Define BA Server Advanced Security

Help and Support Resources

If you do not find answers to your quesions here, please contact your Pentaho technical support representative.

Support-related questions should be submitted through the Pentaho Customer Support Portal at http://support.pentaho.com.

For information about how to purchase support or enable an additional named support contact, please contact your sales representative, or send an email to sales@pentaho.com.

For information about instructor-led training, visit http://www.pentaho.com/training.

Liability Limits and Warranty Disclaimer

Trademarks

Third-Party Open Source Software

For a listing of open source software used by each Pentaho component, navigate to the folder that contains the Pentaho component. Within that folder, locate a folder named licenses. The licenses folder contains HTML.files that list the names of open source software, their licenses, and required attributions.

Contact Us

Global Headquarters Pentaho Corporation
Citadel International, Suite 340
5950 Hazeltine National Drive
Orlando, FL 32822
Phone: +1 407 812-OPEN (6736)
Fax: +1 407 517-4575
http://www.pentaho.com

Sales Inquiries: sales@pentaho.com

Contents

Introduction

This section helps system administrators configure BA Server security, including both its native security features and supported third-party security frameworks.

Prerequisites

Before you begin, you must have *installed* Pentaho software. If you chose to *install the DI server* and its design tool, there is a separate section for *configuring them*.

Expertise

The topics in this section are written for IT administrators who know where data is stored, how to connect to it, details about the computing environment, and how to use the command line to issue commands for Microsoft Windows or Linux.

Tools

We provide a web application, the User Console, which you use to perform most configuration tasks.

Login Credentials

All the tasks that use the User Console, **Administration** page, require that you *log on to the User Console* with the Pentaho administrator user name and password.

Security Overview

This section covers BA Server security and User Console security configuration.

- **Choosing a BA Server Security Method:** Decide if you are going to leave the server set to the default Pentaho security or switch to LDAP, JDBC, hybrid LDAP/JDBC, ApacheDS, or MS Active Directory. Many system admins prefer to use their own authentication method such as LDAP, Active Directory, single sign-on, or custom security tables in an existing RDBMS. There is also information about how to remove all authentication support in rare situations where logins are disfavored.
- **Configuring Security with the User Console:** The console has many options for BA Server configuration through the administration tab, so you will want to restrict access to this section of the console to system administrators only. You might also want to enable https support to ensure secure communication.

The second half of this section covers how to restrict access to data sources using the **Administration** tab in the User Console.

Any user can still use a flat file (CSV, XML, XLS) to establish a data source, and a user that knows JDBC or JNDI connection information to a database can establish a data source on their own at the application server (global) or client tool (local) level. These restrictions only apply to the User Console and its client tools: Analyzer, Dashboard Designer, and Interactive Reporting, as well as any extension, plugin, or service that uses the BA Server for user authentication (such as CDF, or an action sequence). This is a matter of internal security at your organization and is beyond the scope of Pentaho software.

There is no way to restrict access to any content that is not published to the BA Server. So if you create a report in Report Designer but do not publish it to the BA Server, there are no access restrictions available for it. Also, anyone with local user access to the BA Server solution repository (the pentaho-solutions directory) or the database that stores solutions and schedules (jackrabbit and quartz) will be able to see all published BA Server content. The restrictions described below only apply to users accessing the User Console through a Web browser (over a private intranet or the public Internet).

Supported Technologies

The Pentaho BA Platform uses the Spring Security infrastructure, which can work with several common security back ends:

- Flat file
- Active Directory, LDAP, or other directory server
- RDBMS (security tables in an existing database)
- Apache DS
- Microsoft Active Directory (MSAD)

Pentaho's default security back end is an RDBMS, and its Jackrabbit-based security data access object is referred to as **Pentaho**. The security tables and access control lists are installed by default with the BA Server, and can be easily configured through the Administration tab of the User Console. This is a tested, reasonably secure method of managing resource authorization and user authentication, so there should be no reason to change to another security back-end unless you have already deployed one.

Advanced Security

This section discusses six different ways to handle Security other than with default Pentaho security.

Switch to MS Active Directory

1. From User Console **Home** menu, click **Administration**, then select **Authentication** from the left.

   The **Authentication** interface appears. **Local - Use basic Pentaho Authentication** is selected by default.
2. Choose the **External - Use LDAP / Active Directory server** radio button. The **LDAP Server Connection** fields populate with a default URL, user name, and password.
3. Change the **Server URL**, **User Name**, and **Password** as needed.
4. Click **Test Server Connection** to verify the connection to your server and to complete the set up.
5. Click the node to select the **Pentaho System Administrator** user and role to match your configuration. Click **OK**.

   The text box auto-populates with the selected values.
6. For MSAD, choose **Custom Configuration**.
7. For **Users**:
   a) **Search Base** by entering the path where your users are located. Example:

   ```
   CN=Users,DC=MyDomain,DC=com
   ```
   b) **Search Filter** by entering in the attribute that users will login with. Example:

   ```
   (sAMAccountName={0})
   ```
8. For **Roles**:
   a) For **Role Attributes**, enter in the Attribute that is used for roles/groups. Example:

   ```
   CN
   ```
   b) For a **Role Search Filter**, enter in the ObjectClass that defines that these are roles or groups. Example:

   ```
     (&(objectClass=group)(CN=Pentaho*))
   ```
   c) For **Role Search Base**, enter in the path where your roles or groups are located. Example:

   ```
     OU=groups,DC=MyDOmain,DC=com
   ```
   a) Click **Test**.
9. For **Populator**:
   a) For **Group Role Attribute**, enter in the Attribute that is used for groups. Example:

   ```
   CN
   ```
   b) For **Group Search Base**, enter in the path to where your groups are located. Example:

   ```
     OU=groups,DC=MyDOmain,DC=com
   ```
   c) Set the **Group Search Filter** to

   ```
   (member={0})
   ```

   You can set a **Role Prefix** if you need one to filter by.
10. Click **Test**.
11. Click **Save** and restart the BA Server.

The BA Server is now configured to authenticate users against your MSAD server.

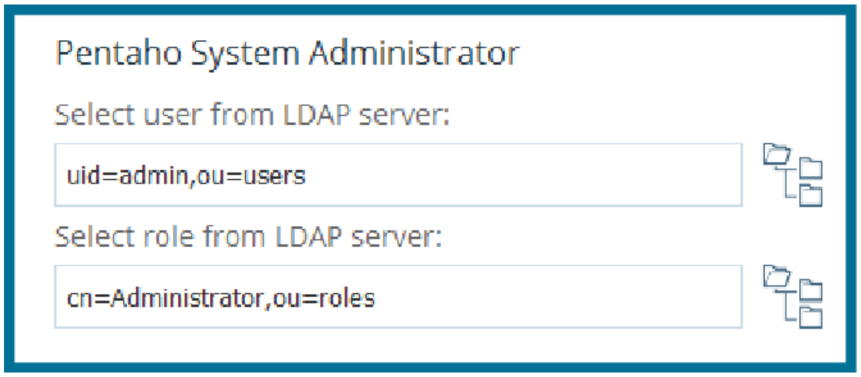
Switch to LDAP

Note: In order to use SSL to connect to your LDAP server, you will need to import the certificate into the JRE's trustStore/keyStore used to by the BA Server (java/lib/security/cacerts).

1. From User Console **Home** menu, click **Administration**, then select **Authentication** from the left.

   The **Authentication** interface appears. **Local - Use basic Pentaho Authentication** is selected by default.
2. Choose the **External - Use LDAP / Active Directory server** radio button. The **LDAP Server Connection** fields populate with a default URL, user name, and password.
3. Change the **Server URL**, **User Name**, and **Password** as needed.
4. Click **Test Server Connection** to verify the connection to your LDAP server and to complete the set up.
5. Click the node to select the **Pentaho System Administrator** user and role to match your LDAP configuration. Click **OK**.

   Note: The Admin user is required for all system-related operations, including the creation of user folders. The Administrator Role is required for mapping a third-party admin role to the Pentaho admin role (Administrator). This is required for all ABS functionality to work properly.



Pentaho System Administrator

Select user from LDAP server:

uid=admin,ou=users

Select role from LDAP server:

cn=Administrator,ou=roles

6. Choose your **LDAP Provider** from the drop-down menu.
7. Configure the LDAP connection as explained in *LDAP Properties*. Click **Test**.

The BA Server is now configured to authenticate users against your LDAP directory server.

