Distributed Role-Based Access Control (DRBAC): Model and Architectures

DRAFT
Prepared by Ravi Sandhu
George Mason University
for
SANDIA National Laboratories

November 13, 1999

Abstract

This report describes a formal model for Distributed Role-Based Access Control (DRBAC) which meets policy requirements provided by Sandia Laboratories. The DRBAC model is based on the well-known RBAC96 model and its derivatives, but includes significant enhancements to meet the needs of a distributed application containing multiple autonomous entities. The report also identifies possible architectures for enforcing the DRBAC model.

1 INTRODUCTION

The objective of this study is to develop a formal model and identify viable architectures for Distributed Role-Based Access Control (DRBAC). This model should in general meet the needs of a distributed application containing multiple autonomous entities. More specifically, the model should meet the requirements of the application described in [AGJN99] and further elaborated in personal communication between SANDIA and GMU collaborators on this project. Moreover, the model should build upon standard definitions and models of Role-Based Access Control (RBAC).

Standard formal RBAC models, such as RBAC96 [SCFY96] are formulated in context of a single monolithic organization. In particular these models typically postulate the existence of a single security officer who has ultimate authority over the entire system. Some existing models, such as ARBAC97 [SBM99], extend RBAC96 to provide for decentralized administration but retain this assumption of a single authority with ultimate power. The assumption of a single all-powerful security administrator is fundamentally inconsistent with distributed systems composed of autonomous systems. Fortunately this assumption is not critical to RBAC96 and can be easily dropped without damaging the integrity of the model. Nevertheless, with cooperating autonomous systems it is important to include a suitable administrative component in the RBAC model to spell out clearly what autonomy is provided to each site. Hence, the need to develop a DRBAC model.

In the next section we describe our methodology for addressing this problem. This methodology has been recently developed by Ravi Sandhu [San99] and is based on four ordered layers called policies, models, architectures and mechanisms in sequence. The next three sections respectively

See Figure 1 in accompanying powerpoint document

Figure 1: 4 Layer Framework for Security Engineering

See Figure 2 in accompanying powerpoint document

Figure 2: 4 Layer Framework for Multilevel Security

See Figure 3 in accompanying powerpoint document

Figure 3: 4 Layer Framework for RBAC

describe the policy of the application domain, define the DRBAC formal model to support this policy and identify viable DRBAC architectures for implementing and enforcing this policy. Discussion of detailed mechanisms and protocols for these architectures is not within the scope of this report.

We assume familiarity with basic concepts of RBAC and in particular with the RBAC96 model which serves as the basis for DRBAC. RBAC96 is briefly reviewed in the appendix. Many papers relevant to RBAC96 are available at `www.list.gmu.edu` by following the publications link.

2 METHODOLOGY

In pursuing this task we have used the four layer framework for security engineering recently introduced by Ravi Sandhu. This framework comprises policy, model, architecture and mechanism layers as shown in Figure 1. These layers are roughly analogous to a network protocol stack with policy at the top end and mechanism at the bottom end. The responsibility of each layer is as follows.[1]

- The policy layer confronts the question as to what are the security goals. Policy descriptions are typically informal.
- The model layer makes the informal statements of the policy layer more rigorous, precise and complete. Formal mathematical models are used here.
- The architecture layer describes the high-level design of the system in terms of major components, servers, brokers, etc., and their interrelationships. Architecture descriptions tend to be informal.
- The mechanism layer consists of concrete protocols and mechanisms which are used to build real systems. At the mechanism layer the issue is how something is achieved. Mechanisms and protocols require detailed technical specifications so they can be implemented.

[1] The issue of assurance runs across all fours layers. It is mentioned here for completeness.

Thus these layers span the question space from what to how. The relationship between successive layers is many-to-many. A given policy can be supported by many models and vice versa. Similarly, a given model can be implemented and enforced by many architectures, and vice versa. Likewise for the architecture and mechanism layers. Hence, the analogy to network protocol layers.

