Seguridad no intrusiva con Acegi Security System for Spring

Carlos Sánchez González
Softgal
Plgno. POCOMACO, parcela I, nave
19, 15190 A Coruña - España
carlos@apache.org

Abstract

Uno de los aspectos que toda aplicación debe considerar es la seguridad, entendiendo como tal la necesidad de saber que el usuario es quien dice ser (autenticación), y permitirle acceso sólo a aquellos recursos necesarios (autorización). Acegi Security System for Spring proporciona la funcionalidad necesaria para adoptar mecanismos de seguridad en aplicaciones Java utilizando características de programación orientada a aspectos, de forma transparente para el desarrollador, sin necesidad de desarrollar código, utilizando para ello el soporte prestado por el framework Spring, pero siendo posible utilizarlo en aplicaciones no desarrolladas con Spring. En este artículo se detallarán las funcionalidades que ofrece y una visión detallada sobre la arquitectura del sistema, así como un simple ejemplo que demostrará la sencillez con la que se puede adoptar su uso.

Keywords: Acegi, Spring Framework, seguridad, autenticación, autorización, java.

1 Introducción

Aplicar una política de seguridad a una aplicación es un aspecto que afecta a prácticamente la totalidad de las aplicaciones empresariales, y si no se adopta desde una perspectiva correcta puede llegar a ser una carga que afectará y lastrará el desarrollo del sistema.

Si bien existe el estándar JAAS (Java Authorization and Authentication Service) que pretende cubrir tanto autenticación como autorización, su adopción dista mucho de ser sencilla y portable, debido a que el soporte proporcionado por los contenedores de aplicaciones dista mucho de ser adecuado, existen incompatibilidades entre distintas implementaciones y cada contenedor requiere una configuración distinta, normalmente con adición de librerías. Por otro lado la funcionalidad proporcionada por Acegi es mucho mayor, y además permite la integración con JAAS, utilizándolo en la fase de autenticación.

Acegi Security System [1] es un framework creado por Ben Alex e íntimamente ligado al proyecto Spring [2], si bien no requiere su utilización en nuestra aplicación, que facilita la tarea de adoptar medidas de seguridad en aplicaciones Java, sean aplicaciones *standalone* o aplicaciones web. Y lo mejor de todo es que es *open source*, sin coste de licencias y con la seguridad añadida que proporciona el respaldo de un enorme y creciente grupo de usuarios que lo están utilizando, y con un manual de referencia con más de 50 páginas que no tiene nada que envidiar a la documentación de un producto comercial.

La arquitectura de Acegi está fuertemente basada en interfaces y en patrones de diseño, proporcionando las implementaciones más comúnmente utilizadas y numerosos puntos de extensión donde nuevas funcionalidades pueden ser añadidas. Esta arquitectura puede hacer un poco difícil seguir el flujo de ejecución al principio, pero una vez comprendida la idea global se acepta como el precio necesario para poder disfrutar de un framework con una gran potencia.

Como ejemplo se mostrará la configuración realizada en el proyecto ONess [3] para la protección de peticiones http en una aplicación web, además de

mencionar uno de los completos ejemplos que se distribuyen con el proyecto.

1 Autenticación

Antes de poder tomar decisiones sobre si un usuario puede acceder o no a un recurso, el usuario debe identificarse para comprobar su identidad. Para ello existe el interfaz *Authentication*, desde el que se puede acceder a tres objetos:

- *principal*, típicamente un nombre de usuario.
- *credentials*, las credenciales del usuario que prueban que es quien dice ser, normalmente su contraseña, aunque podría ser otro tipo de información como certificados electrónicos.
- *authorities*, un lista de los roles que posee el usuario o grupos a los que pertenece.

Cuando el usuario se autentica se crea un objeto *Authentication*, con los dos primeros objetos, principal y credenciales. En el caso de autenticación mediante nombre de usuario y contraseña se creará un objeto *UsernamePasswordAuthenticationToken*.

Acegi proporciona las clases necesarias para que esta autenticación se realice mediante usuario y contraseña, utilizando un adaptador para enlazar con la autenticación proporcionada por un contenedor de aplicaciones como son catalina, jboss, resin o jetty, o utilizando el servicio de Single Sign On que proporciona el proyecto CAS de la universidad de Yale [4].