Manual MSAD Configuration

The server does not recognize any difference among LDAP-based directory servers, including Active Directory. However, the way that you modify certain LDAP-specific files will probably be different for Microsoft Active Directory (MSAD) than for more traditional LDAP implementations. Below are some tips for specific MSAD-specific configurations that you might find helpful.

Binding

MSAD allows you to uniquely specify users in two ways, in addition to the standard DN. If the standard DN is not working, try one of the two below. Each of the following examples is shown in the context of the **userDn** property of the Spring Security **DefaultSpringSecurityContextSource** bean.

Note: The examples in this section use **DefaultSpringSecurityContextSource**. Be aware that you may need to use the same notation (Kerberos or Windows domain) in all of your DN patterns.

Kerberos notation example for pentahoadmin@mycompany.com:

File: **applicationContext-security-ldap.properties**

```
contextSource.providerUrl=ldap\://mycompany\:389
contextSource.userDn=pentahoadmin@mycompany.com
contextSource.password=omitted
```

Windows domain notation example for MYCOMPANY\pentahoadmin:

File: **applicationContext-security-ldap.properties**

```
contextSource.providerUrl=ldap\://mycompany\:389
contextSource.userDn=MYCOMPANY\pentahoadmin
contextSource.password=omitted
```

Referrals

If more than one Active Directory instance is serving directory information, it may be necessary to enable referral following. This is accomplished by modifying the **DefaultSpringSecurityContextSource** bean.

```
<bean id="contextSource"
 class="org.springframework.security.ldap.DefaultSpringSecurityContextSource">
    <constructor-arg value="${contextSource.providerUrl}"/>
    <property name="userDn" value="${contextSource.userDn}"/>
    <property name="password" value="${contextSource.password}"/>
    <property name="referral" value="follow" />
</bean>
```

User DN Patterns vs. User Searches

In the **LdapAuthenticator** implementations provided by Spring Security (**BindAuthenticator** for instance), you must either specify a **userDnPatterns**, or a **userSearch**, or both. If you're using the Kerberos or Windows domain notation, you should use **userDnPatterns** exclusively in your **LdapAuthenticator**.

Note: The reason for suggesting **userDnPatterns** when using Kerberos or Windows domain notation is that the **LdapUserSearch** implementations do not give the control over the DN that **userDnPatterns** does. (The **LdapUserSearch** implementations try to derive the DN in the standard format, which might not work in Active Directory.)

Note, however, that **LdapUserDetailsService** requires an **LdapUserSearch** for its constructor.

User DN Pattern example:

```
<bean id="authenticator"
class="org.springframework.security.providers.ldap.authenticator.BindAuthenticator">
<constructor-arg>
    <ref local="contextSource"/>
    </constructor-arg>
    <propertyname="userDnPatterns">
    <list>
    <value>{0}@mycompany.com
    </value> <!-- and/or -->
    <value>domain\{0}</value>
    </list>
    </property>
</bean>
```

In user searches, the **sAMAccountName** attribute should be used as the username. The **searchSubtree** property (which influences the **SearchControls**) should most likely be true. Otherwise, it searches the specified base plus one level down.

User Search example:

```
<bean id="userSearch"
class="org.springframework.security.ldap.search.FilterBasedLdapUserSearch">
    <constructor-arg index="0" value="DC=mycompany,DC=com" />
    <constructor-arg index="1">
    <value>(sAMAccountName={0})</value>
    </constructor-arg> <constructor-arg index="2">
    <ref local="contextSource" />
    </constructor-arg>
    <property name="searchSubtree" value="true"/>
</bean>
```

Nested Groups

You can remove nested or transitive groups out of Active Directory. In the LDAP popular group filter, enter the following LDAP filter for MSAD nested groups:

```
(member:1.2.840.113556.1.4.1941:={0})
```

This will search down the whole tree of nested groups.

Manual LDAP Configuration

You must have a working directory server with an established configuration before continuing.

Follow the instructions below to switch from Pentaho default security to LDAP security.

1. Stop the BA Server and User Console.
2. Open the `/pentaho-solutions/system/pentaho-spring-beans.xml` file with a text editor, find the following two adjacent lines, and change the **jackrabbit** segment to **ldap** in each:

   From:

   ```
   <import resource="applicationContext-spring-security-jackrabbit.xml" />
   <import resource="applicationContext-pentaho-security-jackrabbit.xml" />
   ```

   To:

   ```
   <import resource="applicationContext-spring-security-ldap.xml" />
   <import resource="applicationContext-pentaho-security-ldap.xml" />
   ```

3. Save and close the file, then edit the `/pentaho-solutions/system/data-access/settings.xml` file and modify the user and role settings to match your LDAP configuration:

   Replace the **pentahoUsers** and **pentahoAdmins** references in the examples below with the correct role names from your LDAP configuration.

   ```
   <!– roles with data access permissions –>
   <data-access-roles>pentahoAdmins</data-access-roles>
   <!– users with data access permissions –>
   <!–
   <data-access-users></data-access-users>
   –>
   <!– roles with datasource view permissions –>
   <data-access-view-roles>pentahoUsers,pentahoAdmins</data-access-view-roles>
   <!– users with datasource view permissions –>
   <!– <data-access-view-users></data-access-view-users> –>
   <!– default view acls for user or role –>
   <data-access-default-view-acls>31</data-access-default-view-acls>
   ```

4. Save and close the file, then edit the following files in the /pentaho/server/biserver-ee/pentaho-solutions/system/ directory and change all instances of the Admin and Authenticated role values to match the appropriate roles in your LDAP configuration: pentaho.xml applicationContext-spring-security.xml

   - pentaho.xml
   - repository.spring.xml
   - applicationContext-spring-security.xml

5. Start the BA Server and the **User Console**. Log into the **User Console** and click on the **Administration** tab.
6. Click on the plus (**+**) symbol next to **Security**, and select **Authentication**.
7. Under **Authentication**, select the **LDAP / Active Directory Based** button.
8. Configure the LDAP connection as explained in *LDAP Properties*. Click **Test**.

The BA Server is now configured to authenticate users against your directory server.

Implementing Nested Roles in LDAP

It is possible to nest user roles such that one role includes all of the users of another role. Doing this external to the core LDAP structure prevents recursive directory queries to find all parents of a given child role. Follow the directions below to modify the BA Server to support nested roles for LDAP and MSAD authentication types.

1. Stop the BA Server or service.

   ```
   sh /usr/local/pentaho/server/biserver-ee/stop-pentaho.sh
   ```

2. Open the `/pentaho/server/biserver-ee/pentaho-solutions/system/applicationContext-spring-security-ldap.xml` file with a text editor.
3. In the **populator** bean definition, replace **DefaultLdapAuthoritiesPopulator** with **NestedLdapAuthoritiesPopulator**

   ```
   <bean id="populator"
    class="org.pentaho.platform.plugin.services.security.userrole.ldap.
    NestedLdapAuthoritiesPopulator">
   ```

4. Save the file, then edit `/pentaho/server/biserver-ee/pentaho-solutions/system/applicationContext-pentaho-security-ldap.xml`.

   This and the next step are only necessary if the roles that serve as "parents" to nested roles cannot be returned by a traditional all authorities search.
5. Add an **extraRoles** bean to the list of transformers in the **ChainedTransformers** bean, and set properties for each parent role (represented by example_role below).

   ```
   <bean id="allAuthoritiesSearch" class="org.pentaho.platform.plugin.services
   .security.userrole.ldap.search.GenericLdapSearch">
       <!-- omitted -->
       <constructor-arg index="2">
           <bean class="org.apache.commons.collections.functors.ChainedTransformer">
               <constructor-arg index="0">
                   <list>
                       <bean class="org.pentaho.platform.plugin.services.security.
   userrole.ldap.transform.SearchResultToAttrValueList">
                           <!-- omitted -->
                       </bean>
                       <bean
    class="org.pentaho.platform.plugin.services.security.userrole.
   ldap.transform.ExtraRoles">
                           <property name="extraRoles">
                               <set>
                                   <value>example_role</value>
                               </set>
                           </property>
                       </bean>
                       <bean class="org.pentaho.platform.plugin.services.security.
   userrole.ldap.transform.StringToGrantedAuthority">
                           <!-- omitted -->
                       </bean>
                   </list>
               </constructor-arg>
           </bean>
       </constructor-arg>
   </bean>
   ```

6. Save the file, close your text editor, and start the BA Server.

   ```
   sh /usr/local/pentaho/server/biserver-ee/start-pentaho.sh
   ```

The BA Server can now efficiently handle nested roles with LDAP or Active Directory authentication.

