SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2010; **00**:1–11
DOI: 10.1002/sec

RESEARCH ARTICLE

Role Mining Based on Permission Cardinality Constraint and User Cardinality Constraint†

Xiaopu Ma[1,2], Ruixuan Li[1], Hongwei Wang[3],Huaqing Li[1], Xiwu Gu[1*]

1.School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei, 430074, China
2.School of Computer and Information Technology, Nanyang Normal University, Nanyang, Henan, 473061, China
3.School of Automation, Huazhong University of Science and Technology, Wuhan, Hubei, 430074, China

ABSTRACT

Constraint is an essential aspect of RBAC and is sometimes argued to be the principle motivation for RBAC. However, most of role mining algorithms don't consider the constraint. Furthermore, they just compare the least cost of the authorization process while don't consider how to assess the accuracy of the derived role state, thus providing the motivation for this work. In this paper, we first define a wide variety of constraints, especially the permission cardinality constraint and user cardinality constraint. We further propose a role mining algorithm to generate roles based on these two kinds of cardinality constraint that considers the similarity between roles in the process of merging roles in order to improve the accuracy of the role state at the same time. Finally, we carry out the experiments to evaluate our approach. The experimental results demonstrate the effectiveness of our proposed algorithm. Copyright © 2010 John Wiley & Sons, Ltd.

KEYWORDS

RBAC; role engineering; cardinality constraint; role similarity

Received . . .

1. INTRODUCTION

Since Role-Based Access Control (RBAC) model has been widely deployed in enterprise security management and enterprise management products [1], people are focusing on how to migrate a non-RBAC system to an RBAC system. At present, there are two methods on creating a comprehensive framework for defining the architectural structure of RBAC: one is the top-down approach and the other is the bottom-up approach [2]. Under the top-down approach, roles can be derived through analyzing the user scenario and business process [3][4]. The roles generated by this method are more suitable for the enterprise and organization's actual situation since it can meet the system's practical function. However, this approach is time consuming and costly because there may be dozens of business processes and tens of thousands of users in an organization [5]. Furthermore, it may analyze business logic again to satisfy the certain change of the organization's department. As a result, researches have proposed to use data mining techniques to discover roles from the existing user permission assignment relationships. Such method is called bottom-up approach [6].

Under the bottom-up approach, roles can be generated automatically or semi-automatically according to the existing user permission assignment relationships before RBAC is implemented. According to the used technologies, the bottom-up role mining algorithms can be divided into four categories. In the first approach, user permission assignment boolean matrix is decomposed into two boolean matrixes: one is user to role assignment boolean matrix, the other is role to permission assignment boolean matrix [7]. The second approach can get the rest of roles by merging or intersecting the role sets based on assuming some original permission sets as roles because the role in RBAC is actually a set of permissions [8]. Furthermore, roles can be taken as points that contain some permission elements, and the hierarchical relationships among roles can be regarded as edges that can discover new roles continuously using graph theory till the algorithm can't find other new roles or the value of objective function decreasing [9]. Finally, it also can apply clustering idea in data mining to discover roles. However, it differs from the traditional partition clustering in which each sample point only belongs to one category while each permission

†Correspondence to: Ruixuan Li, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China. E-mail: rxli@hust.edu.cn

can belong to several categories or roles. Hence, traditional clustering algorithm should be changed to adapt to the role mining [10]. Under the bottom-up approach is likely to ignore the business functions of an organization but can save large amounts of human resources. While the top-down approach is likely to ignore the existing user to permission assignment relationships, a bottom-up approach may not consider business functions in an organization. However, the bottom-up approach is advantageous in that, much of the role engineering process can be automated or semi-automated. Hence, the bottom-up approach for role engineering has raised significant interests in the research community.

In the approach to migrate a non-RBAC system to an RBAC system, however, a key challenge that has not been adequately addressed so far is how to mine roles based on constraints. In other words, most of the existing role engineering approaches just captured the organization's business rules as relate to access control, and reflected these rules in defining, naming, structuring, and constructing a valid set of roles. There are not many techniques on how to mine roles based on constraints since constraint is an important aspect of RBAC. In this paper, our approach takes permission cardinality constraint and user cardinality constraint into consideration in the process of role mining. Furthermore, we also consider the similarity between roles in the process of merging roles in order to improve the accuracy of the role state at the same time. Experiments on performance study are given to prove the superiority of our approach.

The remainder of this paper is organized as follows. We discuss the related work in role mining based on constraint in Section 2 and introduce our role mining algorithm based on permission cardinality constraint and user cardinality constraint in Section 3 and Section 4, and show the experimental results in Section 5. At last, we conclude the paper and discuss the future work in Section 6.

2. RELATED WORK

The most frequently mentioned constraint in context of RBAC is *mutually exclusive constraint* [11], In terms of a many-to-many user to role assignment relationships, this constraint specifies that one individual cannot be a member of both exclusive roles. In terms of a many-to-many permission to role assignment relationships, this constraint specifies that the same permission cannot be assigned to both role. Another example is *cardinality constraint* [12]. There are four types of different cardinality constraints: the cardinality constraint of user is defined as the the maximum number of users to which a role can have, the cardinality constraint of role is defined as the maximum number of roles to which an individual user can belong to, the cardinality constraint of permission is defined as the maximum number of roles to which a permission can belong to and the number of permissions which a role can own should also have cardinality constraint. Although the constraint is essential for the RBAC model, only a few researches took the constraint into account in role mining. According to the different type of constraint, the role mining algorithm based on constraint can be divided into five categories.

