HITACHI
Inspire the Next

Configuring Pentaho to Use Database-Based Security

Contents

Overview

This document covers some best practices on Java database connectivity (JDBC) authentication. In it, you will learn how to set up Pentaho to authenticate with a database-based authentication scheme.

Our intended audience is Pentaho administrators, or anyone with a background in authentication and authorization who is interested in applying JDBC.

The intention of this document is to speak about topics generally; however, these are the specific versions covered here:

Software	Version(s)
Pentaho	7.1, 8.x

The Components Reference in Pentaho Documentation has a complete list of supported software and hardware.

Before You Begin

This document assumes that you have some background in database administration and network authentication and that you have already installed Pentaho. More information about related topics outside of this document can be found at Pentaho Installation and Security Issues.

Use Case: Applying Pentaho to Existing Database-Based Security

Janice administers database-based security on internal applications that use their own authentication. Because these applications do not use Microsoft Active Directory, the users' passwords may be different from their Windows authentication information. To simplify matters, Janice decides to use JDBC security to plug Pentaho into her existing database-based security. She will then be able to manage user access to Pentaho by manipulating their own tables and queries.

You could use a hybrid configuration for this as well. Manual LDAP/JDBC Hybrid Configuration in Pentaho documentation has information on how to do this.

Authentication and Authorization

To configure Pentaho to use a database-based authentication scheme, you must first know how Pentaho uses authentication and authorization, and how Spring Framework fits in.

Authentication occurs when the user logs in with their credentials and the system checks to make sure the user is valid and active. Once the user is validated, the system checks to see what roles the user has, which will define what the user is authorized to do on the server. Roles are assigned only once authentication has occurred, and they handle operational permissions.

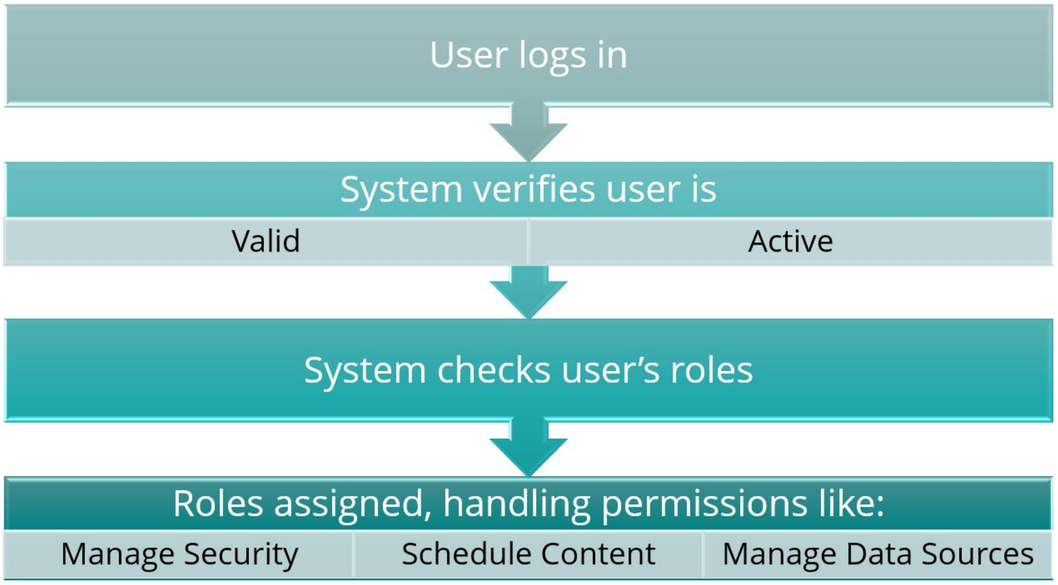

Figure 1: Authentication

Seeing the contents of a report is controlled by Mondrian roles in Analyzer reports and platform roles in Metadata models used in Interactive Reports and Pentaho Report Designer Reports, and should not be confused with authorization, which allows a user to open a report. Seeing the contents of a report is security-constrained access and beyond the scope of this document.

