Role Based Access Control on MLS Systems without Kernel Changes

D. Richard Kuhn
National Institute of Standards and Technology
Gaithersburg, Maryland 20899

Abstract

Role based access control (RBAC) is attracting increasing attention as a security mechanism for both commercial and many military systems. This paper shows how RBAC can be implemented using the mechanisms available on traditional multi-level security systems that implement information flow policies. The construction from MLS to RBAC systems is significant because it shows that the enormous investment in MLS systems can be leveraged to produce RBAC systems. The method requires no changes to the existing MLS system kernel and allows implementation of hierarchical RBAC entirely through site configuration options. A single trusted process is used to map privileges of RBAC roles to MLS labels. Access is then mediated by the MLS kernel. Where C is the number of categories and d the depth of the role hierarchy, the number of roles that can be controlled is approximately $\binom{C/d}{C/2d}^d$.

1 Introduction

Role based access control (RBAC) is an alternative to traditional discretionary (DAC) and mandatory access control (MAC) policies that is attracting increasing attention [1], particularly for commercial applications. The principle motivation behind RBAC is the desire specify and enforce enterprise-specific security policies in a way that maps naturally to an organization's structure. Traditionally, managing security has required mapping an organization's security policy to a relatively low-level set of controls, typically access control lists.

With RBAC, security is managed at a level that corresponds closely to the organization's structure. Each user is assigned one or more *roles*, and each *role* is assigned one or more *privileges* that are permitted to users in that role.

For example, roles in a bank may include the role of teller or accountant. Each of these roles has a set of privileges or transactions that they can perform, including some privileges that are available to both roles. Roles can be hierarchical. For example, some roles in a hospital may be health care provider, nurse, and doctor. The doctor role may include all privileges available to the nurse role, which in turn includes all the privileges available to the health care provider role.

Roles have been used in a variety of forms for computer system security for at least 20 years, and several proposals for incorporating roles into existing access control mechanisms have been published [2], [3], [4]. More recently, formal definitions for general-purpose RBAC notions have been proposed [5], [6], [7].

This paper shows how RBAC can be implemented using the controls available on traditional lattice-based multi-level secure systems. This approach presents a number of advantages:

- Many firms have spent hundreds of millions of dollars building, testing, and maintaining MLS systems. By implementing RBAC using a single trusted process, this investment can be leveraged to produce new systems that have great commercial value without requiring a similarly large investment to build entirely new RBAC systems.
- The assurance process for trusted systems is lengthy and expensive. By confining RBAC to a single trusted process that sits above the MLS kernel, the assurance process should be much less expensive than that required for an entirely new system. Since RBAC is implemented through configuration options, a system can provide RBAC while retaining the same high assurance level.
- Operating RBAC and MLS security simultaneously on a system may be much easier to analyze for assurance purposes. By using only combinations of category labels to implement RBAC, information flow can be protected using the conventional sets of security levels and categories.

2 Implementing RBAC on Multi-level Secure Systems

RBAC can be implemented directly on multi-level secure (MLS) systems that support the traditional lattice based controls. This is significant because it means the enormous investment in MLS systems can be applied to implementing RBAC systems. The method described here can handle approximately $\binom{C/d}{C/2d}^d$ RBAC privileges, where C is the number of categories supported on the MLS system. and d the depth of the role hierarchy.

2.1 MLS Access Controls

MLS access controls make use of a set of labels attached to subjects and objects. The labels define a set of security *levels*, such as CONFIDENTIAL, SECRET, TOP SECRET, and a set of *categories*, such as NATO, NOFORN. Conventional MLS systems implement the military security policy defined by the Bell and LaPadula model [8].

We assume a standard set of features and functions for an MLS system, such as those described in [9] or [10]. The MLS system is assumed to maintain the following sets:

$\mathcal{L}$ = an ordered set of security clearance levels l;

$\mathcal{C}$ = a set of category names c.

Each subject s has a set of category names c_s authorized for use by subject s, and each object o has a set of category names c_o associated with the object. Levels and categories define labels for subjects s and objects o, designated $\lambda(s)$ and $\lambda(o)$ respectively. The labels form a lattice where $\lambda(i) \geq \lambda(j)$ iff $l_i \geq l_j$ and $c_i \supseteq c_j$

For read and activate access, the mandatory access control rules require the simple security property: $\lambda(s) \geq \lambda(o)$. For write access, the $\star$-property controls access. The traditional, or liberal $\star$-property requires that $\lambda(o) \geq \lambda(s)$. The strict $\star$-property, designed to prevent integrity problems as a result of "write-up", requires $\lambda(o) = \lambda(s)$. A variation on the $\star$-property, the trusted liberal $\star$-property, introduced by Bell [11], designates separate labels for read and write, λ_r and λ_w respectively. The simple security rule is applied for λ_r and the $\star$-property for λ_w.