1) The first class constrained role mining algorithm satisfies a cardinality condition that no role contains more than a give number of permissions. The representative algorithm of the first class is proposed by Ravi Kumar and Blundo. In [13], a heuristic algorithm called Constrained Role Miner (CRM) can generate a set of roles that each one satisfies the given cardinality constraint through the user to permission assignment relationships. In [14], another heuristic algorithm is proposed to generate an initial candidate role set on the basis of the existing user to permission assignment relationships firstly. Then the candidate role set is refined and updated in order to ensure that the cardinality constraint is respected, finally the optimization role set is performed by running a lattice reduction procedure.

2) The second class constrained role mining algorithm satisfies a cardinality condition that no role contains more than a given number of users. In the first class constrained role mining may be lead to the user with too many roles. For example, if the user has ten permissions in the current user to permission assignment relationships while the cardinality condition that no role contains more than two permissions, in such situation, at least five roles have to be assigned to the user. The situation may be even worse when only one permission is limited to the role. In other words, the number of roles assigned to the user will increase with the number of permissions that can be assigned to the role decreases. Furthermore, this situation will make the administration difficult at the same time. According to the above case, Hingankar and Sural [15] propose a biclique cover method to generate a set of roles that limits the maximum number of users for a role.

3) The third class constrained role mining algorithm satisfies a cardinality condition that no user contains more than a given number of roles. The representative algorithm of the third class is Role Priority based Approach (RPA) and Coverage of Permissions based Approach (CPA) [16]. The RPA algorithm finds the roles and prioritizes the roles, and then limits the number of roles assigned to a user using this priority order, while the CPA chooses roles by iteratively picking the role with the largest number of permissions that are not yet assigned to that user by any other role, and then imposes no user contains more than a given number of roles.

4) The fourth class constrained role mining algorithm satisfies a cardinality condition that no permission belongs to more than a given number of roles. In [17], Pullamsetty Harika et al. propose the post-processing framework and concurrent processing framework to solve the fourth class constrained role mining problem which is denoted

Security Comm. Networks 2010; **00**:1–11 © 2010 John Wiley & Sons, Ltd.
DOI: 10.1002/sec
Prepared using secauth.cls

as the permission-distribution cardinality constraint. In the first framework, roles are initially generated without considering the cardinality constraint. Once the user to role and role to permission assignment relationships are obtained, it is checked and fixed when the constraints are violated. On the other hand, in the second framework, the constraints will be imposed during the process of role mining in order to mining roles satisfy the existing constraints.

5) The fifth class constrained role mining algorithm satisfies the separation of duty or exception constraint. Suppose, on one hand, the RBAC policy states that any user with the "Manager" role can access the "Project A" file, on the other hand, there exists an exception to this policy stating that all users except John is not allowed to access "Project A" due to a certain conflict of interest requirement. Furthermore, the permission to issue checks and the permission to audit the operations should not be assigned to the same role in order to obey the separation of duty. In [18], Haibing Lu et al. introduce the negative permission for the user and the negative permission for the role to resolve the exceptions and separation of duty respectively. As for the user obtains the negative permission, the user cannot active the negative permission for ever. As for the role obtains the negative permissions, the role cannot active the negative permission for ever.

However, most of the above constrained role mining algorithm only consider single constraint. The combined effect on the mining process when many constraints are considered simultaneously needs to be studied. As for the four cardinality constraints, both of them are mutually exclusive. For example, each user will be assigned to a large number of roles according to the user's permissions in the initial user permission assignment relationships if the number of permissions which a role can own has cardinality constraint. This situation can lead to violate role user cardinality constraint. Another example, each permission will be assigned to a large number of roles if the number of user members owned by one role has cardinality constraint. This situation can lead to violate role permission cardinality constraint. Because both of the cardinality constraints are mutually exclusive, this paper focuses on the user cardinality constraint and permission cardinality constraint in the process of mining roles. Furthermore, they just compare the least cost of the authorization process while don't consider how to assess the accuracy for the derived role state. Hence, in this paper, we will propose a constrained role mining algorithm to generate roles based on permission cardinality constraint and user cardinality constraint at the same time which will consider the similarity between roles in the process of merging roles in order to improve the accuracy of the role state. Finally, we evaluate the method over a number of real-world data sets.

3. PRELIMINARIES

We develop the material in this paper in the context of the NIST standard, the most widely known RBAC model. This model includes RBAC0, RBAC1, RBAC2 and RBAC3 [1].

Definition 1
(RBAC0 Model) The RBAC0 model contains the following components:

- U, P, R, OPS, OBJ, users, permission, roles, operations and resources respectively;
- $UA \subseteq U \times R$, a many-to-many user to role assignment relationships;
- $PA \subseteq R \times P$, a many-to-many role to permission assignment relationships;
- $UPA \subseteq U \times P$, a many-to-many user to permission assignment relationships;
- $ass\_perms(u) = \{p \in P | (u, p) \in UPA\}$, the mapping of user u onto a set of permissions;
- $ass\_users(p) = \{u \in U | (u, p) \in UPA\}$, the mapping of permission p onto a set of users;
- $ass\_perms(r) = \{p \in P | (r, p) \in PA\}$, the mapping of role r onto a set of permissions;
- $ass\_users(r) = \{u \in U | (u, r) \in UA\}$, the mapping of role r onto a set of users.

Furthermore constraint is an important function of RBAC which even can be viewed as an important motivation for RBAC coming into being. Constraint is mostly added to *UA* and *PA*, but sometimes it is also added to the user set and role set in session. The main constraints are defined as follows:

Definition 2
(**Mutually Exclusive Constraint**) The mutually exclusive constraint in *UA* specifies that the same user only can be assigned to one role of a mutually exclusive role set. For instance, a user can't possess both accountant and cashier roles at the same time. The mutually exclusive constraint in *PA* specifies that the same permission can only be assigned to one role of a mutually exclusive role set. For example, suppose that both role *A* and *B* have permissions to give students' score, but perhaps only one role can have this permission.

