

Технологии разработки Интернет приложений: Lab4

Адилов Марсель

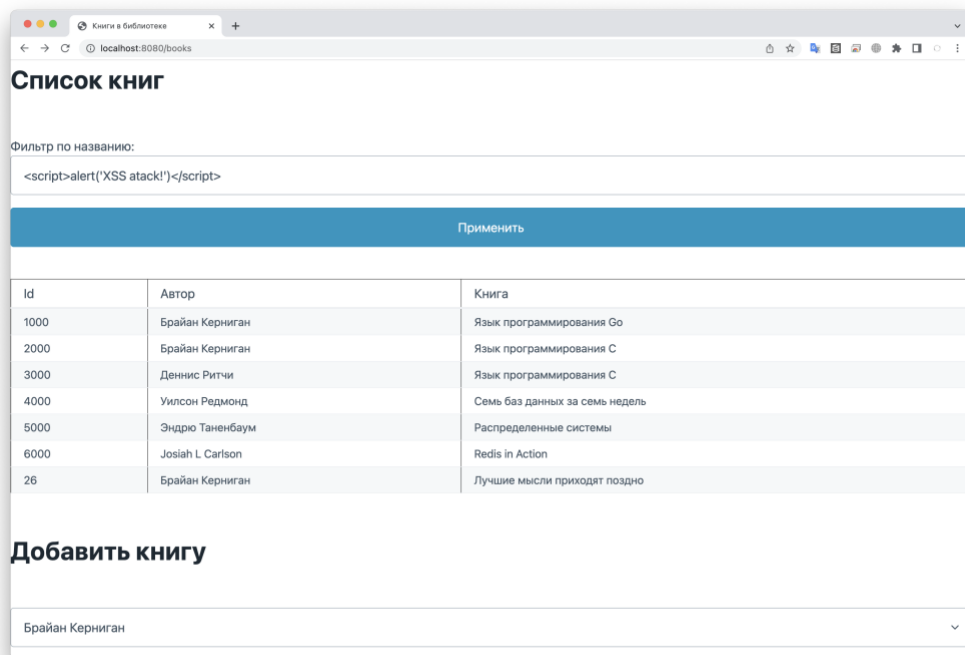
РИ-491223

- ТРИП: Lab4
 - Уязвимости
 - Reflected XSS в поиске книг
 - Persisted (Stored) XSS
 - Cookie injection / Session hijacking
 - Defacing
 - Исправляем уязвимости
 - Reflected XSS в поиске книг
 - Persisted (Stored) XSS / Defacing
 - Cookie injection / Session hijacking

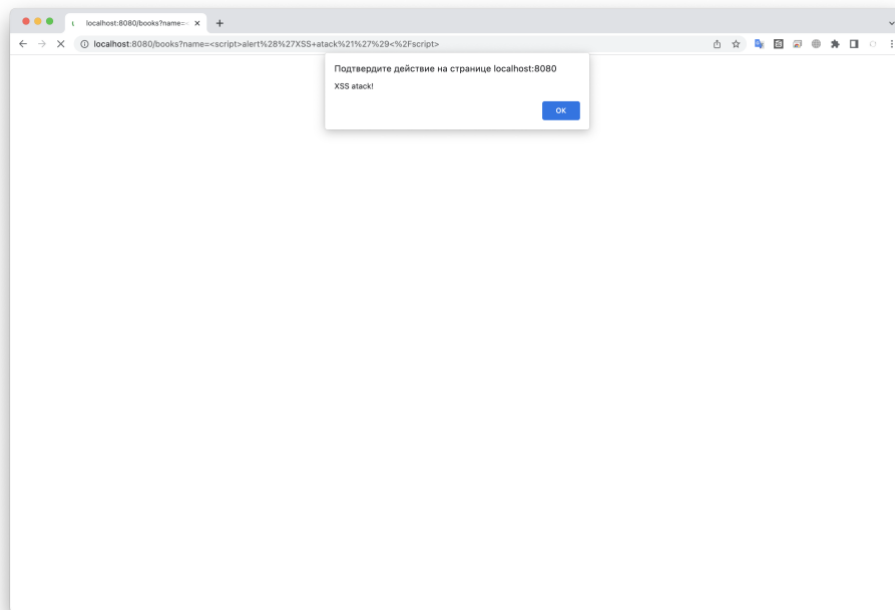
Уязвимости

Reflected XSS в поиске книг

После входа под логином Jhon проверяем поиск книг. Вводим в строку поиска скрипт по вызову баннера `<script>alert('XSS attack!')</script>`.



После подтверждения получаем ответ от сервера и видим баннер XSS attack!, значит скрипт выполнен успешно.

