Securing Portlets With Spring Security

John A. Lewis
Chief Software Architect
Unicon, Inc.

JA-SIG Spring 2008 Conference
28 April 2008

UNICON

Agenda

- JSR 168 Portlet Security
- Spring Security (aka “Acegi”)
- Spring Portlet Security
- Applying Portlet Security
- Resources
- Questions & Answers

JSR 168 Portlet Security

What does the spec give us to work with?

Portal Authentication

- The portal is completely responsible for authentication
  - This means we just use what it gives us – we don't redirect for authentication purpose
- The JSR 168 *PortletRequest* class provides two methods for getting user identity (the same ones as the Servlet spec)

```
String getRemoteUser()

Principal getUserPrincipal()
```

Portal Authorization

- Portals generally provide the ability to assign a set of "Roles" to the User
- The JSR 168 *PortletRequest* class provides a method for getting at these roles (the same ones as the Servlet spec)

```
boolean isUserInRole(String)
```

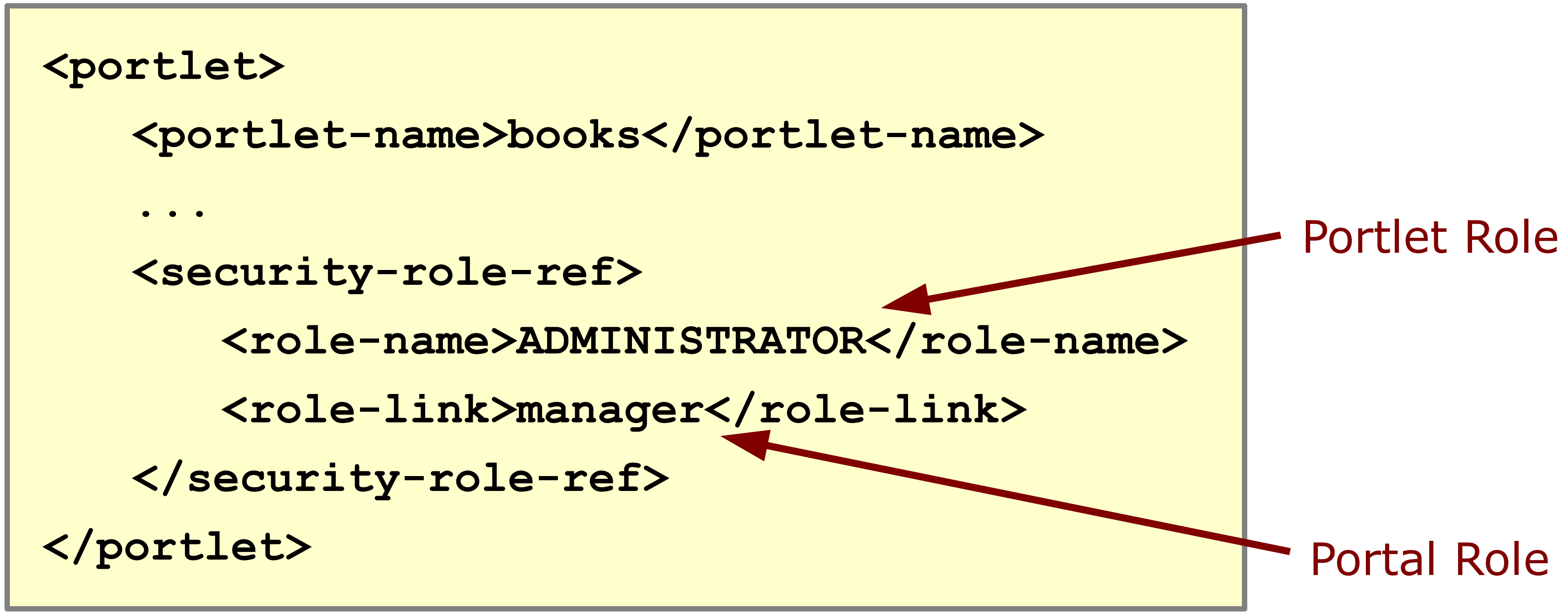
Declaring Portal Roles

- Same as declaring roles for Servlet container-based security
- Include all portal roles that may be used in *web.xml*:

```
...
<security-role>
   <role-name>manager</role-name>
</security-role>
...
```

Mapping Portal Roles To Portlet Roles

- In *portlet.xml*:



```
<portlet>
    <portlet-name>books</portlet-name>
    ...
    <security-role-ref>
        <role-name>ADMINISTRATOR</role-name>
        <role-link>manager</role-link>
    </security-role-ref>
</portlet>
```

Warning!

