CSRF and XSS Protection in Spring Security

What is a csrf attack:

A CSRF (Cross-Site Request Forgery) attack is a type of malicious exploit where an attacker tricks a user into unintentionally performing actions on a web application in which they are authenticated. The attack occurs by manipulating the user's authenticated session to execute unauthorised actions without their knowledge.

Let's understand this using an example:

Scenario: Suppose there's a banking API endpoint for transferring funds:

Endpoint: bank.com/api/transfer
HTTP Method: POST/GET
Parameters: *toAccount, amount*
CSRF Vulnerability:
Without proper CSRF protection, an attacker could craft a malicious website or link that triggers an unintended fund transfer when visited by an authenticated user.

There are multiple ways through which an attack can happen:

1. If the API is **GET**

    - **Link/Image** – For example, the attacker can convince the victim to click this link to execute the transfer or open the page. The image source will be the API. (In case)

```
<a href="http://bank.com/api/transfer?toAccount=attackerAccount&amount=1000">
Show Dog Pictures
</a>
```

2. If the API is **POST**

    - In this case, the attacker will need a <form>:

```
<form action="http://bank.com/transfer" method="POST">
    <input type="hidden" name="toAccount" value="5678"/>
    <input type="hidden" name="amount" value="1000"/>
    <input type="submit" value="Show Kittens Pictures"/>
</form>
```

However, the form can be submitted automatically using JavaScript:

```
<body onload="document.forms[0].submit()">
<form>
```

Implementation of CSRF Protection in Hotel Application

We will implement CSRF protection in the Hotel Application we made in Lecture 15, which has a form login.

Steps:

1. We need to make changes to the **HotelSecurityConfig** class.

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class HotelSecurityConfig {

      @Autowired
      UserDetailsService userDetailsService;

      @Bean
        public  SecurityFilterChain  securityFilterChain(HttpSecurity  http)
throws Exception {
        http
            .authorizeRequests()
            .antMatchers("/user/register").permitAll()
            // Add more authorisation rules as needed
            .and()
            .rememberMe().userDetailsService(userDetailsService)
            .and()
            .formLogin()
            .loginPage("/login").permitAll()
            .and()
            .logout().deleteCookies("remember-me")
            .and()
            .csrf(); // Enable CSRF protection

        return http.build();
      }
}
```

2. We need to include the CSRF Token in the HTML Login Form:

```
<form th:action="@{/login}" method="post" th:object="${loginForm}"
    th:action="@{/login}" style="max-width: 350px; margin: 0 auto;">

    <!-- Include CSRF token -->
    <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />

    <!-- Rest of your form -->
</form>
```