Induced Role Hierarchies with Attribute-Based RBAC

Mohammad A. Al-Kahtani
George Mason University
malkahta@gmu.edu

Ravi Sandhu
NSD Security, Inc. &
George Mason University
sandhu@gmu.edu

ABSTRACT
The Role-Based Access Control (RBAC) model is traditionally used to manually assign users to appropriate roles. When the service-providing enterprise has a massive customer base, assigning users to roles ought to be automated. RB-RBAC (Rule-Based RBAC) provides the mechanism to dynamically assign users to roles based on a finite set of authorization rules defined by the enterprise's security policy. These rules may have seniority relation among them, which induces a roles hierarchy. The main contribution of this paper is to explore the possible discrepancies between the Induced Roles Hierarchy and any existing roles hierarchy. The functional impact of existing discrepancies and ways of reconciling them are discussed.

Categories and Subject Descriptors
D.4.6 [**Operating Systems**]: Security and Protection – *Access Control.*

General Terms
Security

Keywords
Access Control, Roles, RBAC, Attributes, Authorization Rules, Roles Hierarchies.

1. INTRODUCTION
Role-Based Access Control (RBAC) has emerged as a proven and superior alternative to traditional discretionary and mandatory access controls [6, 7]. RBAC greatly simplifies the management of permissions by associating them with roles to which users are assigned, thereby acquiring the roles' permissions. The users are manually assigned to roles based on criteria specified by the enterprise. As the Internet becomes more accessible, an increasing number of service-providing enterprises make their services available to their users via the Internet. RBAC can be used to manage users' access to the enterprise services and resources.

SACMAT'03, June 2-3, 2003, Como, Italy

In many environments, the number of users can reach the hundreds of thousands or millions. Typical examples are banks, utility companies, and popular Web sites, to name a few. This renders manual user-to-role assignment a formidable task. In [1], a new model is introduced to automatically assign users to roles based on a finite set of assignment rules derived from the security policy of the enterprise. These rules take into consideration the attributes of users and any constraints set forth by the enterprise. The rules may have some sort of seniority relations among them. These relations can be used to induce a hierarchy among the roles even if these roles were treated as flat roles. Often times, an enterprise has a role hierarchy that is based on the relations that can be derived from the permissions assigned to roles. In such cases the induced roles hierarchy coexists with the given roles hierarchy.

In this paper, we discuss the nature of this induced role hierarchy (IRH). We also address different types of discrepancies that might occur between the induced role hierarchy and the hierarchy used by the enterprise. The functional impact of these discrepancies and ways of reconciling them are discussed.

The paper is organized as follows. In section 2 we summarize related research. In section 3, RB-RBAC is revisited. In section 4, we introduce the IRH and discuss discrepancies that might exist between IRH and any given roles hierarchy (GRH) derived from the permissions relation among roles. In section 5 we touch on issues that we have not explored in this paper, though they are closely related to the topic discussed. Section 6 concludes the paper.

2. RELATED WORK
The main concept of RBAC is the role, which can be viewed as a semantic construct around which access control policy is formulated. Permissions are associated with roles to which users are assigned based on factors such as their responsibilities and qualifications. Users can be easily reassigned roles. The set of permissions assigned to a role can be modified as deemed needed by the enterprise. The roles can by organized in role hierarchies to reflect the organization's lines of responsibility and authority [7].

Initially, RBAC was motivated by closed-enterprise systems in mind. In this type of environment, the security administrator(s) assign roles manually to users. Park and Sandhu presented RBAC as a sound candidate to control users' access to resources and services in large-scale Web environments [5]. They identified architectures that can be used to implement RBAC on the Web. They also showed how existing technologies can be utilized to

support these architectures. However, the architectures proposed were only in the context of enterprise-wide systems in which systems administrators assign users to roles on the basis of users' responsibilities in the enterprise.

Based on certificates issued by third parties, Herzberg et al. presented a Trust Establishment (TE) system that defines the mapping of strangers to predefined business roles [3]. The system maps users to roles using well-defined logical rules. Each role has one or more rules defining how a client can be assigned that role. The TE system gathers certificates related to a specific client and makes a decision regarding the client's eligibility for a specific role. The system proposed in [3] does not address relations that might exist among different rules. TE system relies on bottom-up buildup of the public key infrastructure (PKI), which imports all the issues related to PKI.