If you are storing your *SecurityContext* in the *PortletSession* with *APPLICATION_SCOPE* (more on this later), make sure these are the same in all your **`<portlet>`** declarations – the first one to be invoked on a page will determine the mapping for all portlets in your webapp.

Security Constraints

- Require a secure transport in *portlet.xml*:

```
<portlet-app>
    ...
    <portlet>
        <portlet-name>accountSummary</portlet-name>
        ...
    </portlet>
    ...
    <security-constraint>
         <display-name>Secure Portlets</display-name>
         <portlet-collection>
             <portlet-name>accountSummary</portlet-name>
        </portlet-collection>
        <user-data-constraint/>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
    </security-constraint>
    ...
</portlet-app>
```

Other Portlet Security Info

- *PortletRequest* has a couple other key security-related methods:

```
String getAuthType()
```

Returns name of authentication scheme used (BASIC_AUTH, CLIENT_CERT_AUTH, custom) or null if user is not authenticated.

```
boolean isSecure()
```

Returns true if the request was made over a secure channel (such as HTTPS)

Portlet User Attributes

- Can also use the **`USER_INFO`** Map available as a *PortletRequest* attribute.
- May contain arbitrary user information:
  - **`user.name.given`**
  - **`user.bdate`**
  - **`user.gender`**
  - etc.
- Some portals expose security-related information here, but this mechanism should be avoided if possible

Spring Security

a.k.a *Acegi Security*
A quick overview

What Is Spring Security?

- Powerful, flexible security framework for enterprise software
- Emphasis on applications using Spring
- Comprehensive authentication, authorization, and instance-based access control
- Avoids security code in your business logic – treats security as a cross-cutting concern
- Built-in support for a wide variety of authentication and integration standards

Spring Security Releases

- Acegi Security (the old name)
  - Current Version: 1.0.7
  - Initial GA Release: May 2006
  - Portlet support in Sandbox
- Spring Security (the new name)
  - Current Version: 2.0.0
  - Initial GA Release: April 2008
  - Portlet support Included
  - Changes packaging from `org.acegisecurity` to `org.springframework.security`

Applications Are Like Onions

- Spring Security can be applied at multiple layers in your application:
  - Apply security as markup is constructed in the Rendering Layer using the supplied JSP taglib
  - Restrict access to areas of web application in the Dispatch Layer based on URL pattern-matching
  - Secure method invocation on the Service Layer to ensure calls are from properly authorized user
  - Provide Access Control Lists (ACLs) for individual objects in the Domain Layer

Spring Portlet Security

Applying Spring Security to Portlets

Portlet Challenges

- Portlets have some key differences from Servlets:
  - No Filters
  - Can't treat URLs like Paths
  - Multiple Request Phases
- These create some challenges in applying the normal Spring Security patterns
- So we need some different infrastructure for wiring Spring Security into our portlet application

Six Main Portlet Security Beans

- PortletProcessingInterceptor
- AuthenticationManager
- AuthenticationDetailsSource
- AuthenticationProvider
- UserDetailsService
- PortletSessionContextIntegrationInterceptor

PortletProcessingInterceptor

- Interceptor that processes portlet requests for authentication by invoking the configured *AuthenticationManager*
- Creates the initial *AuthenticationToken* from the *PortletRequest* security methods

```
<bean id="portletProcessingInterceptor"
    class="org.springframework.security.ui.portlet.
        PortletProcessingInterceptor">
    <property name="authenticationManager"
        ref="authenticationManager" />
    <property name="authenticationDetailsSource"
        ref="portletAuthenticationDetailsSource" />
</bean>
```

Portlet equivalent of *AuthenticationProcessingFilter* used for traditional servlet web applications

AuthenticationManager

- Use normal provider-based *AuthenticationManager* bean
- Declared via special namespace schema:

```
<sec:authentication-manager
   alias="authenticationManager" />
```

Can use multiple providers if you are authenticating from Portlets and Servlets

AuthenticationDetailsSource

- Can be used to check isUserInRole(...) to get list of Portal Roles into the Authentication Request:

```
<bean name=”portletAuthenticationDetailsSource”
    class="org.springframework.security.ui.portlet.
    PortletPreAuthenticatedAuthenticationDetailsSource">
    <property name="mappableRolesRetriever">
        <bean class="org.springframework.security.
        authoritymapping.SimpleMappableAttributesRetriever">
            <property name="mappableAttributes">
                <list>
                    <value>ADMIN</value>
                </list>
            </property>
        </bean>
    </property>
</bean>
```