Definition 3
(**Role-User Cardinality Constraint**) The role-user cardinality constraint can be defined as the limited number of roles that a particular user can have.

Definition 4
(**Role-Permission Cardinality Constraint**) The role-permission cardinality constraint can be defined as the limited number of roles that a particular permission can be assigned.

Definition 5
(**User Cardinality Constraint**) The user cardinality constraint can be defined as the maximum number of user that a role can have in the system.

Definition 6
(**Permission Cardinality Constraint**) The permission cardinality constraint can be defined as the maximum number of permission that a role can have in the system.

Definition 7
(**Prerequisite Constraint**) There are two situations about prerequisite constraint. One is prerequisite constraint of role which specifies that one user can obtain role *A* only after the user has obtained role *B*. The other is prerequisite constraint of permission. A permission called *p* can be assigned to a certain role only after the role possesses permission *q*.

In this paper, we will discuss the role mining algorithm based on permission cardinality constraint and user cardinality constraint firstly. Furthermore, in previous role mining algorithm, new roles are derived by merging existing roles only considering whether the role assignment cost will decrease after merging current two roles while not considering the similarity of the two combination roles, in this situation which will result in lower accuracy of the final role set. For example, Fig.1 shows the third situation of *Go* algorithm [19], that is, roles r_1 and r_2 have common permission set $\{p_2, p_3, p_4, p_5\}$. Then the two roles are merged according to *Go* algorithm to derive a new role r_3 which don't consider the similarity information of both roles. However, the essential concept of role mining is to cluster permission into roles based on the similarity or proximity of the two samples. Therefore, in order to improve the accuracy of role set, we will consider the similarity between roles in the role mining algorithm when merge two roles to acquire a new role. In order to evaluate similarity of two roles, we define the following concepts.

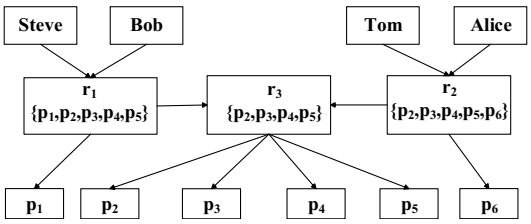

Figure 1. Merge two roles with common permissions

Definition 8
(**Permission Similarity**) The permission similarity between the *i*th role and the *j*th role is defined as

$$PermSim(r_i, r_j) = \frac{ass\_perms(r_i) \bigcap ass\_perms(r_j)}{ass\_perms(r_i) \bigcup ass\_perms(r_j)}$$

Definition 9
(**User Similarity**) The user similarity between the *i*th role and the *j*th role is defined as

$$UserSim(r_i, r_j) = \frac{ass\_users(r_i) \bigcap ass\_users(r_j)}{ass\_users(r_i) \bigcup ass\_users(r_j)}$$

Definition 10
(**Inheritance Hierarchy Structure Similarity**) The inheritance hierarchy structure similarity between the *i*th role and the *j*th role is defined as some linear relation of the number of direct parent and child roles of the *i*th role and *j*th role

$$StructSim(r_i, r_j) = \frac{1}{2} \times \frac{min(|Sen(r_i)|, |Sen(r_j)|)}{max(|Sen(r_i)|, |Sen(r_j)|)} + \frac{1}{2} \times \frac{min(|Jun(r_i)|, |Jun(r_j)|)}{max(|Jun(r_i)|, |Jun(r_j)|)}$$

Definition 11
(**Inheritance Hierarchy Content Similarity**) The inheritance hierarchy content similarity between the *i*th role and the *j*th role is defined as some linear relation of the number of direct parent and child roles of the *i*th role and *j*th role:

$$ContentSim(r_i, r_j) = \frac{1}{2} \times \frac{|Sen(r_i) \bigcap Sen(r_j)|}{|Sen(r_i) \bigcup Sen(r_2)|} + \frac{1}{2} \times \frac{|Jun(r_i) \bigcap Jun(r_j)|}{|Jun(r_i) \bigcup Jun(r_2)|}$$

Definition 12
(**Inheritance Hierarchy Semantics Similarity**) The inherited hierarchy semantics similarity between the *i*th role and the *j*th role is defined as a certain linear relation of two ratios, one of which is between common permission number and all permission number that two roles have, the other is between common user number and total user number that two roles have.

$$SemSim(r_i, r_j) = \frac{1}{4} \times \frac{|rpSen(r_i) \bigcap rpSen(r_j)|}{|rpSen(r_i) \bigcup rpSen(r_j)|} + \frac{1}{4} \times \frac{|rpJun(r_i) \bigcap rpJun(r_j)|}{|rpJun(r_i) \bigcup rpJun(r_j)|} + \frac{1}{4} \times \frac{|ruSen(r_i) \bigcap ruSen(r_j)|}{|ruSen(r_i) \bigcup ruSen(r_j)|} + \frac{1}{4} \times \frac{|ruJun(r_i) \bigcap ruJun(r_j)|}{|ruJun(r_i) \bigcup ruJun(r_j)|}$$

where $rpSen(r_i)$ represents all permissions that belong to the $Sen(r_i)$, $ruSen(r_i)$ represents all users of role $Sen(r_i)$. Hence, we can define the inheritance hierarchy relation similarity as follows:

Definition 13
(**Inheritance Hierarchy Relation Similarity**) The inheritance hierarchy relation similarity between the *i*th role and the *j*th role is defined as some linear relation of the inheritance hierarchy structure similarity, the inheritance hierarchy content similarity and the inheritance hierarchy semantics similarity.

$$RoleSim(r_i, r_j) = wcs \times ContentSim(r_i, r_j) + wses \times SemSim(r_i, r_j) + wscs \times StructSim(r_i, r_j)$$

Let's set $wcs = \frac{1}{2}, wses = \frac{1}{3}, wscs = \frac{1}{6}$, the reason is because the inheritance hierarchy relation similarity between the *i*th role and the *j*th role depends mostly on content similarity on the one hand, on the other

Security Comm. Networks 2010; **00**:1–11 © 2010 John Wiley & Sons, Ltd.
DOI: 10.1002/sec
Prepared using secauth.cls

hand semantics similarity is more important than structure similarity. Of course, users can adjust these parameters based on actual demands in practical applications. Based on above definition, we can define the similarity between roles as follows:

Definition 14
(**Similarity between Roles**) The similarity between the *i*th role and the *j*th role is defined as:

$$Sim(r_i, r_j) = wsp \times PermSim(r_i, r_j)$$
$$+wsu \times UserSim(r_i, r_j)$$
$$+wsh \times RoleSim(r_i, r_j)$$

Where $wsp + wsu + wsh = 1$ and wsp, wsu, wsh are parameters used to adjust the relative importance about the permission similarity between roles, the user similarity between roles and the inheritance hierarchy relation similarity between roles respectively. If we have no prior knowledge of the different similarity between roles, we can set $wsp = wsu = wsh = \frac{1}{3}$.

When the role set is generated, they should be assigned to each user according to existing user to permission assignments. In this procedure, the assignment cost is composed of the number of roles(*R*), user to role assignment relationships(*UA*), permission to role assignment relationships(*PA*), role hierarchy (*RH*) and direct user to permission authorization (*DUPA*). Formally, the role assignment cost is denoted as *WSC* (Weighted Structural Complexity) in role engineering area which can be defined as follows [20]:

Definition 15
(**Weighted Structural Complexity**) Given a five-tuple $W = (wr, wu, wp, wh, wd)$, where $wr, wu, wp, wh, wd \in Q + \bigcup\{\infty\}$, the weighted structural complexity of an RBAC state γ is defined as follows:

$$WSC(\gamma, W) = wr \times |R| + wu \times |UA| + wp \times |PA|$$
$$+wh \times |RH| + wd \times |DUPA|$$

Where $|.|$ denotes the size of the set or relation. Intuitively, in role mining, we would like to find an RBAC state that has the smallest weighted structural complexity. One can adjust the weights to limit the RBAC states to be considered and to meet different optimization objectives. In this paper, we mainly consider two weight schemes $W_1 : wr = wu = wp = wh = wd = 1$, and $W_2 : wr = wu = 1, wp = wh = wd = 2$. Suppose that, in the five-tuple W_1, the cost of appending each element to RBAC state is equal to 1. Hence, *WSC* measures the cost of creating RBAC state in this scheme. As for the other five-tuple W_2, it assumes that the operation of permission is more expensive in this situation. For example, in some systems an authorization policy file is demanded when assigning permission to some role, however, adding a user to a role only needs to add this role into user table.

In role engineering, optimization function can be viewed as a multi-objective optimization problem which generally has a series of objective functions to minimize or maximize in order to generate a set of roles. These conflict objects need balance. Formally, there are two objectives, one is minimum role assignment cost *WSC*, the other is maximum similarity of two roles. A simple method is to define a weighted global optimization function containing various optimal objectives. But similarity is a number between 0 and 1, while *WSC* is not. Hence, the weighted global optimization function can be defined as follows:

Definition 16
(**Weighted Global Optimization Function**) Given a *WSC* of a role state and a similarity between roles *Sim*, the weighted global optimization function can be defined as:

$$GOF(WSC, Sim) = (1 - wf) \times WSC + wf \times WSC \times Sim$$

Where $wf \in [0, 1]$ is a user defined weight factor. If $wf = 0$, only *WSC* of role assignment cost is considered in this situation, if $wf = 1$, only the similarity between roles is considered in this situation (Also in this situation, we can delete *WSC* from the *GOF* function as $GOF = Sim$, where *Sim* will be multiplied by *WSC* is just to promote *Sim* to the range of *WSC* for comparison). To facilitate the experiment, in this paper we set $wf = \frac{1}{2}$ and they can be adjusted as needed in practice.

4. ROLE MINING BASED ON PERMISSION CARDINALITY CONSTRAINT AND USER CARDINALITY CONSTRAINT

4.1. Algorithm Framework

In this section, we will design a role mining algorithm to generate roles based on permission cardinality constraint and user cardinality constraint (which we call UPCRM algorithm) on the one hand, on the other hand minimize the assignment cost of the discovering role set using the global optimal function (GOF) into consideration when merging roles to create new ones. In this algorithm, we will give a same permission cardinality constraint and user cardinality constraint because we don't know what roles are in the system at first. Algorithm 1 gives the framework about how to generate roles based on permission cardinality constraint and user cardinality constraint. The algorithm consists of three phases: initial role set generation, role selection and role state generation.

4.2. Initial Role Set Generation Algorithm

Initial role set can be achieved through two methods: one is from the prerequisite role set, the other is from the initial role set generation algorithm. When all the candidate role don't violate the permission cardinality constraint, for each role r, algorithm examines other roles to find whether they contain all *r's* permissions. If it is true, the count will be added with 1. Then algorithm selects candidate roles with

maximum count that satisfy the user cardinality constraint to produce initial roles, and removes all permissions of new initial roles from all candidate roles which contain initial ones after considering all candidate roles. Repeat above process until no more new initial roles generated. Algorithm 2 describes this idea.

