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

September 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 pharmacist seeks to access clinical data (such as medications) related to a person for whom they are filling a prescription, they should have authorization to do so, however, they should not be authorized to access the same information about a co-worker. There are many written policies (often laws) related to disclosure of broad classes of clinical and personal information. But, often, individual data is not specifically classified within electronic patient record systems. Today, most requests for patient information flows through individuals within a provider or payer organization; the access policy that governs release of the information is enforced only because human beings are skilled at generalizations; that is, they are able to classify an ad hoc request for a particular piece of information about a patient and make a decision about whether or not it should be released to the requestor.

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 providers and payers are struggling with the implementation of access management as they integrate and extend their clinical and administrative 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.

Most companies now have governance councils that ensure that they are in compliance with legislated requirements. To do this, sensitive information must be classified. In addition, the people who may need access to the information need to be grouped (or assigned roles) for the purpose of establishing consistent, auditable access policy. One company who assists health organizations in this effort - 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 Governance-Based Access Control (GBAC) concepts using scenarios from the healthcare domain. We show how jLock, 2AB's commercial JAAS implementation, can be used to support the complex rule requirements of the Healthcare GBAC model. We hope this paper and the demonstration programs will help you understand how JAAS can be leveraged for access management.

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.

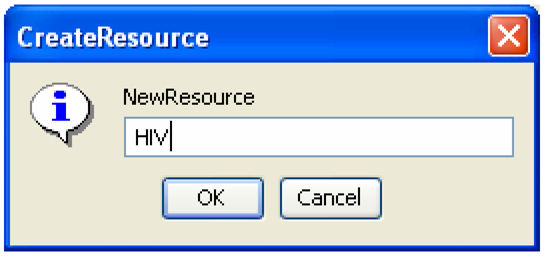
JAAS consists of two parts: Authentication and Authorization

- 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 based on Governance attributes is a key concept of the GBAC model. 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 **CreateResource** Editor as shown below.

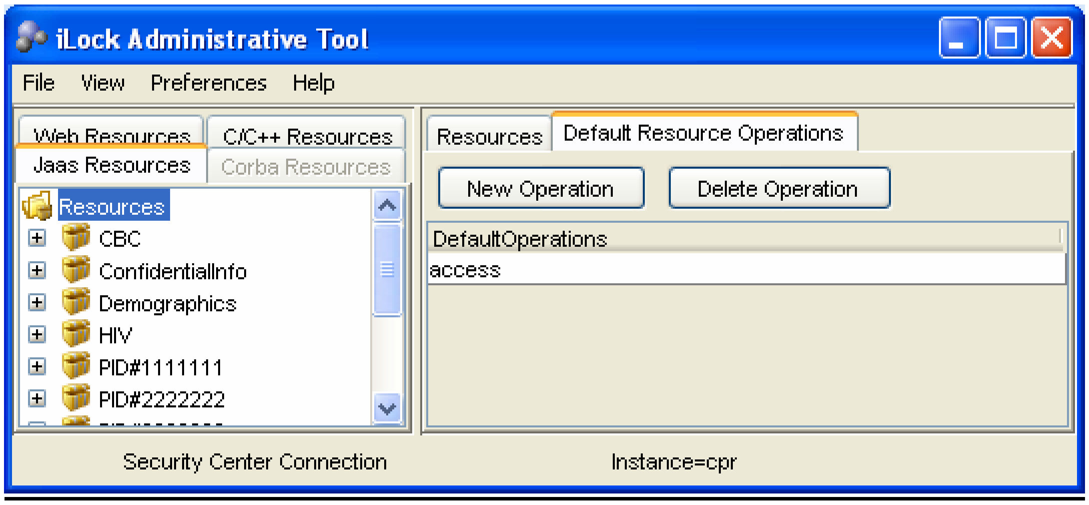


We are using generic classifications in this example; in reality you might use terminology from a medical vocabulary.

A unique string would need to be defined for each information classification. As you will see later, you can specify operations (such as "order" or "results") to further classify the information asset.

In addition to using the graphical user interface, classifications may be entered programmatically or via scripting.

We also support a more complex naming scheme if you want to give each classification a fully qualified structural name. This might be useful if using medical vocabularies to classify information. To do this, you would use the OMG's Resource Access Decision (RAD) facility naming scheme. For some applications, the complex naming is easier to understand. Here we will simply use JAAS permissions which are strings.

