Securing your Application using Spring Security Core Plugin

IS2150

Assoc Prof Kaushik Dutta
Department of Information Systems
School Computing
National University of Singapore

In this tutorial we will learn the basics of Spring Security Core Plugin (Version:2.0-RC2). Reference documentation for this plugin can be found on the following link http://grails-plugins.github.io/grails-spring-security-core/docs/manual/guide/single.html#tutorials

1. First create your Grails application (you may use the Employee Manager Application for this Tutorial)

2. Install the plugin by adding it to BuildConfig.groovy

```
plugins {
   …
   compile ':spring-security-core:2.0-RC2'
}
```

Note that until the 2.0 version of the plugin is officially released, you'll also need to add a custom repository:

```
repositories {
   …
   mavenRepo 'http://repo.spring.io/milestone'
}
```

Run the compile script to resolve the dependencies and ensure everything is correct:
You may compile by Right Click-> Compile on Netbeans or using the following command

```
$ grails compile
```

3. Create the User and Role domain classes using the following command (make sure you are in the directory of your application before executing the following command)

```
$ grails s2-quickstart com.testapp User Role
```

Here com.testapp is the name of the package.

You can choose your names for your domain classes and package; these are just examples (i.e. User & Role)

The script creates this User class:

```
package com.testapp

package test

class User {

   transient springSecurityService

   String username
   String password
   boolean enabled = true
   boolean accountExpired
   boolean accountLocked
   boolean passwordExpired

   static constraints = {
      username blank: false, unique: true
      password blank: false
   }

   static mapping = {
      password column: '`password`'
   }

   Set<Role> getAuthorities() {
      UserRole.findAllByUser(this).collect { it.role } as Set
   }

   def beforeInsert() {
      encodePassword()
   }

   def beforeUpdate() {
      if (isDirty('password')) {
         encodePassword()
      }
   }

   protected void encodePassword() {
      password = springSecurityService.encodePassword(password)
   }
}
```

and this Role class:

```
package com.testapp

class Role {

   String authority

   static mapping = {
      cache true
   }

   static constraints = {
      authority blank: false, unique: true
   }
}
```

and a domain class that maps the many-to-many join class, UserRole:

```
import org.apache.commons.lang.builder.HashCodeBuilder

class UserRole implements Serializable {

   private static final long serialVersionUID = 1

   User user

   Role role

   boolean equals(other) {

      if (!(other instanceof UserRole)) {

         return false

      }

      other.user?.id == user?.id &&

         other.role?.id == role?.id

   }

   int hashCode() {

      def builder = new HashCodeBuilder()

      if (user) builder.append(user.id)

      if (role) builder.append(role.id)

      builder.toHashCode()

   }
```

```
static UserRole get(long userId, long roleId) {

      UserRole.where {

         user == User.load(userId) &&

         role == Role.load(roleId)

      }.get()

   }

   static UserRole create(User user, Role role, boolean flush
= false) {

      new UserRole(user: user, role: role).save(flush: flush,
insert: true)

   }

   static boolean remove(User u, Role r, boolean flush =
false) {

      int rowCount = UserRole.where {

         user == User.load(u.id) &&

         role == Role.load(r.id)

      }.deleteAll()

      rowCount > 0

   }
```

```
    static void removeAll(User u) {
        UserRole.where {
            user == User.load(u.id)
        }.deleteAll()
    }
    static void removeAll(Role r) {
        UserRole.where {
            role == Role.load(r.id)
        }.deleteAll()
    }
    static mapping = {
        id composite: ['role', 'user']
        version false
    }
```

The script has edited grails-app/conf/Config.groovy and added the configuration for your domain classes. Make sure that the changes are correct

Note: These generated files are not part of the plugin - these are your application files. They are examples to get you started, so you can edit them as you please. They contain the minimum needed for the plugin.

4. The plugin has no support for CRUD actions and GSPs for your domain classes; the spring-security-ui plugin supplies a UI for those.

