**Spring Security**

El siguiente módulo que estudiaremos en nuestro repaso a Spring es el de securización de aplicaciones web. Para ello veremos cómo funciona Spring Security.

Para ver cómo podemos desarrollar aplicaciones web seguras con Spring Security lo primero es entender los conceptos básicos asociados con la seguridad:

* **Principal**: entidad que realiza una acción sobre un recurso (usuario, aplicación, sistema…)
* **Autenticación**: determinar si las credenciales de un principal son válidas (es quién dice ser). Existen diferentes mecanismos como basic, digest o X.509. Las credenciales se pueden almacenar en diferentes repositorios (bases de datos, LDAP…)
* **Autorización**: decidir si el principal tiene permitido realizar la acción (tiene permisos para realizar esa acción sobre el recurso). Normalmente se basa en roles (ADMIN, MEMBER, GUEST…)
* **Recurso securizado**: el recurso que se está protegiendo

Spring Security nos permite definir estas políticas proporcionándonos, además, una serie de ventajas:

* Portabilidad: la seguridad no se define a nivel de Servlet y es portable entre diferentes contenedores.
* Flexibilidad: soporta todos los mecanismos de autenticación habituales (cookies, X.509, SSO…), proporciona opciones de configuración para guardar las credenciales (LDAP, properties, BBDD…) y utiliza Spring para la configuración.
* Extensible: casi cualquier aspecto de la securización de Spring Security es extensible (cómo se definen los principal, dónde se almacena la información, criterios con los que se realiza la autorización…)
* Aislamiento conceptual: la lógina de negocio se desacopla de la seguridad (se utilizan interceptores y AOP) y la autenticación y la autorización se encuentran diferenciadas (un cambio en una no afecta a la otra)
* Consistencia: independientemente de las implementaciones de cada paso, el flujo general de securización es siempre el mismo: consultar la información del recurso securizado, obtener la información del principal y determinar si se permite o no el acceso.

Lo primero que debemos hacer para añadir Spring Security a nuestro proyecto es incluir la siguiente configuración al fichero web.xml para enlazar los filtros de seguridad de Spring Security al contenedor de servlets a través de la clase [DelegatingFilterProxy](http://static.springsource.org/spring/docs/3.0.x/api/org/springframework/web/filter/DelegatingFilterProxy.html" \t "_blank):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <filter>      <filter-name>springSecurityFilterChain</filter-name>      <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>  </filter>    <filter-mapping>    <filter-name>springSecurityFilterChain</filter-name>    <url-pattern>/\*</url-pattern>  </filter-mapping> |

springSecurityFilterChain es el bean encargado de hacer que la petición pase por la [cadena de filtros de seguridad](http://static.springsource.org/spring-security/site/docs/3.0.x/reference/security-filter-chain.html), esta actúa envolviendo al recurso securizado como las capas de una cebolla. Los filtros por defecto y el orden en el que se aplican son:

1. HttpSessionContextIntegrationFilter: establece el contexto de seguridad y mantiene la sesión entre peticiones HTTP
2. LogoutFilter: limpia el contexto de seguridad del principal cuando se solicita un logout
3. AuthenticationProcessingFilter: autentica al principal ante una petición de login y actualiza el contexto de seguridad
4. ExceptionTranslationFilter: convierte excepciones de SpringSecurity en respuestas HTTP o redirecciones
5. FilterSecurityInterceptor: autoriza la petición basándose en la configuración establecida para el principal y el recurso securizado

Para configurar Spring Security en nuestro contexto de aplicación (normalmente en un fichero independiente. Ej: hospital-security.xml) podemos utilizar el namespace “security”:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <security:http>    <security:intercept-url pattern="/admin/\*\*" access="ROLE\_ADMIN"/>    <security:intercept-url pattern="/patients/\*\*" access="IS\_AUTHENTICATED\_FULLY"/>    <security:intercept-url pattern="/\*\*" access="ROLE\_ANONYMOUS"/>    <security:form-login login-page="/login.htm"/>    <security:logout/>  </security:http> |

Lo que estamos diciendo con esta configuración es que queremos que sólo los usuarios con rol de administrador tengan acceso a la ruta /admin/\*\*; sólo los usuarios autenticados tengan acceso a la ruta /patients/\*\* y todos tengan acceso al resto de rutas. Los usuarios harán login en la página /login.htm y además debemos declarar explícitamente la función de logout.

