

# Using CAS with Acegi in the Spring Framework

---

Mark Allsop

July-2006

[www.mallsop.com](http://www.mallsop.com)

# Purpose

---

This presentation will show you how to secure your Spring Java Web application in a test environment using CAS and Acegi.

Jars and Java JDK used:

- ❑ Yale CAS client jar 2.1.0.
- ❑ Acegi security jar 1.0.0.
- ❑ Acegi security CAS jar 1.0.0.
- ❑ Java jdk1.5.0\_04



# XML and Servlets

---

- web.xml
- security-context.xml
- exampleUsingCas-servlet.xml

# web.xml

---

```
<filter>
  <filter-name>Acegi Filter Chain Proxy</filter-name>
  <filter-class>org.acegisecurity.util.FilterToBeanProxy</filter-class>
  <init-param>
    <param-name>targetClass</param-name>
    <param-value>org.acegisecurity.util.FilterChainProxy</param-value>
  </init-param>
</filter>
```

```
<filter-mapping>
  <filter-name>Acegi Filter Chain Proxy</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Note: Acegi delegates to a bean proxy in Spring's application context using FiltertoBeanProxy.

All Acegi filters will be configured using an interface of this class.

All must be provided a targetClass to targetBean via an init parameter that points to the bean in the application context.

The application context bean is configured with parameters rather than the filter.

# security-context.xml

---

```
<!-- FILTER CHAIN – order is important -->
<bean id="filterChainProxy" class="org.acegisecurity.util.FilterChainProxy">
  <property name="filterInvocationDefinitionSource">
    <value>
      CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
      PATTERN_TYPE_APACHE_ANT
      /***=channelProcessingFilter,httpSessionContextIntegrationFilter,casProcessingFilter,
        basicProcessingFilter,exceptionTranslationFilter,filterInvocationInterceptor
    </value>
  </property>
</bean>
```

*channelProcessingFilter*: Ensures a web request is delivered over the required channel.

*httpSessionContextIntegrationFilter*: Populates the [SecurityContextHolder](#) with information obtained from the HttpSession.

*httpSessionContextIntegrationFilter*: This filter works hand-and-hand with the HTTP Session object and the Web context to see if the user is authenticated and, if so, then what roles the user has.

*casProcessingFilter*: Processes a CAS service ticket.

*basicProcessingFilter*: Processes a HTTP request's BASIC authorization headers, putting the result into the SecurityContextHolder.

*filterInvocationInterceptor*: Performs security handling of HTTP resources via a filter implementation.

*exceptionTranslationFilter*: Handles any AccessDeniedException and AuthenticationException thrown within the filter chain. This filter is necessary because it provides the bridge between Java exceptions and HTTP responses.

# security-context.xml continued...

---

```
<!-- AUTHENTICATION - inMemoryDaoImpl can be used to easily test CAS with acegi -->
```

```
<bean id="authenticationManager" class="org.acegisecurity.providers.ProviderManager">  
  <property name="providers">  
    <list><ref local="casAuthenticationProvider"/></list>  
  </property>  
</bean>
```

```
<bean id="inMemoryDaoImpl" class="org.acegisecurity.userdetails.memory.InMemoryDaoImpl">  
  <property name="userMap">  
    <value>  
      password=,ROLE_CAS_USER  
    </value>  
  </property>  
</bean>
```

Note: ProviderManager is just a wrapper class around a list of one or more Authentication Providers provided to the class. It cycles thru the list until a compatible provider is located. If it finds one, it returns a fully populated authentication object or throws an authentication exception. The authenticationManager uses the ProviderManager in your xml.

# security-context.xml continued...

---

<!-- This filter is used for acegi to talk to the CAS server. See authenticationManager. -->

```
<bean id="basicProcessingFilter" class="org.acegisecurity.ui.basicauth.BasicProcessingFilter">
  <property name="authenticationManager">
    <ref local="authenticationManager"/></property>
  <property name="authenticationEntryPoint">
    <ref local="basicProcessingFilterEntryPoint"/></property>
</bean>
```