Much of the existing work on engineering security focuses on one layer on another, or mixes up issues from different layers without clearly separating them. In our approach we clearly demarcate the issues at each layer while allowing us to identify dependencies between different layers.

Figure 2 illustrates this framework in context of classical multilevel security (MLS). Multilevel security is concerned with preventing information leakage in a classified military or national security setting. Thus there is a fixed policy of one-directional information flow which is being pursued. To articulate this policy formally we have the well-known lattice-based access control (LBAC) models [San93], also commonly known as the Bell-LaPadula model or mandatory access control. The standard architecture for implementing LBAC is a security kernel [Dep85] and a variety of mechanisms are used for this purpose. Multilevel security has been studied for almost 30 years and it is a positive confirmation that this classic area fits within our four-layer framework. In this context there is one policy and one model. There are however multiple architectures [Not94] in addition to the standard security kernel approach. There are also many mechanisms that have been used by different implementors.

In much of the early work on multilevel security there was only one policy, one model and one architecture and a multiplicity of mechanisms. This has tended to foster a view of security engineering close to the classic top-down software engineering waterfall. Even with the recognition of multiple MLS architectures we still have only one policy and model.

The four layer framework becomes much more compelling when we look at RBAC as illustrated in figure 3. In this case we have a multiplicity of policies. The ability to configure policy is one of the main features of RBAC. There are a number of RBAC models that have been published. They differ in details but they all share some common characteristics. The RBAC96 model was the first comprehensive RBAC model to be published and has emerged as the best known and most authoritative model. As noted earlier RBAC96 does need to be extended to meet the needs of distributed applications containing multiple autonomous entities. Fortunately RBAC96 is very rich in the scope of policies that it can support. In particular it has been shown that RBAC96 can be configured to enforce LBAC (or MLS) [San96] on one hand and discretionary access control [SM98] on the other. This is strong confirmation of the tremendous range of policies that RBAC96 can accommodate. There are many architectures for implementing RBAC in distributed systems, including the user-pull and server architectures [PS99, PSG00]. Conversely, each of these architectures can support models other than RBAC such as attribute-based access control. Finally, a variety of mechanisms and protocols can be used to support these architectures.

3 POLICY

In this task we are concerned with a very specific policy for a distributed application containing multiple autonomous entities. This policy is based on the the requirements of the application described in [AGJN99] and further elaborated in personal communication between SANDIA and

GMU collaborators on this project.[2]

The system consists of a number of physical sites, each of which has a number of simulation-models.[3] Each simulation-model is an autonomous entity with its own security administrator. Access control to services of each simulation-model will be enforced with respect to the roles possessed by the user attempting to make access. A particular simulation-model may recognize only a subset of the entire set of roles in the system.

The main concern of the policy is with administration of the user-role and role-permission relations. The security administrator of each simulation-model will determine what permissions are assigned to each role on that simulation-model. Revoking these permissions is also entirely under control of individual security administrators. User-role assignment requires approval of *all* security administrators of simulation-models where that role has non-empty permissions. Conversely, any single security administrator can revoke a user from a role (provided the simulation-model of that administrator has non-empty permissions for that role).

It is assumed that the security administrator of a simulation-model can assign permissions for that simulation-model to any role at any time. In particular permissions can be granted to a role X even if X currently does not have any permissions for that simulation-model. By doing so the security administrator implicitly accepts all users of X and gains the power to revoke their membership from X.[4]

It is projected that there may be a very shallow hierarchy of roles, if any hierarchy is present at all. It is also projected that a user will employ only one role at a time to access a particular simulation model. These projections have been built into the DRBAC model of the next section, but they are not really critical to the model. They may have greater ramification for the architecture and mechanisms.

System Scale

The scale of the system is as follows.

- Approximately a dozen physical locations
- Approximately 2-3 simulation-models/location
- Fewer than 100 roles
- Fewer than 100 users
- Moderate rate of change

[2]Some of the finer points of this policy need to be revised. These are mostly issues of detailed modeling.

[3]We will be very careful to use the "simulation-model" to distinguish these models from the RBAC models. The term "model" will be used by itself only to mean access control model or security model.

