# Resolving Information Flow Conflicts in RBAC Systems

Noa Tuval[1] and Ehud Gudes[1 2]

[1] Department of Computer Science, Open University, Raanana, Israel.
[2] Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel.

**Abstract.** Recently, Role Based Access Control (RBAC) model has taken place as a promising alternative to the conventional access control models, MAC and DAC. RBAC is more general than those traditional models as was shown by Osborn et al. [17], however, mapping a role based system to a valid MAC configuration is not always possible because certain combinations of permissions that are included in a role's effective privileges may cause information flow. Given a role-based graph where role's permissions refer to labeled data objects, Osborn et al. showed how to find conflicts that are resulted from information flow, but they have not suggested a solution for these conflicts and they have not handled user-role assignments, for the solved scheme. In this paper, we assume a more general model of permissions conflicts than MAC. We introduce an algorithm that handles information flow conflicts in a given role-based graph, corrects the Role-based graph if needed, and proposes a consistent users-roles assignment. As RBAC and information flow are becoming extremely important in Web based information systems, this algorithm becomes very relevant.

**Keywords**: Role based access control, role graph consistency, canonical groups

## 1 Introduction

The RBAC (Role Based Access Control) model [21] has taken place, for several years now, as an alternative to the MAC (Mandatory Access Control) and DAC (Discretionary Access Control) models, as RBAC simplifies the access control management of complex systems, which contain large number of users, objects and applications [10], [17], [19].

According to MAC method, which is more relevant for our discussion – system elements such as data objects, users and sessions are labeled with security labels. The MAC (or LBAC) model can be represented by a lattice of security labels in which information flow is permitted in one direction only – from a low level to a high level [19]. RBAC is more general than the traditional access methods since it can be configured to enforce both MAC and DAC [17], [19]. Recently RBAC has become an important component of many Web systems and various standards specifying it have appeared [8].

An RBAC system can be illustrated as a hierarchial role-based graph, which is composed of a vertex set and an edge set. A vertex indicates a role, which contains certain privileges. An edge connects a role with its direct ancestor. Privileges of a role are of two kinds: *direct privileges* – privileges which are explicitly assigned to this role, and *effective privileges* – which include the role's direct privileges and the inherited effective privileges of all its juniors [15], see Fig. 1.

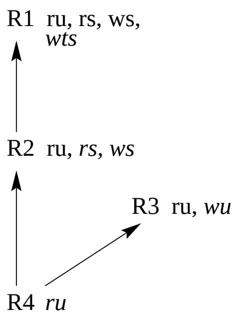


Fig. 1: A role-based graph including roles privileges. Direct privileges are marked by italics.

Conflict of interest must be considered while dealing with security. Several kinds of conflicts have been defined and discussed for the role-based graph model [15], of those, role-role conflict and privilege-privilege conflict, are the most relevant here. However, existing algorithms, which support role graph administration, do not try to solve such conflicts. For example, the PrivilegeAddition algorithm execution in [15], [10] aborts when such conflict is detected.

The problem of mapping a given role based system to a valid MAC configuration, was introduced in [17]. Given a role-based graph in which any role contains permissions to data objects that are labeled with security labels, Osborn et al. pointed at conflicts that are resulted from information flow, but they did not suggest any solution for these conflicts. In this paper we handle role-based graph conflicts from the following aspects:

- We address a more general model of information flow and permission conflicts then just MAC.
- We present an algorithm for detecting conflicts in a role-based graph and correcting them by creating new roles or assigning some of the conflicting permissions to existing roles. We then suggest a valid user-role assignment to the corrected graph. This algorithm is also very useful for Role-based administration.
- The resolution of conflicts is based on partitioning the permission-set to *canonical groups* that do not contain conflicts. Such partition raises interesting theoretical issues, which are also discussed.

The above are the main contributions of this paper. The rest of the paper is organized as follows: Section 2 discusses related work and reviews Osborn's work on information flow conflicts for the role-graph model. Section 3 focuses on role-based graph conflicts. We define several constraints, which are more general than those, which have been defined for the MAC model, and we introduce our role-graph consistency verification and correction algorithm. This is followed by a discussion of theoretical issues, which relate to constructing a partition of the role-graph to non-conflicting collections. Finally, in Section 4 we discuss possible applications and future work.