Zhong, et al. proposed a schema to use RBAC on the Web and a procedure for user-role assignment [9]. Based on legitimacy of information gathered, assignment policies, and the trustworthiness threshold specified by system administrators, the schema assigns a client to a role. Users' trustworthiness represents the degree to which the enterprise believes that a user will not do harm to its systems. It is accumulated gradually over time and drops if harmful actions or potential harmful actions are discovered. There is a major drawback to this approach. A malicious user may logon to the system for an extended period of time without performing any suspicious acts. As time goes on, he acquires a high clearance, which may enable him to inflict damage on the system. Also, the scheme depends on many security parameters, which must be given initial values. This approach leaves determining these values to system administrator(s), but does not provide any guidelines on how to determine them.

Lightweight Directory Access Protocol (LDAP) targets management applications and browser applications that provide read/write interactive access to directories supporting the X.500 models [4]. Roles can be stored in directories and retrieved when needed. LDAP has been augmented to support dynamic groups. A dynamic group is an object with a membership list of distinguished names that is dynamically generated using LDAP search criteria. The dynamic membership list may then be interrogated by LDAP search and compare operations, and be used to identify a group's access control subjects [2]. This feature could be used to automatically assign users to roles in large enterprises. However, implementing LDAP solely for the sake of dynamically assigning users to roles is an unwieldy solution. Also, LDAP returns a simple list of attributes (which represent roles in our case) with no logical structure attached to them. If, for example, a client can assume one of two mutually exclusive roles, LDAP does not provide a simple mechanism to express this.

Yao et al. [8] present an RBAC model that does not recognize role hierarchies explicitly. Instead, they propose a role activation dependency that is dynamic. A set of parameterized rules governs the activation of every role. Their model is rich in terms of expressing the rules, and associated conditions. However, we think that eliminating role hierarchies is a debatable issue to say the least. Role hierarchies have value not only from the user-assignment perspective of roles but also from the permission-assignment perspective. Also, by making the hierarchies implicit via side effects of role activation rules, the model does not explicitly capture various relations that might exist among roles.

Al-Kahtani and Sandhu [1] propose a new RBAC-based model to automate the process of assigning users to roles. Based on the security policy of the enterprise, the model can be used to define a set of authorization rules that take as input the attributes associated with a specific user and produce the role(s) that user is entitled to have. The model recognizes possible seniority relations among authorization rules, however, it stops short from exploring the nature of this relation and its outcomes. This model is further discussed in the next section.

3. THE RB-RBAC MODEL

3.1. The Model Description

In [1], Al-Kahtani and Sandhu modify RBAC such that it becomes rule-based, thus, they refer to it as Rule-Based RBAC or RB-RBAC. In this model, an enterprise defines the set of rules that are triggered to automatically assign users to roles. These rules take into account:

- The attributes of the client that are expressed using attributes' expressions as defined by the language provided by the model.
- Any constraints on using roles.

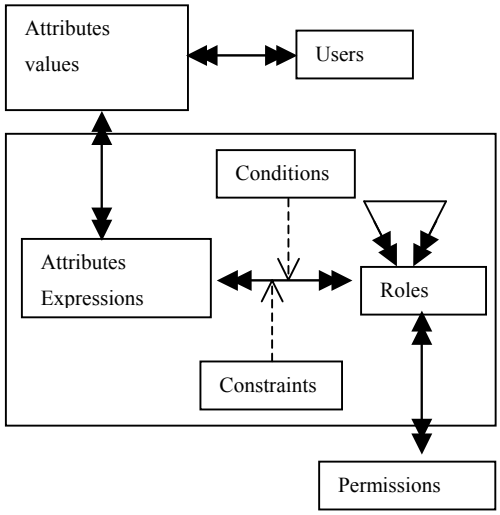

Figure 1: RB-RBAC model

Figure 1 shows that users have many-to-many explicit relation with attribute values. Further, they have many-to-many implicit relation with attribute expressions. One user could have one or more attribute expressions depending on the information he provides. Conversely, two or more users may provide identical attribute expressions. A specific attribute expression corresponds to one or more roles. An example of a rule that yields multiple roles is when a client is entitled to several mutually exclusive roles. The figure also shows that a role may be hierarchically related to one or more roles (in the usual partial order of roles).