[4]This policy has a somewhat undesirable feature. It is possible for any security administrator Alice to revoke any user Bob from any role X. If X already has non-empty permissions in Alice's simulation-model she can revoke Bob from X by the stated revocation policy. If X does not have non-empty permissions, she can assign some permission to X to make it have non-empty permissions and then revoke Bob from X. Finally, she can revoke the permissions from X. Security administrators are presumably trusted not to do such mischief. A different or more elaborate policy can fix this problem but it may not be justified in this application. Misuse detection technlogy can also be used to detect such mischief.

It is anticipated that increased scale will be required in future.

The scale as such does not impact the details of the DRBAC model very much, but it is useful to capture it here because of its impact on the architecture and mechanism layers.

4 DRBAC MODEL

We construct the DRBAC99 model[5] by building upon RBAC96. We must also drop the requirement of RBAC96 that there is a single all-powerful security administrator. This assumption is fundamentally inconsistent with the policy given above. Fortunately it turns out this assumption is not critical to RBAC96 and can be easily dropped without damaging the integrity of the model. The main extension to RBAC96 is in the administrative component.

The basic definition of RBAC96 are used essentially unchanged. We have limited each session to a single active role as indicated in the policy. This assumption can be easily changed if desired without much impact on the rest of the model. Thus we start with the following definition.

Definition 1 [DRBAC99: Basic RBAC96 Components] The DRBAC99 model includes the following RBAC96 components.

- U, a set of users
  R a set of (regular) roles
  P a set of (regular) permissions
  S, a set of sessions
- $UA \subseteq U \times R$, user to role assignment relation
- $PA \subseteq P \times R$, permission to role assignment relation
- $RH \subseteq R \times R$, partially ordered role hierarchy
- $user : S \rightarrow U$, maps each session to a single user (which does not change)
  $roles : S \rightarrow 2^{R \cup AR}$ maps each session s_i to a single role
  session s_i has the permissions $\cup_{r \in roles(s_i)}\{p \mid (\exists r'' \leq r)[(p, r'') \in PA \cup APA]\}$
- each session can have only a single role as stipulated above

The definition of RBAC96 is completed below.

Definition 2 [DRBAC99: Extensions to RBAC96] The DRBAC99 model consists of the following extensions to its RBAC96 components defined above.

- SM, a set of simulation-models $\{sm_1, \ldots, sm_k\}$
- OP, a set of operations $\{op_1, \ldots, op_l\}$
- The set of permissions P is defined as $P = SM \times OP = \{(sm_i, op_j) \mid sm_i \in SM, op_j \in OP\}$

[5] We call it the DRBAC99 model to distinguish it from other DRBAC models that may be developed.

- $sm : P \rightarrow SM$, a many-to-one function that maps each permission to the simulation-model to which it applies so that $sm(p) = sm((sm_i, op_j)) = sm_i$
- SMA, a set of administrative roles $\{sma_1, \ldots, sma_k\}$ one for each simulation model
- The administrative roles are disjoint from the regular roles, so $R \cap SMA = \emptyset$. Also each session has exactly one regular role or administrative role (but not both) associated with it.
- $admin : SM \leftrightarrow SMA$, a one-to-one function that maps each simulation-model to an administrative role
- Each simulation-model sm_i has a unique user designated as its *security administrator*. The security administrator can assign and revoke users to and from the corresponding administrative role $admin(sm_i)$.
- Permission p can be assigned to or revoked from role r by a user who is a member of the administrative role $admin(sm(p))$. No other administrative role or user is authorized to perform this task.
- User u can be assigned to a role $r \in R$ if and only if this is approved by at least one member of every administrative role $ar \in SMA$ for which there exists $p \in P$ such that $(p, r) \in PA$ and $admin(sm(p)) = ar$.
- User u can be revoked from a role $r \in R$ if and only if the revocation is done by any member of any administrative role $ar \in SMA$ for which there exists $p \in P$ such that $(p, r) \in PA$ and $admin(sm(p)) = ar$.

