Spring Security Shiro Plugin - Reference Documentation

Burt Beckwith

Version 3.1.2.BUILD-SNAPSHOT

Table of Contents

Chapter 1. Introduction to the Spring Security Shiro Plugin

The Spring Security Shiro plugin adds some support for using a hybrid approach combining Spring Security and Shiro. It currently only supports Shiro ACLs, since Spring Security ACLs are very powerful but can be very cumbersome to use, and the Shiro approach is straightforward and simple.

The majority of the authentication and authorization work is still done by Spring Security. This plugin listens for Spring Security authentication events and uses the Spring Security `Authentication` instance to build and register a Shiro `Subject` instance. It also removes the Shiro credentials when you explicitly logout.

1.1. History

- Version 3.2
  - Added optional role-based permissions capability
- Version 3.1.1
  - released April 19, 2018
  - Update Spring Security Shiro version to 1.4.0
  - Added a configuration to disable shiroAttributeSourceAdvisor
- Version 3.1.0
  - released April 19, 2018
  - Support for Grails 3.3.0
  - Updated to Spring Security Core v3.2.1
- Version 3.0.1
  - released April 19, 2018
  - Updated Shiro version to 1.4.0
- Version 3.0.0
  - released December 8, 2015
- Version 1.0.0
  - released December 7, 2015
- Version 1.0-RC1
  - released October 05, 2013
- Version 0.1
  - released January 06, 2013

Chapter 2. Usage

To use the plugin, register a dependency by adding it to the `dependencies` block in build.gradle:

```
dependencies {
   ...
   compile 'org.grails.plugins:spring-security-shiro:3.1.2.BUILD-SNAPSHOT'
   ...
}
```

and run the `compile` command to resolve the dependencies:

```
$ grails compile
```

This will transitively install the Spring Security Core plugin, so you'll need to configure that by running the `s2-quickstart` script.

2.1. User Permissions

To use the Shiro annotations and methods you need a way to associate roles and permissions with users. The Spring Security Core plugin already handles the role part for you, so you must configure permissions for this plugin. There is no script to create a domain class, but it's a very simple class and easy to create yourself. It can have any name and be in any package, but otherwise the structure must look like this:

```
package com.mycompany.myapp

class Permission {

   User user
   String permission

   static constraints = {
      permission unique: 'user'
   }
}
```

Register the class name along with the other Spring Security attributes in `application.groovy` (or `application.yml`) using the `grails.plugin.springsecurity.shiro.permissionDomainClassName` property, e.g.

```
grails.plugin.springsecurity.shiro.permissionDomainClassName =
     'com.mycompany.myapp.Permission'
```

You can add other properties and methods, but the plugin expects that there is a one-to-many between your user and permission classes, that the user property name is "user" (regardless of the actual class name), and the permission property name is "permission".

If you need more flexibility, or perhaps to create this as a many-to-many, you can replace the Spring bean that looks up permissions. Create a class in src/main/groovy that implements the `grails.plugin.springsecurity.shiro.ShiroPermissionResolver` interface, and define the `Set<String> resolvePermissions(String username)` method any way you like. Register your bean as the `shiroPermissionResolver` bean in `resources.groovy`, for example

```
import com.mycompany.myapp.MyShiroPermissionResolver

beans = {
   shiroPermissionResolver(MyShiroPermissionResolver)
}
```

2.2. Role Permissions

Shiro can use both user-based and role-based permissions. The Spring Security Shiro plugin includes optional support for this capability. If role-based permissions are enabled, the effective permissions for a user becomes the union of both the user-based permissions and the role-based permissions for the user's roles, and all calls that expect permissions will use the union of the two. The Spring Security Core plugin handles the roles for you but you must configure permissions for the role.

As with the user-based permissions, there is no script to create a domain class, but it is simple to create one. It can also have any name and be in any package, but otherwise the structure must look like this:

```
package com.mycompany.myapp

class RolePermission {
    Role role
    String permission

    RolePermission(Role role, String permission) {
        this.role = role
    this.permission = permission
    }

    static constraints = {
        permission unique: 'role'
    }
}
```

Register the class name along with the other Spring Security attributes in `application.groovy` (or `application.yml`) using the `grails.plugin.springsecurity.shiro.rolePermissionDomainClassName` property, e.g.

```
grails.plugin.springsecurity.shiro.rolePermissionDomainClassName =
'com.mycompany.myapp.RolePermission'
```

You can add other properties and methods, but the plugin expects that there is a one-to-many between your role and permission classes, that the role property name is “role” (regardless of the actual class name), and the permission property name is “permission”.

If you need more flexibility, you can replace the Spring bean that looks up role-based permissions. Create a class in src/main/groovy that implements the `import org.apache.shiro.authz.permission.RolePermissionResolver` interface, and define the `Collection<Permission> resolvePermissionsInRole(String roleString)` method any way you like. Note that the collection of permissions returned are of type `org.apache.shiro.authz.Permission`, not the Permission you have defined in your application. You can use the `grails.plugin.springsecurity.shiro.GormShiroRolePermissionResolver` as an example for your own implementation.

Register your bean as the `shiroRolePermissionResolver` bean in `resources.groovy`, for example

```
import com.mycompany.myapp.MyShiroRolePermissionResolver

beans = {
   shiroRolePermissionResolver(MyShiroRolePermissionResolver)
}
```

2.3. Annotated service methods

Currently only Grails services and other Spring beans can be annotated, so this feature isn't available in controllers. You can use any of RequiresAuthentication, RequiresGuest, RequiresPermissions, RequiresRoles, and RequiresUser. See the Shiro documentation and Javadoc for the annotation syntax.

2.4. Using Shiro directly

You should use the annotations to keep from cluttering your code with explicit security checks, but the standard `Subject` methods will work:

```
import org.apache.shiro.SecurityUtils
import org.apache.shiro.subject.Subject

...

Subject subject = SecurityUtils.getSubject()

subject.checkPermission('printer:print:lp7200')

subject.isPermitted('printer:print:lp7200')

subject.checkRole('ROLE_ADMIN')

subject.hasRole('ROLE_ADMIN')

subject.isAuthenticated()

... etc
```

Chapter 3. Configuration

There are a few configuration options for the Shiro integration.

All of these property overrides must be specified in `grails-app/conf/application.groovy` (or `application.yml`) using the `grails.plugin.springsecurity` suffix, for example

```
grails.plugin.springsecurity.shiro.permissionDomainClassName =
     'com.mycompany.myapp.Permission'
```

Name	Default	Meaning
shiro.active	`true`	if `false` the plugin is disabled
shiro.permissionDomainClassName	*none*, must be set	the full class name of the permission domain class
shiro.rolePermissionDomainClassName	*none*	if set, the full class name of the role permissions domain class
shiro.useCache	`true`	whether to cache permission lookup; if you disable this they will be loaded from the database for every request
shiro.inspectShiroAnnotations	`true`	Whether to enable/disable shiroAttributeSourceAdvisor