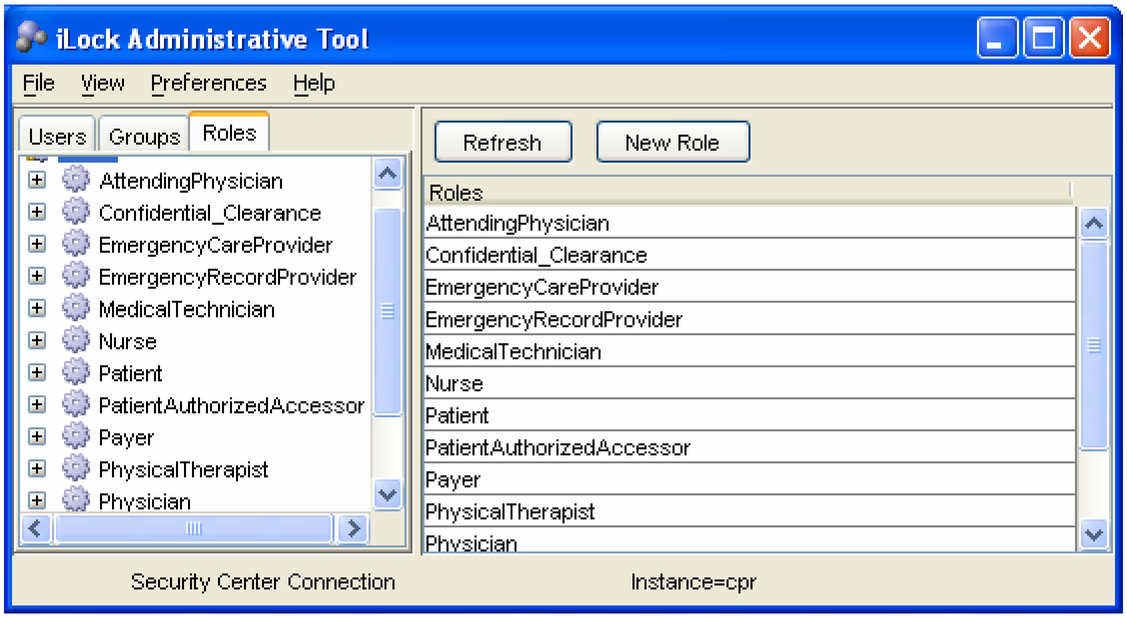


Viewing the GBAC Information Classifications as JAAS Resources

This user interface is designed to precisely match the Java Authentication and Authorization Service where Resource Permissions are defined as a string. We extend JAAS to allow multiple operations on a Resource. For example, for a lab tests, you may set your operations as "order" and "reviewResults." For demographic information, the only operation may be "access."

Classification of People Who May be Required to Access Information

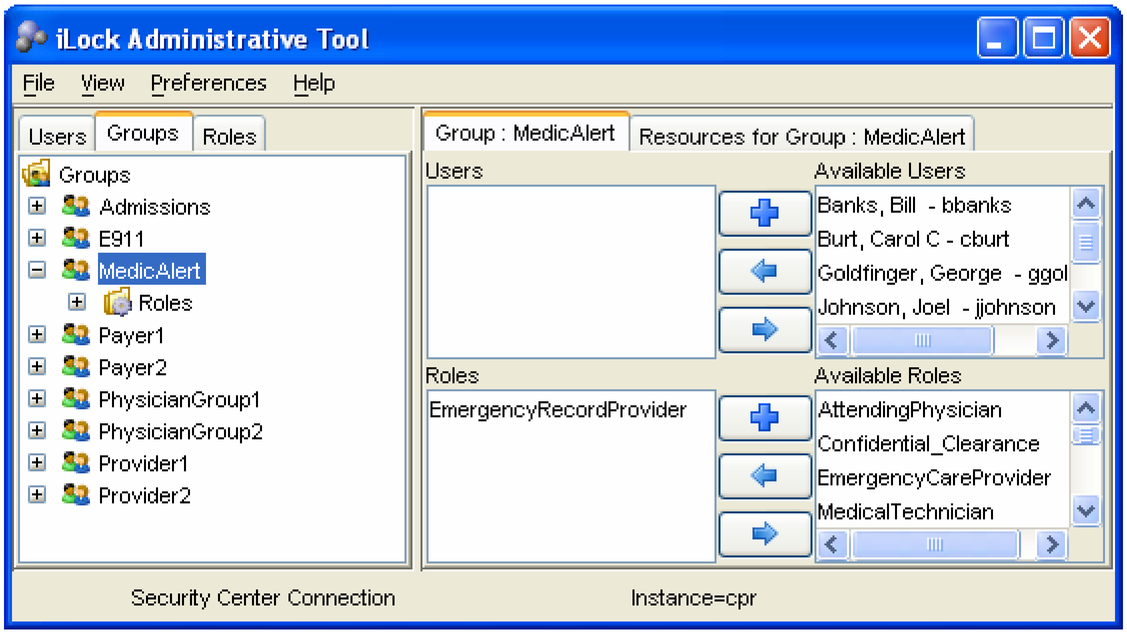
The classification of people in GBAC is based on roles. These roles are defined by the healthcare organization. In the Java Authentication and Authorization Service, a role is a type of Principal that may be assigned to Subjects or Groups. The following shows the roles used in this demonstration.


iLock Administrative Tool
File View Preferences Help
Users Groups Roles
Refresh New Role
Security Center Connection
Instance=cpr

Roles
AttendingPhysician
Confidential_Clearance
EmergencyCareProvider
EmergencyRecordProvider
MedicalTechnician
Nurse
Patient
PatientAuthorizedAccessor
Payer
PhysicalTherapist
Physician

