[Bài đọc] Spring Security - Cross-Site-Request-Forgery

Có một loạt các cuộc tấn công phổ biến mà Spring Security giúp bạn chống lại. Nó bắt đầu với các cuộc tấn công theo thời gian (tức là Spring Security sẽ luôn băm mật khẩu được cung cấp khi đăng nhập, ngay cả khi người dùng không tồn tại) và kết thúc với các biện pháp bảo vệ chống lại các cuộc tấn công kiểm soát bộ nhớ cache, đánh hơi nội dung, nhấp chuột, viết mã trang web và hơn thế nữa.

Không thể đi vào chi tiết của từng cuộc tấn công trong phạm vi của hướng dẫn này. Do đó, chúng ta sẽ chỉ xem xét một biện pháp bảo vệ khiến hầu hết những người mới sử dụng Spring Security không thích nhất: Cross-Site-Request-Forgery.

Cross-Site-Request-Forgery: CSRF

Spring Security theo mặc định sẽ bảo vệ mọi yêu cầu POST (hoặc PUT / DELETE / PATCH) đến bằng mã thông báo CSRF hợp lệ.

Điều đó nghĩa là gì?

CSRF & HTML được hiển thị phía máy chủ

Hãy tưởng tượng một form chuyển khoản ngân hàng hoặc bất kỳ biểu mẫu nào (như biểu mẫu đăng nhập), được @Controllers của bạn hiển thị với sự trợ giúp của công nghệ tạo khuôn mẫu như Thymeleaf hoặc Freemarker.

**<form** action="/transfer" method="post"**>** *<!-- 1 -->*

**<input** type="text" name="amount"**/>**

**<input** type="text" name="routingNumber"**/>**

**<input** type="text" name="account"**/>**

**<input** type="submit" value="Transfer"**/>**

**</form>**

Khi đã bật Spring Security, bạn sẽ không thể gửi biểu mẫu đó nữa. Vì CSRFFilter của Spring Security đang tìm kiếm một thông số ẩn bổ sung trên bất kỳ yêu cầu POST (PUT / DELETE) nào: cái gọi là mã thông báo CSRF.

Nó tạo ra một mã thông báo như vậy, theo mặc định, trên mỗi phiên HTTP và lưu trữ nó ở đó. Và bạn cần đảm bảo đưa nó vào bất kỳ biểu mẫu HTML nào của mình.

**Mã thông báo CSRF & Thymeleaf**

Vì Thymeleaf tích hợp tốt với Spring Security (khi được sử dụng cùng với Spring Boot), bạn có thể chỉ cần thêm đoạn mã sau vào bất kỳ biểu mẫu nào và bạn sẽ nhận được mã thông báo tự động được đưa vào, từ session, vào biểu mẫu của mình. Thậm chí tốt hơn, nếu bạn đang sử dụng "th: action" cho biểu mẫu của mình, Thymeleaf sẽ tự động đưa trường ẩn đó vào cho bạn mà không cần phải thực hiện theo cách thủ công.

**<form** action="/transfer" method="post"**>** *<!-- 1 -->*

**<input** type="text" name="amount"**/>**

**<input** type="text" name="routingNumber"**/>**

**<input** type="text" name="account"**/>**

**<input** type="submit" value="Transfer"**/>**

**<input** type="hidden" name="${\_csrf.parameterName}" value="${\_csrf.token}"**/>**

**</form>**

*<!-- OR -->*

**<form** th:action="/transfer" method="post"**>** *<!-- 2 -->*

**<input** type="text" name="amount"**/>**

**<input** type="text" name="routingNumber"**/>**

**<input** type="text" name="account"**/>**

**<input** type="submit" value="Transfer"**/>**

**</form>**

(1)   Ở đây, chúng ta đang thêm thông số CSRF theo cách thủ công.

(2)   Ở đây, chúng ta đang sử dụng hỗ trợ biểu mẫu của Thymeleaf.

**CSRF & Thư viện Template khác**

Ta không thể trình bày tất cả các thư viện tạo khuôn mẫu trong phần này, nhưng bạn luôn có thể đưa CSRFToken vào bất kỳ phương thức @Controller nào của mình và chỉ cần thêm nó vào model để hiển thị nó trong một view hoặc truy cập trực tiếp dưới dạng thuộc tính yêu cầu HttpServletRequest .

@Controller

public class MyController {

@GetMaping("/login")

public String login(Model model, CsrfToken token) {

*// the token will be injected automatically*

return "/templates/login";

}

}

**CSRF & React hoặc Angular**

Mọi thứ có một chút khác biệt đối với ứng dụng Javascript, chẳng hạn như ứng dụng Single Page React hoặc Angular. Đây là những gì bạn cần làm:

Định cấu hình Spring Security để sử dụng CookieCsrfTokenRepository, nơi sẽ đặt CSRFToken vào cookie "XSRF-TOKEN" (và gửi cookie đó đến trình duyệt)

Làm cho ứng dụng Javascript của bạn nhận giá trị cookie đó và gửi nó dưới dạng tiêu đề "X-XSRF-TOKEN" với mọi yêu cầu POST (/ PUT / PATCH / DELETE).

**Tắt CSRF**

Nếu bạn chỉ cung cấp một API REST không trạng thái trong đó bảo vệ CSRF không có ý nghĩa gì, bạn sẽ vô hiệu hóa hoàn toàn bảo vệ CSRF. Đây là cách bạn sẽ làm điều đó:

@EnableWebSecurity

@Configuration

public class WebSecurityConfig extends

WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.csrf().disable();

}

}

*(Nguồn: https://www.marcobehler.com/guides/spring-security#\_common\_exploit\_protections)*