The figure also shows that a role may correspond to one or more attribute expressions.

3.2. RB-RBAC Usage

In RBAC, individual(s) responsible for user-role assignment explicitly assign users to roles. Nonetheless, RB-RBAC could be used in two different ways to assign users to roles:

- Implicit user-role assignment: In this mode, no human intervention is allowed and RB-RBAC automatically triggers authorization rules to assign users to roles.
- A hybrid of implicit and explicit user-role assignment: Users are assigned to roles either implicitly by invoking the appropriate authorization rules, or explicitly by security administrator(s).

The discussion in this paper is confined to implicit user-role assignment.

3.3. Basic Assumptions

RBAC96 allows specifying constraints to enforce specific business principles [7]. In this paper, for the most part, we assume that no constraints exist. We also use an abridged version of RB-RBAC model. This version recognizes one operator in the right hand side of an authorization rule, namely the "AND" operator. We did so for the sake of simplifying the discussion.

3.4. Seniority Among Authorization Rules

3.4.1. Definition

In some cases, it might be desirable to compare two rules in terms of their attribute expressions to determine what kind of relation exists between the two, if any. In [1], the concept of seniority levels was introduced to capture any relations that might exist among different authorization rules based on their corresponding attribute expressions. Seniority levels are first assigned to the basic building blocks of attributes expressions, namely the attribute pairs. When giving seniority levels to attributes of numeric values, the following method is used:

- For comparative operators {≥, >}, seniority follows the normal order.
- Seniority levels go in reverse order with comparative operators {<, ≤}.

In case of equality operators {=, ≠} and sets, seniority levels –if they exist- must be manually specified.

Clearly, satisfying an attribute pair that has high seniority level implies satisfying all the ones that have lower seniority levels. (See table 1)

Table 1: Seniority among attribute pairs

	Attribute Pair	Remarks
1	Age ≥ 18	If 1 is satisfied, then 2 and 3 are also satisfied
2	Age ≥ 13	If 2 is satisfied, then 3 is also satisfied
3	Age ≥ 6	Only 3 is satisfied

To capture the seniority relations that might exist among authorization rules, we define the dominance binary relation Ð on Attribute_Expressions such that:

$$Ð = \{(AE_i, AE_j) \mid AE_i \rightarrow AE_j \text{ is true}\}$$

Where both AE_i and AE_j belong to Attribute_Expressions and "$\rightarrow$" represents logical implication. We write $(AE_i, AE_j) \in Ð$ which means that AE_i dominates AE_j.

Another way of stating the above relation between AE_i and AE_j is to say that $Rule_i$ is senior to $Rule_j$ (denoted by ≥):

$$Rule_i \geq Rule_j \Leftrightarrow (AE_i, AE_j) \in Ð.$$

If $Rule_i \geq Rule_j$, then this implies that users who satisfy the attribute expression of rule $Rule_i$ also satisfy $Rule_j$ and, hence, are entitled to the roles produced by $Rule_j$. In the context of discussing seniority levels, we give ourselves the freedom to use rules and attribute expressions interchangeably.

3.4.2. The Characteristics of the Dominance Relation Ð

In this section, the nature of relation Ð is closely examined. Assuming we have the authorization rules shown in Table (2), the rightmost column in the table shows the relations among these rules. Figure (2) depicts the relations in table (2).

Table 2: Relations among authorization rules

Attribute Expression	Roles	Relations
AE_1 = Salary> 1000 **Λ** age > 50	r_1	$AE_1 \rightarrow AE_2$, AE_3, and AE_4
AE_2 = Salary> 1000 **Λ** age > 40	r_2	$AE_2 \rightarrow AE_4$ $AE_2 \equiv AE_3$
AE_3 = ┐(Salary ≤ 1000 V age ≤ 40)	r_3	$AE_3 \rightarrow AE_4$ $AE_3 \equiv AE_2$
AE_4 = Salary> 400	r_4	
AE_5 = Age > 60	r_5	Not related to any attribute expression

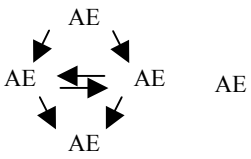

Figure 2: A directed graph representing relation in table 2

One can argue that attribute expressions 2 and 3 are logically equivalent and should be consolidated in one equivalent class. In other words, one of these attributes expressions, say AE_3 should be deleted as is the case in figure (3). The impact of this will be discussed later. As pointed to above, relation Ð is in fact a logical implication.

