JAAS

What is JAAS?

JAAS provides a pluggable framework for authenticating and authorising users (more information is available on the JAAS pages @ Sun Developer Network). Many application servers support JAAS as a means of bringing greater flexibility to the declarative security models of the J2EE (now known as the Java EE) specifications. Whilst intentionally not a full-blown application server, jetty6 supports JAAS to provide greater alternatives for servlet security, and to increase the portability of web applications.

The JAAS support aims to dictate as little as possible whilst providing a sufficiently flexible infrastructure to allow users to drop in their own custom LoginModules. We will discuss how to configure jetty6 for JAAS, and describe the example LoginModules provided with the distribution.

Configuration

Using JAAS with jetty is very simply a matter of declaring a `org.mortbay.jetty.plus.jaas.JAASUserRealm`, creating a jaas login module configuration file and specifying it on the jetty run line. Let's look at an example.

Step 1

Configure a jetty6 `org.mortbay.jetty.plus.jaas.JAASUserRealm` to match the `<realm-name>` in your `web.xml` file. For example, if the `web.xml` contains a realm called `"xyzrealm"`:

```
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>xyzrealm</realm-name>
  <form-login-config>

<form-login-page>/login/login</form-login-page>

<form-error-page>/login/error</form-error-page>
  </form-login-config>
</login-config>
```

Then the following `JAASUserRealm` would be declared in a jetty configuration file:

```
<Call name="addUserRealm">
  <Arg>
    <New
class="org.mortbay.jetty.plus.jaas.JAASUserR
ealm">
      <Set name="name">xyzrealm</Set>
      <Set name="LoginModuleName">xyz</Set>
    </New>
  </Arg>
</Call>
```

Important!

It is imperative that the contents of the `<realm-name>` and the `<Set name="name">` of the JAASUserRealm instance are *exactly* the same

Step 2

Set up your LoginModule in a configuration file, following the syntax rules:

```
xyz {
       com.acme.SomeLoginModule required
debug=true;
    };
```

Important!

It is imperative that the application name to the left of the { is *exactly* the same as the `<Set name="LoginModuleName">`

Step 3

Invoke jetty with the jaas configuration file you created in step 2:

```
java
-Djava.security.auth.login.config=mylogin.co
nf -jar start.jar etc/myjetty.xml
```

A Closer Look at the JAASUserRealm

To allow the greatest degree of flexibility in using JAAS with web applications, the `JAASUserRealm` supports a couple of configuration options. Note that you don't ordinarily need to set these explicitly, as jetty6 has defaults which will work in 99% of cases. However, should you need to, you can configure:

- a policy for role-based authorization (Default: org.mortbay.jetty.plus.jaas.StrictRoleCheckPolicy)
- a CallbackHandler (Default: org.mortbay.jetty.plus.jaas.callback.DefaultCallbackHandler)
- a list of classnames for the Principal implementation that equate to a user role (Default: org.mortbay.jetty.plus.jaas.JAASRole)

Here's an example of setting each of these (to their default values):

```
<New
class="org.mortbay.jetty.plus.jaas.JAASUserR
ealm">
  <Set name="Name">xyzrealm</Set>
  <Set name="LoginModuleName">xyz</Set>
  <Set name="RoleCheckPolicy">
    <New
class="org.mortbay.jetty.plus.jaas.StrictRol
eCheckPolicy"/>
  </Set>
  <Set name="CallbackHandlerClass">

org.mortbay.jetty.plus.jaas.callback.Default
CallbackHandler
  </Set>
  <Set name="roleClassNames">
    <Array type="java.lang.String">

<Item>org.mortbay.jetty.plus.jaas.JAASRole</
Item>
    </Array>
  </Set>
</New>
```

RoleCheckPolicy

The `RoleCheckPolicy` must be an implementation of the `org.mortbay.jetty.plus.jaas.RoleCheckPolicy` interface and its purpose is to help answer the question "is User X in Role Y" for role-based authorization requests. The default implementation distributed with jetty is the `org.mortbay.jetty.plus.jaas.StrictRoleCheckPolicy`, which will assess a user as having a particular role iff that role is at the top of the stack of roles that have been temporarily pushed onto the user or if the user has no temporarily assigned roles, the role is amongst those configured for the user.

Roles can be temporarily assigned to a user programmatically by using the `pushRole(String rolename)` method of the `org.mortbay.jetty.plus.jaas.JAASUserPrincipal` class.

For the majority of webapps, the default `StrictRoleCheckPolicy` will be quite adequate, however you may provide your own implementation and set it on your `JAASUserRealm` instance.

CallbackHandler

A CallbackHandler is responsible for interfacing with the user to obtain usernames and credentials to be authenticated.