LDAP Properties

You can configure LDAP values by editing the `/pentaho-solutions/system/applicationContext-security-ldap.properties` file in your BA Server or DI Server directory, or through the User Console for the BA Server (the LDAP options in the console apply only to the BA Server, not the DI Server).

Connection Information (Context)

These entries define connections involving LDAP users (typically administrators) that can execute directory searches.

LDAP Property	Purpose	Example
contextSource.providerUrl	LDAP connection URL	contextSource.providerUrl=ldap://holly:389/DC=Valyant,DC=local
contextSource.userDn	Distinguished name of a user with read access to directory	contextSource.userDn=CN=Administrator,CN=Users,DC=Valyant,DC=local
contextSource.password	Password for the specified user	contextSource.password=secret

Users

These options control how the LDAP server is searched for usernames that are entered in the Pentaho login dialog box.

Note: The **{0}** token will be replaced by the username from the login dialogue.

Note: The example above defines **DC=Valyant,DC=local** in **contextSource.providerURL**. Given that definition, you would not need to repeat that in **userSearch.searchBase** below because it will be appended automatically to the defined value here.

LDAP Property	Purpose	Example
userSearch.searchBase	Base (by username) for user searches	userSearch.searchBase=CN=Users
userSearch.searchFilter	Filter (by username) for user searches. The attribute you specify here must contain the value that you want your users to log into Pentaho with. Active Directory usernames are represented by**sAMAccountName** ; full names are represented by **displayName**.	userSearch.searchFilter=(sAMAccountName={0})

Populator

The populator matches fully distinguished user names from **userSearch** to distinguished role names for roles those users belong to.

Note: The {0} token will be replaced with the user DN found during a user search; the {1} token is replaced with the username entered in the login screen.

LDAP Property	Purpose	Example
populator.convertToUpperCase	Indicates whether or not retrieved role names are converted to uppercase	populator.convertToUpperCase=false
populator.groupRoleAttribute	The attribute to get role names from	populator.groupRoleAttribute=cn
populator.groupSearchBase	Base (by user DN or username) for role searches.	populator.groupSearchBase=ou=Pentaho

LDAP Property	Purpose	Example
populator.groupSearchFilter	The special nested group filter for Active Directory is shown in the example; this will not work with non-MSAD directory servers.	populator.groupSearchFilter=(memberof:1.2.840.113556.1.4.5941:=({0}))
populator.rolePrefix	A prefix to add to the beginning of the role name found in the group role attribute; the value can be an empty string.	populator.rolePrefix=
populator.searchSubtree	Indicates whether or not the search must include the current object and all children. If set to **false**, the search must include the current object only.	populator.searchSubtree=true

All Authorities Search

These entries populate the BA Server Access Control List (ACL) roles. These should be similar or identical to the Populator entries.

LDAP Property	Purpose	Example
allAuthoritiesSearch.roleAttribute	The attribute used for role values	allAuthoritiesSearch.roleAttribute=cn
allAuthoritiesSearch.searchBase	Base for "all roles" searches	allAuthoritiesSearch.searchBase=ou=Pentaho
allAuthoritiesSearch.searchFilter	Filter for "all roles" searches. Active Directory requires that the **objectClass** value be set to **group**.	allAuthoritiesSearch.searchFilter=(objectClass=group)

All user name search

These entries populate the BA Server ACL users.

LDAP Property	Purpose	Example
allUsernamesSearch.username Attribute	The attribute used for user values	allUsernamesSearch.username Attribute= sAMAccountName
allUsernamesSearch.searchBase	Base for "all users" searches	allUsernamesSearch.searchBase=CN=users
allUsernamesSearch.searchFilter	Filter for "all users" searches	allUsernamesSearch.searchFilter=objectClass=person

Manual JDBC Connection Configuration

You must have existing security tables in a relational database in order to proceed with this task.

Follow the instructions below to switch from Pentaho default security to JDBC security, which will allow you to use your own security tables.

Note: If you are using the BA Server and choose to switch to a JDBC security shared object, you will no longer be able to use the role and user administration settings in the Administration portion of the User Console.

1. Stop the BA Server by running the **stop-pentaho** script.
2. Open `/pentaho-solutions/system/security.properties` with a text editor.
3. Change the value of the provide property to `jdbc`.

4. Open `/pentaho-solutions/system/applicationContext-pentaho-security-jdbc.xml` with a text editor, find this line:

```
<import resource="applicationContext-spring-security-jdbc.xml" />
<import resource="applicationContext-pentaho-security-jdbc.xml" />
     <entry key="Admin" value="Administrator"/>
</util:map>
```

5. By default, the role `Admin` is mapped to `Administrator`. Change `Admin` to the appropriate administrator role in your JDBC authentication database.
6. Verify that the SQL statements are the correct syntax for your database, and that they reference the correct tables, roles, and actions.
7. Save the file and close the editor.
8. Start the server by running the **start-pentaho** script.

The server is configured to authenticate users against the specified database.

Manual LDAP/JDBC Hybrid Configuration

You must have a working directory server with an established configuration, and a database containing your user roles before continuing.

It is possible to use a directory server for user authentication and a JDBC security table for role definitions. This is common in situations where LDAP roles cannot be redefined for BA Server use. Follow the below instructions to switch the BA Server's authentication backend from the Pentaho data access object to an LDAP/JDBC hybrid.

Note: Replace the **pentahoAdmins** and **pentahoUsers** references in the examples below with the appropriate roles from your LDAP configuration.

1. Stop the BA Server and User Console.
2. Open `/pentaho-solutions/system/security.properties.xml` with a text editor.
3. Change the value of the property provider to `ldap`.
4. Open the `/pentaho-solutions/system/pentahoObjects.spring.xml` with a text editor.
5. Find this code block and change the providerName to `jdbc`.

```
  <pen: bean id="UserDetailsService"
class ="org.springframework.security.userdetails.UserDetailsService">
    <pen:attributes>
       <pen:attr key="providerName" value="jackrabbit"/>
    </pen:attributes>

   <pen:publish as-type="INTERFACES">
     <pen:attributes>
    <pen:attr key="priority" value="50"/>
      </pen:attributes>
    </pen:publish>

    </pen:bean>
```

6. Edit the `/pentaho-solutions/system/applicationContext-pentaho-security-jdbc.xml` file and add the following two bean definitions, changing the connection and JDBC details to match your security database.

```

<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
     <property name="driverClassName" value="org.hsqldb:hsql://localhost:9002/
userdb" />
     <property name="url" value="jdbc:hsqldb:hsql://localhost:9002/userdb" />
     <property name="username" value="sa" />
     <property name="password" value="" />
</bean>
<bean id="userDetailsService"
class="org.springframework.security.userdetails.jdbc.JdbcDaoImpl">
     <property name="dataSource">
```

```
        <ref local="dataSource" />
     </property>
     <property name="authoritiesByUsernameQuery">
        <value> <![CDATA[SELECT username, authority FROM
        granted_authorities WHERE username = ?}}></value>
     </property>
     <property name="usersByUsernameQuery">
        <value> <![CDATA[SELECT username,
        password, enabled FROM users WHERE username = ?]]>
        </value>
     </property>
</bean>
```

7. Edit the `/pentaho-solutions/system/data-access/settings.xml` file and modify the user and role settings to match your LDAP configuration:

```
<!– roles with data access permissions –>
<data-access-roles>pentahoAdmins</data-access-roles>
<!– users with data access permissions –>
<!–
<data-access-users></data-access-users>
–>
<!– roles with datasource view permissions –>
<data-access-view-roles>pentahoUsers,pentahoAdmins</data-access-view-roles>
<!– users with datasource view permissions –>
<!– <data-access-view-users></data-access-view-users> –>
<!– default view acls for user or role –>
<data-access-default-view-acls>31</data-access-default-view-acls>
```

8. Save and close the file, then open `/pentaho-solutions/system/applicationContext-pentaho-security-jdbc.xml`. Find this code block and change `Admin` to an appropriate administrator role in your JDBC authentication database.

```
<!-- map ldap role to pentaho security role -->
<util:map id="jdbcRoleMap">
   <entry key="Admin" value="Administrator"/>
</util:map>
```

9. Save and close the file, then open `/pentaho-solutions/system/applicationContext-pentaho-security-jdbc.xml`.

```
<!-- map ldap role to pentaho security role -->
util:map id="jdbcRoleMap">
   <entry key="Admin" value="Administrator"/>
   </util:map>
```

10. Delete the `/tomcat/work/` and `/tomcat/temp/` directories.
11. Start the BA Server and User Console.
12. Log into the User Console.
13. Configure the Pentaho LDAP connection as explained in *LDAP Properties* on page 12.

