OBSERVATIONS ON THE REAL-WORLD IMPLEMENTATION OF ROLE-BASED ACCESS CONTROL

Burkhard Hilchenbach
Schumann Security Software, Inc.
314 Marshall Road
Laurel, MD 20707 USA
Phone: (301) 483-8807
Fax: (301) 483-8349
E-Mail: BHilchenbach@schumannsoftware.com

ABSTRACT

Role-based access control (RBAC) is an emerging concept for security administration for large and decentralized computing environments. The Security Administration Manager (SAM) of Schumann Security Software, Inc. (Schumann) is a tool for enterprise-wide security management that implements many RBAC concepts. Schumann has accrued extensive experience while successfully implementing RBAC at large corporations. This paper summarizes this experience and compares it with the standards which are currently suggested for RBAC. In addition, it briefly discusses issues relating to the migration from conventional security administration to RBAC.

KEYWORDS

Role-Based Access Control, RBAC, Mandatory Access Control, MAC, Discretionary Access Control, DAC, Computer Security, Security Software, SAM, Role Hierarchies, Job Functions

1. Introduction

Security administration in large computer environments is a complex and expensive task. Many companies handle it by giving security administrators ownership of all data. If an update is required, a more or less automated workflow is in place to notify the administrator. This process is slow and error-prone. This practice is not mandatory access control (MAC) (no data labeling, etc.), it is rather a 'degenerated' form of discretionary access control (DAC).

RBAC is considered an alternative to MAC and DAC. RBAC is actually a variant of MAC, however with a new approach on how to organize privileges. Still, it allows for data ownership, but the owner connects 'his' data to roles rather than to actual IDs. Existing security systems already provide primitive elements of RBAC (e.g., user groups), but these features are fragmentary and not standardized. The National Institute of Standards and Technology (NIST) is a driving force behind the move to standardize RBAC. This paper compares the experiences with SAM to the suggested standards. Prerequisite for the understanding of this paper is a familiarity with the basic concepts of RBAC, as for example described in [3].

2. SAM - An Overview

This chapter contains a very brief description of SAM, in order to evaluate the value of the experience with SAM in the context of RBAC.

SAM is a product for enterprise-wide security administration. It was developed by the German-based consultance Schumann AG, and has been commercially available since 1993. With SAM, all security systems and all application security of an enterprise can be administered from a single point of control. SAM does not replace the existing security systems; instead, it uses a 'manager of mangers' (MOM) architecture. SAM interfaces with the systems, loads the security databases, and consolidates them into a system-independent conceptual model. Then, it allows administration thru the generalized entities. For example, a SAM user definition consolidates all user IDs for an employee across all platforms. A deletion of the SAM user will result in immediate deletion (no scripting) of all IDs belonging to that employee across the enterprise.

SAM's ability to centrally administer and audit the security of the whole enterprise proves major benefit. Of course, this alone does not implement RBAC. SAM also provides a number of powerful RBAC features to administer the data. SAM is designed to be the link between the high-level company security policy and the low-level security systems. It recognizes role descriptions at a conceptual level and translates them into the native syntax of the security systems/applications. These features, together with the practical experience the Schumann staff has with implementing RBAC, make SAM interesting in view of the RBAC discussion.

3. Properties of RBAC Systems

This chapter discusses the RBAC-related features of SAM and compares them with the suggested RBAC standard.

3.1. RBAC as an Enterprise-Wide Task

For corporations, roles are actually descriptions of job functions. In the (much-used) example of a bank, roles may be defined to describe a 'teller', a 'branch manager', or an 'account representative'. In heterogeneous computing environments, a role description virtually always includes privileges across several platforms (e.g., MVS and UNIX) and within numerous applications. It is a difficult and time-consuming process to redundantly define roles and administer users across the enterprise. Having a separate RBAC system on all the platforms is helpful, but still requires a lot of work: a TELLER role is needed on the Client/Server DBMS, another TELLER role must be defined on the mainframe access control system, and so on.

