Enterprise Systems

Lecture 17
Spring security (SS)

“Spring Security” used to be known as Acegi

a Spring-based **declarative** security framework for handling **authentication** and **authorisation**, at both the **web request level** and at the **method invocation level**

You can declare “who” invokes “what”!

SS uses dependency injection and AoP
Authentication, authorisation

**Authentication**: process of verifying that the users, also called **principal** (people, program, services) of our application are who they say they are. Successful authentication result in **authenticated principal**

For example when login or when prompted for password

**Authorization**: accessibility of the secured system. Two steps

1. Role assignment: mapping of authenticated
Authorization:

1. User Role assignment: mapping of authenticated principal to one or more roles (authorities)

For example, anonymous user, registers login and become customer

2. Assignment of roles to secured resources of the system. Payment panel should only be visible to Customers who log in
An overview of SS

**Security Interceptor**: intercept access to resources to enforce security AND applies security rules.

You don’t directly use Security Interceptor as it delegates the task to one of these:

- Authentication Manager
- Access Decision Manager
- Run-as Manager
- After-invocation Manager
An overview of SS

**Authentication Manager** determines who you are:
- *principal* for example a username/email)
- *Credentials* for example password).

**Access Decision Manager** decide if you are authorised to access a resource (after passing through Authentication Manager)
An overview of SS

After passing through the previous two managers **Run-as Manager** is used to manage your authentication and access while accessing the resources.

For example: you may be allowed to view (read access) but not modify the contents (write access)

After-invocation Manager enforces security after the access.

For example altering the returned values so that the user is only able to access certain properties of the returned object.
Extension of the previous example to include security

ehcache.xml  -log4j.properties

Shop-spring-hib.xml   //setting for security

- shop/
  |-  dao/ RoleDAO added
  |-  model/ Role…
  |-  usecases/ Modified
  |-  Main.java Modified

This is a simple example that you see SS, we will move to SS3
requires implementing a single method authenticate()

If successful returns an org.acegisecurity.Authentication object

If authentication fails an AuthenticationException will be thrown.

We don’t directly implement A/M, instead we use ProviderManager, an implementation of AuthenticationManager
<bean id="authenticationManager" class="org.acegisecurity.providers.ProviderManager">
    <property name="providers">
        <list>
            <ref local="daoAuthenticationProvider"/>
            <ref local="anonymousAuthenticationProvider"/>
        </list>
    </property>
</bean>
ProviderManger has a list of authentication providers through providers property- out of many, we use two (see code in previous slide)

• DaoAuthenticationProvider:
• AnonymousAuthenticationProvider: authenticates a user as an anonymous user.
1. daoAuthenticationProvider retrieves user info such as username and password from a database
2. daoAuthenticationProvider performs authentication by comparing the username/password retrieved from the database with the principal and credentials passed in an Authentication object from the authentication manager

**Outcome:** Authentication object or AuthenticationException.
so you must create DAO interfaces and their implementations

1. Observe DAO objects for Role
2. Look at lines 214-222
3. Look at line 212

Next we look at them in details.
encrypted passwords

Suppose that we want to store the password encrypted.

We (application) don’t want to know the users passwd, so user-provided password must also be encrypted.

There are various options:

• encoding.PlaintextPasswordEncoder
  No encryption (default)
• encoding.ShaPasswordEncoder
  Does Secure Hash Algorithm (SHA) on the password
• encoding.Md5PasswordEncoder
  Similarly
What is a salt?

Two salt sources available for encryption are:

- **SystemWideSaltSource**
  Provides the same salt for all users

- **ReflectionSaltSource**
  Encoded each user's data using a different salt value (better security)

Also to enable caching UserCache property must be declared.
Controlling access

After authentication *Access Decision Manager* decides if a user has required privileges to access secured resources.

```java
(org.acegisecurity
 .AccessDecisionManager)
```

**Behind the scene a voting mechanism is used:**

*AccessDecisionManager* uses an *AccessDecisionVoter* to decide, on the basis of user’s authorities and configuration required by the resource (Role assignments) to give access or to deny or to abstain vote.

See lines 275-291 for sample of Confs