The BA Server is configured to authenticate users against your directory server.

Implement Single Sign-On

This section contains instructions for configuring the BA Server to work with a single sign-on (SSO) framework. At this time, only Central Authentication Service (CAS) and Integrated Windows Authentication (IWA) are supported. Refer only to the instructions below that apply to the framework you are using.

Switching to Central Authentication Service (CAS)

Pentaho can integrate with Central Authentication Service (CAS). Pentaho only supports CAS integration from a manual build and deployment of a Pentaho WAR or EAR file. You must have a CAS server installed and running before you continue.

1. If the BA Server or User Console servers are currently running, stop them with the **stop-pentaho** script.

2. Download **cas-client-core-3.1.5.jar** and copy it to `biserver-ee/tomcat/webapps/pentaho/web-inf/lib` folder
3. Download **spring-security-cas-client-2.0.5.RELEASE.jar** and copy it to `biserver-ee/tomcat/webapps/pentaho/web-inf/lib` folder.
4. Update **pentaho-spring-beans.xml**.
   a) Add **<import resource="applicationContext-spring-security-cas.xml" />** to the list of imports in the .xml.
5. Update **applicationContext-spring-security-cas.xml**.
   a) Change all the references of this URL **https://localhost:8443/cas** to your working CAS server URL.
   b) If you are using Pentaho with SSL, then update references to this url **http://localhost:8080/pentaho**.
6. Add the following in their respective sections to the **web.xml**.

```
<servlet>
  <servlet-name>casFailed</servlet-name>
  <jsp-file>/jsp/casFailed.jsp</jsp-file>
</servlet>
```

```
<servlet-mapping>
  <servlet-name>casFailed</servlet-name>
  <url-pattern>/public/casFailed</url-pattern>
</servlet-mapping>
```

```
<!--
<listener>
    <listener-class>org.jasig.cas.client.session.SingleSignOutHttpSessionListener</
listener-class>
</listener>
-->
```

7. Start the BA Server and User Console servers with the **start-pentaho** script.

The BA Server is now configured to authenticate users against your central authentication server.

CAS Property Reference

CAS Properties

These properties mostly refer to CAS services:

Property	Description	Example
cas.authn.provider	**Required.** Security back-end that CAS should use. Valid values are memory, jdbc, or ldap	ldap
cas.login.url	**Required.** CAS login URL.	${cas.base.url}/login
cas.ticket.validator.url	**Required.** CAS ticket validator URL.	${cas.base.url}
cas.logout.url	**Required.** CAS logout URL. A service.logout.url will be appended to this URL.	${cas.base.url}/logout?url=
cas.base.url	URL under which all CAS services reside.	https://localhost:8443/cas

Pentaho Properties

These are service URLs that serve as callbacks from the CAS server into the BI Platform:

Property	Description	Example
pentaho.service.url	**Required.** Processes CAS callback.	${_pentaho.service.base.url}/j_spring_cas_security_check

Property	Description	Example
pentaho.service.logout.url	**Required.** URL to go to after CAS logout.	${_pentaho.service.base.url}/Home
pentaho.service.solutions.system.dir	Path to pentaho-solutions/system.	/usr/local/pentaho/server/biserver-ee/pentaho-solutions/system/
pentaho.service.lib.dir	Path to webapp lib directory.	/usr/local/tomcat/common/lib/
pentaho.service.web.xml	Path (including filename) of webapp's web.xml.	/usr/local/tomcat/conf/web.xml
pentaho.service.appctx.cas.xml	Path (including filename) of new applicationContext-spring-security-cas.xml.	/usr/local/pentaho/server/biserver-ee/pentaho-solutions/system/applicationContext-spring-security-cas.xml
pentaho.service.jsp.dir	Path to directory containing webapp's JSPs.	/usr/local/tomcat/webapps/pentaho/jsp/
pentaho.service.spring.beans.xml	Path (including filename) of pentaho-spring-beans.xml.	/usr/local/pentaho/server/biserver-ee/pentaho-solutions/system/pentaho-sping-beans.xml
pentaho.service.base.url	Service base URL.	http://localhost:8080/pentaho
pentaho.service.pentaho.war.dir	Webapp exploded WAR directory.	/usr/local/tomcat/webapps/pentaho/
pentaho.service.webinf.dir	Path to webapp's WEB-INF directory.	/usr/local/tomcat/webapps/pentaho/WEB-INF/

Switching to Integrated Windows Authentication (IWA)

You must download this *patch JAR* before you switch to Integrated Windows Authentication.

This procedure requires Microsoft Windows Server 2008 R2, IIS 7.5, and Internet Explorer. If you are using different versions of any of this software, you may adjust the instructions to fit your needs.

Additionally, you will need to ensure that the following components of IIS are installed before continuing:

- Windows Authentication
- ISAPI Extensions
- ISAPI Filters
- JK 1.2 Connector (isapi_redirect.dll)

Follow these instructions to switch to Integrated Windows Authentication in the BA Server.

1. Stop the BA Server, DI Server, and User Console processes.
2. Copy the downloaded patch JAR to the `/WEB-INF/lib/` directory inside of the deployed Pentaho WAR.

   For most deployments, this will be `/pentaho/server/biserver-ee/tomcat/webapps/pentaho/WEB-INF/lib/`
3. In your IIS configuration, disable anonymous authentication and enable Windows authentication for the site you are serving.
4. Edit the `/WEB-INF/web.xml` file inside of the deployed Pentaho WAR, and change the value of **fully-qualified-server-url** to the URL served by IIS, then save and close the file.
5. Edit the `/tomcat/conf/server.xml` file and set **tomcatAuthentication** to **false** in the **Connector** element for the connector with the **AJP** protocol.

   Note: If this is not already defined, then add it; the example below can be directly pasted into the file.

   ```
   tomcatAuthentication="false"
   ```

6. Save and close the file, then edit `/pentaho-solutions/system/applicationContext-spring-security.xml`. Comment out this code block

```
<![CDATA[CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON PATTERN_TYPE_APACHE_ANT
/**=securityContextHolderAwareRequestFilter,httpSessionContextIntegrationFilter,
httpSessionReuseDetectionFilter,logoutFilter,preAuthenticatedProcessingFilter,
authenticationProcessingFilter,basicProcessingFilter,requestParameterProcessingFilter,
anonymousProcessingFilter,pentahoSecurityStartupFilter,exceptionTranslationFilter,
filterInvocationInterceptor]]>
```

7. Copy and paste this code block immediately after the block you just commented out

```
<![CDATA[CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON PATTERN_TYPE_APACHE_ANT
/**=httpSessionContextIntegrationFilter,httpSessionReuseDetectionFilter,
logoutFilter,preAuthenticatedProcessingFilter,authenticationProcessingFilter,
basicProcessingFilter,requestParameterProcessingFilter,anonymousProcessingFilter,
securityContextHolderAwareRequestFilter,pentahoSecurityStartupFilter,
exceptionTranslationFilter,filterInvocationInterceptor]]>
```

8. Find the **authenticationManager** providers list and add this line to the beginning of it:

```
<ref bean="preAuthAuthenticationProvider" />
```

9. Replace the **authenticationProcessingFilterEntryPoint** bean definition with the following:

```
<bean id="preAuthenticatedProcessingFilterEntryPoint"
    class="org.springframework.security.ui.preauth.
      PreAuthenticatedProcessingFilterEntryPoint" />
```

10. Find the **exceptionTranslationFilter** bean and replace its **authenticationEntryPoint** ref with:

```
<ref local="preAuthenticatedProcessingFilterEntryPoint" />
```

11. Ensure that you have configured Active Directory integration properly. Refer to your Active Directory documentation and *Manual MSAD Configuration* on page 8 for more information.
12. Save and close the server.xml file.
13. Configure Internet Explorer such that your IIS server is in the **local intranet** security zone.
14. Start the BA Server.
15. Access the BA Server through Internet Explorer and ensure that it automatically logs in with the local user account.

Your system should now be configured to sign into the BA Server using local user account credentials.

