Global Information Assurance Certification Paper

Role Based Access Control

Michael Lebkicher

Professor: Eric Cole

November 30, 2000

Introduction

Administration of access control remains a critical and complex aspect of Security Administration. Several models to effect this administration are Mandatory Access Control, Discretionary Access Control and Role-Based Access Control. RBAC represents a natural structure of an organization where functions are grouped into roles and users are permitted to one or more of these roles. While not a new concept, Role-Based Access Control continues to gain wider commercial acceptance as it simplifies and enhances definition, auditing and administration of security access rights

Background

The current set of security criteria and guidelines has grown largely from research and development efforts on the part of the DoD and NIST over a period of years. A well-known U.S. computer security standard is the Trusted Computer System Evaluation Criteria based on DoD security policy. The TCSEC specifies two access controls, discretionary and mandatory.

Discretionary Access Control

DAC permits the granting and revoking of access privileges to be left to the discretion of the individual users allowing them to grant or revoke access to any authority under their control without the intercession of a system administrator **(1)**. It acts as a means of restricting access to resources based on the identity of persons and/or groups to which they belong.

Mandatory Access Control

The most common MAC is used to secure military systems. MAC is a means of restricting access to data based on varying degrees of security requirements for information contained in the objects. Information is associated multi-level security requirements with labels such as TOP SECRET, SECRET, and CONFIDENTIAL **(2).**

Role-Based Access Control

RBAC is a form of mandatory access control, but not based on multilevel security requirements. Rather, access control decisions are determined by the roles individual users take on as part of an organization. RBAC tends to be modeled after the natural structure of an organization where functions are grouped into roles and users are permitted to one or more of these roles. The security policy of the organization determines role membership and the allocation of each role's capabilities Unlike DAC, RBAC's premise is that the organization, not the user, owns the objects being secured. In most implementations, users cannot sub-delegate access permissions on to other users at their discretion **(1)**.

The remainder of this paper will discuss various aspects of Role-Based Access Control and illustrate a practical implementation using Sun Solaris 8.

Role-Based Access Control

Role Definition

"In a nutshell, the essence of Role-Based Access Control is that rights and permissions are assigned to roles rather than to individual users. Users acquire these rights and permissions by virtue of being assigned membership in appropriate roles." The designation of persons and transactions as objects included within a role is based on objective guidelines defined by the organization rather than on subjective decisions of a security administrator. **(3)**

In terms or Unix infrastructures, systems administrators are not infallible and a frequent source of systems failures is human error. Classic Root authority is "all or nothing". There is no safety net for the inexperienced administrator and at the same time users do not have the authority to resolve problems, which they are technically capable of concluding. With RBAC, roles are associated with a user and define profiles that represent sets of tasks that are authorized for execution. Root authority can be segregated into appropriate packages. This reduces the opportunity for persons to go beyond their realm of expertise or to inadvertently or intentionally make a change that results in a system failure. **(6)**

Roles are group oriented. For each role, a set of transactions allocated to the role is maintained. A transaction can be thought of as a program object associated with data. A security administrator adds and deletes transactions to and from roles and also allows and revokes user memberships. **(1)** Security issues are addressed by associating programming code and data into a transaction. Access control does not require any checks on the user's or the program's right to access a data item, since the accesses are built into the transaction.

A key goal for RBAC is to simplify authorization and enable better audit capabilities so that as people change responsibilities they are simply assigned new roles. The categorization of persons within roles is segregated from the specifications of the roles themselves. The role provides a clarification and concise identification of policies for given functions. This allows for the authorities of a person to be easily documented. By contrast, in a typical access list model, one must search the entire set of authorities to develop a clear picture of a person's rights. **(11)**

Role Perspectives

Roles can be considered from different angles: **(10)**

- Group by Organization – This is a classic view where an organization is composed of varying segregated roles to make up the composite whole.
- Group by Relative relationship—This is a view that takes into account the natural interaction of members of an organization. When a person interacts by definition the role interacts. There is also relativity as it applies to the role owner. For example, a manager role differs depending on what is being managed; say projects, people or technology.
- Group by convenience – This is another classic view emphasizing ease of reference where authorizations can be manipulated and assigned in a flexible manner.
- Group by selection -- This is a view grouping by capability. It is akin to access control and used in a workflow process.