## 2 Related Work

The Role-based model was introduced first by Sandhu et al. [21] and was followed by many papers and standards [1], [11], [19], [20], [8]. An important area in RBAC research is assigning users to roles under various constraints. This was addressed in [4], [12]. Information flow models started with the Bell/Lapadulla model and the MAC methodology, and was followed by work in other directions, such as information flow in object-oriented databases [18], and information flow in distributed systems [14]. The latter works are relevant here, because one can derive from those models sets of permissions, which are in conflict for information flow reasons, and if one assigns such permissions to roles, it will get a role-graph model with possible conflicts. Our work mostly relates to Osborn's work, which investigated several aspects of the RBAC model, using the *role-graph* model. The role-graph model is discussed in [1], [2], [15], [16], [19]. In [17] Osborn et al. show the power of RBAC by configuring RBAC to enforce Mandatory and Discretionary Access control policies. Conflicts of interest and their reflection in a role-based graph are demonstrated in [15]. Role and permission administration for the role graph model are described in [10], [15], [23]. In the next section we briefly review those aspects of Osborn et al. work, which are most relevant to the issues we focus on in this paper.

### 2.1 Osborn's previous model

**Mapping role hierarchies to LBAC.** LBAC (Lattice Based Access Control) model uses labels to enforce multilevel security. The allowable paths of information flow in an LBAC system, are defined by Mandatory access rules as follows, where $\lambda$ is the security label of the specified entity:

**Definition 1.** ***Simple Security Property:*** *Subject $s$ can read object $o$ only if* $\lambda(s) \geq \lambda(o)$

**Definition 2.** ***Liberal (Strict)* Property:*** *Subject $s$ can write object $o$ only if* $\lambda(s) \leq (=)\lambda(o)$

In [17] Osborn et al. show how RBAC can be configured to enforce MAC policy. Nevertheless, the structure of role hierarchies that do map to valid LBAC configuration is greatly restricted since several permissions combinations that belong to a role's effective permission-set may cause information flow when assigning the role to a user. Figure 2 illustrates a role-based system, where a role contains permissions to data objects that are labeled with security labels (e.g. **rs** mean "read secret"). All of the roles except the role labeled R7 can be assigned to users without causing any information flow conflict (for example, the roles labeled R1 and R4 can be assigned to unclassified users and the roles labeled R3 and R6 can be assigned to secret users.) Making an assignment of the role labeled R7 to any user is not possible without violating either the Simple Security Property or the Liberal*-Property. Thus, the effective permission-set of R7 is not valid because it can't be assigned to any user without causing information flow.

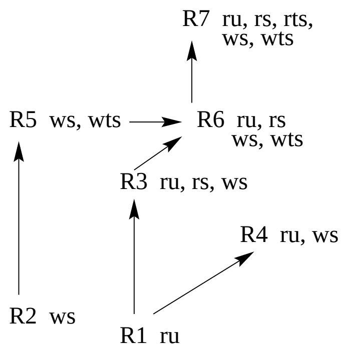


Fig. 2: A role-based graph, which includes a role ($R7$) that can't be assigned to any user, since it contains an illegal effective permission set.

Osborn introduces two more definitions to capture the maximum read level and the minimum write level (if exists) of objects in a role as follows:

**Definition 3.** ***The r-level of a role*** $R$ ***(denoted*** **r-level**$(R)$***)***
r-level(r) *is the least upper bound* $(lub)$ *of the security levels of the objects* $o$ *for which* $(o, r)$ *is in the permissions of* $R$.

**Definition 4.** ***The w-level of a role*** $R$ ***(denoted*** **w-level**$(R)$***)***
w-level(R) *is the greatest lower bound* $(glb)$ *of the security levels of the objects* $o$ *for which* $(o, w)$ *is in the permissions of* $R$*, if such a glb exists. If the glb does not exist, w-level is undefined.*

Using Definitions 3-4, Osborn defines the following constraint on user/role assignments UA:
**Constraint on UA:**

$$(\text{for every}(u, R)\text{that belongs to } UA[\lambda(u) \geq r\text{-}level(R)])$$

$$(\text{for every}(u, R)\text{that belongs to } UA[\lambda(u) \leq w\text{-}level(R)])$$

**Conflicting permissions and conflicting roles.** As was illustrated in Fig. 2, the pair (rts, ws) creates a *permission-permission conflict.* It obviously violates the constraint above and it means that the two permissions must not appear together. This example demonstrates an *information flow* conflict but there can be other kinds of conflicts among permissions such as *Object-based separation of duty*, or existence of both *positive and negative authorizations* on the same object. The current paper is focused on information flow conflicts. Nevertheless the presented algorithm is general enough to be used on any kind of permission-permission conflicts.