```
<bean id="basicProcessingFilterEntryPoint"
      class="org.acegisecurity.ui.basicauth.BasicProcessingFilterEntryPoint">
  <property name="realmName"><value>YOUR Realm</value></property>
</bean>
```

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

# security-context.xml continued...

---

<!-- The test key is used below. See the test certificate keytool setup information in the later slides. -->

```
<bean id="casAuthenticationProvider" class="org.acegisecurity.providers.cas.CasAuthenticationProvider">
  <property name="casAuthoritiesPopulator"><ref local="casAuthoritiesPopulator"/></property>
  <property name="casProxyDecider"><ref local="casProxyDecider"/></property>
  <property name="ticketValidator"><ref local="casProxyTicketValidator"/></property>
  <property name="statelessTicketCache"><ref local="statelessTicketCache"/></property>
  <property name="key"><value>changeit</value></property>
</bean>
```

```
<bean id="casProxyTicketValidator"
  class="org.acegisecurity.providers.cas.ticketvalidator.CasProxyTicketValidator">
  <property
  name="casValidate"><value>https://yourCASserver:8443/cas/proxyValidate</value></property>
  <property name="serviceProperties"><ref local="serviceProperties"/></property>
</bean>
```

```
<bean id="cacheManager" class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean"/>
```

# security-context.xml continued...

---

```
<bean id="ticketCacheBackend"  
    class="org.springframework.cache.ehcache.EhCacheFactoryBean">  
    <property name="cacheManager">  
        <ref local="cacheManager"/>  
    </property>  
    <property name="cacheName">  
        <value>ticketCache</value>  
    </property>  
</bean>
```

```
<!-- Caches CAS service tickets and CAS proxy tickets for stateless connections -->  
<bean id="statelessTicketCache"  
    class="org.acegisecurity.providers.cas.cache.EhCacheBasedTicketCache">  
    <property name="cache"><ref local="ticketCacheBackend"/></property>  
</bean>
```

# security-context.xml continued...

---

```
<bean id="casAuthoritiesPopulator"  
      class="org.acegisecurity.providers.cas.populator.  
      DaoCasAuthoritiesPopulator">  
  <property name="userDetailsService">  
    <ref local="inMemoryDaoImpl" />  
  </property>  
</bean>
```

- Populates the UserDetails associated with a CAS authenticated user.
- CAS does not provide the authorities (roles) granted to a user. It merely authenticates their identity. As the Acegi Security System for Spring needs to know the authorities granted to a user in order to construct a valid Authentication object. Acegi ignores the password because it was enforced by CAS.
- The UserDetails returned by implementations is stored in the generated CasAuthenticationToken, so additional properties such as email addresses, telephone numbers etc can easily be stored.

# security-context.xml continued...

---

```
<!-- Proxy tickets are not used in the test. -->
```

```
<bean id="casProxyDecider" class="org.acegisecurity.providers.cas.proxy.RejectProxyTickets">  
</bean>
```

```
<!-- This class stores the properties that are relevant to the local CAS service. -->
```

```
<bean id="serviceProperties" class="org.acegisecurity.ui.cas.ServiceProperties">  
  <property name="service">  
    <value>https://mylocalipaddress:8443/exampleUsingCAS/j_acegi_cas_security_check</value>  
  </property>  
  <property name="sendRenew"><value>>false</value></property>  
</bean>
```

# security-context.xml continued...

---

```
<!-- HTTP CHANNEL REQUIREMENTS -->
<!-- This prevents the end user from going to http, as it redirects it to https automatically. -->
  <!-- Enabled by default for CAS, as a CAS deployment uses HTTPS -->
  <bean id="channelProcessingFilter"
    class="org.acegisecurity.securechannel.ChannelProcessingFilter">
    <property name="channelDecisionManager"><ref
    local="channelDecisionManager"/></property>
    <property name="filterInvocationDefinitionSource">
      <value>
        CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
        \A/secure/.*\Z=REQUIRES_SECURE_CHANNEL