Add Web Resource Authentication

To configure Web resource authentication in the BA Server to correspond with your user roles, follow the below instructions.

Note: These instructions are valid across all security DAOs.

1. Ensure that the BA Server is not currently running; if it is, run the **stop-pentaho** script.
2. Open a terminal or command prompt window and navigate to the `.../pentaho-solutions/system/` directory.
3. Edit the **applicationContext-spring-security.xml** file with a text editor.
4. Find and examine the following property: `<property name="objectDefinitionSource">`
5. Modify the regex patterns to include your roles.

   The **objectDefinitionSource** property associates URL patterns with roles. **RoleVoter** specifies that if any role on the right hand side of the equals sign is granted to the user, the user may view any page that matches that URL pattern. The default roles in this file are not required; you can replace, delete, or change them in any way that suits you.

You should now have coarse-grained permissions established for user roles.

Restrict or Share Files and Folders

Access to files or folders can be refined using the Pentaho User Console. Each file or folder can either use the default permissions or you can tailor them for specific users and roles.

Prior to performing this task, you need to have determined whether you are going to use the default Pentaho roles, or created specific users and roles. You must also have successfully set up your security back end. Once you establish roles, you can share or restrict files and folders by role-type from the administration view within the User Console.

1. Log into the **User Console** using the administrator role.
2. From the **Browse Files** page, choose the folder you want to set permissions on from the **Folders** pane. If you want to set permissions on a specific file within that folder, click to highlight the file in the center **Files** pane.
3. Click **Properties** in the **Actions** pane on the right.

   The **Properties** window appears.
4. On the **Share** tab, highlight the **Role** that you want to set permissions for, then clear the check box next to **Inherits folder permissions**.

   The **Permissions for [Role]** field becomes accessible.
5. Select the permissions for that role using the check boxes and click **OK**.

The permissions are set for that file or folder and are associated with the selected role.

Secure the User Console and BA Server

This section contains instructions and guidance for enhancing the security of the BA Server and User Console on an application server level via Secure Sockets Layer (SSL). SSL provides verification of server identity and encryption of data between clients and the BA Server.

Configure SSL (HTTPS) in the Pentaho User Console and BA Server

By default, the BA Server and Pentaho User Console are configured to communicate over HTTP. To switch to HTTPS, follow the instructions below that apply to your scenario.

Enable SSL in the BA Server With a Certificate Authority

If you already have an SSL certificate through a certificate authority such as Thawte or Verisign, all you have to do to use it with the Pentaho BA Server and User Console is configure your application server to use it. Apache provides documentation for configuring Tomcat for CA-signed certificates: *http://tomcat.apache.org/tomcat-6.0-doc/ssl-howto.html*. Just follow those procedures, and skip the sections below that deal with self-signed SSL certificates.

After the application server is configured to use your certificate, you must modify the base URL tokens for both the BA Server and the User Console. Make sure you follow the directions for *changing the BA Server Base URL*; without executing those changes, your server will not work over HTTPS.

Enable SSL in the BA Server With a Self-Signed Certificate

This process explains how to enable SSL in the BA Server with a self-signed certificate. These steps don't show how to generate a self-signed certificate, or how to configure Tomcat to use it. For more information on SSL certificates in Tomcat, consult *the Tomcat documentation*, beginning with the *Quick Start* section.

Trusting a Self-Signed Certificate

The procedure below assumes that an SSL certificate is generated and Tomcat is configured to use it.

The instructions below explain how to complete the trust relationship between the BA Server (when it is configured for SSL) and the User Console.

1. Change to the home directory of the user account that starts the BA Server and Pentaho User Console processes or services.

   ```
   cd ~
   ```

   Using the default settings suggested by Pentaho, this will be `/home/pentaho/`.
2. Execute the following command, changing the storepass (**pass** in the example) and keypass (**pass2** in the example) accordingly:

   ```
   keytool -export -alias tomcat -file tomcat.cer -storepass pass -keypass pass2 -
   keystore .keystore
   ```
3. Change to the `$PENTAHO_JAVA_HOME/jre/lib/security/` directory.

   ```
   cd $PENTAHO_JAVA_HOME/jre/lib/security/
   ```

   The **PENTAHO_JAVA_HOME** variable was established during your production installation procedure. If you are on Windows, environment variables are surrounded by percent signs, as in: `cd %PENTAHO_JAVA_HOME%\jre\lib\security\`. If you get an error about this path not being valid, then use **JAVA_HOME** instead of **PENTAHO_JAVA_HOME**.
4. Execute the following command, changing the alias (**tomcat** in the example), the file path to the certificate (the current user's home directory in the example), and the storepass (**pass** in the example) accordingly:

   ```
   keytool -import -alias tomcat -file ~/tomcat.cer -keystore cacerts -storepass pass
   ```

   Note: If the path to your certificate involves spaces, you must either escape the spaces (on Linux, Unix, and OS X), or put double quotes around the path (on Windows) in order for the command to work properly.

5. Execute the following command and make note of the MD5 sum for the **tomcat** entry:

   ```
   keytool -list -keystore cacerts
   ```

6. Change back to the home directory of the user account that starts the BA Server and User Console, and run this command:

   ```
   keytool -list -keystore .keystore
   ```

7. Compare the **tomcat** entry's MD5 sum to the one you generated previously and ensure that they match.

   If these sums do not match, you've made a mistake someplace in the certificate trust process. Go through the steps again and ensure that you're working with the right user accounts and directories.

The BA Server is now configured to allow access via SSL.

Change the BA Server Fully Qualified URL

If you switch from HTTP to HTTPS, you must also change the BA Server's tokenized fully qualified URL value to accommodate for the new port number. Follow the directions below to change the fully qualified URL.

1. Stop the BA Server if it is currently running.

   ```
   /etc/init.d/pentaho stop
   ```

2. Navigate to the `WEB-INF` directory inside of the pentaho.war.
   cd /home/pentaho/pentaho/server/biserver-ee/tomcat/webapps/pentaho/WEB-INF/
3. Edit the **web.xml** file.

   ```
    vim ./web.xml
   ```

4. Locate the element and replace its value to match your new SSL-enabled port number.

   ```
   <context-param>
       <param-name>base-url</param-name>
       <param-value>http://localhost:18443/pentaho/</param-value>
   </context-param>
   ```

   Modify the port number in the example to match your configuration.
5. Save and close the file.
6. Start the BA Server and ensure that it is available through HTTPS on the specified port.

   ```
   /etc/init.d/pentaho start
   ```

The BA Server is now configured to communicate on an SSL-aware port.

Use the Apache Web Server (httpd) For Socket Handling

Tomcat's socket handling abilities are not quite as robust as Apache httpd's are, especially when it comes to system error handling because Tomcat performs all its socket handling through the Java VM. Since Java is designed to be cross-platform, it lacks some system-specific optimizations; socket optimization is one such deficiency. In situations where the BA Server is hit with a large number of dropped connections, invalid packets, or invalid requests from invalid IP addresses, httpd would do a much better job of dropping these error conditions than Tomcat would. Therefore, you can improve BA Server security by fronting Tomcat with httpd.

A side-effect of this configuration is increased performance when delivering static content from the BA Server. For this reason, the same procedure below is covered in the section called *Optimize BA Server Performance*. If you have already followed the Apache httpd procedure in that guide, there is no need to perform it again with the instructions below.

Using Apache httpd With SSL For Delivering Static Content

You can use the Apache httpd Web server to handle delivery of static content and facilitation of socket connections, neither of which is done efficiently through Tomcat alone, especially under heavy traffic or when accepting connections from the Internet.

1. Install Apache 2.2.x -- with SSL support -- through your operating system's preferred installation method. For most people, this will be through a package manager. It's also perfectly valid to download and install the reference implementation from *http://www.apache.org*.

   It is possible to use Apache 1.3, but you will have to modify the instructions on your own from this point onward.
2. If it has started as a consequence of installing, stop the Apache server or service.
3. Retrieve or create your SSL keys.

