Copyright
Copyright © 2006-2007 Securent, Inc. All Rights Reserved.

Restricted Rights Legend
This software and documentation is subject to and made available only pursuant to the
terms of the Securent Inc. License Agreement and may be used or copied only in
accordance with the terms of that agreement. It is against the law to copy the software
except as specifically allowed in the agreement. This document may not, in whole or in
part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium
or machine-readable form without prior consent, in writing, from Securent, Inc.

THE SOFTWARE AND DOCUMENTATION ARE PROVIDED, AS IS. WITHOUT WARRANTY OF
ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR
FITNESS FOR A PARTICULAR PURPOSE. FURTHER, Securent DOES NOT WARRANT,
GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS
OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS,
ACCURACY, RELIABILITY, OR OTHERWISE.
Introduction

This document provides guidelines for deployment of the **ACEGI Agent** explains the features of using Securent customized ACEGI authorization solution for applications running in the Spring framework.

**ACEGI Approach**

ACEGI Security provides comprehensive security services for J2EE-based enterprise software applications, particularly, applications using the Spring Framework. ACEGI provides a strong security cover to these applications while switching server environments, where the user must reconfigure the security features.

ACEGI stipulates the Authentication as well as Authorization features of an application. In order to make the security cover more explicable and stronger, the security configuration is customized in terms of Securent functionalities which will henceforth attend the authorization part assuming that ACEGI still meets the authentication feature.

When a user tries to access a Spring-based security implementation, the authentication of the user is left upto the ACEGI, whereas the user authorization is performed by Securent by customizing the accessibility features in terms of the later.

Take for example, in a banking application protected by ACEGI, user *Mary* has access to Report6 only. In order to make the authorization process more stringent, the user info of Mary is reconfigured using the advanced features of Securent (e.g. creating rule based policy for report6 involving usertype, user attributes etc). When Mary tries to access Report6, assuming that login authentication has already been done by ACEGI, the authorization permission will be delivered by Securent.

**Steps to integrate ACEGI agent with a web application:**

1. Copy libraries `spring.jar`, `acegi_agent.jar`, `acegi_security.jar`, `pep.jar` and `papclient.jar` from `PROJECTHOME\WEB-INF\lib` directory where `PROJECTHOME` is the root directory of the web application.

2. Copy the `pep_config.xml` file from `PROJECTHOME\WEB-INF\classes` folder and update the parameters in the following way:
   - specify applicationgroup and application values to the application group and web-application for which you want to check the permissions
   - modify the URL value of `<pdp>` to the url on which PDP Server is running
   - modify URL value of `<api>` to the url on which PAP Server is running

3. Add `securent.agentconfig` to Java property for specifying the `pep_config.xml` path. For example: `-Dsecurent.agentconfig=<pep_config file path>

4. Edit `PROJECTHOME\WEB-INF\web.xml` and add following filter entries before `</web-app>` tag.
<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</url-pattern>
</filter-mapping>

Modify the value of <url-pattern> i.e 'URL_pattern' to the URL pattern to be secured.

5. Create users who are going to use this web application under the application with the name specified in pep_config.xml within Securent.

6. In order to protect urls, edit the Spring configurations file which located in PROJECTHOME\WEB-INF\applicationContext.xml by default in the following ways:
   I. add following bean entries between <beans> and </beans> tags:

   <bean id="filterChainProxy" class="org.acegisecurity.util.FilterChainProxy">
     <property name="filterInvocationDefinitionSource">
       <value>
         CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON PATTERN_TYPE_APACHE_ANT
         URL_pattern*=filterInvocationInterceptor
       </value>
     </property>
   </bean>

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

   <bean id="filterInvocationInterceptor"
        class="net.securent.agent.acegi.SecurentFilterSecurityInterceptor">
     <property name="authenticationManager">
       <ref bean="authenticationManager"/>
     </property>

     <property name="accessDecisionManager">
       <ref local="securentDecisionManager"/>
     </property>

     <property name="objectDefinitionSource">
       <value>
         CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON PATTERN_TYPE_APACHE_ANT
         URL_pattern*=role_allowed
       </value>
     </property>
   </bean>