Database Structure

To work with Pentaho and Spring Framework, you can use any database structure as long as it has, at a minimum, the equivalent of the following three tables. You can find information for setting up User Security in the Pentaho documentation.

1. **Users**: containing user information
2. **Authorities**: containing role information
3. **Granted_Authorities**: combining users and roles granted each user. In practice, there should be a one-to-one relationship between user and role in the table, so if a user has four roles, there should be four entries for the user, one for each role.

You can find more information on these topics in the following sections:

- Table Declarations
- Table Population

Table Declarations

These table declarations are minimal settings for JDBC security:

1. Create table `users`:

```
CREATE TABLE USERS (
USERNAME VARCHAR2(50) NOT NULL PRIMARY KEY,
PASSWORD VARCHAR2(50) NOT NULL,
ENABLED INTEGER DEFAULT 1 NOT NULL,
DESCRIPTION VARCHAR2(100);
```

2. Create table `authorities`:

```
CREATE TABLE AUTHORITIES(
AUTHORITY VARCHAR(50) NOT NULL PRIMARY KEY,
DESCRIPTION VARCHAR(100));
```

3. Create table `granted_authorities`:

```
CREATE TABLE GRANTED_AUTHORITIES(
USERNAME VARCHAR(50) NOT NULL,
AUTHORITY VARCHAR(50) NOT NULL,
CONSTRAINT FK_GRANTED_AUTHORITIES_USERS FOREIGN KEY(USERNAME) REFERENCES
USERS(USERNAME),
CONSTRAINT FK_GRANTED_AUTHORITIES_AUTHORITIES FOREIGN KEY(AUTHORITY)
REFERENCES AUTHORITIES(AUTHORITY))
```

Table Population

Populate your tables like this:

Users Table

By default, we use the PlaintextPasswordEncoder, which reads the password as it is in the database. You can use a different password encoder if you want. Here is example code to illustrate the `users` table with plaintext passwords:

```
INSERT INTO USERS VALUES('gabellard','Password1',1,NULL);
INSERT INTO USERS VALUES('wfaulkner','Password1',1,NULL);
INSERT INTO USERS VALUES('clopez','Password1',1,NULL);
INSERT INTO USERS VALUES('skemparaju','mypassword',1,NULL);
```

Authorities Table

Example code to illustrate the `authorities` table:

```
INSERT INTO AUTHORITIES VALUES('DBPentAdmins','Super User');
INSERT INTO AUTHORITIES VALUES('DBPentHR','HR Users');
INSERT INTO AUTHORITIES VALUES('DBPentFinance','Finance Users');
INSERT INTO AUTHORITIES VALUES('DBPentUsers','User has not logged in');
INSERT INTO AUTHORITIES VALUES('DBPentSales','Sales Users');
```

Granted_Authorities Table

Example code to illustrate the `granted_authorities` table with a one-to-one relationship between users and roles:

```
INSERT INTO GRANTED_AUTHORITIES VALUES('gabellard','DBPentAdmins');
INSERT INTO GRANTED_AUTHORITIES VALUES('gabellard','DBPentUsers');
INSERT INTO GRANTED_AUTHORITIES VALUES('clopez','DBPentUsers');
INSERT INTO GRANTED_AUTHORITIES VALUES('clopez','DBPentFinance');
INSERT INTO GRANTED_AUTHORITIES VALUES('wfaulkner','DBPentUsers');
INSERT INTO GRANTED_AUTHORITIES VALUES('wfaulkner','DBPentHR');
INSERT INTO GRANTED_AUTHORITIES VALUES('skemparaju','DBPentSales');
INSERT INTO GRANTED_AUTHORITIES VALUES('skemparaju','DBPentUsers');
```

Configuring Pentaho to Use JDBC Security

The following steps assume that Pentaho has already been installed. Make sure you follow these steps after you have stopped the Pentaho Server.

