Лабораторная работа №4

Разработка безопасных web приложений

Подготовили студенты РИ-581220

Дьячков Владислав Николаевич

Родионов Леонид Владиславович

Reflected XSS

В строке поиска книг напишем следующую конструкцию: <script>alert(‘<Произвольный текст>’)</script>. При нажатии на кнопку «Применить» увидим, что скрипт выполнится:

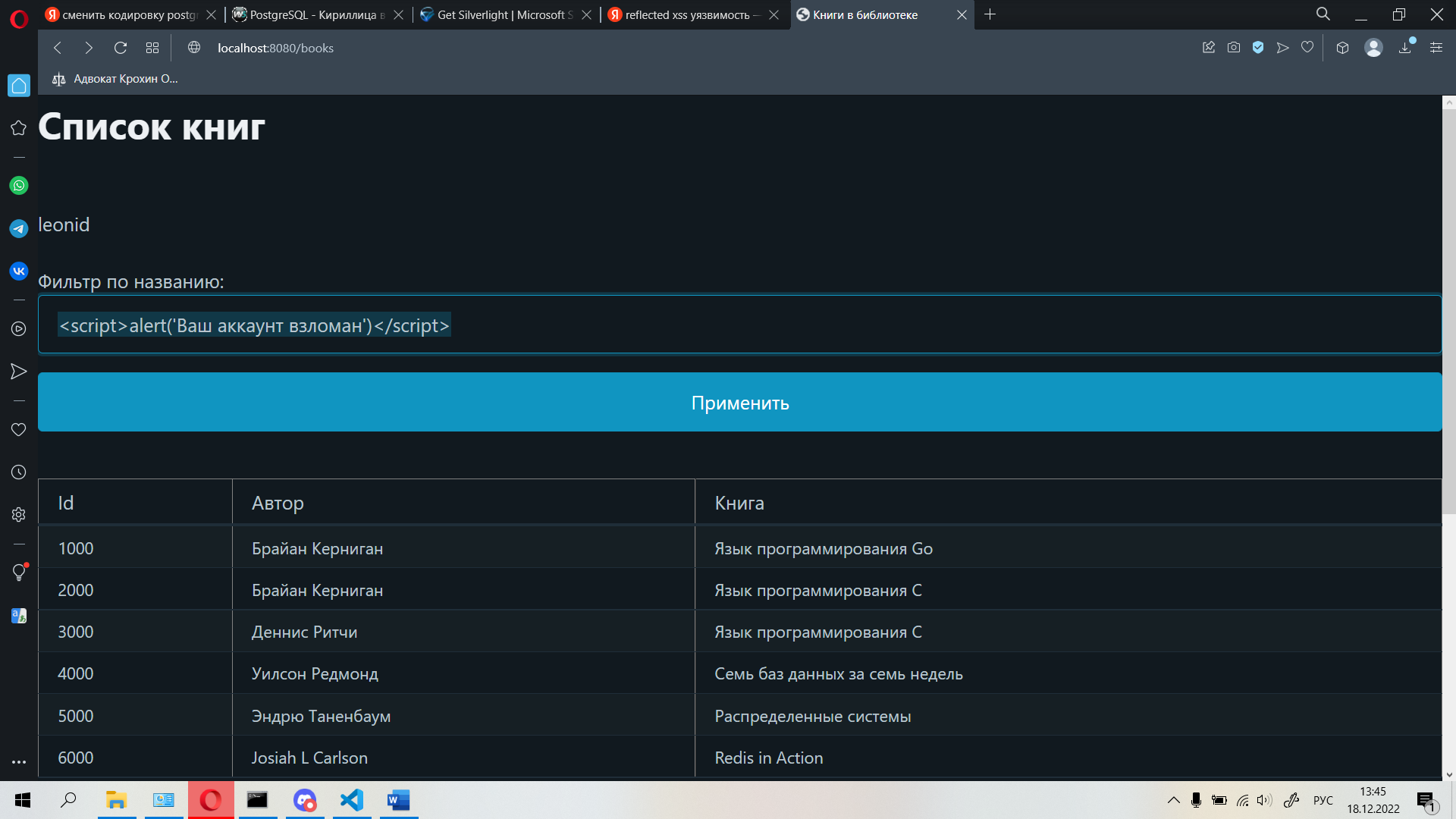


Рисунок 1 – эксплуатация Reflected XSS

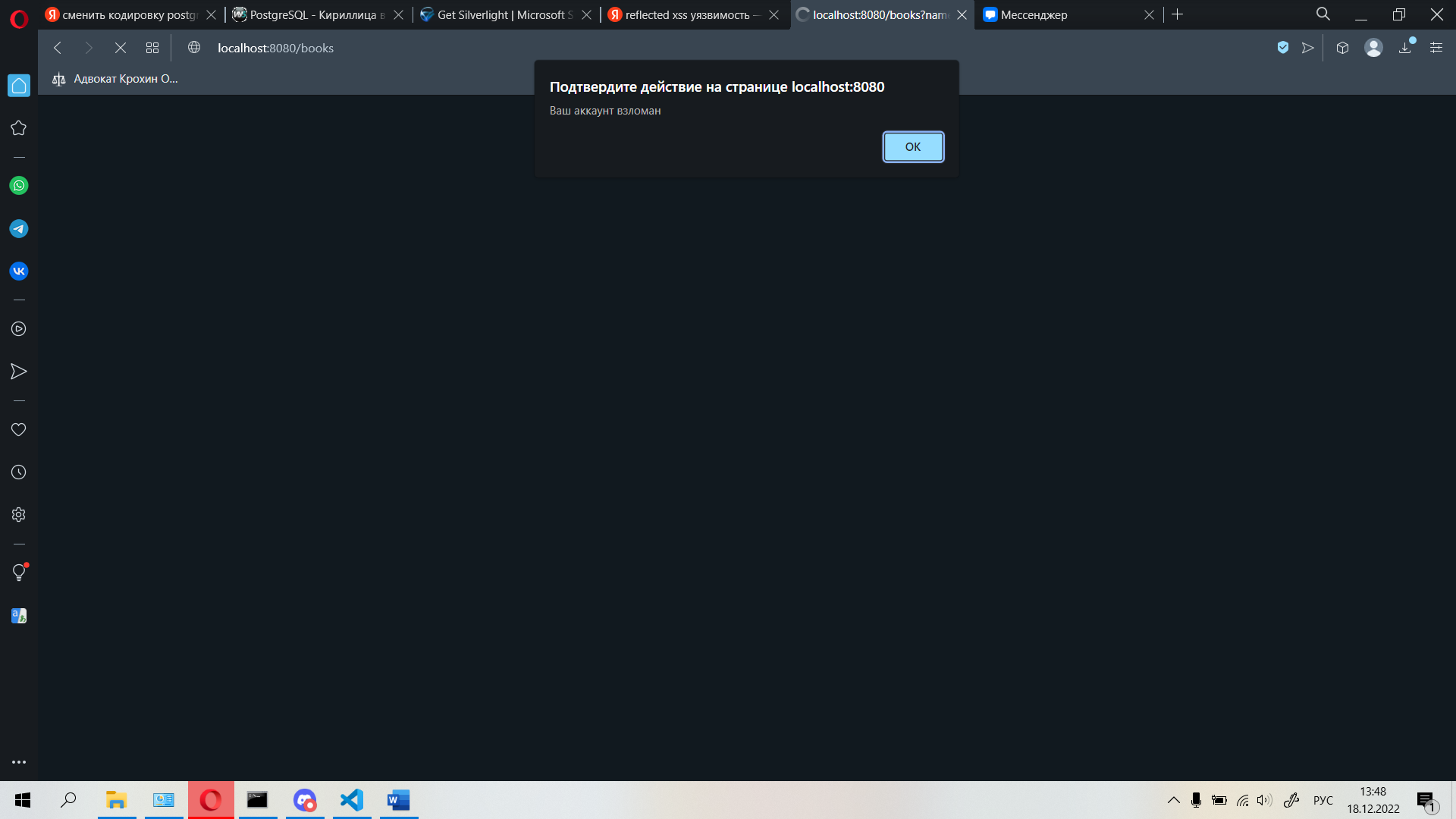
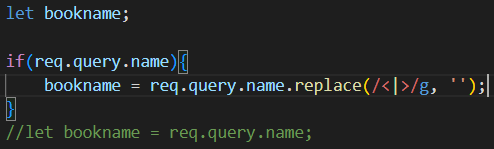


Рисунок 2 – Результат Reflected XSS

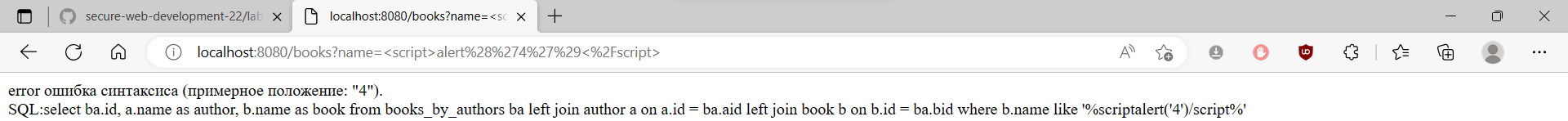
Reflected XSS может использоваться для одноразового получения информации из базы данных сайта и может быть применена для «разведки»

Исправить данную уязвимость можно следующим образом:



Если в поле req.query.name не пустое, из него убираются все символы < и >, позволяющие вставлять теги html

Теперь при попытке вставить скрипт в поле поиска, будет следующий результат:



Как можно заметить, из пользовательского ввода удалились треугольный скобки и сервер, не выполнив скрипт, просто выдал ошибку sql-запроса.

Stored XSS используется тогда, когда к сайту происходит новое подключение. Таким образом, например, можно красть cookie файлы посетителей.

