Table of Contents

Once the plugin is installed, you are ready to go. The default configuration is to use signed JWT's for tokens. However, you can select any other token storage strategy:

1. Grails Cache.
2. Memcached.
3. GORM.
4. Redis.
5. Provide your own: implement TokenStorageService and register it in `resources.groovy` as `tokenStorageService`.

Plugin configuration

This plugin depends on Spring Security Core 2.x. Make sure your application is compatible with that version first.

This plugin is compatible by default with Spring Security core traditional, form-based authentication. The important thing to remember is: you have to separate the filter chains, so different filters are applied on each case.

The stateless, token-based approach of this plugin is incompatible with the HTTP session-based approach of Spring Security, core, so the trick is to identify what URL patterns have to be stateless, and what others have to be stateful (if any).

To configure the chains properly, you can use the `grails.plugin.springsecurity.filterChain.chainMap` property:

Listing 1. `grails.plugin.springsecurity.filterChain.chainMap`

```
grails.plugin.springsecurity.filterChain.chainMap = [
    '/api/**': 'JOINED_FILTERS,-exceptionTranslationFilter,-
authenticationProcessingFilter,-securityContextPersistenceFilter,-
rememberMeAuthenticationFilter',  // Stateless chain
    '/**': 'JOINED_FILTERS,-restTokenValidationFilter,-
restExceptionTranslationFilter'
// Traditional chain
]
```

To understand this syntax, please read the Spring Security Core documentation. Long story short: `JOINED_FILTERS` refers to all the configured filters. The minus (-) notation means all the previous values but the neglected one.

So the first chain applies all the filters except the stateful ones. The second one applies all the filters but the stateless ones.

Make sure that the stateless chain applies not only to your REST controllers, but also to the URL's where this plugin filters are listening: by default, `/api/login` for authentication, `/api/logout` for logout and `/api/validate` for token validation.

The difference is that, in a traditional form-based authentication, Spring Security will respond with an HTTP 302 redirect to the login controller. That doesn't work for an API, so in the stateless approach, an HTTP 401 response will be sent back.