Examples of Healthcare Roles

Groups can also be very useful, and roles may be assigned to Groups as shown below.


iLock Administrative Tool
File View Preferences Help
Users Groups Roles
Group : MedicAlert
Resources for Group : MedicAlert
Security Center Connection
Instance=cpr

Groups
Admissions
E911
MedicAlert
Roles
Payer1
Payer2
PhysicianGroup1
PhysicianGroup2
Provider1
Provider2

Users

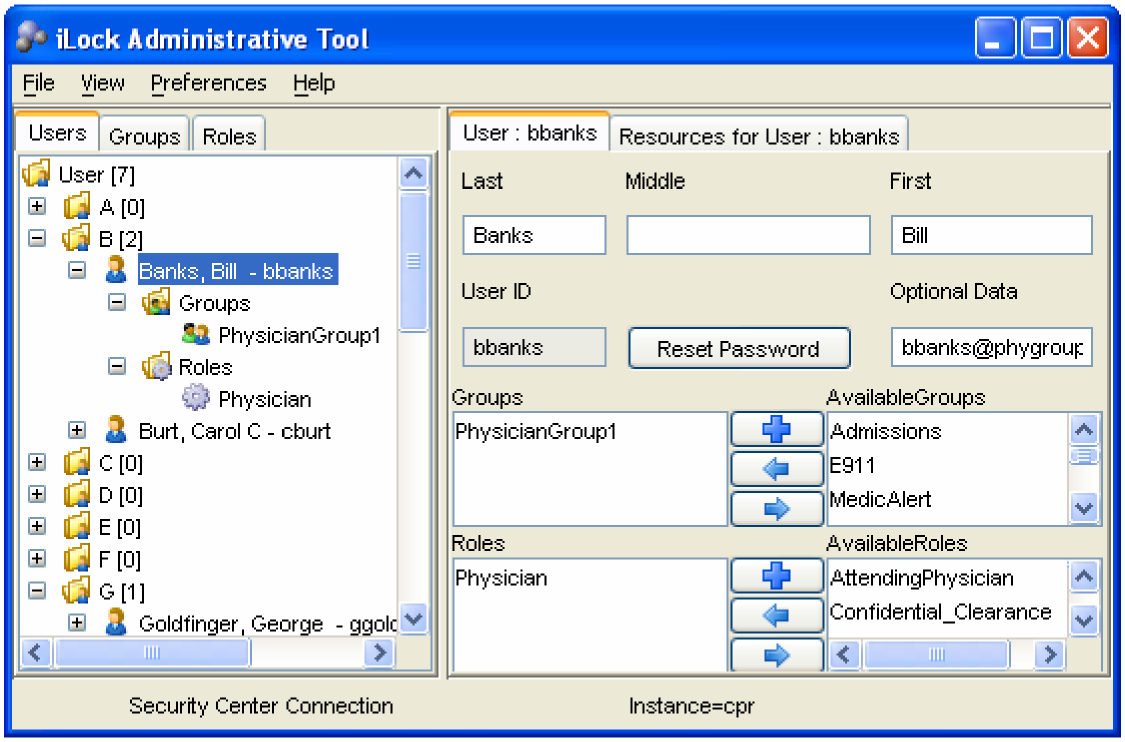
Available Users
Banks, Bill - bbanks
Burt, Carol C - cburt
Goldfinger, George - ggol
Johnson, Joel - jjohnson

Roles
EmergencyRecordProvider

Available Roles
AttendingPhysician
Confidential_Clearance
EmergencyCareProvider
MedicalTechnician

Assigning roles to groups
EmergencyRecordProvider role will be assigned to anyone in the Group MedicAlert

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. For this demo, we assign roles as follows: Bill Banks is a Physician, Ed Harris is an EmergencyRecordProvider (because of association with group MedicAlert) and Nancy Nurse is a Nurse and an EmergencyCareProvider.



User [7]
- A [0]
- B [2]
  - Banks, Bill - bbanks
    - Groups
      - PhysicianGroup1
    - Roles
      - Physician
  - Burt, Carol C - cburt
- C [0]
- D [0]
- E [0]
- F [0]
- G [1]
  - Goldfinger, George - ggolc

