

SecurityDesignNotes

Copyright 2004 The Apache Software Foundation

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

* Design Goals:

1. Implement Jetspeed security using JAAS.
2. Use AOP/interception when appropriate for authorization.

Approach:

A user is represented by a combination of Subject
(`java.security.auth.Subject`) and Preferences (`java.util.prefs.Preferences`).

The `UserPrincipal` provide the link between Subject and Preferences as user
preferences are stored under:

```
/user/${UserPrincipal}
```

Authentication

Authentication leverage JAAS.

Jetspeed default implementation provide an `RdbmsLoginModule` that allows
user authentication against user credentials stored in a database.

Upon authentication a Subject is created with the following principals:

- `UserPrincipal` (e.g. `/user/theUsername`)
- `RolePrincipal`
- `GroupPrincipal`

Authorization

* What are we securing:

Many portal resources are being secured: pages, portlets, tabs, etc.

Portal resources have various security policy associated to them for the following type of Principals:

- UserPrincipal
- RolePrincipal
- GroupPrincipal

* How are we securing it:

Securing resources will leverage a custom policy implementation and permissions. The custom policy implementation (o.a.j.security.auth.RdbmsPolicy) will allow to apply policies as follow:

```
grant principal o.a.j.security.UserPrincipal "theUserPrincipal" {
    permission o.a.j.security.PagePermission "mypage", "view";
    permission o.a.j.security.PortletPermission "myportlet",
"view,edit,minimize,maximize";
    permission o.a.j.security.TabPermission "mytab", "view";
};

grant principal o.a.j.security.RolePrincipal "theRolePrincipal" {
    permission o.a.j.security.PagePermission "mypage", "view";
    permission o.a.j.security.PortletPermission "myportlet",
"view,edit,minimize,maximize";
    permission o.a.j.security.TabPermission "mytab", "view";
};

grant principal o.a.j.security.GroupPrincipal "theGroupPrincipal" {
    permission o.a.j.security.PagePermission "mypage", "view";
    permission o.a.j.security.PortletPermission "myportlet",
"view,edit,minimize,maximize";
    permission o.a.j.security.TabPermission "mytab", "view";
};
```

* The following permission are anticipated:

TODO

* How do we access it:

Given that a user is authenticated that the current subject is the one of that user, we would check for permission doing:

```
Subject.doAs(loginContext.getSubject(), new PrivilegedAction()
{
    public Object run()
    {
        PortletPermission perm1 = new PortletPermission("myportlet", "view");
        System.out.println("\t\t[TestRdbmsPolicy] Check perm1.");
        AccessController.checkPermission(perm1);
    }
});
```

SecurityDesignNotes

```
        return null;
    });
}
```

* Configuring the new policy

The java security provider is set at runtime through the `Policy.setPolicy()` static access.

Check the `TestRdbmsPolicy` and javadoc for more info.

* The RDBMS policy implementation:

The assumption is made that policies will always look like:

```
grant principal ...
```

Therefore the RDBMS policy implementation will grant access for all classes.

Codebase support and `CodeSource` support is NOT implemented.

* Link between the Security and Preferences services:

The link between the Security service and the Preferences services is performed using `security_principal.name` and `pref_node.full_path`

The name of a user principal in the security layer should match the full path of that user preferences root in the prefs layer:

e.g: `/user/theUserPrincipal`

* Managing groups and roles:

Use `java.util.prefs.Preferences` to store roles and groups as user preferences.

```
<preferences EXTERNAL_XML_VERSION="1.0">
<root type="user">
<map />
  <node name="group1">
    <map />
      <node name="groupid1.1">
        <map />
          <node name="groupid1.1.1">
            <map />
            </node>
          </node>
        </node>
      </node>
    </node>
  <node name="role1">
    <map />
    <node name="roleid1.1">
```

```
<map />
  <node name="roleid1.1.1">
    <map />
    </node>
  </node>
</node>
</root>
```

This defines the following group and role hierarchy:

/group1/groupid1.1/groupid1.1.1

and

/role1/roleid1.1/roleid1.1.1

Where map can define group or role custom properties

E.g: For roles, we could have a rule custom property (or a pointer to a rule) that allow rule based role definition tied to some rule engine (Drools for instance) and is validated when the isInRole method is invoked.

For group, a portal could use group to describe organization and have custom property such as address, city, etc associated with the organization/group.

Readings:

<http://java.sun.com/j2se/1.4.2/docs/api/index.html>

<http://www.ufr-info-p7.jussieu.fr/hf/jdk1.3/docs/guide/security/Acl.html>

<http://www.theserverside.com/resources/AspectJreview.jsp>

<http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-spec.doc2.html>

<http://www-106.ibm.com/developerworks/library/j-jaas/?n-j-442>