Go to content
Menu

- Home
- Web development
- Github
- Twitter

SAML Authentication with AngularJS and Spring Security

Published on Oct 26, 2015

At Onegini we're developing a web application with AngularJS in the front end and Spring Boot with Spring Security in the back-end. During the initial development, we were using basic authentication (username & password) for the log-in, but this doesn't integrate well in a corporate environment. The users have an existing corporate account and expect Single Sign On for all internal applications that is done via SAML.

Spring Boot doesn't have a module (yet) to handle SAML within Spring Security, but there's an example project on GitHub that does the job. It contains a Java class for the Spring Security configuration. The configuration needs to be adapted to your own environment and then the basics work. However, there are a few things that are tricky if you combine it with a Single Page App (SPA) like our AngularJS front-end.

CSRF protection

The SAML flow doesn't work if CSRF protection is enabled in the Spring Security configuration. ~~We have solved this with two implementations of the `WebSecurityConfigurerAdapter`. The configuration for the `"/saml/**"` endpoints is without CSRF protection. The other implementation handles the other endpoints of the application and has CSRF protection enabled.~~ You can solve this in Spring Security 4 with the following configuration:

```
http
    .csrf()
    .csrfTokenRepository(csrfTokenRepository())
    .ignoringAntMatchers("/saml/**")
    .and()
    .addFilterAfter(csrfHeaderFilter, CsrfFilter.class);
```

Log-in with redirects

Spring has an extensive tutorial about Angular JS and Spring Security, but it assumes that the user can enter their username and password in the SPA which sends the credentials to the server and gets a result. This doesn't match the SAML flow. An unauthorised user is redirected to the Identity Provider (IdP). The IdP presents a log-in form if the user doesn't have a valid session and after log-in the IdP redirects them back to the application (Service Provider) with a SAML assertion that contains user information.

Return to the correct URL

The default settings of the ui-router in AngularJS produces URLs like `http://localhost/#/products/details/12345`. Your Servlet gets a request for `/` and after a successful SAML authentication you are sent to `http://localhost/`. The location hash `#/products/details/12345` is lost during the authentication. This is no issue if your users should always see the main page of the application, but in our case we wanted to support deep linking.

The solution was to enable the hmtl5mode in the $locationProvider and add a base tag to the HTML. When this feature is enabled, the URLs look like `http://localhost/products/details/12345`. The ServletRequest will contain the full URL and the user is redirected to the same URL after completing the authentication. This means you have to create a request mapping with Spring MVC to handle `/products/details/12345`. This mapping can return the same view as the root request mapping of your application.

There’s a catch: it won’t work for legacy browsers that have no support for history.pushState (MSIE 9, Opera Mini, Android 4.1).

Prevent redirects for XMLHttpRequests

Even if your application is great, people will leave their browsers for a lunch break. After the lunch break their session is lost. In a classic web application the user clicks on a link or submits a form and is redirected to the log-in page or IdP. In our SPA the action triggers an XMLHttpRequest (XHR) to exchange data or fetch a template fragment.

Spring Security will return a 302 redirect to the SAML log-in which redirects to some other page to log-in. This is not the response you expected for the request of product 12345. It’s impossible to intercept the 302 redirect on the client side so you have to prevent it on the server side.

The back-end needs to identify your request as XHR, but AngularJS doesn’t send a header for it if you use version 1.3 or up. Add the following line to the config of your AngularJS module:

```
$httpProvider.defaults.headers.common["X-Requested-With"] = 'XMLHttpRequest';
```

Now you can identify an XHR in the back-end when this header is present.

Instead of redirecting an unauthorised XHR we respond with a “401 — Unauthorized” status by extending the SAMLEntryPoint class that you have configured for Spring Security SAML:

```
public class XhrSamlEntryPoint extends SAMLEntryPoint {

  @Override
  public void commence(HttpServletRequest request, HttpServletResponse response,
                       AuthenticationException e) throws IOException, ServletException {
    if (isXmlHttpRequest(request) && e instanceof InsufficientAuthenticationException) {
      response.sendError(HttpServletResponse.SC_UNAUTHORIZED, e.getMessage());
      return;
    }
    super.commence(request, response, e);
  }

  private boolean isXmlHttpRequest(HttpServletRequest request) {
    return "XMLHttpRequest".equalsIgnoreCase(request.getHeader("X-Requested-With"));
  }
}
```

We want to show a message to the user that the session is lost and they have to log in again. The first step was to configure an interceptor in the module config:

```
$provide.factory('notAuthorizedInterceptor', function ($q, $rootScope) {
  return {
    responseError: function (response) {
      if (response.status === 401) {
        $rootScope.unauthorized = true;
      }
      return $q.reject(response);
    }
  }
});

$httpProvider.interceptors.push('notAuthorizedInterceptor');
```

If the back-end responds with a 401 status code, it sets the `unauthorized` variable of the `$rootScope` to `true`. Now we can show a message to the user:

```
<div ng-if="$root.unauthorized === true">
  <p>Please <a href="javascript:window.location.reload()">log in</a> again</p>
</div>
<div ng-hide="$root.unauthorized === true">
  <div ui-view="main"></div>
</div>
```

We want the browser to reload the page by making a request to the server that is not an XHR. The `$window` service of AngularJS prevents a complete reload and that’s the reason why the href contains `javascript:window.location.reload()`. This reload results in a redirection to the SAML IdP and it brings you back to the application, but now as a logged in user. Mission accomplished.