Last	Middle	First
Banks		Bill

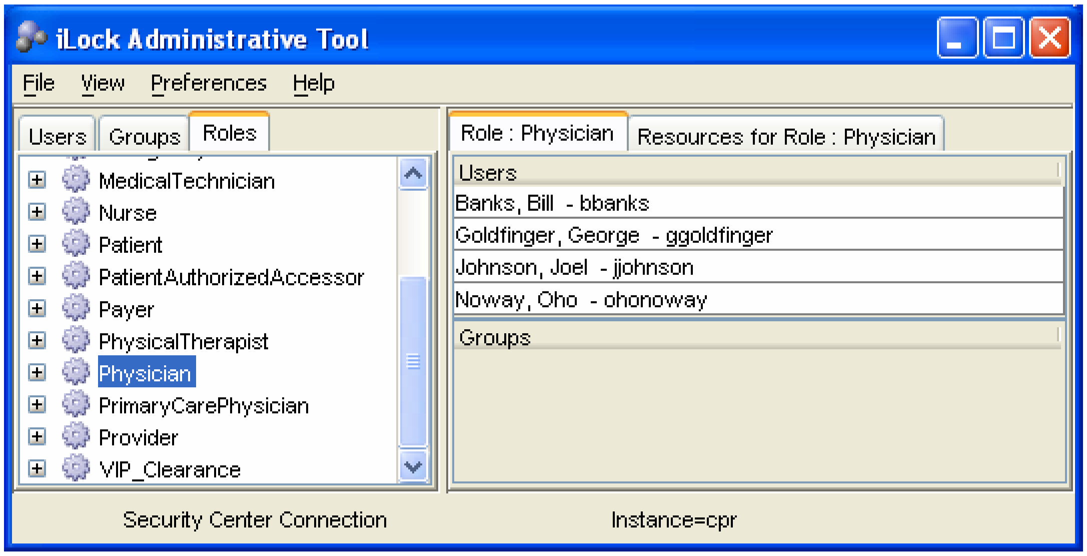
User ID	Optional Data
bbanks	bbanks@phygroup

Groups	AvailableGroups
PhysicianGroup1	Admissions
	E911
	MedicAlert

Roles	AvailableRoles
Physician	AttendingPhysician
	Confidential_Clearance

Security Center Connection Instance=cpr

Bill Banks has the Role Physician and is in the Group PhysicianGroup1



- MedicalTechnician
- Nurse
- Patient
- PatientAuthorizedAccessor
- Payer
- PhysicalTherapist
- Physician
- PrimaryCarePhysician
- Provider
- VIP_Clearance

Users
Banks, Bill - bbanks
Goldfinger, George - ggoldfinger
Johnson, Joel - jjohnson
Noway, Oho - ohonoway

Groups

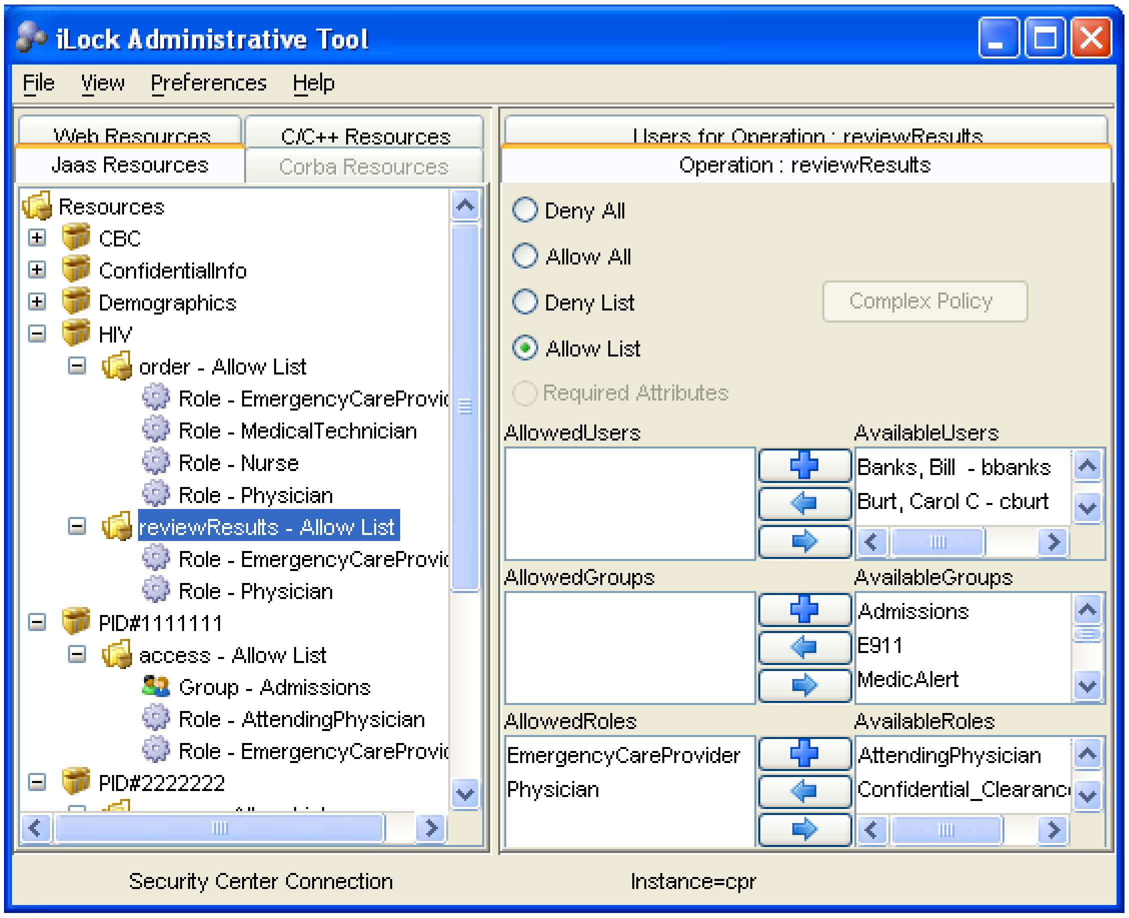
Security Center Connection Instance=cpr

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

Definition of GBAC Rules for Information Access

Access Control rules may be defined in Governance-Based Access Control (GBAC) based upon the roles of individuals, relationships between the requestor and the information source (i.e. attending physician) and/or the context within which information is requested (i.e. emergency). These rules will have to be defined for each health organization.