The basic constraint for conflicting permissions states that no role's effective permission-set may contain two permissions, which have been defined to be in conflict. When such a role is discovered it should be modified into one or more consistent roles. This modification is discussed later in the current paper.

Conflicting roles demand even tighter restrictions: If two roles are declared to be in a *Role-role conflict*, then a user authorized to one of the roles, must not be authorized to any of the permissions of the other role. As role-role conflicts apply great limitations on user-role assignments, they induce organizing graph's roles into collections of roles that can be assigned together. Nyanchama and Osborn [15] developed the following technique for partitioning the graph's roles to non-conflicting role collections, using a role-role Matrix $C$, which describes pairs of conflicting roles. For any two roles $r_i$ , $r_j$ that have been defined to conflict – $c_{i,j}$ is set to 1. Next, the dual matrix of $C$ (in which 1's are substituted for 0's vise versa) represents pairs of roles that can be assigned together. The dual matrix is represented by a graph and the cliques in this graph correspond to set of roles, which can be assigned together.

**Role graph administration.** Algorithms for manipulating role-based graph are also described in [15]. Graph administration includes role addition, role deletion, permission addition and deletion and edge insertion and deletion. Revised algorithms, which improve the original addition algorithms, are introduced in [10]. While the previous algorithms assume that when a permission $p$ is added to a role, all permissions that might be implied by $p$ are presented in the role automatically, the late addition algorithms actually add those permissions. The role-graph manipulating algorithms that are introduced in [15] and [10] only detect conflicting permissions and roles. The resolving of such conflicts is not dealt with. In case that a permission addition creates a conflict, the permission addition is not performed [10] or the algorithm execution abort [15].

## 3 Resolving role-based graph conflicts – our approach

### 3.1 Assignments Validation constraints for the MAC model

We first rewrite Osborn constraints in a slightly more general way, by using the concept of *effective permissions.*

**effective permissions** - the set of permissions assigned directly or inherited via the role-hierarchy.

**Valid effective permission-role constraint:** Effective permission set which is assigned to role $r$ is valid only if: $r\text{-}level(r) \leq w\text{-}level(r)$.

**Valid user-role assignment constraint:** A valid role r can be assigned to user $u$ only if: $\lambda(u) = r\text{-}level(r)$.

The *Valid user-role assignment constraint* satisfies MAC rules, since the minimal write level of objects in a valid role $r$, is greater than the maximal read level of objects in $r$, as the *Valid effective permission set for a role r constraint* assures. Therefore, no information flow occurs while using these assignments constraints. Next we discuss a more general model for permission information flow conflicts.

### 3.2 Assignments Validation constraints – Extended model

The model that has been described above is based on the MAC rules, which explicitly determine allowable paths of information flow. In this section we use more general definitions for conflict of interest that may cause information flow. For example Myer's model [14] refers to a system that includes two sets of objects, one that contains objects to which a read operation is defined and the second set contains objects to which a write operation is defined. It also supports multiple independent policies on the same object. The model defines the conditions under which a write operation can be performed on a certain object. When such operations are not allowed then a permission conflict can be defined. Our extended model below can then apply also to Myer's [14] or Samarati et al.[18] models. In the following definitions of the extended model, we use Read and Write operations, but in fact we could have used any conflicting permissions as explained above. Also, in the rest of the discussion we assume the hierarchical model of roles, where a role inherits all permissions of the roles below it.

**Extended model Definitions:**

- ***r-set*** is a set of objects for which a read permission is defined.
- ***r-set(R)*** contains all the read permissions that are assigned to a role $R$, where $r\text{-}set(R) \subseteq r\text{-}set$.
- ***w-set*** is a set of objects for which a write permission is defined.
- ***w-set(R)*** contains the write permissions that are assigned to a role $R$, where $w\text{-}set(R) \subseteq w\text{-}set$.
- ***c-set(R)*** is a set of conflicting permissions pairs of the form $(r_i, w_j)$ that can't exist together in a role $R$, where $r_i \in r\text{-}set$ and $w_j \in w\text{-}set$.
- ***R-set*** is the set of roles.
- ***R-set(u)*** contains the roles that are assigned to a user $u$, where $R\text{-}set(u) \subseteq R\text{-}set$.