Algorithm 1. UPCRM

Input:*U*,*P*,*UPA*
Prerequisite Role Set(*PreRoleSet*)
User Cardinality Constraint(*UC*)
Permission Cardinality Constraint(*PC*)
Output:*UA* and *PA* satisfy *UC* and *PC* or no output
1: **if** (*UC*, *PC* and *UPA* are contradictory) **then**
2: return no solution
3: **end if**
4: Complete *PreRoleSet* according to *UPA*
5: Call initial role set generation algorithm to produce *InitialRoleSet*
6: **while** there can be merged again **do**
7: Call role selection algorithm to select two roles
8: Call role state generation algorithm to update role's state and derive new *RoleSet*
9: **end while**
10: Return *RoleSet*

If the number of all users' permissions exceeds permission cardinality constraint after removing prerequisite roles' permission set from the user permission assignment relationships. Thus the following alternative algorithm can be adopted to produce initial role set: Suppose the permission cardinality constraint is k, user cardinality constraint is l. In the first loop, algorithm selects k permissions at random from all permission set. Then scans all users, if current user's permission contains those k randomly selected permissions, adds 1 to the user count of the k permission. The k permission's user count is derived after traversing all users. In the second loop, the algorithm randomly selects other permission set that is different from previous permission. Then repeats above steps to derive user count of current selected k permissions until the algorithm can't generate new k different permissions any more. Finally, sorts all k randomly selected permissions in descending order according to the user count. Then traverses this sorted list to find the first k permission set satisfied with the user cardinality constraint l and makes them as an initial role. Removes all permissions of the initial role from the user permission assignment relationships and reviews the rest of user role assignment whether meets the *CRM* algorithm requires. If meets, the algorithm will generate other initial roles according to *CRM* algorithm. Otherwise, the algorithm will continue to select from list. It is easy to see that can be generated the rest of initial role set through the *CRM* algorithm.

4.3. Role Selection Algorithm

We use the following method to select a pair of roles: sorts the role sets in descending order by the number of permissions contains in the role set, if the number of permissions in both of the role set is same, then sorts them by the number of users in descending order. Furthermore, if the number of permissions and the number of users are same, sorts the role set with hierarchical firstly then the non-hierarchical ones secondly. Finally, both of the role sets will be sorted randomly.

Algorithm 2 Initial Role Generation Algorithm

Input:*U*, *P*,*UPA*
Prerequisite Role Set(*PreRoleSet*)
User Cardinality Constraint(*UC*)
Permission Cardinality Constraint(*PC*)
Output: InitialRoleSet
1: **for** *i*:=1 **to** sizeof(*U*) **do**
2: **if** (u_i's permission set contains permission set of some prerequisite role k) **then**
3: Remove permission set of prerequisite role k from u_i's permission set
4: **end if**
5: **end for**
6: Generate InitialCandidateRole set according to each user's remaining permission set
7: Sort InitialCandidateRole set in descending order by the size of permission set
8: Divide InitialCandidateRole set into disMeetRoles and MeetRoles according to PC
9: **while** sizeof(disMeetRoles)$>$ 0 **do**
10: **for** i:=1 **to** sizeof(MeetRoles) **do**
11: Calculate $MeetRoles_i$'s user count within disMeetRoles
12: **end for**
13: Select candidate role R_k with maximum count and add it into InitialRoleSet
14: Remove all of the permission containing R_k's permission set from disMeerRoles
15: Update disMeetRoles and MeetRoles
17: **end while**
18: Add all roles within MeetRoles into InitialRoleSet
19: Return *InitialRoleSet*

The main idea behind sorting the role sets based on the descending order by the number of permissions is: when the algorithm processes the subset and intersection from role state later, new roles having the larger number permission will generate firstly by algorithm, which will cause faster speed of algorithm convergence. Furthermore, new generated roles should be added to the role set to be processed to form new roles, but current pending roles has been sorted according to the above sort method, so algorithm just needs to insert current new roles directly into where it needs to be based on straight insertion sort, it isn't required to sort the whole role set once again.

Security Comm. Networks 2010; **00**:1–11 © 2010 John Wiley & Sons, Ltd.
DOI: 10.1002/sec
Prepared using secauth.cls

This can reduce the time complexity of the algorithm from $O(n \times logn)$ to $O(n)$ (where n is the number or roles).

4.4. Role State Generation Algorithm

There are two disadvantages of the initial role set generated by the algorithm. First, there is no hierarchy for the initial role set, hence the current role state can't reflect the actual situation of the enterprise. Second, the weighted structural complexity can't reach the minimum value. Hence, we will use role state generation algorithm to minimize the weighted structural complexity in order to achieve integrated role state.