- Step 1: Copy JDBC Driver
- Step 2: Change Pentaho's Default Security Provider
- Step 3: Connect Pentaho to Your Database
- Step 4: Map the Administrator Role
- Step 5: Map the Administrator User

Step 1: Copy JDBC Driver

Filling in your own installation path, copy the JDBC driver to:

```
<installation path>/pentaho-server/tomcat/lib/
```

Step 2: Change Pentaho's Default Security Provider

Pentaho's default security provider is Jackrabbit. To change this to JDBC, follow these steps:

1. Locate the file `<installation path>/pentaho-server/pentaho-solutions/system/security.properties`.
2. Change the provider from `provider=jackrabbit` to `provider=jdbc`.
3. Save the file.

Step 3: Connect Pentaho to Your Database

Since you have already copied the JDBC driver to tomcat/lib, you can now connect Pentaho to your database with these steps:

1. Locate the file `<installation path>/pentaho-server/pentaho-solutions/system/applicationContext-spring-security-jdbc.properties`.
2. Add the correct database information. This example uses PostgreSQL.

```
# The fully qualified Java class name of the JDBC driver to be used
datasource.driver.classname=org.postgresql.Driver

# The connection URL to be passed to our JDBC driver to establish a
connection
datasource.url=jdbc:postgresql://localhost:5432/jdbc_auth

# The connection username to be passed to our JDBC driver to establish a
connection
datasource.username=postgres

# The connection password to be passed to our JDBC driver to establish a
connection
datasource.password=password

# The SQL query that will be used to validate connections from this pool
before returning them to the caller.
# This query must be an SELECT statement that returns at least one row.
# HSQLDB: SELECT 1 FROM INFORMATION_SCHEMA.SYSTEM_USERS
# MySQL, H2, MS-SQL, POSTGRESQL, SQLite: SELECT 1
# ORACLE: SELECT 1 FROM DUAL
datasource.validation.query=SELECT 1

# the maximum number of milliseconds that the pool will wait (when there
are no available connections)
# for a connection to be returned before throwing an exception, or <= 0 to
wait indefinitely. Default value is -1
datasource.pool.max.wait=-1

# The maximum number of active connections that can be allocated from this
pool at the same time, or negative for no limit. Default value is 8
datasource.pool.max.active=8


# The maximum number of connections that can remain idle in the pool,
without extra ones being destroyed, or negative for no limit. Default value
is 8
datasource.max.idle=4

# The minimum number of active connections that can remain idle in the
pool, without extra ones being created when the evictor runs, or 0 to
create none. Default value is 0
datasource.min.idle=0
```

Step 4: Map the Administrator Role

Next, map the Administrator role correctly using these steps:

1. Locate the file `<installation path>/pentaho-server/pentaho-solutions/system/applicationContext-pentaho-security.jdbc.xml`.
2. Change the `<entry key>` to the admin role value from the database from

```
<util:map id="jdbcRoleMap">
<entry key="Admin" value="Administrator"/>
</util:map>
```

   to

```
<util:map id="jdbcRoleMap">
<entry key="DBPentAdmins" value="Administrator"/>
</util:map>
```

3. Save the file.

Step 5: Map the Administrator User

Once you have the Administrator role mapped, map the Administrator user:

1. Locate the directory `/pentaho-server/pentaho-solutions/system`.
2. Open the `repository.spring.properties` file.
3. Locate the following line:

```
singleTenantAdminUserName=admin
```

4. Change it to map an administrator user in your database authentication:

```
singleTenantAdminUserName=DBAdminUser
```

5. Save and close the file.

Understanding Queries Against Your JDBC Security

Now that Pentaho Server has been configured to use your JDBC security, you can find further information about the queries against your database in this section.

This section is provided for information, should you want to explore these areas further. Actions in this section are not *a required part of the configuration.*

- Spring Framework Queries
- Pentaho Queries

Spring Framework Queries

As a user logs in, two queries are fired: one to get information about the user, and one to find out what roles the user belongs to. You can find this in the log with these steps:

1. Locate the file `<installation path>/pentaho-server/tomcat/webapps/pentaho/WEB-INF/classes/log4j.xml`.
2. Add the following categories:

```
<category name="org.springframework.security">
      <priority value="DEBUG"/>
</category>
```