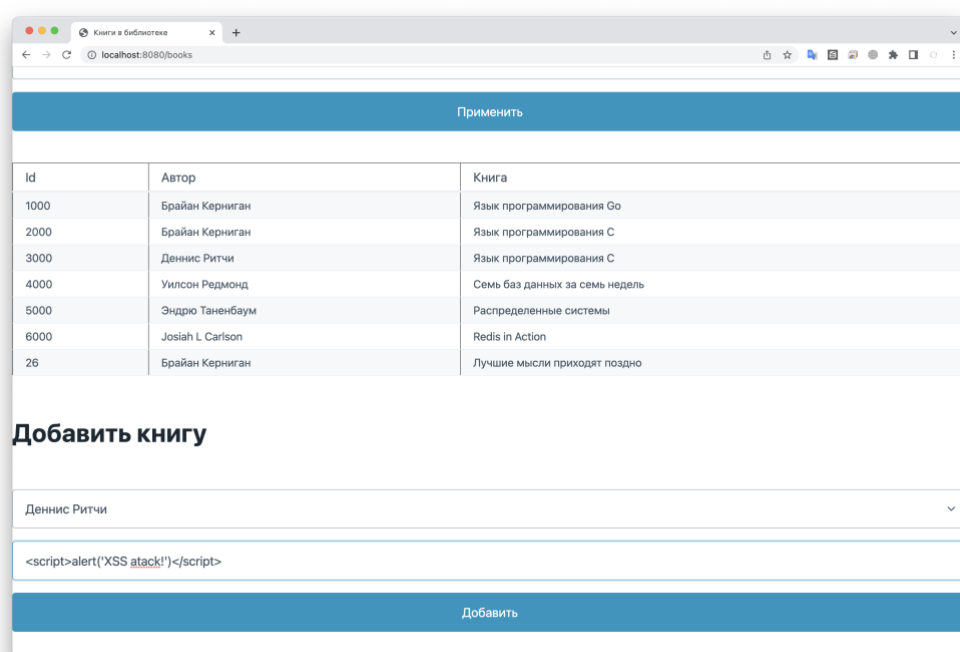


Теперь можно скопировать ссылку

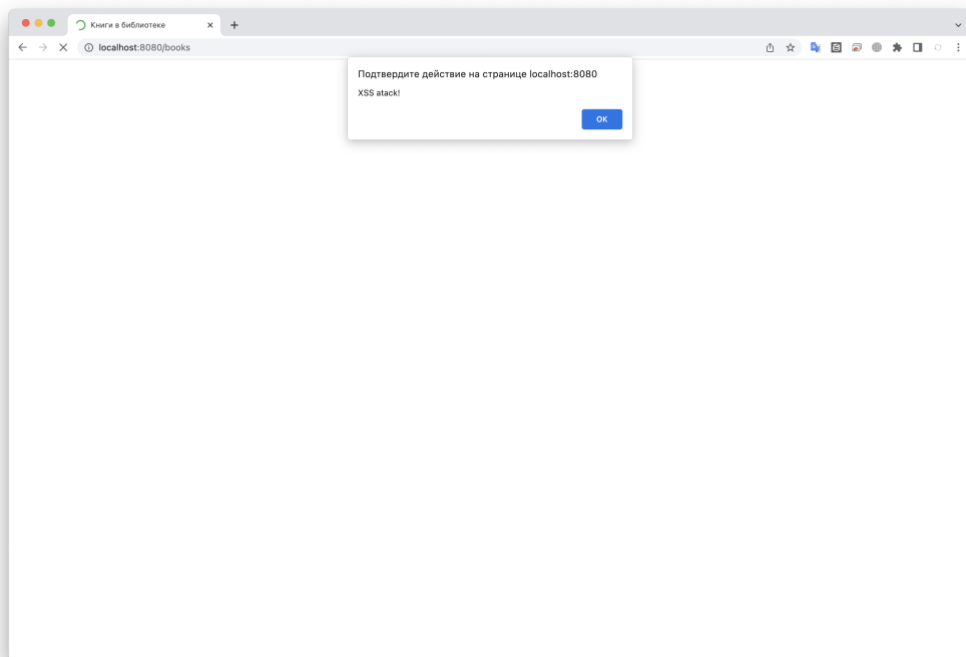
`http://localhost:8080/books?name=%3Cscript%3Ealert%28%27XSS+atack%21%27%29%3C%2Fscript%3E`, укоротить её для скрытия видимого слова `script` и отправить жертве.

Persisted (Stored) XSS

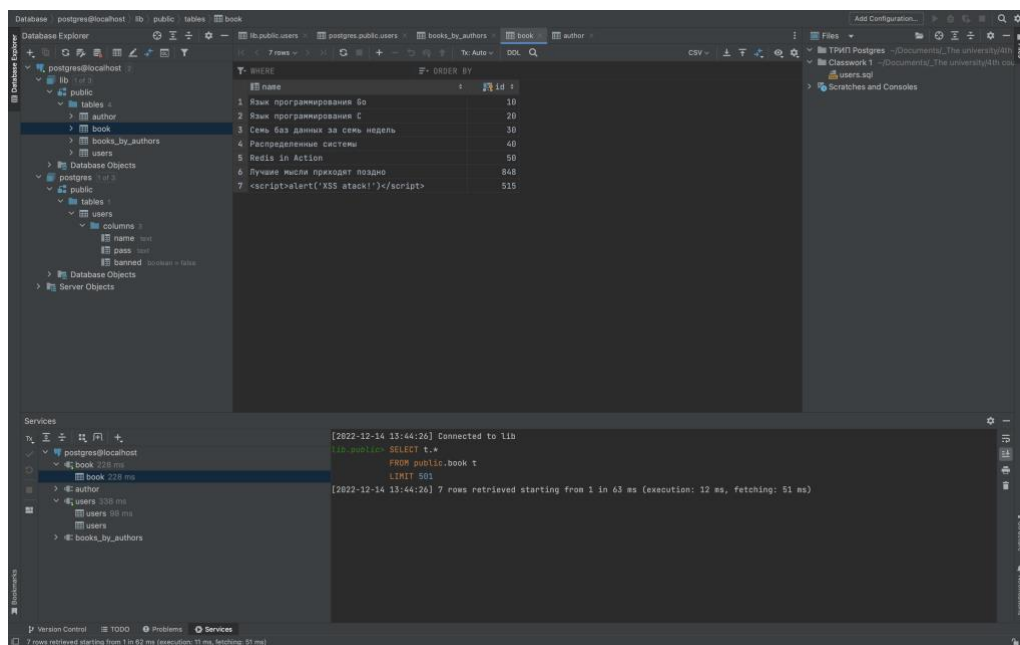
При создании книги вводим в поле скрипт `<script>alert('XSS attack!')</script>`.