Role Hierarchies

A role hierarchy defines roles that have unique attributes and that may contain other roles; that is, one role may implicitly include the operations that are associated with another role. Operations and roles are typically subject to organizational policies or constraints. When operations overlap, hierarchies of roles can be established. Role hierarchies facilitate organizing roles to reflect authority and responsibility. Roles can have overlapping responsibilities and privileges. Users belonging to different roles may need to perform common operations. In these cases, RBAC provides an efficient way avoid repeatedly specifying these general operations for each role that gets created. **(4)**

Role Subsidiarity

Tasks should be performed by the appropriate role that has suitable responsibility regardless of the authorities implied by the structure of the organization. As previously mentioned, role hierarchy tends to organize roles according to their shared capabilities. Subsidiarity lends a different slant and to this theory and is really an extension. Role hierarchy does not fully identify organizational role complexity in terms of inheritance. Subsidiary illustrates the sometime incomplete nature of hierarchy. For example, inheritance often has exclusions due to subsidiarity. An IT development project manager should not change code even though he manages the project and depending on matrixing may even manage the programmers. His authority does not extend completely within the hierarchy. This is often a classic scenario where managers who were technicians find it hard to make the transition.

Integrity

Only authorized roles should modify data and processes and then only in authorized ways. The difficulty of the problem is determined by the complexity of the related transaction. A practical approach is simply for transactions to be certified and trusted. Bank tellers may execute a savings deposit transaction, requiring read and write access to specific fields within savings and transaction log files. An accounting supervisor may be able to execute correction transactions, requiring exactly the same read and write access to the same files as the teller. The difference is in the process executed and the values written to the transaction log file suggesting similar but different transaction objects.

Least Privilege

The principle of least privilege is important for meeting integrity objectives. It requires that a user be given only the privilege necessary to perform a job. Ensuring least privilege requires identifying what the user's function is, determining the minimum set of privileges necessary, and restricting the user to a set of roles with only those privileges. By excluding users from transactions that are unnecessary for the performance of their duties, those transactions cannot be used to circumvent organizational security policy. Through the use of RBAC, enforced minimum privilege is easily achieved. **(4)**

Separation of Duties

Separation of duties is determined from organizational policy. RBAC facilitates splitting administration of the role-user relationship from that of the role-permission relationship. **(3)** Since many users in an organization typically perform similar functions and have similar access rights, user functions are separated by role. Separation of duties requires that for particular sets of transactions, no single role be allowed to execute all transactions within the set. As an example, there are separate transactions needed to initiate a payment and to authorize a payment. No single role should be capable of executing both transactions. Separation of duties by role is valuable in deterring fraud since fraud can occur if an opportunity exists for collaboration between various job-related capabilities.

A role can be further qualified with affiliation. A person's access could limited to a geographic or departmental boundary. For example, a role of branch manager could be qualified by an affiliation to a particular branch thereby conferring branch manager permission only within that branch.

Both static and dynamic forms of separation exist. Static separation means that roles which have been specified as mutually exclusive cannot be included together in a user's set of authorized roles. Dynamic separation means that while users may be authorized for roles that are mutually exclusive, those exclusive roles cannot be active at the same time. In other words, static separation of duty enforces the mutual exclusion rule at the time of role definition while dynamic separation of duty enforces the rule at the time roles are selected for execution by a user. **(9)**

Role Activation

Once a role is established it can be enabled dynamically and only under certain conditions within a pre-defined scope. These constraints are separate from the typical RBAC policy constraints of *where*, *when*, and *what*. The most common additional activation considerations include: **(10)**

- Event-triggered – Activation occurs depending on particular circumstances and is a composite condition where other conditions or events must be enabled for the role to activate.
- Qualification -- This is similar to typical role criteria but reflects a dynamic attribute that suggests a person may have a different role depending on circumstance.
- Delegation or transfer -- This is related to *hierarchy* and also to *grant*. It speaks to whether or not the granter retains the authority as it is passed on within a hierarchy. As an example, a manager could *delegate* authority to a subordinate, as empowerment while the manager would *transfe*r authority to a replacement if the manager left the organization.
  There are other theoretical considerations some of which are:
  - What conditions constitute the requirements where delegation/transfer can occur?
  - How often can the actions take place in series down the chain?
  - Can the original owner re-obtain the authority in a closed loop when?

