Securing a web application on Glassfish using JAAS - Pt 2

User Rating: / 2
Poor Best Rate

Written by David Salter

Thursday, 24 August 2006 09:19

Securing a Web Application on Glassfish using JAAS - Part 2

In my previous article, I explained how to secure web applications deployed to the Glassfish application server using JAAS authentication. The article explained how to configure a file based security realm and how to use BASIC HTTP authentication. This second article explains how to configure web applications to use HTTP FORM based authentication in conjunction with JDBC realms within Glassfish.

amazon.com Privacy Get Widget
Cloud Computing: Web-Based...
Beginning Java EE 6 Platfo...
Real World Java EE Pattern...
Java EE 5 Development with...

Glassfish 1 only supports JDBC realms from version 1 UR1 onwards. This is the version used throughout this article. All the code samples within the article are compiled and deployed using NetBeans 5.5 beta 2. The samples of code included in this article are all available as part of a sample application to download. Details of how to download this sample application are included at the end of the article.

Configuring HTTP FORM Based Authentication

Configuring FORM based authentication in a web application is achieved by configuration within the web.xml file for the application. In order to define FORM based authentication, a login page and a login-error page must also be defined as shown in the following XML:

```
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>developinjava</realm-name>
    <form-login-config>
        <form-login-page>/login.jsp</form-login-page>
        <form-error-page>/loginerror.jsp</form-error-page>
    </form-login-config>
</login-config>
```

Whenever a request is made to a secured page from an unauthenticated user, the application will be forwarded to the page specified in the <form-login-page> section. If the logon process fails for any reason (for example the user gets their password wrong), the application will display the page specified in the <form-error-page> section. After a successful login, the user will be forwarded to the relevant page.

The FORM login page

The form login page must include a HTML form where the action and names of the username and password input boxes are all predefined constants. The action of the form must be defined as "j_security_check" and the username and password input boxes must be called "j_username" and "j_password" respectively. The following HTML gives an example of a login form.

```
<form method="POST" action="j_security_check">
    <table>
    <tr><td>User name:</td><td><input type="text" name="j_username" /></td></tr>
    <tr><td>Password:</td><td><input type="password" name="j_password" /></td></tr>
    <tr><td><input type="submit" value="Login" /></td></td></tr>
    </table>
</form>
```

Thats all there is to configuring HTTP FORM based authentication in a web application. With FORM based authentication, you can easily secure a web application whilst keeping total control over the look and feel of the logon process. In the next section, I'll describe how to configure JDBC realms within Glassfish concluding with a sample application showing a simple secured web application.

Configuring a JDBC Realm in glassfish

Configuring a JDBC realm within Glassfish is a relatively simple process. We need to define a database that contains a list of usernames/passwords and their security groups. Once the database has been defined, the realm needs to be configured within the Glassfish administration console. For simplicity in the sample application provided with this article, these tables have been created within the sun-appserv-samples Derby database provided with Glassfish. They can however be deployed to any database as required.

Creating a security database

A simple security database for a JDBC realm consists of 2 tables. The first of these contains a list of users and their passwords. The second table provides a list of security groups/roles that each user has access to. The basic schema for these tables is as follows (primary and foreign keys should be applied as appropriate for the relevant database):

```
create table USERTABLE(USERID varchar(20), PASSWORD varchar(20))
```

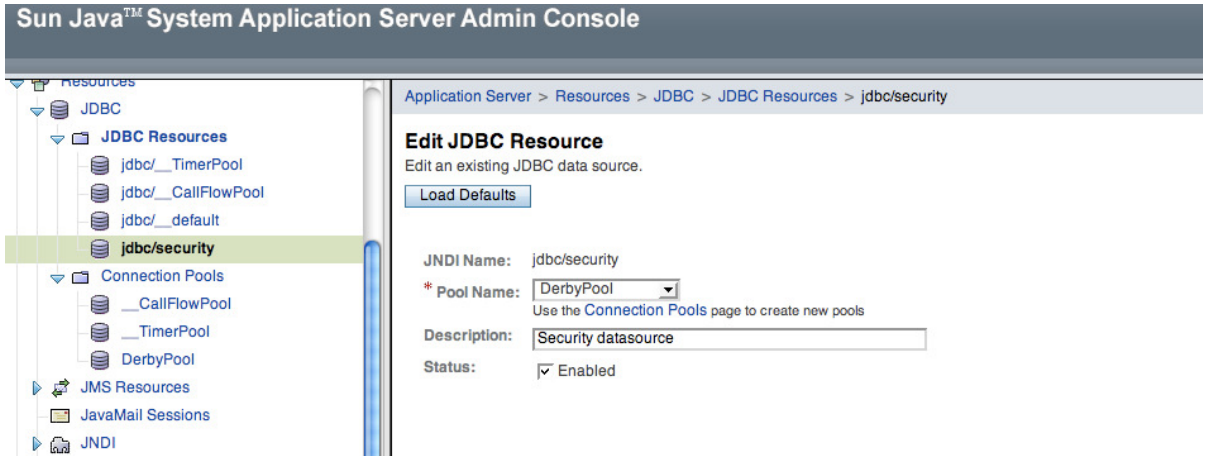
```
create table GROUPTABLE(USERID varchar(20), GROUPID varchar(20))
```

Sample data can be inserted into these tables using:

```
insert into USERTABLE(USERID,PASSWORD) values ('developinjava', 'secret')
insert into GROUPTABLE(USERID,GROUPID) values ('developinjava', 'USERS')
```

For ease of development, you can see that the password has not been encrypted in the user table. This would probably not be acceptable for a live application, but has benefits for development and testing (particularly when configuring security). Glassfish supports using encrypted password in the database. This is discussed later.