   <bean id="exceptionTranslationFilter"
        class="org.acegisecurity.ui.ExceptionTranslationFilter">
     <property name="authenticationEntryPoint">
       <ref local="authenticationProcessingFilterEntryPoint"/>
     </property>
   </bean>
II. Update the following bean entries:

- In the bean entry with id = 'filterChainProxy', update the value of 'URL_pattern' to the URL pattern to be secured in lower case. [For example /* or /log/portfolio.htm]

- In the bean entry with id = 'filterInvocationInterceptor', update the value of 'URL_pattern' to the URL pattern to be secured in lower case. [For example /* or /log/portfolio.htm]

- Also update the value of 'role_allowed' to the role specified in the class which implements 'Authentication' interface. If using sample implementation provided with the agent which is 'SecurentAuthentication' then the value of 'role_allowed' should be 'SECURENT_USER'.

- In the bean entry with id = 'authenticationProcessingFilterEntryPoint', update the value of the property 'loginFormUrl' to the URL which should be displayed if the user requests a secure HTTP resource but is not authenticated.

7. In order to protect the methods, edit the Spring configurations file which is (by default) located at PROJECTHOME\WEB-INF\applicationContext.xml in the following way:

I. Add following bean entries after <beans> tags

```xml
<bean id="autoProxyCreator"
     class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">
  <property name="interceptorNames">
    <list>
      <value>Interceptor_Names</value>
    </list>
  </property>
</bean>
```
8. Update the following bean entries

- In the bean entry with id = 'Interceptor_Name', update the value of 'Method' to the methods to be secured.
  For Example: com.devx.tradingapp.web.PortfolioItemBean.setGainLoss (or) com.devx.tradingapp.web.PortfolioItemBean.set

- In the bean entry with id = 'Interceptor_Name', update the value of 'role_allowed' to the role specified in the class which implements 'Authentication' interface. For example, if you are using sample implementation provided with the agent (which is 'SecurentAuthentication') then the value of 'role_allowed' should be 'SECURENT_USER'.

- In the bean entry with id = 'Interceptor_Name', update the value of id from 'Interceptor_Name' to any unique name.
  For example: methodSecurityInterceptor

- In the bean entry with id = 'autoProxyCreator', update the 'Interceptor_Names' to the interceptorName/s (typically these are bean id values for interceptors which are to be secured. And Bean_Names to its associated bean.
  For Example:
  <bean id="portfolioBean"
  class="com.devx.tradingapp.web.PortfolioItemBean"></bean> then update 'Bean_Names' to 'portfolioBean'

9. Add the following code before the return statement in handleRequest() method of the controller which handles a request coming from login page:

* Statement which will create object of the class which implements 'Authentication' interface, say 'auth' based on the user information coming from login page like username,password etc.('SecurentAuthentication' provided with agent requires only username)

* Add the following statements:

    HttpSession httpSession = ((HttpServletRequest) httpServletRequest).getSession();
    //Where httpServletRequest is an object of type HttpServletRequest
    sc.setAuthentication(auth);
    // where 'auth' is an object created in above step
10. If you want to use Sample controller provided with the agent then perform following steps:-
   I. Edit SampleAcegiController.java and modify the contents by referring the inline comments.
   II. Put servlet.jar, acegi-security.jar and acegi_agent.jar files in the classpath and compile it.
   III. Copy the compiled class to the PROJECTHOME\WEB-INF\classes directory.
   IV. Copy SampleAcegi-servlet.xml to PROJECTHOME\WEB-INF directory.
   V. Edit SampleAcegi-servlet.xml and modify the value of 'key' in <prop> tag to the page starting with '/' which should display after login.

11. Restart the server and run the web application.

After authentication when a user tries to access a secured resource (i.e. a jsp page in this case) will receive an HTTP error code 403 which is the official error code to indicate access denied or will get the requested resource based on the permissions specified within 'Securent'.

NOTE: If we want to create these resources in 'Securent' set <record></record> value to "true" in the pep_config.xml file.