In fact, role can be seen as a set of permissions. The mainly four kinds of relationships between the permission sets are quality, subset, intersection and irrelevant. This is just the idea of *Go* algorithm proposed by Zhang et al. However, in the *Go* algorithm there are three disadvantages. First, a new role is created when there exists subset or intersection between two roles, for example, in Fig.1, new role r_3 $(ass\_perms(r_3) = \{p_2, p_3, p_4, p_5\})$ is generated, however, this new role can't be assigned to any user because there is no role with permissions $\{p_1\}$ or $\{p_6\}$, hence the new role is redundant. Second, in *GO* algorithm it only considers how to minimize the number of edges, that is, minimizing the cost of user role assignments, role permission assignments and role role hierarchy relationships maintenance, it doesn't consider others cost such as the number of roles and the cost of directly authorization about permission to user. Third, *GO* algorithm just simply uses whether it can reduce the edges as evaluation factor when it processes subset and intersection set, but does't consider the similarity information between two roles. In order to overcome the above disadvantages, our role state generation algorithm is also divided into four situations as follows:

1) If two roles have the same permissions, we can remove a role from the role set so that only one of the two roles is left and its user set is sum of the two roles' user set. In this situation, it can reduce the user role assignment, role permission assignment and role number because one role is deleted, that is, it reduces the weighted structural complexity. Here we don't need to calculate similarity degree, the only thing we need to take into consider is whether the remain role's user count exceeds the user cardinality constraint. If true, both of the current roles can't be processed as above, if not, above processing can be taken on the two roles. Example of such situation is shown in Fig.2, from which we can see that permission role assignment is reduced by 3 and the number of roles is reduced by 1.

2) If one role's permissions are subset of another, an edge is created from the role with super-set permission set to the role with sub-set permission set and the sub-set permission set is removed from the super-set permission set. Above operation occurs if and only if global optimal function value reduces after operation. Such example is shown in Fig.3. In this situation, any operation can't violate

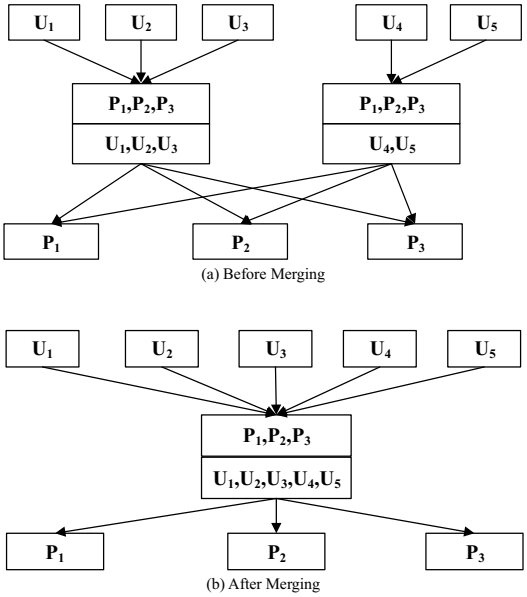

Figure 2. Processing two roles with same permissions

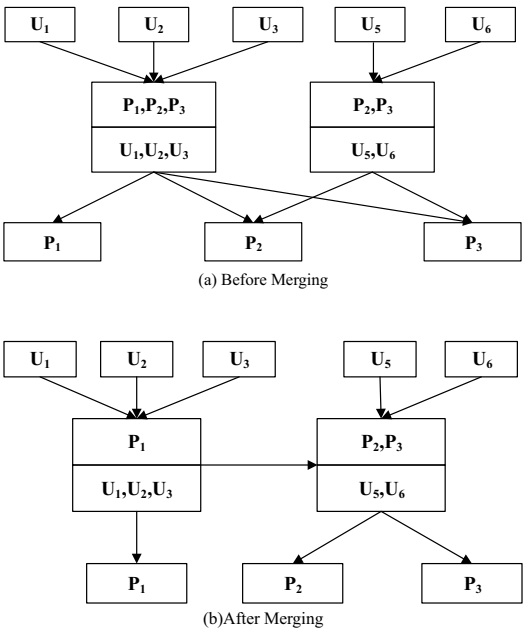

Figure 3. Processing one role's permissions are subset of another role's

the user cardinality constraint because there is nothing to do with the users.

3) If two roles have common permission set (intersection), a role containing common permission set is created whose users are a union set of two parent roles' users. However, the user of this role won't explicitly be written in the algorithm because of the down-inheritance of the user set. Hence, two edges from parent roles to new

role are created to show the hierarchical relation between roles. Finally, remove the common permission set from the two parent roles. The processing of this situation occurs if and only if global objective function value decreases after processing. Such example is shown in Fig.4. In this situation, any operation can't violate the user cardinality constraint because there is nothing to do with the users.

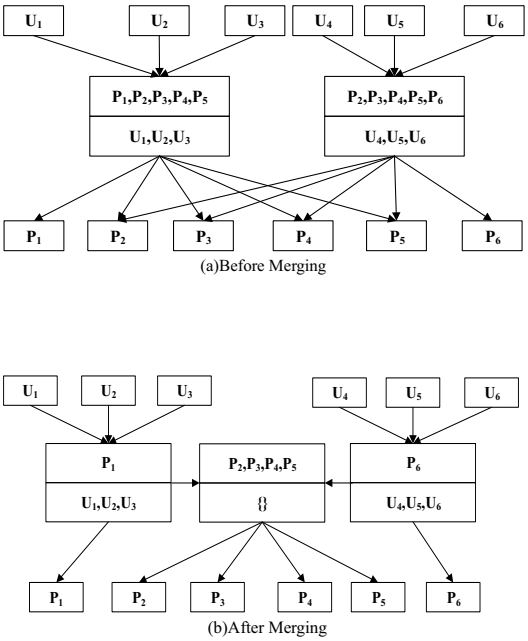

Figure 4. Processing two roles with common permission set

4) If two roles have no relation (irrelevant), in this situation, nothing is done because this will neither bring the reduction of weighted structure complexity value nor the similarity value.

In these operations, we don't consider the permission cardinality constraint because each operation won't increase the original role's permission count. For example, in the first situation only one role is removed by the algorithm which can't change the role's permission count, so that it won't violate permission cardinality constraint. In the second situation, all of the child role's permissions are removed from the parent role's permission set, and it doesn't change all of the permissions of the child role. So all operations on subset will only reduce the number of role's permission. In the third situation, although new roles are generated, their permissions are the common part of the two roles. New role will meet with the permission cardinality constraint when the parent role satisfies the permission cardinality constraint. Furthermore, all the permissions of the new roles are removed from the parent roles which will cause the parent role's permission count to decrease. Hence it won't violate permission cardinality constraint in this situation. Algorithm 3 describes the details.

