http://www.mindissoftware.com/spring-security-oauth2/

**Spring Security with OAuth2**

For a cloud-native application, security is a big concern and brings new challenges. Luckily, the “silver bullet”, i.e, the open source software, makes life so easy. This time, it’s Spring security and OAuth2.

**1. OAuth2 Introduction**

Though OAuth2 is an authorization protocol, due to the close relationship of authentication and authorization, many OAuth2 servers such as Facebook, Twitter, or GitHub provider both authentication and authorization services.

In a simplified view, OAuth2 workflow involves a resource owner (using a user-agent such as a browser), an OAuth2 client, an OAuth2 server and a resource server. An OAuth2 server has two endpoints: one authorization uri and one access token uri. The authorization endpoint is used to authenticate a resource owner and ask the resource owner to grant the requested scope. The access token endpoint is used to generate an access token for the requested scope. A resource server also has a resource uri that is ued by OAuth2 client to access resources.

When [an auth workflow](https://tools.ietf.org/html/rfc6749#section-4) starts, an OAuth2 client first directs the resource owner to send a grant request to the authorization endpoint. In the request, the client provides client id, client secret , requested scope, local state and a redirection uri. The client id, client secret and redirection uri are pre-configured in an OAuth2 server. When the OAuth2 server receives the grant request, it starts the authentication and grant process. At the end, it sends the grant or deny result back to the OAuth2’s redirection uri. The result includes an an authorization code and the client’s original local state.

Then the OAuth2 client requests an access token from the the authorization server’s token endpoint by including the authorization code received in the previous step. The authorization code is used to authenticate the OAuth2 client. If valid, the authorization server responds back with an access token, an option refresh token and other parameters.

If everything works well, the OAuth2 client uses the access token to request resources from the resource server’s endpoint uri.

When including @EnableOAuth2Sso in a Spring boot application, an OAuth2 client authentication filter is added to the Spring http security chain. It can be used to authenticate users and request access code for resource access.

**2. Spring Security**

The best thing about Spring security is that it is [self-contained and plug-and-play](http://docs.spring.io/spring-security/site/docs/current/reference/html/technical-overview.html).

**2.1. Core Components**

* **SecurityContextHolder** : the place to store security context data. Default is ThreadLocal storage. Can be changed to MODE\_GLOBAL or MODE\_INHERITABLETHREADLOCAL.
* **SecurityContext**: hold security data and possibly request-specific data.
* **Authentication**: representing of a principal in Spring security. It includes granted authoritiy and user details.
* **GrandtedAuthority**: granted permissions.
* **UserDetails**: basic user information.
* **UserDetailsService**: the DAO for user details and granted authoritiy.

**2.2. Authentication in Spring**

When an application obtains a username and a password, it creates a token that is an instance of UsernamePasswordAuthenticationToken, which is an instance of the Authentication interface. The token is passed to an instance of AuthenticationManager for validation. On success, a fully populated Authentication is returned and a security context is established by calling SecurityContextHolder.getContext().setAuthentication(…​).

In Spring Web application, when a unauthenticated user access a secured resource, ExceptionTranslationFilter will catch an exception and launch an AuthenticationEntryPoint. An authentication system has its AuthenticationEntryPoint implementation to start the authentication token collection. The token is verified by AuthenticationManager to create Authentication in SecurityContextHolder.

To store SecurityContext between requests, SecurityContextPersistenceFilter by default stores it as an HttpSessionattribute in http request. A stateless RESTful web service authenticate on every request. SecurityContextPersistenceFilter is also used to clear SecurityContextHolder after each request.

In Spring security, the default implementation of AuthenticationManager is called ProviderManager. The ProviderManager delegates to a list of configured AuthenticationProvider instances. Each provider will either throw an exception or return a valid Authentication object. For example, DaoAuthenticationProvideruses an UserDetailsService to get UserDetails to authenticate a user.

**2.3. Authorization in Spring**

Spring security uses Filters to authorize access to secure objects such as web requests. An instance of AbstractSecurityInterceptor has a consist workflow:

1. Get “configuration attributes” of a secure object.
2. Call AccessDecisionManager.decide() with secure object, configuration attributes and a valid Authentication to authorize the access.
3. Optionally change Authentication for the request.
4. Let the request work.
5. Call AfterInvocationManager if configured.

An example AccessDecisionManager implementation is RoleVoter. It uses ROLE\_prefixed configuration attribute string to authorize access.

#### 4. Web Security Spring has a number of filters to process a request. Security-related filters in their execution order are:

* SecurityContextPersistenceFilter to setup SecurityContext.
* ConcurrentSessionFilter to update SessionRegistry.
* Authentication filters such as UsernamePasswordAuthenticationFilter or BasicAuthenticationFilter to authenticate a request.
* RememberMeAuthenticationFilter to remember a request from a cookie.
* AnonymousAuthenticationFilter: if still not authenticated, creates an anonymous Authentication object.
* ExceptionTranslationFilter to catch any security exception so that either returns an error or launch an appropriate AuthenticationEntryPoint. It saves the current request before invoking AuthenticationEntryPoint thus retries the request after authentication.
* FilterSecurityInterceptor decides which security constrains apply to a request.

The execution order of the filters matters. The order is represented by an integer number. For example, the default order number of ConcurrentSessionFilter is 200. Spring executes filters based on their order numbers, from small to big.

Spring security uses DelegatingFilterProxy to decide how a request should be handled by different filters.

**2.4. Security configuration**

When add @EnableWebSecurity to an instance of WebSecurityConfigurerAdapter, a set of default security settings, including enabling basic authentication, login form, CSRF attack protection etc, are in effect. Next, one needs to register the DispatcherServlet by implementing the WebApplicationInitializer interface, usually by extending an abstract class such as AbstractSecurityWebApplicationInitializer.

To specify what authentication is used and where to apply authorization, we need to customize the WebSecurityConfigurerAdapter.configure(HttpSecurity http) method.

Spring security has build in support for a /logout endpoint which will clean up the session data. However, it’s a POST request. When CSRF protection is enable, a POST request needs to provide a token to be included in the request. The token can be configured to be sent in cookie or in http header.

**3. Authentication with OAuth2**

[The Spring Boot with OAuth2 tutorial](https://spring.io/guides/tutorials/spring-boot-oauth2) is a very good introduction to Spring OAuth2.

**3.1 Authentication with OAuth2**

@EnableOAuth2Sso is used to enable OAuth2 authentication. When @EnableOAuth2Sso is used without a WebSecurityConfigurerAdapter, then all paths are secured. The OAuth2 client filter is inserted before BasicAuthenticationFilter and an authentication entry point is configured. If there is an existing WebSecurityConfigurerAdapter provided by the user and annotated with @EnableOAuth2Sso, it is enhanced by adding an OAuth2 client filter and an authentication entry point.

The OAuth2 client filter is OAuth2ClientAuthenticationProcessingFilter. It is used to acquire an OAuth2 access token from an authorization server and load an Authentication object into SecurityContext. However, it needs the OAuth2ClientContextFilter filter to redirect a request to authorization server. The two filters use UserRedirectRequiredException to pass control. Both need to be wired into the filter chain and OAuth2ClientContextFilter should execute before the main Spring security filter.

**3.2 Authorization with OAuth2**

An OAuth2 server provides a set of endpoints to answer requests for authorization code and access code. @EnableAuthorizationServer does all the dirty works by default.

@EnableResourceServer declares a resource server that is protected by the access token. By default, it creates a security filter with @Order(3). It should run before the main application security.

*Written on August 10, 2016*