kích Thước Tập Tin ($upload\_size)**: Giới hạn kích thước tập tin giúp ngăn chặn các cuộc tấn công DoS thông qua tải lên tập tin quá lớn.
* **Sử Dụng move\_uploaded\_file**: Hàm này đảm bảo tập tin được tải lên qua HTTP POST, điều này là cần thiết để ngăn ngừa các lỗ hổng như tập tin được tải từ các nguồn không đáng tin cậy.
* **Xử Lý Tên Tập Tin**: Dùng **basename()** để lấy tên tập tin và loại bỏ các đường dẫn tương đối có thể dẫn đến lỗ hổng Path Traversal.
* **HTML Escaping When Outputting File Paths**: Sử dụng **htmlspecialchars()** khi hiển thị đường dẫn tập tin để tránh XSS.

Những thay đổi này tăng cường bảo mật cho phần tải tập tin và giúp bảo vệ ứng dụng web khỏi các mối đe dọa an ninh phổ biến.