To appear in the 14th Annual Computer Security Applications Conference, December 1998

Role Based Access Control Framework for Network Enterprises

Dan Thomsen, Dick O'Brien, and Jessica Bogle
Secure Computing Corporation
2675 Long Lake Road
Roseville, MN 55113
thomsen@securecomputing.com

Abstract

A business's success depends on its ability to protect valuable business assets in an increasingly hostile environment. Protecting information requires a cost, not only in purchasing security components, but also in ensuring that those security components are properly managed. Role Based Access Control (RBAC) shows promise for making security administration easier, thus reducing the cost of managing security components.

RBAC provides a convenient layer of abstraction by describing access control patterns. This paper presents an RBAC framework comprised of seven abstract layers. Multiple layers allow users to work with a layer they understand. Thus a balance can be struck between fine grained access control and ease of management. The goal is to provide easy security management for a wide variety of network applications. The NAPOLEON tool which implements parts of the framework is also described.[1]

1. Introduction

Administrating security systems is a complex task. In order to enforce a tight security policy many security constraints must be expressed. Security constraints can be classified into two broad categories, those required by the application, and those required by the local security policy. The RBAC framework presented here is broken down into seven abstract layers. The first four layers deal with the application specific constraints. The remaining layers focus on the site specific constraints.

This approach applies the divide and conquer principle to the tough problem of security management. Rather than place all the burden of security management on a system administrator in the field, the application developers share the burden by creating basic security building blocks. These building blocks capture complex application specific security constraints, freeing the local system administrators from configuring the many detailed constraints. The application developers are the people who best understand the application and can best describe the application security constraints.

Only the local administrators know the local security policies, thus they are the only ones who can describe their security constraints. The application designers cannot create security policies that apply to all sites. Thus the local system administrators must have the capability to create their own building blocks, should those prepared by the application developer be insufficient.

The goal of the RBAC framework is to centrally control access to a wide variety of network resources. This means incorporating diverse applications on a variety of hosts, legacy applications, and applications with unsophisticated security mechanisms. Incorporating these diverse resources into a consistent framework for security specification and enforcement creates many challenges that are discussed throughout the paper.

The remainder of this section discusses how network resources and applications are modeled in the framework. Section 2 discusses the layers in the RBAC framework, Section 3 describes the NAPOLEON policy tool based on the framework, and Section 4 summarizes the paper.

1. This work was supported by DARPA under contract F30602-97-C-0245.

1.1. Modeling the application

Before access to network resources can be granted those resources must be understood. This means that the network applications must be incorporated into the model. Applications are written in different languages and run on a variety of hosts with different security mechanisms. A universal description of applications is needed that is independent of their implementations.

Currently there are two widely accepted frameworks for developing distributed network applications: CORBA and Microsoft's COM/DCOM [1], [2]. Both frameworks use an interface definition language (IDL) to describe how an application can be accessed. The IDL definition expresses the application in an object oriented framework by listing each object's publicly available methods. Thus an object oriented approach provides a good foundation for the RBAC framework. Access is either granted to an object method or it is denied.

Creating this object oriented model of the application is simple in the CORBA and COM/DCOM environments. The IDL file that describes an object's publicly available methods can be parsed and automatically incorporated into a security management tool.

To incorporate a legacy application into the framework, an IDL file must be created. This involves defining the legacy application in terms of objects and object methods. This is similar to how applications can be wrapped with a CORBA interface for the CORBA environment. However, the interface does not have to connect to the legacy application. The concern for the RBAC framework is, IF a method can be accessed not HOW. As you will see in the next section a component can be created to translate between the RBAC framework and the legacy applications enforcement mechanism.

1.2. Enforcing the RBAC framework

This section provides an overview of how the RBAC framework is enforced in a heterogeneous network environment.

Making access control decisions centrally for the entire enterprise would likely create a performance bottleneck. Centralized decision making also leads to a single point of failure that could shutdown the entire network. While these performance problems could be mitigated by duplicate security servers, performance would still lag local enforcement. So we prefer centralized management of the policy, with the security decisions being enforced as close as possible to the application.