Role Administration

One of RBAC's advantages is the administrative capability it facilitates. Once the transactions of a role are established within a system, they tend to remain relatively constant or change slowly over time. This simplifies the understanding and management of privileges. Roles can be updated without having to directly update the privileges for every user on an individual basis. **(2)** The administrative task consists of granting and revoking membership to the set of specified named roles within the system. When a new person enters the organization, the administrator grants membership to an existing role. User membership into roles can be revoked easily and new memberships established as job assignments dictate. **(1)**

A strength of RBAC as an access control mechanism is its flexibility in that it secures at the level of operations and an operation may theoretically be anything. This is contrasted to other access control mechanisms where bits or labels are associated with information record. These bits or labels indicate relatively simple operations, such as, read or write, which can be performed on an information record. A goal for implementing RBAC is to allow operations associated with roles to be as general as possible while not adversely impacting the administrative flexibility or the behavior of applications. **(5)**

Role-Based Management

RBAC is an extension of access control mechanisms and a tool to simplify and improve administration of access rights within an organization. A superset of RBAC is Role Based Management that is based in organizational roles and includes duties along with authorities. RBM identifies the concepts of obligation policies and object orientation. **(12)**

- Obligation policies describe necessary actions on the part of roles in addition to authorizations. For example, security administrators are obligated ensure proper configurations related to vulnerability analysis and to maintain surveillance with intruder detection software.
- Object orientation suggests that an emphasis on defining a role in terms of obligation and authority in terms of a defined set of management policies will allow for organizational restructuring without the need for policy manipulation.

RBM illustrates more of the complexity of formalizing and administering role definitions. Its roles are developed from role classes to speak to the need to segregate subsets of multiple similar roles in organizations. It is presented here merely as an example of the extensibility of RBAC to formally administer and achieve an organization's authorization paradigm.

An Implementation

Solaris 8 provides a simplified RBAC function, the essentials of which will be discussed as a practical illustration. Sun's implementation of RBAC includes these features: **(7)**

- Authorization - A right that is used to grant access to a restricted function
- Execution profile - A facility for grouping authorizations and commands with special attributes; for example, user and group IDs
- Role - A special user account intended for performing a particular set of tasks

Sun's implementation of RBAC uses four files to facilitate privileged access. **(7)**

Extended User Attributes Database -- **user_attr --** (user:::attr)

This file associates users and roles with authorizations and execution profiles. It also allows roles to be assigned to a user. A role is a form of user account that is used for specific tasks. Roles cannot be assigned to other roles.

Users are granted privileged access within this file by associating them either with profiles, authorizations, or roles. Profiles are contained in **prof_attr** and point to either authorizations in **auth_attr** or commands in **exec_attr**. A user will *su* to obtain a given role's authority. To obtain a profile's commands, they are run in special shells pfsh (Bourne), pfcsh (C), and pfksh(Korn). These shells execute with special operating systems interfaces to effect RBAC. **(6)**

Fields within the file are:

- user – user or role name
- attr - Optional list of key-value pairs that describe the security attributes to be applied when the user runs commands. There are four valid keys:
  - *auths* specifies a list of authorization names chosen from names defined in the **auth_attr** database.
  - *profiles* contains a list of profile names chosen from **prof_attr.** A profile associates commands and attributes and determines which ones a user can execute.
  - *roles* can be assigned to the user using a list of role names.
  - *type* is defined as "normal" for a user record or as "role" for a role record.

Authorization Attributes Database – **auth_attr** –
(authname:::short_desc:long_desc:attr)

This file defines authorizations and their attributes and identifies the associated help file. An authorization grants access to a restricted function. It is a unique string that identifies what is being authorized as well as who created the authorization. It may be assigned directly to users or roles in the **user_attr** database or to execution profiles that in turn are assigned to users.

