Security

At the the end of this session

You will be familiar with

- Common Spring Security Terms
- Securing controller actions
- Securing GSP sections
- Accessing security access within the service layer

Security Basics

Security is broken down into two phases

- Authentication
  - Are you allowed access to secured areas of the application?
- Authorization
  - What level of access do you have (User, Admin?)

Spring security uses Users to grant Authentication and Roles (sometimes referred to as Authorities) to grant levels of authorization.

Spring Security

Spring Security is the de-facto plugin for securing your Grails application. It uses the core spring security libraries and wraps it into the grails framework along with providing an extensible security plugin interface to connect to 3rd party authentication platforms such as Google, Twitter, Facebook and LDAP.

http://grails.org/plugin/spring-security-core

http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/

Components

- Filter Chain
- Authentication Token
- Exception Filter
- Authentication Manager
- User Details Service
- Security Context Holder
- Annotations
- Helpers: Service, Util and Tag `<sec:...\>`

Components

Filter Chain

Within the filter chain are all filters need to to create various tokens (Anonymous, x509, Basic Auth, Facebook, Twitter, Impersonation, etc…)

http://grails-plugins.github.io/grails-spring-security-core/guide/filters.html

http://docs.spring.io/spring-security/site/docs/3.0.x/reference/security-filter-chain.html

Components

Authentication Token

When a filter in the chain is able to gather enough information about the end user, it will create an authentication token that will be passed into the authentication provider for authentication.

Components

Security Interceptor Filter

This catches security exceptions and routes them to the appropriate handlers. Things such as:

Authentication Required

Access Denied

Password Expired /Account Locked

etc…

http://docs.spring.io/spring-security/site/docs/3.0.x/reference/core-web-filters.html

Components

Authentication Manager

This registers all authentication providers configured for the applications and will authenticate tokens matching their respective providers. For example, a `TwitterAuthToken` will be authenticated using the `TwitterAuthProvider`.

http://grails-plugins.github.io/grails-spring-security-core/guide/authenticationProviders.html

Components

User Details Service

This takes a fully authenticated `Authetication` token and fills in user details, typically from the local database with information such as full name, account status and role or `Authority` access.

Components

Security Context Holder

Once a fully authenticated token has been built containing the UserDetails for the authenticated user, it is stored in the SecuirityContextHolder (SCH).

http://grails-plugins.github.io/grails-spring-security-core/guide/authenticationProviders.html

User Details Service

This takes a fully authenticated `Authetication` token and fills in user details, typically from the local database with information such as full name, account status and role or `Authority` access.

Components

Annotations

Controllers and controller actions are typically secured using the @Secured annotation. For more complex role groups, you can use the @Authorities annotation. Alternatively you can use a simple Map in your config or a DB bases `Reqestmap` approach.

http://burtbeckwith.com/blog/?p=1398

http://grails-plugins.github.io/grails-spring-security-core/guide/authenticationProviders.html

Components

Helpers

There are some helpers available within your grails application to provide you high level access to the internals of the security framework. This includes the `SpringSecurityService`, the `SpringSecuirityUtils` and the `<sec:.../>` tag library.

Workshop 1

Add dependencies to `BuildConfig.groovy`

```
plugins {
    compile ":spring-security-core:2.0-RC4"
}
```

Install the new dependency

```
grails package
```

Create the domain classes

You will need users, roles and a many to many relationship between them. You can use the `s2-quckstart` command to do this:

```
grails s2-quickstart com.opi User Role
```

This will create user and role domain classes along with the many-to-many domain. It will also configure your application so everything is **protected by default** except the index page and static assets.

User Registration and Management

You can use spring-security-ui to add user management and registration to your application. Configure dependencies in `BuildConfig.groovy`

```
plugins {
    compile ":spring-security-ui:1.0-RC2"
    compile ":mail:1.0.6"
    compile ":famfamfam:1.0.1"
    runtime ":jquery:1.11.1"
    compile ":jquery-ui:1.10.3"
}
```

Adding User Registration

You'll need to override the default login form and user registration. You can find the default versions inside the plugin's source code. just copy them to the respective location in your project and update them to your needs.

```
grails-app/views/login/auth.gsp
grails-app/views/register/index.gsp
```

Optional: Greenmail

If you don't want to configure an email account for registration, you can use the Greenmail plugin for development. Then browse to **/greenmail**

http://grails.org/plugin/greenmail

In `BuildConfig.groovy` "plugins" section

```
compile ":greenmail:1.3.4"
```

Greenmail

In `Config.groovy`

```
environments {
    development {
        grails.mail.port =
            com.icegreen.greenmail.util.ServerSetupTest.SMTP.port
    }
}
```

Securing Controller and Actions

Of of the most common tasks of Spring Security is securing a controller action. To allow access to any authorized user. This is done via the annotation `@Secured` for example:

```
@Secured(['ROLE_USER'])
controller QuestionController {
    @Secured(['ROLE_ADMIN'])
    def delete() {
...
```

Showing or Hiding text

You often need to control the rendering of content based on role within a GSP. This can be done via the <sec:/> tag library. For example:

```
<sec:ifAnyGranted roles="ROLE_ADMIN">
    Secret Content
</sec:ifAnyGranted>
```

SpringSecurityService

The SpringSecurityService is injected as a bean named `springSecurityService` wherever you need it within your application.

- Access current authentication or authorization information
- Reauthenticate
- Clear caching information
- Encode passwords

Service Access to User

Passing authorities into a service method

```
List roles = springSecurityService
                .authentication
                .authorities
myService.withdrawMoney(roles)
```

Service Access to User

Passing username into a service method

```
String username = springSecurityService
                      .loadCurrentUser()
                      .username
myService.getFavorites(username)
```

SpringSecurityUtils

Static methods that provide access to spring security internals without the need for injecting the SpringSecurityService.

Get authorities, check role access, register custom filters or providers.

http://grails-plugins.github.io/grails-spring-security-core/guide/helperClasses.html

Resources

Spring Security

http://docs.spring.io/spring-security/site/docs/3.2.4.RELEASE/reference/htmlsingle/

https://github.com/spring-projects/spring-security

Grails Spring Security Plugin

http://grails.org/plugin/spring-security-core

http://grails-plugins.github.io/grails-spring-security-core/guide/

Burt Beckwith's Blog

http://burtbeckwith.com/blog/?p=1090 http://burtbeckwith.com/blog/?p=1398

Overview of Spring Security Components

http://prezi.com/mczbfzjyvczl/grails-spring-security-core/