The centralized management tool grants users access to objects. Once a change is made the tool translates the security policy from the RBAC framework to the target's native security mechanism, which is then transported to the target.

For example, if a user was given access to the Internet via the security management tool, the tool translates that request into a number of modifications to a firewall Access Control List (ACL). These modifications are then communicated to the firewall, which implements the changes.

The location of each application is known. The tool must push the security information out to the application making the access control decisions. Since we are assuming a heterogeneous network the central security policy must be translated into security mechanisms for each host making up the enterprise. If the target host already has an understanding of roles, or has a unified access control mechanism like CORBA, the translation process is easy. If the host does not understand roles the translation of the policy becomes more difficult.

For legacy applications the translation from the RBAC framework to the legacy application's security mechanisms is harder still. For example, protecting an FTP server on a Unix host first requires describing the FTP server in object oriented terms. For enforcement the policy must be translated to the equivalent user accounts and file permissions. While this translation is difficult, the important point is that the legacy applications can be included into the centralized policy management, albeit at a higher cost.

1.3. Divide and conquer

There are two primary users of the RBAC framework, application developers and local system administrators. The hard problem of creating a security policy must be divided between these two groups. The approach is for the application developers, using their in-depth knowledge of the application, to create generic security components. These security components serve to hide the application specific details. The local system administrators use these security components as the security building blocks to customize the security policy for their organization.

Just as it is important to document software design to facilitate application maintenance, it is important to document the security components of the application. When the application developers create the security building blocks not only are they creating tools for the local system administrators, but they are also documenting security design and usage. Permanent documentation is critical for the long term maintenance of an application, since application developers may leave or not remember details.

In the RBAC framework described in the next section there are two role hierarchies, one for the application developers, and one for the local system administrators. This allows each type of user to express the constraints they understand best.

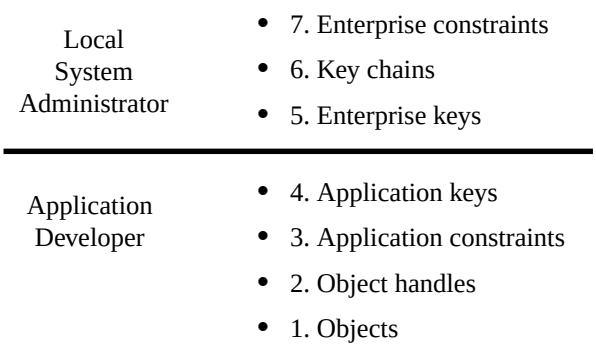

Figure 1. Abstract layers in RBAC framework

2. RBAC framework layers

One difficulty with any security enforcement mechanism is to ensure that the policy is accurately expressed, in an easily understood manner. Even with tools, describing a consistent security policy is a huge task. Each system administrator has a different understanding of the security mechanisms available to them. The RBAC framework provides seven abstract layers of security policy specification. This allows an administrator to quickly implement a course grained security policy, or precisely specify a fine grained policy. As their understanding of the enterprise grows, they can further customize the policy using the more detailed layers of abstraction.

At the highest layer users perform simple operations that are easy to understand. At this layer administrators hand out sets of predefined access rights. If this layer does not provide them with enough detailed security control, they can proceed to the next layer down where they have finer control of the access rights.

Figure 1 shows the abstract layers of the model. Each layer is described in detail below, starting with the finest security control to the broadest security control. The first four layers are the responsibility of the application designer. The final three layers are the responsibility of the local system administrator.

It is important to note that the first four layers are designed to be used by the application developers to create generic solutions for most sites. This means that at each layer there is documentation describing the rational for the security constraint and its intended use. This provides crucial guidance for local system administrators and application owners should they need to customize the security policy.

2.1. Layer 1 - Objects

The basic building block of this RBAC framework is an object. An object is an abstract description of some kind of data in the system. An example of an object might be a patient record or an ftp proxy.

Each object has a name and a set of public methods that can be used to access the object. The object can only be accessed through the public methods. Object attributes and private methods are hidden and cannot be accessed.