2.2 MLS to RBAC Mapping

A role can be thought of as a set of permissions on privileges. RBAC can then be implemented on an MLS system by establishing a relationship between privilege sets within the RBAC system and category sets within the MLS system.

To implement RBAC, a trusted interface function is developed to ensure that the assignment of levels and categories to users is controlled according to the RBAC rules. No modifications to the MLS system are necessary. Roles and their associated privilege sets must be mapped by the interface function to sets of categories. The trusted interface operates according to the rules given in Section 2.1. Each time a user establishes a session, the interface presents the user's role options, then checks to ensure that the user is authorized for the requested role. The trusted interface then sets the subject's categories according to a mapping function that determines a unique combination of categories for the role requested. (See Figure 1.)

A problem arises in the choice of the mapping function. One possibility is the one-to-one assignment of MLS categories to RBAC privileges. This approach is used in the Data General DG/UX B2 Secure System [12]. For small numbers of privileges, this is an efficient solution. DG/UX supports up to 128 separate roles. Users can simply be assigned a set of categories that correspond to the privileges of their roles, then access is handled by the MLS system.

Unfortunately, most MLS systems support a relatively small number of categories and levels, typically 64 to 128 of each. Obviously, if the MLS system were testing that $l_s \geq l_o \wedge c_o = c_s$, rather than $l_s \geq l_o \wedge c_o \subseteq c_s$, then we could simply use subsets of categories to map to privileges, giving a total of 2^c mappings. But since we want to be able to control access to RBAC privileges simultaneously with MLS access control without changing the MLS system, we need a method that can uniquely represent a large number of privileges using MLS categories and levels.

One alternative is to establish a mapping between RBAC privileges and pairs of MLS categories. This approach would support a total of $(n^2 - n)/2$ privilege mappings. If 64 categories are available on the MLS system, then $2,016$ privileges could be mapped to MLS categories. This is a more reasonable number, but large organizations may require many more individual privileges to be controlled. Also, in some applications only a very small number of categories may be available. If only 10 categories were available, then only 45 privileges could be controlled in this manner.

A more generalized approach is to use combinations of categories. For c categories, the largest number of privileges that can be distinguished is

$$\binom{c}{c/2}$$

With 64 categories, this would be 1.83×10^{17}.

2.3 Construction of Category Sets

This section describes a method of implementing RBAC by mapping from roles to categories at system initialization time. Only category sets are used; security levels are not needed to control access to RBAC-protected objects. This makes it possible to use RBAC simultaneously with the information-flow policies supported on MLS systems.

2.3.1 Roles and Privilege Sets

Let R be a tree of roles and associated privileges, where the root R_0 represents one or more privileges that are available to all roles in the system. Child nodes represent more specialized privilege sets. A child node R_j can access all privileges associated with role R_j and any associated with roles R_i, where R_i are any ancestor nodes of R_j. The privilege sets are assumed to be disjoint. If roles exist with overlapping privilege sets, then new roles can be created with the common privileges and existing roles can inherit from them. For example, if R_i and R_j have privilege sets $P(R_i)$ and $P(R_j)$ that overlap, then

1. create a new role R_k with privilege set $P(R_i) \cap P(R_j)$
2. remove privileges in $P(R_i) \cap P(R_j)$ from R_i and R_j
3. modify the role hierarchy so that role R_i and R_j inherit from R_k, and R_k inherits from the role that R_i and R_j previously inherited from.

Let

C = total number of categories on the MLS system to be used to implement RBAC.

d = maximum depth of child nodes from the root, where the root is level 0. This is equivalent to the maximum level of the leaf nodes.

The categories from C will be assigned to roles and privilege sets. If the tree is relatively balanced, then C/d categories are available at each level for representing privilege sets. To distinguish between privilege sets, combinations of categories are used. At each level in the tree, where n is the number of categories available for representing roles at that level, the number of privilege sets that can be distinguished is $\binom{n}{n/2}$. Using C/d categories at each of d levels, the total number of privilege sets in the tree is therefore (depending on how well balanced the tree is) approximately $\binom{C/d}{C/2d}^d$.

