CHAPTER 23

USER ROLE-BASED SECURITY MODEL FOR A DISTRIBUTED ENVIRONMENT *

S. Demurjian, T.C. Ting, J. Balthazar, H. Ren, and C. Phillips
Computer Science & Engineering Department, The University of Connecticut
steve@engr.uconn.edu, ting@engr.uconn.edu

P. Barr
The Mitre Corporation, Eatontown, New Jersey
poobarr@mitre.org

Abstract A distributed resource environment (DRE) allows distributed components (i.e., servers, legacy systems, databases, COTs, printers, scanners, etc.) to be treated akin to OS resources, where each component (resource) can publish services (an API), that are then available for use by clients and resources alike. DREs have lagged in support of security. To address this deficiency, this paper concentrates on proposing a technique for seamlessly integrating a role-based security model, authorization, authentication, and enforcement into a DRE, including our prototyping with the JINI DRE.

Keywords: Security, roles, distributed computing, authorization, authentication.

1. INTRODUCTION AND MOTIVATION

The emergence of distributed computing technology such as DCE [10], CORBA [7], and DCOM [5], has enabled the parallel and distributed processing of large, computation-intensive applications. The incorporation of security has often been dependent on programmatic effort. For example, while CORBA has confidentiality, integrity, accountability, and availability services, there is no cohesive CORBA service that ties to-

*This work partially supported by the Mitre Corporation and a AFOSR grant.

gether them with authorization and authentication. However, there has been significant progress in distributed authentication in Kerberos [6] and Cheron [3], security metric analysis and design [9], Internet security via firewalls [8], role-based access control on Web-based intranets [12], and security for mobile agents [14, 15].

Our specific interest is in distributed applications that plug-and-play, allowing us to plug in (and subtract) new "components" or *resources* where all of the resources (e.g., legacy, COTS, databases, servers, etc.) have services that are published (via APIs) for use by distributed application components. The resources, their services, and the clients, interacting across the network, comprise a *distributed resource environment (DRE).* Our goal in this paper is to leverage the infrastructure of a DRE to support and realize role-based security. In such a setting, we propose specialized security resources that interact with non-security resources and clients, to authorize, authenticate, and enforce security for a distributed application in a dynamic fashion. To demonstrate the feasibility of our approach, we exploit Sun's DRE JINI [1]. JINI promotes the construction and deployment of robust and scalable distributed applications. In JINI, a distributed application is conceptualized as a set of services (of all resources) being made available for discovery and use by clients. Resources in JINI discover and then join the Lookup Service, registering their services for network availability. However, JINI lacks the ability to restrict what a client can and cannot do, i.e., the services of a resource are available to any and all clients without restriction.

Our main purpose of this paper is to examine the incorporation of a role-based approach to security within a DRE, in general, and JINI, in particular, which supports the selective access of clients to resources. We propose security specific resources for authorization of clients based on role, authentication of clients, and enforcement to insure that a client only uses authorized services. We provide role-based access to services based on our previous object-oriented efforts [2], without programmatic changes to a resource, allowing the resource to dynamically discover security privileges from security resources. In the remainder of this paper, Section 2 provides brief background on JINI, Section 3 proposes a role-based security model for a DRE, Section 4 synopsizes our prototyping with JINI, and Section 5 contains our conclusions.

2. JINI

JINI allows stakeholders to construct a distributed application by federating groups of users (clients) and the resources that they require [1]. In JINI, the resources register services which represent the methods (sim-

ilar to an API) that are provided for use by clients (and other resources). A Lookup Service is provided, and operates as a clearinghouse for resources to register services and clients to find services. The Lookup Service arbitrates all interactions by resources (e.g., discovering Lookup Services, registering services, renewing leases, etc.) and by clients (e.g., discovering Lookup Services, searching for services, service invocation, etc.). After discovery has occurred, the resources register their services on a class-by-class basis with the Lookup Service. The class is registered as a service object which contains the public methods available to clients coupled with a set of optional descriptive service attributes. The service object is registered as a proxy, which contains all of the information that is needed to invoke the service. One limitation of this process is that once registered, a resource's services are available to all clients. The registration of services occurs via a leasing mechanism. With leasing, the services of a resource can be registered with the Lookup Service for a fixed time period or forever (no expiration). The lease must be renewed by the resource prior to its expiration, or the service will become unavailable. From a security perspective, the lease that is given by a resource to its services is not client specific. Once leased, a service is available to all, even if the service was intended for a targeted client or group of clients. Our work seeks to overcome this limitation.