Based on the extended model definitions, we can determine the following assignments constraints:

**Valid permission set for a role $R$ constraint – extended model:**
Permission set which is assigned to a role $R$ is valid only if

```
    for any $r_i \in r\text{-}set(R), w_j \in w\text{-}set(R)$
        $(r_i, w_j) \notin c\text{-}set(R)$
```

**Valid user-role assignment constraint – extended model:**
A valid role $R$ can be assigned to a user $u$ only if

```
    for any $r_i \in r\text{-}set(R), w_j \in w\text{-}set(R)$
      for any $R_k \in R\text{-}set(u)$ /* already assigned set of roles */
        for any $r_m \in r\text{-}set(R_k), w_n \in w\text{-}set(R_k)$
            $(r_i, w_n) \notin c\text{-}set(R) \cup c\text{-}set(R_k)$
            $(r_m, w_j) \notin c\text{-}set(R) \cup c\text{-}set(R_k)$
```

We like to show now that the above constraints will prevent information flow in the general case. Assume there is information flow from object $X$ to object $Y$. Then there must be a sequence of permissions:

$$P_1(X_1 = X), P_2(X_2), \ldots P_n(X_n = Y)$$

that caused this information flow. In this sequence there must be two permissions in conflict, otherwise there wouldn't be such a flow (proof by induction). Now if these two permissions are assigned to the same role, or to two roles assigned to the same user, the constraint above will detect it, and prevent such assignment. Therefore no information flow is possible. Next we present an algorithm to check the above constraints.

## 3.3 Role graph Consistency Verification Algorithm

In this section we introduce our resolution, which refers to the model that was described above. It shall be noticed that external regulations as administrative constrains, must be considered before operating the algorithm since such constraints might change the graph's initial configuration.

The purpose of our algorithm is twofold. The first is to check the validity of a given role graph and correct it if needed. The second is to find a valid user-role assignment to the corrected role graph. The algorithm is divided into the following phases:

**a. Creating a consistent graph**
Based on the extended model definitions, the algorithm checks every role's effective permission set for validity, while performing a recursive top-down walk on the role graph $G$. The algorithm's output is $G'$, the corrected role-based graph. The algorithm first phase is presented in Alg. 1

Since the algorithm descends the graph top-down, it assures that when a role is consistent (or resolved to be consistent), all its sons are already consistent. Therefore, the role-based graph which is output by this algorithm is consistent.

**b. Handling user-role assignments** The algorithm's second phase handles user assignments to the corrected role graph, and is shown in Alg. 2. This algorithm is very similar to the algorithm presented in [15], except that it accepts as input the already *m*odified role-based graph generated by Alg. 1. and in the last

**Algorithm 1** Role graph consistency verification algorithm

**Input** $G(N, \rightarrow)$ a role-based graph possibly containing information flow conflicts.

**Output** $G'(N, \rightarrow)$ a role graph based on $G$, which does not include any information flow conflict.

**Method** Resolving permission assignment conflicts

```
1: copy $G$ to $G'$
2: for every connected component of $G'$ do
3:     for every root of component of $G'$ do
4:         if root is consistent then exit
5:         for every son of root do
6:             perform a recursive Depth-first walk, in which:
7:             for every role $R$ do
8:                 for every $r_i$ , $w_j$ where $r_i \in r\text{-}set(R)$ , $w_j \in w\text{-}set(R)$ do
9:                     if $(r_i, w_j) \in c\text{-}set(R)$ then
10:                        create the canonical groups for $R$ effective permission set
11:                for every canonical group $cg$ do
12:                    combine $cg$ permissions to an existing role or create a new role
13:                    for $cg$ or for a part of it, according to the system's policy
14: Phase b
```

step it may handle additional user-user constraints as follows. At the last step one may assign users to the found cliques or to subsets of them. However, when individual users are assigned, additional user-user constraints such as separation of duty constraints may be present. A general scheme for user-role assignment with general constraints was presented in [12] using the techniques of Constraint processing (CSP). This technique is very powerful for solving various types of constraints and can works also in the distributed case as was shown in [13]. Next we discuss in detail the correction of inconsistent roles.

**The problem of correcting Role inconsistency.** Once we find an inconsistent role, we are faced with the problem of correcting it, and create a valid new and consistent role graph. There may be several approaches to solving this problem, And we discuss two of them here.

The first approach tries to locate the permissions, which are most "problematic", and remove them from the checked role. If we represent the permissions and the conflicts between them as a graph, we like to find minimal set of permissions that will remove all conflicts. This problem is equivalent to the Vertex-Cover problem and is known to be NP-hard [7]. For any permission removed from the checked role, we should first check if it already exists in another role, and only if it does not exist, we should create a new role for it. This new role may contain the removed permission plus all consistent permissions with it, which were in the checked role. The problem with this approach is that it tends to split the original permissions to too many groups.