Jetty ships with the `org.mortbay.jetty.plus.jaas.DefaultCallbackHandler` which interfaces the information contained in the request to the Callbacks that are requested by LoginModules. You can replace this default with your own implementation if you have specific requirements not covered by the default.

Role Principal Implementation Class

When LoginModules authenticate a user, they usually also gather all of the roles that a user has and place them inside the JAAS Subject. As LoginModules are free to use their own implementation of the JAAS Principal to put into the Subject, jetty needs to know which Principals represent the user and which represent his/her roles when performing authorization checks on `<security-constraint>s`. The example LoginModules that ship with jetty all use the `org.mortbay.jetty.plus.jaas.JAASRole` class. However, if you have plugged in some other LoginModules, you must configure the classnames of their role Principal implementations.

Sample Login Modules

At the time of writing, jetty6 ships with 3 sample LoginModule implementations:

- `org.mortbay.jetty.plus.jaas.spi.JDBCLoginModule`
- `org.mortbay.jetty.plus.jaas.spi.PropertyFileLoginModule`
- `org.mortbay.jetty.plus.jaas.spi.DataSourceLoginModule`
- `org.mortbay.jetty.plus.jaas.ldap.LdapLoginModule`

We'll take a look at all of these, but first, a word about password handling in jetty6, as it applies to all LoginModules.

Passwords/Credentials

Passwords can be stored in clear text, obfuscated or checksummed. The class `org.mortbay.util.Password` should be used to generate all varieties of passwords,the output from which can be cut and pasted into property files or entered into database tables.

```
> java -cp lib/jetty.jar
org.mortbay.jetty.security.Password
Usage - java org.mortbay.util.Password
[<user>] <password>
> java -cp lib/jetty.jar
org.mortbay.jetty.security.Password me you
you
OBF:20771x1b206z
MD5:639bae9ac6b3e1a84cebb7b403297b79
CRYPT:me/ks90E221EY
```

Read more on Securing Passwords.

JDBCLoginModule

The JDBCLoginModule stores user passwords and roles in a database that are accessed via JDBC calls. You can configure the JDBC connection information, as well as the names of the table and columns storing the username and credential, and the name of the table and columns storing the roles.

Here is an example login module configuration file entry for it using an HSQLDB driver:

```
jdbc {

org.mortbay.jetty.plus.jaas.spi.JDBCLoginMod
ule required
      debug="true"
      dbUrl="jdbc:hsqldb:."
      dbUserName="sa"
      dbDriver="org.hsqldb.jdbcDriver"
      userTable="myusers"
      userField="myuser"
      credentialField="mypassword"
      userRoleTable="myuserroles"
      userRoleUserField="myuser"
      userRoleRoleField="myrole";
      };
```

There is no particular schema required for the database tables storing the authentication and role information. The properties `userTable`, `userField`, `credentialField`, `userRoleTable`, `userRoleUserField`, `userRoleRoleField` configure the names of the tables and the columns within them that are used to format the following queries:

```
select <credentialField> from <userTable>
where <userField> =?
select <userRoleRoleField> from
<userRoleTable> where <userRoleUserField> =?
```

Credential and role information is lazily read from the database when a previously unauthenticated user requests authentication. Note that this information is only cached for the length of the authenticated session. When the user logs out or the session expires, the information is flushed from memory.

Note that passwords can be stored in the database in plain text or encoded formats, using the org.mortbay.jetty.security.Password class.

DataSourceLoginModule

Similar to the JDBCLoginModule, but this LoginModule uses a DataSource to connect to the database instead of a jdbc driver. The DataSource is obtained by doing a jndi lookup on `java:comp/env/${dnJNDIName}`

Here is a sample login module configuration for it:

```
ds {

org.mortbay.jetty.plus.jaas.spi.DataSourceLo
ginModule required
     debug="true"
     dbJNDIName="ds"
     userTable="myusers"
     userField="myuser"
     credentialField="mypassword"
     userRoleTable="myuserroles"
     userRoleUserField="myuser"
     userRoleRoleField="myrole";
    };
```

PropertyFileLoginModule

With this login module implementation, the authentication and role information is read from a property file.

```
props {

org.mortbay.jetty.plus.jaas.spi.PropertyFile
LoginModule required
        debug="true"
        file="/somewhere/somefile.props";
      };
```

The `file` parameter is the location of a properties file of the same format as the `etc/realm.properties` exampl e file. The format is:

```
<username>: <password>[,<rolename> ...]
```

Here's an example:

```
fred: OBF:1xmk1w261u9r1w1c1xmq,user,admin
harry: changeme,user,developer
tom:
MD5:164c88b302622e17050af52c89945d44,user
dick: CRYPT:adpexzg3FUZAk,admin
```

The contents of the file are fully read in and cached in memory the first time a user requests authentication.

