Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего образования  
УРАЛЬСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ ИМЕНИ ПЕРВОГО ПРЕЗИДЕНТА РОССИИ Б. Н. ЕЛЬЦИНА  
(УрФУ)

ОЦЕНКА РАБОТЫ

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4  
по дисциплине «Разработка безопасных веб-приложений»

Дата Подпись Ф.И.О.

Преподаватель Мирвода С.Г.

Студент Синадский М.Н.

Студент Дьячков М.Д.

Студент Киселев Д.А.

Группа РИ-491223

Екатеринбург 2023

## Создание БД

Устанавливаем PostgreSQL добавляем все таблицы из файла lab4 -> db:

Изображение выглядит как текст

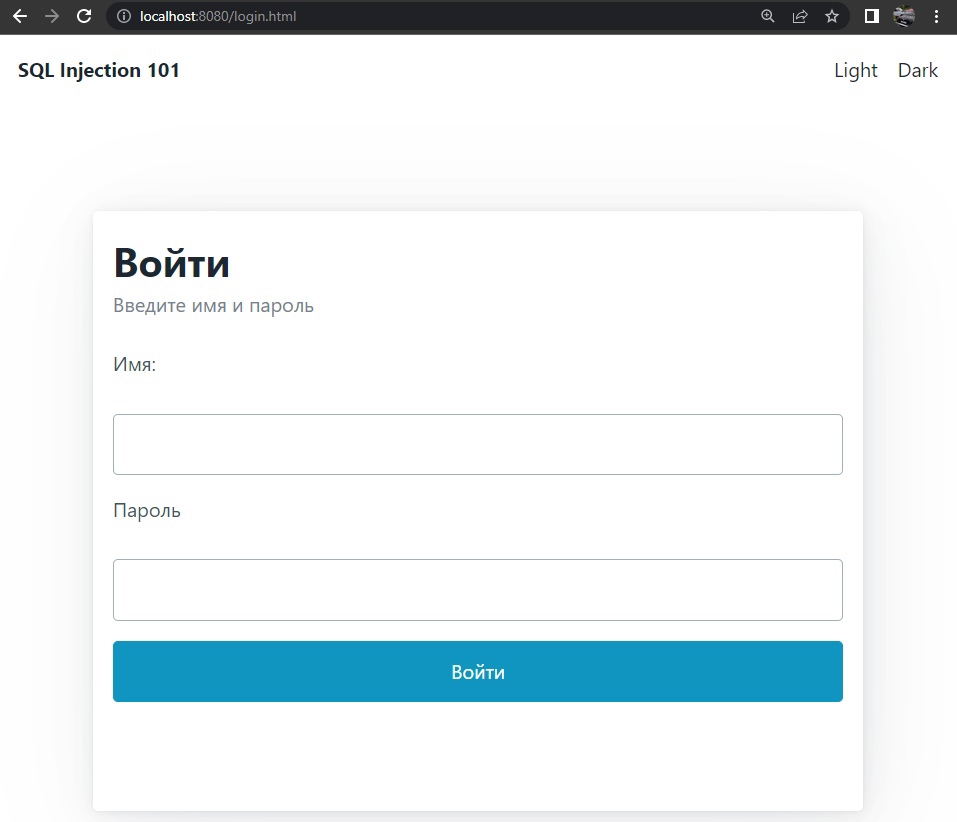
Автоматически созданное описание

Устанавливаем nodejs. Переходим в папку lab4и выполняем в ней команду npm install.

Изображение выглядит как текст

Автоматически созданное описание

Запускаем сервер через команду npm start, теперь нам доступен сайт.



Входим на сайт и видим список книг и авторов.

Изображение выглядит как стол

Автоматически созданное описание

## Reflected XSS

Получили список книг по фильтру.

Изображение выглядит как текст

Автоматически созданное описание

Используя строку поиска или напрямую через ссылку, можно сформировать следующий запрос:

<img src=1 onerror='javascript:alert(document.cookie)'/>

Это позволит украсть cookie жертвы, перешедшей по ссылке.