Another approach uses the concept of *canonical groups.* A canonical group is a maximal set of permissions with no conflict in it. The problem of finding

**Algorithm 2** Handling user assignments to the consistent graph
**Input** $G'(N, \rightarrow)$ a role graph based on $G$, which does not include any information flow conflict.
**Output** Legal user-role assignments.

```
   Phase1 Performing a search for conflicting permissions that are assigned to separate roles and representing the conflicts by a graph
1: for every two roles $R_i$ , $R_j$ such that there exist permissions $p_1$ in $R_i$ and $p_2$ in $R_j$ that are in conflict do
2:     add the edge $(R_i, R_j)$ to the role conflict graph $GC$.
   Phase2 Constructing GC' – GC graph complement and Getting potentially legal user-role assignments
3: for every couple of roles $R_i$ , $R_j$ which appear in the role-based graph $G'$: do
4:     if the edge $(R_i, R_j)$ does not appear in $GC$ then
5:         add $(R_i, R_j)$ to $GC'$.
6: find all the cliques in graph $GC'$.                ▷ Get legal user-role assignments
   Phase3 Perform the actual user-role assignment considering all constraints
```

canonical groups is discussed in Section 3.4. For now we assume that the permission set was divided into a set of canonical groups. Then we suggest the following heuristics: When an inconsistent role $R$ is found, the graph correction has to be performed using the algorithm: Resolve_Using_Canonical_Groups

**Resolve_Using_Canonical_Groups** For every canonical group $cg$, which includes permissions that have to be removed from the effective permission set of $R$:

1. In case that there is not any role to which the permissions that are included in $cg$ are assigned – create a new role $R_N$ and assign $cg$ permissions to it. $R_N$ will be put above the role that contains the maximal subset of $cg$, in the role hierarchy.
2. In case that the role graph contains already a consistent role $R'$ to which $cg$ permissions are assigned:
   (a) If $R'$ does not contain any permission except $cg$ permissions – there is no need to perform any change.
   (b) In case that $R'$ – to which $cg$ permissions are assigned, contains additional permissions – the decision whether a creation of a new role for $cg$ permissions is needed – is depended on the system designer policy.
3. In case that the role graph contains a consistent role $R''$ to which only a part of $cg$ permissions are assigned – create a new role, $R_N$ , for those $cg$ permissions that are not included in $R''$. In case $R''$ contains only permissions which are in cg, put $R_N$ above $R''$ in the role hierarchy, otherwise perform step 2b above.

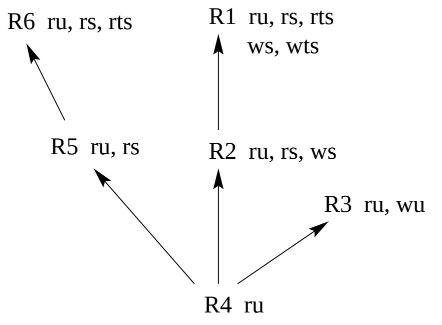
*In any case*: Delete $cg$ permissions from the original role's permission set.

**Theorem 1.** *The two algorithms, the verification and the resolving algorithms result with a consistent role-based graph.*

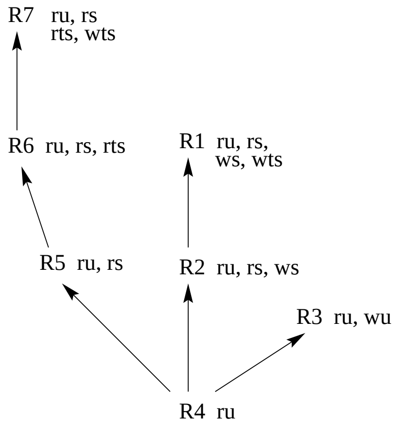
The proof uses mathematical induction and is omitted here for space reasons.

Note that the heuristics above is not necessarily “optimal”. One may define optimal by the following criteria: Find a division of the permission set of the checked role, such that the total number of changes (including role addition, permission addition and permission deletion) is minimized. We are still investigating this problem. The other issue is what we called ”system policy”. it is possible that an organization will have some policy constraints regarding the composition of roles and the hierarchy of roles, e.g. a specific branch of a hierarchy should not be modified. Such constraints should be taken into account when we assign the cannonical groups to roles or create new roles. We plan to investigate this issue too in the future.