Algorithm 3 Role State Generation Algorithm

Input: Initial Role Set (InitialRoleSet)
Output: Final Role Set (Roles)
1: **While** there can be more combination **do**
2: Call role selection algorithm to process R_i and R_j from InitialRoleSet
3: **if** ($R_i's$ Permissions equal to R_j) **then**
4: Deal with the equal condition
5: **end if**
6: Calculate the current global optimal function GOF1
7: **if** ($R_i's$ permissions is subset or superset of R_j) **then**
8: Deal with sub-set condition
9: Calculate global optimal function GOF2
10: **if** (GOF2>GOF1) **then**
11: Revoke subset operations
12: **end if**
13: **end if**
14: Calculate current global optimal function GOF1
15: **if** (R_i and R_j have common permission set) **then**
17: Deal with the intersection condition
18: Calculate global optimal function GOF2
19: **if** (GOF2> GOF1) **then**
20: Revoke intersection operations
21: **end if**
22: **end if**
23: **end while**
24: Return *Roles*

5. EXPERIMENTAL RESULTS

5.1. Experimental Data Generation

In the experiment, permission cardinality constraint is a random number between $minPermisValue$ (exclusive) and $|P|$. Similarly, user cardinality constraint is a random number between $minUserValue$ (exclusive) and $|U|$. Where $minPermisValue$ and $minUserValue$ are defined as follows:

Definition 17
The expression of the minimum number of permissions all users have in the user permission assignment relationships is defined as: $minPermsValue = Min|ass\_perms(u_i)|(u_i \in U)$.

Definition 18
The expression of the minimum number of permissions belongs to users in the user permission assignment relationships is defined as: $minUserValue = Min|ass\_users(p_i)|(p_i \in P)$.

The reason for the random number is because the permission cardinality constraint generated in this way is $minPermsValue + 1$ at least. Hence in the original role algorithm generating process, there exists at least one user whose permission number meets permission cardinality constrain. Thus the original role generation algorithm can

Security Comm. Networks 2010; **00**:1–11 © 2010 John Wiley & Sons, Ltd.
DOI: 10.1002/sec
Prepared using secauth.cls

be continued. Of course, all of the following users still may not meet permission cardinality constraint after removing permission set of the minimum user. In this situation, any selected part mentioned in the original role generation algorithm can be adopted to generate original roles. But experiment proves that such situation occurs rarely, which at least doesn't exist in our two experiment data set. The algorithm will run on two different actual data sets. Table 1 gives the details of the data.

Table I. The Parameters of the Experimental Data Set

DataSet	$\|U\|$	$\|P\|$	$\|UPA\|$	Density
Healthcare	46	46	1486	70%
EMEA	35	3046	7220	6.8%

5.2. Experimental Evaluation

Fig.5(a) demonstrates the relationships between permission cardinality constraint and number of roles based on Healthcare data set, and Fig.5(b) exhibits the relationships between permission cardinality constraint and WSC based on Healthcare data set, in which *CRM* algorithm is proposed by R.Kumar et al. in [13]. In order to decrease the interference from other factors, user cardinality constraint is masked when considering the influence of permission cardinality constraint, similarly, permission cardinality constraint is also masked when considering user cardinality constraint in the following figures. From Fig.5(a) and Fig.5(b), it is easy to see that our algorithm greatly reduces the WSC value of the role state and the cost of role assignment although our algorithm very slightly increases the number of roles.

Fig.6(a) demonstrates the transformation relations between user cardinality constraint and number of roles on Healthcare data set and Fig.6(b) exhibits the relations between user cardinality constraint and WSC on the same set, where algorithm 1 and 2 is proposed by Manisha Hingankar et al. in [15]. From Fig.6(a) and Fig.6(b), it can be seen that the relations between user cardinality constraint and WSC is the same as that in Fig.5. That is the number of roles increases a little more, but the value of WSC, or the cost of role assignment decreases greatly because role state is optimized by algorithm in accordance to the global structure complexity value based on the original role state.

Fig.7(a) demonstrates the transformation relations between permission cardinality constraint and the number of roles on EMEA data set and Fig.7(b) exhibits the relations between permission cardinality constraint and WSC on the same set. Similar to Healthcare data set, user cardinality constraint is set as the number of all users in system, in other words, the effect of user cardinality constraint is shielded. Fig.7(a) shows that the changing trend of permission cardinality constraint and roles number on EMEA is the same as that on Healthcare set. But

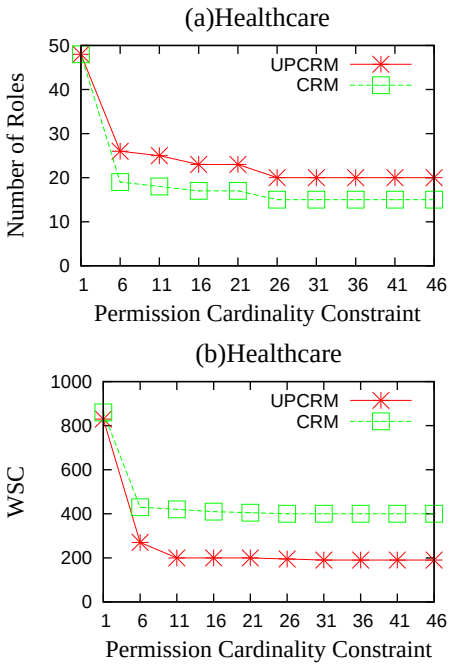

Figure 5. Performance Comparsion under Permission Cardinality Constraint in Healthcare