3. In the `pentaho.log`, you will see:

```
DEBUG
[org.springframework.security.web.authentication.AnonymousAuthenticationFil
ter] SecurityContextHolder not populated with anonymous token, as it
already contained:
'org.springframework.security.authentication.UsernamePasswordAuthentication
Token@f81a4943: Principal:
org.springframework.security.core.userdetails.User@fc211e2b: Username:
skemparaju; Password: [PROTECTED]; Enabled: true; AccountNonExpired: true;
credentialsNonExpired: true; AccountNonLocked: true; Granted Authorities:
Authenticated,DBPentSales,DBPentUsers; Credentials: [PROTECTED];
Authenticated: true; Details:
org.springframework.security.web.authentication.WebAuthenticationDetails@25
5f8: RemoteIpAddress: 127.0.0.1; SessionId:
4713A775E8A737E8ED4A3E2E768B5653; Granted Authorities: Authenticated,
DBPentSales, DBPentUsers'
```

The queries that result in this information are found in `<installation path>/pentaho-server/pentaho-solutions/system/applicationContext-spring-security-jdbc.xml`.

4. The `usersByUsernameQuery` loads the username and password:

```
SELECT username, password, enabled FROM USERS WHERE username = ? ORDER BY
username
```

5. The `authoritiesByUsernameQuery` loads the roles the user belongs to:

```
SELECT username, authority FROM GRANTED_AUTHORITIES WHERE username = ?
ORDER BY authority
```

Pentaho Queries

When you start Pentaho, it will connect to the database to gather information about the user and authorities for Pentaho authorization. The Administrator role and user are used to retrieve this information.

To see this information in the `pentaho.log`, follow these steps:

1. Locate the file `<installation path>/pentaho-server/tomcat/webapps/pentaho/WEB-INF/classes/log4j.xml`.

2. Add the following categories:

```
<category name="org.pentaho.platform.engine.security">
      <priority value="DEBUG"/>
</category>
```

3. In the `pentaho.log`, you will see:

```
DEBUG
[org.pentaho.platform.plugin.services.security.userrole.jdbc.JdbcUserRoleLi
stService$AllAuthoritiesMapping] RdbmsOperation with SQL [SELECT
distinct(authority) as authority FROM AUTHORITIES ORDER BY authority]
compiled

DEBUG
[org.pentaho.platform.plugin.services.security.userrole.jdbc.JdbcUserRoleLi
stService$AllUserNamesInRoleMapping] RdbmsOperation with SQL [SELECT
distinct(username) as username FROM GRANTED_AUTHORITIES where authority = ?
ORDER BY username] compiled

DEBUG
[org.pentaho.platform.plugin.services.security.userrole.jdbc.JdbcUserRoleLi
stService$AllUserNamesMapping] RdbmsOperation with SQL [SELECT
distinct(username) as username FROM USERS ORDER BY username] compiled
```

Note that these are the same queries shown in `<installation path>/pentaho-server/pentaho-solutions/system/applicationContext-pentaho-security-jdbc.xml`.

4. The `allAuthoritiesQuery`, used to show all the roles on the Authorities table, is written as:

```
SELECT distinct(authority) as authority FROM AUTHORITIES ORDER BY authority
```

5. These roles are later displayed under **Users & Roles** on the Administration Perspective in the Pentaho User Console (PUC):