By selecting a Resource (information classification) operation, you can modify the policy as shown below. Note that jLock allows you to specify rules for different operations on Resources. For example, you see below that there are order and reviewResults operations on the HIV resource. These operations can be customized for each Resource (class of information).



Resources
- CBC
- ConfidentialInfo
- Demographics
- HIV
  - order - Allow List
    - Role - EmergencyCareProvic
    - Role - MedicalTechnician
    - Role - Nurse
    - Role - Physician
  - reviewResults - Allow List
    - Role - EmergencyCareProvic
    - Role - Physician
- PID#1111111
  - access - Allow List
    - Group - Admissions
    - Role - AttendingPhysician
    - Role - EmergencyCareProvic
- PID#2222222

Operation : reviewResults

Deny All
Allow All
Deny List
[✓ Allow List]
Required Attributes

AllowedUsers	AvailableUsers
	Banks, Bill - bbanks
	Burt, Carol C - cburt
AllowedGroups	AvailableGroups
	Admissions
	E911
	MedicAlert
AllowedRoles	AvailableRoles
EmergencyCareProvider	AttendingPhysician
Physician	Confidential_Clearanc

Because GBAC often requires decisions to be based on context or relationship-based information that can only be evaluated at the time of the access request (as it may change dynamically), our demonstration will make use of a dynamic attribute service. Dynamic attributes are "groups" or "roles" that are assigned dynamically to a user at the time of the access request. For example, there may be information that is provided to an "attending physician" that is not available to other healthcare workers (even if they are physicians). In this demonstration, physicians will be provided a view that only includes the patients for which they are the attending physician. The access policy for patient information is based on the AttendingPhysician role. This role is dynamically associated with the physician at the time of the access request. Because dynamic attribution of roles and/or groups is a part of the access management service, they are easily audited and maintained separate from the business application. Access policy remains simple, easy to understand, administer and audit.

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="cpr";
};
```

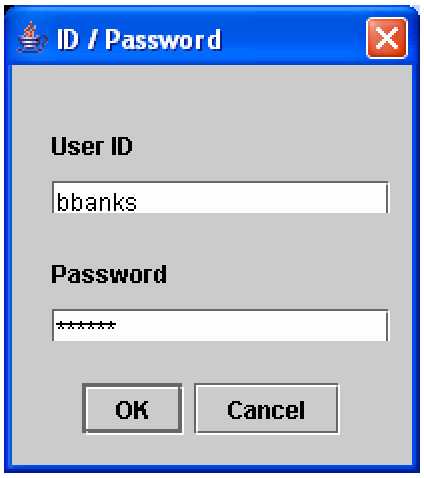
JAAS Login Configuration File (jaas_config.txt)

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");                    ......
```

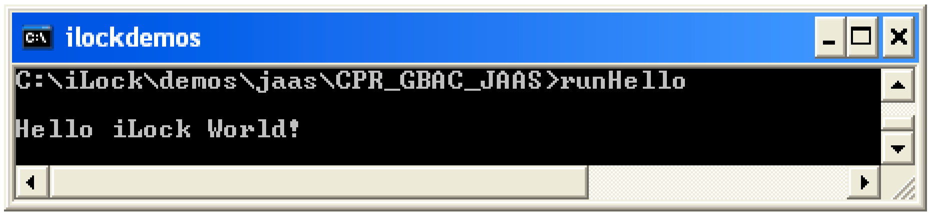
JAAS Authentication Code Sample (HelloJAAS.java)

That is all the code and configuration you need! When you run the example (runHello.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.

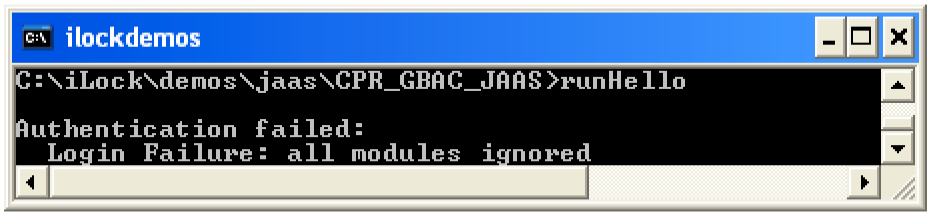
Assuming you typed a valid User ID and Password, the example program results will, as you might expect, look like the following:



```
C:\iLock\demos\jaas\CPR_GBAC_JAAS>runHello

Hello iLock World!
```

Authentication Succeeded

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



```
C:\iLock\demos\jaas\CPR_GBAC_JAAS>runHello

Authentication failed:
  Login Failure: all modules ignored
```

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 (*bbanks* 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, Bill Banks, has the following jLock security attributes.

