Отчет по аудиту безопасности приложения

Автор: Давид Гуликян

Дата: 5 мая 2025 года

Введение

Цель данного аудита — выявить и устранить уязвимости в веб-приложении, включая XSS, Information Disclosure, SQL Injection, CSRF, Include и Upload. В отчете описаны найденные уязвимости, методы защиты и примеры исправленного кода.

XSS (Cross-Site Scripting)

Проблема: XSS позволяет злоумышленнику внедрить вредоносный скрипт, который выполняется в браузере пользователя. В исходном коде данные пользователя (например, fio, bio) выводились без экранирования, что позволяло вставить <script>alert('hack')</script>.

Решение: Все данные, выводимые на страницу, экранируются с помощью функции htmlspecialchars() в PHP, чтобы предотвратить выполнение скриптов.

Пример кода:

До исправления (admin.php):

<td><?php echo $app['fio']; ?></td>

После исправления:

<td><?php echo htmlspecialchars($app['fio'], ENT_QUOTES, 'UTF-8'); ?></td>

Information Disclosure

Проблема: Утечка конфиденциальной информации через ошибки подключения к базе данных. Например, в db_connect.php выводилось сообщение die('Ошибка подключения: ' . $e->getMessage()), раскрывающее детали сервера.

Решение: Отключено отображение ошибок с помощью ini_set('display_errors', 0), а ошибки записываются в лог-файл. Вместо детализированного сообщения выводится общее уведомление.

Пример кода:

До исправления (db_connect.php):

die('Ошибка подключения: ' . $e->getMessage());

После исправления:

ini_set('display_errors', 0);

error_log($e->getMessage(), 3, 'errors.log');

die('Произошла ошибка. Обратитесь к администратору.');

SQL Injection

Проблема: SQL-инъекции позволяют злоумышленнику выполнять произвольные SQL-запросы через уязвимые поля ввода. В исходном коде уже использовались подготовленные запросы PDO, что предотвращает эту уязвимость.

Решение: Подтверждено, что все запросы используют prepare и execute с параметрами, что исключает инъекции.

Пример кода:

Пример безопасного запроса (admin.php):

$stmt = $pdo->prepare('SELECT password_hash FROM admins WHERE login = ?');

$stmt->execute([$admin_login]);

CSRF (Cross-Site Request Forgery)

Проблема: CSRF позволяет злоумышленнику выполнять действия от имени авторизованного пользователя. Формы в приложении не были защищены, что делало их уязвимыми.

Решение: Добавлены CSRF-токены для проверки подлинности запросов. Токен генерируется и проверяется при отправке формы.

Пример кода:

Генерация токена (form.php):

$_SESSION['csrf_token'] = bin2hex(random_bytes(32));

echo '<input type="hidden" name="csrf_token" value="' . $_SESSION['csrf_token'] . '">';

Проверка токена:

```
if (!isset($_POST['csrf_token']) || $_POST['csrf_token'] !==
$_SESSION['csrf_token']) {
    die('Неверный CSRF-токен');
}
```

Include и Upload

Проблема: Include-уязвимость связана с включением произвольных файлов, Upload — с загрузкой вредоносных файлов. В приложении отсутствует функционал загрузки файлов, поэтому Upload-уязвимости нет. Для Include использовались фиксированные пути, но без проверки.

Решение: Добавлена проверка существования файлов перед их включением с помощью file_exists().

Пример кода:

До исправления (form.php):

```
require_once 'db_connect.php';
```

После исправления:

```
$file = 'db_connect.php';
if (file_exists($file)) {
    require_once $file;
} else {
    die('Файл не найден');
}
```

Заключение

В ходе аудита устранены уязвимости XSS, Information Disclosure, SQL Injection, CSRF и Include. Код доступен в репозитории: https://github.com/davidgulikan/web7.

Все изменения протестированы, приложение стало безопаснее.