**Молдавский Государственный Университет**

**Факультет Математики и Информатики**

Специальность: Прикладная Информатика

Предмет: “Безопасность Web-Приложений”

**Индивидуальная работа**

**Выполнил:** Sirbu Alexandru

**Преподаватель:** Boris Visnevschi

Кишинёв 2024

# Описание

В качестве индивидуальной работы я создал приложение “Блог”, в котором можно создавать посты, создав при этом безопасную среду.

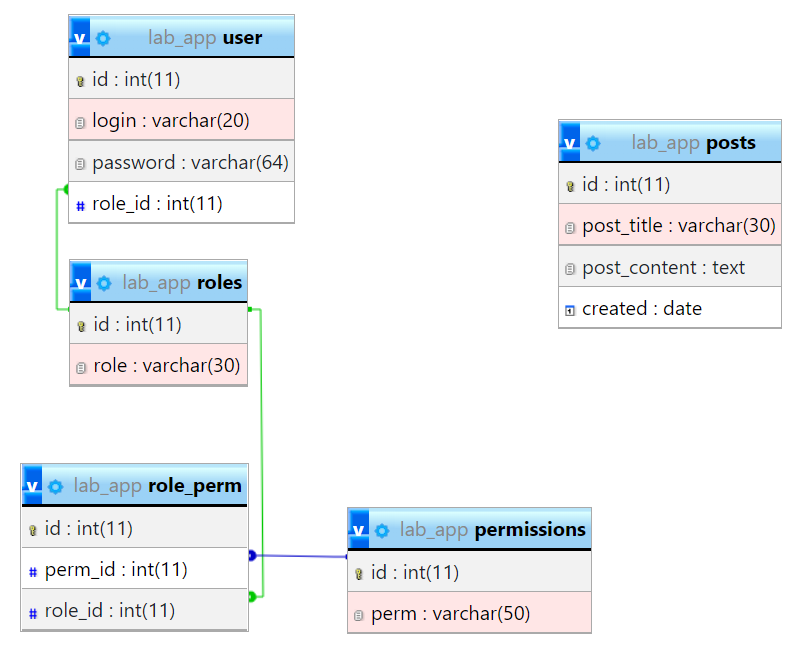
В моем приложении имеются 6 страниц:

* **Главная страница –** страница с постами веденного вами блога.
* **Страница регистрации** — форма с полями регистрации, где пользователь может создать аккаунт.
* **Страница авторизации** - форма с полями авторизации, где пользователь может войти в аккаунт.
* **Admin Panel** —панель администрирования, содержащая из двух страниц:
  + **Post Manager** – управление постами (админ создает и удаляет, менеджер только создает)
  + **User Manager** – управление ролями зарегистрированных аккаунтов (доступно только админу)

# Базы данных

Самая важная часть любого современного приложение – база данных. И конечно мы ее создадим.

Схема мой базы данных:



1. **Таблица user** — пользователи.
2. **Таблица roles** — список ролей пользователей.
3. **Таблица role\_perm** — разрешения для определенных ролей.
4. **Таблица permissions** — список разрешений.
5. **Таблица posts** – посты.

# Атаки веб – приложений

## **SQL-Injection**

Это один из распространённых способов взлома сайтов и программ, работающих с базами данных, основанный на внедрении в запрос произвольного SQL-кода.

Внедрение SQL, в зависимости от типа используемой СУБД и условий внедрения, может дать возможность атакующему выполнить произвольный запрос к базе данных (например, прочитать содержимое любых таблиц, удалить, изменить или добавить данные), получить возможность чтения и/или записи локальных файлов и выполнения произвольных команд на атакуемом сервере.

Атака типа внедрения SQL может быть возможна из-за некорректной обработки входных данных, используемых в SQL-запросах.

