



Using jLock's Java Authentication and Authorization Service (JAAS) for Governance-Based Access Control (GBAC)

August 2005
www.2ab.com

Introduction

Access management is a simple concept. Every business has information that needs to be protected from unauthorized disclosure. To protect information, companies define policies - often mandated by legislation - that govern who can access specific classes of business and/or personal information. For example, if a police investigator seeks to access transaction data related to a suspected terrorist from a private bank, they should have authorization to do so, however, they should not be authorized to access the same information about someone who is not suspected of any illegal activity. That is, there is a policy that specifically governs the release of information collected about individuals to other individuals. Or is there? The answer today is: "Probably not." What does exist are written policies (or a law) related to disclosure of broad classes of business and personal information. But, often, individual data is not specifically classified within organizations. Requests for information flow through individuals within an organization. The policy is enforced only because human beings are skilled at generalizations; that is, we expect someone in authority to be able to classify an ad hoc request for a particular piece of information and make a decision.

Access Management software has a simple goal. It allows the human who previously acted as a guardian of sensitive information to be removed from the process without loss of access control. This sounds simple, but most businesses are struggling with the implementation of access management as they integrate and extend their applications. This is because machines cannot classify information or make access decisions unless they are explicitly programmed with algorithms to accomplish this. When you take the responsibility for access decisions away from human beings, it becomes necessary to insert software guards into your applications.

CGI has recently released a whitepaper, *"Governance-Based Access Control: Enabling improved information sharing that meets compliance requirements,"* introducing a new model for access control. This model, called Governance-Based Access Control (GBAC), is focused on the classification of information assets for the purpose of information sharing in an environment where:

- Many organizations may require access to information
- Information may be accessed by, or shared with, external users
- Everyone may be subject to compliance with multiple authorities and jurisdictions

In a previous whitepaper, *"Using jLock's Java Authentication and Authorization Service (JAAS) for Application-Level Security,"* we described a systematic approach to using JAAS to manage the complexity associated with software access management. We explained the JAAS service-oriented architecture (SOA), which maintains a clean separation of concerns between application functionality and access management, and discussed the types of JAAS features that a scalable JAAS implementation should support.

In this whitepaper, we explore the Governance-Based Access Control (GBAC) model and use the scenarios presented in the CGI whitepaper as the basis of the sample access policy and the coding examples. We show how jLock, 2AB's commercial JAAS implementation, can be used to support the complex rule requirements of the GBAC model. In a complementary whitepaper, we explore an alternative model, the OMG's Resource Access Decision (RAD) facility, to support GBAC. We hope these papers will help you understand the GBAC model and how existing standards and tools can be leveraged to implement it.



What is the Java Authentication and Authorization Service?

The Java Authentication and Authorization Service (JAAS) defines the standard programming interface for building these software guards in a Java environment. Prior to JAAS, security mechanisms in Java were strictly code-based. That is, you granted permissions based on the code that was running – there was no way to grant permissions based on the identity (or credentials) of the user of the application. For this reason, any user-based access control mechanisms had to be coded directly into the business application (typically requiring new database tables and/or directory infrastructure). When access policy or audit requirements changed, application software had to be modified, tested and redeployed. Additionally, when access policy needs to be examined, or applications audited for conformance, a code review was required.

Access management solutions, such as commercial implementations of JAAS, provide scalable alternatives to the costly embedding of access control mechanisms and access policy. They allow application software guards to leverage services that enable access policy to be modified, tested and deployed dynamically without application code changes. This enables your developers to concentrate on providing business software. Access management solutions efficiently enable high performance access controls in distributed environments while allowing centralized management of access policy. Any commercial access management solution includes application programming interfaces (APIs) and policy management tools. JAAS defines these APIs for the Java environment.

Application security must address any security-related requirements not provided by the runtime security infrastructure. In the area of access management, any requirement to restrict a) the usage of application features or b) access to business and personal information is part of “application security.” Often these restrictions on access to sensitive information are based on legislation. For this reason, the Governance-Based Access Control Model is being progressed as a means to classify information for the purpose of assigning access policy. The CGI whitepaper, "Governance-Based Access Control (GBAC)," provides an excellent overview of this model and is the basis of the examples in this whitepaper.

There are many excellent overviews of JAAS on Sun’s JavaSoft Web site. For that reason, we will assume the reader has some familiarity with the JAAS model, and we will focus on how the JAAS model can be leveraged to provide the fine-grain access control requirements of “application security” in an environment that uses the Governance-Based Access Control (GBAC) model.

JAAS consists of two parts: Authentication and Authorization

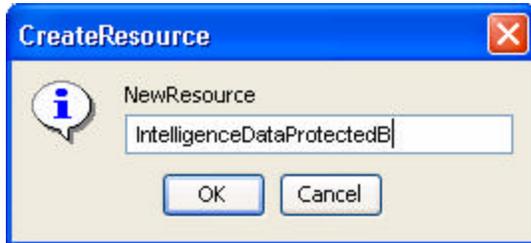
- Authentication answers the question: “How do I know that you are who you say you are?” The goal of authentication is to securely determine who is executing Java code, regardless of whether or not the code is part of a standalone Java application, a servlet, an applet or an Enterprise Java Bean.
- Authorization answers the question: “Now that I know who you are, how do I know if you are allowed to access the information or application feature that you are requesting?” The goal of authorization is to protect business and personal information and sensitive application features from being used by people who legitimately have access to the application and some subset of its functionality.