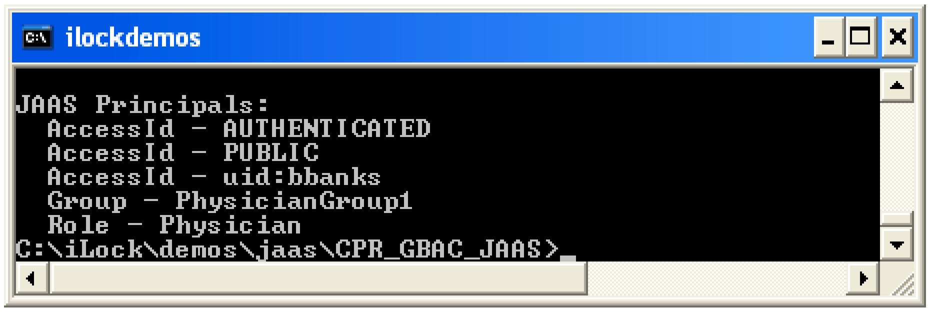
- AccessId: *bbanks*
- Role: *Physician*
- Group: *PhysicianGroup1*

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:



```
JAAS Principals:
  AccessId - AUTHENTICATED
  AccessId - PUBLIC
  AccessId - uid:bbanks
  Group - PhysicianGroup1
  Role - Physician
C:\iLock\demos\jaas\CPR_GBAC_JAAS>
```

Extending JAAS Authorization for CPR 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 relationship-based and context sensitive polices to be supported. Note that after the user has authenticated, it is still necessary to determine if the user has permission to access patient information. The code snippets below show use of the AccessController and the more powerful AccessManager (which allows custom actions such as "order" and "reviewResults.")

```
String gbac_class = new String("Demographics");
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 “Demographics” information using a JAAS AccessController

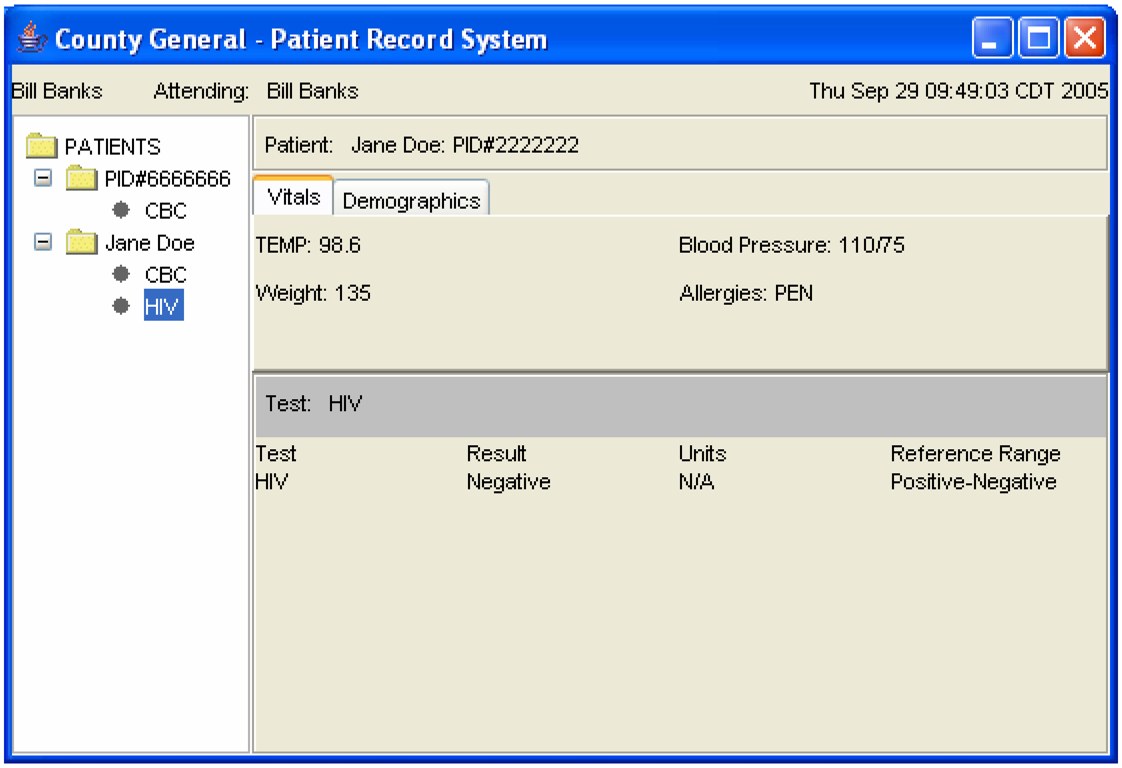
```
String gbac_class = new String("HIV");

 JaasResource jr = new JaasResource (gbac_class);
 if (am.accessAllowed(jr, "reviewResults", lc.getSubject())) {
                    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 “HIV” lab results using a jLock AccessManager with custom operations

A Patient Record Demonstration

Next, we want to show how you might build an application that integrates patient demographic and clinical data while only displaying the information that a user is authorized to view. We call this demonstration program the “County General - Patient Record System.” The source code for the demonstration program below is freely available.

