Grails Application Development

Part 8 - Security

Ardhika

Objectives

- To learn and build a custom security components for the Grails application

Session Plan

- Built in Grails Security
- Securing Views - Fields / Sections of Screen
- Securing Controllers
- Securing the application
- Creating a CODEC
- Security Via Plugins

Built in Security

- DB access through GORM - all SQL escaped - prevents injection attacks
- Scaffolding escapes all fields in HTML
- Link creating tags produce all escaped HTML - prevents code injection
  - encodeAsURL()
- Easy codec implementation to encode text strings, password etc.

Securing Views

- When you want to secure a field or set of fields
- You can check the user and role using grails <g:if>
  - Surround the field(s) to be protected with <g:if>
- Alternatively you could create a special tag to do that
  - Spring Security Plugin does that based on user-role
- For custom intentions
  - Owner can remove somebody’s membership or discussion or message
  - Only members can post a discussion or reply
- Send flags from Controllers - canPost, canDeleteReply etc.
- Or Create a well named readable tag that suits your application

Ardhika

Securing views - Method #1

- Only the owner can edit or delete the Circle
- First let us create a utility method in User domain class

```
 boolean sameAs(User other) {
     this.id == other.id
 }
```

- In Circle controller's show action change the last line

```
[circleInstance: circleInstance]
```

- to

```
def isOwner = circleInstance.owner.sameAs(session.loggedInUser)
[circleInstance: circleInstance, isOwner:isOwner]
```

- Now isOwner will be set to true if the logged in user is owner of the circle

Securing views - Method #1

- Now go to the circle/show.gsp
- You will see the form at the bottom which holds the buttons(links) for Edit & Delete
- Surround the form with a <g:if>

```
<g:if test="${isOwner}">
   <g:form>
      <fieldset class="buttons"> ...
      </fieldset>
   </g:form>
</g:if>
```

- Now go to All circles menu and check this out

Securing views - Method #2

- How can we do the same with the tags?
- We are going to need that sameAs() in User domain
- Change the CircleController's show action last line to

```
[circleInstance: circleInstance]
```

- Create a Tag in LearnTagLib.groovy

```
 def isLoggedInUserOwnerFor = { attrs, body ->
    def circle = attrs.circle
    if(circle.owner.sameAs(session.loggedInUser))
       out << body()
 }
```

- Now in circle/show.gsp change the <g:if> tag to

```
<g:isLoggedInUserOwnerFor circle="${circleInstance}">
```

- Check this out!

Securing Controllers

- Check the myCircles() action of the CircleController
- It just accesses the user object stored in session
- What will happen if you invoke circles/myCircles without logging in first?
- Present code will throw exceptions & fail

```
def myCircles(){
 def userId=session.loggedInUser.id
 def user=User.get(userId)
 def circles=user.getCircles()
 render (view:"circles", model:[circleInstanceList: circles,
                              circleInstanceTotal: circles.count])
}
```

Securing Controllers

- If the user has not logged in we should put the user to login

```
def myCircles(){
 def loggedInUser = session.loggedInUser
 if(!loggedInUser) {
    redirect(controller:"user", action:"login")
 }
 else {
    def userId=loggedInUser.id
    def user=User.get(userId)
    def circles=user.getCircles()
    render (view:"circles",
               model:[circleInstanceList: circles,
                      circleInstanceTotal: circles.count])
 }
}
```

- Doing the same for all actions in all controllers? Tedious!

Securing Controllers - Using interceptors

- Instead securing actions - Secure Controllers
- Intercept every request that gets into the actions
- Designate a function to execute while intercepting
- You can intercept before or after a request

```
//set before interceptor to exceute checkAuth function
def beforeInterceptor = [action:this.&checkAuth]
//Define checkAuth function
def checkAuth() {
 if(!session.loggedInUser) {
    redirect(controller:"user", action:"login")
    return false
 }
 return true
}
```

- Put this in CircleController

Securing Controllers - Using interceptors

- We need to do this in every controller
- If we do this in the user controller this will intercept requests for login, authenticate, create(register) and save(register)
- Obviously you don't have to login to register yourself
- Fortunately you can have an exclusion list
- In UserController you can write

```
def beforeInterceptor = [action:this.&checkAuth,
                   except:['login', 'logout','create', 'save']]
```

- But there is a method better than this!
- Define filters at the app level

Securing at app level - Filters

- Create a new -> Filter with name LearnSecurity or (grails command create-filters)
- This creates LearnSecurityFilters class conf folder

```
class LearnSecurityFilters {

    def filters = {
        all(controller:'*', action:'*') {
            before = {

            }
            after = { Map model ->

            }
            afterView = { Exception e ->

            }
        }
    }
}
```

Ardhika

Securing at app level - Filters

- before
  - code executes before action
- after
  - code executes after action but before view rendering
  - Do something to the model if needed
- afterView
  - code executes after rendering the view
  - Can handle exceptions if there are any

Implementing a Filter

- There is a default filter for all controllers & actions
- Let us add code in the before block
- But exclude check for User controllers actions for login and register

```
before = {
  if(!session.loggedInUser &&
      !((controllerName == 'user') &&
       (actionName == 'login' || actionName == 'authenticate' ||
        actionName == 'create' || actionName == 'save'))) {
            redirect(controller:"user", action:"login")
            return false
  }
}
```

Implementing a Filter

- Filters syntax helps you to make it neat without those clumsy conditions
- Refer Grails Reference for Filters
- All controllers and actions except the following

```
allExceptLoginAndRegister(controller:'user',
              action:'(login|authenticate|create|save)',
                 invert:true) {
 before = {
    if(!session.loggedInUser) {
       redirect(controller:"user", action:"login")
       return false
    }
 }
}
```

Codecs

- Codec as used to transform a string and doing the reverse
  - encodeAsHTML() & decodeHTML()
- There are variety of in built codecs in Grails
- Play a major part in security
- can build a custom codec for password hashing/encryption

- Let us try a technique - reverse and store password
- Codec is a groovy class with name ending in Codec stored in the utils folder

Create a Codec

- ReversedPasswordCodec.groovy in grails-app/utils

```
class ReversedPasswordCodec {
 static encode = { str->
    return str.reverse()
 }

 static decode = { str->
    return str.reverse()
 }
}
```

- Now while saving the user encode the password

```
userInstance.password.encodeAsReversedPassword()
```

- Can be done in domain class beforeInsert also
- Similarly encode the password in the authenticate action

Security Plugins

- Can be installed using install-plugin grails command
- Plugins are like mini projects with a set of components that we can reuse
- Spring Security Plugin
  - Comes with User, Role userRole domain classes
  - Login and logout actions and views
  - Tag library for checking the roles (securing portions of views)

Thank You!

Ardhika

Bala Sundarasamy
bala@ardhika.com