There are a number of papers from various sources that focus on how to build an implementation of JAAS. These papers explain in detail the concepts, design center and classes that are used in a JAAS implementation. The classes include LoginContext, LoginModule, CallbackHandler, Subject, Principal, Permission and AccessController. We will discuss those concepts in this paper as they become visible to the Java programmer but will not discuss details of their implementation. This paper will focus on how a Java developer uses a commercial implementation of JAAS in conjunction with the GBAC model. We will introduce the jLock AccessManager which extends the JAAS model to support the context-sensitive policy requirements of GBAC. We will also explore how user identity and access policies are managed using jLock Security Center administrative tools. We will outline the steps you need to take to use jLock’s JAAS with the GBAC model.



Classification of Information

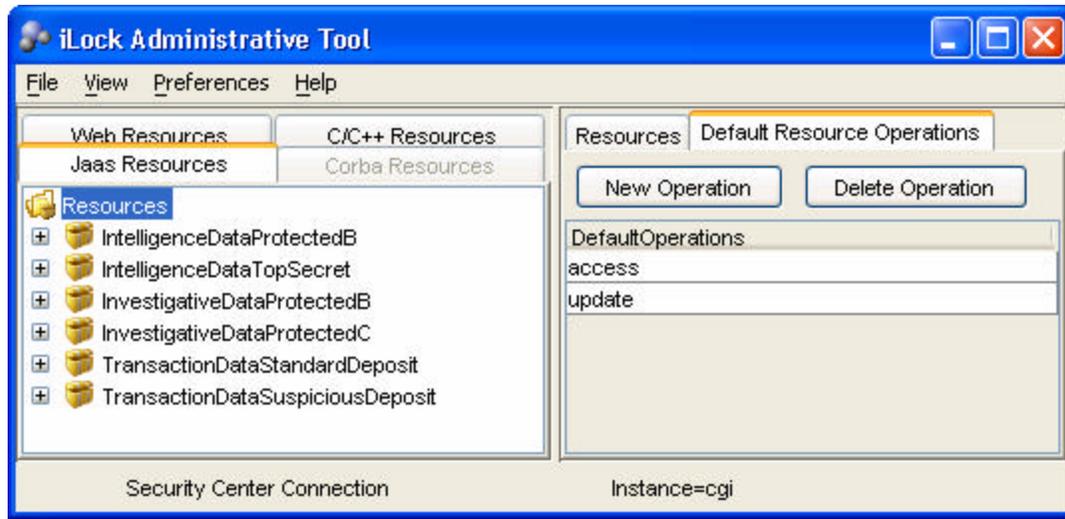
The classification of information assets is based on Governance attributes as outlined in Table 1 of the GBAC whitepaper. Each classification of information is mapped to a JAAS resource or Permission. In the Java Authentication and Authorization Service (JAAS) specification, this is a unique string. You would create resources for each possible classification that needs to be protected using the **Create Resource** Editor as shown below.



A unique string would need to be defined for each information classification.

We also support a more complex naming scheme if you want to give each classification a fully qualified structural name. To do this, you would use the OMG's Resource Access Decision (RAD) facility naming scheme. In addition to using the graphical user interface, classifications may be entered programmatically or via scripting.

In the screen shot shown below, we have defined a resource for each information classification listed in Table 1 of the GBAC Whitepaper.



Viewing the GBAC Information Classifications as JAAS Resources

This user interface is designed to precisely match the Java Authentication and Authorization Service where Resource Permissions are defined as a string.



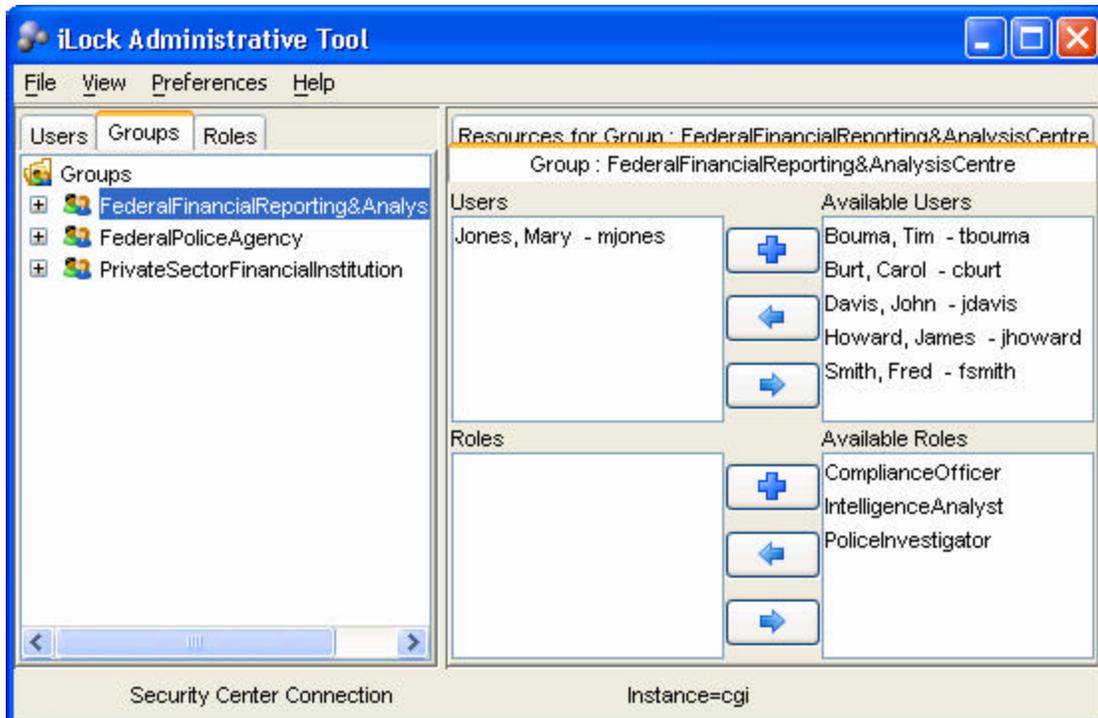
Classification of People Who May be Required to Access Information

The classification of people in GBAC is based on roles. These roles are defined in Figure 4 in the GBAC whitepaper. In the Java Authentication and Authorization Service, a role is a type of Principal that may be assigned to Subjects or Groups. The following shows the roles as defined in the GBAC Whitepaper for each organization.