3. A DRE ROLE-BASED SECURITY MODEL

In a DRE, all of the different resources are treated in a consistent fashion, allowing all of the clients and resources to be seamlessly integrated. Clients consult the Lookup Service to locate and subsequently execute the services of the found resource that are necessary to carry out their respective tasks. However, DREs are lacking in their support of security. When a resource registers its services with the Lookup Service, there is no way for the resource to dictate which service can be utilized by which client. If the resource wants to control access to its services, it must do so programmatically, putting in client-specific code within the implementation of the service. We are extending the security capabilities of a DRE to allow resources to selectively and dynamically control who can access its services (and invoke their methods), based on the role of the client. Our solution exploits the DRE, by defining dedicated resources to authorize, authenticate, and enforce role-based security for the distributed application. The remainder of this section is organized to propose and discuss: a software architecture for role-based security in a DRE (Section 3.1), the security resources and services for such an

architecture (Section 3.2), and the usage of the solution by clients and resources (Sections 3.3 and 3.4).

3.1. A SOFTWARE ARCHITECTURE

A software architecture for supporting role-based security in a DRE is presented in Figure 1.1, and contains: a set of clients that seek to utilize a set of resources, one or more Lookup Services that allow clients to find resources (and their services), and three security-specific resources.

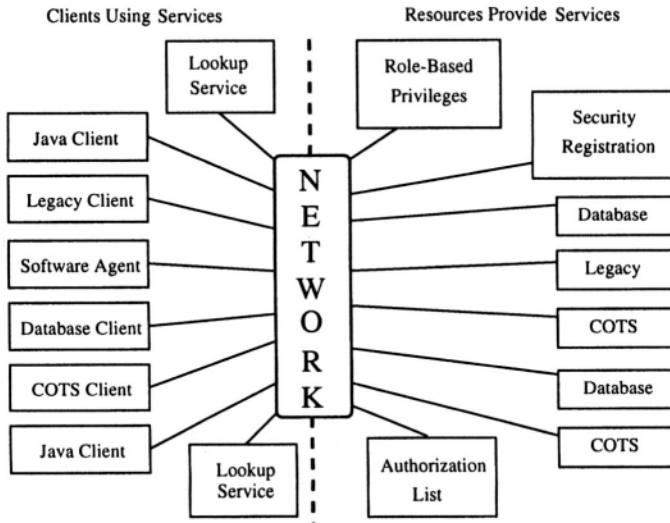

Figure 1.1. General Architecture of Clients and Resources.

The *Role-Based Privileges* resource tracks the services of each resource, and for every service, the methods that are defined. Each user role may be granted access at varying levels of granularity, allowing a role to be assigned to a resource (able to use all services and their methods), to a service (able to use all of its methods), or to individual methods (restricted to specific methods of a service). The *Authorization-List* resource maintains a list of all users, and for each user, tracks the roles that they have been authorized to play. Finally, the *Security Registration* resource tracks the current active users uniquely identified by a triad of <name, IP address, user role>. The architecture supports:

1 Role-based authorization to grant and revoke resources, their services, and their methods for use by clients. This is accomplished using the Role-Based Privileges and Authorization-List resources.

A special security client (see Section 3.3) can be utilized by the security officer to manage this security data.

2 Client authentication for the verification of the identity and role of client. This is supported by the Security Registration resource, which tracks all active clients (name, IP address, role), and is used by resources whenever a client attempts to access a service.

3 Customized resource behavior so that the client and its role dynamically determines if a particular service of a resource can be used. A resource utilizes all three security specific resources to control access to its services by clients.

The term client is used in a general sense; resources can function as clients to access other resources as needed to carry out their functions.