Since the methods are the only way to interact with the application, controlling access to the methods is sufficient to control access to the application [4]. The RBAC framework supports a mechanism for grouping methods into sets that can be assigned to roles. Thus the security policy is built up out of layered sets of methods. Read or write permissions are not associated with objects. A role can either access a method or it cannot.

Method parameters do not need to be included in the access control decision. It is the responsibility of the method creator to ensure that the parameters are valid. For example suppose a parameter value of “DANGER” must only be entered by a supervisor. To capture this in the framework an additional method is created. The new method is identical to the first except that it checks the parameter to ensure it is not “DANGER”. The original method can then be assigned to supervisors, while only the new method can be assigned to others.

Each CORBA object is described in an Interface Description Language (IDL) file. The IDL file describes the name of the object and its publicly accessible methods. The Network Application POLicy EnvirONment (NAPOLEON[1]) tool was created to read in IDL files to specify application specific security policy. The NAPOLEON tool is described further in Section 3. DCOM has an object description language similar to IDL.

Organizing objects. The goal of the RBAC framework is to provide enterprise wide security. For an entire enterprise there can be thousands of objects. Thus the objects must be presented to the application developers via some structure.

Clearly the first step is to organize the objects into the applications to which they belong. If an object belongs to more than one application, the object can be treated as a separate application, or service.

Once grouped by application, the objects can be organized by the software modules that contain them. However, to system administrators in the field, organizing objects by

1. NAPOLEON is named after the layered dessert rather than the historical figure.

module does not make much sense, since that is a software engineering construct rather than an administration construct. Thus the objects need to be organized differently based on who is using the framework.

Cascading objects. There is an issue of how the user's identity is used to make access control decisions on secondary objects. Consider a distributed application where one object (*A*) calls another object (*B*) to calculate an answer. *B* may have different access constraints. The RBAC framework requires that object *A*, be aware of any access constraints of any object it uses. *A* is responsible for making sure that object *B* has all the information needed to make its access control decisions. Object *A* must also handle the case where *B* denies access. This approach is in line with good software engineering practices.

2.2. Layer 2 - Object handles

Usually an object has several methods that are often accessed in conjunction. Rather than having to assign each method to a role individually, the methods can be grouped into a set. The set provides a convenient handle for granting access to an object. The object handle has a text description indicating the kind of access represented by this set of methods and how it should be used.

For example, suppose a medical health application has a patient record object that contains the following methods

- getPrimaryPhyscian()
- getDiagnosis()
- setDiagnosis()
- getBloodPressure()
- setBloodPressure()

There may be many roles on the system that need to access all the "get" methods listed above. Thus the methods can be grouped into an object handle called *Read-Only*, with the following description, "Give the user read access to the patient record".

The object handle captures the way the object is used. It is important to make a distinction between an object handle and the methods needed by a particular role. Here is an example of an incorrect use of an object handle. Suppose a handle called *Doctor* was created for a patient record. If the hospital Administrator role needed the same access as the doctor role it would be very confusing to give the Administrator role the *Doctor* handle. A better object handle name would be *Maintainer* or, if the handle contained all the methods, *All* would be a good descriptive name.

2.3. Layer 3 - Application constraints

Roles accurately describe which types of people need access to certain types of objects. However, a complete policy needs to also express which individuals have access to specific object instances. For example, just because the doctor role has a set of accesses to a patient record does not mean the doctor role should access all patient records. A doctor can only access the patient records for those people currently assigned to the doctor.

There have been several approaches for creating an instance level policy for roles [5], [6], [7]. To encompass these approaches we propose that a set of conditions can be specified when an object handle is assigned to a role. These conditions must be satisfied before access is granted to the methods in the handle set.

The scope of the conditions is broad. The conditions can be based on any combination of the following

- attributes of the user accessing the object
- attributes of the object itself [5]
- other records, such as the current emergency room team in a hospital [6]
- time based constraints
- user's history of accesses

Together these constraints can express most security policy conditions.