5 DRBAC ARCHITECTURES

In this section we identify different architectures for enforcing DRBAC99 and discuss issues arising in these architectures. There are four components whose architectures we need to consider. For convenience we consider each one separately but there are interdependencies. We also note that that the authentication architecture is not considered here. Our discussion is limited to the authorization architecture. Moreover, hybrid architectures are also possible.

5.1 Permission-Role Assignment

By definition of the policy, permission-role assignment to regular roles is a local matter at each simulation-model. Thus each simulation model can do this task in whatever way it likes. When a permission is assigned to a role which previously had no permissions in the simulation-model in question, there may be a need to make this fact known to other places. Likewise, when the last permission is removed from a role.

5.2 Permission-Role Enforcement

Permission-role enforcement at a given simulation-model is also entirely a local matter. Once a simulation-model knows a user's role it can easily enforce appropriate permissions for that role

See Figure 4 in accompanying powerpoint document

Figure 4: Server-Mirror Architecture for User-Role Enforcement

See Figure 5 in accompanying powerpoint document

Figure 5: Server-Pull Architecture for User-Role Enforcement

See Figure 6 in accompanying powerpoint document

Figure 6: User-Pull Architecture for User-Role Enforcement

since this solely under control of the simulation-model.

5.3 User-Role Assignment

The policy requires user-role assignment to be approved by all relevant simulation-models. Thus this activity requires coordination. It also requires knowledge of which users are in the security administrator role for each simulation-model, as well as knowledge of which simulation-models need to be consulted. We can consider two extreme architectures here.

- No central coordinator. While this architecture is theoretically feasible it would not seem appropriate to have the required complex protocols in this application.
- With central coordinator. The central coordinator could be a special site for this purpose.

In either case we must accommodate unilateral user-role revocation by any simulation-model that has assigned permission to that role.

5.4 User-Role Enforcement

For user-role enforcement we have three architectures illustrated in Figures 4, 5 and 6.[6] Each box in these diagrams represents an autonomous entity.

In the server-mirror architecture the user-role database is replicated or mirrored at each simulation-model. Each simulation-model is responsible for maintaining and enforcing this information.

In the server-pull architecture, each simulation-model consults a central user-role authorization server to look up a given user's roles.

In the user-pull architecture each user first obtains digital credentials which securely specify the user's role. These credentials are presented to the simulation-models to obtain access.

At this point we do not have sufficient information to make a recommendation for choosing a specific architecture.

[6] A user-mirror architecture is not considered reasonable, since the user-enforcement is done on th user's machine.

6 CONCLUSION

In this report we have followed the four layer framework of Ravi Sandhu. We have defined the policy and the DRBAC99 model to support this policy. We have identified alternate architectures for implementing and enforcing DRBAC99. Additional research and input is required before a specific architecture can be recommended. The choice of architecture may also be influenced by the choice of mechanisms and protocols in the software architecture of the project. Architecture and mechanism would need some concurrent study before a good choice can be made.

References

[AGJN99] H.R. Ammerlahn, M. Goldsby, M. Johnson, and D. Nichol. A geographically distributed enterprise simulation system. Preprint, SANDIA, 1999.

[Dep85] Department of Defense National Computer Security Center. *Department of Defense Trusted Computer Systems Evaluation Criteria*, December 1985. DoD 5200.28-STD.

[Not94] LouAnna Notargiacomo. Architectures for mls database management systems. In M. Abrams, S. Jajodia, and H. Podell, editors, *Information Security : An Integrated Collection of Essays*. IEEE Computer Society Press, 1994.

[PS99] Joon Park and Ravi Sandhu. Smart certificates: Extending x.509 for secure attribute services on the web. In *Proceedings of 22nd NIST-NCSC National Information Systems Security Conference*, Arlington, VA, October 18-21 1999.

[PSG00] Joon Park, Ravi Sandhu, and SreeLatha Ghanta. Rbac on the web by secure cookies. In Atluri and Hale, editors, *Database Security XIII: Status and Prospects*. Kluwer, 2000.

[San93] Ravi S. Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11):9–19, November 1993.