Only needed if we are using Portal Roles for our security decisions

AuthenticationProvider

- PreAuthenticatedAuthenticationProvider processes pre-authenticated authentication request (from *PortletProcessingInterceptor*)
- A valid *PreAuthenticatedAuthenticationToken* with non-null principal & credentials will succeed

```
<bean id="portletAuthenticationProvider"
    class="org.springframework.security.providers.preauth.
        PreAuthenticatedAuthenticationProvider">
    <sec:custom-authentication-provider />
    <property name="preAuthenticatedUserDetailsService"
        ref="preAuthenticatedUserDetailsService" />
</bean>
```

UserDetailsService

- Bean that knows how to populate user details (including *GrantedAuthorities*) for the authenticated user
  - *PreAuthenticatedGrantedAuthoritiesUserDetailsService* will use purely data contained in the *PreAuthenticatedAuthenticationToken*

```
<bean name="preAuthenticatedUserDetailsService"
    class="org.springframework.security.providers.preauth.
        PreAuthenticatedGrantedAuthoritiesUserDetailsService" />
```

Can also use any other *UserDetailsService* that can populate *UserDetails* by username, such as *JdbcUserDetailsManager* or *LdapUserDetailsManager*

PortletSessionContextIntegrationInterceptor

- Interceptor that retrieves/stores the contents of the *SecurityContextHolder* in the active *PortletSession*
- Without this, every request would trigger a full authentication cycle
- Default is to use **`APPLICATION_SCOPE`**

```
<bean id="portletSessionContextIntegrationInterceptor"
    class="org.springframework.security.context.
        PortletSessionContextIntegrationInterceptor" />
```

Portlet equivalent of *HttpSessionContextIntegrationFilter*, used for traditional servlet web applications

Using The Two Interceptors

- Add them to our Portlet's *HandlerMapping*:

```
<bean id="portletModeHandlerMapping"
    class="org.springframework.web.portlet.handler.
        PortletModeHandlerMapping">

    <property name="interceptors">
        <list>
            <ref bean="portletSessionContextIntegrationInterceptor"/>
            <ref bean="portletProcessingInterceptor"/>
        </list>
    </property>

    <property name="portletModeMap">
        <map>
            <entry key="view"><ref bean="viewController"/></entry>
            <entry key="edit"><ref bean="editController"/></entry>
            <entry key="help"><ref bean="helpController"/></entry>
        </map>
    </property>

</bean>
```

Warning! This ordering is critical – they will not work correctly if they are reversed!

Applying Portlet Security To The Rendering Layer

Customizing our markup
based on security information

Spring Security JSP TagLib

- Allows us to access authentication information and to check authorizations
- Useful for showing/hiding information or navigation controls based on security info

```
<%@ taglib prefix="sec"
    uri="http://www.springframework.org/security/tags" %>

<p>Username: <sec:authentication property="principal.username"/></p>

<sec:authorize ifAllGranted="ROLE_USER">
    <p>You are an authorized user of this system.</p>
</sec:authorize>

<sec:authorize ifAllGranted="ROLE_ADMINISTRATOR">
    <p>You are an administrator of this system.</p>
</sec:authorize>
```

Warning: Don't rely on this to restrict access to areas of the application. Just because navigation doesn't appear in the markup doesn't mean a clever hacker can't generate a GET/POST that will still get there.

Applying Portlet Security To The Dispatch Layer

Controlling where users can go in the application

Secure Portlet Request Dispatching

- Portlet Requests don't have a path structure, so we can't use the path-based patterns of *FilterSecurityInterceptor* to control access
- Something standard may be added in the future – perhaps a *ConfigAttributeDefinition* for various aspects of Portlet Requests that we can use as an *ObjectDefinitionSource*

Using a *HandlerInterceptor*

- Best practice in Spring 2.0 is to build a custom *HandlerInterceptor* for your Portlet
- Compare contents of *SecurityContextHolder. getContext(). getAuthentication()* with Portlet Mode, Window State, Render Parameters – whatever you want to use to determine permission
- Throw a *PortletSecurityException* if access is not permitted, otherwise allow processing to proceed

Using Annotations

- If using Spring 2.5 Annotation-based Dispatching, use Security Annotations as well
  - ApplicationContext entry:

```
<sec:global-method-security secured-annotations="enabled" />
```

  - Annotated method:

```
import org.springframework.security.annotation.Secured;
...
    @Secured({"ROLE_ADMIN"})
    @RequestMapping(params="action=view")
    public String deleteItems(RequestParam("item") int itemId) {
        ...
```

Applying Portlet Security To The Service Layer

Making sure Services are invoked by only by user with proper permissions

AccessDecisionManager

- Standard Spring Security bean for making decisions about access to resources

```
<bean id="accessDecisionManager"
    class="org.springframework.security.vote.
        AffirmativeBased">
    <property name="decisionVoters">
        <list>
            <bean class="org.springframework.security.
                vote.RoleVoter" />
            <bean class="org.springframework.security.
                vote.AuthenticatedVoter" />
        </list>
    </property>
</bean>
```

