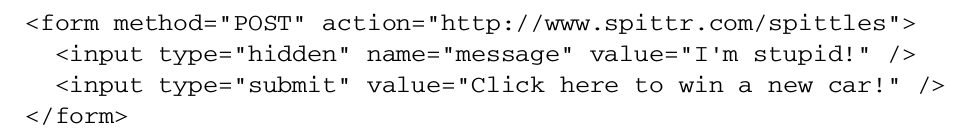
***Preventing Cross-site request forgery***

But what if that POST request comes from another website? And what if that POST request is the result of submitting the following form on that other site?



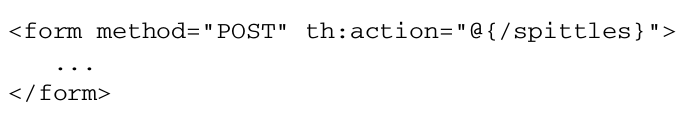
Let’s say that you’re tempted by the offer of winning a new car and you click the button—you’ll submit the form to http://www.spittr.com/spittles. If you’re already logged in to spittr.com, you’ll be broadcasting a message that tells everyone that you made a bad decision.

This is a simple example of a cross-site request forgery ( CSRF ). Basically, a CSRF attack happens when one site tricks a user into submitting a request to another server, Starting with Spring Security 3.2, CSRF protection is enabled by default. In fact, unless you take steps to work with CSRF protection or disable it, you’ll probably have trouble getting the forms in your application to submit successfully.

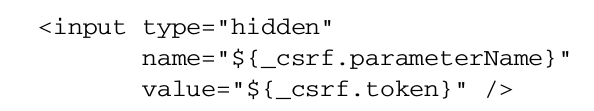
Spring Security implements CSRF protection with a synchronizer token. State changing requests (for example, any request that is not GET , HEAD , OPTIONS , or TRACE ) will be intercepted and checked for a CSRF token. If the request doesn’t carry a CSRF token, or if the token doesn’t match the token on the server, the request will fail with a CsrfException.

This means that any forms in your application must submit a token in a \_csrf field. And that token must be the same as the one calculated and stored by the server so that it matches up when the form is submitted. Fortunately, Spring Security makes this easy for you by putting the token into the

request under the request attributes. If you’re using Thymeleaf for your page template, you’ll get the hidden \_csrf field automatically, as long as the <form> tag’s action attribute is prefixed to come from the Thymeleaf namespace:



If you’re using JSP for page templates, you can do something very similar:



Even better, if you’re using Spring’s form-binding tag library, the <sf:form> tag will automatically add the hidden CSRF token tag for you. Another way of dealing with CSRF is to not deal with it at all. You can disable Spring Security’s CSRF protection by calling

csrf().disable() in the configuration, as shown in the next listing:

Be warned that it’s generally not a good idea to disable CSRF protection. If you do, you leave your application open to a CSRF attack.