Поспелова Тоня, 27/1

Отчет об аудите приложения

1. **Защита от XSS**

В моем приложении можно заметить, что защита от XSS не везде реализована. Для повышения уровня защиты от XSS можно использовать фильтрацию данных, приведение к типу и экранирование.

1) Фильтрация ввода данных

Фильтрация данных поможет исключить нежелательные символы и значения. Используем preg_match и strip_tags.

2) Приведение к типу

Приведение к типу поможет убедиться, что данные имеют правильный формат.

3) Экранирование вывода данных

Использование htmlspecialchars для экранирования вывода данных, чтобы предотвратить выполнение вредоносных скриптов.

В файле functions.php в функции valid() добавлена проверка на существование массива выбранных языков и наличия исключительно числовых значений в нем.

```php
if (empty($_POST['languages']) || !is_array($_POST['languages'])) {
  setcookie('languages_error', '1', time() + 24 * 60 * 60);
  $errors = TRUE;
} else {
  foreach ($_POST['languages'] as $language) {
    if (!is_numeric($language)) {
      $errors['languages'] = true;
      break;
    }
```

В файле index.php улучшен вывод сообщений

```php
if (!empty($_COOKIE['pass'])) {
  $messages[] = htmlspecialchars('<a href="login.php">Войдите</a> с логином <strong>' .
  $_COOKIE['login'] . '</strong> и паролем <strong>' . $_COOKIE['pass'] .
  '</strong> для изменения данных.');
}
```

```php
if ($errors['fio']) {
  setcookie('fio_error', '', 100000);
  $messages[] = htmlspecialchars('<div class="error">Заполните имя.</div>');
}
if ($errors['email']) {
  setcookie('email_error', '', 100000);
  $messages[] = htmlspecialchars('<div class="error">Заполните почту.</div>');
}
```

и т.д.

Для безопасной авторизации добавлено в файл admin.php

```php
$username = htmlspecialchars($_SERVER['PHP_AUTH_USER']);
$password = htmlspecialchars($_SERVER['PHP_AUTH_PW']);
```

Также изменен вывод данных из базы

```php
while($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
    echo "<tr>";
    echo "<td>" . intval($row["id"]) . "</td>";
    echo "<td>" . htmlspecialchars($row["names"]) . "</td>";
    echo "<td>" . htmlspecialchars($row["tel"]) . "</td>";
    echo "<td>" . htmlspecialchars($row["email"]) . "</td>";
    echo "<td>" . htmlspecialchars($row["dateB"]) . "</td>";
    echo "<td>" . htmlspecialchars($row["gender"]) . "</td>";
    echo "<td>" . htmlspecialchars($row["biography"]) . "</td>";
    echo "<td>" . htmlspecialchars($row["languages"]) . "</td>";
    echo "<td><a href='edit.php?id=" . intval($row["id"]) . "'>Изменить</a></td>";
    echo "<td><a href='delete.php?id=" . intval($row["id"]) . "'>Удалить</a></td>";
    echo "</tr>";
}
```

В файле delete.php добавлено приведение идентификатора к числовому типу и безопасное удаление записи

```php
if(isset($_GET['id']) && is_numeric($_GET['id'])) {
    $id = intval($_GET['id']);
```

2. Защита от Information Disclosure

Для обеспечения защиты от утечки информации в вашем приложении необходимо исправить сообщения об ошибках. Исключения и ошибки не должны раскрывать внутренние детали системы (например, структуры базы данных, пути файлов, версии ПО). Подробные сообщения об ошибках должны

быть видимы только администраторам, в то время как пользователям должна показываться общая информация.

Для этого в файлы index.php, admin.php и edit.php добавим следующий код:

```php
ini_set('display_errors', 0);
ini_set('log_errors', 1);
ini_set('error_log', 'error.log');
```

Также заменим все функции session_start, чтобы убедиться, что сессии защищены и имеют ограниченное время жизни

```php
session_start([
    'cookie_lifetime' => 86400,
    'read_and_close'  => true,
]);
```

3. Защита от SQL Injection

Улучшение защиты от SQL Injection вашего приложения могут включать использование библиотек абстракций, ORM, подготовленных запросов и правильное экранирование данных.

В моем приложении все SQL-запросы и подключения выполняются с использованием PDO, например:

```php
function connectToDatabase() {
    include '../4/p.php';
    $db = new PDO('mysql:host=127.0.0.1;dbname=u67314',
    $user, $pass, array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8"));
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    return $db;
}
```

```php
$stmt = $db->prepare("SELECT names, tel, email, dateB, gender, biography
FROM application WHERE id = ?");
$stmt->execute([$id]);
```

4. Защита от CSRF

Защита от CSRF (межсайтовой подделки запроса) предполагает использование POST-запросов для изменения данных на сервере, таких как добавление,

обновление или удаление записей, а GET-запросы для получения данных, что реализовано в моем приложении.

Реализация устаревания сессии с помощью параметра session.cookie_lifetime:

```
ini_set('session.cookie_lifetime', 3600);
```

Также можно внедрить токены CSRF для защиты от подделки межсайтовых запросов

Генерация и установка CSRF токена

```
if (!isset($_SESSION['csrf_token'])) {
  $_SESSION['csrf_token'] = bin2hex(random_bytes(32));
}
```

Вставка CSRF токена в форму

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

Проверка CSRF токена при обработке запроса

```
  if (!empty($_POST['csrf_token']) && $_POST['csrf_token'] !== $_SESSION['csrf_token']) {
    exit("CSRF атака обнаружена");
  }
```

5. **Защита от Include и Upload уязвимости**

Include уязвимость:
В моем приложении фунеции типа include, require, include_once, require_once не подконтрольны пользователю и путь к файлу не формируется на основе пользовательского ввода. Я использую только надежные относительные пути, например

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

Upload уязвимость:
В моем приложении нет мест, где пользователь может загружать файлы на сервер, поэтому данной уязвимости нет.