2.3.2 Assignment of Categories to Privilege Sets

Privilege sets are associated with categories as follows:

1. A role at the root of the tree, with privileges available to all users, is associated with a randomly selected category. This category is removed from the set of categories available to designate roles.
2. Roles at level l of the tree, where n_l indicates the number of nodes at level l, are associated with unique sets of categories drawn from the set of remaining categories. The number of categories needed for level l is the smallest number c such that $n_l \leq \binom{c}{c/2}$. Choose c categories from the remaining set of categories. Remove these c categories from the set of categories available to designate roles.
3. From the set of c categories chosen in step 2, assign a unique set of categories to each privilege set at level l. Step 2 ensures that there are enough categories to make all the sets different.

One way of implementing this step is to generate a list L_1 of numbers from 1 to $2^c - 1$, then extract from this list a second list L_2 containing all numbers whose binary representation contains $c/2$ bits. Each bit is associated with a category. Assign to each privilege set at level l a different number from L_2. Then label each privilege in a privilege set with category i if and only if bit i in the binary representation is a 1. For example, the mapping from bits to categories in Table 1 shows how the procedure works for $c = 3$ categories. Extracting all sets of 2 categories from the list gives $\{c_2, c_1\}$, $\{c_3, c_1\}$, $\{c_3, c_2\}$. (These are highlighted with brackets in Table 1. It would also be possible to have three distinct sets of one category each; two are used simply to demonstrate the procedure.) Because all of the numbers associated with privilege sets have $c/2$ bits, each privilege set will be labeled with a different set of categories.

4. Repeat steps 2 and 3 until all privilege sets have been assigned a set of categories.

L_1 $1..2^3 - 1$	L_2:binary $x_3x_2x_1$	Categories $C_p = \bigcup_{j\|x_j=1} c_j$
1	001	c_1
2	010	c_2
3	{011}	$\{ c_2, c_1 \}$
4	100	c_3
5	{101}	$\{ c_3, c_1 \}$
6	{110}	$\{ c_3, c_2 \}$
7	111	c_3, c_2, c_1

Table 1.

2.3.3 Assignment of Categories to Roles

Each role must be able to access all privileges associated with its privilege set and all privilege sets associated with roles that it inherits, i.e., roles that are represented by ancestor nodes in the role hierarchy. Categories are assigned to roles as follows:

1. Assign to role R_i the set of categories assigned to its privilege set.
2. For each ancestor role R_j from which role R_i inherits its privileges, add to the labels for role R_i the categories associated with the privilege set for R_j.

2.4 Analysis of MLS to RBAC Mapping

MLS systems typically provide 64 to 128 categories for labeling privileges. The construction described in the previous section will provide a capability for approximately $\binom{C/d}{C/2d}^d$ roles. Tables 2 and 3 show the number of roles that can be controlled for various combinations of depth and breadth (branching factor) of role hierarchies.

Depth	Max. Branching Factor	Max. Roles
5	924	6×10^{14}
10	20	1×10^{13}
15	6	4.7×10^{10}
20	3	3.4×10^{9}

Table 2. Number of Roles Supported with 64 Categories

Depth	Max. Branching Factor	Max. Roles
5	5,200,300	3.8×10^{33}
10	924	4.5×10^{29}
15	70	4.7×10^{27}
20	20	1.0×10^{26}
25	10	1.0×10^{25}
30	6	2.2×10^{23}
40	3	1.2×10^{19}

Table 3. Number of Roles Supported with 128 Categories

2.5 Example of MLS to RBAC Mapping

Figure 2 shows an example of category labeling for a hierarchical privilege set defining 36 roles. The tree has a depth of 2 and a maximum branching factor of 6. A total of 9 categories are needed. The privilege sets assigned to a role are those labeling the role's node in the tree, plus the labels of any ancestor nodes. For example, role R_{33} has categories a, b, d, g, and i.

Consider roles R_0, R_1, and R_{20}. Privileges authorized for role R_0 are assigned category a. Privileges authorized for role R_1 are assigned categories a, b and c (a from role R_0 and b and c from role R_1). Privileges authorized for role R_{20} are assigned categories a, b, c, g, and h. (a from role R_0; b and c from role R_1 and g and h from role R_{20}). A user who establishes a session at role R_1 will be assigned categories a, b and c. Note that this user can access the privileges assigned to role R_0 because the user has category a. A user who establishes a session at role R_{20} will be assigned categories a, b, c, g, and h. This user can access all inherited privileges, but not any other privilege sets because all others have at least one category not assigned to role R_{20}.