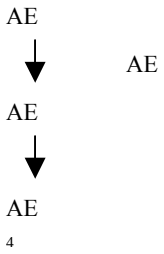

Figure 3: Collapsing equivalent expressions of figure 2

4. INDUCED ROLES HIERARCHY AND GIVEN ROLES HIERARCHY

4.1 Induced Role Hierarchy

It could be argued that the most crucial step in utilizing RB-RBAC model is specifying the authorization rules. This means determining what logical combination of attribute pairs a user is required to have in order for that user to be assigned to a role(s). The starting point for this process could be a security policy document the enterprise strives to implement. This document is expected to list a set of flat roles the permissions of which may not be disclosed for a two-fold reason:

- Knowing the exact permissions of each role is not needed to specifying authorization rules.
- The enterprise may want to hide the details of the permissions and their level of granularity.

As a by-product of specifying the authorization rules, an induced hierarchy among the roles is generated if the authorization rules hold seniority relations among them. In this hierarchy, the role(s) produced by a dominant rule will be senior to the ones produced by a subordinate rule. In order to assemble IRH, we define dominance binary relation **R** on Roles such that:

$R = \{(r_i,r_j) \mid (\exists\ Rule_i)\ (\exists Rule_j)\ [AE_i \rightarrow r_i\ \mathbf{\Lambda}\ AE_j \rightarrow r_j\ \Lambda\ (AE_i, AE_j) \in Ð]\}$

such that $(r_i,r_j) \in R$ means that r_i is senior to r_j in IRH.

IRH specifies the relations that exist among roles based on the relations that exist among the authorization rules that produce them. This does not necessarily reflect the lines of responsibilities and authority as viewed by the enterprise. A junior role in IRH inherits all the users assigned to its ancestor(s). Formally:

$(r_i,r_j) \in R \rightarrow r_{i\ users} \subseteq r_{j\ users}$

Figure 4 shows different ways of depicting IRH that corresponds to attributes expressions in figure 3. In figure 4 (a) and (b), we maintain roles r_2 and r_3 as separate entities. However, the authorization rules set that produces figure 4 (a) will have 2 authorizations rules with identical attribute expressions such that one of these rules yields r_2 while the other yields r_3. On the other hand, the authorization rules set that corresponds to figure 4 (b) will have one rule that produces r_2 and r_3 simultaneously. From a functional standpoint, the two figures are identical.

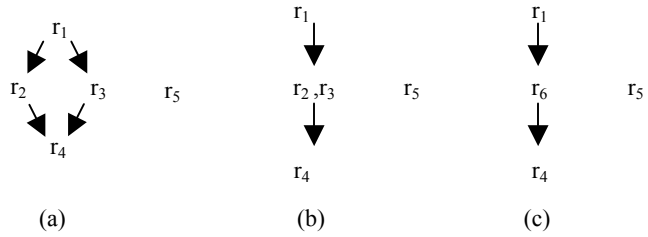

Figure 4: IRH generated by the rules in table 2

Figure 4 (c) shows the case in which r_2 and r_3 are collapsed into one role (r_6). From a functional perspective, this indicates that r_6 is given the permissions of r_2 and r_3. From IRH standpoint, r_6 is given the user assigned to r_2 and r_3. Nonetheless, collapsing roles is not always a prudent course of action. The following are examples for situations in which functionality is adversely affected when roles are combined:

a) When roles have different natures such as a striker and a defender in a video game. Combining these roles yields a role that is meaningless in this context.

b) When the new role has so much permissions, the corresponding workload of which cannot be shouldered by a single user.

c) When combining roles results in a violation of the principle of separation of duties. An example for this is the roles of programmer and tester. A user can be assigned to the programmer role, and thus becomes able to perform certain operations on the source code that he developed. Alternatively, he may choose the tester role where he can test code developed by other programmers but not his. If these two roles where combined, this dynamic separation will be violated.

