Login into Geronimo

Login Service API

Login into Geronimo is mediated by the **Login Service** implemented by the `org.apache.geronimo.security.server.JaasLoginService` gbean. To login into the server is to establish a **security session** with the Login Service. Login Service will perform authentication based on the application security requirements. Application security requirements are enforced by the **security realm**. Client application tells Geronimo server the name of the security realm it wants to use. Refer to the Geronimo and JAAS section for a discussion on the name parameter passed by the client to the security implementation.

It is useful to look briefly at the org.apache.geronimo.security.server.JaasLoginService API:

- **JaasSessionId connectToRealm(String realm-name)**
  This method allows client to select the security realm and to initiate security session. **Security-session-id** is returned to the client.
  Client is supposed to hold on to it.
- **JaasLoginModuleConfiguration[] getLoginConfiguration(JaasSessionId session-id)**
  Return array of **login module configurations** for the security realm associated with the **session-id**.
  JaasLoginModuleConfiguration includes login module name, control flag, login module options, etc.
- **boolean performLogin(JaasSessionId session-id, int login-configuration-idx, Callback[] callbacks)**
  Perform login for the login module in the security realm associated with the session-id.
  Login module is selected by the login-configuration-idx.
  Callbacks is the array of Callback objects required by the login module and populated by the client.
- **boolean performCommit(JaasSessionId session-id, int login-configuration-idx)**
  Commit login results the login modules in the security realm associated with the session-id.
  Login module is selected by the login-configuration-idx parameter.
- **Principal loginSucceed(JaasSessionId session-id)**
  Signal that overall login succeeded for the **security session**. This method returns **IdentificationPrincipal** for the session.
- **Set synchPrincipals(JaasSessionId session-id, Set principals)**
  Synchronize principals between **client-side subject** and **session subject**. All principals from the client subject are added to the session subject and
  serializable principals from the session subject are added to the client subject.

Any client that has a reference to the JaasLoginService gbean can use it's API to log into Geronimo.

Login into Geronimo the portable way: JaasLoginCoordinator

Client side in the Geronimo login process is represented by the `org.apache.geronimo.security.jaas.client.JaasLoginCoordinator` login module. **JaasLoginCoordinator** drives login process using the **Geronimo Login Service API** and computes authentication result based on the JAAS login module combination semantics.

As a **Login Module**, JaasLoginCoordinator can be configured in the remote client or in the Geronimo server for use by the locally deployed component (such as a servlet).

JaasLoginCoordinator is invoked by the JAAS framework (as any other login module would be) in a sequence of **`initialize`**, **`login`**, and **`commit`** calls.

Next you can at each step more in detail.

JaasLoginCoordinator.initialize()

On initialization step, JaasLoginCoordinator connects to the Login Service etc. Because JaasLoginCoordinator represents authentication client, it keeps it's own Subject instance.

JaasLoginCoordinator.login()

Step 1: Initiate security session with the Login Service by calling connectToRealm(realmName); Realm name is passed as an input parameter from the application.
A new **security session** is started by the Login Service and saved in the **active-logins** map. Security session identifier is returned to the **JaasLoginCoordinator**.

Two notes should be made here:

- One is that **the security session holds an instance of it's own Subject** (distinct from the Subject held in the JaasLoginCoordinator). This Subject will be populated with principals from the login modules configured into security realm.

- The other is that each `JaasLoginModuleConfiguration` within security session contains `wrapPrincipals` boolean flag. If set to true, the login module (**Login Domain**) will be wrapped with the special **login module proxy**. The type of this proxy is `org.apache.geronimo.security.jaas.WrappingLoginModuleProxy` and it has special behavior within it's commit() method. It produces additional **Principals** that hold association of a **principal to the login domain** and a **principal to the security realm**.

In particular `org.apache.geronimo.security.DomainPrincipal` will be added for every Principal instance instantiated by the original login module (login domain) and RealmPrincipal will be added for each DomainPrincipal when login module is committed.

Step 2: Based on the **security-session-id** retrieve an array of JAAS login module configurations wired into the security realm by calling: `JaasLoginModuleConfiguration[] getLoginConfiguration(JaasSessionId session-id)`. For further details refer to #JaasLoginModuleConfiguration in the Login Service API section.