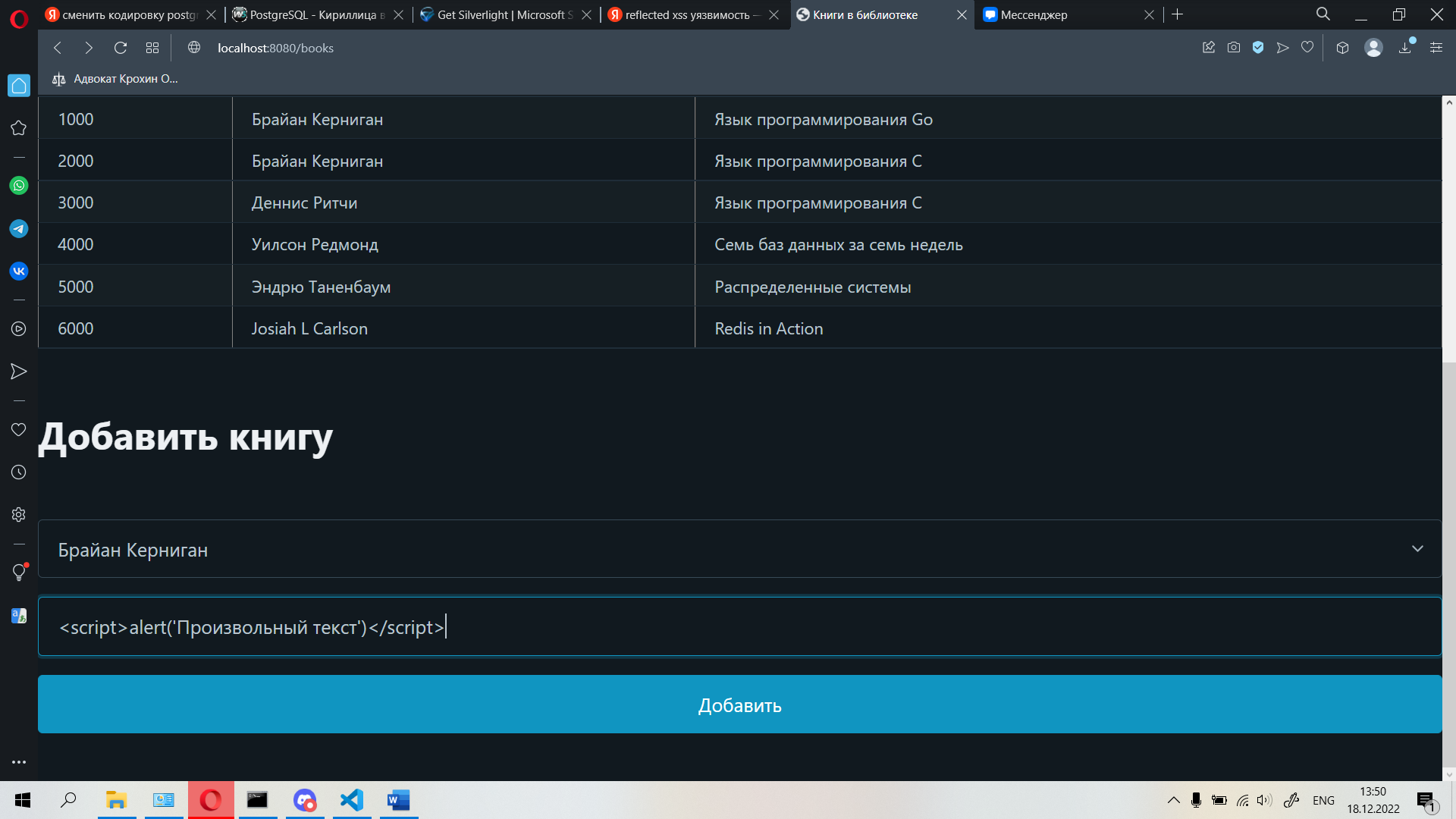


Рисунок 3 – Эксплуатация Stored XSS

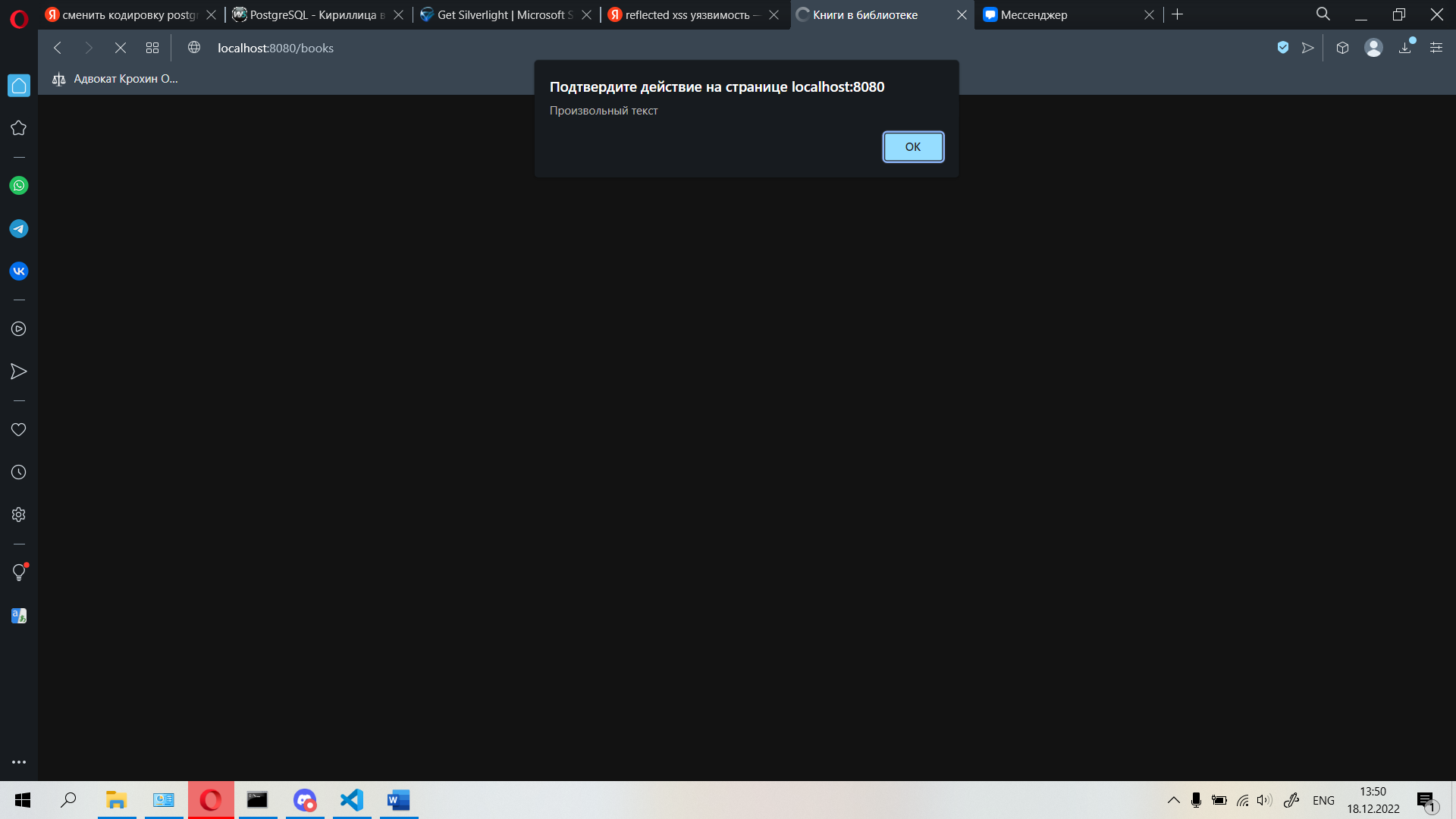


Рисунок – При авторизации на сайте