3.2. SECURITY RESOURCES/SERVICES

This section examines the Role-Based Privileges, Authorization List, and Security Registration resources (see Figure 1.1) via a role-based approach to discretionary access control [2, 4, 11, 13]. In a DRE, the computational and abstraction model is to define, for each resource, a set of one or more services, and for each of the services, to define a set of one or more methods. However, there is no a priori way to selectively control which client can utilize which resources (and its services and their methods). We leverage our past work [2] on selectively allowing the methods defined on object-oriented classes to be assigned on a role-by-role basis as a basis to selectively control which clients can access which services and methods of which resources. The role-based security model presented herein will focus on the ability to grant and revoke privileges on resources, services, and/or methods to clients playing roles.

3.2.1 Role-Based Privileges Resource. This resource is utilized by the security officer to realize the defined security policy for a distributed application to: define user roles; grant access of user roles to resources, services, and/or methods; and, when appropriate, revoke access. The Role-Based Privileges resource is utilized by the resources (e.g., legacy, COTS, database, Java server, etc.) that comprise the distributed application to dynamically determine if a client has the required permission to execute a particular service. To facilitate the discussion, consider the definitions:

Definition 1: A *Resource* is a system (e.g., a legacy, COTS, database, Web server, etc.) that provides functions for the distributed application via a collection of n services, $S_1, S_2, ..., S_n$.

Definition 2: A *Service,* $S_i, i = 1..n$, is composed of p_i methods M_{i1}, M_{i2}, ..., M_{ip_i} where each method M_{ij}, $j = 1..p_i$ is similar to an object-oriented method, and each method represents a subset of the functionality provided by the service.

Definition 3: A *Method* M_{ij}, $j = 1..p_i$ of a service S_i is defined by a signature (method name, parameter names/type, and return type).

Definitions 4, 5, and 6: Each resource has a unique *resource identifier* that allows the DRE to differentiate between replicated resources. Each service has a unique *service identifier* to distinguish the services within a particular resource. Each method has a unique *method signature* that permits overloading of method names while allowing the methods of the same service to be distinguished.

Each triple of <resource identifier, service identifier, method signature> uniquely identifies the method across the distributed application.

Given these definitions, we can now define the concept of user role. In our past work [2], we proposed a user-role definition hierarchy to characterize the different kinds of individuals (and groups) who ail require different levels of access to an application. For the purposes of this discussion, we focus on the leaf nodes of the hierarchy.

Definition 7: A *user role, UR*, is a uniquely named entity that represents a specific set of responsibilities against an application. Privileges are granted and revoked as follows:

- UR can be granted access to resource R, denoting that UR can utilize all of R's services, $S_1, S_2, ..., S_n$, and, for all $S_i, i = 1..n$, all of the p_i methods $M_{i1}, M_{i2}, ..., M_{ip_i}$.
- UR can be granted access to a subset of the services of resource R, denoting that UR can utilize all of the methods defined by that subset.
- UR can be granted specific access to a method via the triple of <resource identifier, service identifier, method signature>.

Once granted, access to resources, services, and/or methods can be selectively or entirely revoked by a security officer. The granularity of user roles may be fine or coarse at the discretion of a security officer. Given these definitions, the Role-Based Privileges resource maintains: a *resource list*, indexed by <resource identifier> and for each resource, a list of all user roles granted access; a *service list*, indexed by <resource identifier, service identifier>, and for each service, a list of all user roles granted access; a *method list*, indexed by <resource identifier, service

identifier, method signature>, and for each method, a list of all user roles granted access; and a *user-role list*, indexed by <role name, role identifier>, and for each user role, a list of all resources, services, and/or methods, to which that user roles has been granted access. The information of the Role-Based Privileges resource can be manipulated by the different clients that are part of the distributed application:

- Each resource must register with the Role-Based Privileges resource, so that the master resource list, service list, and method list, can be dynamically modified. Resources must be allowed to register and un-register services/methods. The Register service in Figure 1.2, supports these actions.
- Each resource, when consulted by a client (e.g., GUI, software agent, another resource, etc.), asks the Security Registration resource if the client has registered, (see Section 3.2.3) and if so, asks the Role-Based Privileges resource if the client has been granted access to a service/method pair based on the role of the client. The Query Privileges service in Figure 1.2 supports these actions.
- There is a Security Client (see Section 3.3), utilized by the security officer to define and remove user roles and to grant and revoke privileges (resources, services, and/or methods). The Grant-Revoke-Find service in Figure 1.2 supports these actions.

