Realms

Added by Jan Bartel, last edited by Joakim Erdfelt on May 03, 2010

Realms

Security realms allow you to secure your web applications against unauthorized access. Protection is based on authentication (identifying who is requesting access to the webapp) and access control (restricting what can be accessed and how it is accessed within the webapp).

A webapp statically declares its security requirements in the web.xml file. Authentication is controlled by the `<login-config>` element. Access controls are specified by `<security-constraint>` and `<security-role-ref>` elements. When a request is received for a protected resource, the web container checks if the user performing the request is authenticated, and if the user has a role assignment that permits access to the requested resource.

The Servlet Specification does not address how the static security information in the `WEB-INF/web.xml` file is mapped to the runtime environment of the container. Jetty does this with the "realm" concept.

A realm has a unique name, and is composed of a set of users. Each user has authentication information (e.g. a password) and a set of roles associated with him/herself.

You may configure one or many different realms depending on your needs. A single realm would indicate that you wish to share common security information across all of your web applications. Distinct realms allow you to partition your security information webapp by webapp.

When a request to a web application requires authentication or authorization, Jetty will use the `<login-config>` `<realm-name>` element in the web.xml file to perform an **exact match** to a realm defined in a jetty xml configuration file (or programmatically).

Realm definitions in jetty xml configuration files are placed in a section like this:

```
<Set name="UserRealms">
  <Array type="org.mortbay.jetty.security.UserRealm">
   <Item>
    ...
   </Item>
   <Item>
    ...
   </Item>

   ...

  </Array>
</Set>
```

Alternatively, you may define a realm for just a single webapp in a context deployment file:

```
<Configure class="org.mortbay.jetty.webapp.WebAppContext">
  <Set name="contextPath">/test</Set>
  <Set name="war"><SystemProperty name="jetty.home" default="."/>/webapps/test</Set>
```

```
   ...
  <Get name="securityHandler">
    <Set name="userRealm">
      <New class="org.mortbay.jetty.security.HashUserRealm">
            <Set name="name">Test Realm</Set>
            <Set name="config"><SystemProperty name="jetty.home"
default="."/>/etc/realm.properties</Set>
      </New>
    </Set>
  </Get>
</Configure>
```

Jetty provides a number of different realm types from which you can choose.

HashUserRealm

This realm is a simple realm whose autentication and authorisation information is stored in a properties file. Each line in the file contains a username, a password, and 0 or more role assignments. The format is:

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

where:

- **username** is the user's unique identity
- **password** is the user's (possibly obfuscated or MD5 encrypted) password
- **rolename** is the user's role

For example:

Contents of property file

```
admin: CRYPT:ad1ks..kc.1Ug,server-administrator,content-administrator,admin
other: OBF:1xmk1w261u9r1w1c1xmq
guest: guest,read-only
```

The HashUserRealm is configured with a name and a reference to the location of the properties file:

```
<Item>
  <New class="org.mortbay.jetty.security.HashUserRealm">
     <Set name="name">Test Realm</Set>
    <Set name="config"><SystemProperty name="jetty.home"
default="."/>/etc/realm.properties</Set>
  </New>
</Item>
```

You can also configure it to check the properties file regularly for changes and reload when changes are detected. The `refreshInterval` is in seconds:

```
<Item>
  <New class="org.mortbay.jetty.security.HashUserRealm">
    <Set name="name">Test Realm</Set>
    <Set name="config"><SystemProperty name="jetty.home"
```

```
default="."/>/etc/realm.properties</Set>
    <Set name="refreshInterval">5</Set>
    <Call name="start"></Call>
  </New>
</Item>
```

JDBCUserRealm

In this implementation, authentication and role information is stored in a database accessed via JDBC. A properties file defines the JDBC connection and database table information. Below is an example of a properties file for this realm implementation:

```
jdbcdriver = org.gjt.mm.mysql.Driver
url = jdbc:mysql://localhost/jetty
username = jetty
password = jetty
usertable = users
usertablekey = id
usertableuserfield = username
usertablepasswordfield = pwd
roletable = roles
roletablekey = id
roletablerolefield = role
userroletable = user_roles
userroletableuserkey = user_id
userroletablerolekey = role_id
cachetime = 300
```

The format of the database tables is:

users
id
user
pwd

user_roles
user_id
role_id

roles
id
role

Where:

- **users** is a table containing one entry for every user consisting of:
  - **id** - the unique identity of a user
  - **user** - the name of the user
  - **pwd** - the user's password (possibily obfuscated or MD5 encrypted)
- **user-roles** is a table containing one row for every role granted to a user:
  - **user_id** - the unique identity of the user
  - **role_id** - the role for a user
- **roles** is a a table containing one role for every role in the system:
  - **id** is the unique identifier of a role
  - **role** is a human-readable name for a role

If you want to use obfuscated, MD5 hashed or encrypted passwords the 'pwd' column of the 'users' table must be large enough to hold the obfuscated, hashed or encrypted password text plus the appropriate prefix.

The pseudo-sql to set up each of these tables would look like:

users

```
(
  id integer primary key,
  username varchar(100) not null unique key,
  pwd varchar(50) not null
);
```

user_roles

```
(
  user_id integer not null,
  role_id integer not null,
  unique key (user_id, role_id),
  index(user_id)
);
```

roles

```
(
  id integer primary key,
  role varchar(100) not null unique key
);
```

A JDBCRealm is defined with the name of the realm and the location of the properties file describing the database:

```
<Item>
  <New class="org.mortbay.jetty.security.JDBCUserRealm">
  <Set name="name">Test JDBC Realm</Set>
  <Set name="config">etc/jdbcRealm.properties</Set>
  </New>
  </Item>
```

Please also see the page on JAAS authentication and authorization with Jetty.

Labels: None