Examples of GBAC Roles

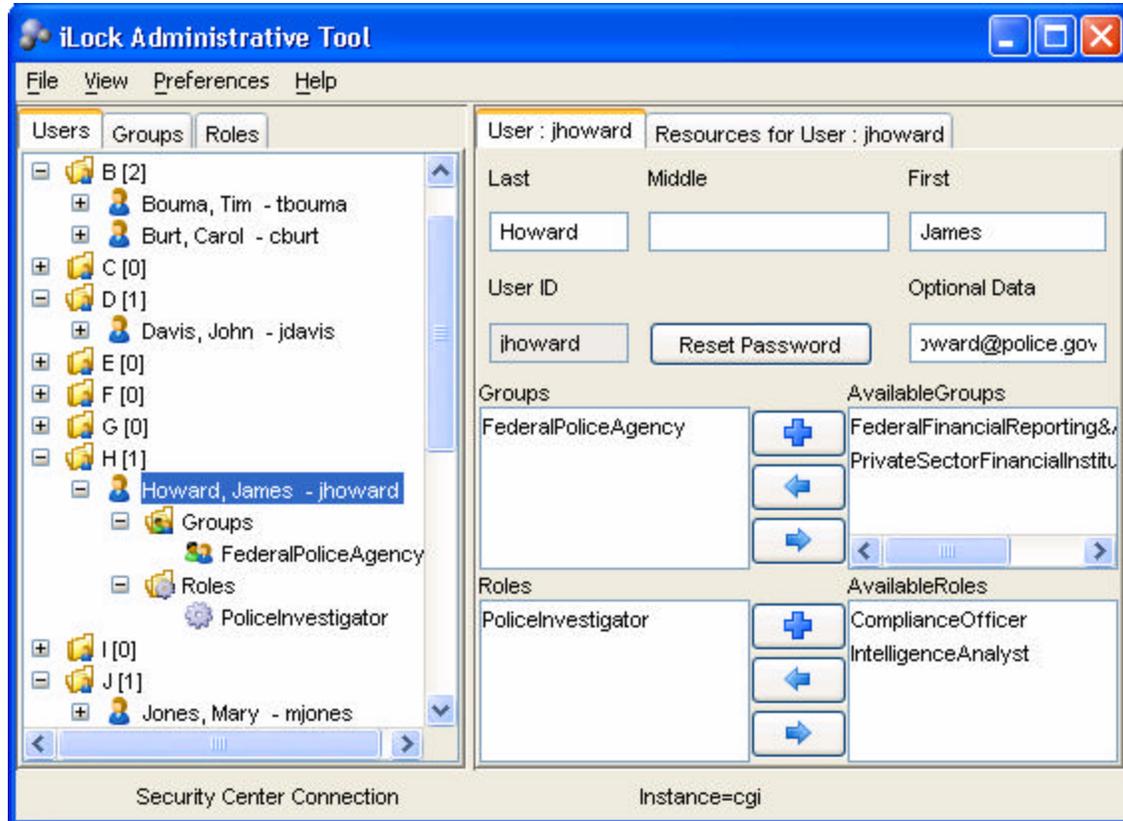
If desired, groups could also be defined as shown below:



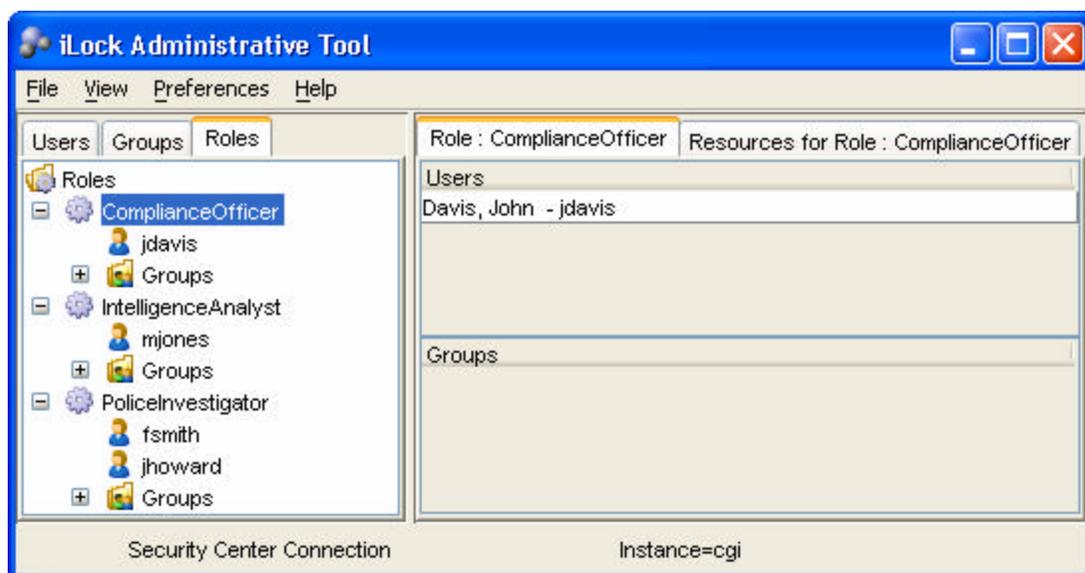
Examples of GBAC Groups



Of course, you will also need to define Users. The Identity Manager in jLock also allows you to create users and assign passwords. Password format requirements can be set using the **Preferences** menu. We can use the Identity Manager **Users** tab to assign roles to our users. We assign GBAC roles as follows: John Davis is a Compliance Officer, Mary Jones is an Intelligence Analyst and James Howard is a Police Investigator.