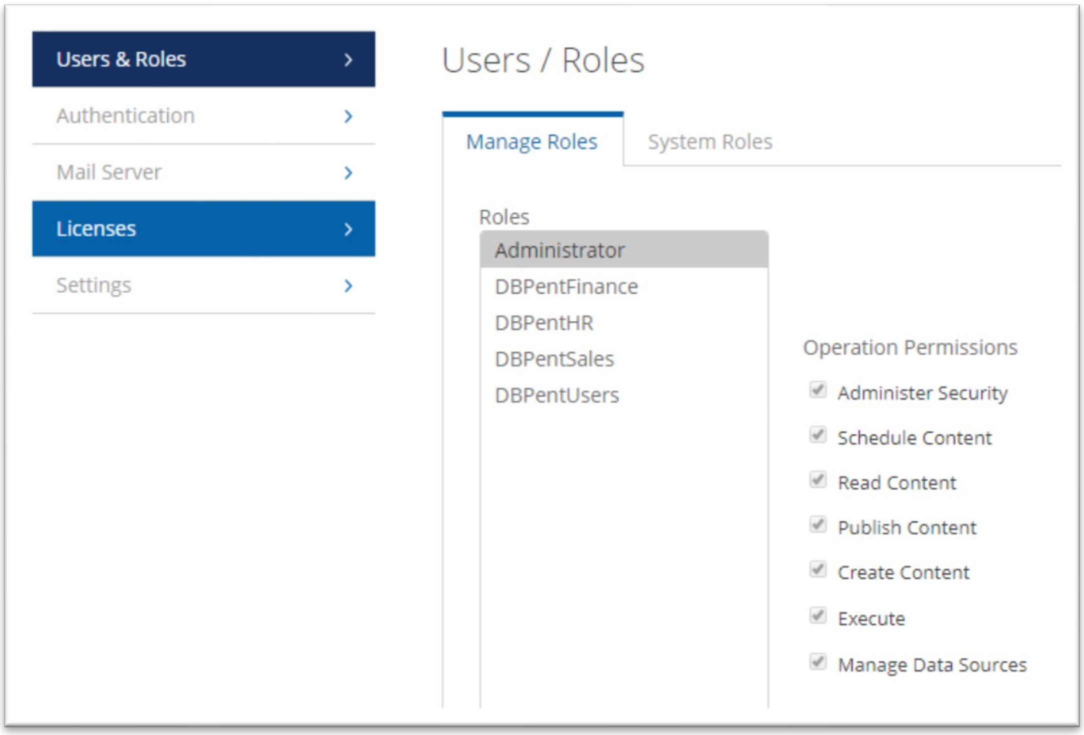

Figure 2: Users & Roles in the PUC

6. The `allUsernamesInRoleQuery`, which gets all users who belong to a specific role, is written as:

```
SELECT distinct(username) as username FROM GRANTED_AUTHORITIES where
authority = ? ORDER BY username
```

7. The `allUsernamesQuery`, which shows the users on the Share or Permissions tab, is written as:

```
SELECT distinct(username) as username FROM USERS ORDER BY username
```

