Table of Contents

The token validation filter looks for the token in the request and then tries to validate it using the configured token storage implementation.

If the validation is successful, the principal object is stored in the security context. This allows you to use in your application `@Secured`, `springSecurityService.principal` and so on.

`springSecurityService.currentUser` expects a `grails.plugin.springsecurity.userdetails.GrailsUser` to perform a DB query. However, this plugins stores in the security context just a principal `Object`, because it does not assume you are using domain classes to store the users. Use `springSecurityService.principal` instead.

This plugin supports RFC 6750 Bearer Token specification out-of-the-box.

Sending tokens in the request

The token can be sent in the `Authorization` request header:

Listing 1. Accessing a protected resource using `Authorization` request header

```
GET /protectedResource HTTP/1.1
Host: server.example.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJleHAiOjE0MjI5OTU5MjIsInN1YiI6ImppbWkiLCJyb2xlcyI6WyJST0xFX
0FETUlOIiwiUk9MRV9VU0VSIl0sImlhdCI6MTQyMjk5MjMyMn0.rA7A2Gwt14LaYMpxNRtrCdO24RGrfHt
ZXY9fIjV8x8o
```

Or using form-encoded body parameters:

Listing 2. Accessing a protected resource using body parameters

```
POST /protectedResource HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

access_token=eyJhbGciOiJIUzI1NiJ9.eyJleHAiOjE0MjI5OTU5MjIsInN1YiI6ImppbWkiLCJyb2xl
cyI6WyJST0xFX0FETUlOIiwiUk9MRV9VU0VSIl0sImlhdCI6MTQyMjk5MjMyMn0.rA7A2Gwt14LaYMpxNR
trCdO24RGrfHtZXY9fIjV8x8o
```

If you need to use the `GET` HTTP method (to render images in an `img` tag, for example), you can also send the access token in a query string parameter named `access_token`:

If you disable the bearer token support, you can customise it further:

```
grails.plugin.springsecurity.rest.token.validation.useBearerToken = false
grails.plugin.springsecurity.rest.token.validation.headerName = 'X-Auth-Token'
```

If you still want to have full access and read the token from a different part of the request, you can implement a TokenReader and register it in your `resources.groovy` as `tokenReader`.

You must disable bearer token support to register your own `tokenReader` implementation.

Anonymous access

If you want to enable anonymous access to URL's where this plugin's filters are applied, you need to:

1. Configure `enableAnonymousAccess = true` (see table below).
2. Make sure that the `anonymousAuthenticationFilter` is applied before `restTokenValidationFilter`. See how to configure filters for more details.

For example, with this configuration:

Sample configuration to allow anonymous access

```
grails {
    plugin {
        springsecurity {
            filterChain {
                chainMap = [
                    '/api/guest/**':
'anonymousAuthenticationFilter,restTokenValidationFilter,restExceptionTranslationF
ilter,filterInvocationInterceptor',
                    '/api/**': 'JOINED_FILTERS,-anonymousAuthenticationFilter,-
exceptionTranslationFilter,-authenticationProcessingFilter,-
securityContextPersistenceFilter',
                    '/**': 'JOINED_FILTERS,-restTokenValidationFilter,-
restExceptionTranslationFilter'
                ]
            }

            //Other Spring Security settings
            //...

            rest {
                token {
                    validation {
                        enableAnonymousAccess = true
                    }
                }
            }
        }
    }
}
```

The following chains are configured:

1. `/api/guest/**` is a stateless chain that allows anonymous access when no token is sent. If however a token is on the request, it will be validated.
2. `/api/**` is a stateless chain that doesn't allow anonymous access. Thus, the token will always be required, and if missing, a Bad Request reponse will be sent back to the client.
3. `/**` (read: everything else) is a traditional stateful chain.

Validation Endpoint

There is also an endpoint available that you can call in case you want to know if a given token is valid. It looks for the token in a HTTP header as well, and if the token is still valid, it renders guide:authentication[its JSON representation]. If the token does not exist, it will render a `grails.plugin.springsecurity.rest.login.failureStatusCode` response (`401` by default).

The relevant configuration properties for the validation endpoint are:

Table 1. Validation endpoint configuration options

Config key	Default value
`grails.plugin.springsecurity.rest.token.validation.active`	`true`
`grails.plugin.springsecurity.rest.token.validation.headerName`	`X-Auth-Token`
`grails.plugin.springsecurity.rest.token.validation.endpointUrl`	`/api/validate`

Note that `headerName` is only considered if `grails.plugin.springsecurity.rest.token.validation.useBearerToken` is set to `false`. Otherwise (the default approach), as per RFC 6750, the header name will be `Authorization` and the value will be `Bearer TOKEN_VALUE`.