Figure 3 shows a portion of Figure 2, with privilege sets associated with various roles. Each of the privileges, P_1 and P_2 associated with role R_0 is labeled with category a. Therefore any user authorized for role R_0, or any role that inherits privileges from R_0 (e.g. R_1, R_7, etc.), can access privileges P_1 or P_2. Note that a user authorized only for R_0 cannot access privileges such as P_5, P_6, P_7, because these are labeled with categories a, b, and c, but R_0 has only category a. A user authorized for role R_1, or any role that inherits from R_1 can access P_5, P_6, P_7, because R_1 has categories a, b, and c.

3 Discussion and Future Directions

Several authors have discussed the relationship between MLS lattice based systems and RBAC. Nyanchama and Osborn [13] and Sandhu [14] presented methods for simulating lattice based MLS systems in RBAC. Osborn [15] investigated the interaction between RBAC and mandatory access control rules, showing that significant constraints exist on the ability to assign roles to subjects without violating MAC rules.

One advantage of the approach described in this paper is that it allows RBAC to be operated simultaneously with MAC. Because the roles-to-categories construction allows the implementation of a large role hierarchy with a relatively small number of categories, the remaining categories can be used to implement the traditional multilevel security model. If the RBAC system does not embed data accesses in processes or roles, then one set of categories can be used to implement RBAC, with the remaining categories available for implementing MAC. If the processes or transactions available to users are labeled with a level of *system-low* and with categories according to the construction of Section 2, then system users can activate any process available to their role, and apply the process to any data for which they are cleared by virtue of MAC clearance level and categories. This architecture may be particularly advantageous in a military system that must support both roles and MAC security. For example, a system for satellite photo analysts could provide a role structure to control access to photos that are classified into different clearance levels and categories.

One possible limitation of the construction in Section 2 is that the role to category mapping must be regenerated if changes are made in the role structure. In practice, however, role structures change relatively slowly, and the mapping can be regenerated automatically without impacting users. Another potential problem is that the hierarchy created by the algorithm must be a tree, rather than a lattice hierarchy. This should not be a serious limitation because, to our knowledge, existing role based systems use tree hierarchies. Note that the data objects controlled by MAC rules can still be organized into a lattice. The MAC system will use both levels and categories, while the RBAC system uses only a set of categories with all processes labeled at system-low.

An MLS system designed using the "traditional" $\star$-property would encounter constraints on assigning roles to subjects [15]. In particular, a role R is assignable to an untrusted subject only if all of the following hold:

- *w-level* of $R \geq$ *r-level* of R
- $\lambda(s) \geq$ *r-level* of R
- $\lambda(s) \leq$ *w-level* of R

where *r-level* is the maximum security level of any object readable by processes in role R, and *w-level* is the minimum security level of any object writable by processes in role R. Since a role might require read and write access to objects at a broad range of security levels, this constraint could theoretically present a problem in implementing RBAC with MAC. However, practical applications provide a way around this limitation. In practice, the traditional $\star$-property is relaxed to allow write access if the data written does not depend on the data read [10], reducing constraints on role assignment depending on the degree to which there is independence between read and write data in "typical" applications. Another approach worth investigating is the use of Bell's "liberal $\star$-property" [11]. It would be interesting to investigate existing systems that have a need for both roles and MAC to evaluate the practical implementation of RBAC on real-world MLS system applications.

4 Conclusions

Because of both cost and trust considerations, it is desirable to build RBAC systems on a proven MLS operating system. From a cost standpoint, it will normally be much easier to build RBAC as a single trusted process, then rely on the MLS to control access to objects, than to modify the kernel of a secure system or build a new one from the ground up. Trust and assurance may be even more important considerations. The assurance process for a secure computing system is lengthy and expensive. MLS systems on the market today have had extensive evaluations and years of use in the field, largely by military organizations. The addition of RBAC to these systems can make them much more useful for commercial applications. The method described in this paper will make it possible to leverage the large investment in these systems to produce RBAC systems that are in demand for commercial use.

For further information on this or other NIST RBAC research, contact the NIST Office of Technology Partnerships.