Una vez creado este objeto *Authentication* se pasa al *AuthenticationManager*, que a partir del principal y las credenciales determina si éstas concuerdan con las esperadas, añadiéndole al objeto *Authentication* las authorities correspondientes en caso afirmativo o lanzando una excepción de tipo *AuthenticationException* en caso contrario.

Acegi proporciona una implementación del gestor de autenticación *AuthenticationManager* que debería ser suficiente para la mayoría de los casos, el *ProviderManager*. Esta clase tan sólo delega la autenticación en una lista de proveedores configurable, cada uno de los cuales implementa el interfaz *AuthenticationProvider*. Entre las implementaciones de proveedores suministradas con el proyecto se encuentran las necesarias para realizar la autenticación contra varios servidores de aplicaciones (catalina, jboss, resin y jetty), contra un fichero de configuración JAAS, contra CAS (la solución Single Sign On de la universidad de Yale [4]), y contra un objeto de acceso a datos usando *DaoAuthenticationProvider*, que es el comúnmente usado puesto que es el que permite acceder a la información almacenada en una base de datos.

El proveedor *DaoAuthenticationProvider* merece una mención especial. Esta implementación delega a su vez en un objeto de tipo *AuthenticationDao*, un interfaz que define un objeto de acceso a datos con un único método *loadUserByUsername* que permite obtener la información de un usuario a partir de su nombre de usuario. Acegi proporciona dos implementaciones de este interfaz, *InMemoryDaoImpl*, en la que la información de los usuarios se guarda en memoria, útil para la realización de pruebas, y JdbcDaoImpl, que accede a una base de datos a través de JDBC. Realizar implementaciones de este interfaz es sumamente sencillo y en el proyecto ONess [3] se encuentra disponible una implementación que utiliza el mapeador objeto-relacional Hibernate.

Entre otras características que también se proporcionan de forma transparente, tan sólo estableciendo unos parámetros de configuración, son soporte para cifrado de contraseñas (SHA y MD5), caché de la información de autenticación y redirección automática de peticiones http a canales seguros https para aquellas urls que deseemos.

Para el caso de aplicaciones web existen tres formas de que un usuario se autentique:

- Utilizando autenticación de tipo BASIC, definida en el RFC 1945, el usuario introduce su usuario y contraseña en una simple ventana emergente del navegador. Es necesario en el caso de que se quieran añadir características de seguridad a servicios web.
- Autenticándose mediante un formulario web, es la forma más habitual ya que permite integrar el formulario de login en la aplicación web.

- Utilizando el servicio de autenticación central CAS de la Universidad de Yale [4], en caso de que se requieran características de *Single Sign On*, de forma que el usuario sólo tiene que autenticarse una vez para todos los servicios que puedan proporcionarse en el ámbito de una empresa, incluso en distintos servidores y desarrollados con distintos lenguajes de programación.

Cada una de las formas anteriores requiere de la configuración del filtro correspondiente en el descriptor de aplicación web, *BasicProcessingFilter*, *AuthenticationProcessingFilter* o *CasProcessingFilter* respectivamente. La forma de configurarlos es definir los filtros como del tipo *FilterToBeanProxy*, delegando, según un parámetro de inicialización, en uno de los filtros anteriores definidos en el contexto de aplicación de Spring, lugar donde pueden ser más fácilmente configurados.

Los clientes llamados "ricos" o aplicaciones *standalone* también están soportados, utilizando un gestor de autenticación remoto cuya implementación utiliza un servicio web en el lado del servidor, utilizando *RemoteAuthenticationManager* y *RemoteAuthenticationProvider*.

2 Autorización

Una vez el usuario está autenticado entra en juego la parte del sistema encargada de la autorización, con el fin de permitir que el usuario acceda sólo a aquellos recursos a los que tiene permiso. Para ello Acegi intercepta las llamadas a los objetos, utilizando proxies dinámicos u orientación a aspectos basada en AspectJ, o las peticiones http, utilizando filtros, y actúa en consecuencia. Así permite restringir tanto llamadas a métodos de determinadas clases o instancias, así como acceso a urls.