MethodSecurityInterceptor

```
<bean id="myService" class="sample.service.MyService">
   <sec:intercept-methods
      access-decision-manager-ref="accessDecisionManager">
      <sec:protect method="sample.service.MyService.*"
         access="IS_AUTHENTICATED_FULLY" />
      <sec:protect method="sample.service.MyService.add*"
         access="ROLE_ADMINISTRATOR" />
      <sec:protect method="sample.service.MyService.del*"
         access="ROLE_ADMINISTRATOR" />
      <sec:protect method="sample.service.MyService.save*"
         access="ROLE_ADMINISTRATOR" />
   </sec:intercept-methods>
</bean>
```

Applying Portlet Security To Servlets

Using the whole web/portlet application as one secure bundle

Bridging The Gap

- We can reuse the Portlet *SecurityContext* in getting resources from Servlets in the same web application
- Useful for securing:
  - AJAX Calls
  - Dynamic Images
  - PDF Reports
- Need to get Portlets and Servlets to share session data to do this

Portlets & Servlets Sharing Session

- Possible according to JSR 168 (PLT 15.4)
  - Must be in the same webapp
  - Portlet must use **`APPLICATION_SCOPE`**
- Sometime tricky in practice
  - Portlet requests go thru Portal webapp URL
  - Servlet requests go thru Portlet webapp URL
  - Session tracking via **`JSESSIONID`** Cookie usually uses URL path to webapp – not shared!

Tomcat 5.5.4 +

On **`<Connector>`** element set **`emptySessionPath=true`**

Apply Servlet Filter Chain

- In *web.xml*:

```
<filter>
    <filter-name>securityFilterChainProxy</filter-name>
    <filter-class>org.springframework.web.filter.
        DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping>
    <filter-name>securityFilterChainProxy</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

FilterChainProxy

- Since the portal handles authentication, you only need a few entries in this bean:

```
<bean id="servletSecurityFilterChainProxy"
    class="org.springframework.security.util.
        FilterChainProxy">
    <sec:filter-chain-map path-type="ant">
        <sec:filter-chain pattern="/**"
            filters="httpSessionContextIntegrationFilter,
                exceptionTranslationFilter,
                filterSecurityInterceptor" />
    </sec:filter-chain-map>
</bean>
```

HttpSessionContextIntegrationFilter

- If session sharing is working properly, it will populate the SecurityContextHolder using the same SecurityContext as the Portlet side

```
<bean id="httpSessionContextIntegrationFilter"
    class="org.springframework.security.context.
        HttpSessionContextIntegrationFilter" />
```

This will only work if *PortletSessionContextIntegrationInterceptor* is storing in the **`APPLICATION_SCOPE`** of the *PortletSession* (which is the default)

ExceptionTranslationFilter

- Since we are relying on the Portal for authentication, then an Exception means that authentication has already failed
- *PreAuthenticatedProcessingFilterEntryPoint* returns **SC_FORBIDDEN** (HTTP 403 error)

```
<bean id="exceptionTranslationFilter"
   class="org.springframework.security.ui.
      ExceptionTranslationFilter">
   <property name="authenticationEntryPoint">
      <bean class="org.springframework.security.ui.preauth.
         PreAuthenticatedProcessingFilterEntryPoint" />
   </property>
</bean>
```

FilterSecurityInterceptor

- Secure resource URLs accordingly
- Use the same *AuthenticationManager* and *AccessDecisionManager* as in the portlet

```
<bean id="filterSecurityInterceptor"
    class="org.springframework.security.intercept.web.
        FilterSecurityInterceptor">
    <property name="authenticationManager"
        ref="authenticationManager" />
    <property name="accessDecisionManager"
        ref="accessDecisionManager" />
    <property name="objectDefinitionSource">
        <sec:filter-invocation-definition-source>
            <sec:intercept-url pattern="/resources/**"
                access="IS_AUTHENTICATED_FULLY" />
        </sec:filter-invocation-definition-source>
    </property>
</bean>
```

Resources

Places to go to actually use this stuff!

Resources

- Spring Security 2.0 Website
  - http://static.springframework.org/spring-security/site/
- Sample Applications
  - Small sample included in Spring Security distro
  - Bigger sample on the Spring Portlet Wiki

http://opensource.atlassian.com/confluence/spring/display/JSR168/

Questions & Answers

John A. Lewis
Chief Software Architect
Unicon, Inc.

jlewis@unicon.net
www.unicon.net