Теперь при каждой загрузке страницы со списком книг у пользователей выполняется скрипт и появляется соответствующий баннер.



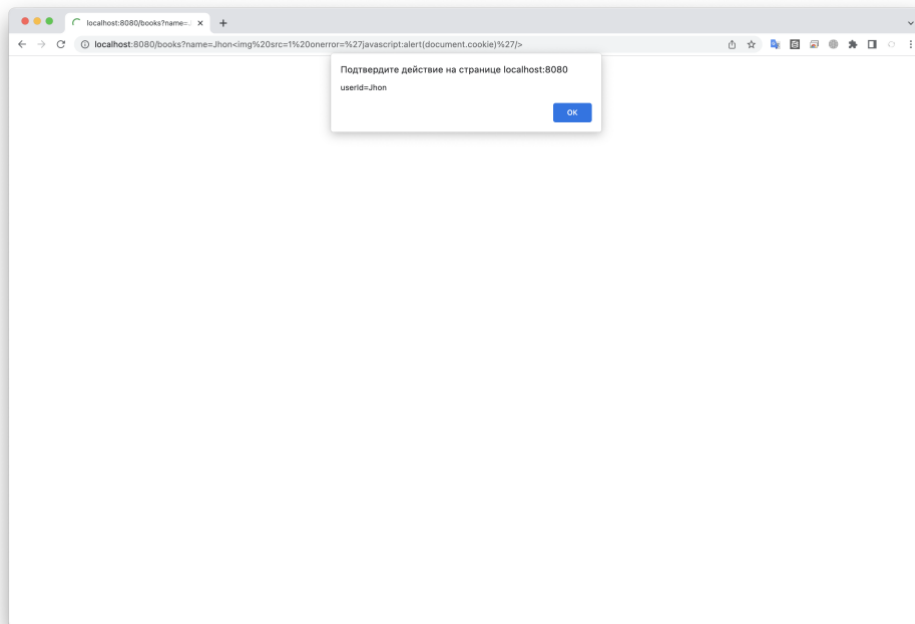
Проверяем в базе данных и видим, что скрипт действительно хранится в таблице книг.



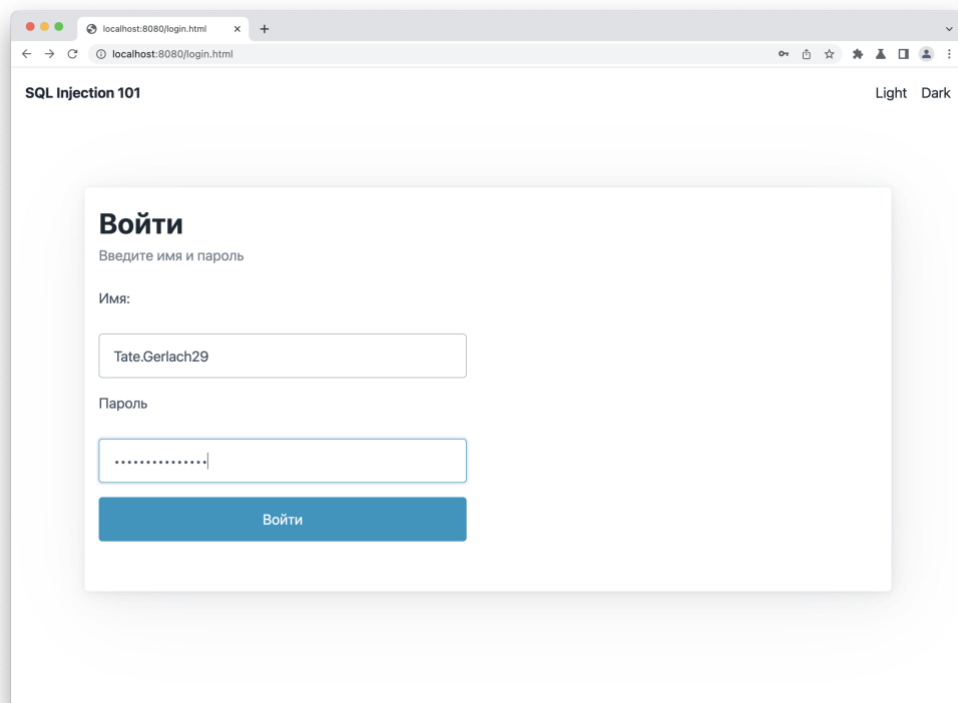
Cookie injection / Session hijacking

После входа на сайт под именем пользователя Jhon вводим в адресную строку
`http://localhost:8080/books?name=Jhon<img src=1`