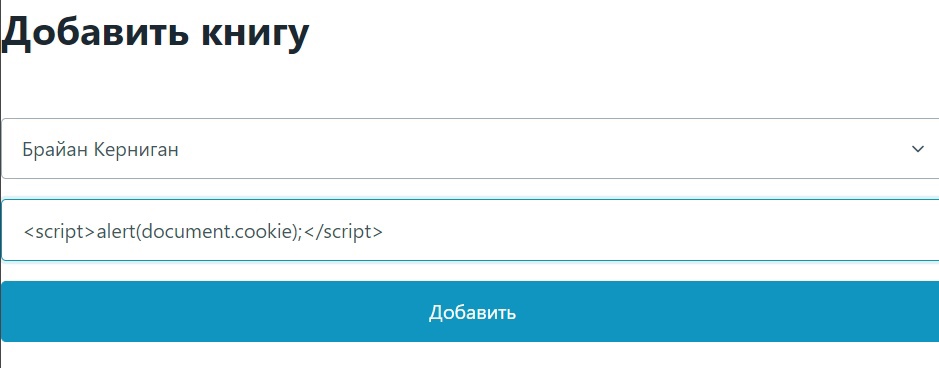
Если вместо alert написать POST-запрос, то, получив такую вредную ссылку, пользователь не только покажет cookies в alert'e, но и отправит их на сервер злоумышленника.

Изображение выглядит как текст

Автоматически созданное описание

## Persisted XSS

Добавляем в раздел "название" исполняемый JS-код, который позже будет выполняться, когда другие пользователи будут его смотреть (он хранится в базе данных как название книги, поэтому persisted (хранимый) XSS).



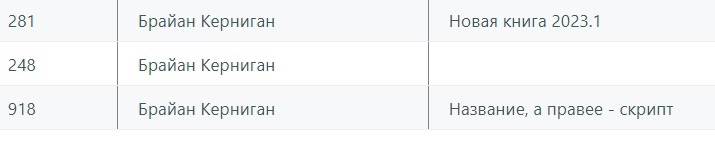
Теперь при любом открытии страницы с книгами /books нам выводится сообщение с кукой. Тут тоже можно встроить отправку данных на сторонний сервис, как это описано в 8.1.