James Howard has the Role PoliceInvestigator and is in the Group FederalPoliceAgency



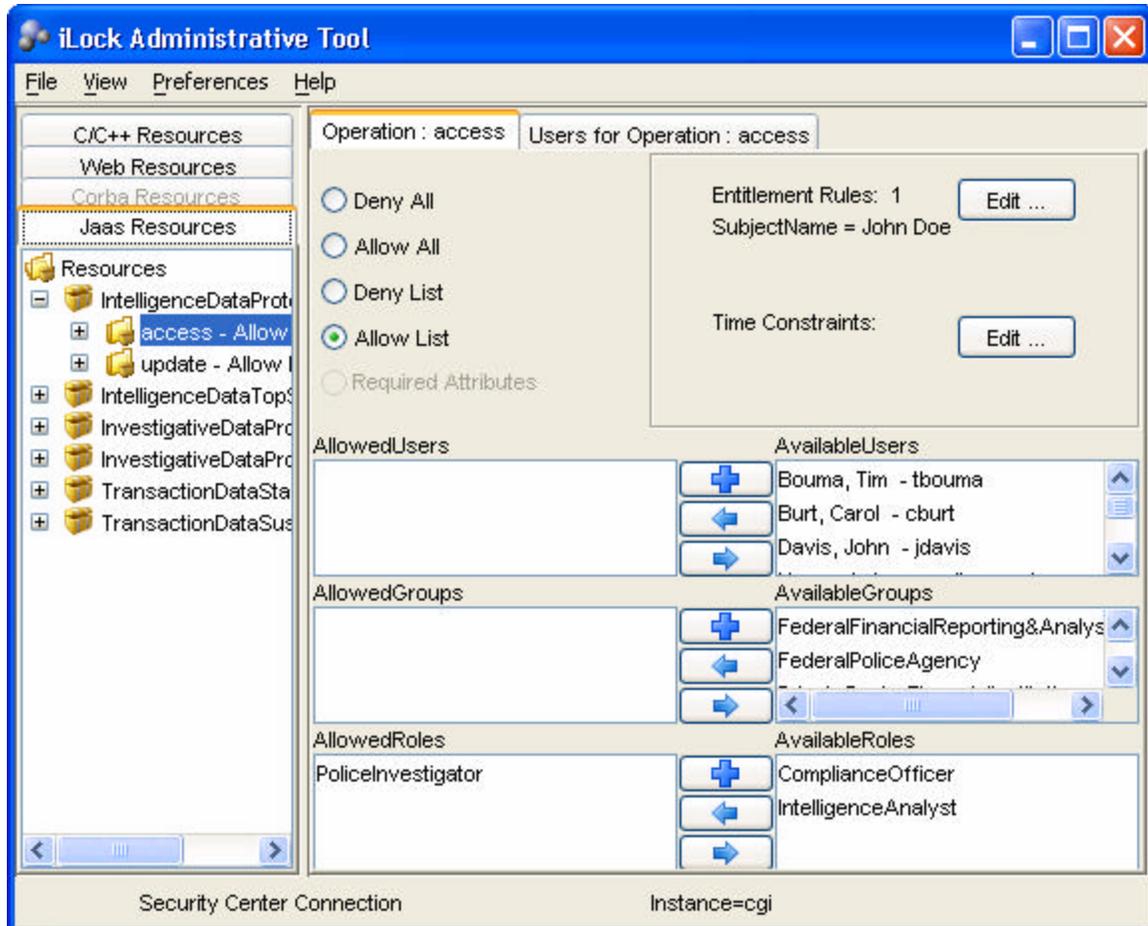
The Role view allows you to see who is assigned a Role



Definition of GBAC Rules for Information Access

Access Control rules are defined in GBAC based upon the roles of individuals and context information related to the information. The rules we define below are outlined in Table 2 of the GBAC whitepaper.

By selecting a Resource (these are the GBAC classifications) operation, you can modify the policy as shown below. Note that jLock allows you to specify rules for different operations on Resources (the GBAC classification). For example, you see that there are access and update rules which may be defined differently.



Because GBAC requires context information be provided at the time of the access request, an entitlement rule is added. Entitlement rules allow the program to pass context information at the time of the access request. This information is evaluated as part of the policy. For example, the scenario in the GBAC whitepaper has two different examples of this. Some of the rules are scoped to information regarding someone with the Subject Name of "John Doe" as shown above. In another GBAC rule, the transaction amount of a deposit must be greater than \$10,000. This type of policy is an example of what jLock calls entitlement rules. The JAAS model of authorization has no mechanism for supporting this type of rule. For this reason, jLock has extended JAAS to enable complex, context-sensitive and entitlement-based policies. Entitlement rules are supported in the Power Edition of jLock.



Here is how the entitlement rule would be specified using the Entitlement Editor:

Name	Value	Value Type	Relationship
SubjectName	John Doe	String	Equal

OK Cancel

In English the policy defined in the two screens shown above would read:

Anyone who has been assigned the role "PoliceInvestigator" is allowed access (to the named resource) with the constraint that the "SubjectName" (which is of data type String) is Equal to "John Doe."

To create multiple constraints, you must define multiple entitlement rules. The demo that we are providing also includes policy for access to the resource TransactionDataSuspiciousDeposit that includes multiple entitlement rules as follows:

Anyone who has been assigned the role "IntelligenceAnalyst" is allowed access (to the named resource) with the constraint that the "TranType" (which is of data type String) is Equal to "Cash" and the "Amount" (which is of data type Decimal) is Greater Than 10000.

The Entitlement rule would be defined in the editor as shown below.

