Отчет об аудите

1. Cross-Site Scripting (XSS)

Уязвимость: отсутствие экранирования вывода данных из базы данных может привести к внедрению вредоносного JavaScript-кода.

```
echo '<tr>';
echo '<td>' . $userData['names'] . '</td>';
echo '<td>' . $userData['phones'] . '</td>';
echo '<td>' . $userData['email'] . '</td>';
echo '<td>' . $userData['dates'] . '</td>';
echo '<td>' . $userData['gender'] . '</td>';
echo '<td>' . $userData['biography'] . '</td>';
```

Решение: использование функции htmlspecialchars для экранирования данных перед выводом. (admin.php)

```
echo '<tr>';
echo '<td>' . htmlspecialchars($userData['names'], ENT_QUOTES, 'UTF-8') . '</td>';
echo '<td>' . htmlspecialchars($userData['phones'], ENT_QUOTES, 'UTF-8') . '</td>';
echo '<td>' . htmlspecialchars($userData['email'], ENT_QUOTES, 'UTF-8') . '</td>';
echo '<td>' . htmlspecialchars($userData['dates'], ENT_QUOTES, 'UTF-8') . '</td>';
echo '<td>' . htmlspecialchars($userData['gender'], ENT_QUOTES, 'UTF-8') . '</td>';
echo '<td>' . htmlspecialchars($userData['biography'], ENT_QUOTES, 'UTF-8') . '</td>';
```

2. Information Disclosure

Уязвимость: приложение может раскрыть внутреннюю информацию (например, ошибки базы данных), которая может быть использована злоумышленниками для проведения атак.

```
if (!isset($_GET['id'])) {
    echo "Ошибка: ID пользователя не указан.";
    exit();
}

$stmt = $db->prepare("SELECT * FROM application WHERE id = ?");
$stmt->execute([$_GET['id']]);
$userData = $stmt->fetch(PDO::FETCH_ASSOC);

if (!$userData) {
    echo "Пользователь с указанным ID не найден.";
    exit();
}
```

Решение: включение пользовательских сообщений об ошибках вместо отображения внутренних ошибок. Логирование ошибок для внутреннего аудита вместо их отображения пользователю. (edit_user.php)

```php
if (!isset($_GET['id'])) {
    error_log("User ID not provided in request.");
    echo "Ошибка: ID пользователя не указан.";
    exit();
}

$stmt = $db->prepare("SELECT * FROM application WHERE id = ?");
$stmt->execute([$_GET['id']]);
$userData = $stmt->fetch(PDO::FETCH_ASSOC);

if (!$userData) {
    error_log("User with ID {$_GET['id']} not found.");
    echo "Пользователь с указанным ID не найден.";
    exit();
}
```

3. SQL Injection

Уязвимость: SQL Injection - это метод атаки, при котором злоумышленник может вставить или "инжектировать" свой собственный SQL-код в запросы к базе данных, что может привести к утечке данных, их изменению или удалению.

Решение: в текущем коде уже используется подготовленный запрос для выборки данных пользователя по логину: (login.php)

```php
$stmt = $db->prepare("SELECT id, pass FROM login_pass WHERE login = ?");
$stmt -> execute([$_POST['login']]);
```

Этот подход обеспечивает защиту от SQL Injection, так как параметры запроса передаются отдельно и обрабатываются драйвером базы данных.

4. CSRF (Cross-Site Request Forgery)

Уязвимость: отсутствие защиты от CSRF может позволить злоумышленнику выполнить нежелательные действия от имени пользователя.

```php
if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST['update'])) {

    $stmt = $db->prepare("UPDATE application SET names = ?, phones = ?, email = ?, dates = ?, gender = ?, biography = ? WHERE id = ?");
    $stmt->execute([
        $_POST['names'],
        $_POST['phones'],
        $_POST['email'],
        $_POST['dates'],
        $_POST['gender'],
        $_POST['biography'],
        $_GET['id']
    ]);

    header("Location: admin.php");
    exit();
}
```

Решение: использование CSRF-токенов для всех форм, выполняющих изменения данных. (admin.php)

```php
session_start();
if (empty($_SESSION['csrf_token'])) {
    $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}


session_start();
if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST['update'])) {
    if (!hash_equals($_SESSION['csrf_token'], $_POST['csrf_token'])) {
        die("Ошибка CSRF: недопустимый токен.");
    }

    $stmt = $db->prepare("UPDATE application SET names = ?, phones = ?, email = ?, dates = ?, gender = ?, biography = ? WHERE id = ?");
    $stmt->execute([
        htmlspecialchars($_POST['names']),
        htmlspecialchars($_POST['phones']),
        htmlspecialchars($_POST['email']),
        htmlspecialchars($_POST['dates']),
        htmlspecialchars($_POST['gender']),
        htmlspecialchars($_POST['biography']),
        htmlspecialchars($_GET['id'])
    ]);

    header("Location: admin.php");
    exit();
}
```

5. Include (Local File Inclusion)

Уязвимость: использование функции include с внешними данными может привести к атакам LFI (Local File Inclusion), что позволяет злоумышленникам включить произвольные файлы.

```php
include('form.php');
```

Решение: Использование жестко заданных путей для включаемых файлов. Например, при динамическом включении файлов, убедитесь, что включаются только файлы из разрешенного списка. (index.php)

```php
    $allowed_files = ['form.php', 'another_form.php'];
if (in_array($file, $allowed_files)) {
    include($file);
} else {
    echo 'Недопустимый файл для включения';
}
```

6. Upload

Уязвимость: возникает, когда пользователи могут загружать файлы на сервер без должной проверки и ограничения. Это может привести к загрузке и выполнению вредоносного кода на сервере.

Решение: в текущем коде функциональность загрузки файлов отсутствует. Однако, если такая функциональность будет добавлена, необходимо реализовать следующие меры безопасности: ограничить типы файлов, которые могут быть загружены (например, только изображения), проверять MIME-тип файла и его расширение, использовать уникальные имена файлов для предотвращения перезаписи существующих файлов, хранить загруженные файлы вне корневого каталога веб-сервера, установить ограничения на размер загружаемых файлов.