Fields within the file are:

- authname - A unique character string used to identify the authorization in the format *prefix*.[*suffix*]. The prefix identifies the granter of the authority and the suffix indicates what is being authorized. When authname ends with the word

grant, the authname serves as a grant authorization and lets the user delegate related authorizations to other users. This is a noted exception that is DAC-like.

- short_desc - A short descriptive name suitable for displaying in user interfaces, such as in a scrolling list in a GUI.
- long_desc - A longer description suitable for display in the help text of an application.
- This field identifies the purpose of the authorization, the applications in which it is used, and the type of user interested in using it.
- attr - An optional list key-value pairs that describe attributes of an authorization.
    - *help* identifies a help file in HTML.

Profile Execution Attributes Database -- **exec_attr** -- name:policy:type:::id:attr

An execution attribute associated with a profile is a command that can be run by those users or roles to which the profile is assigned.

Fields within the file are:

- name - The name of the profile.
- policy - The security policy associated with this entry. Currently, suser, the Root superuser policy model, is the only valid policy entry
- type - The type of entity. Currently, the only valid type is cmd (command).
- id - A string identifying the entity. Commands should have the full path or a path with a wild card. Arguments can be specified by creating a script and using this field to select it.
- attr - An optional list of key-value pairs that describe the security attributes to apply to the entity at execution. There are four valid keys: euid, uid, egid, and gid. These are typically used to grant Root authorities.
    - *euid and uid* contain a single user name or a numeric user ID. Commands designated with euid run with the effective UID indicated, which is similar to setting the setuid bit on an executable file. Commands designated with uid run with both the real and effective UIDs.
    - *Egid and gid* contain a single group name or numeric group ID. Commands designated with egid run with the effective GID indicated, which is similar to setting the setgid bit on an executable file. Commands designated with gid run with both the real and effective GIDs

Execution Profile Attributes Database – **prof_attr** – (profname:::desc:attr)

This file defines profiles, lists the profile's assigned authorizations, and identifies the associated help file. An execution profile is a way to group authorizations and commands with special attributes, and assign them to users or roles.

Fields within the file are:

- profname - The name of the profile.
- desc - A long description. This field notes the purpose of the profile.
- attr - An optional list of key-value pairs that describe the security attributes to apply to the object upon execution. There are two valid keywords
    - *help* identifies a help file in HTML.
    - *auths* specifies a list of authorization names chosen from those names defined in the auth_attr file.

The following is an example of using RBAC to delegate to a departmental user the ability to administer removable media and printers. The example is incomplete and does not represent a full set of commands necessary for either function. The elements of the example were obtained from Sun's Administration Guide and have not been validated for accuracy.**(7)**

Johnsmith is authorized to set the system date and obtains role authority from *deptrole*
Deptrole authorizes printer and device administration by pointing to related profiles

Extended User Attributes Database -- user_attr -- (user::::attr)
johnsmith::::type=normal;auths=solaris.system.date, profiles=All:roles=deptrole
deptrole::::type=role, profiles= Printer Management,Device Management

All authorizes standard Unix commands
Printer and *Device Management* authorize those functions. *Printer* points to the Execution Database and *Device* points to the Authorization Database.

Profile Attributes Database -- prof_attr -- (profname:::desc:attr)
All: standard Solaris user:help=All.html
Printer Management:::manage print jobs:help=printmgmt.html
Device Management:::control access to removable
media:auths=solaris.device.*:help=devmgmt.html

The Execution database identifies *lp* and *cancel* as example commands. The printer entries are only a sample subset of the full set of entries.

Execution Attributes Database -- exec_attr -- name:policy:type:::id:attr
Printer management:suser.cmd:::/etc/init.d/lp:euid=0
Printer management:suser:cmd:::/usr/bin/cancel:euid=0

The Authorization database contains entries for system date and device allocation and configuration. This device entries are only a sample subset of the full set of entries.

Authorization Attributes Database -- auth_attr -- authname:::short_desc:long_desc:attr
solaris.system.date:::set date and time::help=sysdate.html
solaris.device.:::device allocation:help=DevAllocHdr.html
solaris.device.allocate:::allocate device;help=DevAllocate.html
solaris.device.config:::configure device:help=DevConfig.html

Summary

Access is the ability to do something with a computer resource (e.g., use, change, or view). Access control is the means by which the ability is explicitly enabled or restricted in some way. Computer-based access controls can prescribe not only who or what process may have access to a specific system resource, but also the type of access that is permitted.