Name	Value	Value Type	Relationship
TranType	Cash	String	Equal
Amount	10000	Decimal	Greater Than

OK Cancel

Example of defining multiple entitlements to an access rule



Using JAAS Authentication as Part of a GBAC Solution

JAAS authentication is based on the Pluggable Authentication Module (PAM) architecture. Leveraging an architecture that supports ‘plug-ins’ for authentication ensures that Java applications can be independent of the underlying authentication mechanism. This has the advantage that new or revised authentication mechanisms can be plugged in without modifying the application code. That is, management of User IDs and Passwords (or other methods of authentication) are removed from the application’s concern. For this example, we will leverage the dialog-based User ID and Password authenticator that is supplied with the jLock product.

The first thing you need to do is specify the JAAS implementation that you are using. This is done with a login configuration file. This may be done on the command line when you invoke your application.

```
java -Djava.security.auth.login.config=config.txt ...
```

The **jaas_config.txt** file, supplied with the example, is shown below. It specifies an application name (JaasDemo) and the jLock plug-in class for the LoginModule. We are also specifying the instance name of the Security Center repository that holds identity and policy information.

```
/** Login Configuration for the GBAC JAAS Demo Applications **/  
  
JaasDemo  
{  
    com.twoab.jaas.LoginModuleUP required instance="cgi";  
};
```

JAAS Login Configuration File (jaas_config.txt)

This is the Java code for a class that prints “Hello iLock World” if user authentication succeeds. The two JAAS methods your application needs to invoke to use a JAAS authenticator are shown in **bold font**.

```
public HelloJAAS() {  
    LoginContext lc = null;  
  
    /** Create a LoginContext object. */  
    try {  
        lc = new LoginContext("JaasDemo", new DialogCallbackHandlerUP());  
  
    } catch (LoginException le) {  
        System.out.println("Cannot create LoginContext. " + le.getMessage());  
        System.exit(1);  
    } catch (SecurityException se) {  
        System.out.println("Cannot create LoginContext. " + se.getMessage());  
        System.exit(-1);  
    }  
  
    try {  
        lc.login();  
    }  
    catch (LoginException le) {  
        System.out.println("\nAuthentication failed:");  
        System.out.println(" " + le.getMessage());  
        System.exit(1);  
    }  
    System.out.println("\nHello iLock World!\n");  
    .....
```

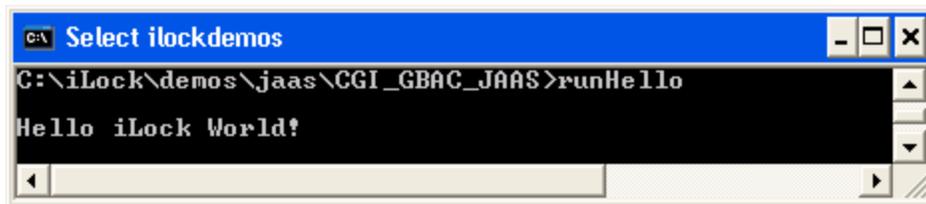
JAAS Authentication Code Sample (HelloJAAS.java)

That is all the code and configuration you need! When you run the example (runDialog.bat), at the point where the **lc.login()** is called, the following dialog will appear.



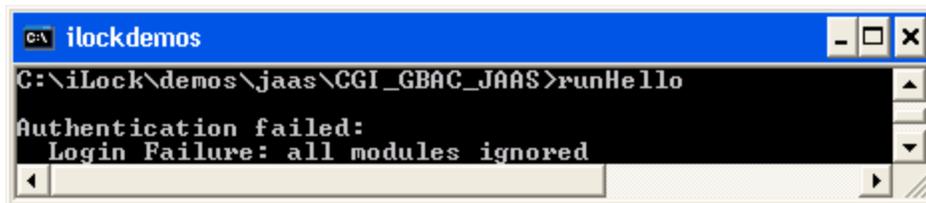
Type in a User ID and Password as shown above and click **OK**. jLock will authenticate the user.

Assuming you typed a valid User ID and Password, the example program results will, as you might expect, look like the following:



Authentication Succeeded

Of course, if you should fail to provide a valid User ID and/or Password, you will see this:



Authentication Failed

The HelloJAAS demo program has obviously written no Java code to manage Users or Passwords, or to do the work required to authenticate the user (in this case verify the password). That is the great thing about the JAAS architecture; just “plug in” jLock, and it securely manages all that for you! jLock also ensures that the password is never available in clear text. jLock securely stores and transmits password information - even if you are not using an encrypted transport protocol.



Now we are ready to explore JAAS authorization. To understand the JAAS Authorization model, you must first understand a little more about what happens when you authenticate using JAAS. When the user (*mjones* in the example above) was authenticated, a **Subject** object was created. A Subject represents the entity that was authenticated – that is, the entity that has been able to prove their identity. A Java **Principal** is a “security attribute” or “credential” that can be associated with one or more Subjects. During the authentication process, the jLock authenticator acquired the credentials of the Subject and associated them with the subject by creating the appropriate Principal objects. A user (i.e. Subject) may always be able to prove their identity, but their credentials (i.e. Principals) may change over time. For this reason, security access policy is defined in terms of the security attributes (or in Java terminology Principals) that are associated with the Subject at the time identity was authenticated. jLock supports three types of Principals: 1) AccessIdPrincipal, 2) RolePrincipal and 3) GroupPrincipal. These map to the UserIds, Groups and Roles shown in the Identity Manager. These are the fundamental building blocks of access policy. In the section above, you can see that the user, Mary Jones, has the following jLock security attributes.