Cuando se intercepta una petición a un recurso protegido se comienza una cadena de eventos que finalizará permitiendo el acceso al recurso o lanzando una excepción *AccessDeniedException*. La cadena comienza en un objeto de tipo *AccessDecisionManager*, que a partir del objeto *Authentication* y de los parámetros de configuración decide si la llamada debe proseguir. Acegi proporciona tres implementaciones de *AccessDecisionManager* que se basan en el concepto de una votación, pero diferenciando las reglas de decisión:

- UnanimousBased: permite el acceso si no hay votos negativos
- AffirmativeBased: permite el acceso si un voto es afirmativo
- ConsensusBased: permite el acceso si el número de votos positivos es mayor o igual que el de negativos

Al igual que en la autenticación el *ProviderManager* delegaba en una lista de *AuthenticationProvider*s, en el caso de la autorización el *AccessDecisionManager* delega la facultad de emitir votos en objetos de tipo *AccessDecisionVoter*. Se proporcionan dos implementaciones de éste último interfaz:

- *RoleVoter*, que comprueba que el usuario presente un determinado rol, comprobando si se encuentra entre sus *authorities*.
- *BasicAclEntryVoter*, que a su vez delega en una jerarquía de objetos que permite comprobar si el usuario supera las reglas establecidas como listas de control de acceso.

El primer caso es el más común, proporcionando una autenticación basada en grupos o roles, donde se permite el acceso si el usuario pertenece a alguno de los configurados como requeridos. En el segundo caso se permite restringir el acceso a objetos a nivel de instancia, caso que será discutido más adelante.

En ambos casos el sistema que intercepta las llamadas debe ser configurado. En el caso de las aplicaciones web se hará mediante la configuración de un filtro en el fichero web.xml.

Los posibles recursos que se pueden proteger son

- urls, mediante un filtro en el descriptor de aplicación web, *FilterSecurityInterceptor*.
- métodos de objetos definidos en el contexto de aplicación de Spring, utilizando *MethodSecurityInterceptor*.
- cualquier PointCut definible en AspectJ, mediante *AspectJSecurityInterceptor*.

2 Autorización a nivel de instancia mediante listas de control de acceso

Existen casos en los que la protección de las llamadas a métodos no es suficiente, necesitando protegerse de distinta forma distintas instancias de una clase. Como ejemplo se puede pensar en un sistema de ficheros, en los que cada archivo tiene distintos permisos, según si el usuario que accede a ellos es el dueño del archivo, pertenece al grupo del dueño o no cumple ninguna de las opciones anteriores.

Acegi proporciona en sus últimas versiones el soporte necesario para implementar seguridad basada en listas de control de acceso. Las clases clave que se deben conocer son

- BasicAclEntryVoter obtiene las ACLs del objeto llamado y vota sobre si se debe permitir el acceso a él o no.
- BasicAclAfterInvocationProvider permite denegar el acceso a un objeto después de que el método se haya invocado, útil cuando no se puede saber a priori.
- BasicAclAfterInvocationollectionFilteringProvider, similar al anterior, elimina los objetos a los que el acceso no ha sido permitido en aquellos métodos que devuelven colecciones.

Para más detalles sobre ACLs se recomienda consultar el completo manual de referencia de Acegi.

2 Ejemplo

Como ejemplo se utilizará el proyecto ONess [3], subproyectos *user-model* y *user-webapp*. En este proyecto se ha configurado una aplicación web para proteger sus recursos en peticiones http. Se han omitido partes no relevantes para este ejemplo, pero que pueden ser consultadas en la página web y en el repositorio de código fuente del proyecto, entre otros la configuración necesaria para utilizar autenticación basada en HTTP BASIC o CAS o las clases necesarias para ejecutarlo.

Acegi utiliza el contexto de aplicación de Spring para definir la configuración necesaria. Para aquellas personas que no están familiarizados con Spring decir que tan sólo es necesario crear un fichero /WEB-INF/applicationContext.xml con el contenido que se muestra en la Fig. 1 y añadir a /WEB-INF/web.xml el contenido de la Fig. 2. En este segundo fichero se configura un *listener* que procesa el primero automáticamente cada vez que el contenedor de aplicaciones inicia la aplicación web, y unos filtros que procesan todas las peticiones que llegan.

La autenticación se realiza a través de un objeto de acceso a datos DAO implementado con el mapeador objeto-relacional Hibernate, authenticationDao, para acceder a la información de usuarios almacenada en una base de datos. Como alternativa para realizar pruebas también se incluye comentada la definición de un DAO de tipo *InMemoryDaoImpl*. Como proveedor de autenticación se utiliza por tanto *DaoAuthenticationProvider*, configurándose con una caché de usuarios basada en EHCache. Por comodidad no se ha activado el cifrado de contraseñas, acción que puede realizarse con tan sólo descomentar la línea indicada. El gestor de autenticación *ProviderManager* tan sólo tendrá como proveedor el anteriormente mencionado, ya que el único repositorio de usuarios será la base de datos.