        \A/j_acegi_cas_security_check.*\Z=REQUIRES_SECURE_CHANNEL
        \A.*\Z=REQUIRES_SECURE_CHANNEL
      </value>
    </property>
  </bean>
```

# security-context.xml continued...

---

<!-- More of the https redirect.-->

```
<bean id="channelDecisionManager" class="org.acegisecurity.securechannel.ChannelDecisionManagerImpl">
    <property name="channelProcessors">
        <list>
            <ref local="secureChannelProcessor"/>
            <ref local="insecureChannelProcessor"/>
        </list>
    </property>
</bean>
```

```
<bean id="secureChannelProcessor" class="org.acegisecurity.securechannel.SecureChannelProcessor"/>
<bean id="insecureChannelProcessor" class="org.acegisecurity.securechannel.InsecureChannelProcessor"/>
```

<!-- HTTP REQUEST SECURITY -->

```
<bean id="exceptionTranslationFilter" class="org.acegisecurity.ui.ExceptionTranslationFilter">
    <property name="authenticationEntryPoint"><ref local="casProcessingFilterEntryPoint"/></property>
</bean>
```

# security-context.xml continued...

---

```
<bean id="casProcessingFilter" class="org.acegisecurity.ui.cas.CasProcessingFilter">
  <property name="authenticationManager"><ref local="authenticationManager"/></property>
  <property name="authenticationFailureUrl"><value>/casfailed.jsp</value></property>
  <property name="defaultTargetUrl"><value>/secure/</value></property>
  <property name="filterProcessesUrl"><value>/j_acegi_cas_security_check</value></property>
</bean>
<!-- The location of your CAS server login page.-->
<bean id="casProcessingFilterEntryPoint" class="org.acegisecurity.ui.cas.CasProcessingFilterEntryPoint">
  <property name="loginUrl"><value>https://alphaserver.lan.here.org:8443/cas/login</value></property>
  <property name="serviceProperties"><ref local="serviceProperties"/></property>
</bean>
<!--Role voting.-->
<bean id="httpRequestAccessDecisionManager" class="org.acegisecurity.vote.AffirmativeBased">
  <property name="allowIfAllAbstainDecisions"><value>>false</value></property>
  <property name="decisionVoters">
    <list>
      <ref bean="roleVoter"/>
    </list>
  </property>
</bean>
```

# security-context.xml continued...

---

<!-- Note the order that entries are placed against the objectDefinitionSource is **critical**.

The FilterSecurityInterceptor will work from the top of the list down to the FIRST pattern that matches the request URL. Accordingly, you should place MOST SPECIFIC (ie a/b/c/d.\*) expressions first, with LEAST SPECIFIC (ie a/.\* ) expressions last -->

```
<bean id="filterInvocationInterceptor" class="org.acegisecurity.intercept.web.FilterSecurityInterceptor">
  <property name="authenticationManager"><ref local="authenticationManager"/></property>
  <property name="accessDecisionManager"><ref local="httpRequestAccessDecisionManager"/></property>
  <property name="objectDefinitionSource">
    <value>
      CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
      \A/secure/super.*\Z=ROLE_WE_DONT_HAVE
      \A/secure/.*\Z=ROLE_CAS_USER
    </value>
  </property>
</bean>

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

<!-- end of security context -->
```

# exampleUsingCAS-servlet.xml

---

```
<bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basename"> <value>messages</value> </property>
</bean>

<bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="urlMap">
        <map><entry key="/secure/index.html"><ref bean="exampleUsingCASController"/></entry></map>
    </property>
</bean>

<bean id="exampleUsingCASController"
    class="org.here.exampleUsingCAS.web.ExampleUsingCASController">
    <property name="successView" value="view"/> <!-- goes to /secure/index.jsp -->
<property name="failureView" value="noView"/> <!-- goes to /casFailed.jsp -->
</bean>