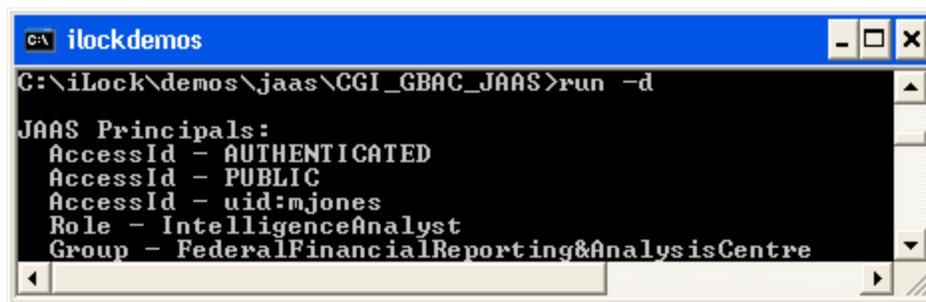
- AccessId: *mjones*
- Role: *IntelligenceAnalyst*
- Group: *FederalFinancialReporting&AnalysisCentre*

If you add the following code to the example, you can see that the LoginContext allows navigation to a Subject that manages a set of Principals.

```
.....
    java.util.Set prin_set = lc.getSubject().getPrincipals();
    java.util.Iterator it = prin_set.iterator();
    while (it.hasNext() == true) {
        java.lang.Object obj = it.next();
        if (obj instanceof AccessIdPrincipal) {
            System.out.println("AccessId - " +
                ((AccessIdPrincipal)obj).getName();
            )
        }
        else if (obj instanceof GroupPrincipal) {
            System.out.println("Group - " +
                ((GroupPrincipal)obj).getName();
            )
        }
        else if (obj instanceof RolePrincipal) {
            System.out.println("Role - " +
                ((RolePrincipal)obj).getName();
            )
        }
        else {
            System.out.println("Unknown principal type");
        }
    }
}
```

Code to display the names of the Principals associated with the authenticated Subject

Running with this code, you will see the output below following authentication:





Extending JAAS Authorization for GBAC

The JAAS Authorization model extends the code-centric, Java security architecture that uses a security policy to specify what access is granted to executing code (such as access to files, sockets or specific operations). The extension allows security access policy to be defined based on the credentials associated with the user of the code. Just as a commercial JAAS Authentication may be plugged in, the JAAS Authorization model also allows vendors to offer commercial solutions that offer scalability, management and enhanced support for sophisticated access policy.

There are limitations in the JAAS Authorization model in Sun's reference implementation. For example, Sun's reference implementation requires that grant statements that define access policy be placed in policy files for each user and that the application use the Java Security Manager (in the same way that grant statements and policy files are used for code-centric security). Since it obviously is not practical (or secure) to manage user-based access policy in local, plain-text files for a large user community, JAAS providers such as 2AB offer solutions that allow identity and access policy to be managed separately from the application. Sun's reference implementation also requires that any code that requires user-based access control be placed in a separate class and executed only via Subject.doAs (or doAsPrivileged) methods. That sets the scope of the user-based software guard to the class where the sensitive code is located. jLock does not preclude the use of the do.As operations for access management but does support the insertion of software guards that use the JAAS Principal-based authorization model without the requirement to segment the code into separate classes. Notice that while we can certainly run this application with the Java Security Manager installed (adding a few permissions to the java.policy file), this demo does not require the Security Manager to leverage the jLock JAAS features. You simply insert your sensitive code in a try block and check for the appropriate permission before running it. Remember, you are not checking whether the code has access to the resource, you are only checking whether or not the application should provide the resource to the user.

jLock supports the use of the JAAS AccessController for checking access permissions and also provides a more powerful AccessManager that enables entitlement-based policies (such as those defined in the GBAC whitepaper) to be supported. Note that after the user has authenticated, it is still necessary to determine if the user has permission to access IntelligenceDataProtectedB information. The code snippets below show use of the AccessController and the more powerful AccessManager.

```
String gbac_class = new String("IntelligenceDataProtectedB");
try {
    ResourcePermission p = new ResourcePermission(gbac_class);
    AccessController.checkPermission(p);
    System.out.println("Access to " + gbac_class + " Info is granted");
}
catch (com.twoab.jaas.AccessControlException ace) {
    System.out.println("Sorry - Access to " + info_class + " Info is denied");
}
```

**Code to protect access to the "IntelligenceDataProtectedB" information
using a JAAS AccessController**

```
EntitlementData ed = new EntitlementData[1];
ed[0] = new EntitlementData("SubjectName", "John Doe");
String gbac_class = new String("IntelligenceDataProtectedB");

JaasResource jr = new JaasResource (gbac_class);
if (am.accessAllowed(jr, "access", lc.getSubject(), ed)) {
    System.out.println("Granted access to " + jr.toString() + " Info");
} else {
    System.out.println("Denied access to " + jr.toString() + " Info ");
}
```

**Code to protect access to the "IntelligenceDataProtectedB" information
using a jLock AccessManager with Entitlement context data**



It is that simple to add GBAC authorization to your application. We have provided simple demonstration programs as well as a full prototype so that you can run to see how this works. When you run the simple demo and authenticate using *mjones* as the User ID, you will see the following dialog showing the resources that Mary Jones, an IntelligenceAnalyst is allowed to access (sample code sets subject John Doe, transaction type of cash and amount of \$12,000).

```
C:\ iLockdemos
C:\iLock\demos\jaas\CGI_GBAC_JAAS>runJaasAM -d

JAAS Principals:
  AccessId - AUTHENTICATED
  AccessId - PUBLIC
  AccessId - uid:mjones
  Role - IntelligenceAnalyst
  Group - FederalFinancialReporting&AnalysisCentre

Checking permission to access GBAC classified Information

Granted access to IntelligenceDataProtectedB Info
Granted access to IntelligenceDataTopSecret Info
Granted access to InvestigativeDataProtectedB Info
Denied access to InvestigativeDataProtectedC Info
Denied access to TransactionDataStanadardDeposit Info
Granted access to TransactionDataSuspiciousDeposit Info
```