Изображение выглядит как текст

Автоматически созданное описание

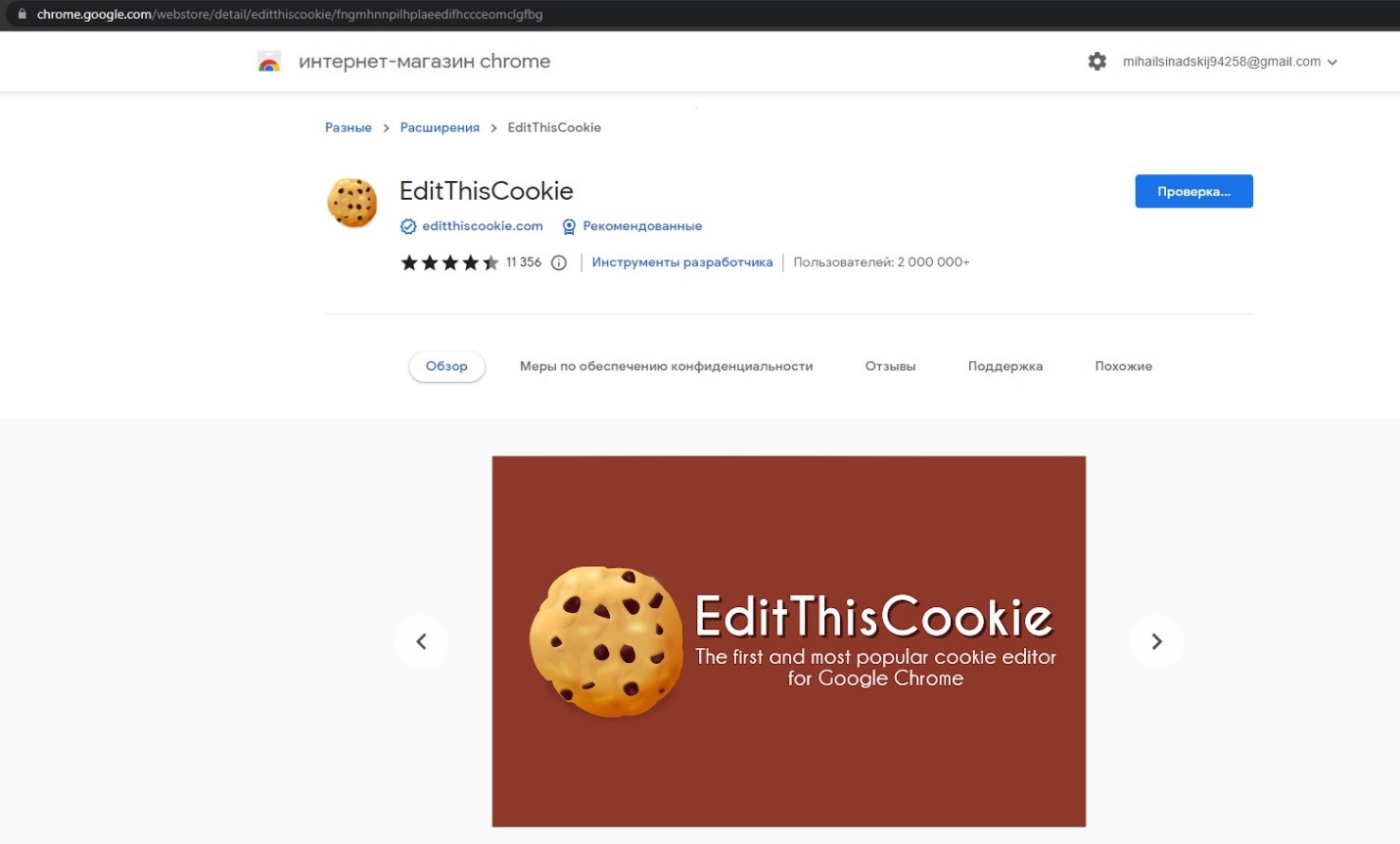
Изображение выглядит как текст

Автоматически созданное описание



## Cookie Injection

Для просмотра содержимого и редактирования cookies установили расширение Chrome "EditThisCookie".

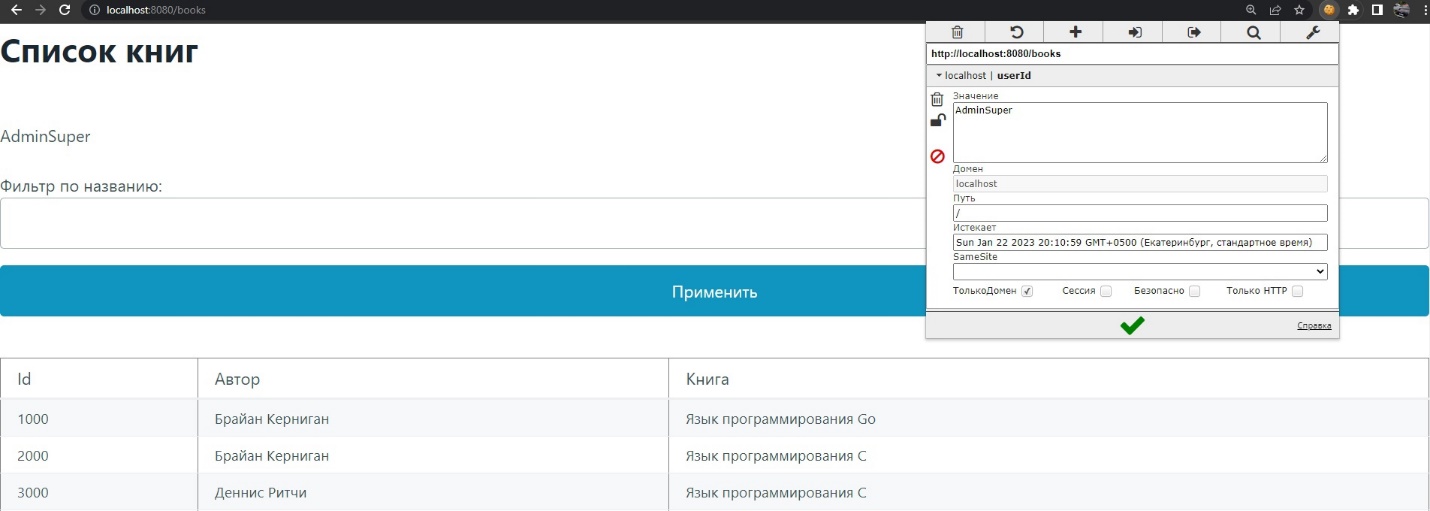


Открыли для редактирования cookie.

Изображение выглядит как стол

Автоматически созданное описание

Передаем сайту информацию, что теперь пользователь Super-Admin.  
Отправляем ему эту куку при обновлении страницы.



