CZU: 004.65 DOI: 10.46727/c.steam-2023.p279-284

РАЗРАБОТКА ПРИЛОЖЕНИЯ С ОБРАБОТКОЙ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ SPRING SECURITY

Olga CERBU, dr. conf. universitar
https://orcid.org/0000-0002-6278-7115
Tatiana SESTACOVA, dr. conf universitar
https://orcid.org/0000-0002-6995-4254
Roman RUDI
Государственный Университет Молдовы, Факультет Информатики
Технический университет Молдовы

Резюме. В рамках данной работы была представлена возможность создания веб-приложения с использованием Spring Security. Фреймворк предоставляет конфигурацию различных аспектов безопасности приложения, такие как аутентификация, авторизация и защита от интернет-атак. В статье представлено краткое ознакомление с возможностями Spring Security в контексте создания веб-приложения на Java.
Ключевые слова: Spring Security, Authentication, Authorisation, CSRF, CORS.

Abstract. As part of this work, the possibility of creating a web application using Spring Security was presented. The framework provides configuration for various aspects of application security, such as authentication, authorization, and protection against Internet attacks. This article provides a brief introduction to the capabilities of Spring Security in the context of creating a web application in Java.
Keywords: Spring Security, Authentication, Authorisation, CSRF, CORS.

Введение

Spring Security — это платформа, обеспечивающая аутентификацию, авторизацию и защиту от распространенных атак. Благодаря первоклассной поддержке защиты как императивных, так и реактивных приложений, он является де-факто стандартом для защиты приложений на основе Spring[1].

Основные функции Spring Security включают:

1) Аутентификацию: Spring Security предоставляет механизмы аутентификации, позволяющие проверить личность пользователя. Это может включать в себя аутентификацию на основе пароля, токенов, сертификатов и других методов.
2) Авторизацию: После аутентификации Spring Security предоставляет возможность определить, какие действия и ресурсы доступны аутентифицированным пользователям. Это позволяет управлять правами доступа и определять, кто может выполнять какие операции.
3) Управление сеансами: Spring Security обеспечивает управление жизненным циклом сеансов пользователей, включая создание, хранение и завершение

сеансов. Это особенно важно для безопасности приложений, которые работают с чувствительными данными.

4) Защиту от атак: Spring Security предоставляет механизмы защиты от распространенных атак, таких как атаки CSRF (межсайтовой подделки запроса), инъекции SQL, XSS (межсайтового скриптования) и другие [2].

Spring Security интегрируется со всеми другими технологиями Spring, такими как Spring Boot, Spring MVC, Spring Data, Spring AOP, Spring Restful API, Spring OAuth, Spring Batch и другие. Он позволяет задавать параметры безопасности приложения.

Для добавления Spring Security к веб-приложению, достаточно добавить зависимость в pom.xml для Maven и в build.gradle, если используете Gradle.

Добавление зависимости для Maven:

```
<dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Аутентификация

Аутентификация в Spring Security играет ключевую роль в обеспечении безопасности приложений, позволяя идентифицировать пользователей и проверять их подлинность. Spring Security предоставляет различные способы аутентификации, а также множество настроек для обеспечения безопасности приложения. Основой настройки безопасности приложения является создание и настройка многочисленных фильтров безопасности, проверяющих различные аспекты запроса, такие как CORS, наличие CSRF-токена, проверка на правильность введённых данных, наличие сессионного токена или JWT и прочее.

Spring Security поддерживает различные способы аутентификации пользователей:

1) Через форму входа при создании Spring MVC приложения. Для этого используется задание фильтра в конфигурации приложения: SecurityFilterChain.
2) Через JSON запрос с данными логина и пароля. На сервере данные обрабатываются и сравниваются с имеющимися в базе данных. Пароль возможно захэшировать любим алгоритмом хэширования с добавлением соли. Часто этот метод используется при создании RESTful приложения.
3) Sessions – вид аутентификации по умолчанию. Сессии позволяют передавать в файле Cookie сессионный токен. Жизненный цикл токена истекает при закрытии браузера или по истечении времени, установленном в конфигурации приложения.

4) JWT – JSON Web Token – токен, который создаётся на сервере. В нём хранится необходимая для авторизации информация, такая как роль пользователя, дата истечения срока годности токена и прочее. JWT хранится в кэше браузера или в Cookie файле в зависимости от настройки на фронтенде и не истекает при закрытии браузера или выключении системы. Он отправляется каждый раз с запросом пользователя в качестве дополнительной нагрузки, чтобы авторизовать пользователя для доступа к соответствующим эндпоинтам.
5) OAuth 2.0 (Open Authorization 2.0) - это протокол авторизации, который позволяет веб-приложениям получать доступ к ресурсам от имени пользователя или другого приложения без необходимости раскрывать его учетные данные (например, пароль). OAuth 2.0 используется для авторизации пользователя и предоставления третьей стороне (клиенту) ограниченного доступа к ресурсам на сервере, где пользователь имеет учетную запись. Этот протокол широко используется для реализации аутентификации и авторизации в современных веб-приложениях и API.

Для настройки OAuth необходимо зайти на сайт, поддерживающий аутентификацию с помощью удалённого сервиса, такой как Google, GitHub или Facebook. Для каждого сайта есть документация, которая укажет, как добавить функционал аутентификации.

Пример конфигурации через форму входа:

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
     http
          .authorizeRequests(authorize -> authorize
               .anyRequest().authenticated()
          )
          .formLogin(withDefaults())
          .httpBasic(withDefaults());
     return http.build();
}
```

Пример конфигурации через OAuth2.0:

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
    return http
        .oauth2Login()
        .loginPage("/login")
        .and()
            .logout()
            .logoutSuccessUrl("/")
```

```
        .build();
}
```

Пример настройки конфигурации OAuth2.0 для GitHub в application.yml:

```
spring:
  security:
    oauth2:
      client:
        registration:
          github:
            client-id: id клиента
            client-secret: секретный ключ клиента
            scope:
              - email
              - public-profile
```

Авторизация

Авторизация в Spring Security позволяет контролировать доступ пользователей к защищенным ресурсам, таким как URL, методы и доменные объекты. Spring Security предоставляет гибкие инструменты для определения правил доступа и разграничения доступа пользователей. Давайте рассмотрим основные аспекты авторизации:

1) Роли (Roles): Spring Security позволяет определять роли, которые пользователи могут иметь. Вы можете присвоить роли пользователям и настроить правила доступа на основе ролей. Например, вы можете разрешить доступ только администраторам (hasRole('ADMIN')) или только зарегистрированным пользователям (hasRole('USER')).
2) Разрешения (Permissions): Вы можете определять разрешения (permissions) для конкретных действий в вашем приложении. Например, разрешения могут относиться к операциям CRUD (создание, чтение, обновление, удаление) над определенными ресурсами. Spring Security позволяет использовать разрешения для более детального контроля доступа.
3) Выражения (Expressions): Spring Security поддерживает выражения в правилах доступа, позволяя создавать более сложные и гибкие правила. Вы можете использовать выражения SpEL (Spring Expression Language) для определения правил доступа, например: hasRole('ADMIN'), hasAuthority('ROLE_ADMIN’) и hasPermission('resource', 'read').
4) Аннотации: Spring Security предоставляет аннотации, такие как @PreAuthorize и @PostAuthorize, которые можно использовать над методами в контроллерах. Эти аннотации позволяют определить правила доступа к конкретным методам.

Пример доступа пользователя к методу с предварительной проверкой роли:

```
@RestController
public class MyController {
    @GetMapping("/admin")
    @PreAuthorize("hasRole('ADMIN')")
    public String adminPage() {
        return "admin";
    }
}
```

Методы обеспечения безопасности приложения

Spring Security предоставляет множество механизмов и фильтров, чтобы обеспечивать защиту приложений от распространенных атак. Ниже представлены некоторые из атак, от которых Spring Security обеспечивает защиту, и способы, как это происходит:

1) Фиксация сессии (`Session Fixation`): Spring Security защищает от атаки фиксации сессии, предотвращая злоумышленникам возможность использования фиксированных сессионных идентификаторов. По умолчанию, Spring Security генерирует новую сессионную идентификацию после аутентификации пользователя.
2) `Clickjacking` - это атака, при которой злоумышленник скрывает вредный контент за видимым интерфейсом вашего сайта, чтобы обмануть пользователя и получить его действия. Spring Security предоставляет защиту от clickjacking с помощью заголовка HTTP X-Frame-Options, который позволяет вам определить, какие сайты могут встраивать ваш сайт в <iframe>. Spring Security по умолчанию защищает от кликджекинга, отключая рендеринг страницы внутри iframe. Это делается путем установки заголовка X-Frame-Options в значение DENY.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .headers()
            .frameOptions()
            .sameOrigin();
}
```

Этот код настроит заголовок X-Frame-Options на "SAMEORIGIN", что означает, что ваш сайт может быть встроен только в фреймы с тем же источником (origin).

3) Межсайтовая подделка запросов (`Cross-Site Request Forgery, CSRF`): Spring Security предоставляет встроенную защиту от атак CSRF с использованием токена CSRF (CSRF token). Токены CSRF генерируются и

вставляются в формы, и их проверка выполняется при отправке запросов. Это предотвращает атаки, при которых злоумышленники пытаются выполнить действия от имени авторизованного пользователя.

Если вы хотите отключить CSRF-защиту, вы можете использовать disable(), но это рекомендуется только в определенных случаях, например, если ваше приложение не имеет состояния и не предполагает ввод данных пользователем.

4) CORS-атака (Cross-Origin Resource Sharing). Настройка CORS позволяет определять правила доступа к ресурсам на разных источниках (доменах) и обеспечивает безопасное взаимодействие между клиентскими приложениями и сервером.

Настройка CORS в конфигурационном файле:

```
@Bean
CorsConfigurationSource corsConfigurationSource() {
     CorsConfiguration configuration = new CorsConfiguration();
     configuration.setAllowedOrigins(Arrays.asList("https://example.co
m"));
     configuration.setAllowedMethods(Arrays.asList("GET","POST"));
     UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
     source.registerCorsConfiguration("/**", configuration);
     return source;
}
```

5) Защита от инъекций: Spring Security не является инструментом для защиты от инъекций напрямую, но обеспечивает защиту на уровне аутентификации и авторизации, что уменьшает риск атак, связанных с инъекциями. Для защиты от инъекций вам также рекомендуется использовать валидацию входных данных и параметризованные запросы при работе с базой данных.

Выводы

Spring Security – мощный фреймворк, который предоставляет много встроенных функций для защиты от различных веб-атак, предоставляет возможность аутентификации и авторизации. В статье были представлены лишь два варианта аутентификации: с помощью сессий и JWT, – однако существуют и другие методы. Помимо интеграции с классической системой Spring MVC он также поддерживает работу с RESTful и SOAP сервисами. Spring Security – гибкий фреймворк, который можно компоновать с множеством сервисов Spring.

Библиография

1. Spring Docs - https://docs.spring.io/
2. WALLS, C. Spring in Action. 6th Edition.