Complex constraints. Many of the complex constraints discussed in other RBAC models are described in the application constraints layer. For example, consider a dynamic separation of duties policy where the clerk issuing a check cannot be the same clerk who made the check request. An application constraint is placed on the *issueCheck* method to ensure that the user executing the method is not the person listed as the requestor. Other complex constraints, such as time based, or history based constraints can be captured as well.

Most complex constraints are application specific and can be expressed by the application designer. However, the application designer cannot levy any constraints on specific users, or constraints that span several applications. Those constraints are addressed in layer 6 - enterprise constraints.

There are several an open issues regarding the complexity of the constraints. Should the complexity be limited so that the constraint is easy to understand? Should the application developer be allowed to put in arbitrary code? Is there a limit on the complexity of policy that can be enforced? More research is required in this area.

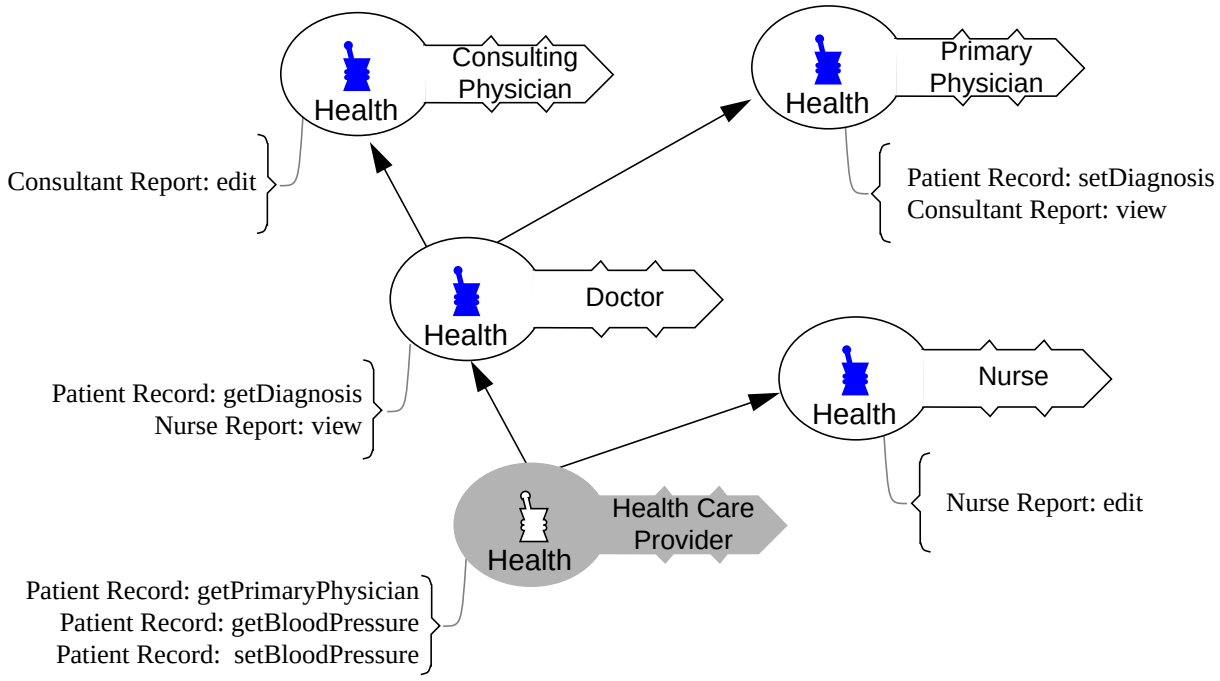

Figure 2. A sample Application key hierarchy showing how methods are inherited, Patient Record, Nurse Report and Consultant Report are objects

2.4. Layer 4 - Application keys

The security information encoded by the application developer needs to be put in an easy to use form for the local system administrator. This collection of security information is an application key. An application key is much like a door key. When a key is given to a user they can access the resources represented by the key. In the RBAC framework application keys are a set of methods and application constraints on those methods. An application key can be considered an application specific role.