## **Защита от атак типа внедрение SQL-кода**

1. Фильтрация строковых параметров
2. Фильтрация целочисленных параметров
3. Усечение входных параметров
4. Использование параметризованных запросов



## **XSS-Injection**

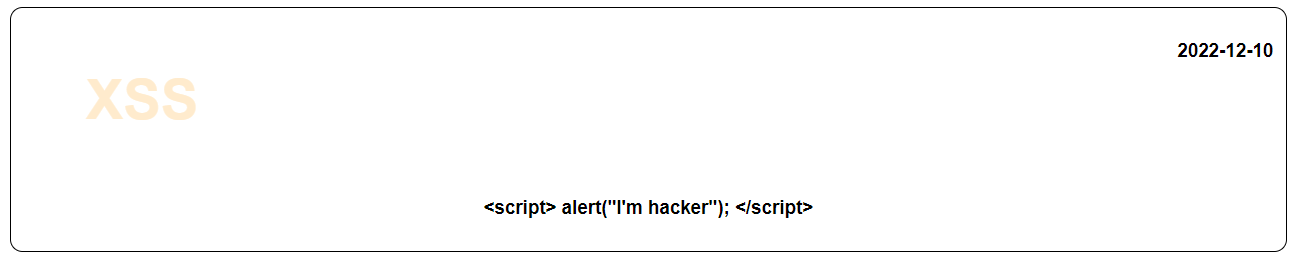
Тип атаки на веб-системы, заключающийся во внедрении в выдаваемую веб-системой страницу вредоносного кода (который будет выполнен на компьютере пользователя при открытии им этой страницы) и взаимодействии этого кода с веб-сервером злоумышленника. Является разновидностью атаки «Внедрение кода».

Специфика подобных атак заключается в том, что вредоносный код может использовать авторизацию пользователя в веб-системе для получения к ней расширенного доступа или для получения авторотационных данных пользователя. Вредоносный код может быть вставлен в страницу как через уязвимость в веб-сервере, так и через уязвимость на компьютере пользователя.

## **Защита от атак типа межсайтовый скриптинг**

1. Кодирование управляющих HTML-символов, JavaScript, CSS и URL перед отображением в браузере.
2. Кодирование входных данных.
3. Регулярный ручной и автоматизированный анализ безопасности кода и тестирование на проникновение.
4. Указание кодировки на каждой web-странице (например, ISO-8859-1 или UTF-8) до каких-либо пользовательских полей.





## **Authentication Bypass**

Обход аутентификации для сайтов и веб-приложений – это несанкционированный доступ к административному разделу или разделам сайта и скриптам, обеспечивающим прямое взаимодействие с базой данных и файловой системой сервера.

Authentication Bypass может быть выполнен, эксплуатируя уязвимости кода сайта, ошибки публикации ресурса, а также из-за ошибок в настройках и уязвимостями программного обеспечения сервера.

Возможность обхода аутентификации на сайте всегда приводит к его взлому, так как:

1. Атакующий проходит в административный раздел сайта с максимальным уровнем доступа
2. Атакующий получает доступ к закрытым разделам сайта, или файлам, напрямую взаимодействующим с базой данных или файловой системой сервера

## **Защита от обхода аутентификации**

1. Правильное использование заголовков HTTP
2. Использование ролей, при которых разные пользователи имеют нужные им роли, которые имеют различные привилегии, без которых нельзя иметь доступ к странице.

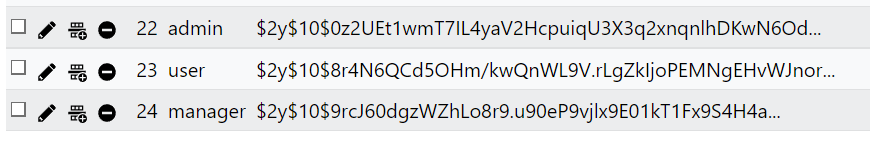
## **База данных и пароли**

Когда встал вопрос хранения паролей, конечно, первой идеей было просто записывать их в открытом виде в соответствующей табличке в базе данных. И все бы ничего, если бы доступ к ней действительно напрямую клиенты получить не могли. Но, к сожалению, в различных веб-приложениях по-прежнему иногда работает такая известная всем SQL-инъекция, не говоря уже о других потенциальных уязвимостях.