Access control decisions are often based on the roles individual users take on as part of an organization. A role specifies a set of transactions that a user or set of users can perform within the context of an organization. RBAC provide a means of naming and describing relationships between individuals and transactions, thus providing a method of meeting the secure processing needs of many commercial and civilian government organizations.

Under the RBAC framework, users are granted membership into roles based on their competencies and responsibilities in the organization. The operations that a user is permitted to perform are based on the user's role. User membership into roles can be revoked easily and new memberships established as job assignments dictate. Role associations can be established when new operations are instituted, and old operations can be deleted as organizational functions change and evolve.

A properly administered RBAC system enables users to carry out a broad range of authorized operations, and provides great flexibility and breadth of application. Security administrators can control access at a level of abstraction that is natural to the way that enterprises typically conduct business. This is achieved by statically and dynamically regulating users' actions through the establishment and definition of roles, role hierarchies, relationships, and constraints.

Major advantages to RBAC are flexibility and low overhead. Flexibility allows for the enforcement of least privilege, conflict between duties, dynamic and/or static separation of duties. It allows for minimal effort in allowing and revoking the user's role based on job and responsibilities. Decentralization of administrative tasks is also made possible and allows RBAC to reflect the current tendency towards distributed organizational structure**. (8)**

References

1. Ferrailo & Kuhn**.** "Role Based Access Controls." 15th National Computer Security Conference 1992.
URL http://hissa.ncsl.nist.gov/rbac/paper/rbac1.html 11/1/00.

2. Barkley. ""Application Engineering in Health Care". 1995. Second Annual CHIN
URL http://hissa.ncsl.nist.gov/rbac/proj/paper/paper.html 11/1/00.

3. Sandhu, Ravi. "Report on the First ACM Workshop on Role-based Access Control, Gaithersburg, Maryland". Dec. 1, 1995
URL: http://www.chacs.itd.nrl.navy.mil/ieee/cipher/old-conf-rep/conf-rep-RBAC96.html 11/1/00.

4. "An Introduction to Role Based Access Control." NIST CSL
URL: http://csrc.ncsl.nist.gov/nistbul/csl95-12.txt 11/1/00.

5. Barkley. " Implementing Role Based Access Control Using Object Technology."
First ACM Workshop on Role-Based Access Control. 1995
URL**:** http://hissa.ncsl.nist.gov/rbac/rbacot/titlewkshp.html 11/1/00.

6. Secure Enterprise Computing with the Solaris 8 Operating System
URL: http://www.sun.com/software/white-papers/wp-s8security/ 11/1/00

7. Role Based Access Control; Solaris 8 System Administration Guide, Volume 2
URL:
http://docs.sun.com/ab2/coll.47.11/SYSADV2/@Ab2PageView/26238?Ab2Lang=C&Ab2Enc=iso-8859-1 11/1/00.

8. "Role Based Access Control." 15th National Computer Security Conference
1992.
URL: http://hissa.ncsl.nist.gov/rbac 11/1/00.

9. Barkley, Kuhn, Rosenthal, Skall, Cincotta. **"**Role-Based Access Control for the Web". 1998. CALS Expo International & 21st Century Commerce 1998: Global Business Solutions for the New Millennium.
URL: http://hissa.ncsl.nist.gov/rbac/cals-paper.html 11/1/00.

10. Goh, Cheh; Baldwin, Adrian. Towards a More Complete Model of Role. 1998.
URL: http://www.hpl.hp.com/techreports/98/HPL-98-92.html

11. E. Lupu, D. Marriott, M. Sloman, N. Yialelis. A Policy Based Role Framework for Access Control ACM/NIST Workshop on Role-Based Access Control. 1995
URL: http://www-dse.doc.ic.ac.uk/~ecl1/papers/rbac95/rbac95.pdf

12. E. Lupu, M. Sloman. Reconciling Role Based Management and Role Based Access Control., Second Role Based Access Control Workshop. Nov. 1997
URL: http://www-dse.doc.ic.ac.uk/~ecl1/papers/rbac97/Rbac97.pdf