   If you do not know how to generate self-signed certificates, refer to the OpenSSL documentation. Most production environments have SSL certificates issued by a certificate authority such as Thawte or Verisign.
4. Check to see if you already have the Tomcat Connector installed on your system. You can generally accomplish this by searching your filesystem for **mod_jk**, though you can also search your **http.conf** file for **mod_jk**. If it is present, then you only need to be concerned with the Apache httpd configuration details and can skip this step. If it is not there, then the Tomcat Connector module needs to be installed. If you are using Linux or BSD, use your package manager or the Ports system to install **mod_jk**. For all other platforms, visit the *http://www.apache.org/dist/tomcat/tomcat-connectors/jk/binaries/*, then click on the directory for your operating system. The module will be either an **.so** (for Linux, BSD, OS X, and Solaris) or **.dll** (for Windows) file. Save it to your Apache modules directory, which is generally `C:\Program Files\Apache Group\Apache2\modules\` on Windows, and `/usr/lib/apache2/modules/` on Unix-like operating systems, though this can vary depending on your Apache configuration.
5. Edit your **httpd.conf** file with a text editor and add the following text to the end of the file, modifying the paths and filenames as instructed in the comments:

   Note: Some operating systems use modular httpd configuration files and have unique methods of including each separate piece into one central file. Ensure that you are not accidentally interfering with an auto-generated mod_jk configuration before you continue. In many cases, some of the configuration example below will have to be cut out (such as the **LoadModule** statement). In some cases (such as with Ubuntu Linux), httpd.conf may be completely empty, in which case you should still be able to add the below lines to it. Replace **example.com** with your hostname or domain name.

```
# Load mod_jk module
# Update this path to match your mod_jk location; Windows users should change
 the .so to .dll
LoadModule    jk_module  /usr/lib/apache/modules/mod_jk.so
# Where to find workers.properties
# Update this path to match your conf directory location
JkWorkersFile /etc/httpd/conf/workers.properties
# Should mod_jk send SSL information to Tomcat (default is On)
JkExtractSSL On
# What is the indicator for SSL (default is HTTPS)
JkHTTPSIndicator HTTPS
# What is the indicator for SSL session (default is SSL_SESSION_ID)
JkSESSIONIndicator SSL_SESSION_ID
# What is the indicator for client SSL cipher suit (default is SSL_CIPHER)
JkCIPHERIndicator SSL_CIPHER
# What is the indicator for the client SSL certificated (default is SSL_CLIENT_CERT)
JkCERTSIndicator SSL_CLIENT_CERT
# Where to put jk shared memory
# Update this path to match your local state directory or logs directory
JkShmFile     /var/log/httpd/mod_jk.shm
# Where to put jk logs
# Update this path to match your logs directory location (put mod_jk.log next to
 access_log)
JkLogFile     /var/log/httpd/mod_jk.log
# Set the jk log level [debug/error/info]
JkLogLevel    info
# Select the timestamp log format
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
# Send everything for context /examples to worker named worker1 (ajp13)
# JkOptions indicates to send SSK KEY SIZE
JkOptions +ForwardKeySize +ForwardURICompat -ForwardDirectories
# JkRequestLogFormat
JkRequestLogFormat "%w %V %T"
# Mount your applications
JkMount /pentaho/* tomcat_pentaho
# Add shared memory.
# This directive is present with 1.2.10 and
# later versions of mod_jk, and is needed for
# for load balancing to work properly
JkShmFile logs/jk.shm
<VirtualHost example.com
ServerName example.com
JkMount /pentaho default
JkMount /pentaho/* default
JkMount /sw-style default
JkMount /sw-style/* default
JkMount /pentaho-style default
JkMount /pentaho-style/* default
</VirtualHost>
```

6. In your Apache configuration, ensure that SSL is enabled by uncommenting or adding and modifying the following lines:

```
LoadModule ssl_module modules/mod_ssl.so
Include conf/extra/httpd-ssl.conf
```

7. Save and close the file, then edit `/conf/extra/httpd-ssl.conf` and properly define the locations for your SSL certificate and key:

```
SSLCertificateFile "conf/ssl/mycert.cert"
SSLCertificateKeyFile "conf/ssl/mycert.key"
```

8. Ensure that your SSL engine options contain these entries:

```
SSLOptions +StdEnvVars +ExportCertData
```

9. Add these lines to the end of the **VirtualHost** section:

```
JkMount /pentaho default
JkMount /pentaho/* default
JkMount /sw-style default
JkMount /sw-style/* default
JkMount /pentaho-style default
JkMount /pentaho-style/* default
```

10. Save and close the file, then create a **workers.properties** file in your Apache conf directory. If it already exists, merge it with the example configuration in the next step.
11. Copy the following text into the new **workers.properties** file, changing the location of Tomcat and Java, and the port numbers and IP addresses to match your configuration:

Note: Remove the **workers.tomcat_home** setting if you are using JBoss.

```
workers.tomcat_home=/home/pentaho/pentaho/server/biserver-ee/tomcat/
workers.java_home=/home/pentaho/pentaho/java/
worker.list=tomcat_pentaho
worker.tomcat_pentaho.type=ajp13
```

Apache httpd is now configured to securely and efficiently handle static content for Tomcat. You should now start Tomcat and httpd, then navigate to your domain name or hostname and verify that you can access the Pentaho Web application.

How to Change the Administrator Role

The default administrator role in the BA Server is **Admin**. If you need to give this privilege level to a different role name, follow these instructions.

Note: Role names are case sensitive, so take special care when typing in the new role name.

1. Open the `/pentaho/server/biserver-ee/pentaho-solutions/system/pentaho.xml` file with a text editor.

2. Find the **<acl-voter>** element, and replace its **<admin-role>** property with the new administrator role (NewAdmin in the examples in this procedure).

   ```
   <admin-role>NewAdmin</admin-role>
   ```

3. Find the **<acl-publisher>** element, and appropriately replace all instances of **Admin** in the properties inside of the **<default-acls>** and **<overrides>** elements.

   ```
   <acl-entry role="NewAdmin" acl="ADMIN_ALL" />
   ```

4. Save the file, then open `applicationContext-spring-security.xml`
5. Find the **filterInvocationInterceptor** bean, and modify its **objectDefinitionSource** property accordingly.

   You may need to consult the Spring Security documentation to complete this step. The appropriate documentation is at *http://static.springsource.org/spring-security/site/reference.html*

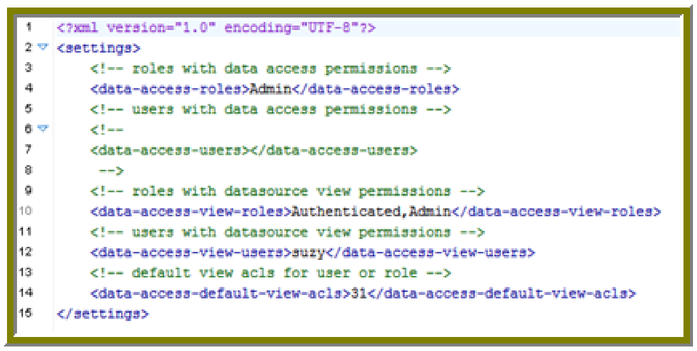
   ```
   <property name="objectDefinitionSource">
       <value>
           <![CDATA[
           CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
           ...
           \A/admin.*\Z=NewAdmin
           ...
           ]]>
       </value>
   </property>
   ```

You have successfully changed the administrator role.

Assign Data Source Permissions for the User Console

By default, authenticated users have view-only permissions to data sources in the User Console. Users with administrative permissions can create, delete, and view data sources. If you want to fine tune permissions associated with data sources, you must edit the appropriate **settings.xml** file. Follow the instructions below to edit the file.

1. Go to `...\biserver-ee\pentaho-solutions\system\data-access` and open **settings.xml**.
2. Edit the settings.xml file as needed. The default values are shown in the sample below. You can assign permissions by individual user or by user role. If you are using LDAP, you can define the correct ACLs value for view permissions; the default value is "31."