В вопросах безопасности вообще принято предполагать худшее и готовить план действий и защиту даже на такой случай. Будем считать, что злоумышленник нашел в веб-приложении лазейку, тем или иным способом радостно выгружает себе таблицу с именами и паролями пользователей и дальше уже распоряжается ими, как ему вздумается.

## **Правильное хранение паролей**

1. Использование надежных хэш функций, такие как: SHA256, SHA512, KECCAK256, RipeMD, WHIRLPOOL, ну или как в нашем случае – BCRYPT.
2. Отказ от использования ненадежных хэш функций, таких как SHA1.



## **Пользовательские роли**

Пользовательские роли предоставляют пользователям ряд разрешений на уровне клиента за рамками общих разрешений, полученных при распределении пользователей. С помощью пользовательских ролей можно предоставить пользователю определенную роль или функцию в клиенте. Назначить роли пользователям может только администратор клиента, и это может быть сделано только из Консоль управления. Пользовательские роли устанавливаются по желанию, и назначать их пользователям необязательно. У пользователей без пользовательской роли разрешения основаны на их распределении.

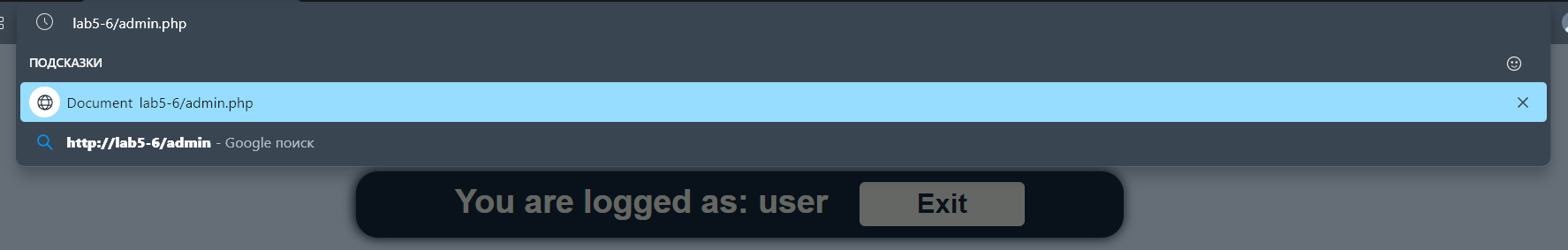
## **Правильное хранение паролей**

1. Использование ролей пользователей при помощи базы данных.
2. Управления доступом при помощи ролей.