## Session hijacking

Данные передаются по незащищённому протоколу (HTTP), поэтому можно подсмотреть содержимое передаваемых пакетов.  
Ищем Cookies, находим userId одного из легитимных пользователей. Теперь, если злоумышленник подставит в EditCookie себе этот userId, сайт будет думать, что он и есть тот легитимный пользователь (хотя на самом деле это злоумышленник).

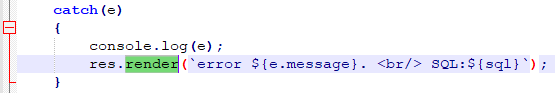
Изображение выглядит как текст

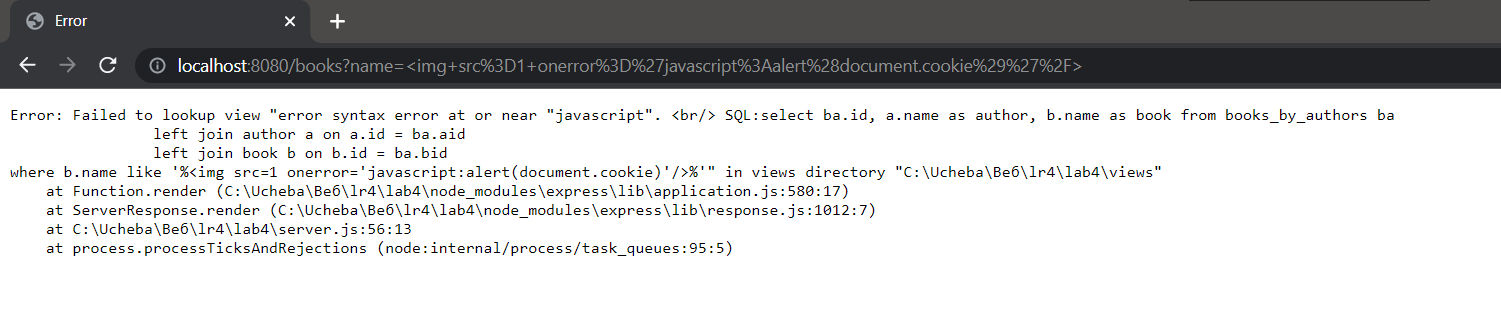
Автоматически созданное описание

## Защита

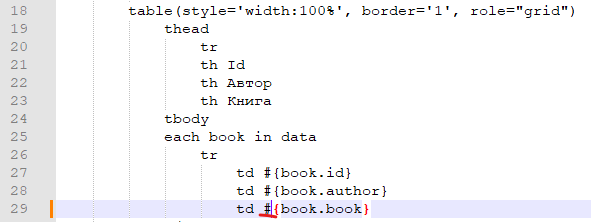
**Защита от Reflected XSS:**

Доработаем функцию обработки ошибок и заменить функцию res.send() на res.render(). Разница заключается в параметре ContentType: в первом случае это application/json, во втором – text/html.

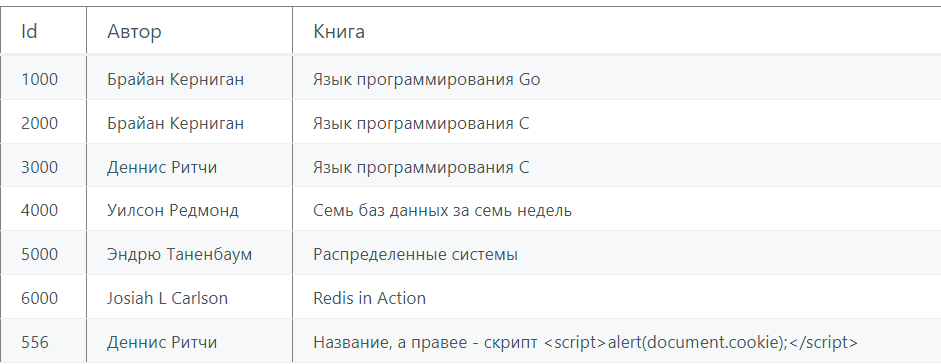




**Защита от Persisted XSS:**

Заменим знак «!» на «#» в файле шаблона Pug. 

Теперь скрипт записывается в строку названия книги и не выполняется.



**Защита от Session hijacking (захват сессионной cookie):**

Изменив на httpOnly, мы получим доступ к файлам cookie только со стороны одного клиента, который высылает куку, и сервера, который эту куку обрабатывает. То есть содержимое cookie не может быть просмотрено (изменено) с помощью JavaScript.