In many situations, security officers come across a hierarchy that is not derived from the authorization rules. An example of such given roles hierarchy (GRH) is the one that reflects the current business practice of the enterprise. Inheritance of permissions flows upward in the GRH. In an ideal world, these two hierarchies should be mirror images of each other. The implicit assumption is that there is correlation between users-to-role assignment and permission-to-role assignment. However, sometimes discrepancies exist amongst the two hierarchies. The next section describes possible discrepancies, their functional implications, and how to resolve them.

4.2 Possible Discrepancies between IRH and GRH

As discussed earlier, IRH was introduced as a by-product of the relations that exist amongst different authorization rules, which are extracted from the security policy that an enterprise wants to put in place. We assume that the process of interpreting the security policy into authorization rules is flawless. Figure 5 shows two types of inheritance:

- User-role assignment inheritance, which flows from senior roles to their junior roles.

- Permission-role inheritance, which flows in the opposite direction.

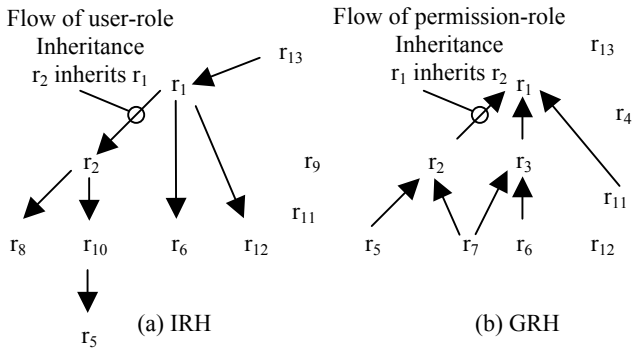

Figure 5: An example of discrepancies between IRH and GRH

A node in the figure denotes a role, while an edge captures the nature of relation between the two nodes at its ends. From an IRH perspective, the possible discrepancies between two hierarchies could be classified into the following categories:

4.2.1 Missing nodes

This category could be divided further depending on the location of the missing node:

4.2.1.1 Leaf node

In Figure 5, node r_7 is missing in IRH, which means that no authorization rule assigns users to that role. But since the permissions of r_7 are inherited by r_2 and r_3, both are captured in IRH, this scenario neither poses a threat to the system's security, nor does it reduce its functionality. That is:

$$(r_i \in GRH \wedge r_i \notin IRH \wedge (\exists r_j : r_j \in IRH \wedge r_j \geq r_i)$$

$$\wedge \neg (\exists r_k : r_i \geq r_k)) \rightarrow \text{no harm}$$

The (≥) relation is used as defined in [7]. To reconcile the two hierarchies, r_7 is deleted from GRH and its permissions is added to its immediate ancestor(s). In real world, r_7 could be the lowest role to which the enterprise does not intend to assign users. Rather, that role is nothing but a building block for constructing senior roles. If none of r_7 ancestors in GRH belongs to IRH, then one of the following is true:

- The security policy, which was used to derive IRH, has overlooked parts of the business practice of the enterprise and, hence, some functionality is missing. In this case, the policy needs to be modified such that an authorization rule will assign users to the missed role.
- The business practice followed by the enterprise has created unnecessary roles to which no users are to be assigned. These roles have to be deleted from the GRH.

4.2.1.2 Non-leaf and non-root node

r_3 is missing in the IRH part of Figure 5. This could result from the enterprise recognizing r_3 as a semantic construct that groups several permissions, but not seeing any need for assigning users to it. From a functionality standpoint, no harm is done so long as at least one of r_3 senior roles is part of IRH. Formally speaking:

$$(r_i \in GRH \wedge r_i \notin IRH \wedge r_j \geq r_i \wedge r_j \in IRH \rightarrow \text{no harm}$$

This discrepancy could be reconciled by removing r_3 from GRH and assigning its permissions to its immediate senior role(s). If none of r_3 ancestors in GRH belongs to IRH, then we are faced with a situation similar to the one discussed in the case (I) above.