LdapLoginModule

Note that the LdapLoginModule ships in a separate jar, in $JETTY-HOME/lib/ext/jetty-ldap-jaas.jar. It requires JDK1.5 or above.

```
ldaploginmodule {

org.mortbay.jetty.plus.jaas.spi.LdapLoginMod
ule required
   debug="true"

contextFactory="com.sun.jndi.ldap.LdapCtxFac
tory"
   hostname="ldap.example.com"
   port="389"
   bindDn="cn=Directory Manager"
   bindPassword="directory"
   authenticationMethod="simple"
   forceBindingLogin="false"
   userBaseDn="ou=people,dc=alcatel"
   userRdnAttribute="uid"
   userIdAttribute="uid"
   userPasswordAttribute="userPassword"
   userObjectClass="inetOrgPerson"
   roleBaseDn="ou=groups,dc=example,dc=com"
   roleNameAttribute="cn"
   roleMemberAttribute="uniqueMember"
   roleObjectClass="groupOfUniqueNames";
   };
```

Writing Your Own

If you want to implement your own custom LoginModule, there are two classes to be familiar with:

AbstractLoginModule.java

```
package org.mortbay.jetty.plus.jaas.spi;

public abstract class AbstractLoginModule
implements LoginModule
{
  ...
  public abstract UserInfo getUserInfo
(String username) throws Exception;
}
```

UserInfo.java

```
package org.mortbay.jetty.plus.jaas.spi;

public class UserInfo
{

  public UserInfo (String userName,
Credential credential, List roleNames)
  {
    ...
  }

  public String getUserName()
  {
    ...
  }

  public List getRoleNames ()
  {
    ...
  }

  public boolean checkCredential (Object
suppliedCredential)
  {
     ...
  }
}
```

The `org.mortbay.jetty.plus.jaas.spi.AbstractLoginModule` implements all of the `javax.security.auth.spi.LoginModule` methods. All you need to do is to implement the `getUserInfo` method to return a `org.mortbay.jetty.plus.jaas.UserInfo` instance which encapsulates the username, password and role names (note: as {{java.lang.String}}s) for a user.

The `AbstractLoginModule` does not support any caching, so if you want to cache `UserInfo` (eg as does the `org.mortbay.jetty.plus.jaas.spi.PropertyFileLoginModule`) then you must provide this yourself.

Example JAAS WebApp

There is an example of authentication and web authorization in the jetty distribution in `examples/test-jaas-webapp`. It uses the PropertyFileLoginModule to perform authentication based on a simple properties file. To use it with the jetty maven plugin:

```
cd examples/test-jaas-webapp
mvn jetty:run
```

Alternatively, to use it instead with jetty standalone:

```
cd examples/test-jaas-webapp
mvn clean install
cd ../../
java -jar start.jar etc/jetty.xml
etc/jetty-jaas.xml
```

Then surf to http://localhost:8080/test-jaas/index.html

Other Goodies

RequestParameterCallback

As all servlet containers intercept and process a form submission with action `j_security_check`, it is usually not possible to insert any extra input fields onto a login form with which to perform authentication: you may only pass `j_username` and `j_password`. For those rare occasions when this is not good enough, and you require more information from the user in order to authenticate them, you can use the JAAS callback handler `org.mortbay.jetty.plus.jaas.callback.RequestParameterCallback`. This callback handler gives you access to all parameters that were passed in the form submission. To use it, in the `login()` method of your custom login module, add the `RequestParameterCallback` to the list of callback handlers the login module uses, tell it which params you are interested in, and then get the value of the parameter back. Here's an example:

FooLoginModule.java

```
public class FooLoginModule extends
AbstractLoginModule
{
        .
        .
        .

     public boolean login()
        throws LoginException
     {
        .
        .
        .
        Callback[] callbacks = new
Callback[3];
        callbacks[0] = new NameCallback();
        callbacks[1] = new ObjectCallback();

        //as an example, look for a param
named "extrainfo" in the request
        //use one RequestParameterCallback()
instance for each param you want to access
        callbacks[2] = new
RequestParameterCallback ();

((RequestParameterCallback)callbacks[2]).set
ParameterName ("extrainfo");
        .
        .
```

```
        .
        callbackHandler.handle(callbacks);
        String userName =
((NameCallback)callbacks[0]).getName();
        Object pwd =
((ObjectCallback)callbacks[1]).getObject();
        List paramValues =
((RequestParameterCallback)callbacks[2]).get
ParameterValues();

        //use the userName, pwd and the
value(s) of the parameter named "extrainfo"
to
        //authenticate the user
        .
```

```
        .
        .
    }
```