Roles are by their nature not limited to a single operating system or platform. The value of a comprehensive RBAC system is fully realized when applied at the enterprise level. Future standards may even define RBAC on two levels: A 'local RBAC' standard for single applications and a 'global RBAC' standard for enterprise-wide security administration systems.

3.2. Roles Versus Groups

Many existing security systems support the concept of grouping. A group is a named collection of users. It can hold a set of privileges to resources. Users may be members in one or more groups. A user that is a member in a group inherits the privileges of the group.

Groups implement some aspects of roles as defined for RBAC. For example, a group called 'teller' can be used to implement the role of a bank teller. All tellers' IDs belong to the 'teller' group. The question groups beg is this: Is RBAC essentially a very sophisticated kind of group concept, enhanced by features like group hierarchy, cardinality, dynamic and static separation, and others?

SAM makes a major distinction between groups and roles, both logically and technically. It features an enhanced version of groups, and *additionally*, roles. SAM *groups* are a versatile tool that can be used for a multitude of tasks. The three most important ones are listed below, and *one of them* is to use the groups to depict a role hierarchy as suggested by NIST(‘group-role trees’). SAM roles are a second entity type. They are designed to describe a job function in a way that goes beyond the features of roles as described in the NIST model. They are located between the group-role trees and the users. They can also be perceived as a special type of group-role tree leaf.

This chapter describes the different types of group usage including group-role trees. The following chapter describes the SAM roles, their main features, and their interaction with groups. For clarity, this text will describe the relationship of a user to roles and groups with different terms: a user is *connected* to a role, but the user is a *member* in a group.

Typically, groups are used for three different tasks:

1. Groups are used to collect sets of privileges that belong together in a technical sense rather than by the 'semantics' of a role. A typical group is called TSO-USER. It contains all privileges required in order to run under TSO. This group is not a role, but it can be *used* by a role like TELLER: Because a teller needs to work under TSO, the TELLER role enforces the membership in the group TSO-USER.
2. Groups are used to hold access rights that can be organized by an external hierarchy. The most common example is a geographical hierarchy. For example, a file hierarchy represents authorizations to files containing world-wide data on level 0. Accesses to files about the US are stored on level 1; those to single state files on level 2; and finally those to county files on

level 3. It is important not to confuse this kind of hierarchy with a role hierarchy! This hierarchy is generated by an external structure (in this case: geography), not by the structure that may be found in the roles of an organization.
3. Groups are used for role trees. The groups in a role hierarchy hold privileges and automatically inherit privileges from parent groups to child groups and finally to users. These group-role trees cover the requirements defined by NIST for role-to-role relationships.

Administrators using a system like RACF are already using a simple three-level role tree:
1. The ALL-group in RACF contains privileges for all users, thus representing the root of all hierarchies.
2. The user groups in RACF hold privileges for a specific role.
3. On the lowest possible level, access rights are given to single users for special tasks.

Other systems like Novell Netware have more sophisticated role trees, but mostly with a fixed number of levels. The SAM group-role trees allow any number of intermediate levels.

We made the following real-life observations regarding group-role trees:
1. The role hierarchies are typically rather flat. There are seldom more than three levels between the root and single user.
2. We have very rarely come across the request to allow for a more general type of connectivity between the roles than the one a hierarchy provides. Undeniably, there may be situations in which it would be desirable to be able to define a general (cycle-free) graph. On the other hand, for many customers this would actually decrease the overview and complicate things. We think that the implementation effort for this (cycle detection, etc.) is better spent on other areas.
3. We have not yet had a case where a customer wanted to implement a cardinality, i.e. to have an upper limit of possible members in a role. We have yet to find that the concept of cardinality is of major practical or commercial interest.
4. There are always exceptions. For example, a part of a bank hierarchy could be EMPLOYEE -> TELLER -> SUPERVISING_TELLER -> BRANCH_MANAGER. This hierarchy makes sense organizationally and the inheritance works for 95% of all privileges. The remaining 5 % are exceptions in which a generally superior, more specialized role has fewer privileges than the inferior one! For example, we encountered a case where branch managers were only allowed to *read* files which their tellers (basically their subordinates) were allowed to *update*. For systems that always enforce inheritance, this destroys the father-son relationship of the corresponding roles and forces the administrator to put them side-by-side. We found that RBAC systems should be able to handle exceptions. For example, the 'read' access defined for BRANCH_MANAGER could override (rather than: add to) the 'update' coming from the TELLER role.