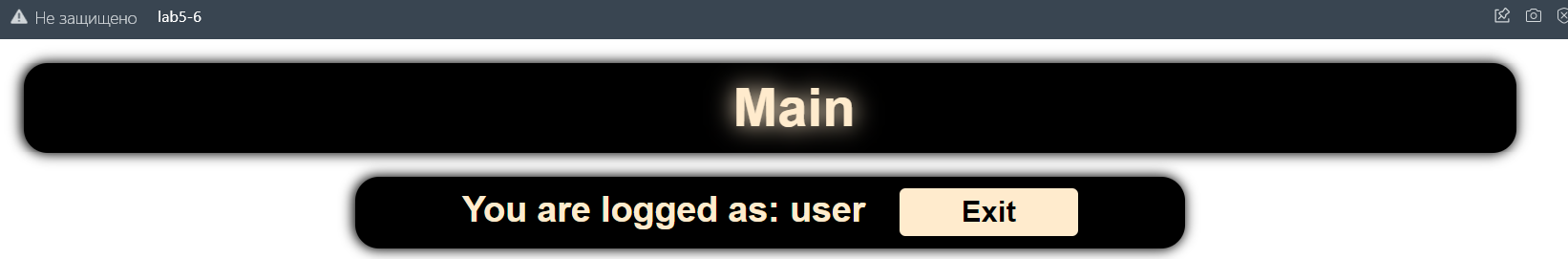




Попытка взлома user’ом



Отказ на переход в admin panel, /admin.php

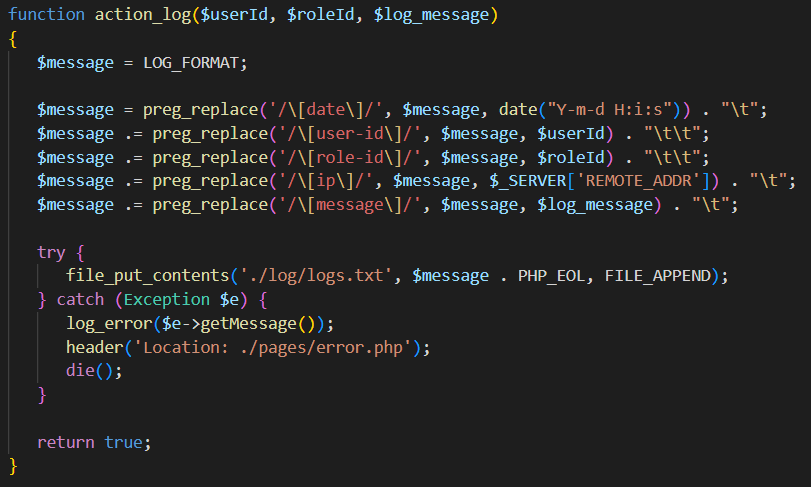


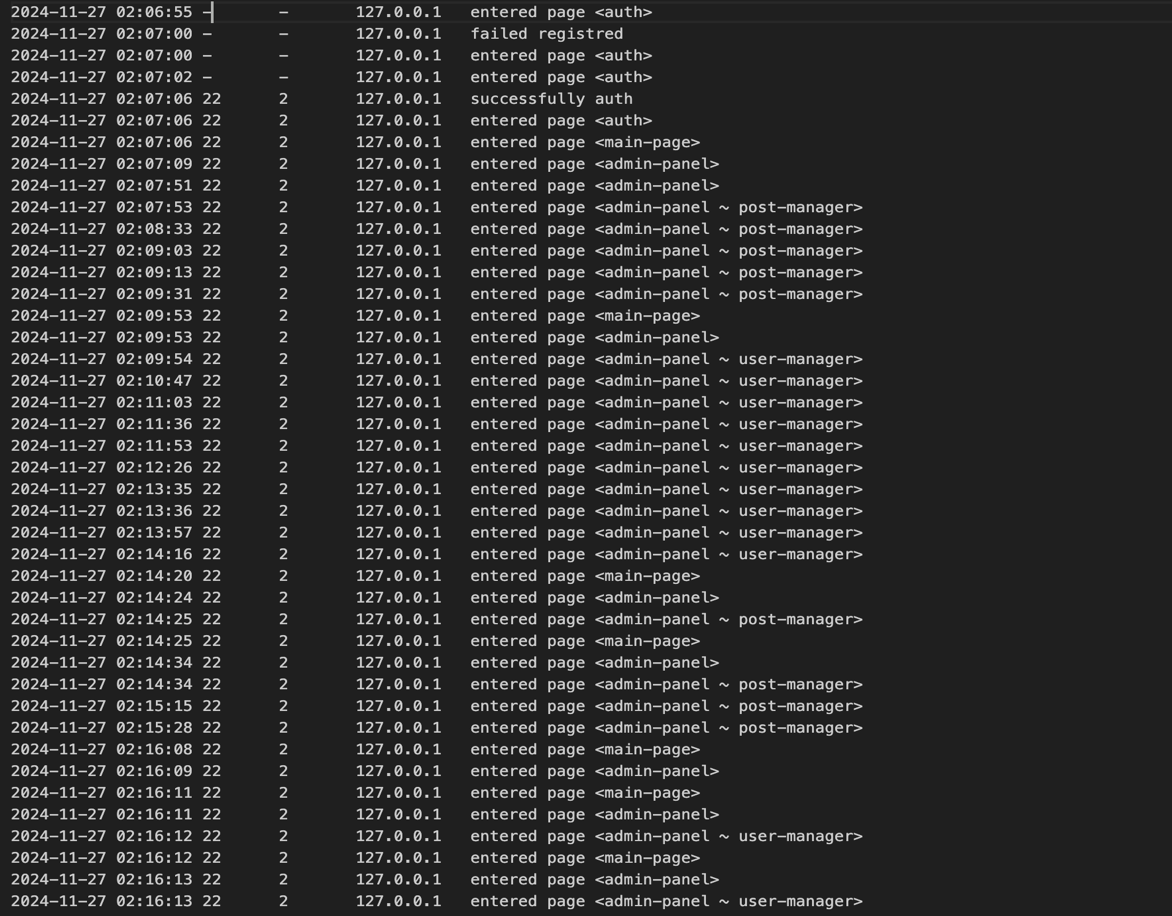
## **Логирование действий**

**Логирование действий** необходимо в ходе работы пользователей с автоматизированной системой, в которой непрерывно происходит изменение существующей информации, добавление новых данных, совершение различных действий, способных повлиять на ход дальнейших событий в соответствии с заложенными в программу моделями бизнес-процессов.

## **Использование логирования**

1. Используя функцию action\_log, записывающую информацию о действиях пользователя, мы можем отслеживать движения в нашем приложении.
2. Формат информации: LOG\_FORMAT = '[[date]] [user-id] [ip] [role-id] [message]'.