When Bill Banks, a Physician, logs into the system, a custom view of patient information is generated. Note that he is only allowed to view information that he is authorized to see. That is, there may be information in the database, based on the GBAC rules, that he is not authorized to view. 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.


County General - Patient Record System
Bill Banks
Attending: Bill Banks
Thu Sep 29 09:49:03 CDT 2005
PATIENTS
PID#6666666
CBC
Jane Doe
CBC
HIV
Patient: Jane Doe: PID#2222222
Vitals
Demographics

TEMP: 98.6 Blood Pressure: 110/75

Weight: 135 Allergies: PEN

Test: HIV

Test	Result	Units	Reference Range
HIV	Negative	N/A	Positive-Negative

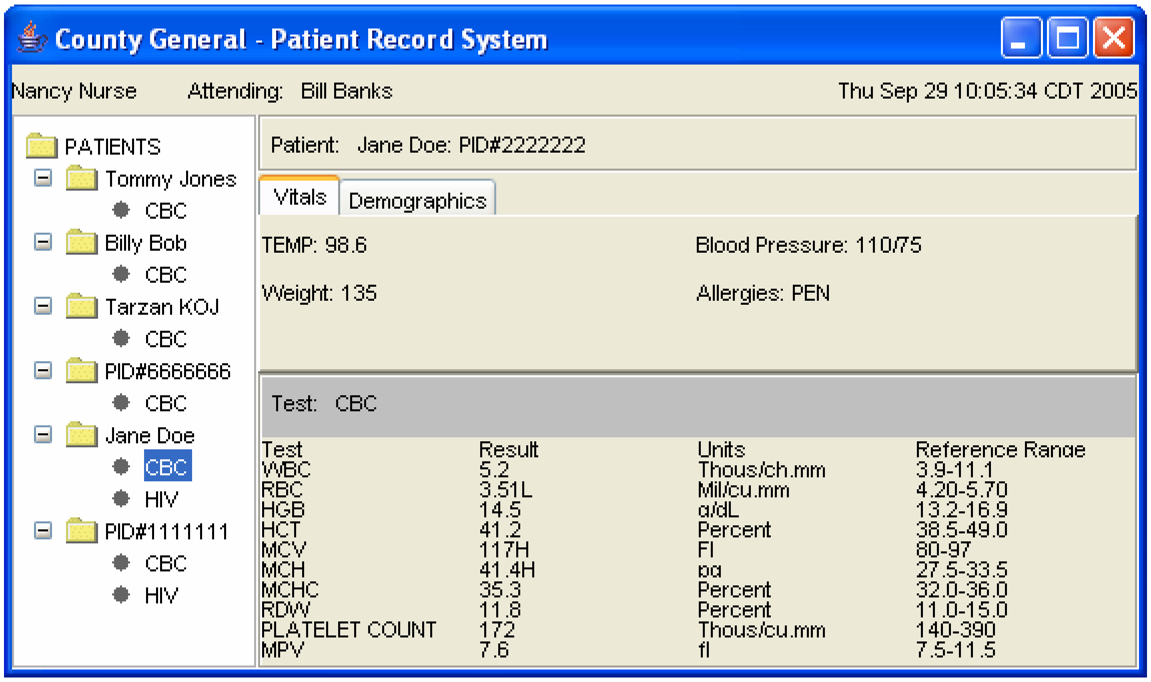
Information Authorized for Bill Banks, a Physician

Here is the line of code (Guard) that is inserted into the application to determine whether or not to display a patient in the tree. A similar access control check is made on each document type to determine the information to display under the patient. 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(patient.toResource(),operation,lm.getSecurityAttributes());
```

Code in prototype to determine whether or not a patient can be viewed by the user

Below we see the same portal when Nancy Nurse, a nurse, is logged in. Notice that the nurse has access to more patients in her view as she may be assisting multiple physicians. If you select test results for a patient, however, access to those results may be restricted. For example, the nurse can see that an HIV test has been ordered, but is not allowed access to the results.


County General - Patient Record System

Nancy Nurse Attending: Bill Banks Thu Sep 29 10:05:34 CDT 2005

- PATIENTS
  - Tommy Jones
    - CBC
  - Billy Bob
    - CBC
  - Tarzan KOJ
    - CBC
  - PID#6666666
    - CBC
  - Jane Doe
    - CBC
    - HIV
  - PID#1111111
    - CBC
    - HIV

Patient: Jane Doe: PID#2222222

Vitals | Demographics

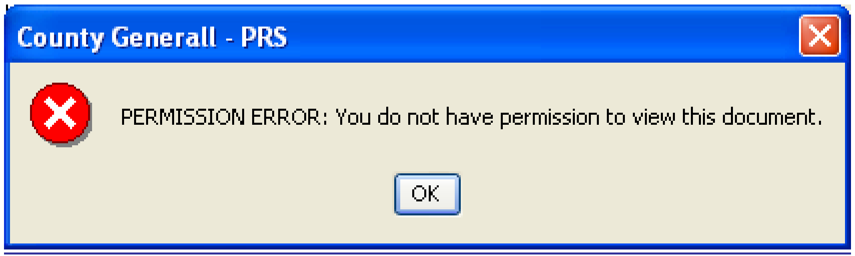
TEMP: 98.6 Blood Pressure: 110/75

Weight: 135 Allergies: PEN

Test: CBC

Test	Result	Units	Reference Range
WBC	5.2	Thous/ch.mm	3.9-11.1
RBC	3.51L	Mil/cu.mm	4.20-5.70
HGB	14.5	g/dL	13.2-16.9
HCT	41.2	Percent	38.5-49.0
MCV	117H	Fl	80-97
MCH	41.4H	pg	27.5-33.5
MCHC	35.3	Percent	32.0-36.0
RDW	11.8	Percent	11.0-15.0
PLATELET COUNT	172	Thous/cu.mm	140-390
MPV	7.6	fl	7.5-11.5

Patient View based on authorization for Nancy Nurse, a Nurse


County Generall - PRS

PERMISSION ERROR: You do not have permission to view this document.

Message received when Nancy Nurse tries to view HIV results

You may have also noticed that some of the patient names are not shown (they are identified only by a PID#). The AccessManager is also consulted regarding whether or not to display patient names on the graphical user interface.