Figures 3a, 3b illustrate the algorithm’s operation on a given inconsistent role based graph $G$. Note that for this example, we assumed that $R1$ is the only inconsistent role in the graph, therefore the canonical groups can be assigned to existing roles without checking those role’s permission sets for consistency. As explained earlier, the general algorithm can handle multiple inconsistent roles. Figure 4a shows the resultant role-conflicts graph and Fig. 4b shows the resultant dual graph $GC'$ and the resulting conflict-free cliques. Finally, as discussed earlier, the actual user/role assignment may be performed using the methods presented in [12].

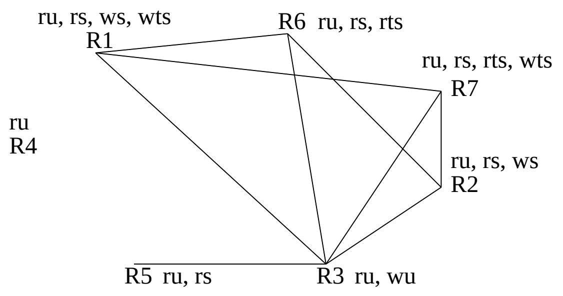


(a) Inconsistent Graph $G$. The role labeled $R1$ contains an inconsistent permission-set

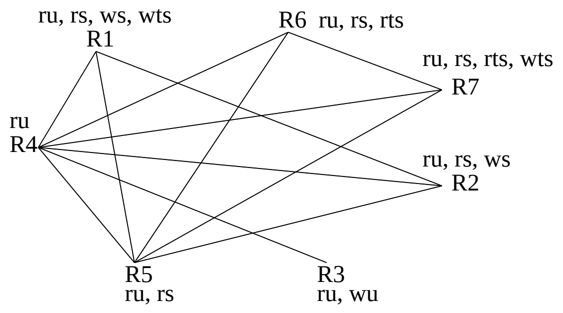


(b) Consistent Graph $G'$. $R1$ permission-set is divided into two canonical groups: $\{ru, rs, ws, wts\}$, which remains in R1 and $\{ru, rs, rts, wts\}$, which demands the creation of a new role – $R7$

Fig. 3: Resolving information flow conflicts in a role based graph



(a) $GC$ - the conflicting-roles graph which is derived of $G'$



(b) $GC'$ - the dual non-conflicting-roles graph

$$G_1 : \{R_1, R_2, R_4, R_5\}$$
$$G_2 : \{R_3, R_4\}$$
$$G_3 : \{R_4, R_5, R_6, R_7\}$$

Fig. 4: Finding conflict-free cliques for user-role assignment.

### 3.4 The problem of finding canonical groups

The problematic permission-set which is assigned to the role labeled R7 in Fig. 2, can be divided into non-conflicting groups in several ways, as it is illustrated in Table 1. One possible division is to include the read permissions in one group and the write permissions into another group, as is shown in line 1 of the table. An alternative resolution is to partition the permissions set into canonical groups. Canonical groups are defined to be maximal groups of non-conflicting elements, as is demonstrated in the second line of Table 1. In this case first the two conflicting permissions are divided between the two groups, and then the rest of the permissions can be assigned to both of the groups.

|  | **ru, rs, rts, ws, wts** |
|---|---|
| **1.** | ru, rs, rts / ws, wts |
| **2.** | rts, ru, rs, wts / ws, ru, rs, wts |

Table 1: Two optional divisions for the inconsistent permission set (ru, rs, rts, ws, wts) to non-conflicting groups .

Obviously, not all resolutions can be satisfied by forming only two cannonical groups, but one is usually interested in minimizing the number of such groups in order to minimize the required changes to the role-graph.

The problem of dividing a permission set into a minimal number canonical groups is similar to the problem of finding cliques in an undirected graph. A clique of a graph is a maximal complete sub-graph. Covering vertices by cliques (Vertex Clique Cover or Clique Partition), is an NP-complete problem [7], [22], and is also equivalent to the Graph Coloring problem: A graph has a vertex clique cover of size $k$ iff its complement graph can be colored with $k$ colors such that adjacent vertices have different colors [7]. Finding a minimal coloring can be done using brute-force search [6], [22], [24], but more efficient algorithms exist for some special cases. The *four-color theorem* establishes that all *planar graphs* (graphs that can be drawn in a plane without graph edges crossing) are 4-colorable [22].