A common theme in many RBAC approaches is to have a role hierarchy that is used to capture relationships between roles [3]. The role hierarchy also describes constraints. As mentioned earlier there are two role hierarchies. The first hierarchy is used by the application developer to show the relationship between application keys, and is called the application key hierarchy.

For example in a medical application you might have role relationships similar to Figure 2. The figure indicates that the doctor role has all the privileges of the health care provider, plus privileges unique to the doctor role. Both the consulting and primary physicians have more, but different permission than the doctor.

The application developer must include each method of the application in at least one key, otherwise users would not be able to access that method

Abstract roles. The role hierarchy is similar to an object oriented class hierarchy and has all the benefits and drawbacks of that approach. One example is the concept of abstract roles. Abstract roles are the equivalent of abstract classes in object oriented languages. It is intended that no users be assigned to an abstract role. The abstract role serves as a description of some permissions that other roles can inherit.

For example, a doctor may not inherit all the permissions of a nurse. A nurse may prepare reports that doctors can read but not write. If the doctor role inherited from the nurse role, the doctor role would have the extra permission. Instead an abstract role called health care provider is created that has all the inheritable permissions. No users can be assigned to the health care provider role. Abstract roles provide a way of indicating information about the structure of the role hierarchy.

Multiple inheritance. Another major problem that a object oriented hierarchy faces is multiple inheritance. For the application key hierarchy multiple inherence does not cause a problem.

First consider an application that does not have any application constraints. Each application key consists of a set of methods. If a key inherits from two different keys the two methods sets are combined. This is because if a method is inherited multiple times, there is no ambiguity,

since the user can either access the method or not. It does not matter which key the method was inherited from.

If a key inherits from two keys with application constraints there are two possibilities. The first is that the methods do not overlap. In this case the methods and conditions can be combined with no ambiguity. If however, the two keys contain the same method with different constraints, the constraints are logically ORed together. Thus a user may satisfy either constraint to gain access to the method.

2.5. Layer 5 - Enterprise keys

When an application is installed into a network, a description of all its objects and an initial set of application keys are installed in the management tool. This is the first layer in the framework in which users are incorporated. There is a one to one mapping between application keys and enterprise keys, except for abstract keys which do not appear as enterprise keys. Enterprise keys can be assigned to users.

Local system administrators grant access by assigning the enterprise keys to users, see Figure 3. Once assigned the user has access to the object methods listed in the key as long as any application constraints associated with the key are satisfied. Note that keys are not capabilities, and cannot be passed between users.

2.6. Layer 6- Key chains

As the application developers create a role hierarchy to describe the constraints of the application, the local administrators describe their constraints with their own role hierarchy. In keeping with the key theme local system administrators group keys into sets called key chains.

A key chain is simply a collection of enterprise keys. A key chain may contain other key chains. As a result the local system administrator is able to build up a complex hierarchy that fits the needs of their site. For example, if the basic user at the site has access to three applications, those enterprise keys can be group into a single key chain, see Figure 4.

Key chains express the local security policy constraints. The local system administrator can build a policy as complex or as simple as they like by proper structuring of the key chains.

Multiple inheritance for key chains is resolved the same as multiple inheritance in the application key hierarchy.