4.2.1.3 Stand-alone node

r_4 in GRH represents this case. It entails harm only if the following holds:

$$(\forall r_i \in GRH,\ r_{4\ \text{permission set}} - \cup(r_{i\ \text{permission set}}) \neq \phi$$

If the above formula holds, then some permissions of r_4 can never be used. This indicates a flaw in either the security policy, or the business practice of the enterprise. An example for this case is the security officer who works for a bank but reports to a security company. None of the bank employees is senior to that officer although his permissions are recognized by the bank's role hierarchy.

4.2.1.4 Root node

Assuming that node r_1 in IRH is missing. This results in a loss of functionality since no user can be assigned to r_1 and uses its permissions. In this case, the policy has to be modified.

4.2.2 Additional Nodes

In the following cases, no functionality is ignored by the IRH, but the security policy has added to IRH roles that have no permissions associated with them.

4.2.2.1 Leaf node

In Figure 5, node r_8 is an example of this case. To reconcile the hierarchies, r_8 must be removed and the security policy must be modified such that the authorization rule(s), which produces r_8, must be altered so that it does not yield this problematic role. Alternatively, the current business practice has to be revised to incorporate r_8 into GRH with the appropriate permissions. IRH provides us with a useful insight into the permission set of r_8, that is, it should be a proper subset of the permission set of r_2.

4.2.2.2 Non-Leaf and non-root node

r_{10} exists in the IRH but not in GRH. If r_{10} has a single child, which belongs to GRH, then one can assume that r_{10} permissions set is identical to that of its child, however, the set of users assigned to r_{10} is a subset of its child's users set. From a functional standpoint, r_{10} is redundant to its child, r_5 in this case, because its users will be confined to the permissions associated with r_5. Role r_{10} should be removed from IRH and the authorization rules should be modified so they yield r_5 instead of r_{10}. Alternatively, r_{10} can be added to GRH with permission set such that:

$$r_{5\ \text{permission set}} \subset r_{10\ \text{permission set}} \subset r_{2\ \text{permission set}}$$

However, if r_{10} has more than one child, which are nodes in GRH, then r_{10} can be added to GRH such that:

$$r_{10\ \text{permission set}} = \cup\ r_{i\ \text{permission set}}$$

where $r_i \in GRH \wedge r_{10} \geq r_i$

4.2.2.3 Stand-alone node

This role has no functional purpose and, thus, has to be discarded. An example for this is r_9. The security policy should be modified to reflect this.

4.2.2.4 Root node

Assume there is a role in IRH that is senior to r_1, say r_{root}, then we have 2 possibilities:

- If r_1 is the only child, then $r_{1\ permission\ set} = r_{root\ permission\ set}$ which results in functional redundancy. This can be solved by removing r_{root} from IRH.
- If r_{root} has more than one child that belong to GRH, then:

$$r_{root\ permission\ set} = \cup\ r_{i\ permission\ set}$$

where: r_i is the immediate child of r_{root}.

GRH may be modified to adopt r_{root}.

4.2.3 Missing Edges

The enterprise business practice sees a functional relation between r_1 and r_{11} and captures that in the form of an edge between these roles in GRH. However, the security policy does not recognize that and, therefore, no user-role inheritance exists between them. The users assigned to r_1 are capable of utilizing the permissions attached to r_{11} since they are a subset of r1 permissions even if IRH fails to reveal this hidden relation. This can be eliminated by modifying the policy so that the authorization rule that generates r_1 becomes senior to the one that yields r_{11}.

4.2.4 Additional Edges

IRH has the edge that links r_1 and r_{12}, which GRH does not recognize. From functional stance, this should not be a problem since it is acceptable to assign a user to roles that are not functionally related. Since the users assigned to r_1 are also capable of activating r_{12}, to reconcile the two hierarchies, the permissions set of r_1 need to be modified to include that of r_{12}, which results in introducing an edge between the two roles in GRH. Formally:

$$(r_i,r_j) \in R\ \Lambda\ \neg(r_j \geq r_i)\ \Lambda \neg(r_i \geq r_j) \rightarrow \text{modify r1}_{\ permission\ set}$$

