Table of contents

Shiro support for the Spring Security plugin - Reference Documentation

Authors: Burt Beckwith

Version: 0.1

Table of Contents

1 Introduction

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 0.1
    - released January 06, 2013

2 Usage

The first step is to add a dependency for the plugin in `BuildConfig.groovy`:

```
plugins {
    …
    compile ':spring-security-shiro:0.1'
}
```

This should pull in the `spring-security-core` plugin transitively but you can also explicitly add in a dependency for that, in particular if you want to use a newer version than the plugin requires:

```
plugins {
    …
    compile ':spring-security-core:1.2.7.3'
}
```

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 `Config.groovy` using the `grails.plugins.springsecurity.shiro.permissionDomainClassName` property, e.g.

```
grails.plugins.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/groovy or src/java 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

```groovy
import com.mycompany.myapp.MyShiroPermissionResolver

beans = {
    shiroPermissionResolver(MyShiroPermissionResolver)
}
```

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 `org.apache.shiro.authz.annotation.RequiresAuthentication`, `org.apache.shiro.authz.annotation.RequiresGuest`, `org.apache.shiro.authz.annotation.RequiresPermissions`, `org.apache.shiro.authz.annotation.RequiresRoles`, and `org.apache.shiro.authz.annotation.RequiresUser`. See the Shiro documentation and Javadoc for the annotation syntax.

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:

```groovy
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
```

3 Configuration

There are a few configuration options for the Shiro integration.

> ⚠️ All of these property overrides must be specified in `grails-app/conf/Config.groovy` using the `grails.plugins.springsecurity` suffix, for example
>
> ```
> grails.plugins.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.useCache	true	whether to cache permission lookups; if you disable this they will be loaded from the database for every request