[San96] Ravi S. Sandhu. Role hierarchies and constraints for lattice-based access controls. In Elisa Bertino, editor, *Proc. Fourth European Symposium on Research in Computer Security*. Springer-Verlag, Rome, Italy, 1996. Published as *Lecture Notes in Computer Science, Computer Security–ESORICS96*.

[San99] Ravi Sandhu. *RBAC Lecture Notes*. Various lectures (including INFS 767 in Fall 1999) since March, 1999.

[SBM99] Ravi Sandhu, Venkata Bhamidipati, and Qamar Munawer. The ARBAC97 model for role-based administration of roles. *ACM Transactions on Information and System Security*, 2(1), February 1999.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[SM98] Ravi Sandhu and Qamar Munawer. How to do discretionary access control using roles. In *Proceedings of 3rd ACM Workshop on Role-Based Access Control*, pages 47–54, Fairfax, VA, October 22-23 1998. ACM.

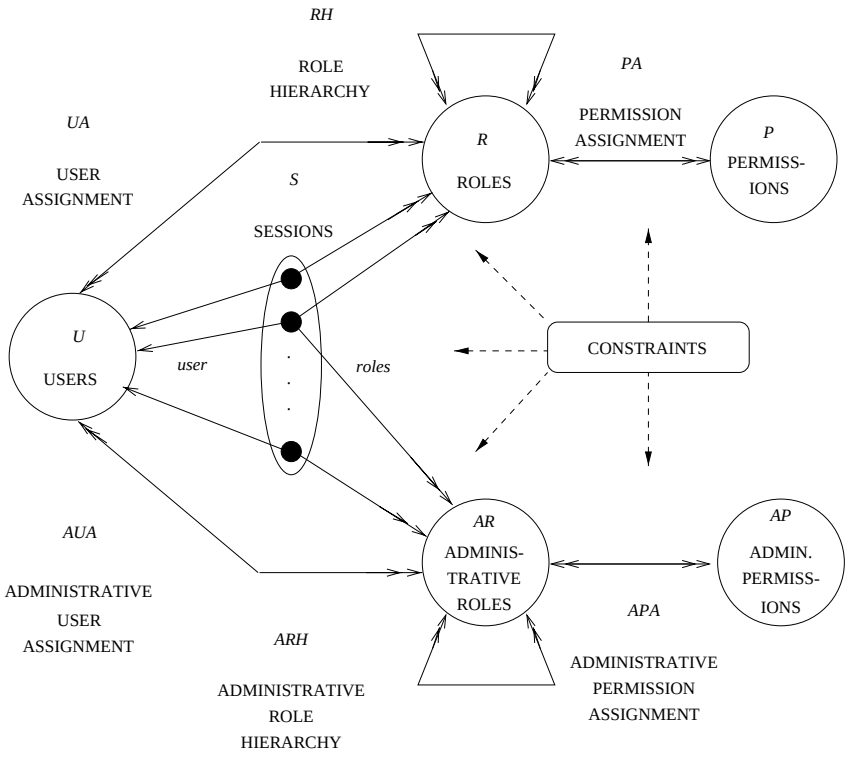


Figure 7: Summary of the RBAC96 Model

APPENDIX: THE RBAC96 MODEL

This appendix gives a brief review of the RBAC96 model [SCFY96]. Strictly speaking RBAC96 is a family of 4 models. Figure 7 illustrates the most general model in this family. For simplicity we use the term RBAC96 to refer to the family of models as well as its most general member.

The top half of figure 7 shows (regular) roles and permissions that regulate access to data and resources. Intuitively, a user is a human being or an autonomous agent, a role is a job function or job title within the organization with some associated semantics regarding the authority and responsibility conferred on a member of the role, and a permission is an approval of a particular mode of access to one or more objects in the system or some privilege to carry out specified actions. The bottom half shows administrative roles and permissions.

