Using jLock's JAAS in Distributed Application Environments

The **Using jLock's Java Authentication and Authorization Service (JAAS) for Application-Level Security** white paper provides a detailed overview of how to use jLock's scalable JAAS implementation within a single Java program. Many applications, however, use distributed technologies such as Java's Remote Method Invocation (RMI), Java Messaging Service (JMS) or CORBA to access business logic in remote servers. The server process may need to restrict access or filter information based upon the identity of an authenticated user. This requires that the server process have access to the credentials of the user who was authenticated by the client process. It is rare for a distribution technology to support the transparent delegation of user identity and/or credentials to the server process. This is a task that typically must be managed by the business application. jLock extends the JAAS permission paradigm to distributed environments and allows applications to transmit the authenticated identity information securely (even if the environment does not support encrypted data transmission).

2AB's jLock implementation of JAAS allows the client to obtain an identity token that can be passed to the server and used to re-establish the login context of the authenticated user. This ensures that permissions are consistent across the distributed environment. The identity token is encrypted and can be safely passed between processes (using RMI, JMS, SOAP, CORBA, FTP or any other messaging infrastructure) without the overhead of encrypting the entire message payload if desired. An iLock token can only be obtained after a user has authenticated and has a short lifespan (a few minutes). Passing an identity token to a server is more secure and more efficient than re-authenticating on the server.

We demonstrate this technique with a simple human resources RMI server that supports an Interface that returns a Compensation object. One of the attributes of the Compensation object is "salary." The client authenticates the user exactly as before (for a complete explanation of JAAS authentication, see the white paper referenced above). The two JAAS methods applications need to invoke to use a JAAS authenticator are shown in **bold** font.

```
LoginContext lc = null;
try {
    lc = new LoginContext("JaasDemo", new DialogCallbackHandlerUP());
} catch (LoginException le) {
    …
} catch (SecurityException se) {
   …
}
try {
      lc.login();
} catch (LoginException le) {
    System.out.println("\nAuthentication failed:");
     …
}
System.out.println("\nHello iLock World!\n");    // Authentication Succeeded!
```

JAAS RMI Client Authenticating User (HRRmiClient.java)

The client then obtains an identity token and passes that token to the RMI server when requesting compensation information for an employee (identified by "empl_id").

```
byte[] token = com.twoab.jaas.Context.getToken();

try {
   Compensation co = hr.getCompensation(empl_id, token);
   …
}   ...
```

Code to Create Token and Pass to Remote Object in RMI Server (HRRmiClient.java)

In this example, the RMI server uses a filtering technique. The requested Compensation object is returned but filters out the salary if the user that is logged in is not authorized to see that information. The other attributes of compensation (i.e. the job code of the employee) are available to the user. The implementation of the remote object is shown in the two boxes below. First, the token is used to set the context for the permission check to that of the remote user.

```
public class CompensationRmiImpl extends PortableRemoteObject
    implements CompensationRmiInterface {

    public Compensation getCompensation(int id, byte[] token) {

    int jobcode=2501;       // jobcode and salary would normally be obtained from HR database
    String salary = “$75,000”;

    Compensation co = new Compensation(id, jobcode, salary);

    try {
      ctx = new com.twoab.jaas.Context(token);
    } catch (com.twoab.jaas.ContextException ex) {
        …
    }
```

Code to Set Context to Match the Remote User (CompensationRMIImpl.java)

The server then makes the check to determine if access to Salary is allowed to the user. If the user is allowed to see Salary information, then the salary amount will remain in the Compensation object that is being returned to the client. If not, then salary will be filtered by placing a “Not Authorized” message in the salary attribute.

```
    try {

            ResourcePermission p = new ResourcePermission("Salary");
            AccessController.checkPermission(ctx, p);

            System.out.println("Access to Salary Info is granted");
    }
          catch (com.twoab.jaas.AccessControlException ace) {
             System.out.println("Sorry - Access to SalaryInfo is denied");

             co.setSalary("User is Not Authorized for Salary Info”);
    }
    …
    return co;
}
```

Code to Check Permission for Salary and Filter Salary Attribute (CompensationRMIImpl.java)

jLock allows the JAAS APIs to be useful for both local and remote Java environments. If your environment also spans other programming languages, talk to 2AB about companion products that allow your application security to remain consistent even when your application spans services written in different programming languages. Our goal is to make application security easy to understand and easy to implement.

For more information, please visit our Web site at http://www.2ab.com or e-mail info@2ab.com.