Los interceptores de URLs se aplican según el orden en el que se definan, por lo que debemos poner los más específicos primero. También es posible activar Expression Language (utilizando el atributo use-expressions) para definir interceptores más flexibles:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <security:http use-expressions="true">    <security:intercept-url pattern="/admin/\*\*" access="hasRole('ROLE\_ADMIN') and hasIpAddress("10.21.31.13")"/>    <security:intercept-url pattern="/patients/\*\*" access="isAuthenticated() and hasAnyRole('ROLE\_ADMIN','ROLE\_USER','ROLE\_POWERUSER')"/>    <security:intercept-url pattern="/\*\*" access="permitAll"/>    <security:form-login login-page="/login.htm"/>    <security:logout/>  </security:http> |

Existen muchas expresiones predefinidas de las que podemos hacer uso pero hay que recordar siempre que NO se pueden aplicar ambas sintaxis a la vez.

Veamos cómo podríamos configurar nuestro formulario de login:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <form action="<c:url value="j\_spring\_security\_check"/>" method="POST">    <input type="text" name="j\_username"/>    <br/>    <input type="password" name="j\_password"/>    <br/>    <input type="submit" name="submit" value="login"/>  </form> |

j\_username y j\_password son los parámetros esperados para autenticación por formulario según la especificación [JSR-154](http://jcp.org/aboutJava/communityprocess/final/jsr154/index.html) mientras que j\_spring\_security\_check sería el sustituto del estándar j\_security\_check e indica la URL de autenticación.

**Autenticación**

¿Cómo comprueba Spring Security si el usuario y el password proporcionados son válidos? Tenemos diferentes opciones a la hora de configurar la autenticación a través del Authentication Provider. Para ello configuraremos la propiedad user-service, [que podemos apuntar contra un repositorio en memoria, una base de datos JDBC, un LDAP…](http://static.springsource.org/spring-security/site/docs/3.0.x/reference/ns-config.html#ns-web-xml):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <security:authentication-manager>     <security:authentication-provider>       <security:user-service>          <!-- Implementación específica del user-service, ver: <http://static.springsource.org/spring-security/site/docs/3.0.x/reference/ns-config.html>#ns-web-xml -->          ...       </security:user-service>     </security:authentication-provider>   </security:authentication-manager> |

Normalmente los passwords de los usuarios se guardarán codificados con un algoritmo de hash (MD5, SHA…) además, es buena idea incluir un salt para evitar ataques de fuerza bruta y basados en diccionarios. Lo normal es utilizar una de las propiedades constantes del objeto del usuario para generar este hash, por ejemplo, su identificador:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <security:authentication-manager>      <security:authentication-provider>        <security:password-encoder hash="sha">         <security:salt-source user-property="id"/>        </security:password-encoder>        <security:user-service>           ...        </security:user-service>      </security:authentication-provider>    </security:authentication-manager> |

**Autorización**

Una de las herramientas más utilizadas para proteger recursos es la Tag Library de Spring Security. Ésta nos permite definir zonas protegidas que se generarán sólo si el usuario cumple con nuestros criterios de autorización y también nos permite acceder a propiedades del Principal:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <%@ taglib prefix="security" uri="<http://wwww.springframework.org/security/tags>" %>    <!-- Acceso al principal -->  Bienvenido <security:authentication property="principal.username"/>!    <!-- Autorización mediante rol -->  <security:authorize access="hasRole('ROLE\_ADMIN')">  Sólo un usuario administrador podrá ver este texto  </security>    <!-- Autorización mediante URL securizada -->  <security:authorize url="/admin/deletePatient">  <a href="/admin/deleteDeletePatient?patient\_id=1">Borrar usuario</a>  </security> |

Con Spring Security también es posible securizar métodos mediante AOP utilizando configuración XML, anotaciones de Spring o anotaciones JSR-250:

**Mediante XML**

|  |  |
| --- | --- |
| 1  2  3 | <security:global-method-security>    <security:protect-pointcut expression="execution(\* com.hospital.services.impl.PatientServiceImpl.delete\*(..)" access="hashRole('ROLE\_ADMIN')"/>  </security:global-method-security> |

**Mediante anotaciones Spring**

Primero debemos activar las anotaciones de Spring Security

|  |  |  |
| --- | --- | --- |
| 1 | <security:global-method-security secured-annotations="enabled"/> | |
| 1  2  3  4  5  6  7  8 | public class PatientServiceImpl implements PatientService {      @Secured("ROLE\_ADMIN")    public void deletePatient(Integer patient id) {     ...    }    } |

**JSR-250**

Primero debemos activar las anotaciones de JSR-250

|  |  |  |
| --- | --- | --- |
| 1 | <security:global-method-security jsr250-annotations="enabled"/> | |
| 1  2  3  4  5  6  7  8 | public class PatientServiceImpl implements PatientService {      @RollesAllowed("ROLE\_ADMIN")    public void deletePatient(Integer patient id) {     ...    }    } |