Step 3: Having to account for the remote and local scenarios, the `JaasLoginCoordinator` wraps each login-module in the `JaasLoginModuleConfiguration[]` array it got from the `JaasLoginService` with the `LoginModuleProxies`. `LoginModuleProxies` are login modules themselves (obviously). `LoginModuleProxy` is sub-classed with the `ServerLoginProxy` and `ClientLoginProxy`. `ClientLoginProxy` is further sub-classed by the `WrappingClientLoginProxy`.

We are now going to concentrate on the `ServerLoginProxy`. In keeping with the JAAS API, login modules wrapped by the login module proxies are invoked with the **`initialize`**, **`login`**, and **`commit`** sequence.

There are several details you have to keep in mind about this.

- The Subject instance passed to the `initialize()` method for every login module proxy is `JaasLoginCoordinator` owned Subject instance (representing Subject on the **client side**).
- Callback handler is passed by the client that initiated login procedure (for example a servlet).
- Shared state for login modules is synchronized between `JaasLoginCoordinator` (the client side) and `JaasLoginService` (the server side) at the end of initialization loop.

Step 4: Let the login procedure begin! Here is the place where the **JAAS login module semantic** is actually enforced by comparing the result of the `login()` method call for each login module proxy and login module configuration control flag. For further details on this procedure refer to the Geronimo and JAAS section.

Note that this computation is done by the **`JaasLoginCoordinator`** which is **authentication client** and not by the `JaasLoginService` itself.

Now we are going to look into what happens within the `ServerLoginProxy.login()` method. There is an array of `ServerLoginProxies[]` that correspond to the array of `JaasLoginModuleConfigurations[]` retrieved from the `JaasLoginService`. Each `ServerLoginProxy` is constructor-injected with the login-module control-flag, client-side Subject, `JaasLoginModuleConfiguration` array index, a reference to the JaasLoginService and **security-session-id**.

`ServerLoginProxy.login()` method first retrieves an array of `Callbacks[]` from the `JaasLoginService` that are configured for the corresponding login module in the **security realm**:
`Callback[] LoginService.getServerLoginCallbacks(security-session-id, login-module-index)`.
We leave it out to figure out how it is done. The important thing at this time is that you can pass this callback array to the **callback-handler** (injected during **`initialize()`** method call and supplied by the authentication client (see above)). `callback-handler.handle(Callbacks[])` populates server callbacks array with client data.

Now `ServerLoginProxy.login()` method asks the `JaasLoginService` to perform the actual login by passing it the security-session-id, login-module-configuration index, and an array of populated callbacks. As a result, **security-session** is retrieved from the **active-logins** map, and corresponding login module (configured in the **security realm** under login module index) is invoked to perform the login.

A point to note here is that security realm login modules are initialized at the time when server-side callbacks are retrieved by the `ServerLoginProxy` in preparation for login. (Not an obvious place to look). All information to the security realm login module comes from the security session (it is on the server-side of course).

It looks like we are logged in, or at least close...

JaasLoginCoordinator.commit()

If overall authentication succeeds (according to the security realm policy), `JaasLoginService.commit()` is called. `Login-module-proxy.commit()` is called for every proxy in the login module proxy array. It is here that all principals in the security realm login modules are collected (and possibly wrapped into the `DomainPrincipal` and `RealmPrincipal`) and then added to the **Subject** in the **security session** (server-side). At the end of the commit-loop, `Principals` between `JaasLoginCoordinator Subject` (**client-side**) and **security session Subject** (**server-side**) are synchronized. `Principals` from the `JaasLoginCoordinator Subjects` are **added** to the **security session Subject** (in case of the server-side `JaasLoginCoordinator` this is an empty set) and serializable `Principals` from the security session Subject are added to the `JaasLoginCoordinator Subject`.

At the very end `JaasLoginCoordinator.commit()` method notifies the `JaasLoginService` of login success: `LoginService.loginSucceed(security-session-id)`. As a result, `JaasLoginService` registers it's session Subject with the `ContextManager` and generates a **subject-id** based on the Subject. It then wraps this subject-id into the `IdentificationPrincipal`, adds it to the set of `Principals` in the `Subject` and returns `IdentificationPrincipal` to the `JaasLoginCoordinator`.

`JaasLoginCoordinator` adds `IndentificationPrincipal` into it's own Subject.

Authentication complete!!!