```
if (patient.isVIP()) {
  vip =  am.accessAllowed(vipResource,operation,lm.getSecurityAttributes())
  .........  set up name or id in display based on access decision ...........
}
```

Code in prototype to decide whether or not user is allowed to view a patient name

The CPR application uses the access manager as a software guard to provide access decisions before displaying any information related to patients. The policy is easily administered using tools that are simple to use and easy for business people to understand.

Summary

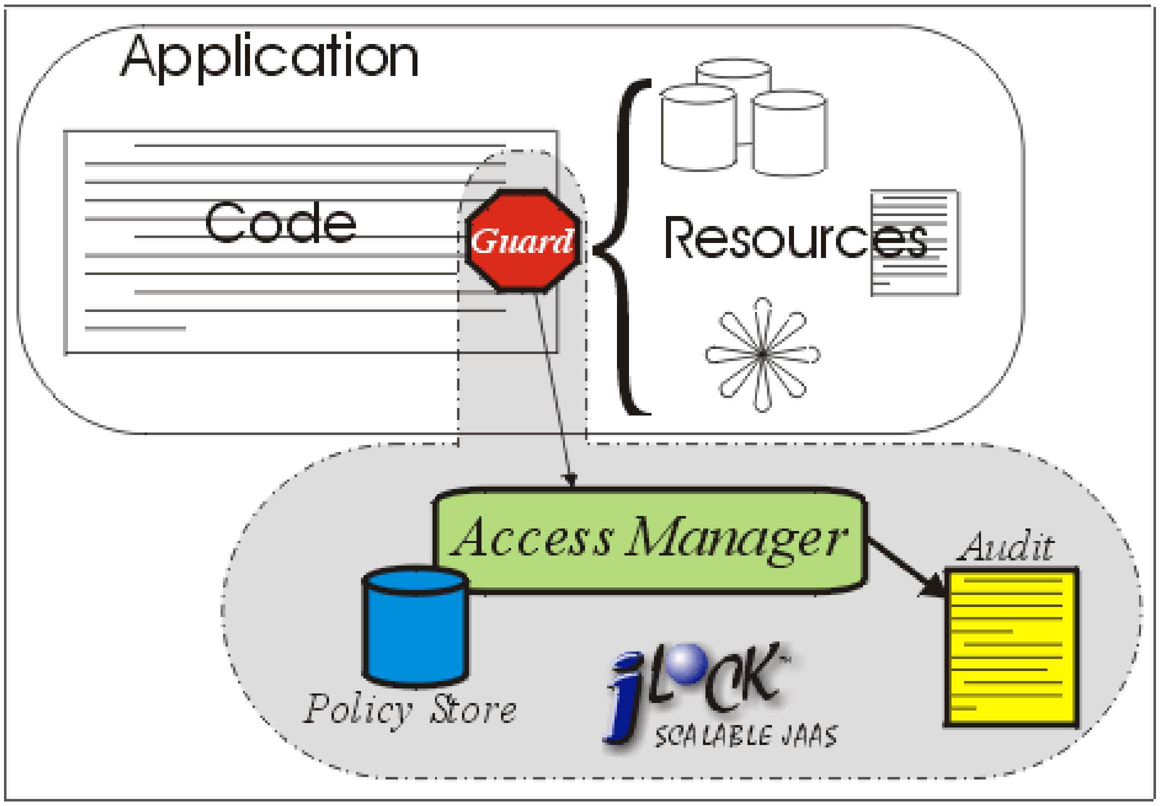
In this paper we have outlined how the jLock JAAS service can be used to implement a Service Oriented Architecture for application-level security that meets the unique requirements of Governance-Based Access Control in Healthcare.

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 Governance-based access control 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