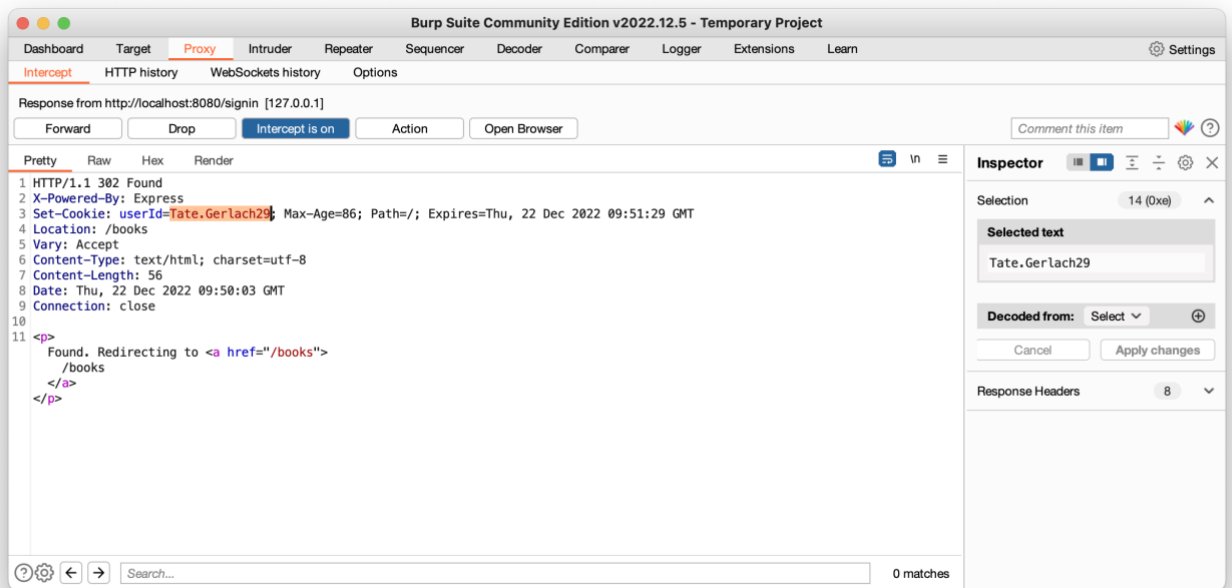
onerror='javascript:alert(document.cookie)'/> и получаем на выходе идентификатор Cookie `userID = Jhon`. Используем его для Cookie injection.



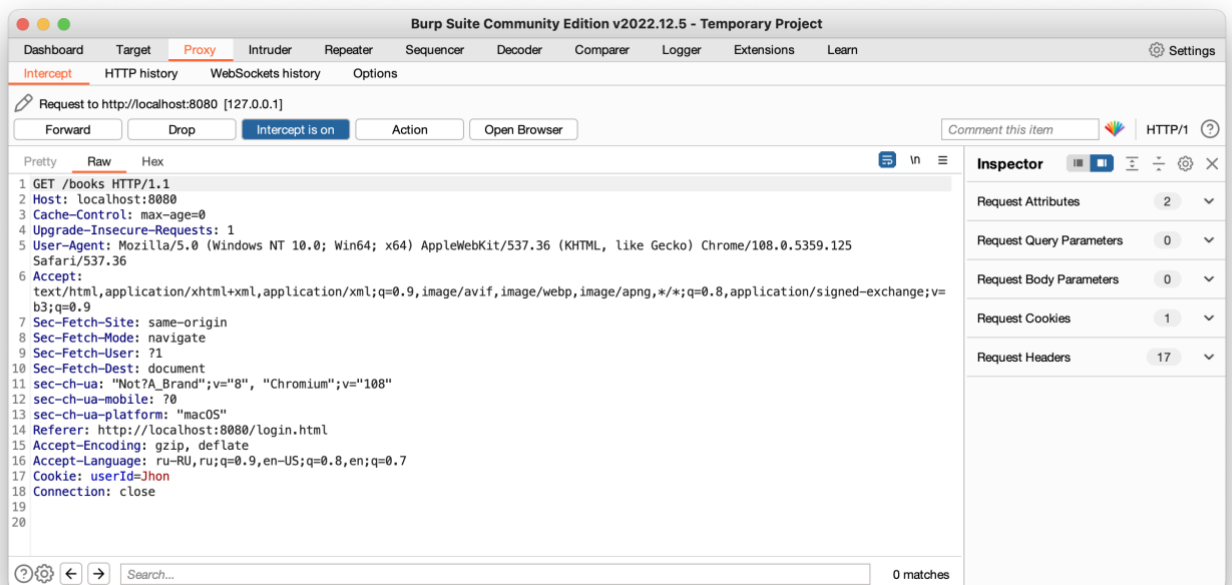
Воспользуемся программой Burp Suite и включим в нем Proxy Intercept. После заходим на страницу входа с другого браузера и вводим данные другого пользователя для успешного входа.