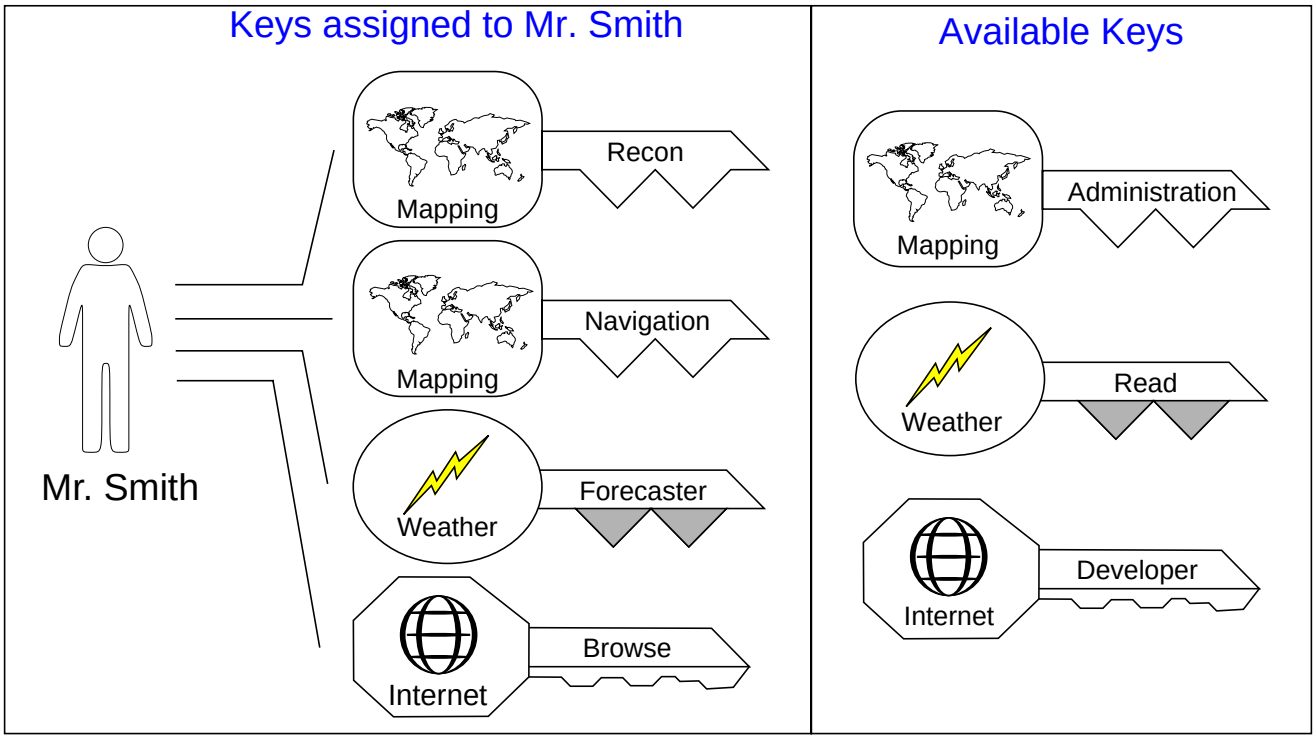

Figure 3. Assigning users access by giving them keys

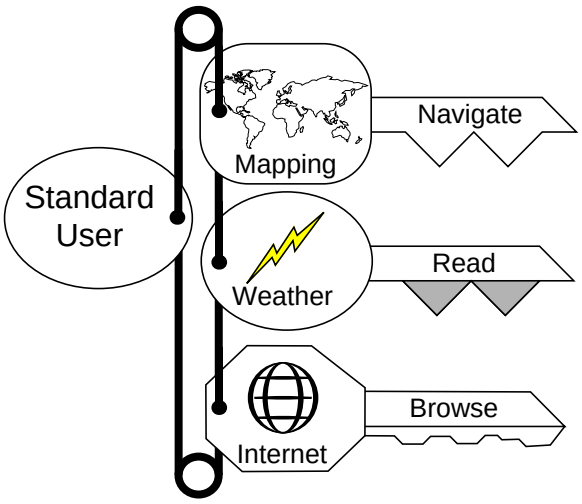

Figure 4. Local system administrators can group keys into key chains

Access to an object is granted as long as one complete set of access constraints is satisfied.

2.7. Layer 7- Enterprise constraints

There are complex enterprise security constraints that only the local system administrator knows. Unlike application constraints that are applied to application keys, enterprise constraints can only be applied to key chains. This is because the enterprise keys are the basic building block, shipped with the application. Allowing constraints to be associated with enterprise keys would confuse later system administrators.

Since enterprise constraints are applied to key chains, the constraints can span several applications. If constructed and managed properly the constraints can span the entire enterprise.