Mary Jones, an IntelligenceAnalyst, requesting access to John Doe's information

However, if you run the demo and authenticate using *jhoward* as the User ID, you will see that James Howard, a PoliceInvestigator, is allowed to access the following classes of information:

```
C:\ iLockdemos
C:\iLock\demos\jaas\CGI_GBAC_JAAS>runJaasAM -d

JAAS Principals:
  AccessId - AUTHENTICATED
  AccessId - PUBLIC
  AccessId - uid:jhoward
  Group - FederalPoliceAgency
  Role - PoliceInvestigator

Checking permission to access GBAC classified Information

Granted access to IntelligenceDataProtectedB Info
Denied access to IntelligenceDataTopSecret Info
Granted access to InvestigativeDataProtectedB Info
Granted access to InvestigativeDataProtectedC Info
Denied access to TransactionDataStanadardDeposit Info
Granted access to TransactionDataSuspiciousDeposit Info
C:\iLock\demos\jaas\CGI_GBAC_JAAS>
```

James Howard, a Police Investigator, requesting access to John Doe's information

All of the tools for creating and managing the access policy are provided by jLock. The jLock architecture supports dynamic policy updates that take effect immediately, so applications do not need to be restarted when access policies change. jLock also supports remote policy administration.



[A GBAC Demonstration Based on the CGI Whitepaper](#)

Next we want to show how you might build an application that integrates intelligence data, investigative data and transaction data while only displaying the information that a user is authorized to view. We call this demonstration program the “Terrorist Information Portal.” The source code for the demonstration program below is freely available.

When Mary Jones, an Intelligence Analyst, logs into the system, she is provided access to information sorted by the agency that owns it. Note that she is only allowed to view information that she is authorized to see. That is, there may be more information in the database, based on the GBAC rules, that she is not authorized to view. That is, the portal dynamically constructs the user view based upon the results of consultation with a GBAC access controller. To change the information available, no code is written – the GBAC rules are simply changed using the graphical administration tools provided to the security administrator.

The screenshot shows a web application window titled "Terrorist Information Portal". The user is Mary Jones, and the current view is "InfoClass: Transaction Data" as of "Fri Sep 02 16:22:11 CDT 2005".

On the left is a tree view of agencies:

- AGENCIES
 - Private Sector Financial Institution
 - TransactionDataSuspiciousDeposit**
 - Federal Police Agency
 - InvestigativeDataProtectedB
 - Federal Financial & Reporting Center
 - IntelligenceDataProtectedB
 - IntelligenceDataProtectedB
 - IntelligenceDataTopSecret

The main content area shows details for the selected agency: "Agency: Private Sector Financial Institution: PSFI-0001". There are tabs for "Stats" and "Demographics".

Key statistics shown:

- Bank: CitiCorp
- Officers: 50
- Revenue: 15B
- Rating: 93

The "Type" is "TransactionDataSuspiciousDeposit". Below this is a table of transactions:

SubjectName	Trantype	Amount	Date
John Doe	Cash	\$100,000	11/14/2004
John Doe	Cash	\$11,000	11/10/2004
John Doe	Cash	\$12,000	12/14/2004
John Doe	Cash	\$14,200	1/24/2005
John Doe	Cash	\$13,000	11/4/2004
John Doe	Cash	\$11,000	11/10/2004
John Doe	Cash	\$16,200	11/4/2004
John Doe	Cash	\$17,200	11/4/2005

Information Authorized for Mary Jones, an Intelligence Analyst

Here is the line of code (Guard) that is inserted into the application to determine whether or not to display an agency in the tree. A similar access control check (which also passes entitlement information, such as the SubjectName and Amount) is made on each document type to determine the documents to displayed under the agency. In this way, the access policy remains separate from the application and can be modified dynamically using the policy administrative tools shown earlier in this paper.

```
boolean view = true;  
view = am.accessAllowed(agency.toResource(),operation,lm.getSecurityAttributes());
```

Code in prototype to determine whether or not an agency can be viewed by the user



Below we see the same portal when John Davis, a Compliance Officer, is logged in. Notice that information is not shared with the Private Sector Financial Institution and so John's view does not even include those agencies.

The screenshot shows the 'Terrorist Information Portal' interface for user John Davis. The user is logged in as 'John Davis' and the 'InfoClass' is 'Transaction Data'. The date and time are 'Fri Sep 02 17:13:51 CDT 2005'. The left sidebar shows a tree view under 'AGENCIES' with 'Private Sector Financial Institution' expanded, containing 'TransactionDataStandardDeposit' (selected) and 'TransactionDataSuspiciousDeposit'. The main content area shows details for 'Agency: Private Sector Financial Institution: PSFI-0001'. There are tabs for 'Stats' and 'Demographics'. The 'Stats' tab is active, showing 'Bank: CitiCorp', 'Officers: 50', 'Revenue: 15B', and 'Rating: 93'. Below this, the 'Type' is 'TransactionDataStandardDeposit'. A table lists transaction data for 'John Doe' with columns for 'Subject Name', 'Trantype', 'Amount', and 'Date'.

Subject Name	Trantype	Amount	Date
John Doe	Cash	\$1,000	11/4/2004
John Doe	Cash	\$1,000	11/10/2004
John Doe	Check	\$1,200	12/4/2004
John Doe	Cash	\$1,200	1/4/2005
John Doe	Cash	\$1,200	11/4/2004
John Doe	Cash	\$1,000	12/10/2004
John Doe	Check	\$1,200	8/4/2004
John Doe	Cash	\$1,200	9/4/2005

Information Authorized for John Davis, a Compliance Officer

And finally we look at the view of a Police Investigator.

