УДК 004

Использование CSRF с Springfox

Еровлева Регина Викторовна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Еровлев Павел Андреевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье будут рассмотрены возможности csrf атак на сайты, возможные методы защиты и реализация в Spring boot. Исследование будет проводится в среде разработки IntelijIdea с фрейморков Spring.
Ключевые слова: Spring Security, Spring, Spring boot

Using CSRF with Springfox

Erovleva Regina Viktorovna
Sholom-Aleichem Priamursky State University
Student

Erovlev Pavel Andreevich
Sholom-Aleichem Priamursky State University
Student

Abstract
The article discusses the possibilities of csrf attacks on sites, possible protection methods and implementation in Spring boot. The research will be carried out in the IntelijIdea development environment with Spring frameworks.
Keywords: Spring Security, Spring, Spring boot

При создании web приложений всегда встает вопрос о безопасности своего продукта, никому не хочется лишиться клиентов, или их данных, а если web приложение — это сайт банка, то возможно и денег. Сейчас существует самый популярный метод взлома web приложений, через csrf токен. Он позволяет отправить post запрос с нужными данными на сервер так, чтобы сервер его обработал.

Цель работы – описать возможность scrf, методы взломов Rest API сайтов и методы их защиты, реализация в Spring.

Исследованиями в данной теме занимались следующие авторы. Е.А.Зайцева рассмотрела варианты возможных атак межсайтовой подделкой запросов [1]. И.С.Эккерт и М.О.Тюмин провели исследования

существующих уязвимостей в веб-приложениях, привели их классификации и возможности предотвратить их [2]. Т.В.Азарнова, П.В.Полухин разработали механизмы повышения эффективности и качества тестирования уязвимостей межсайтового подделки запросов (csrf) с помощью фаззинга [3]. М.Т.Нгуен провел тестирование методов машинного обучения в задаче классификации http запросов с применение технологии tf-idf [4].

Если разработчик разрабатывает REST API с помощью Spring и Spring Security, то, вероятно он, сталкивался с разделом конфигурации CSRF. Подделка межсайтовых запросов или CSRF — это метод, при котором пользователя обманом заставляют отправить запрос приложению, в которое он вошел.

Например, предположим, что клиент банка XYZ в настоящее время вошел в web-приложение. И в web-приложении есть REST API для перевода денег другим людям(рис.1).

```
http://bank-xyz.com/api/transfer?amount=1000&target=OtherAccount
```

Рисунок 1 – Post запрос

В данном примере этот вызов API переведет 1000 рублей в «OtherAccount». Если сам пользователь лично выполнит этот запрос, проблем не будет. Однако, что, если кто-то с плохими намерениями пришлет эту ссылку знакомому, скрытую в электронном письме, и он случайно нажмет на нее? В таком случае знакомы бы неохотно отправил этому человеку 1000 рублей.

Можно предположить, что это плохой API в целом, потому что он использует GET запросы. Да, это так, но изменение метода на HTTP не поможет тоже. Хакер может создать свою собственную веб-страницу со скрытой формой для банковского API (рис.2).

```
<form
  name="transfer"
  action="http://bank-xyz.com/api/transfer"
  method="post"
  style="display: none">
  <input type="hidden" name="amount" value="1000" />
  <input type="hidden" name="target" value="OtherAccount" />
  <button>Submit</button>
</form>
```

Рисунок 2 – Скрытая форма перевода

А затем эта форма автоматически отправляется через некоторый JavaScript (рис.3).

```
document.transfer.submit()
```

Рисунок 3 – JS скрипт

Таким образом, даже если использовать правильный метод HTTP, пользователя все равно можно обманом заставить выполнить определенные запросы.

Одно из решений данной проблемы - заставить REST API генерировать уникальный токен и отправлять его как часть ответа. Затем клиент должен передать этот токен обратно в REST API для следующего вызова, после чего REST API ответит новым токеном и так далее.

Это предотвращает злонамеренные запросы к REST API, поскольку никто не знает, какой будет токен.

Spring Security имеет встроенный механизм, и его можно включить с помощью «.csrf()» конфигурации (рис.4).