This completes the basic description of the RBAC framework. The next section discuss the NAPOLEON tool that implements portions of the framework.

3. NAPOLEON policy tool

The previous sections discussed a general RBAC framework for providing consistent roles across an enterprise. The NAPOLEON tool implements the portion of the framework used by the application developer.

3.1. NAPOLEON implementation

The current implementation focuses on CORBA applications. NAPOLEON serves as an IDL reader that allows the user to create new application policies, specify and store the policy, and translate the policy into a security enforcement mechanism.

Creating new policies. A new application policy is created by the user selecting an IDL file and assigning it to an application. In CORBA an object is called an interface. Each interface has a description of the publicly accessible

- methods
- read only attributes
- read-write attributes
- inherited interfaces

The RBAC framework only includes methods, so accessing attributes has to be modeled as methods based on the attribute name with read or read-write appended to it. Inherited interfaces basically include methods from other objects with unique names. There is no method overloading in CORBA.

When reading in an IDL file the tool automatically creates an object handle called *ALL* for each object/interface. The *ALL* handle contains all the accessible items in the interface.

Specifying policy. Once the IDL file is loaded the user can start specifying the policy. The user can

- create new object handles (see Figure 5)
- create application keys
- assign object handles to keys

The application keys are the equivalent of application specific roles. However, NAPOLEON does not provide an interface to show the role's hierarchical nature to each other as described in Figure 2. A hierarchy is a convenient design tool, but so far in the examples we have considered the number of application specific roles is small. Thus little benefit is gained by showing the hierarchical relationship between application keys.

Currently the application constraint layer is not implemented. In the future when application constraints are implemented they will be associated with the binding between the object handle and the application key. As the user creates policy components they are stored permanently in a relational database.

Translating the policy. To see the impact of having the application developer design policy pieces, the NAPOLEON prototype was integrated with the Open Group Research Institute's Adage policy tool. Adage provides command line and GUI interfaces for specifying security