En la aplicación web es necesario añadir dos filtros, *Acegi Security System for Spring Http Session Integration Filter*, que hace que la información de autenticación esté disponible para sucesivas peticiones del usuario al guardarla en la sesión, y *Acegi Authentication Processing Filter* para procesar el formulario de login de un usuario. El primer filtro no requiere configuración, mientras que el segundo la delega en el contexto de aplicación de Spring, definiendo un bean authenticationProcessingFilter, donde se referencia el gestor de autenticación anteriormente configurado y la página a la que ir en caso de error en el login, entre otros, y authenticationProcessingFilterEntryPoint, donde se configura la página donde se encuentra el formulario de login.

En cuanto a autorización, las decisiones se tomarán basándose en los roles del usuario utilizando *RoleVoter*, y puesto que tan sólo existe ese *AccessDecisionVoter* no influirá el gestor de decisiones, optando por un *AffirmativeBased*.

En la aplicación web se configurará un filtro *Acegi HTTP Request Security Filter* para restringir el acceso

a determinadas urls. Este filtro al igual que los anteriores se configura mediante el contexto de aplicación de Spring, donde se define un FilterSecurityInterceptor, filterInvocationInterceptor, que define los roles necesarios para acceder a las urls utilizando comodines, y se define también securityEnforcementFilter, donde se enlazan el interceptor, para el caso en el que el usuario ya está autenticado, y el punto de entrada, para el caso contrario.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>

 <!-- AUTENTICACION -->

 <!-- Data access object which stores authentication information -->
 <!-- Hibernate implementation -->
 <bean id="authenticationDao"
    class="net.sf.oness.user.model.dao.UserHibernateDao">
  <property name="sessionFactory">
   <ref bean="sessionFactory" />
  </property>
 </bean>

 <!-- Implementacion util para pruebas
 <bean id="authenticationDao"
    class="net.sf.acegisecurity.providers.dao.memory.InMemoryDaoImpl">
  <property name="userMap">
    <value>
      marissa=koala,ROLE_USER,ROLE_ADMIN
      dianne=emu,ROLE_USER
      scott=wombat,ROLE_USER
      peter=opal,disabled,ROLE_USER
    </value>
  </property>
 </bean>
 -->

 <bean id="daoAuthenticationProvider"
    class="net.sf.acegisecurity.providers.dao.DaoAuthenticationProvider">
  <property name="authenticationDao"><ref local="authenticationDao"/></property>
  <property name="userCache"><ref local="userCache"/></property>
  <!-- Descomentar para activar cifrado de contraseñas
  <property name="passwordEncoder"><ref local="passwordEncoder"/></property>
  -->
 </bean>

 <bean id="passwordEncoder"
    class="net.sf.acegisecurity.providers.encoding.Md5PasswordEncoder"/>

 <bean id="userCache"
    class="net.sf.acegisecurity.providers.dao.cache.EhCacheBasedUserCache">
  <property name="minutesToIdle"><value>5</value></property>
 </bean>

 <bean id="authenticationManager"
    class="net.sf.acegisecurity.providers.ProviderManager">
  <property name="providers">
   <list>
    <ref local="daoAuthenticationProvider"/>
   </list>
  </property>
 </bean>

 <!-- Filtro web -->
```

```
<bean id="authenticationProcessingFilter"
   class="net.sf.acegisecurity.ui.webapp.AuthenticationProcessingFilter">
 <property name="authenticationManager">
  <ref bean="authenticationManager"/>
 </property>
 <property name="authenticationFailureUrl">
  <value><![CDATA[/show.do?page=.login&login_error=1]]></value>
 </property>
 <property name="defaultTargetUrl"><value>/</value></property>
 <property name="filterProcessesUrl"><value>/security_check</value></property>
</bean>

<bean id="authenticationProcessingFilterEntryPoint"
   class="net.sf.acegisecurity.ui.webapp.AuthenticationProcessingFilterEntryPoint">
 <property name="loginFormUrl">
  <value><![CDATA[/show.do?page=.login]]></value>
 </property>
 <property name="forceHttps"><value>false</value></property>
</bean>