3.3. SAM Roles: The Combination of Role Trees, Individuality, and Locality

What factors are required to fully describe a job function of a user? It is very difficult to give a generalized answer to this question. In dealing with the real-life situations we found that privilege-inheriting roles like those in group-role trees are only one of three important factors.

The easiest way to explain this is using the example of a European bank for which we implemented RBAC. Among other tasks, we had to redefine the privileges for tellers. First of all, a group-role tree was defined. One of the nodes, TELLER, described which kind of resources tellers had to have privileges for.

The second factor consisted of security-relevant attributes of the individual. No existing security system just manages privileges, they also contain user attributes that are important in the context of security administration. The most elementary kind of attribute enforces the plain existence of a user in a system (for example: All tellers must be defined in UNIX system 'ABC'!). In the banking example, all users had an attribute ‘probation period’. Tellers in their probation period were allowed to read, but not to update some datasets. Thus, the user attribute 'probation period' was security-relevant. Of course, it would have been possible to describe this situation with two roles, namely 'teller in probation period' and 'teller not in probation period'. The problem was that there were five security-relevant attributes that led to an explosion of the number of roles needed. If there were an average of only five possible values for each of the attributes, this would lead to 5^5 (3125) possible roles for TELLERs only.

The third factor that influenced the privileges of an ID was a geographical hierarchy. The hierarchical structure of some access rights has been discussed in the previous chapter about groups. At this bank, some country-wide files were accessible to all tellers, others only to a teller working in a certain region, still others were branch-specific. We set up a group hierarchy that depicted this structure. The individual attribute 'branch' of a teller pointed to a certain leaf in the tree; the access rights from upper nodes were inherited to that leaf.

These tellers are a strong example for *individuals* having *roles* at a *location*. Access rights frequently must combine these three factors: individuality, role, and locality. For example, a teller (role) 'John Doe' is in the trial period (individuality) and works in Berlin (locality). The role tree role TELLER defines that he has access to datasets with the low-level qualifier CUSTOMER.ACCOUNTS. The locality defines that the high-level qualifier for the datasets is BERLIN. The individual attribute 'trial period' defines that he has read access only. The combination of the three factors leads to 'read' access to BERLIN.CUSTOMER.ACCOUNTS.

As any example, this one can not match all possible situations. However, it is very typical and shows that a group-role tree inheriting privileges on its own is usually not sufficient to describe access control efficiently. If we had only had group-role trees for this customer, hundreds of roles would have been necessary for tellers alone, corrupting the very concept of roles.

SAM-roles are designed to solve this problem. SAM-roles can hold privileges and attributes (individuality), make users member in a certain node of a geographic or other hierarchy (locality), and also connect them to role trees (role). The SAM-roles describe the job function not as a fixed set of privileges, but rather as a skeleton which has to be combined with the other factors to create the complete access right ‘set’.

A user can be connected to more than one SAM-role, but the roles themselves can not be structured hierarchically. Although implemented with different means, logically SAM-roles can be conceived as a special and powerful type of leaf in a group-role tree. They work like template users holding attributes, privileges, memberships in geographical hierarchies, and pointing to upper nodes in the group-role tree.