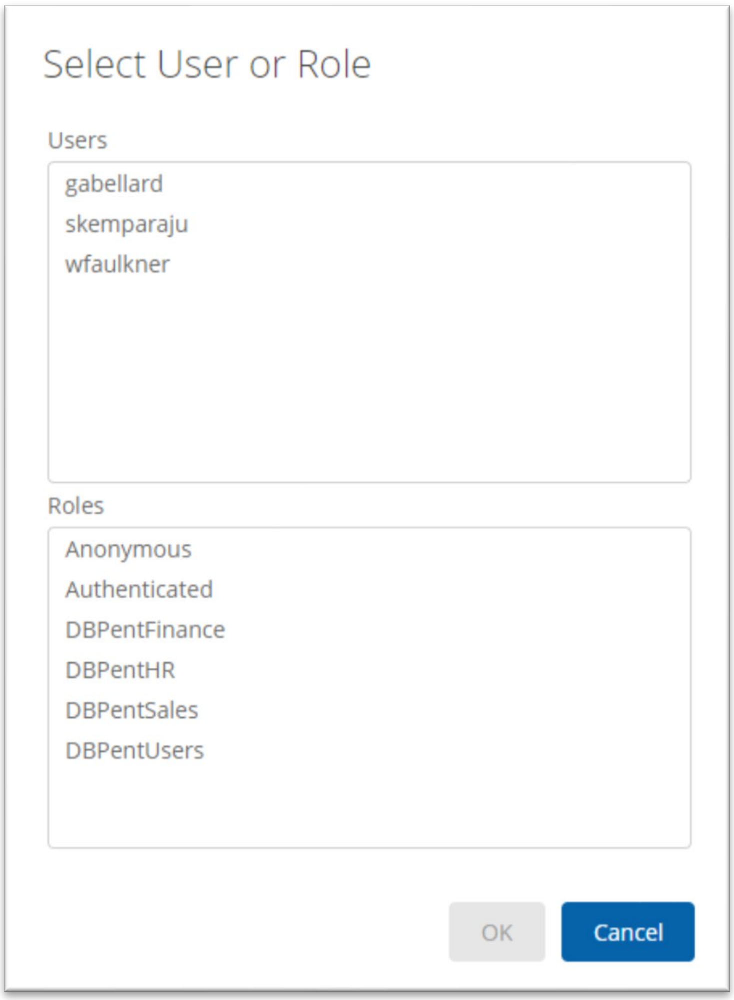

Figure 3: Select User or Role

8. Check the `pentaho.log` for the Administrator role and user:

```
DEBUG
[org.springframework.security.core.userdetails.cache.EhCacheBasedUserCache]
Cache hit: true; username: gabellard
DEBUG [org.pentaho.platform.engine.security.SecurityHelper]
rolesForUser:[Authenticated, Administrator, DBPentUsers, Anonymous]
```

These must be active in your database, or you will not be able to connect to your database to extract the roles and users necessary for the Pentaho Server to function properly.

Known Issues

In this section are a few known issues to be aware of, as well as solutions for each.

Database and Table Structure are Different

If you are already using a different table structure for your JDBC authentication, make sure you use an alias for the different field names, as illustrated in this example:

```
SELECT userid as username, 'password' as password, 'enabled' as enabled
FROM USERS_ROLES WHERE userid= ? ORDER BY userid
```

Browse File Keeps Spinning with No Results

After configuring JDBC security, you may run into an issue where your browse file constantly spins but does not show anything. The `Catalina` log shows an example:

```
SEVERE: The RuntimeException could not be mapped to a response, re-throwing
to the HTTP container

org.pentaho.platform.api.repository2.unified.UnifiedRepositoryException:
exception while getting tree rooted at path "/"

Reference number: 2f863f91-f38f-4176-91a2-a0fb43a73af2

at
org.pentaho.platform.repository2.unified.ExceptionLoggingDecorator.callLogT
hrow(ExceptionLoggingDecorator.java:512)
```

The `pentaho.log` will show comparable results. This can happen for any of the following reasons:

- One of the queries in either configuration file is returning a null value.
- Passwords, roles, or granted roles are null.
- Users are not properly disabled.
- A role is not properly mapped to a user in the `granted_authorities` table.

To fix this, follow these steps:

1. Locate the file `<installation path>/pentaho-server/tomcat/webapps/pentaho/WEB-INF/classes/log4j.xml`.
2. Add the following category to the file:

```
<category
name="org.pentaho.platform.repository2.unified.ExceptionLoggingDecorator">
<priority value="DEBUG"/>
</category>
```

3. Restart the Pentaho Server.
4. Log back in again and choose **Browse Files**.

Passwords Stored in Cleartext

By default, Pentaho connects to your database using a cleartext password stored in the file `<installation-path>/pentaho-server/pentaho-solutions/system/applicationContext-spring-security-jdbc.properties`.

A workaround to this is to use an account or database login that is only for this database. You will need `READ ONLY` permissions for this. Do not use something like a system administrator account or similar.

Related Information

Here are some links to information that you may find helpful while using this best practices document:

- Manual LDAP/JDBC Hybrid Configuration
- Pentaho Installation
- Pentaho Components Reference
- Security Issues
- User Security
- Spring Framework
- Starting and Stopping the Pentaho Server

Finalization Checklist

This checklist is designed to be added to any implemented project that uses this collection of best practices, to verify that all items have been considered and reviews have been performed. (Compose specific questions about the topics in the document and put them in the table.)

Name of the Project: ____

Date of the Review: ____

Name of the Reviewer: ____

Item	Response	Comments
Did you install Pentaho before beginning?	YES________ NO________	
Did you familiarize yourself with how Pentaho uses authentication and authorization?	YES________ NO________	
Are you familiar with Spring Framework?	YES________ NO________	
Did you follow the table declaration settings for JDBC security?	YES________ NO________	
Did you stop the Pentaho server before starting your configuration?	YES________ NO________	
Did you use aliases for the different field names, if you have a different table structure for your JDBC authentication?	YES________ NO________	