To simplify the presentation, we have omitted return types. For example, the majority of the methods will return a success/failure flag, the Check_Privileges will return a yes/no, and the Finds will return result sets. Note that the services in Figure 1.2 represents a general characterization of the services for all three security specific resources.

3.2.2 Authorization-List Resource. This resource maintains profiles on the clients (e.g., users, tools, software agents, etc.) that are actively utilizing services within the distributed application. The identification of users is more problematic in a distributed setting, since a user may not be an actual person, but may be a legacy, COTS, database, agent, etc. This leads to the definition:

Definition 8: A client profile, CP, characterizes all of the pertinent information needed by a resource to dynamically verify whether a client can access the desired triple of <resource identifier, service identifier, method signature>.

The Authorization-List resource maintains the client profiles using two services (see Figure 1.2). The Client Profile service is utilized by the

Figure 1.2. The Services and Methods for Security Resources.

security officer, via a Security Client (see Section 3.3), to create and manage the profiles for clients. The Authorize Role service is also utilized to verify whether a client has registered with a role (see Section 3.4).

3.2.3 Security Registration Resource. This resource is utilized by clients for identity registration (client id, IP address, and user role) and by the Security Client (see Section 3.3). The Register Client service (see Figure 1.2), allows a client to have access to resources and their services. Every non-security resource utilizes the Security Registration resource to dynamically determine if the client trying to invoke the service has registered via the IsClient_Registered method. If the client has not registered, the resource will deny service.

3.3. SECURITY CLIENT PROCESSING

To further explain the security processing in the DRE, Figure 1.3 contains a depiction of a Security Client and a General Resource (e.g., legacy, COTS, database, etc.). For the Security Client, Figure 1.3 contains the services from the three security resources that can be used to establish the security policy by creating/finding clients, authorizing roles to clients, and granting, revoking, and finding the privileges that a role has against a resource, service, and/or method. For the General Resource, there is the requirement to register itself, its services, and their

methods with the Role-Based Privileges resource (see Figure 1.3). Registration allows entries to be created on the resource, service, and method lists that can then be accessed via the Security Client. Note that the Security Client and General Resource must discover the services in Figure 1.3, prior to their invocation, as represented by the dashed arrows.

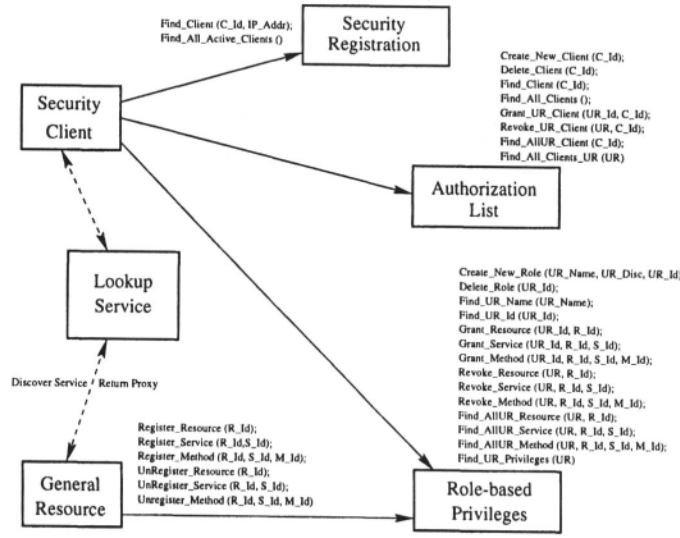

Figure 1.3. Security Client and Database Resource Interactions.