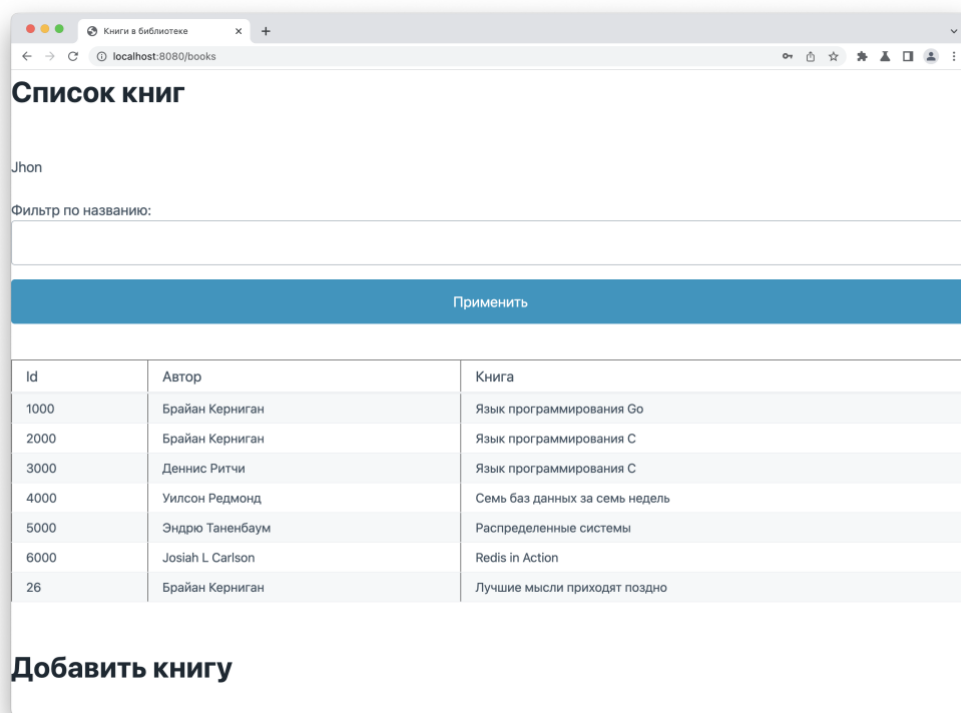
После нажатия кнопки "Войти" в программе Burp Suite мы меняем пойманный ответ в строке Set-Cookie с `userID=Tate.Gerlach29` на `userID=Jhon`.



Далее мы посылаем запрос на сервер, и уже видно, что cookie usedID=Jhon.

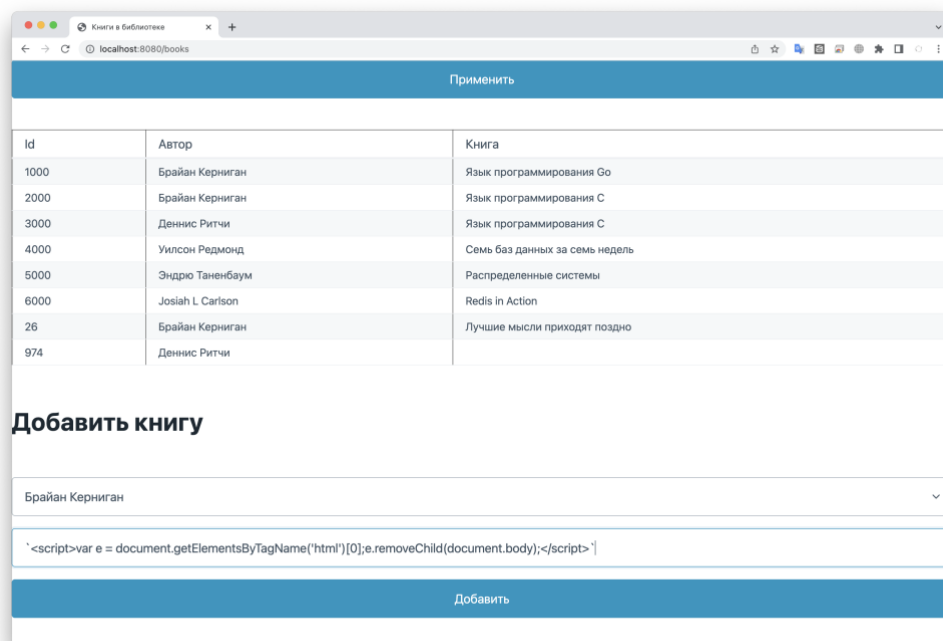


По итогу мы входим на сайт под акаунтом Jhon.

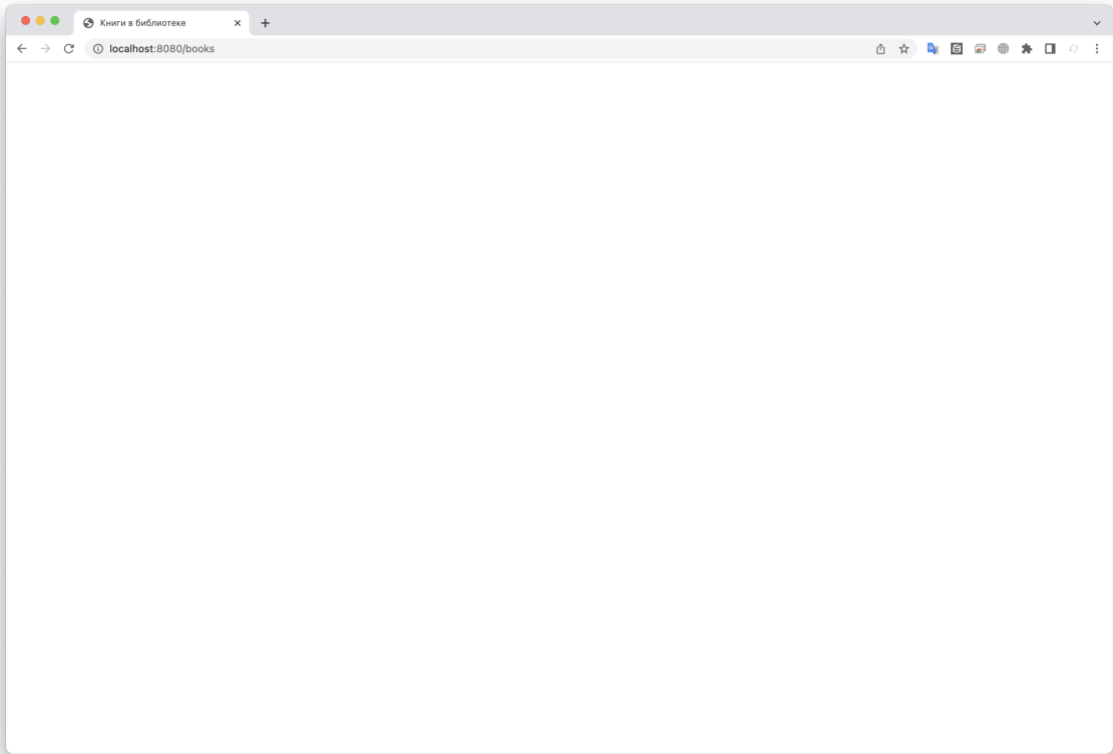


Defacing

В поле добавления книги вводим скрипт по удалению содержимого сайта `<script>var e = document.getElementsByTagName('html')[0];e.removeChild(document.body);</script>`.