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <settings>
       <!-- roles with data access permissions -->
       <data-access-roles>Admin</data-access-roles>
       <!-- users with data access permissions -->
       <!--
       <data-access-users></data-access-users>
        -->
       <!-- roles with datasource view permissions -->
       <data-access-view-roles>Authenticated,Admin</data-access-view-roles>
       <!-- users with datasource view permissions -->
       <data-access-view-users>suzy</data-access-view-users>
       <!-- default view acls for user or role -->
       <data-access-default-view-acls>31</data-access-default-view-acls>
   </settings>
   ```

3. Save your changes to the **settings.xml** file.
4. Restart the User Console.

Secure SQL Access for Dashboards

The Dashboard Designer has an SQL filter that allows greater control over a database query. By default, this feature is restricted to administrative users. To change these settings, follow the instructions below:

1. Ensure that the BA Server is not currently running; if it is, run the **stop-pentaho** script.
2. Open the `/pentaho-solutions/system/dashboards/settings.xml` file with a text editor.

3. Locate the following line and modify it accordingly:

```
<!-- roles with sql execute permissions -->
<sql-execute-roles>Administrator</sql-execute-roles>
```

Note: Values are separated by commas, with no spaces between roles.

4. Locate the following line and modify it accordingly:

```
<!-- users with sql execute permissions -->
<sql-execute-users>Administrator</sql-execute-users>
```

Note: Values are separated by commas, with no spaces between user names.

5. Save and close the text editor.
6. Restart the BA Server with the **start-pentaho** script.

The SQL filter function is now available in Dashboard Designer to the users and roles you specified.

Remove Security by Allowing Anonymous Access

You cannot remove the security mechanisms built into the BA Server, but you can bypass them by giving all permissions to anonymous users. The BA Server automatically assigns all unauthenticated users to the **anonymousUser** account and the **Anonymous** role. The procedure below will grant full BA Server access to the **Anonymous** role and never require a login.

1. Stop the BA Server.

   ```
   sudo /etc/init.d/pentaho stop
   ```

2. Open the `/pentaho/server/biserver-ee/pentaho-solutions/system/applicationContext-spring-security.xml` file with a text editor and ensure that a default anonymous role is defined.

   You may have changed this role, or it may not be properly defined for some other reason. Match your bean definition and property value to the example below. The username does not matter in this particular bean; only the role name.

   ```
   <bean id="anonymousProcessingFilter"
    class="org.springframework.security.providers.anonymous.AnonymousProcessingFilter">
   <!-- omitted -->
       <property name="userAttribute" value="anonymousUser,Anonymous" />
   </bean>
   ```

3. Now find the **filterSecurityInterceptor** bean in the same file, and the **objectDefinitionSource** property inside of it, and match its contents to the example below:

   This step allows Pentaho client tools to publish to the BA Server without having to supply a username and password.

   ```
   <bean id="filterInvocationInterceptor"
    class="org.springframework.security.intercept.web.FilterSecurityInterceptor">
       <property name="authenticationManager">
           <ref local="authenticationManager" />
       </property>
       <property name="accessDecisionManager">
           <ref local="httpRequestAccessDecisionManager" />
       </property>
       <property name="objectDefinitionSource">
           <value>
               <![CDATA[ CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON\A/.*
   \Z=Anonymous,Authenticated ]]> </value>
       </property>
   </bean>
   ```

4. Save the file, then open `pentahoObjects-spring.xml` in the same directory.
5. Change the **IAclVoter** class to **PentahoAllowAnonymousAclVoter**

   ```
   <beans>
   <!-- omitted -->
       <bean id="IAclVoter"
    class="org.pentaho.platform.engine.security.acls.voter.PentahoAllowAnonymousAclVoter"
    scope="singleton" />
   <!-- omitted -->
   </beans>
   ```

6. Save the file, then open `pentaho.xml` in the same directory.
7. In the <anonymous-authentication> part of the **<pentaho-system>** section, define the anonymous user and role.

   This is the same user and role you will use when assigning ACLs in the next step.

   ```
   <pentaho-system>
   <!-- omitted -->
       <anonymous-authentication>
           <anonymous-user>anonymousUser</anonymous-user>
           <anonymous-role>Anonymous</anonymous-role>
       </anonymous-authentication> <!-- omitted -->
   </pentaho-system>
   ```

8. Using the same anonymous user and role from before, adjust the ACLs accordingly and remove all ACL overrides.

```
<pentaho-system>
<!-- omitted -->
    <acl-publisher>
        <default-acls>
            <acl-entry role="Anonymous" acl="ADMIN_ALL" />
            <acl-entry role="Authenticated" acl="ADMIN_ALL" /> </default-acls>
            <!-- remove any active overrides entries -->
    </acl-publisher>
<!-- omitted -->
</pentaho-system>
```

9. Adjust the <acl-voter> properties such that the new anonymous user has administrator privileges.

```
<pentaho-system> <!-- omitted -->
    <acl-voter>
        <admin-role>Anonymous</admin-role>
    </acl-voter> <!-- omitted -->
</pentaho-system>
```

10. Save the file and close the text editor.

You have now effectively worked around the security features of the BA Server.

If you are using the relational metadata database model, refer to *Removing Security from Metadata Domain Repository* for the next few steps.

Troubleshooting

Adjusting authorization and authentication settings will often involve making multiple configuration changes without the benefit of testing each of them individually. Your first attempt at implementing a different security DAO or performing intensive user and role modifications will probably not work perfectly. Below are some tips for adjusting log file output, and examining logs for signs of configuration errors.

Increase Security Log Levels in the BA Server

The security logging facilities of the BA Server are set to ERROR by default, which is not always verbose enough for troubleshooting and testing. The below procedure explains how to increase the level of detail in the BA Server logs that deal with security-related messages.

1. Stop the BA Server.

   ```
   sh /usr/local/pentaho/server/biserver-ee/stop-pentaho.sh
   ```
2. Open the `/pentaho/server/biserver-ee/tomcat/webapps/pentaho/WEB-INF/classes/log4j.xml` file with a text editor.
3. Use XML comments (<!-- -->) to disable all of the **Threshold** parameters in all of the **appender** elements.
4. Change the priority value in the **<root>** section to one of: **WARN**, **ERROR**, **FATAL**, or **DEBUG**, depending on which level you prefer.

   ```
   <root>
     <priority value="DEBUG" />
     <appender-ref ref="PENTAHOCONSOLE"/>
     <appender-ref ref="PENTAHOFILE"/>
   </root>
   ```
5. Add the following loggers directly above the root element:

   ```
   <!-- all Spring Security classes will be set to DEBUG -->
   <category name="org.springframework.security">
     <priority value="DEBUG" />
   </category>

   <!-- all Pentaho security-related classes will be set to DEBUG -->
   <category name="org.pentaho.platform.engine.security">
     <priority value="DEBUG" />
   </category>
   <category name="org.pentaho.platform.plugin.services.security">
     <priority value="DEBUG" />
   </category>
   ```
6. Save and close the file, then edit the Spring Security configuration file that corresponds with your security backend in the `/pentaho/server/biserver-ee/pentaho-solutions/system/` directory.

   The file will be one of:

   - applicationContext-spring-security-memory.xml
   - applicationContext-spring-security-jdbc.xml
   - applicationContext-spring-security-ldap.xml
7. Find the **daoAuthenticationProvider** bean definition, and add the following property anywhere inside of it (before the </bean> tag):

   ```
   <property name="hideUserNotFoundExceptions" value="false" />
   ```
8. Save the file and close the text editor.
9. Start the BA Server.

   ```
   sh /usr/local/pentaho/server/biserver-ee/start-pentaho.sh
   ```

BA Server security logging is now globally set to DEBUG, which is sufficiently verbose for debugging security configuration problems. All BA Server messages will be collected in the `/pentaho/server/biserver-ee/logs/pentaho.log` file.

When you are finished configuring and testing the BA Server, you should adjust the log levels down to a less detailed level, such as ERROR, to prevent pentaho.log from growing too large.

Enable Extra LDAP Security Logging

If you need yet more LDAP-related security details in **pentaho.log**, or if you are specifically having difficulty with LDAP authentication configuration, follow the below process.

Note: These instructions are for testing and pre-production only. Usernames and passwords will be displayed in the log file in plain text.

1. Stop the BA Server.

```
sh /usr/local/pentaho/server/biserver-ee/stop-pentaho.sh
```

2. Open the `/pentaho/server/biserver-ee/pentaho-solutions/system/applicationContext-spring-security-ldap.xml` file with a text editor.
3. Change the reference in the first **constructor-arg** property of the **daoAuthenticationProvider** element to **ldapAuthenticatorProxy**

```
<constructor-arg>
    <ref bean="ldapAuthenticatorProxy" />
</constructor-arg>
```

4. Save the file, then create a new file called `applicationContext-logging.xml` in the same directory.
5. Copy the following text into the new file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/
dtd/spring-beans.dtd">
<beans>
    <bean id="ldapAuthenticatorProxy"
 class="org.springframework.aop.framework.ProxyFactoryBean">
        <property name="proxyInterfaces">
        <value>org.springframework.security.providers.ldap.LdapAuthenticator</value>
        </property>
        <property name="target">
            <ref bean="authenticator" />
        </property>
        <property name="interceptorNames">
            <list>
                <value>loggingAdvisor</value>
            </list>
        </property>
    </bean>
    <bean id="loggingAdvisor"
 class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
        <property name="advice">
            <ref local="loggingInterceptor" />
        </property>
        <property name="pattern">
            <value>.*</value>
        </property>
    </bean>
    <bean id="loggingInterceptor"
 class="org.pentaho.platform.engine.security.LoggingInterceptor" />
</beans>
```

6. Save the file, then open `pentaho-spring-beans.xml`.
7. Add the following import line to the list of files:

```
<import resource="applicationContext-logging.xml" />
```

8. Save and close the file, then start the BA Server.

```
sh /usr/local/pentaho/server/biserver-ee/start-pentaho.sh
```

You will now have verbose LDAP-specific log messages in **pentaho.log** that include login credentials for every user that tries to log in.