The screenshot shows the 'Terrorist Information Portal' interface for user James Howard. The user is logged in as 'James Howard' and the 'InfoClass' is 'Intelligence Data'. The date and time are 'Fri Sep 02 17:02:25 CDT 2005'. The left sidebar shows a tree view under 'AGENCIES' with 'Federal Financial & Reporting Center' expanded, containing 'IntelligenceDataProtectedB' (selected) and 'IntelligenceDataProtectedB'. The main content area shows details for 'Agency: Federal Financial & Reporting Center: FFRAC-0001'. There are tabs for 'Stats' and 'Demographics'. The 'Stats' tab is active, showing 'Uniformed Officers: 250', 'Special Forces: 0', 'Undercover: 15', and 'Rating: 100'. Below this, the 'Type' is 'IntelligenceDataProtectedB'. A text block shows subject information: 'SubjectName: John Doe', 'Address: 1700 Hwy 31', 'Sex: M', 'City: Calera', and 'State: AL'.

Information Authorized for James Howard, a Police Investigator



Summary

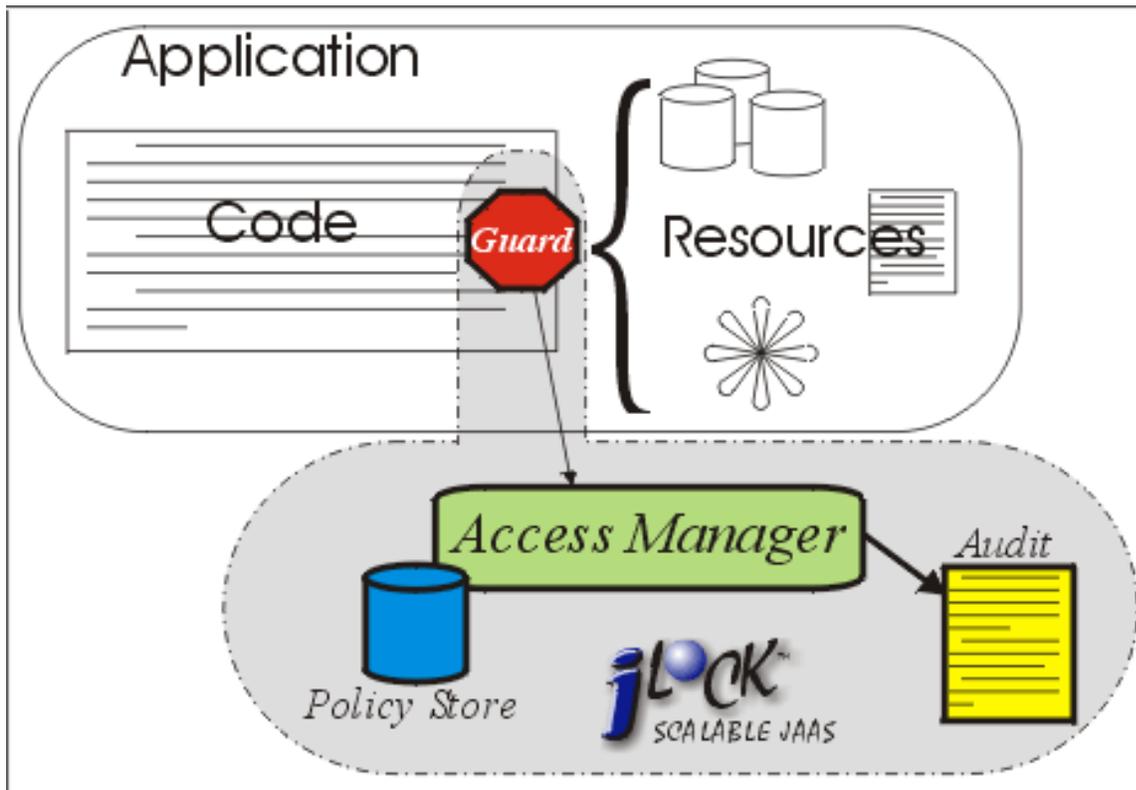
In this paper we have outlined how the jLock JAAS service can be used to implement a Service Oriented Architecture for application-level security that meets the unique requirements of Governance-Based Access Control as defined by CGI in their GBAC whitepaper.

The trend towards a service-oriented architectural approach to dealing with application-level security is evident in recent analyst reports. For example:

META Group predicted in late 2003: "as businesses begin to put more focus on design for application securability and service oriented architecture, application-specific security mechanisms will migrate to infrastructure."

A JAAS implementation such as jLock provides APIs that enable you to authenticate and easily integrate access control checks within your business applications. JAAS supports a pluggable architecture that allows you to select your JAAS vendor based upon your requirements for authentication and access policy support.

Utilizing JAAS, your business developers simply insert AccessController or AccessManager calls (Software Guards) at the points in the software where sensitive resources are exposed. This Guard consults with the jLock Access Manager who evaluates the policy and advises the Guard on allowing access.



The JAAS architecture enables many different policy models to be leveraged by a Java business application.

JAAS supports a service-oriented architecture for authentication and authorization



Challenge 2AB!

Are you still not sure if jLock can help with your GBAC requirements? Challenge us to prove it. Send us four or five examples of your access management requirements. We'll configure jLock with policies you can use and send you an evaluation copy of jLock, complete with a working demo so you can see how to leverage jLock within your application. We'll even send you the source code for the demo so your development staff can take a look at exactly how little we had to do to insert a guard! Go ahead... challenge us. What have you got to lose – an increasingly difficult access management problem?

2AB, Inc.
1700 Highway 31
Calera, Alabama 35040
877.334.9572 (toll-free)
challenge@2ab.com