После нажатия кнопки "Добавить" сайт перезагружается и ничего не показывает, так как при каждой загрузке страницы выполняется введенный нами скрипт, который попал в таблицу книг.

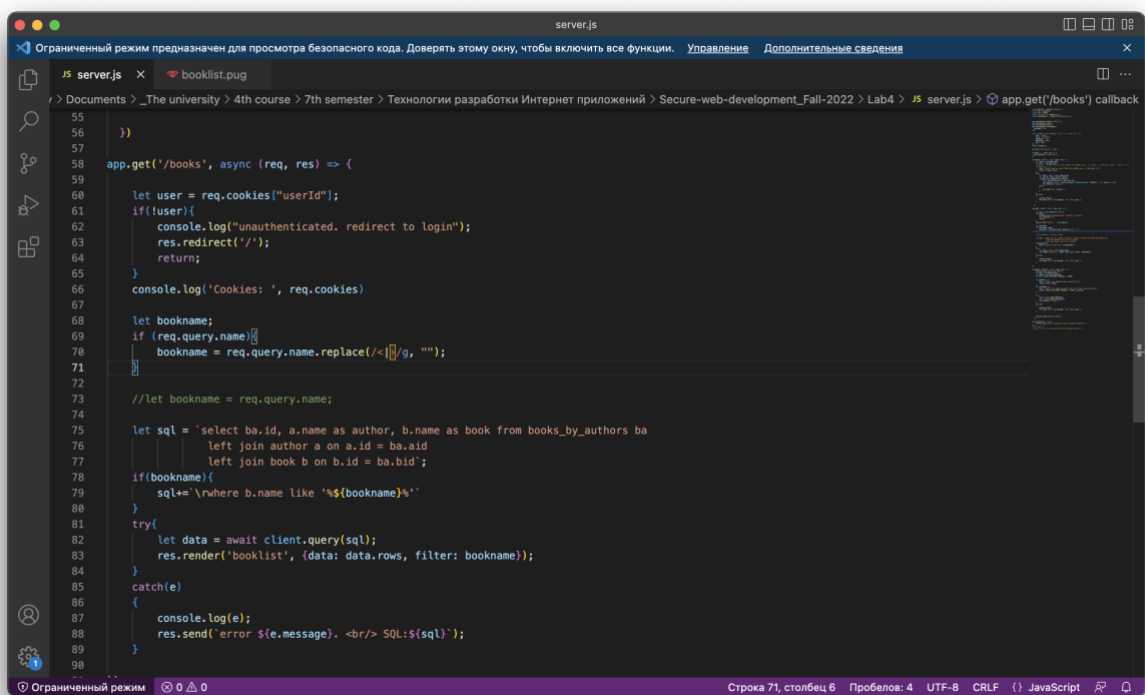


Исправляем уязвимости

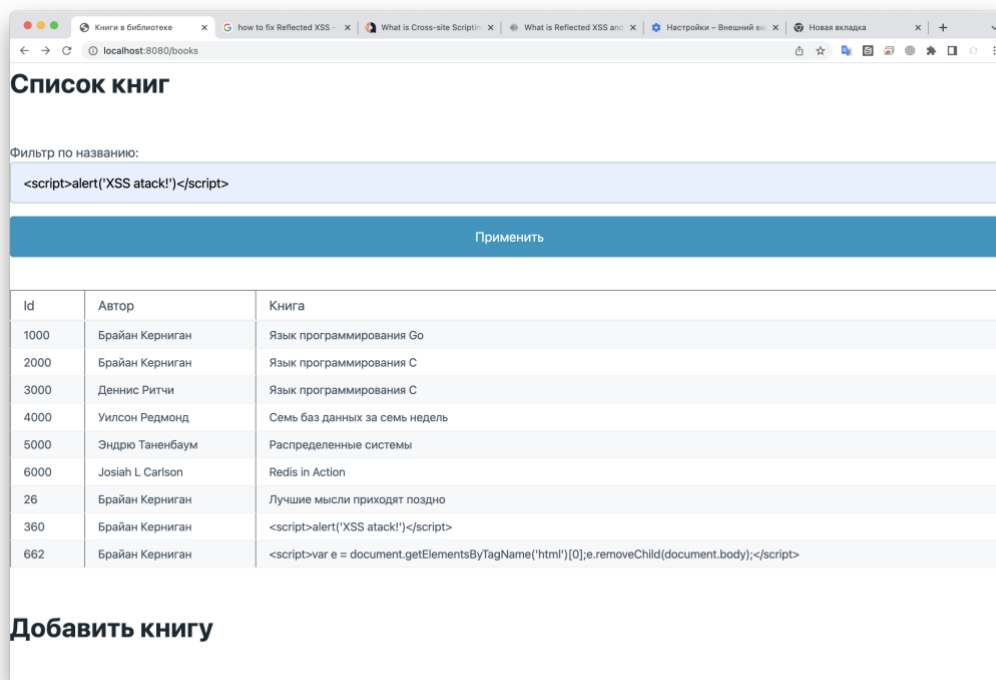
Reflected XSS в поиске книг

В server.js будем делать замену символов:

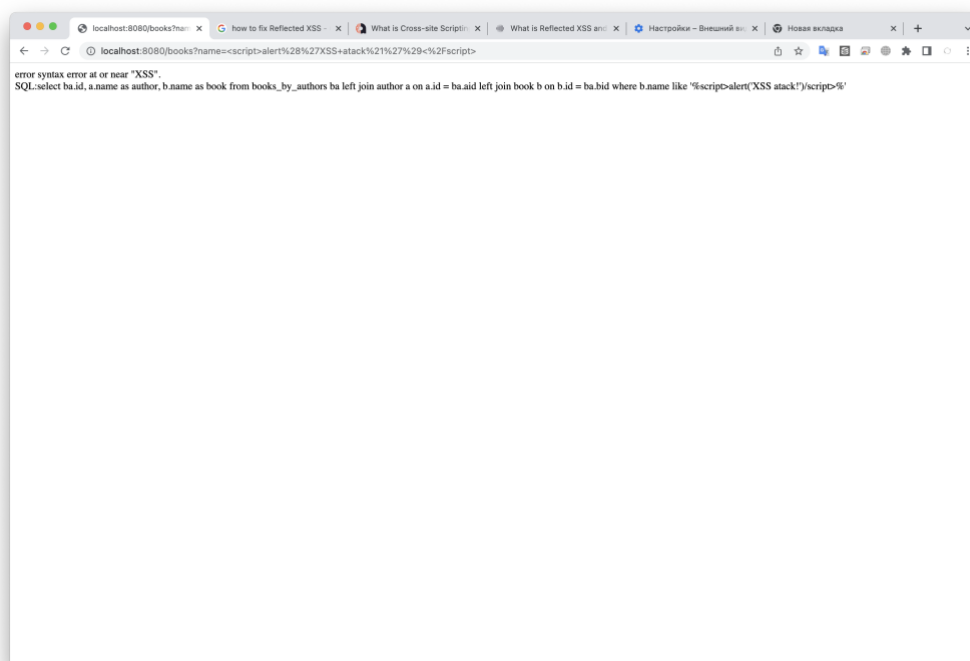
```
let bookname;
if (req.query.name){
  bookname = req.query.name.replace(/<|>/g, "");
}
```



Теперь проверим. В поле поиска книги введем скрипт по вызову баннера `<script>alert('XSS atack!')</script>`.



После выполнения никакого баннера не было, следовательно уязвимость закрыта.