4.2.5 Inconsistency

Normally, user-role assignment inheritance and permission-role inheritance flow in opposite directions. Figure 6 shows a case in which this normal behavior is violated due to a discrepancy between IRH and GRH. Part (a) of the figure suggests that since r_2 is senior to r_3, all users in r_2 will be able to exercise the permissions of r_3 in addition to those of r_2. Accordingly, one can assume that r_3 permissions set is included in r_2. However, part (b) shows that r_2 permissions set is a subset of r_3 permissions set, which indicates that any user assigned to r_3 should be able to use the permissions assigned to r_2. This result contradicts the one derived from part (a). This contradiction manifests itself graphically as two arrows flowing from r_2 and r_3. Either the policy or the role-permission assignment has to be modified.

5. DISCUSSION

RBAC96 allows specifying constraints to enforce specific functional requirements and business principles, the most notable of which is the principle of separation of duties. In this paper, we assume that no constraint is specified. However, the RB-RBAC honors separation of duty that is expressed externally.

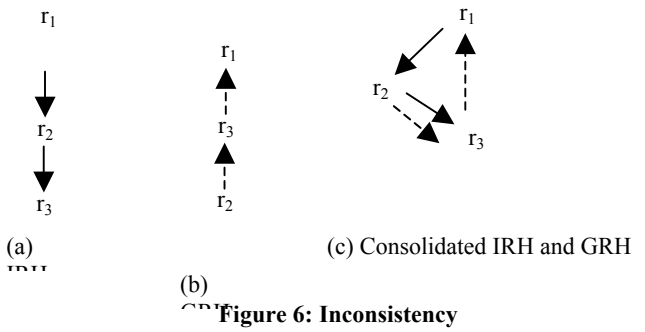

Figure 6: Inconsistency

For example, if r_i and r_j are conflicting roles, then if we have authorization rule:

$$rule_i \rightarrow_{ri}, \text{and } r_j$$

then r_i is considered inconsistent and the policy must be modified so that this inconsistency is removed. One suggested way to discover this inconsistency is to declare a set of conflicting roles and check if any authorization rule has more than one member of that set in its right-hand side. Also, the language provided by RB-RBAC can be used to express constraint. The above example can be expressed using the full language set of RB-RBAC as follows:

$$rule_i \rightarrow_{ri} \text{XOR } r_j$$

6. CONCLUSION

We have discussed the nature of the authorization rules in RB-RBAC, specifically, the seniority relation that might occur amongst them. We showed that this relation can induced a role hierarchy, which can be assembled based on the direction of user-role assignment inheritance. We gave an example for a situation where this induced role hierarchy coexists with a given role hierarchy. The discrepancies that might arise between the induced role hierarchy and the given role hierarchy were explored. Different types of discrepancies were identified and their functional implications where discussed. Also, possible solutions to reconcile these discrepancies were suggested.

7. REFERENCES

[1] Al-Kahtani, M., and Sandhu, R. A model for attribute-based user-role assignment. Proceedings of the18th Annual Computer Security Applications Conference (Las Vegas NV, December 2002).

[2] Dynamic Groups for LDAPV3 draft-haripriya-dynamicgroup-00.txt, (October 2001).

[3] Herzberg, A., Mass, Y., and Mihaeli, J. Access control meets public key infrastructure, or: Assigning roles to strangers. Proceedings of the 2000 IEEE Symposium on Security and Privacy, 2000.

[4] Lightweight Directory Access Protocol (v3), RFC2251, (December 1997).

[5] Park, J., Sandhu, R., and Ahn, G. Role-based access control on the web. ACM Transactions on Information and System Security, Vol. 4, No 1, 2001.

[6] Sandhu, R., Bhamidipati, V., and Munawer, Q. The

ARBAC97 model for role-based administration of roles. ACM Transactions on Information and System Security. Vol.2, No.1, February 1999, 105-135.

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

[8] Yao, W., Moody, K., Bacon, J. A model of OASIS role-based access control and its support for active security. SACMAT'01 (Chantilly VA, May 2001).

[9] Zhong, Y., Bhargava, B., and Mahoui, M. Trustworthiness based authorization on WWW. In IEEE workshop on "Security in Distributed Data Warehousing" (New Orleans, October 2001).