<bean id="viewResolver" class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
    <property name="basename"><value>views</value></property>
</bean>
```

# Index.jsp in the secure directory

---

```
<% @ page import="org.acegisecurity.context.SecurityContextHolder" %>
<% @ page import="org.acegisecurity.Authentication" %>
<% @ page import="org.acegisecurity.GrantedAuthority" %>
<% @ page import="org.acegisecurity.adapters.AuthByAdapter" %>
Welcome to the Secure area - exampleUsingCAS<br>
<% Authentication auth = SecurityContextHolder.getContext().getAuthentication();
if (auth != null) { %>
Authentication object is of type: <%= auth.getClass().getName() %><BR><BR>
Authentication object as a String: <%= auth.toString() %><BR><BR>
Authentication object holds the following granted authorities:<BR><BR>
<%   GrantedAuthority[] granted = auth.getAuthorities();
      for (int i = 0; i < granted.length; i++) { %>
        <%= granted[i].toString() %> (getAuthority(): <%= granted[i].getAuthority() %>)<BR>
<%   }
      if (auth instanceof AuthByAdapter) { %>
        <BR><B>SUCCESS! Your container adapter appears to be properly configured!</B><BR><BR>
<%   } else { %>
        <BR><B>SUCCESS! Your web filters appear to be properly configured!</B><BR>
<%   }
      } else { %>
        Authentication object is null.<BR>
        This is an error and your Acegi Security application will not operate properly until corrected.<BR><BR>
<%   } %>
```

# casFailed.jsp – in the non secure directory

---

```
<% @ taglib prefix='c' uri='http://java.sun.com/jstl/core' %>
<% @ page import="org.acegisecurity.ui.AbstractProcessingFilter"%>
<% @ page import="org.acegisecurity.AuthenticationException"%>

<html>
  <head>
    <title>Login to CAS failed!</title>
  </head>
  <body>
    <h1>Login to CAS failed!</h1>
    <font color="red">
      Your CAS credentials were rejected.<BR><BR>
      Reason: <%= ((AuthenticationException)
        session.getAttribute(AbstractProcessingFilter.ACEGI_SECURITY_LAST_EXCEPTION_KEY)).getMessage() %>
    </font>
  </body>
</html>
```

# Creating test certificates

---

Setup a test CAS server somewhere. Install a temp certificate.  
We'll call our server "alphaserver".

Edit server.xml in your tomcat conf directory.

Uncomment the port="8443" connector section.

This enables SSL access on port 8443 (the default for https is 443, but just as Tomcat uses 8080 instead of 80 to avoid conflicts, 8443 is used instead of 443 here).

Now create a SSL key for Tomcat to send to connecting clients.

Go to C:\jdk1.5.0\_04\bin. When you run keytool below, specify a password value of "changeit".

You can create a self-signed key for testing purposes with one of the following commands:

```
%JAVA_HOME%\bin\keytool -genkey -alias tomcat -keyalg RSA (Windows)
```

```
$JAVA_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA (Unix)
```

# Creating test certificates...continued.

---

Once Tomcat is restarted, you should be able to access <https://localhost:8443/>.

How To put the your alphaserver certificate into your local pc`s keystore for testing:

1. Download the alphaserver.cer and place in the user 'web' home folder to your  
jdk1.5.0\_04\jre\lib\security folder.

3. Cd to jdk1.5.0\_04\jre\lib\security in DOS.

4. Run the tool:

```
keytool -import -v -file alphaserver.cer -keypass changeit -keystore cacerts -storepass changeit -  
alias alphaserver
```

You should see: Trust this certificate? [no]: yes

Certificate was added to keystore [Storing cacerts]

This allows your localhost to trust the alphaserver certificate. Otherwise you will get a CAS rejection error.

Manual tests may work, but acceptance tests may not work. It does not seem to like the certificate warning.

To add a CA-issued key pair, see the Tomcat documentation. (I did not need to do this.)



# Related Links

---

<http://forum.springframework.org/>

<http://www.acegisecurity.org/>