каждый раз выполняется скрипт

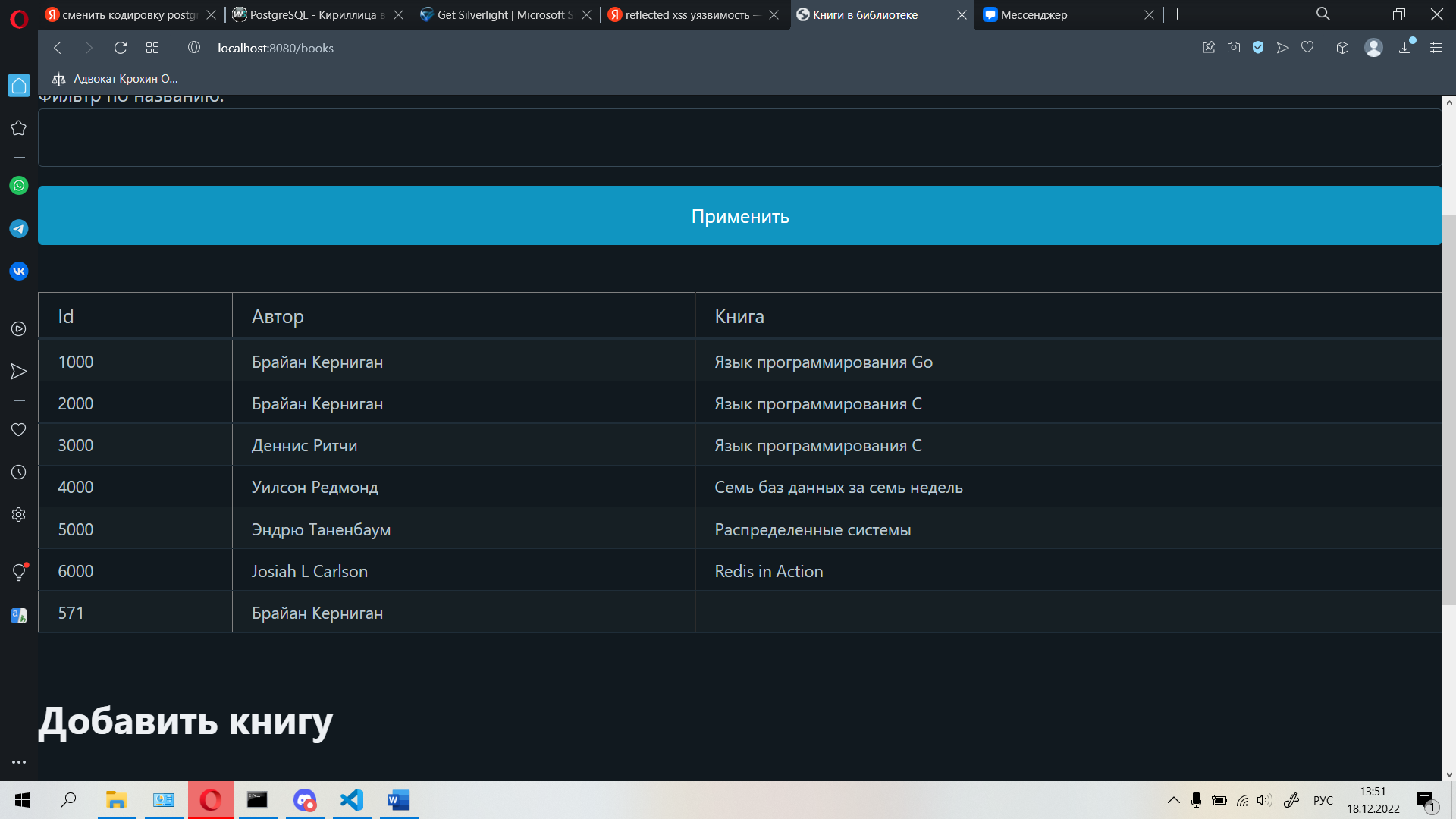
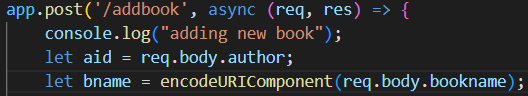
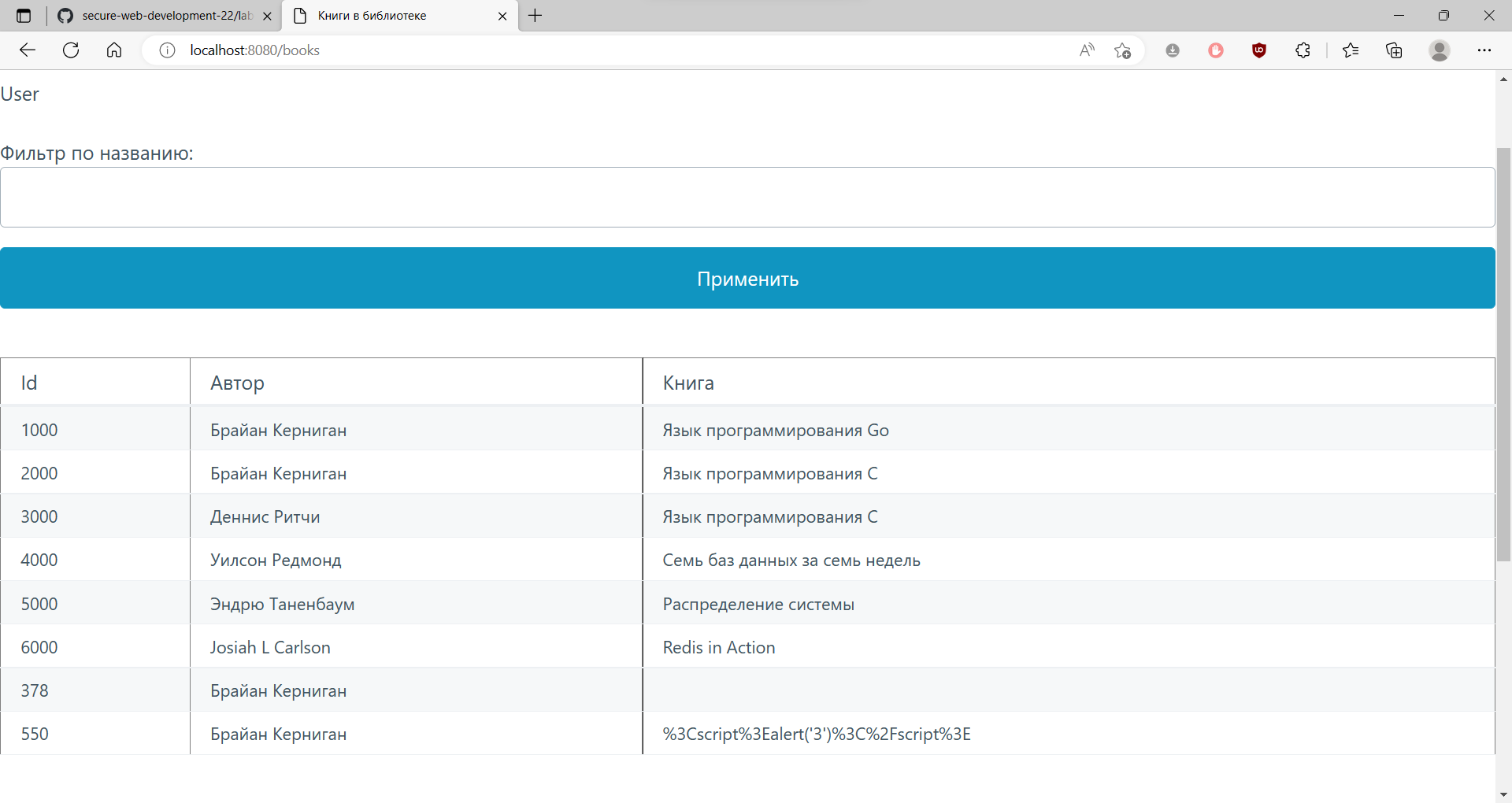