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .csrf()
        .csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse());
}
```

Рисунок 4 – Подключение csrf

При использовании этой конфигурации ответ будет содержать файл cookie с именем «XSRF-TOKEN», и Spring будет ожидать заголовок «X-XSRF-TOKEN» с каждым запросом.

Добавление защиты CSRF означает, что нужно настроить клиентскую часть для отправки этого заголовка обратно. Эта настройка также применяется к клиенту пользовательского интерфейса Swagger, созданному с помощью Springfox. Springfox уже поставляется с поддержкой CSRF, это прописано в документации. Ее можно включить с помощью конфигурации (рис.5)

```
@Bean
SecurityConfiguration security() {
    return SecurityConfigurationBuilder.builder()
        .enableCsrfSupport(true)
        .build();
}
```

Рисунок 5 – Подключение csrf в Springfox

Эта конфигурация пока не работает должным образом. Причина этого, заключается в том, что токен CSRF извлекается только один раз во время загрузки страницы.

Из-за этого ограничения будет работать только первый запрос. После этого нужно обновить страницу пользовательского интерфейса Swagger, чтобы попробовать любой последующий запрос.

Поэтому пользовательский интерфейс Swagger имеет собственный механизм для изменения запроса перед его отправкой. Это можно сделать через «requestInterceptor» конфигурацию. Однако Springfox не предлагает возможности настроить «requestInterceptor» и это означает, что единственное решение - предоставить собственный пользовательский интерфейс.

Для начала - добавим «Swagger UI WebJar» в проект. Если используется Maven, то можно сделать это, добавив зависимость (рис.6).

```
<dependency>
    <groupId>org.webjars</groupId>
    <artifactId>swagger-ui</artifactId>
    <version>3.51.2</version>
</dependency>
```

Рисунок 6 – Добавление зависимости

После этого можно добавить файл с именем «index.html» в папку статических ресурсов (рис.7).

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Swagger UI</title>
    <link rel="stylesheet" type="text/css" href="/webjars/swagger-ui/3.51.2/swagger-ui.css" />
    <link rel="icon" type="image/png" href="/webjars/swagger-ui/3.51.2/favicon-32x32.png" sizes="32x32" />
    <link rel="icon" type="image/png" href="/webjars/swagger-ui/3.51.2/favicon-16x16.png" sizes="16x16" />
    <link rel="stylesheet" type="text/css" href="./index.css" />
</head>

<body>
<div id="swagger-ui"></div>

<script src="/webjars/swagger-ui/3.51.2/swagger-ui-bundle.js" charset="UTF-8"> </script>
<script src="/webjars/swagger-ui/3.51.2/swagger-ui-standalone-preset.js" charset="UTF-8"> </script>
<script src="./index.js" charset="UTF-8"></script>
</body>
</html>
```

Рисунок 7 – Создание страницы «index.html»

Этот HTML-файл основан на исходном HTML-файле, предоставленном Swagger UI. Основное отличие состоит в том, что изменен путь для ссылки на «WebJar» и удалены встроенные стили и встроенные скрипты. Вместо

этого создадим файлы index.js и index.css в той же папке. В index.css вставим тот же стиль, что и в исходном HTML-шаблоне (рис.8).

```css
html {
  box-sizing: border-box;
  overflow: -moz-scrollbars-vertical;
  overflow-y: scroll;
}

*, *:before, *:after {
  box-sizing: inherit;
}

body {
  margin:0;
  background: #fafafa;
}
```

Рисунок 8 – Создание css

После завершениятсобственной настройки можно безопасно отключить пользовательский интерфейс Swagger, созданный Springfox. Чтобы отключить его, добавим в application.properties свойство (рис.9).

```
springfox.documentation.swagger-ui.enabled=false
```

Рисунок 9 – Отключение интерфейса

Следующим шагом является правильная настройка пользовательского интерфейса Swagger, чтобы больше не получать пустую страницу. Пользовательский интерфейс Swagger имеет множество настраиваемых параметров. Но их необязательно нужно настраивать все это самостоятельно, так как Springfox уже предоставляет API с большинством настроенных значений:

«/swagger-resources» содержит конфигурацию для пользовательского интерфейса Swagger и для ссылки на соответствующую спецификацию Swagger JSON. Эта спецификация JSON содержит информацию о том, какие конечные точки содержит API.

Далее идет «/swagger-resources/configuration/ui» API. Этот API содержит конфигурацию того, как должен выглядеть пользовательский интерфейс. Например, эта конфигурация сообщает, какая информация должна быть свернута в пользовательском интерфейсе, а какая - нет. Большинство этих параметров можно настроить, предоставив «UiConfiguration» компонент.

И, наконец, есть «/swagger-resources/configuration/security» API. Этот API содержит конфигурацию, необходимую для настройки OAuth2 и других параметров, связанных с безопасностью.