*Two-colorable* graphs are exactly *bipartite graphs.* A *bipartite graph* is a set of graph vertices decomposed into two disjoint sets such that no two graph vertices within the same set are adjacent. A bipartite graph is a special case of a *k-partite graph* with $k = 2$. A graph is bipartite iff all its cycles are of even length [22]. To determine if a graph $G = (V, E)$ is bipartite, we perform a BFS search on it with a little modification, whose run time is $O(V + E)$ [7].

A related problem is covering the edges of a graph with a minimum number of cliques, which is known as the Clique Cover problem, and is also an NP-complete problem. For Clique Cover there is a solving algorithm, which is polynomial time heuristic from the 1970's. Gramm et al. present an improved version for this heuristic, using data reduction techniques, in [9].

In practice, it may very well be that two cannonical groups will be sufficient for resolving the inconsistency. So it will be worthwhile first to check if the graph is bi-partite, before attempting to use the more complex algorithms for finding more than two cannonical groups.

Note that the second algorithm, the user-role assignment also requires finding cliques, so similar theoretical problems arise. Finally, the actual user-role assignment under user-user constraints also present some theoretical questions, some of them are NP hard and some are polynomial and can be reduced to a network-flow problem. This is discussed in detail in [13].

### 3.5 Extending administration algorithms

Nyanchama and Osborn's basic algorithms for adding a permission to a role and for adding a role to the role based graph [15], contain the following lines for conflict of interest detection which checks any role that belongs to the graph's roles, after making the addition:

**If** *effective(r) contains a pair of privileges which is in p-conflicts* **then abort**.

Our solution in this case is to perform the *Role graph consistency verification algorithm*, which checks any role for consistency and makes graph corrections using the algorithms discussed above. Note that the organization (system) policies may be different when a permission is added to a role, or when a role is added to a hierarchy and these should be taken into account.

## 4 Discussion and future work

The algorithms above can be useful in several applications and also raise several outstanding issues. These issues and future research are discussed in this section.

1. **User-role assignment and delegation**. As was discussed above the role hierarchy does not limit user-role assignments only to roles along the same hierarchy. However, if we allow delegation of roles from one user to another, the problem becomes more difficult. Such delegation requires running the verification algorithm for each such delegation, which may create considerable overhead. One possible restriction is to allow delegation only along the role hierarchy. Obviously, if a user of role $R_i$ delegates to a user with role $R_j$ where $R_i < R_j$ then no role conflict will occur, since originally the graph is consistent and role $R_i$ has inherited all permissions of $R_j$. Therefore, if users are assigned to roles only along a single hierarchy, delegation will be consistent and no verification will be needed (except for constraints such as SOD). Otherwise, the algorithm needs to be executed for each such delegation.
2. **Dynamic user-role assignment**. The restrictions above on static role assignment (i.e. assigning users only conflict-free cliques) may put heavy restriction on a real life system. Similar to the MAC policy, if we enforce the static * property, then an employee who has a top-secret job cannot write an email to his wife who has a confidential level. The *water-mark* policy used in MAC systems can be used here. Thus, roles can be assigned dynamically to an application based on its dynamic needs, and therefore role conflicts must be checked dynamically. Furthermore, the application may require a temporary assignment of hierarchy between roles. In that case, the verification algorithm will need to be executed as well. Note that even in the dynamic case, to detect the conflicts one does not need to create a log of the operations performed. If we extend the "water-mark" idea to permissions and allow assignment of conflicting roles to the same user, then we just need to create a log of permissions and make sure no conflicting permissions were assigned.

3. **Distributed systems**. The algorithm can be also used in distributed systems. For example, a system composed of two sub-systems, anyone of them contains data-base and files. Suppose that a user gets access to sub-system1, by assigning him to a role $R$ that belongs to sub-system1's roles. In case that the user is also assigned to certain roles in sub-ststem2, the system has to check whether the new permissions he gets are not conflicting with his sub-system2's permissions. This check and the corrections that follow it are made using the *Role graph consistency verification algorithm*.
4. **Integrating multi-domains**. The problem of integrating the security requirements of multiple domains is a serious problem in our cooperative business world (see [3], [5]). The integration of such multiple domains requires the mapping of two separate role hierarchies into a single hierarchy. Such integration will require an algorithm, which is basically a generalization of the single verification and correction algorithm shown here and is a topic of future research
5. **The problem of optimal correction**. As was demonstrated in Section 3, there may be several policies to correct an inconsistency in role definition. The criteria of optimality, is not very obvious, and both the criteria and the algorithm required to satisfy it are a subject for future research. Related to that is the issue of satisfying system policy and organization constraints.
6. **Other kinds of permission-permission conflicts**. Although our model can be implemented on any kind of permission-permission conflicts, it refers mostly to information flow conflicts. Solving other kinds of permission-permission conflicts may force different criteria for division to canonical groups than has been introduced here. This is also a topic for future work.