Рисунок – Добавилась «книга»

Для устранения этой уязвимости можно воспользоваться функцией encodeURIComponent(), которая заменяет служебные символы мнемониками. У этой функции есть минус: она кодирует таким образом и русские символы, поэтому уместно воспользоваться функцией replace, как в предыдущем пункте, однако если пользовательский ввод не предполагает русских символов, ей все же можно воспользоваться, поэтому мы ее все-таки рассмотрим. Эту функцию необходимо применить для пользовательского ввода req.body.bookname, считывающегося переменной bname:



Тогда скрипт не будет исполняться, так как служебные символы пользовательского ввода в базе данных будут выглядеть следующим образом:



**Session hijacking**

Потенциальная уязвимость находится при отправке запросов на сервер. Там можно поменять значение cookie на другого пользователя:

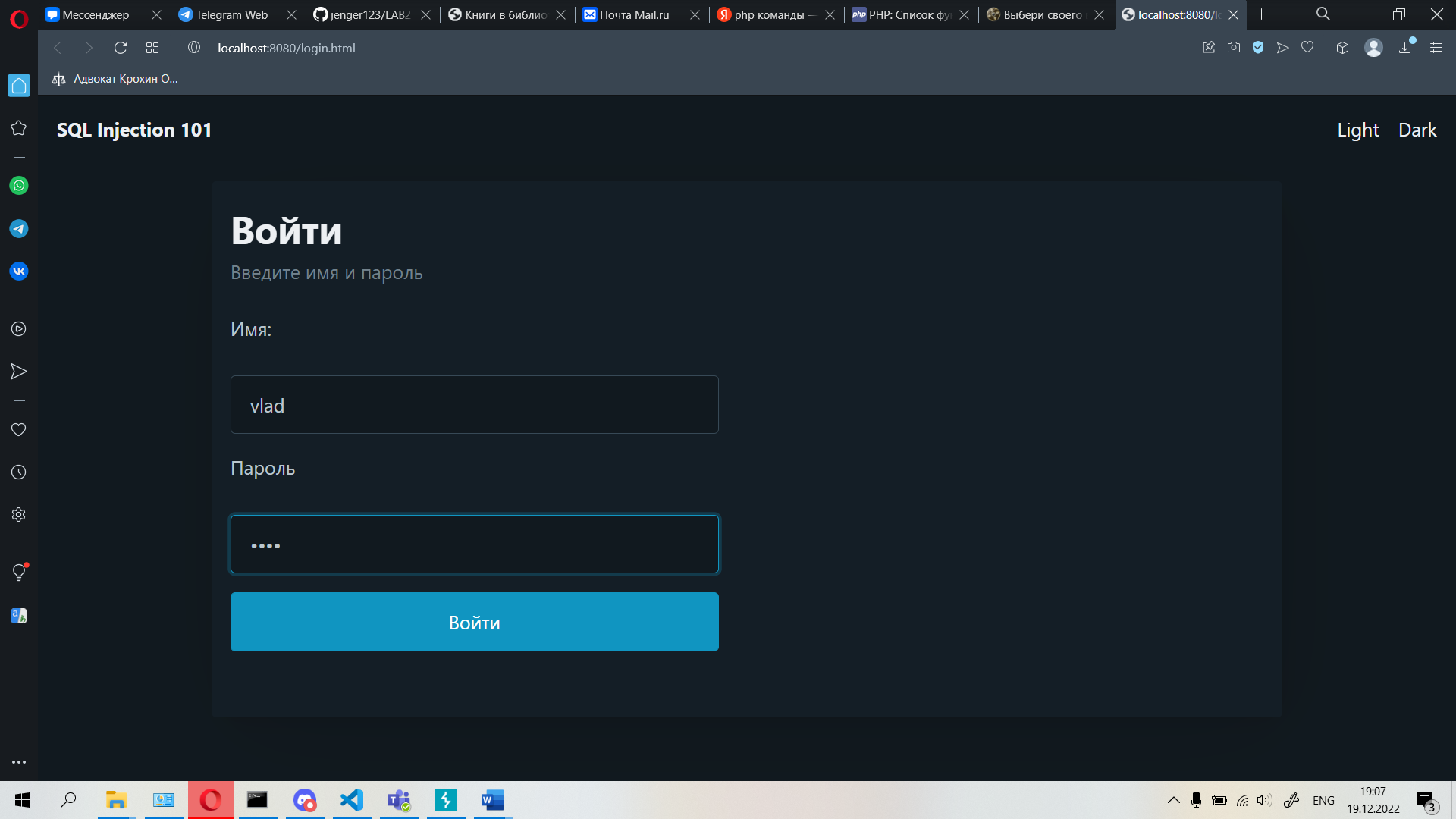


Рисунок 6 – Авторизуемся как vlad

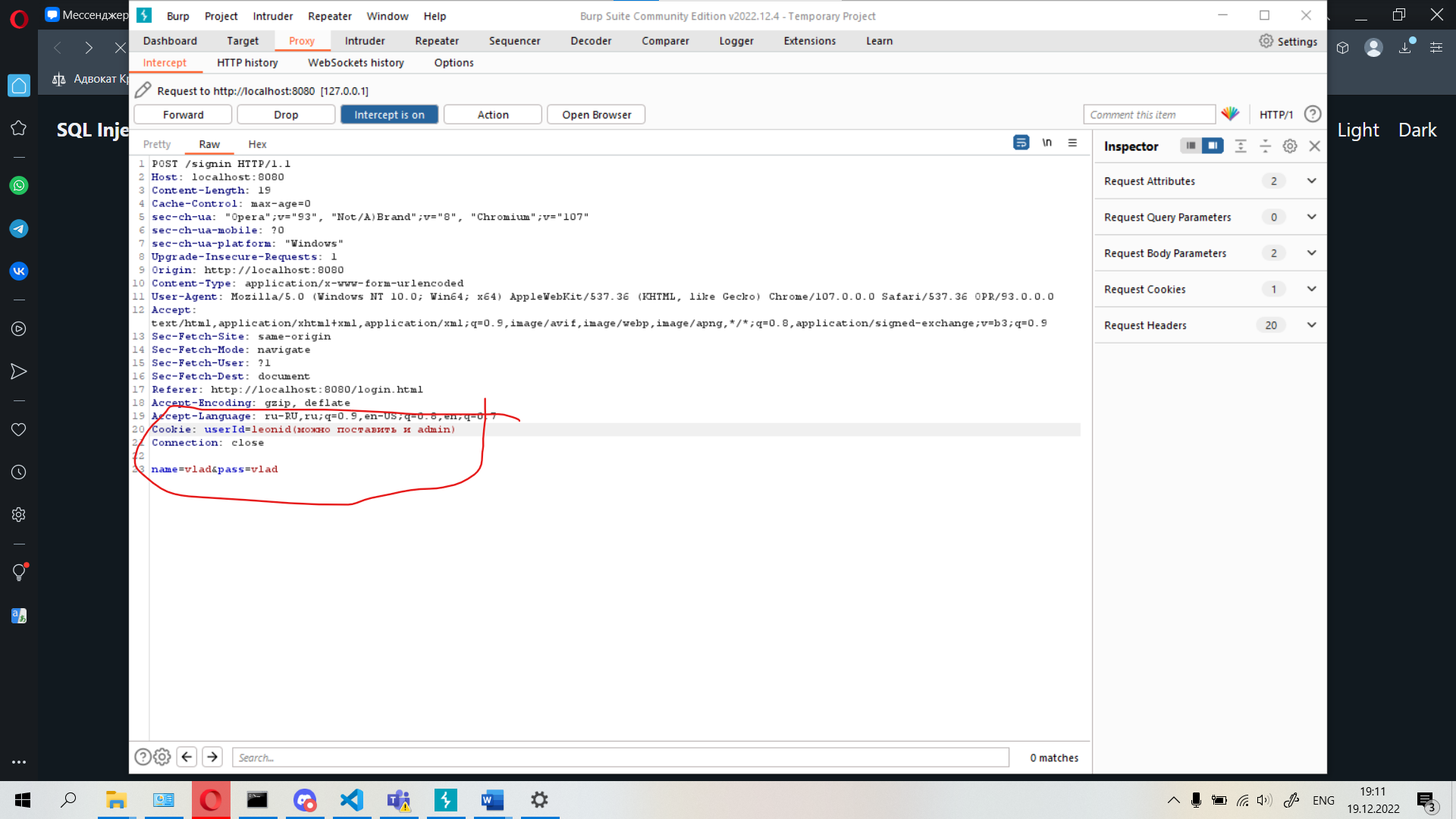
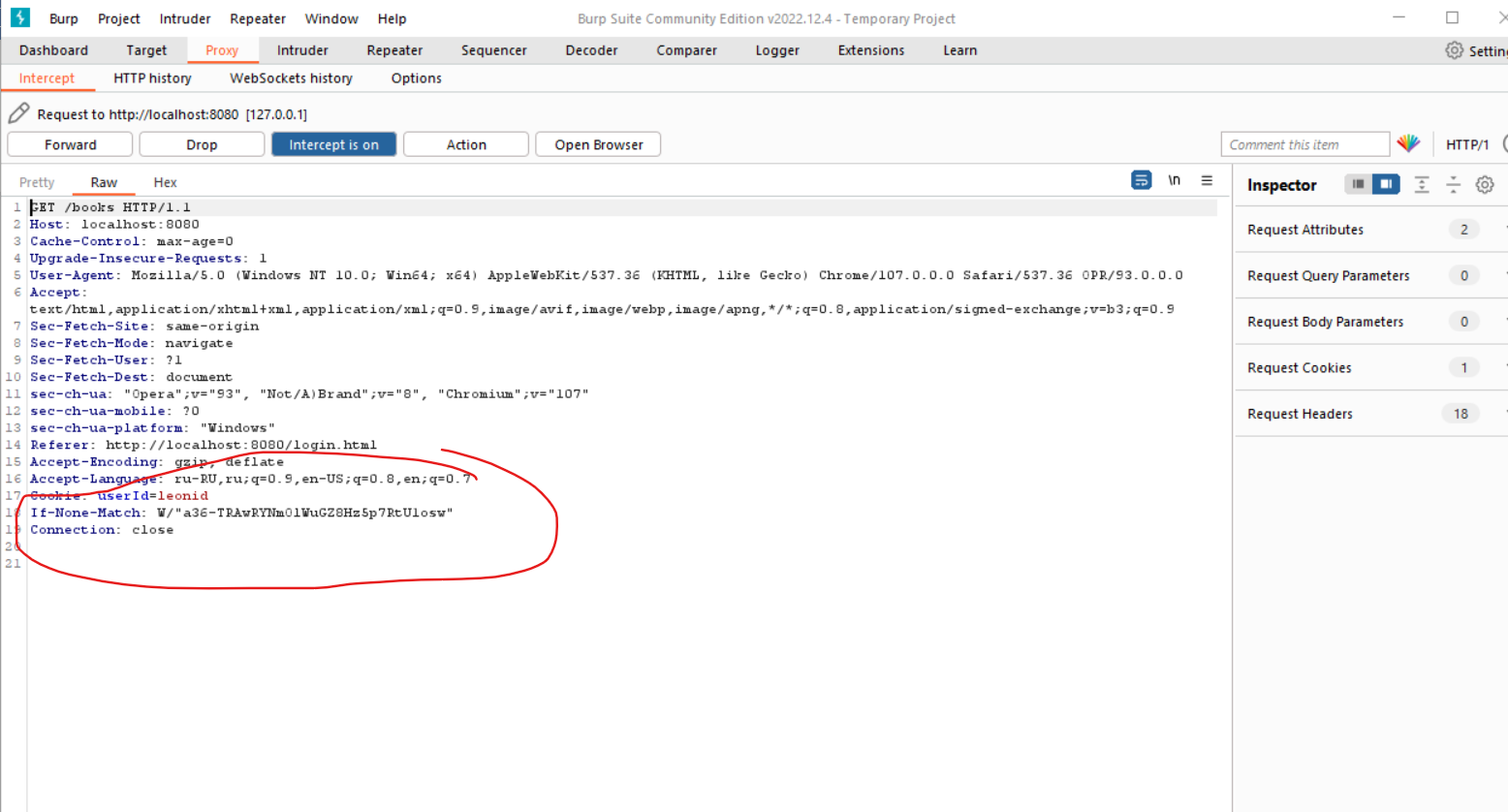