Теперь, чтобы загрузить конфигурацию с пользовательским интерфейсом Swagger, добавим функцию в index.js(рис.10).

```
async function fetchJsonConfig(url) {
  const credentials = 'same-origin';
  const headers = {
    'Accept': 'application/json',
    'Content-Type': 'application/json'
  };
  const response = await fetch(url, {headers, credentials});
  return await response.json();
}
```

Рисунок 10 – Настройка JS

После этого добавляем в файл JS функцию (рис.11).

```
window.onload = async function() {
  const uiConfig = await fetchJsonConfig('/swagger-resources/configuration/ui');
  const securityConfig = await fetchJsonConfig('/swagger-resources/configuration/security');
  const resources = await fetchJsonConfig('/swagger-resources');
  window.ui = SwaggerUIBundle({
    url: '',
    dom_id: '#swagger-ui',
    urls: resources,
    ...securityConfig,
    ...uiConfig,
    presets: [
      SwaggerUIBundle.presets.apis,
      SwaggerUIStandalonePreset
    ],
    plugins: [
      SwaggerUIBundle.plugins.DownloadUrl
    ],
    oauth2RedirectUrl: '/webjars/swagger-ui/3.51.2/oauth2-redirect.html',
    showMutatedRequest: true,
    modelPropertyMacro: null,
    parameterMacro: null,
    layout: 'StandaloneLayout'
  });
};
```

Рисунок 11 – Настройка JS

Теперь если обновить пустую страницу пользовательского интерфейса Swagger, то появляется интерфейс Swagger с перечисленными операциями (рис.12).

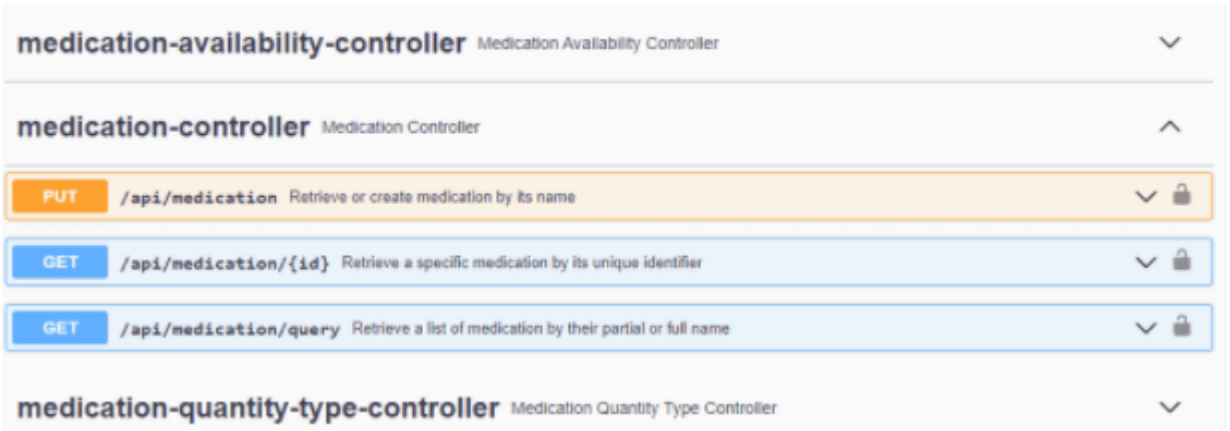

Рисунок 12 – Пользовательский интерфейс

Несмотря на то, что теперь можно правильно видеть операции, поддержка CSRF по-прежнему не включена. Чтобы включить ее, необходимо прочитать файл cookie XSRF-TOKEN и передать его в качестве заголовка.

Чтобы прочитать этот файл cookie, можно либо проанализировать «document.cookie» самостоятельно, либо использовать такую библиотеку, как «universal-cookie» (рис.13).

```
<script crossorigin src="https://unpkg.com/universal-cookie@4/umd/universalCookie.min.js"></script>
```

Рисунок 13 – Добавление скрипта cookie

После этого изменим «window.onload» функцию для загрузки библиотеки cookie (рис.14).

```
window.onload = async function() {
  const uiConfig = await fetchJsonConfig('/swagger-resources/configuration/ui');
  const securityConfig = await fetchJsonConfig('/swagger-resources/configuration/security');
  const resources = await fetchJsonConfig('/swagger-resources');

  const cookies = new UniversalCookie();

};
```

Рисунок 14 – Изменение JS

Затем настроим «SwaggerUIBundle» для чтение файла cookie и передачу его в качестве заголовка (рис.15).

```
window.ui = SwaggerUIBundle({
    url: '',
    dom_id: '#swagger-ui',
    urls: resources,
    ...securityConfig,
    ...uiConfig,


    requestInterceptor: request => {
      request.headers['X-XSRF-TOKEN'] = cookies.get('XSRF-TOKEN');
      return request;
    },

  });
```

Рисунок 15 – Настройка передачи cookie

Теперь если перезагрузить приложение, и зайти на начальную страницу, то можно увидеть, что токен был передан (рис.16).