5 Acknowledgments

I am grateful to Sylvia Osborn for many helpful comments and suggestions.

References

[1] R. Sandhu, E.J. Coyne, and C.E. Youman, editors. *Proceedings of the First ACM Workshop on Role Based Access Control.* ACM, 1996.

[2] R.W. Baldwin. Naming and grouping privileges to simplify security management in large databases. In *Proceedings, IEEE Computer Society Symposium on Research in Security and Privacy.* IEEE Computer Society, 1990.

[3] D.J. Thomsen. Role-based application design and enforcement. In *Database Security IV: Status and Prospects.* North-Holland, 1991.

[4] D.F. Sterne. A TCB subset for integrity and role-based access control. In *15th National Computer Security Conference.* NIST/NSA, 1992.

[5] D. Ferraiolo and D.R. Kuhn. Role based access control. In *15th National Computer Security Conference.* NIST/NSA, 1992.

[6] D. Ferraiolo, J. Cugini, and D.R. Kuhn. Role based access control: Features and motivations. In *Annual Computer Security Applications Conference.* IEEE Computer Society Press, 1995.

[7] R. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role based access control models. *IEEE Computer*, 29(2), February 1996.

[8] D. Bell and L. LaPadula. Secure computer systems: Mathematical foundations and model. Technical Report M74-244, Mitre Corp., 1973.

[9] M. Gasser. *Building a Secure Computer System.* Van Nostrand Reinhold, 1988.

[10] C. Pfleeger. *Security in Computing.* Prentice Hall, 1989.

[11] D.E. Bell. Secure computer systems. In *Proceedings, 3rd annual computer security application conference*, 1987.

[12] W.J. Meyers. RBAC emulation on trusted DG/UX. In *Proceedings of the Second ACM Workshop on Role Based Access Control.* ACM, 1997.

[13] M. Nyanchama and S. Osborn. Modeling mandatory access control in role-based security systems. In *Proceedings of the IFIP WG 11.3 ninth annual working conference on database security.* Chapman and Hall, 1995.

[14] R. Sandhu. Role hierarchies and constraints for lattice-based access controls. In *Computer Security - ESORICS 96, pp. 65-79.* Springer Verlag, 1996.

[15] S. Osborn. Mandatory access control and role-based access control revisited. In *Proceedings of the Second ACM Workshop on Role Based Access Control.* ACM, 1997.