Рисунок 7 – меняем cookie

После этого сервер уже будет воспринимать нас как leonid:



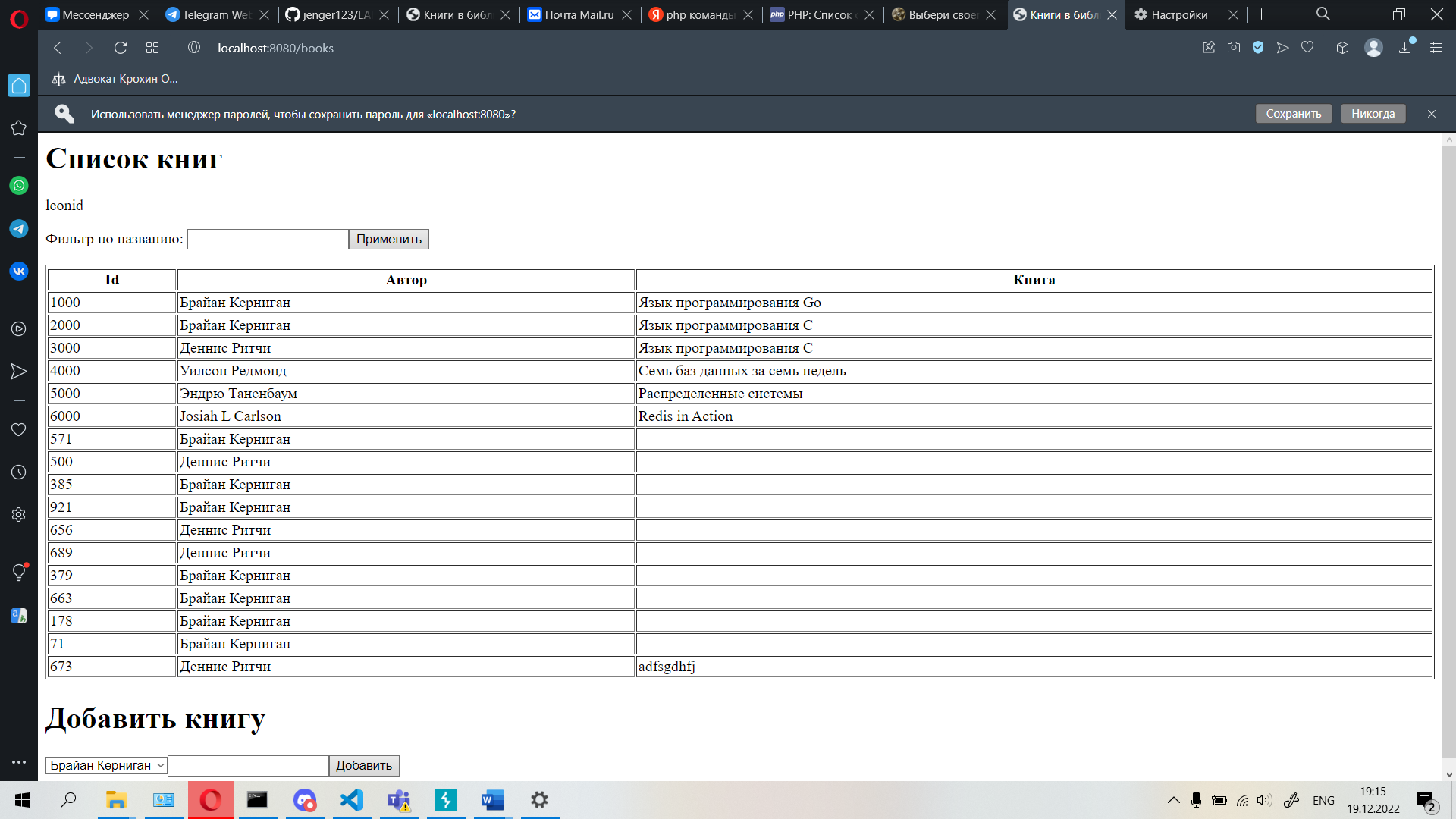


Рисунок 8 – Регистрировались как Vlad

Данная уязвимость является потенциальной, поскольку браузер не выполняет код, записанный в cookie.

**Сookie Injection**

Смысл данной уязвимости в следующем: если мы получили сессионный cookie, то можем получить доступ к защищенному ресурсу, добавив к запросу перехваченный куки.

В нашем случае защищаемый ресурс это адрес <http://localhost/books>

Заходим на этот ресурс и с помощью прокси-сервера добавляем к запросу сессионные куки:

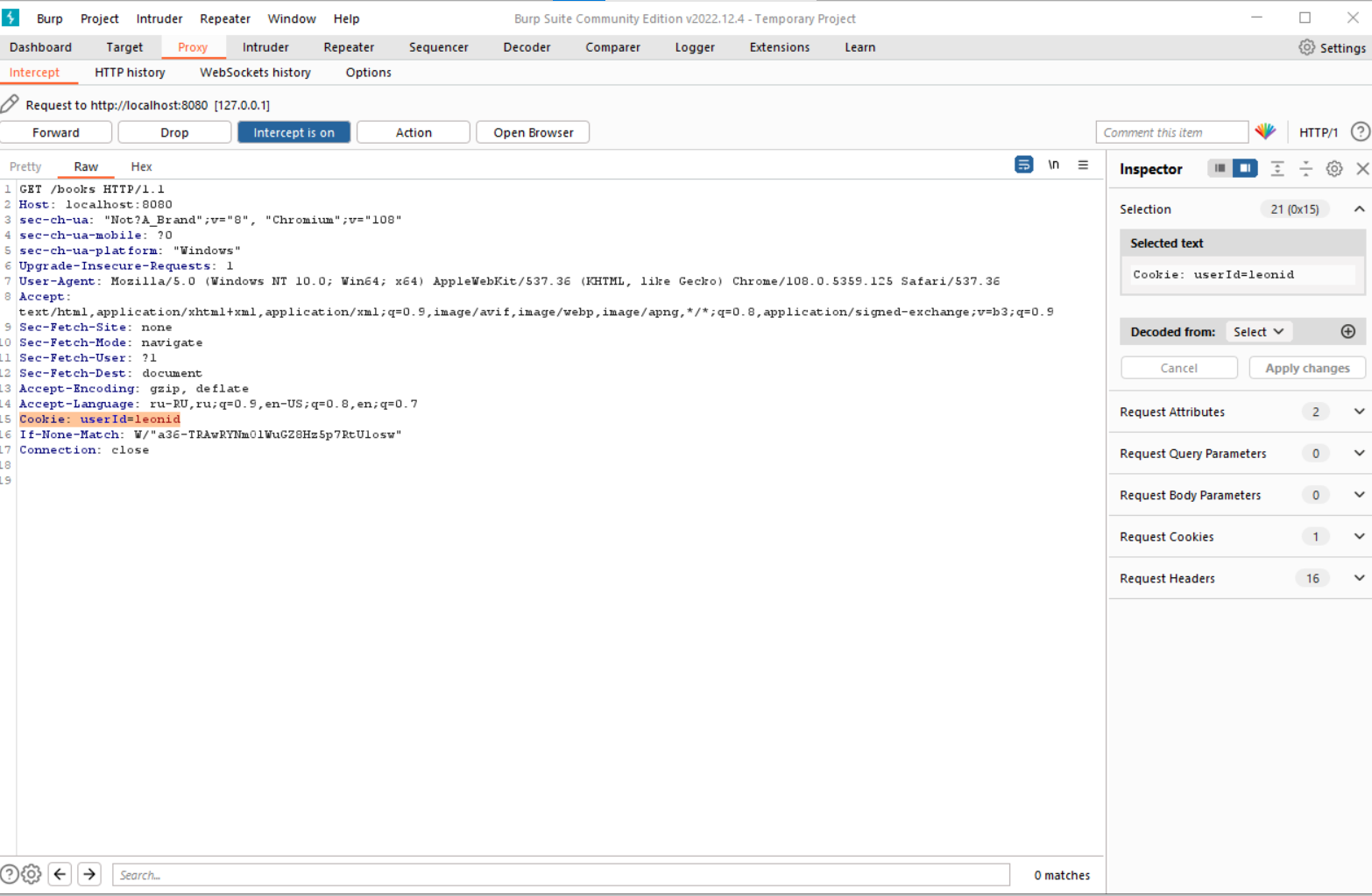


Рисунок 9 – Добавили сессионные куки

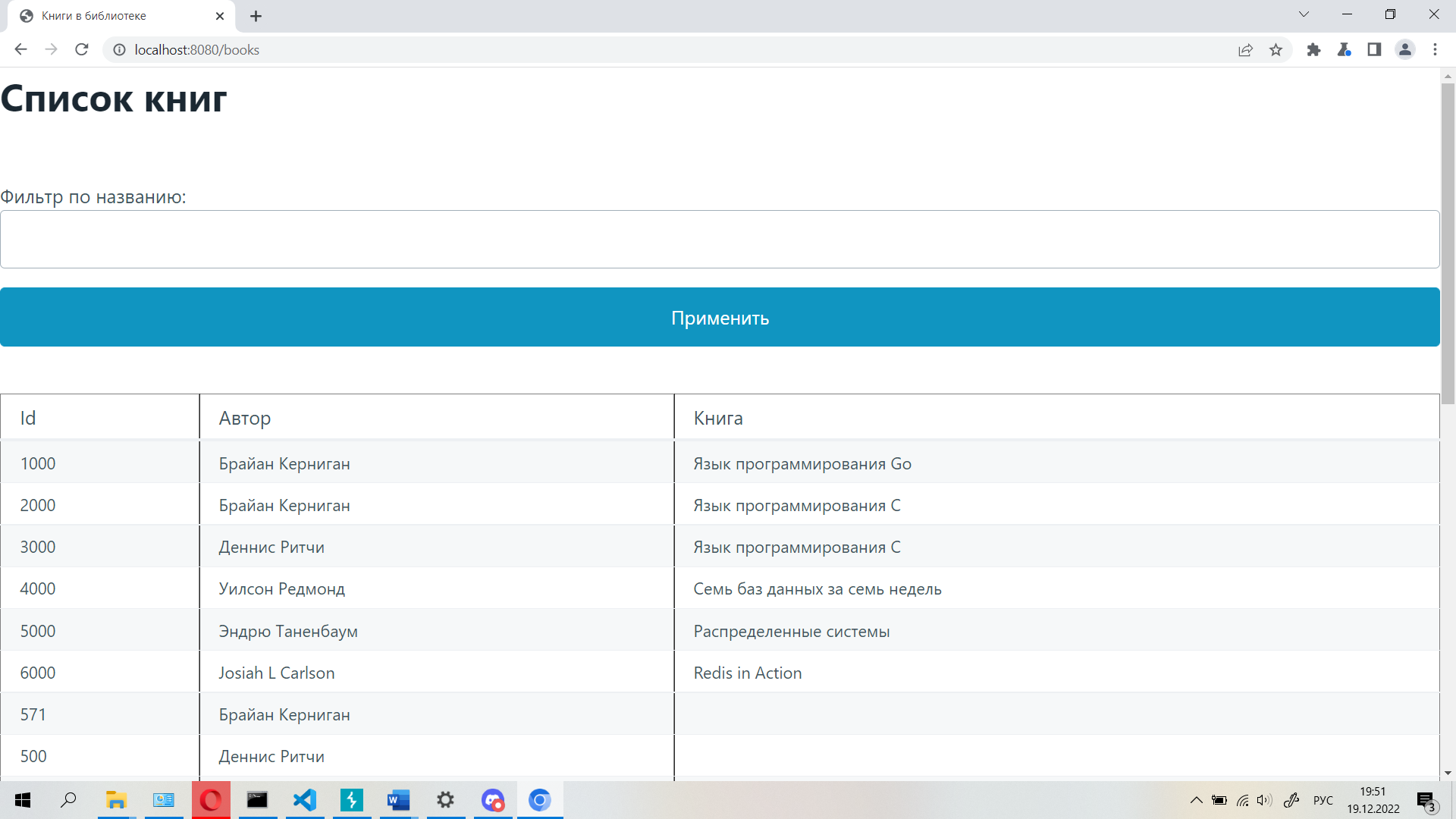


Рисунок 10 – Попали на защищенный ресурс минуя авторизацию с помощью перехваченного куки