Roles are organized in a partial order or hierarchy, so that if $x > y$ then role x inherits the permissions of role y, but not vice versa. In such cases, we say x is senior to y. By obvious extension we write $x \geq y$ to mean $x > y$ or $x = y$. Figure 8 illustrates examples of role hierarchies by means of Hasse diagrams. In these diagrams senior roles are shown towards the top and junior roles towards the bottom.

Each session relates one user to possibly many roles. The idea is that a user establishes a session (e.g., by signing on to the system) and activates some subset of roles that he or she is a member

of. When a senior role is activated the permissions of all junior roles can be used in that session. At the same time a user assigned to a senior role may activate sessions in which only some of the junior roles are activated.

The components of RBAC96 are formally defined below.

- U, a set of users
  R and AR, disjoint sets of (regular) roles and administrative roles
  P and AP, disjoint sets of (regular) permissions and administrative permissions
  S, a set of sessions
- $UA \subseteq U \times R$, user to role assignment relation
  $AUA \subseteq U \times AR$, user to administrative role assignment relation
- $PA \subseteq P \times R$, permission to role assignment relation
  $APA \subseteq AP \times AR$, permission to administrative role assignment relation
- $RH \subseteq R \times R$, partially ordered role hierarchy
  $ARH \subseteq AR \times AR$, partially ordered administrative role hierarchy
  (both hierarchies are written as $\geq$ in infix notation)
- $user : S \rightarrow U$, maps each session to a single user (which does not change)

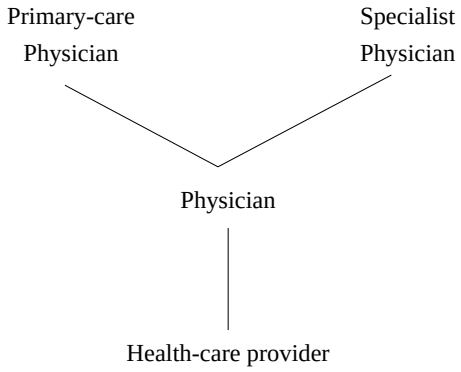
  $roles : S \rightarrow 2^{R \cup AR}$ maps each session s_i to a set of roles and administrative roles $roles(s_i) \subseteq \{r \mid (\exists r' \geq r)[(user(s_i), r') \in UA \cup AUA]\}$ (which can change with time)

  session s_i has the permissions $\cup_{r \in roles(s_i)}\{p \mid (\exists r'' \leq r)[(p, r'') \in PA \cup APA]\}$
- there is a collection of constraints stipulating which values of the various components enumerated above are allowed or forbidden.

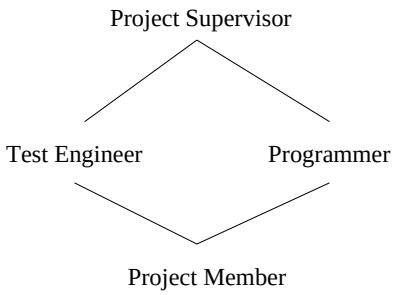
Example of constraints include mutually exclusive roles (i.e., roles that cannot be assigned to the same user) and mutually exclusive permissions (i.e., permissions that cannot be assigned to the same role).

The RBAC96 family of models consists of the following members.

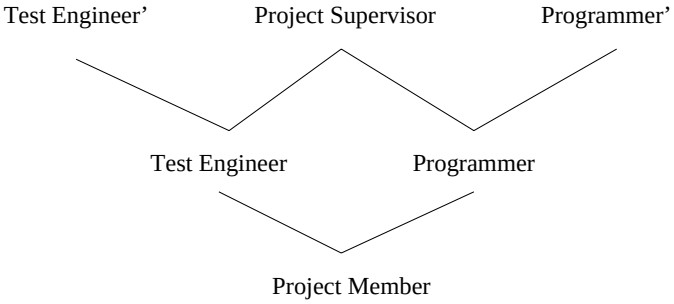
- RBAC_0: no role hierarchy, no constraints
- RBAC_1: role hierarchy, no constraints
- RBAC_2: no role hierarchy, constraints
- RBAC_3: role hierarchy, constraints



(a)



(b)



(c)

Figure 8: Examples of Role Hierarchies