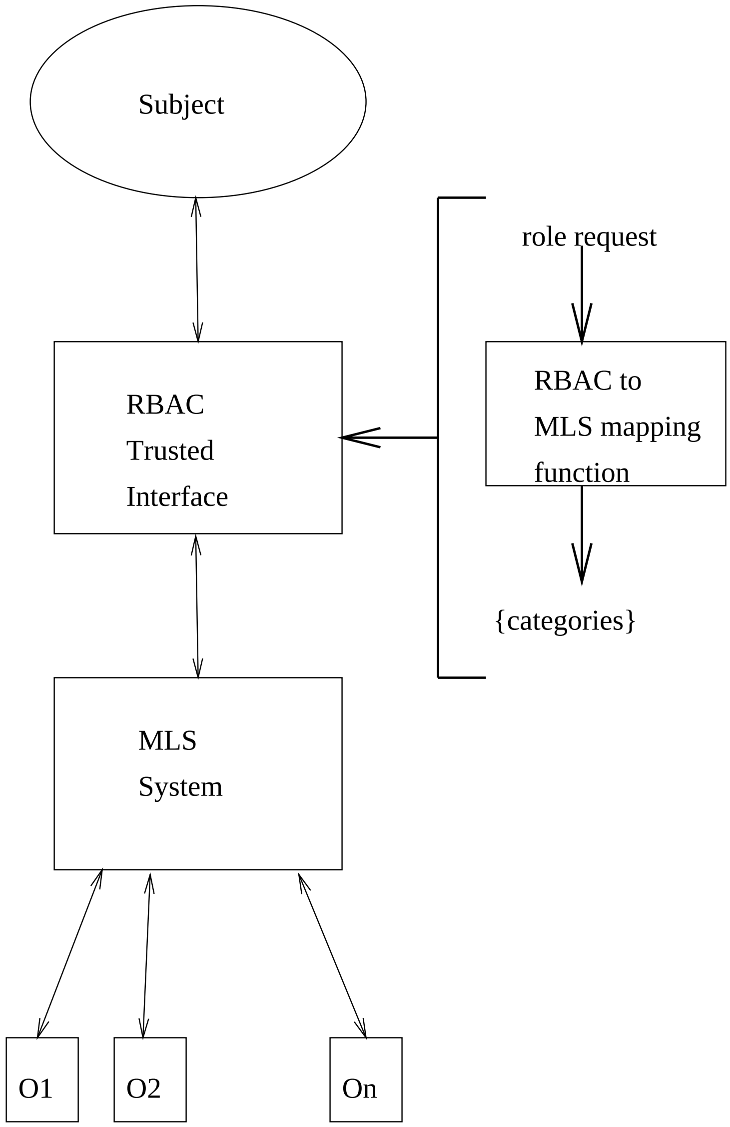

Figure 1: RBAC/MLS Interface

Figure 2:

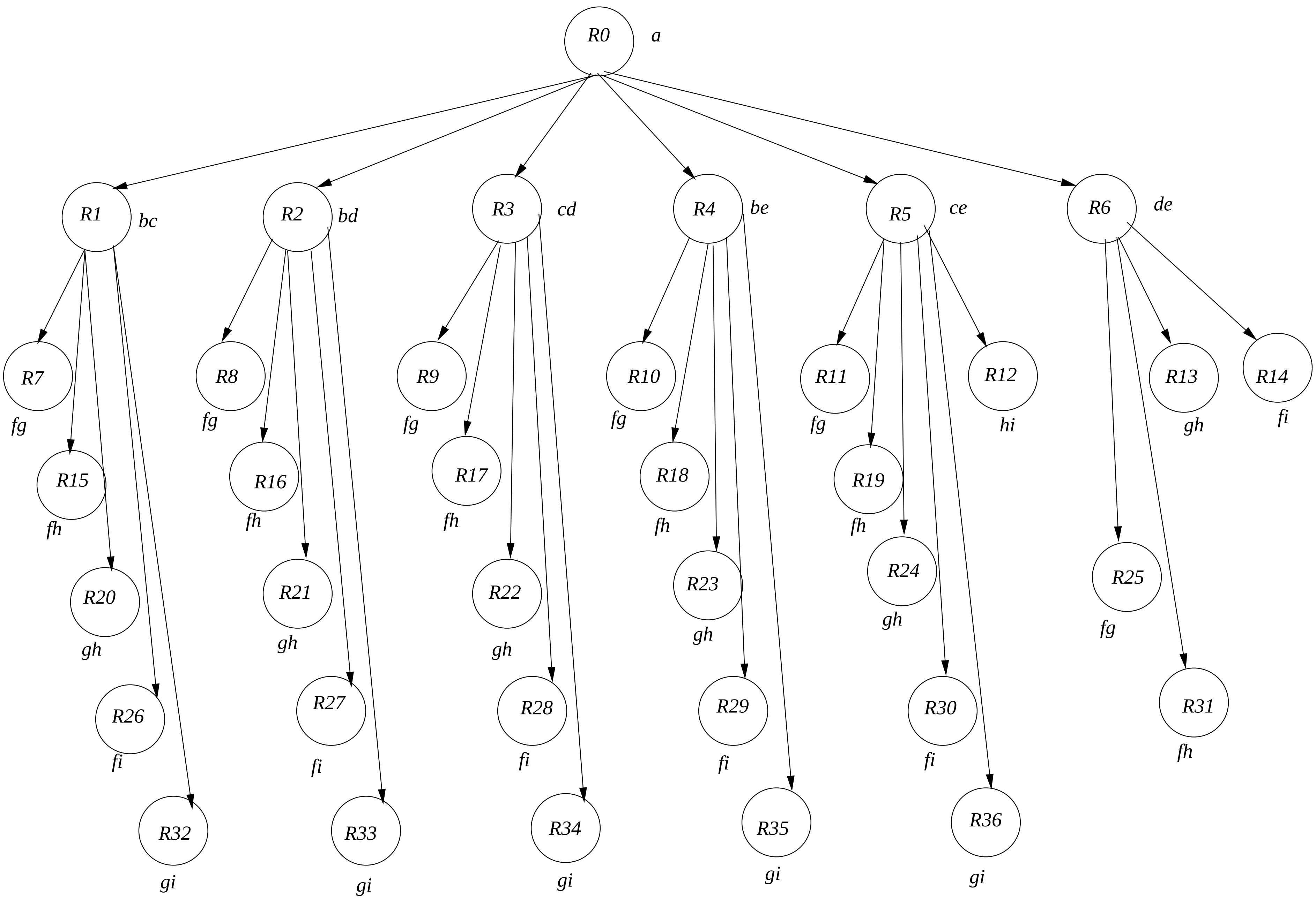

Figure 2. Example of Hierarchical Privilege Mapping

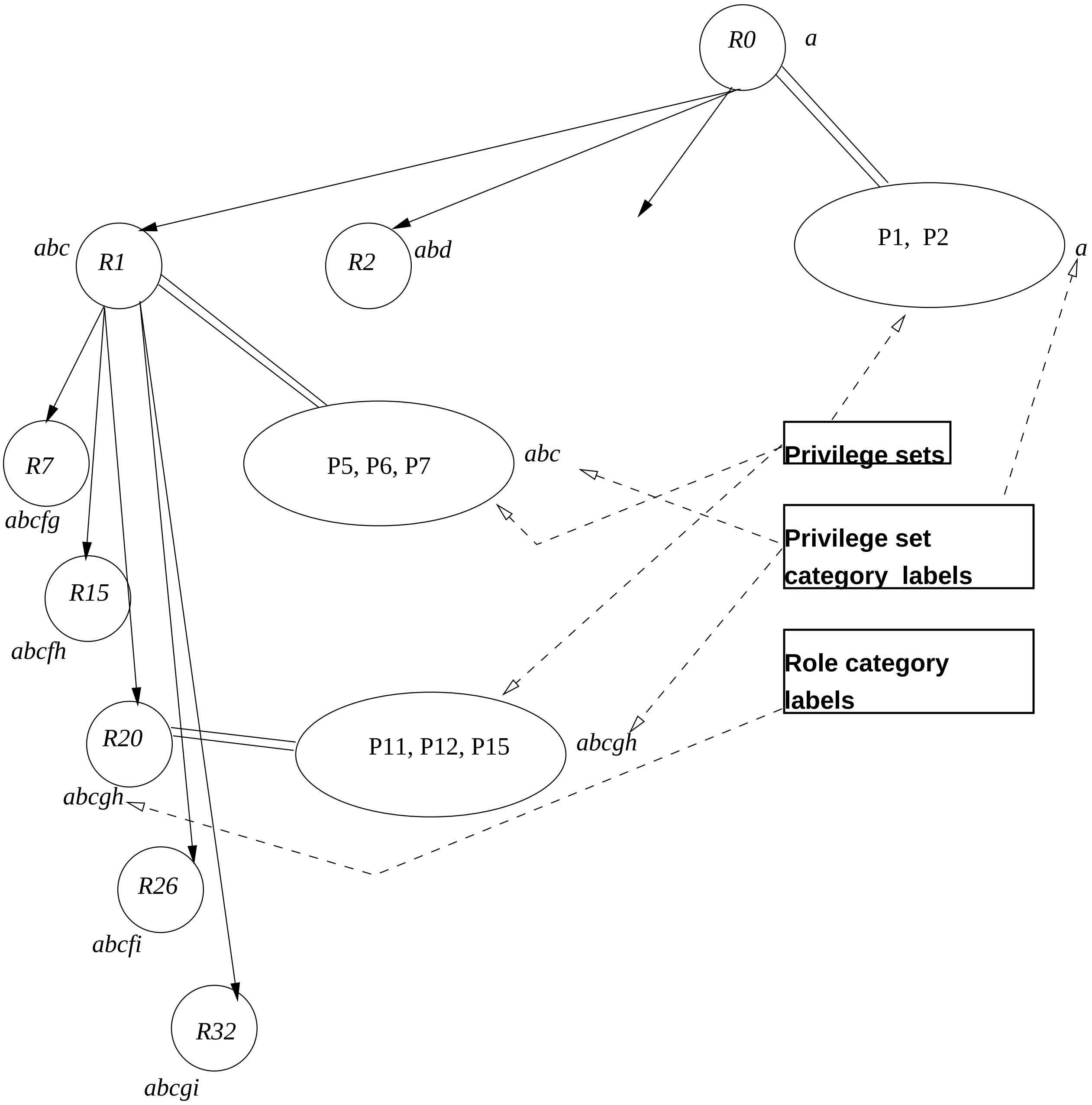

Figure 3: Roles and Privilege Sets with Category Labels