A solution for the banking situation described above was not supported by SAM off-shelf. Nevertheless, SAM is probably the one product on the market that comes closest to that goal. Currently requiring customization are the descriptions of the relationships between the elements, for example, how the individual attribute 'trial period' will affect the privilege to

CUSTOMER.ACCOUNTS. By its very nature, this portion will always need some customization, however, we are working on a standardized way to create it.

The RBAC standard may accomodate the requirement that a role must be able to control *every* attribute or privilege which may be given a user in a security system. A role might look more like a template user than like a group in the traditional sense. This requirement could be called the 'Rule of completeness': A role definition must be capable of completely holding all definitions that are needed to fulfill this role.

3.4. Role Intersections and Contradictions

In the RBAC approach of NIST, roles are exclusively utilized in order to grant privileges to users. As discussed above, roles should also be able to handle user attributes. Even the privileges themselves have attributes: For example, privileges in a lot of security systems allow to specify START-TIME and END-TIME. The privilege is valid only if used within the time range as specified by these values (for example, from 9 AM to 5 PM).

Unfortunately, attributes add a new dimension of complexity to RBAC. For an attribute like START-TIME, there are basically four ways a role can control its value:

- The role enforces an explicit value: All tellers have access starting from 9 AM.
- The role enforces that START-TIME is empty: Tellers' access to this resource *must not* have a start time restriction.
- The role does not specify anything for START-TIME: There may be a time restriction for a specific teller, given either manually by the administrator or by another role, or there is no restriction.
- The role enforces a value interval: The START-TIME for tellers may be between 7 AM and 10 AM, with a default of 9 AM.

A role in an RBAC system should at least allow for definition of a specific value, or to leave the attribute unspecified.

Attribute values lead directly to a second, larger problem: intersecting and contradicting definitions of roles. What if a user is member in the role TELLER *and* the role BRANCH_MANAGER, but both roles need to enforce a different START-TIME? Role contradictions can even occur on a higher level. What if one role gives 'write' access to a file, and another explicitly denies access to that file (explicit denials are a special case of privileges supported by most access control systems)? The experience with SAM shows that in all real-life environments, RBAC will have to handle intersecting and/or contradicting roles. The RBAC design of NIST contains the concepts of static separation (a user may not be connected to two roles at the same time) and dynamic separation (a user may not use the privileges of two roles at the same time). These interesting concepts allow the administrator to describe certain mutual exclusions that will avoid a lot of conflicts upfront. However, they do not replace a well-defined conflict handling of the RBAC system for all the (very realistic) cases where the administrator *is not aware* of a conflict.

Finally, the system behavior must be defined when a user changes the role, for example, from TELLER to BRANCH_MANAGER. What happens to all the definitions from the old role which are unspecified in the new role? Are they deleted or not? Early SAM versions deleted the old definitions and inserted the new ones, which surprisingly turned out to be a serious flaw. For example, even if both roles required the definition of the user in RACF, SAM deleted the RACF user ID and reinserted it. This resulted in a password reset -- something the administrator definitively did not want to achieve by changing the role!

The disconnection of roles remains a critical issue. If the TELLER role is simply disconnected, SAM will remove all privileges that originated from that role. This is not always desired, especially if some of the privileges were manually altered and the administrator wants to keep these changes.

It is outside the scope of this paper to discuss in detail the issues in the context of intersections, contradictions, and change management. The bottom line is that the experience with SAM shows that these issues need careful consideration when defining a RBAC standard because of their high relevance in practice.

3.5. Role Types and Individual Privileges

SAM recognizes different types of roles. The strongest one essentially implements the NIST standard. Definitions of the role are copied to the user and enforced at all times. This is great news for every auditor: only by looking at the role, he/she can tell which access rights all tellers have. Even when using that strong role, a SAM administrator may add additional privileges to the individual users which are connected to the role. This already seriously weakens the NIST standard, where every privilege must be defined by roles. It would be easy for Schumann to add that restriction -- however, no customer ever asked for it. Even if it were implemented, the administrator could still misuse this 'clean' role concept by writing a user-specific role like ROLE_OF_JOHN_DOE and attach it to only one user. There are no short-cuts for structured thinking and the disciplined use of roles. No system can actually force the administrator to work role-based; it can only support and simplify it. RBAC is primarily a school of thought, not a collection of tools. To our thinking, it will not be of much use to prohibit authorizations given directly to users. At one of our client sites, we reduced the amount of privileges given to single user IDs by 95%. The remaining 5% are carefully monitored. This is probably the highest practical level of role-based work that can be achieved in real environments.