   Install the “spring-security-ui” plugin by adding it to BuildConfig.groovy
   plugins {
     …
     compile ':spring-security-ui:1.0-RC1'
   }

5. Create a controller that will be restricted by role.

```
$ grails create-controller com.testapp.Secure
```

This command creates `grails-app/controllers/com/testapp/SecureController.groovy`. Add some output so you can verify that things are working:

```
package com.testapp

import grails.plugin.springsecurity.annotation.Secured

class SecureController {

   @Secured(['ROLE_USER'])

   def index() {

      render 'Secure access only'

   }

}
```

6. Run the application. Before we secure the page, navigate to http://localhost:8080/EmployeeManager/secure to verify that you can see the page without being logged in.
7. Shut down the app (using CTRL-C) and edit grails-app/conf/BootStrap.groovy to add the security objects that you need

```
import com.testapp.Role
import com.testapp.User
import com.testapp.UserRole

class BootStrap {

   def init = { servletContext ->

      def adminRole = new Role(authority: 'ROLE_ADMIN').save(flush: true)

      def userRole = new Role(authority: 'ROLE_USER').save(flush: true)

      def testUser = new User(username: 'me', password: 'password')

      testUser.save(flush: true)

      UserRole.create testUser, adminRole, true

      assert User.count() == 1

      assert Role.count() == 2

      assert UserRole.count() == 1

   }}
```

Some things to note about the preceding BootStrap.groovy:

- The example does not use a traditional GORM many-to-many mapping for the User<->Role relationship; instead you are mapping the join table with the UserRole class. This performance optimization helps significantly when many users have one or more common roles.
- We explicitly flushed the creates because BootStrap does not run in a transaction or OpenSessionInView.

8. Edit grails-app/controllers/SecureController.groovy to import the annotation class and apply the annotation to restrict access.

```
package com.testapp

import grails.plugin.springsecurity.annotation.Secured

class SecureController {

   @Secured(['ROLE_ADMIN'])

   def index() {

      render 'Secure access only'

   }

}
```

OR

```
package com.testapp

import grails.plugin.springsecurity.annotation.Secured

@Secured(['ROLE_ADMIN'])

class SecureController {

   def index() {

      render 'Secure access only'

   }

}
```

You can annotate the entire controller or individual actions. In this case you have only one action, so you can do either.

Then override as suggested:

```
grails s2ui-override auth
grails s2ui-override layout
grails s2ui-override user com.testapp
grails s2ui-override role com.testapp
```

Finally added the secured annotations to your controllers, i.e.:

```
package com.testapp
import grails.plugin.springsecurity.annotation.Secured

@Secured(['ROLE_ADMIN'])
class RoleController extends grails.plugin.springsecurity.ui.RoleController {
}

================================
package com.testapp
import grails.plugin.springsecurity.annotation.Secured
@Secured(['ROLE_USER'])
class UserController extends grails.plugin.springsecurity.ui.UserController
{
}
```

Add the following to Config.groovy by replacing default code rules

```
grails.plugin.springsecurity.rejectIfNoRule = false

grails.plugin.springsecurity.fii.rejectPublicInvocations = false

grails.plugin.springsecurity.securityConfigType = 'InterceptUrlMap'

grails.plugin.springsecurity.interceptUrlMap = [
    '/':                              ['permitAll'],
    '/index':                         ['permitAll'],
    '/index.gsp':                     ['permitAll'],
    '/**/js/**':                      ['permitAll'],
    '/**/css/**':                     ['permitAll'],
    '/**/images/**':                  ['permitAll'],
    '/**/favicon.ico':                ['permitAll'],
    '/login/**':                      ['permitAll'],
    '/logout/**':                     ['permitAll']
]
```

9. **Run grails run-app again and navigate to http://localhost:8080/EmployeeManager/secure.**
   This time, you should be presented with the login page. Log in with the username and password you used for the test user, and you should again be able to see the secure page.

10. **Test the Remember Me functionality.**
    Check the checkbox, and once you've tested the secure page, close your browser and reopen it. Navigate again the the secure page. Because a is cookie stored, you should not need to log in again. Logout at any time by navigating to http://localhost:8080/bookstore/logout.

References:

http://grails.org/How+to+install+Spring+Security+plugin+in+a+new+Grails+project

http://stackoverflow.com/questions/20156277/grails-spring-security-ui-user-and-role-management-access

http://stackoverflow.com/questions/20696211/access-denied-and-getting-sorry-youre-not-authorized-to-view-this-page-after/20716079#20716079?newreg=0af4fc1977b24c069d2cb5184997dde9

http://ajibrans.wordpress.com/2012/02/04/spring-security-plugin-with-grails-1-3-7/

http://grails.1312388.n4.nabble.com/spring-security-2-0-RC2-annotations-td4652662.html

http://burtbeckwith.com/blog/?p=2003