## **Обработка ошибок в приложении**

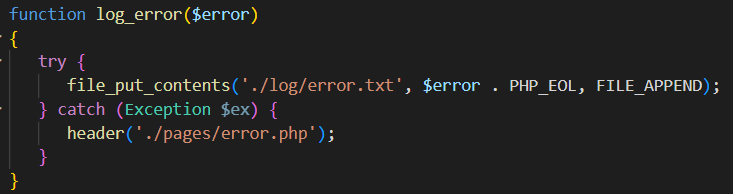
**Неважно, насколько мы хороши в программировании, иногда наши скрипты содержат ошибки. Они могут возникать из-за наших промахов, неожиданного ввода пользователя, неправильного ответа сервера и по тысяче других причин.**

**Обычно скрипт в случае ошибки «падает» (сразу же останавливается), с выводом ошибки в консоль.**

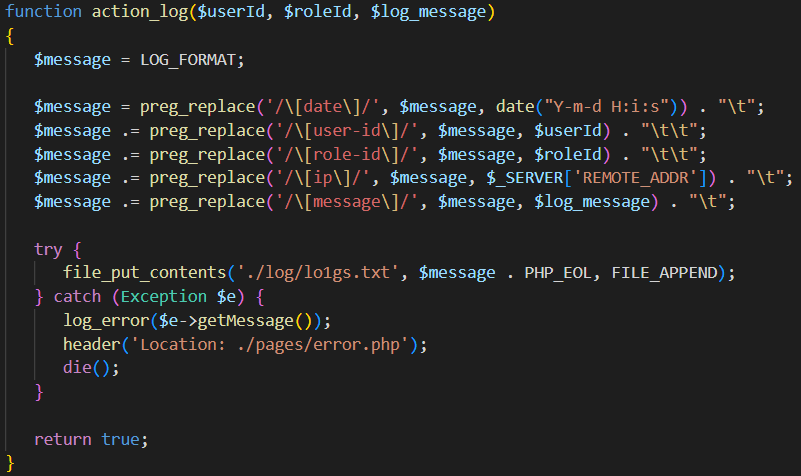
**Но есть синтаксическая конструкция try..catch, которая позволяет «ловить» ошибки и вместо падения делать что-то более осмысленное.**.

## **Использование логирования**

1. Используя функцию log\_error, мы можем отслеживать ошибки в нашем приложении.
2. При помощи try..catch мы можем отлавливать их.



Использование в логировании действий



Использование в связи с базой данных