3.4. CLIENT PROCESSING

Finally, to fully illustrate the process, we present an example in Figure 1.4, with flow via the numbered service invocations and returned results. To reduce the confusion in the figure, we have omitted all of the discoveries and proxy returns that would be required for the actions labeled 1, 2, 5, 6, and 8. The actions that occur can be illustrated with the method call represented by the arrow labeled 1. Register_Client. Prior to this method call, the GUI Client would ask the Lookup Service for a resource that provides the Register Client Service (part of the Security Registration resource as shown in Figure 1.2). The Lookup Service would return a proxy to the Register Client Service, and the GUI would use this proxy to execute the Register_Client method.

With the discovery/proxy process described, the example begins by the client making itself known by registering with the Security Registration resource. The arrows labeled 1, 2, 3, and 4 facilitate the registration process, by requiring the client to register itself with the

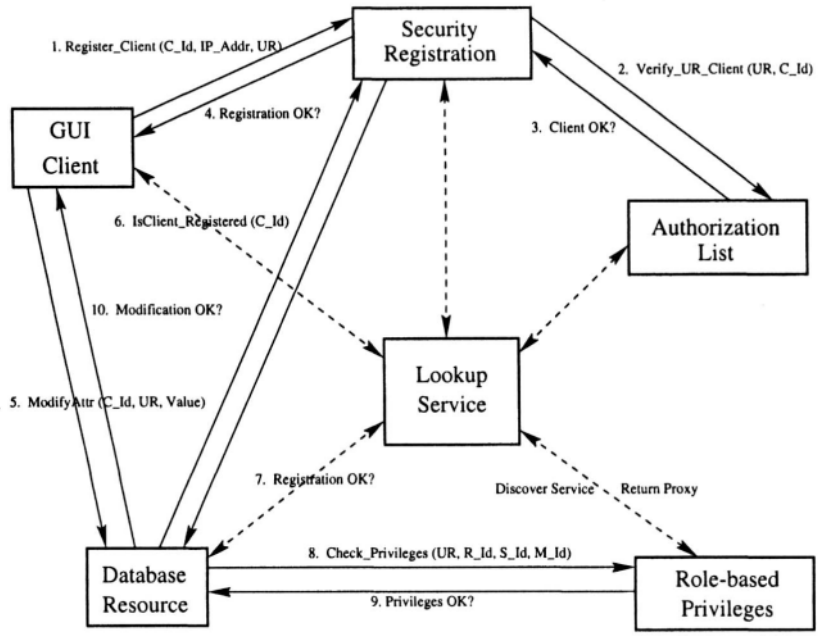

Figure 1.4. Client Interactions and Service Invocations.

Security Registration resource (arrow 1), which in turn interacts with the Authorization-List resource to determine if the client has been authorized to play the desired role (arrow 2). Arrows 3 and 4 complete the process and will return either success or failure. For this discussion, we assume that success is returned. Note that clients that have not registered will still be able to discover resources and services via the Lookup Service. But, they will be prohibited from executing those services if they have not registered. After the Client has successfully registered, it can then discover services via the Lookup Service. Suppose that the Client has discovered the ModifyAttr method that is part of the Update Database Service for the Database Resource (arrow 5 in Figure 1.4). When the Database Resource receives the ModifyAttr invocation request, the first step in its processing is to verify if the client has registered by interacting with the Security Registration resource (arrows 6 and 7). If so, then the Database Resource must then check to see if the client playing the particular role has the required privileges to access the ModifyAttr method, which is accomplished via arrows 8 and 9 by consulting the Role-Based Privileges resource. If the Database Resource receives a response to indicate that the GUI Client has the privileges to access the method, it will then execute the ModifyAttr method and return a status to the Client (arrow 10).

4. PROTOTYPING WITH JINI

This section reviews our prototyping efforts with JINI to support our security model as presented in Section 3. We have implemented the prototype on Windows NT 4.0, LINUX, and UNIX computing platforms, using Java 1.3, MS Access and ORACLE for database management, and JINI 1.3. To support the prototyping effort, we employ a university application where students can query course information and enroll in classes, and faculty can query and modify the class schedule. Our prototype has fully designed and implemented the security resources: Security Registration, Role-Based Privileges, and Authorization-List. These resources, along with two Security Client GUIs, one for policy making and one for policy enforcement (authorizations), make up a reusable Security Client (see Section 3.3) and its security services (Figure 1.2). A security officer can now define, manage, and modify the security privileges dynamically in support of university security policy. Note that additional details on our prototyping can be found at our web site for this project [16].

5. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed and explained an approach that can authorize, authenticate, and enforce a role-based security solution that operates within a DRE. As presented in Section 3, our architecture (see Section 3.1) defined the basis of a role-based security model for a DRE, specified Security Registration, Authorization-List, and Role-Based Privileges resources and their services (see Section 3.2), and detailed the processing of both clients and resources (see Sections 3.3 and 3.4). We prototyped our model from Section 3 using JINI as described in Section 4, and including the clients, resources, security resources, and a Security Client on heterogeneous hardware/OS platforms. The work presented herein represents the first step in an ongoing effort with one doctoral and two masters students. There are a number of issues under investigation: negative privileges to specifically define which resources, services, and/or methods are not available to a client based on role; incorporation of timing and timestamp to allow privileges to expire for a client based on role, which is related to the JINI leasing mechanism; definition and utilization of predicates to allow methods to be invoked by clients only if parameter values are within authorized ranges; and, investigation of the incorporation of our role-based security model and concepts for a DRE into an agent-based environment. Overall, we are concentrating our efforts to define security solutions for distributed applications operating within a DRE.

References

[1] K. Arnold, et al., *The JINI Specification*, Addison-Wesley, 1999.

[2] S. Demurjian and T.C. Ting, "Towards a Definitive Paradigm for Security in Object- Oriented Systems and Applications", *Journal of Computer Security*, Vol. 5, No. 4, 1997.

[3] A. Fox and S. Gribble, "Security on the Move: Indirect Authentication Using Kerberos", *ACM MOBICON 96*, Rye, NY, 1996.

[4] F. H. Lochovsky and C. C. Woo, "Role-Based Security in Data Base Management Systems", in *Database Security: Status and Prospects*, C. Landwehr (ed.), North-Holland, 1988.

[5] Microsoft Corporation, *The Component Object Model (Technical Overview)*, Microsoft Press, Redmond, WA, 1995.

[6] C. Nueman and T. Ts'o, "An Authorization Service for Computer Networks", *Comm. of the ACM*, Vol. 32, No. 9, Sept. 94.

[7] Object Management Group, *The Common Object Request Broker: Architecture and Specification, Rev. 2.0*, MA, July 1995.

[8] Oppliger, R. "Internet Security: Firewalls and Beyond", *Comm. of the ACM*, Vol. 40, No. 5, May 1997.

[9] M. Reiter and S. Stubblebine, "Authentication Metric Analysis and Design", *ACM Trans. On Information and System Security*, Vol. 2, No. 2, May 1999.

[10] W. Rosenberry, D. Kenney, and G. Fischer, *Understanding DCE*, O'Reilly & Associates, 1992.

[11] R. Sandhu, et al., "Role-Based Access Control Models", *IEEE Computer*, Vol. 29, No. 2, Feb. 1996.

[12] R. Sandhu and J. Park, "Decentralized User-Role Assignment for Web-based Intranets", *Proc. of the 3rd ACM Wksp. on Role-Based Access Control*, Fairfax, VA, Oct. 1998.

[13] D. Spooner, "The Impact of Inheritance on Security in Object-Oriented Database Systems", in *Database Security, II: Status and Prospects*, C. Landwehr (ed.), North-Holland, 1989.

[14] V. Swarup, "Trust Appraisal and Secure Routing of Mobile Agents", *Proc. of 1997 Workshop on Foundations for Secure Mobile Code (DARPA)*, March 1997.

[15] Walsh, T., Paciorek, N., and Wong, D. "Security and Reliability in Concordia", *Proc. of the 31st Hawaii Intl. Conf. on System Sciences (HICSS'98)*, 1998.

[16] http://www.engr.uconn.edu/ steve/urbsdreproj.html