<!-- AUTORIZACION -->

<bean id="roleVoter" class="net.sf.acegisecurity.vote.RoleVoter"/>

<bean id="accessDecisionManager"
   class="net.sf.acegisecurity.vote.AffirmativeBased">
 <property name="allowIfAllAbstainDecisions"><value>false</value></property>
 <property name="decisionVoters">
   <list>
  <ref local="roleVoter"/>
   </list>
 </property>
</bean>

<!-- Filtro web -->

<bean id="securityEnforcementFilter"
   class="net.sf.acegisecurity.intercept.web.SecurityEnforcementFilter">
 <property name="filterSecurityInterceptor">
  <ref bean="filterInvocationInterceptor"/>
 </property>
 <property name="authenticationEntryPoint">
  <ref local="authenticationProcessingFilterEntryPoint"/>
 </property>
</bean>

<bean id="filterInvocationInterceptor"
   class="net.sf.acegisecurity.intercept.web.FilterSecurityInterceptor">
 <property name="authenticationManager">
  <ref bean="authenticationManager"/>
 </property>
 <property name="accessDecisionManager">
  <ref bean="accessDecisionManager"/>
 </property>
 <property name="objectDefinitionSource">
  <value>
   CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
   PATTERN_TYPE_APACHE_ANT
   /secure/**=ROLE_ADMIN
   /**/*create*=ROLE_USER,ROLE_ADMIN
   /**/*edit*=ROLE_USER,ROLE_ADMIN
   /**/*update*=ROLE_USER,ROLE_ADMIN
   /**/*delete*=ROLE_USER,ROLE_ADMIN
  </value>
 </property>
</bean>
```

```
</beans>
```

Figura 1. Spring application context

```
<!-- carga el contexto de aplicación de /WEB-INF/applicationContext.xml -->
<listener>
 <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<filter>
 <filter-name>Acegi Security System for Spring Http Session Integration Filter</filter-
name>
 <filter-class>net.sf.acegisecurity.ui.HttpSessionIntegrationFilter</filter-class>
</filter>
<filter>
 <filter-name>Acegi Authentication Processing Filter</filter-name>
 <filter-class>net.sf.acegisecurity.util.FilterToBeanProxy</filter-class>
 <init-param>
  <param-name>targetClass</param-name>
  <param-value>net.sf.acegisecurity.ui.webapp.AuthenticationProcessingFilter</param-
value>
 </init-param>
</filter>
<filter>
 <filter-name>Acegi HTTP Request Security Filter</filter-name>
 <filter-class>net.sf.acegisecurity.util.FilterToBeanProxy</filter-class>
 <init-param>
  <param-name>targetClass</param-name>
  <param-value>net.sf.acegisecurity.intercept.web.SecurityEnforcementFilter</param-
value>
 </init-param>
</filter>

<filter-mapping>
 <filter-name>Acegi Security System for Spring Http Session Integration Filter</filter-
name>
 <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
 <filter-name>Acegi Authentication Processing Filter</filter-name>
 <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>Acegi HTTP Request Security Filter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Figura 2. web.xml

Otro completísimo ejemplo "*contacts*" puede encontrarse en la distribución de Acegi Security, incluyendo autorización basada en listas de control de acceso, y acceso remoto desde aplicaciones *standalone*.

8 Conclusiones

Acegi es uno de los mejores frameworks de seguridad existentes en Java, potente y flexible a la vez que sencillo de configurar, sin necesitad de modificar código ya existente y portable entre distintos contenedores de aplicaciones sin necesidad de cambios. Su integración con Spring hace que sea el recomendado para añadir funcionalidades de seguridad a las aplicaciones que utilizan ese magnífico framework, cuyo número crece día a día, si bien puede ser utilizado en cualquier tipo de aplicación sin ningún problema.

En este artículo se dado una visión global del framework y se ha mostrado con un ejemplo cómo se pueden proteger las urls de una aplicación web sin necesidad de modificar ni una línea de código. Para mayor información se recomienda la lectura del manual de referencia, realmente completo.

Agradecimientos

A Ben Alex, principal desarrollador del proyecto Acegi, por su colaboración.

Referencias

[1] Acegi Security System for Spring *http://acegisecurity.sourceforge.net*.

[2] Spring Framework *http://www.springframework.org*.

[3] ONess *http://oness.sourceforge.net*

[4] CAS (Central Authentication Service). *Universidad de Yale, http://www.yale.edu/tp/auth/*.

[5] Weblog del autor *http://www.jroller.com/page/carlossg*