[Bài đọc] Spring Security - Spring Integrations

Spring Security & Spring Framework

Đối với hầu hết bài viết này, bạn chỉ chỉ định cấu hình bảo mật trên tầng web của ứng dụng của bạn. Bạn đã bảo vệ một số URL nhất định bằng antMatcher hoặc regexMatchers bằng WebSecurityConfigurerAdapter’s DSL. Đó là một cách tiếp cận hoàn toàn tốt và tiêu chuẩn để bảo mật.

Ngoài việc bảo vệ cấp web của bạn, còn có ý tưởng "bảo vệ theo chiều sâu". Điều đó có nghĩa là ngoài việc bảo vệ URL, bạn có thể muốn bảo vệ chính logic nghiệp vụ của mình. Hãy nghĩ: @Controllers, @Components, @Services hoặc thậm chí @Repositories của bạn.

Method Security

Cách tiếp cận đó được gọi là bảo mật phương thức và hoạt động thông qua các chú thích(annotations)  mà về cơ bản bạn có thể đặt trên bất kỳ phương thức công khai nào của Spring bean của mình. Bạn cũng cần bật bảo mật phương thức một cách rõ ràng bằng cách đặt chú thích @EnableGlobalMethodSecurity trên ApplicationContextConfiguration của bạn.

@Configuration

@EnableGlobalMethodSecurity(

prePostEnabled = true, *// (1)*

securedEnabled = true, *// (2)*

jsr250Enabled = true) *// (3)*

public class YourSecurityConfig extends WebSecurityConfigurerAdapter{

}

(1)   Thuộc tính prePostEnabled cho phép hỗ trợ các chú thích @PreAuthorize và @PostAuthorize của Spring. Hỗ trợ có nghĩa là Spring sẽ bỏ qua chú thích này trừ khi bạn đặt cờ thành true.

(2)   Thuộc tính secureEnabled cho phép hỗ trợ chú thích @Secured. Hỗ trợ có nghĩa là Spring sẽ bỏ qua chú thích này trừ khi bạn đặt cờ thành true.

(3)   Thuộc tính jsr250Enabled cho phép hỗ trợ chú thích @RolesAllowed. Hỗ trợ có nghĩa là Spring sẽ bỏ qua chú thích này trừ khi bạn đặt cờ thành true.

**Sự khác biệt giữa @PreAuthorize, @Secured và @RolesAllowed là gì?**

@Secured và @RolesAllowed về cơ bản giống nhau, mặc dù @Secured là một chú thích dành riêng cho Spring đi kèm với sự phụ thuộc vào spring-security-core và @RolesAllowed là một chú thích được chuẩn hóa, trong sự phụ thuộc javax.annotation-api. Cả hai chú thích đều lấy một chuỗi authority/role làm giá trị.

@ PreAuthorize / @ PostAuthorize cũng là các chú thích cụ thể (mới hơn) của Spring và mạnh hơn các chú thích ở trên, vì chúng có thể chứa không chỉ các authority/role mà còn chứa bất kỳ biểu thức SpEL hợp lệ nào.

Chúng ta hãy xem các chú thích này hoạt động:

@Service

public class SomeService {

@Secured("ROLE\_CALLCENTER") *// (1)*

*// == @RolesAllowed("ADMIN")*

public BankAccountInfo get(...) {

}

@PreAuthorize("isAnonymous()") *// (2)*

*// @PreAuthorize("#contact.name == principal.name")*

*// @PreAuthorize("ROLE\_ADMIN")*

public void trackVisit(Long id);

}

}

(1)   Như đã đề cập, @Secured có authority/role làm tham số. @RolesAllowed, tương tự như vậy. Lưu ý: Hãy nhớ rằng @RolesAllowed ("ADMIN") sẽ kiểm tra một thẩm quyền được cấp ROLE\_ADMIN.

(2)   Như đã đề cập, @PreAuthorize chấp nhận các authorities, nhưng cũng có bất kỳ biểu thức SpEL hợp lệ nào.

Spring Security & Spring Web MVC

Đối với việc tích hợp với Spring WebMVC, Spring Security cho phép bạn thực hiện một số việc:

* Ngoài antMatchers và regexMatchers, bạn cũng có thể sử dụng mvcMatchers. Sự khác biệt là, trong khi antMatchers và regexMatchers về cơ bản khớp các chuỗi URI với các ký tự đại diện, mvcMatchers hoạt động giống hệt như @RequestMappings.
* Đưa principal(Đại diện cho user) hiện đã được xác thực của bạn vào phương thức @Controller / @RestController.
* Đưa CSRFToken phiên hiện tại  của bạn vào phương thức @Controller / @RestController.
* Xử lý chính xác bảo mật để xử lý yêu cầu không đồng bộ.

@Controller

public class MyController {

@RequestMapping("/messages/inbox")

public ModelAndView findMessagesForUser(@AuthenticationPrincipal CustomUser customUser, CsrfToken token) { // (1) (2)

// .. find messages for this user and return them ...

}

}

(1)   @AuthenticationPrincipal sẽ chèn principal nếu người dùng được xác thực hoặc null nếu không có người dùng nào được xác thực. Chính này là đối tượng đến từ UserDetailsService / AuthenticationManager của bạn!

(2)   Hoặc bạn có thể đưa CSRFToken phiên hiện tại vào từng phương thức.

Nếu bạn không sử dụng chú thích @AuthenticationPrincipal, bạn sẽ phải tự mình tìm nạp Principal thông qua SecurityContextHolder.

@Controller

public class MyController {

@RequestMapping("/messages/inbox")

public ModelAndView findMessagesForUser(CsrfToken token) {

SecurityContext context = SecurityContextHolder.getContext();

Authentication authentication = context.getAuthentication();

if (authentication != null && authentication.getPrincipal() instanceof UserDetails) {

CustomUser customUser = (CustomUser) authentication.getPrincipal();

*// .. find messages for this user and return them ...*

}

*// todo*

}

}

Spring Security & Thymeleaf

Spring Security tích hợp tốt với Thymeleaf. Nó cung cấp một phương ngữ Spring Security Thymeleaf đặc biệt, cho phép bạn đặt các biểu thức bảo mật trực tiếp vào các mẫu HTML Thymeleaf của bạn.

**<div** sec:authorize="isAuthenticated()"**>**

This content is only shown to authenticated users.

**</div>**

**<div** sec:authorize="hasRole('ROLE\_ADMIN')"**>**

This content is only shown to administrators.

**</div>**

**<div** sec:authorize="hasRole('ROLE\_USER')"**>**

This content is only shown to users.

**</div>**

*(Nguồn: https://www.marcobehler.com/guides/spring-security#\_spring\_integrations)*