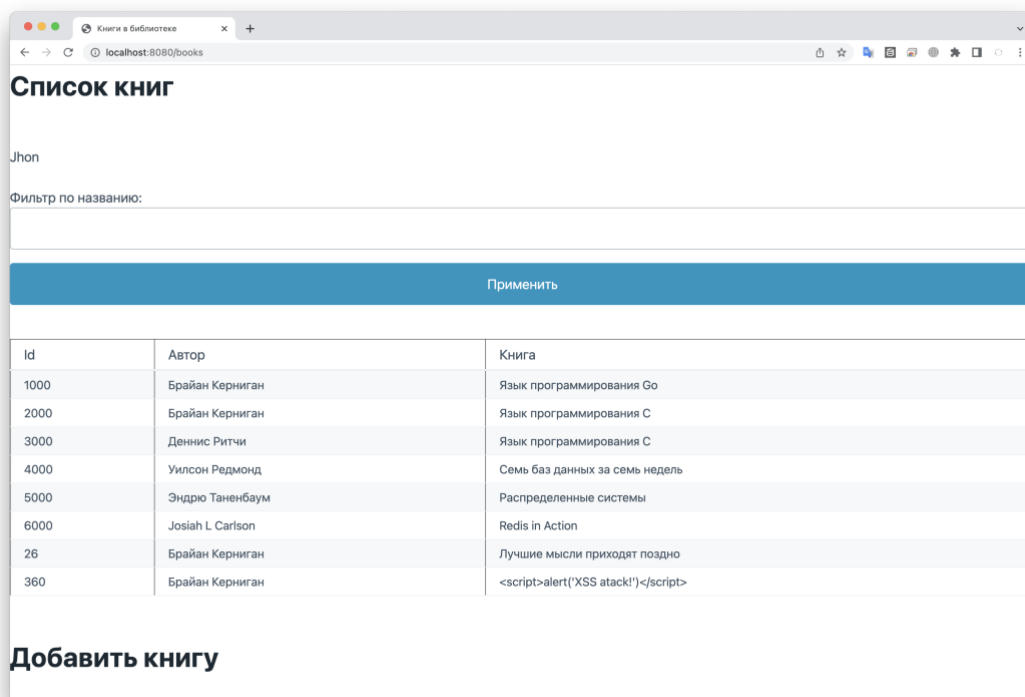


Persisted (Stored) XSS / Defacing

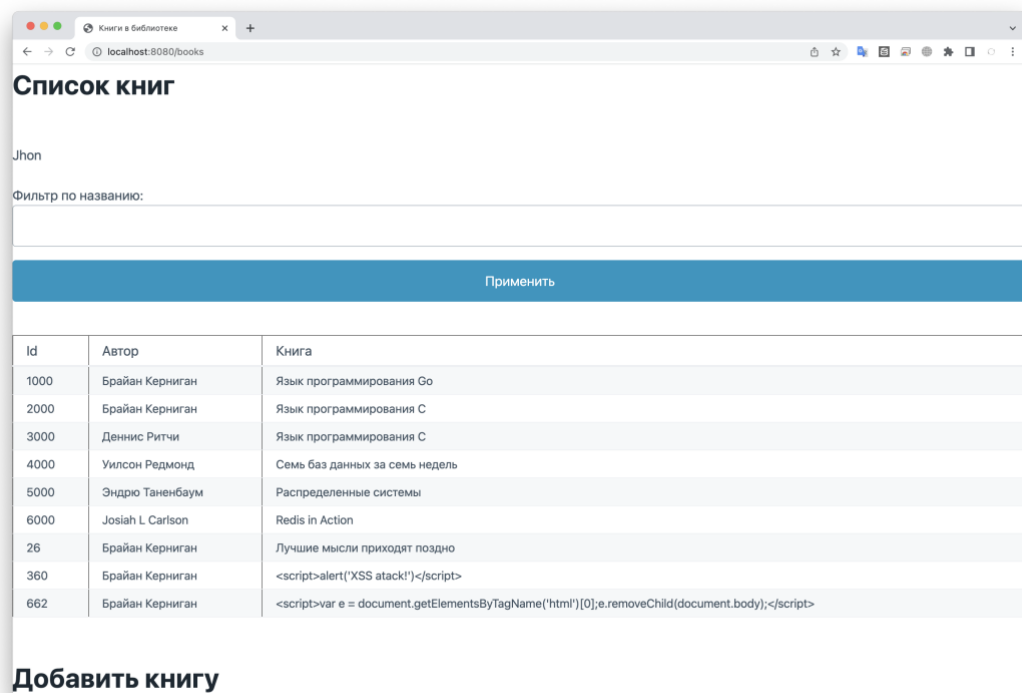
В секции файла booklist.pug, отвечающей за отображение списка книг, для отображения названия книг `!{book.book}` стоит восклицательный знак. Поэтому скрипт срабатывает, ведь код обрабатывается. Для предотвращения поставим решетку `#{book.book}`.

```
booklist.pug
10 form(action='/books', method='GET')
11   span(id='lblFilter') #filter
12   span(id='filter')
13   p
14     | Фильтр по названию:
15     input(type='text', name='name', value='')
16     input(type='submit', value='Применить')
17
18   table(style='width:100%', border='1', role='grid')
19     thead
20       tr
21         th Id
22         th Автор
23         th Книга
24     tbody
25       each book in data
26         tr
27           td #{book.id}
28           td #{book.author}
29           td #{book.book}
30
31   h1 Добавить книгу
32   form(action='/addbook', method='POST')
33     <select id='author' name='author'>
34       <option value='1'>Брайан Керниган</option>
35       <option value='2'>Деннис Ритчи</option>
36     </select>
37     input(type='text', name='bookname', value='')
38     input(type='submit', value='Добавить')
39
40   script.
41     var f = document.getElementById('lblFilter');
42     f.innerHTML = '<b>#filter</b>';
43
44     var f = document.getElementById('lblUser');
45     const userId = (';'+document.cookie.split(';').pop().split(':')[0]);
46     f.innerHTML = userId;
```

Проверим, впишем в поле добавления книги скрипт по вызову баннера `<script>alert('XSS attack!')</script>`. После добавления страница перезагружается, а баннер не высвечивается. Уязвимость закрыта.



Это также решает уязвимость для Defacing. В поле добавления книги вводим скрипт по удалению содержимого сайта `<script>var e = document.getElementsByTagName('html')[0];e.removeChild(document.body);</script>`. После перезагрузки страницы видим, что все содержимое осталось на месте.



Cookie injection / Session hijacking

В файле `server.js` добавим атрибуты `HttpOnly` и `Secure` для Cookie. `res.cookie('userId', userId, {maxAge: oneDayToSeconds, httpOnly: true, secure: true});`.

```

25
26 app.get('/', (req, res) => {
27   res.redirect('/login.html');
28 })
29
30 app.post('/signin', async (req, res) => {
31   let login = req.body.name;
32   let pass = req.body.pass;
33   //let sql = "SELECT name as result FROM users WHERE name = '" + login + "' AND pass = md5('" + pass + "')";
34   let sql = {
35     text: "SELECT name as result FROM users WHERE name = $1 AND pass = $2",
36     values: [login, pass]
37   };
38   try {
39     let data = await client.query(sql);
40     let userId = data.rows[0].result;
41     if(data.rows.length > 0 && userId) {
42       const oneDayToSeconds = 24 * 60 * 60;
43       res.cookie('userId', userId, {maxAge: oneDayToSeconds, httpOnly: true, secure: true});
44       res.redirect('/books');
45     } else {
46       res.send('fail $(login)');
47     }
48   } catch (e) {
49     console.log(e);
50     res.send('error $(e.message). <br/> SQL:$(sql)');
51   }
52 })
53
54 app.get('/books', async (req, res) => {
55   let user = req.cookies['userId'];
56 })
57
58
59

```

Теперь проверим, сработало ли. Зайдем на сайт в качестве пользователя Jhon и вставим в адресную строку следующее: `http://localhost:8080/books?name=Jhon`. По выполнению нам не высвечивается

идентификатор userID из Cookie, поэтому нам нечего будет использовать для Cookie injection.
Уязвимость закрыта.