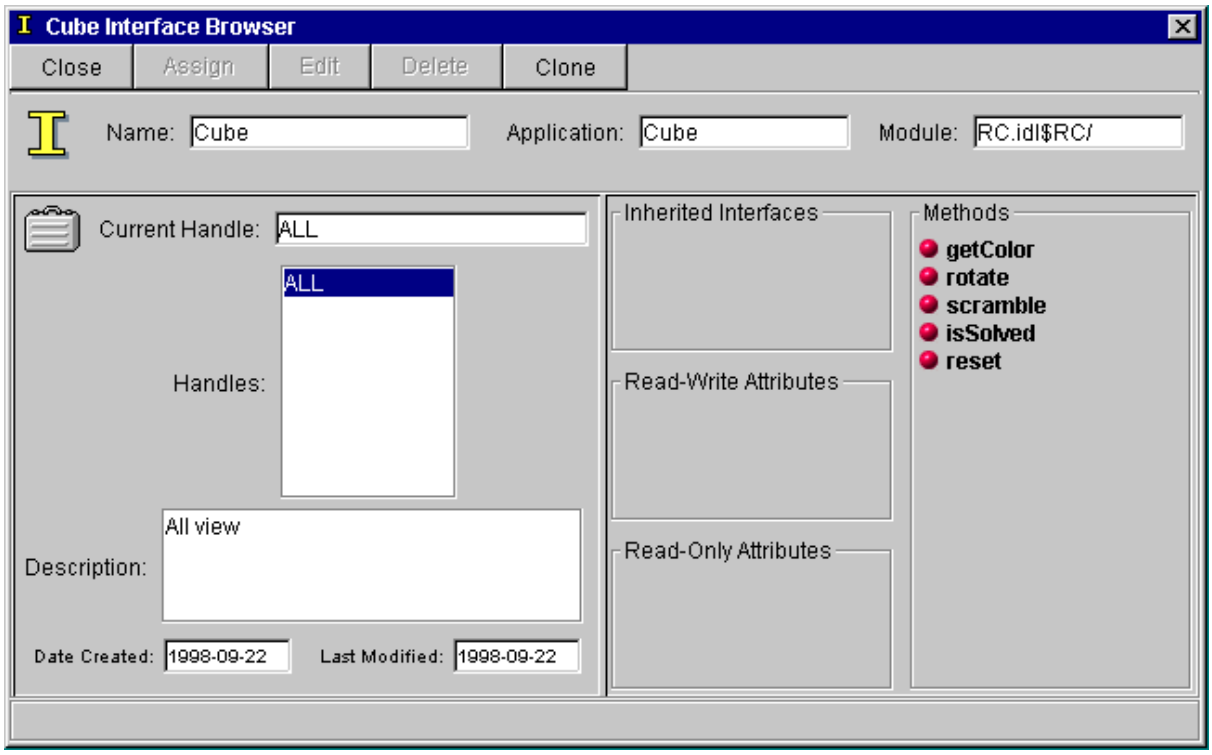

Figure 5. Using the NAPOLEON tool to create object handles.

policies [9]. The application keys created by the developer were translated into Adage rules using the command interface. Then using Adage the local system administrator added users and groups to the NAPOLEON created rules.

Adage also has a decision engine that returns yes/no answers to requests for resources. The decision engine was the target enforcement mechanism. By intercepting CORBA request and rerouting them to the decision engine, we were able to dynamically control access to CORBA object methods.

The NAPOLEON prototype showed that it was extremely useful for portions of the security policy to be designed with the application. The local system administrator was able to implement a fine grained policy in minutes, without having to analyze the application.

3.2. Future prototype work

The most challenging layers of the RBAC framework are the constraint layers. Our next task will be to implement constraints so we can better understand the complexity and challenges placed on the users. We hope to create classes of constraints that make trade-offs between fine grained security and performance. One of the first areas we will look at is having the application designer specify workflow semantics as information flows through the system.

Another area that must be explored is the mapping of security policy to different mechanisms. A complex policy involving history constraints could be expressed in the tool, but could not be enforced by a number of platforms. Handling the loss of fine grained security by translation, must be researched further.

4. Summary

The RBAC framework described in this paper allows for the creation of tight access control policies for the entire enterprise. It allows the incorporation of a wide variety of applications on different hosts with different access control mechanisms.

The approach simplifies the security administration for the enterprise by taking a divide and conquer approach. Security policy managers can focus at the level of detail they are familiar with, modifying details only as necessary. The Napoleon prototype has demonstrated the value of well designed, layered, security building blocks, indicating that the RBAC framework provides a means of balancing the complexity of administrating a large network with the need for controlling security access at a fine level of detail.

References

[1] Orfali R., and Harkey D., “Client/Server Programming with Java and CORBA,” Wiley, New York, 1998.

[2] Sessions R., “Com and Dcom: Microsoft’s Vision for Distributed Objects,” Wiley, New York, December 1997.

[3] Sandhu R.S., Coyne E.J., Feinstein H.L, and Youman C.E. “Role-Based Access Control Models,” *Computer*, 29:38-47, Feb 1996.

[4] Barkley J., “Implementing Role-Based Access Control Using Object Technology”, *Proc. of the First ACM Workshop on Role-Based Access Control*, pp. II93-II98, November 1995.

[5] Thomsen D. J., “Role Based Application Design and Enforcement,” *Database Security, IV Status and Prospects*, edited by S. Jajodia and C.E. Landwehr, pp. 151-168, North Holland, New York 1991.

[6] Thomas R., “Team-Based Access Control (TMAC),” *Proc. Second ACM Workshop on Role-Based Access Control*, pp. 13-22, November, 1997.

[7] Thomas R. K. and Sandhu R. S., “Task-Based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-oriented Authorization Management,” Proc. Eleventh Annual IFIP WG 11.3 Working Conference on Database Security, Fall 1997.

[8] Thomsen D. J., “Integrity Issues in Secure Systems,” May 1991, Master Thesis, University of Minnesota.

[9] Zurko M. E. and Simon R. T., “User-Centered Security,” *New Security Paradigms Workshop*, September 1996.