Log Output Analysis

The information you need to look for in **pentaho.log** in order to troubleshoot security configuration problems should be fairly self-evident. If you are having trouble, consult the examples below.

When you request a page that is protected but you are not yet logged in, you should see an exception in the log which looks like this:

```
DEBUG [ExceptionTranslationFilter] Access is denied (user is anonymous);
            redirecting to authentication entry point
 org.springframework.security.AccessDeniedException:
            Access is denied
```

When the username and/or password doesn't match what's stored in the backend, you should see a log message like this:

```
WARN [LoggerListener] Authentication event
            AuthenticationFailureBadCredentialsEvent: suzy; details:
            org.springframework.security.ui.WebAuthenticationDetails@fffd148a:
 RemoteIpAddress: 127.0.0.1;
            SessionId: 976C95033136070E0200D6DA26CB0277; exception: Bad credentials
```

When the username and password match, you should see a log message that looks like the example below. After the **InteractiveAuthenticationSuccessEvent**, one of the filters will show the roles fetched for the authenticated user. Compare these roles to the page-role mapping found in the **filterInvocationInterceptor** bean in **applicationContext-spring-security.xml**.

```
WARN [LoggerListener] Authentication event InteractiveAuthenticationSuccessEvent:
            suzy; details:
 org.springframework.security.ui.WebAuthenticationDetails@fffd148a: RemoteIpAddress:
            127.0.0.1; SessionId: 976C95033136070E0200D6DA26CB0277 DEBUG
            [HttpSessionContextIntegrationFilter] SecurityContext stored to
 HttpSession:
            'org.springframework.security.context.SecurityContextImpl@2b86afeb:
 Authentication:

 org.springframework.security.providers.UsernamePasswordAuthenticationToken@2b86afeb:
 Username:
            org.springframework.security.userdetails.ldap.LdapUserDetailsImpl@d7f51e;
 Password: [PROTECTED];
            Authenticated: true; Details:
 org.springframework.security.ui.WebAuthenticationDetails@fffd148a:
            RemoteIpAddress: 127.0.0.1; SessionId: 976C95033136070E0200D6DA26CB0277;
 Granted
            Authorities: ROLE_CTO, ROLE_IS, ROLE_AUTHENTICATED'
```

If you are troubleshooting LDAP problems, look for log output similar to this:

```
DEBUG [DirMgrBindAuthenticator] (LoggingInterceptor) Return value: LdapUserInfo:

 org.springframework.security.providers.ldap.LdapUserInfo@1f31c64[dn=uid=suzy,ou=users,ou=sy
            suzy.pentaho@pentaho.org, uid=uid: suzy, userpassword=userpassword:
 [B@e17c9c,
            businesscategory=businesscategory: cn=cto,ou=roles,ou=system,
 cn=is,ou=roles,ou=system,
            objectclass=objectClass: organizationalPerson, person, groupOfUniqueNames,
            inetOrgPerson, top, uniquemember=uniquemember: cn=cto, ou=roles, cn = is ,
ou = roles,
            sn=sn: Pentaho, cn=cn: suzy}]
```

LDAP Roles Are Not "Admin" and "Authenticated"

You must not use **Admin** and **Authenticated** roles in your LDAP. Instead you must configure your system to use **pentahoAdmins** and **pentahoUsers** or other easily identifiable role names. Edit `/pentaho-solutions/system/applicationContext-spring-security.xml`. At the bottom of this file, you will find a number of lines that look like: `A/docs/.*Z=Anonymous,Authenticated`.

These are entries for URL Security. They are regular expressions to match a path on the browser's URL that require the user to be a member of the defined role to gain access. In the example above, both Anonymous and Authenticated get access. In the example above, use **pentahoUsers** in the place of **Authenticated**. by entering `A/docs/.*Z=Anonymous,pentahoUsers`. For all entries that show Authenticated, replace it with **pentahoUsers** or your chosen name. Replace **Admin** with **pentahoAdmins** or your chosen name. For the change from **Authenticated** to **pentahoUsers** replace all occurrences. For **Admin** to **pentahoAdmins** you need to be a little more careful because there are some entries that look like this: `A/admin.*Z=pentahoAdmins`.

Edit the `/pentaho-solutions/system/repository.spring.xml` file and change:

```
<bean id="singleTenantAuthenticatedAuthorityName" class="java.lang.String">
    <constructor-arg value="Authenticated" />
   </bean>
```

to:

```
<bean id="singleTenantAuthenticatedAuthorityName" class="java.lang.String">
    <constructor-arg value="pentahoUsers" />
   </bean>
```

and:

```
<bean id="singleTenantAdminAuthorityName" class="java.lang.String">
    <constructor-arg value="Admin" />
   </bean>
```

to:

```
<bean id="singleTenantAdminAuthorityName" class="java.lang.String">
    <constructor-arg value="pentahoAdmins" />
   </bean>
```

With LDAP Authentication, the PDI Repository Explorer is Empty

If you log into a solution repository from Spoon before you switch the authentication to LDAP, then the repository IDs and security structures will be broken. You won't see an error message, but the solution repository explorer will be empty and you won't be able to create new folders or save PDI content to it. To fix the problem, you will have to delete the security settings established with the previously used authentication method, which will force the DI Server to regenerate them for LDAP.

Caution: Following this procedure will destroy any previously defined DI repository users, roles, and access controls. You should back up the files that you delete in these instructions.

1. Stop the DI Server
2. Delete the security and default directories from the following directory: `/pentaho-solutions/system/jackrabbit/repository/workspaces/`
3. Start the DI Server

You should now have a proper LDAP-based DI Repository that can store content and create new directories.

LDAP Incorrectly Authenticates User IDs That Do Not Match Letter Case

Some LDAP implementations are case-insensitive, most notably Microsoft Active Directory. When using one of these LDAP distributions as a BA Server authentication backend, you might run into an issue where a valid username with invalid letter cases will improperly validate. For instance, if **Bill** is the valid user ID, and someone types in **bILL** at the User Console login screen, that name will authenticate, but it might have improper access to parts of the BA Server.

The fix for this is documented: *LDAP Authenticates User IDs That Do Not Match Case* on page 32.

LDAP Authenticates User IDs That Do Not Match Case

Some LDAP implementations are case-insensitive for user names, most notably Microsoft Active Directory. You might run into an issue where a user name typed into the login screen does not exactly match the letter case of that user's ID in the directory, but the server will authenticate it anyway and may give the user improper access to parts of the BA Server. For example, if **Bill** is the valid user ID, and someone types in **bILL** at the User Console login screen, the incorrectly typed one will authenticate, but it may have improper access to parts of the BA Server.

Follow these instructions to force case-sensitivity and fix this potential security risk.

1. Stop the BA Server.
2. Edit the `/pentaho/server/biserver-ee/pentaho-solutions/system/applicationContext-spring-security-ldap.xml` file.
3. Find the **daoAuthenticationProvider** bean, and below the last **</constructor-arg>** therein, and add the **<property>** definition shown in the example:

```
<property name="userDetailsContextMapper">
    <ref local="ldapContextMapper" />
</property>
```

4. After the **</bean>** tag for **daoAuthenticationProvider**, add the following bean definition, changing the **ldapUsernameAttribute** from **samAccountName** to the value that matches your environment:

```
<bean id="ldapContextMapper"
 class="org.pentaho.platform.engine.security.UseridAttributeLdapContextMapper">
    <property name="ldapUsernameAttribute" value="samAccountName" />
</bean>
```

5. Start the BA Server.

The BA Server will now force case sensitivity in LDAP usernames.