**Acknowledgement** We thank Max Binshtok for helping with the editing and LaTeX of the manuscript.

## References

1. G. J. Ahn, Specification and Classification of Role-Based Authorization Policies: IEEE Computer Society, 2003.
2. A. Belokosztolszki, D. Eyers, K. Moody, Policy Contexts: Controlling Information Flow in Parameterised RBAC, IEEE Computer Society, 2003.
3. P. Belsis, S. Gritzalis: A scalable Security Architecture enabling coalition formation between autonomous domains. Proceedings of ISSPIT2005, Athens, Greece, 2005.
4. E. Bertino, E. Ferrari, V. Atluri: The Specification and Enforcement of Authorization Constraints in Workflow Management Systems, ACM Trans. Inf. Systems. Security. 2(1): 65-104 (1999).
5. E. Bertino, J. Joshi, R. Bhatti, A. Ghafoor: Access-Control Language for Multidomain Environments. IEEE Internet Computing 8(6): 40-50 (2004).
6. N. Christofides: An Algorithm for the Chromatic Number of a Graph, Computer J. 14, 38-39, 1971.
7. T. Cormen, C. Leiserson, R. Rivest: Introduction to Algorithms, MIT Press, Cambridge, 83, 89 506-539, 1990.

8. D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn and R. Chandramouli: Proposed NIST Standard for Role-Based Access Control, ACM Transactions on Information and System Security, Vol. 4, No. 3, August 2001, Pages 224-274.
9. J. Gramm, J. Guo, F. Huffner, R. Niedermeir: Data Reduction, Exact and Heuristic Algorithms for Clique Cover, In Proceedings of the 8th Workshop on Algorithm Engineering and Experiments (ALENEX 2006), Miami, USA, January 2006.
10. C. M. Ionita, S. Osborn: Privilege administration for the role graph model. In Proc.IFIP WG11.3 Working Conference on Database Security, July 2002.
11. J. Joshi, E. Bertino, B. Shafiq, A. Ghafoor: Dependencies and Separation of Duty Constraints in GTRBAC, SACMAT'03 June 2-3, 2003.
12. I. Moodahi, E. Gudes, O. Lavee, A. Meisels: A SecureWorkflow Model Based on Distributed Constrained Role and Task Assignment for the Internet, ICICS 2004: 171-186.
13. I. Moodahi, E. Gudes, A. Meisels: A three tier architecture for Role/User assignment for the Internet, submitted for a journal publication.
14. A. C. Myers, B. Liskov: A Decentralized Model for Information Flow Control (1997) Proceedings of the 16th ACM Symposium on Operating Systems Principles, Saint-Malo, France, October 1997.
15. M. Nyanchama, S. Osborn: The Role Graph Model and Conflict of Interest, ACM Transactions on Information and Systems Security, vol. 2, no. 1, (1999) 3-33.
16. S. Osborn: Information Flow Analysis of an RBAC system, SACMAT02, June 3-4, 2002.
17. S. Osborn, R. Sandhu and Q. Munawer: Configuring Role-Based Access Control to enforce Mandatory and Discretionary access control policies, ACM Trans. Information and system security, 3(2): 1-23, 2000.
18. P. Samarati, E. Bertino, A. Ciampichetti, S. Jajodia: Information Flow Control in Object-Oriented Systems. IEEE Trans. Knowl. Data Eng. 9(4): 524-538 (1997).
19. R. Sandhu: Lattice-based access control models, IEEE Computer 26, 11, 9-19, 1993.
20. R. Sandhu: Role Hierarchies and constraints for lattice-based Access Controls, Proc. Fourth European on Research in Computer Security, Rome, Italy, September 25-27, 1996.
21. R. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman: Role-based access control models, IEEE Computer 29, 2, 38-47, 1996.
22. S. Skiena: Finding a Vertex Coloring, 5.5.3 in Implementing Descrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley, pp. 141, 214-215, 1990.
23. H. Wang, S. Osborn: An Administrative Model for Role Graphs, Proc. IFIP WG11.3 Working Conference on Database Security, Estes Park, Colorado, 2003.
24. H. Wilf, Backtrack: An O(1) Expected Time Algorithm for the Graph Coloring Problem, Info. Proc. Let. 18, 119-121, 1984.