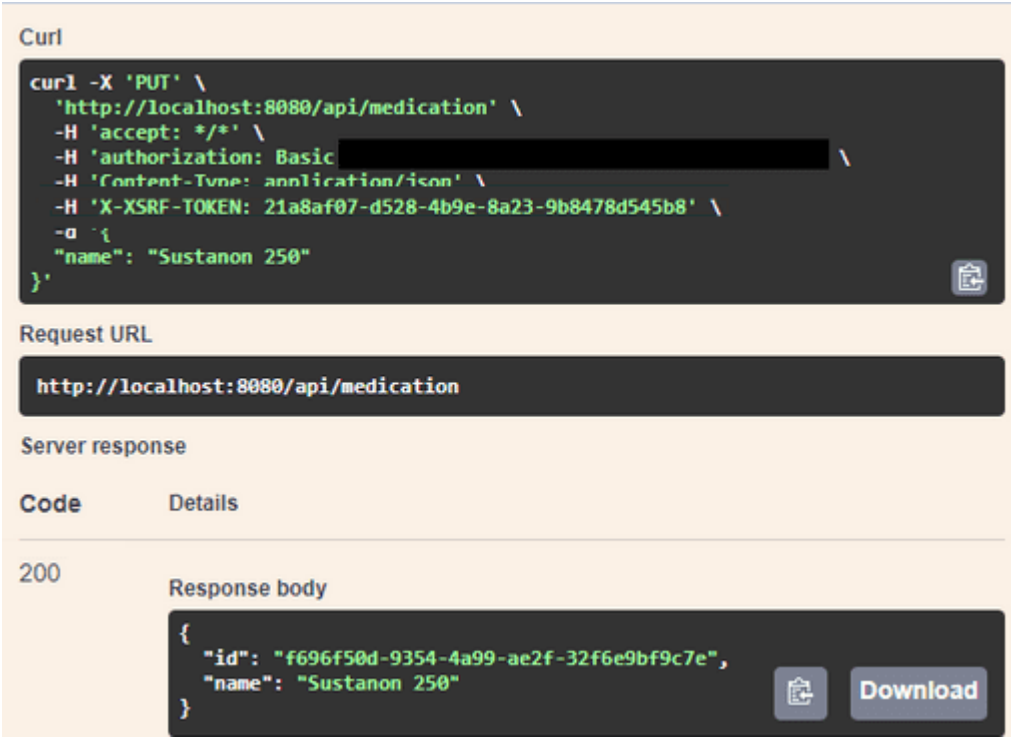

Рисунок 16 – Передача токена

Подделка межсайтовых запросов или CSRF - один из наиболее распространенных векторов атак. Фактически, до 2017 года он входил в топ-10 OWASP. Одна из причин, по которой его больше нет в этом списке, - встроенная поддержка такими фреймворками, как Spring Security.

Несмотря на то, что существуют другие контрмеры против CSRF, все же рекомендуется включить защиту CSRF. И хотя Springfox еще не поддерживает должным образом защиту CSRF, добавить ее самостоятельно, настроив пользовательскую страницу пользовательского интерфейса Swagger, несложно.

Библиографический список

1. Зайцева Е.А. Межсайтовая подделка запроса: пример атаки csrf // Advanced science 2018. №2(8). С. 151-153.
2. Эккерт И.С., Тюмин М.О. Основные уязвимости современных веб-приложений на примере csrf и xss уязвимостей // Аллея науки 2018. №8(24). С. 741-746.
3. Азарнова Т.В., Полухин П.В Механизмы повышения эффективности и качества тестирования уязвимостей межсайтового подделки запросов (csrf) с помощью фаззинга // Modern science 2016. №12(1). С. 11-18.
4. Нгуен М.Т. Тестирование методов машинного обучения в задаче классификации http запросов с применение технологии tf-idf // Системный анализ и информационные технологии 2019. №4. С. 119-131.