Now that there is a security database defined, this needs to be configured within Glassfish. Again for simplicity, I have chosen to use the "DerbyPool" that is preconfigured in Glassfish. Depending upon your needs, you may wish to define a separate database pool for security. To create a JDBC Resource to allow the JDBC realm to query the database, we need to add a new JDBC resource in the "Resources | JDBC | JDBC Resources" section within the Glassfish administration console. The following screenshot shows an example of how this is configured. The JNDI Name has been specified as "jdbc/security" and uses the DerbyPool connection pool.



Creating a JDBC realm

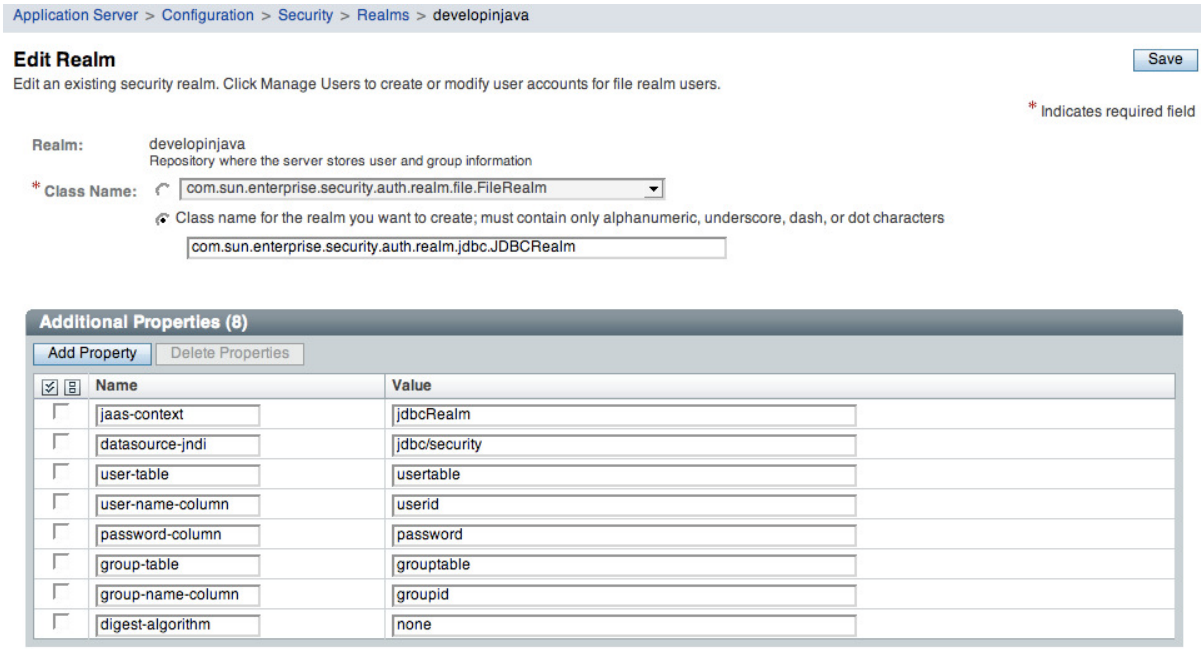
A JDBC realm allows usernames/passwords/groups to be defined dynamically within a set of database tables. JDBC realms are created within Glassfish using the administration console. To create the realm, open up the administration console in Glassfish and open the "Configuration | Security | Realms" tree node. Select the "Realms" node and select the "New..." button displayed in the righthand side panel. This process is described in further detail in the first article.

Create a new realm with the name "developinjava". The classname for the JDBC realm is not included in the drop down "Class Name" combo-box on the new realm pane, so it has

to be entered manually. The class name should be entered as "com.sun.enterprise.security.auth.realm.jdbc.JDBCRealm". Several additional properties are required for this realm which describe how the security database is accessed. These additional properties are listed in the table below (the names of the properties are self explanatory):

jaas-context	jdbcRealm
datasource-jndi	jdbc/security
user-table	usertable
user-name-column	userid
password-column	password
group-table	grouptable
group-name-column	groupid
digest-algorithm	none

A screen shot of the realm definition in the Glassfish admin console is shown below:



As noted above, when defining the JDBC realm, several parameters are used to define the connection to the database. In this example, the digest-algorithm is configured to be "none", effectively meaning that plain text passwords are compared with the passwords stored in the database. To make this more secure, we could have specified "MD5" as the digest-algorithm and then stored MD5 encoded passwords in the database.

Sample application

The above shows how to configure FORM based authentication against a JDBC JAAS realm when deployed on Glassfish. To conclude this article, a sample application can be downloaded that uses the techniques discussed. Within the application, the user is presented with an unsecured web page. From here it is possible to access a secured web page. To access this page, the FORM based login is shown. The application also shows how it is possible to get the username of the logged on user and how it is possible to invoke a servlet to log off the current user. The sample application was developed in NetBeans 5.5 beta 2 and can be downloaded from here.