SAM also serves weaker role types. The weakest one simply copies its definitions over to the user and releases control altogether afterwards. This is certainly not a type of role 'control' NIST (and Schumann) envisions, but it can help in step-by-step migration to RBAC.

3.6. Separation of Duties

Although SAM is a central point of control, security maintenance does not have to take place at a central location and by one person. On the contrary, the RBAC concept encourages the decentralization of administration and the separation of duties. For example, the central security administrator defines and maintains the roles, while the decentral administrators (e.g., the branch managers) use the roles to quickly administer IDs. This relieves the administrator from day-to-day work; first by the benefits of RBAC itself, but also by assigning the day-to-day work to decentral administrators. The separation of duties is an important part of the total concept of RBAC.

4. RBAC Implementation

As mentioned above, RBAC is more a school of thought than a collection of software features. For companies, the implementation of RBAC means no less than the business process reengineering (BPR) of their security administration. Starting from scratch is simple, but beginning in a complicated environment with dozens of security systems, tens of thousands of user definitions, and millions of authorizations, can be very difficult. Strictly speaking, the migration to RBAC is not within the scope of work of a standards organization like NIST. However, thinking about the migration process in advance provides a lot of insight into the strengths or weaknesses of a standard and helps to make the standard widely accepted. For this reason, this chapter briefly lists some aspects of the migration.

For Schumann, the necessity to deal with the migration process was critical. We would probably not have sold a single copy of SAM if we could not show our customers a realistic way to RBAC. Because the initial situations of companies are very different, this process is very difficult to standardize. Still, our experience allows us to identify some common factors of a successful migration to RBAC:

- Each RBAC system must have the ability to load the existing security definitions ('Initial Load') and to administer them as they are. Systems that have to start from scratch or do not administer the old structures will not have the slightest chance of success. An incremental way for implementing RBAC on top of preexisting systems is needed.
- SAM is designed to be the link between the role descriptions (in the company security policy) and the security systems. This leads to two approaches for RBAC implementation: starting from the policy side or from the security system side.
  1. The *top-down* approach starts from the role descriptions. Very often, there is no policy at all, so we have to start from scratch by defining all privileges that, for example, a teller should have. A role TELLER is created holding these privileges. Teller IDs are connected to the role TELLER and the role takes control over the existing privileges at the IDs. The remaining privileges (that are not controlled by the role) are revised. The goal is to minimize the number of these privileges.
  2. The *bottom-up* approach starts from the existing teller definitions. A teller ID is searched whose privileges are as close as possible to a desirable standard. A role TELLER is now created after this ID. The automatic creation of a role after an existing user ID is a feature each RBAC system should have. This teller is now connected to his own role, which does not result in any modification of his/her privileges. Other tellers are added, and again, the number of privileges that are not controlled by the role is minimized. Finally, the role is described as the de-facto standard for tellers.

  We have used both approaches for SAM, with the majority of our customers preferring the bottom-up method, or a mixture of both.
- The existing security definitions are always inconsistent and wrong. A simple comparison of users that are supposed to have the same role will show immense differences and will be of little help in the definition of the role.
- The best way to define roles is to set up a taskforce consisting of 'old hands'. Their knowledge is the best source of information. A taskforce consisting of 'outsiders' only will not be efficient. It is also a bad idea to evaluate privileges by looking at their creation date and other kinds of 'historic' information. This data is always extremely misleading. Instead, the taskforce must set up an evaluation method that measures existing privileges based on the actual needs of a specific role. Traces of current privilege usage may be helpful.
- Access control systems are business critical applications. Updates must be carried out very carefully. A safe implementation strategy must be in place. For a transition period, this strategy must include a fallback solution that reestablishes the old privileges if the privileges from the roles are not sufficient. During the creation and connection of a role like TELLER,

the existing privileges for some of the teller IDs will change. The system must be able to track these updates, and a plan must exist of how to define the new policies in the roles ('Consistency Maintenance').

- During the implementation of RBAC, user privileges are homogenized and simplified. In reality, this includes the deletion of superfluous privileges for users, often on a surprisingly large scale. A factor that should not be underestimated is the psychological effect this has on the users. Even if privileges are removed that are *completely superfluous*, questions arise as to why they were removed and employees may feel threatened.
- The implementation of RBAC frequently opens up questions of a very general kind: What resources have to be secured, and which user attributes are security-relevant? The (often implicit) approach of companies to secure everything and to take everything into account leads to massive and unnecessarily complex sets of security definitions. RBAC asks for simplifications and the open discussion of these issues. At one of our customer sites, implementing RBAC led to a reduction of security-relevant user attributes from >100 to 5.
- The implementation of RBAC contains a lot of manual work. For some SAM customers we have written generators that automatically create roles from existing user definitions. Yet, all generators were customized programs. Their program logic was heavily dependent on the specific situation. Large portions of the work will remain manual.
- RBAC can not be achieved by a stand-alone effort of the security administration department. Input from the organization department (role definitions, workflows, etc.), the human resources (HR) department (user data), and other departments is essential. The definition of roles will even sometimes affect the work of these departments. At one of our customers, the data HR was maintaining was slightly changed in order to serve as input for automatic, role-based creation of IDs. This had benefits not only for security administration, but also for HR.

5. Conclusions

Our experience with SAM indicates that the following issues should be carefully considered when designing a standard for RBAC:

- Roles are by their nature enterprise-wide. Standards for 'local' and 'global' RBAC may be desirable.
- The concepts of roles and groups intersect. There are good reasons to have both. Common grounds and differences should be documented well.
- The actual privileges of a user are composed of his/her role, individuality, and locality. A good RBAC system should support all three concepts.
- Roles must be able to cover all security-related definitions. This typically includes user and privilege attributes.
- Standards for handling intersecting and contradicting roles must be set. The progress of migration from one role to the other must be well defined.
- Privileges given to users directly will remain; roles can only minimize their number.
- Hierarchical relationships between roles is normally sufficient. Cardinality seems to be a rare requirement. The ability to handle exceptions is definitely necessary.
- The separation of duties is encouraged by RBAC and should be enforced by a RBAC standard.
- A description of the role engineering process is desirable in order to increase the acceptance of the standard. However, such a standard is difficult to write. We would prefer

a standard that allows to create roles starting from existing definitions (bottom-up engineering).

References

[1] John F. Barkley, “Implementing Role-Based Access Control using Object Technology”, *First ACM Workshop on Role-Based Access Control,* Gaithersburg, MD 1995

[2] John F. Barkley, Anthony V. Cincotta, David F. Ferraiolo, D. Richard Kuhn, “Role-Based Access Control in Large Networked Applications”, National Institute of Standards and Technology, 1996

[3] David F. Ferraiolo, Janet A. Cugini, and D. Richard Kuhn, "Role-Based Access Control (RBAC): Features and Motivations", *11th Annual Computer Security Applications Proceedings*, 1995

[4] David F. Ferraiolo, and D. Richard Kuhn, "Role-Based Access Control", *15th National Computer Security Conference*, Vol II, pp 554-563 1992

[5] SAM Customer Manuals, Release 2.1, Schumann Unternehmensberatung AG, 1995

[6] Trusted Computer Security Evaluation Criteria, DoD 5200.28-STD, Department of Defense, 1985