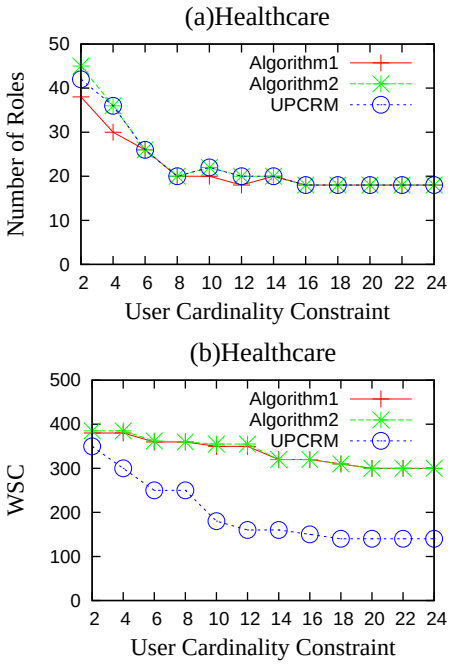

Figure 6. Performance Comparison under User Cardinality Constraint in Healthcare

Fig.7(b) indicates that WSC is not the same as the trend on Healthcare set. It becomes smaller firstly but then lager.

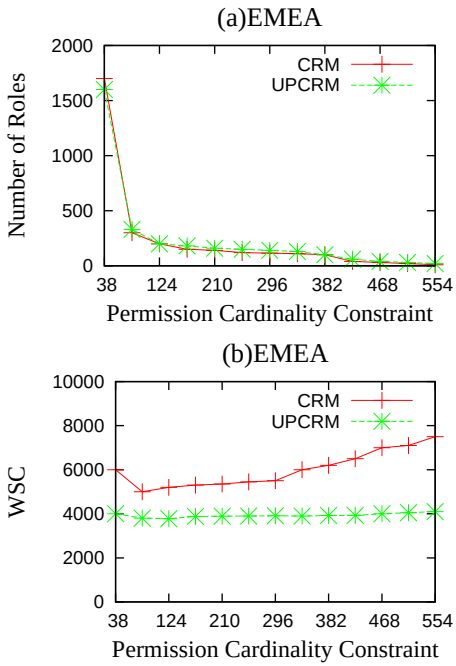

Figure 7. Performance Comparison unber Permission Cardinality Constraint in EMEA

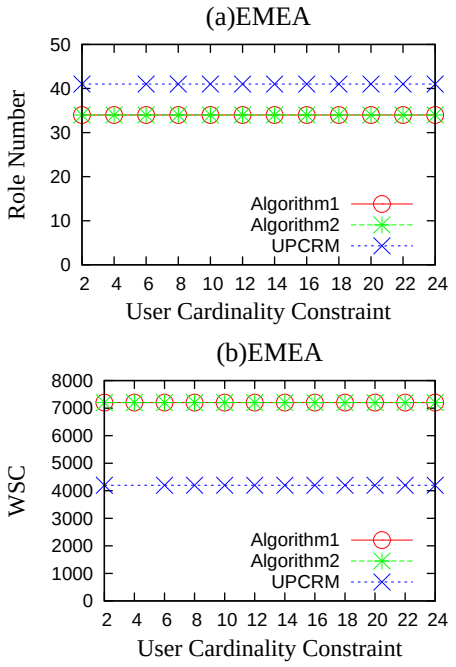

Figure 8. Performance Comparison unber User Cardinality Constraint in EMEA

Fig.8(a) demonstrates the transformation relations between user cardinality constraint and the number of roles on EMEA data set and Fig.8(b) shows the relations between user cardinality constraint and WSC on the same set. From Fig.8(a) and Fig.8(b), we also can see that algorithm 1 and 2 produce the same roles number and WSC, but our algorithm reduces the WSC.

6. CONCLUSION

In this paper, we consider the permission cardinality constraint and user cardinality constraint in role mining algorithm. We first provide a clear definition of the problem and demonstrate the process of algorithm. Then we propose a role generating algorithm that meets permission cardinality constraint and user cardinality constraint at the same time. Furthermore, we adopt the global function to generate the final role state in order to lower the assignment cost and make the final role state to be hierarchical. Finally, we verify the validity and high efficiency of the algorithm through two actual data sets.

7. ACKNOWLEDGEMENTS

This work is supported by the Joint Fund for National Natural Science Foundation of China and Henan Province for Fostering Talents under grants U1304619, National Natural Science Foundation of China under grants 61173170, 61300222, 61433006 and Innovation Fund of Nanyang Normal University under grants ZX2013013. We thank the anonymous reviewers for their helpful comments.

REFERENCES

1. David F.Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security* 2001;4(3):224-274.
2. Edward J.Coyne. Role engineering. *In Proceedings of the 1st ACM Workshop on Role-Based Access Control*,1996.
3. E.B.Fernadez, J.C.Hawkins. Determing role rights from use cases. *In Proceedings of the 2th ACM Workshop on Role-based Access Control*,1997;121-125.
4. Anne Baumgrass, Mark Strembeck and Stefanie Rinderle-Ma. Deriving Role Engineering Artifacts from Business Processes and Scenario Models. *In Proceedings of the 16th ACM Symposium on Access Control Models and Technologies (SACMAT)*, New York USA:ACM, 2011; 11-20.
5. Xiaopu Ma, Ruixuan Li and Zhengding Lu. Role mining based on weights. *In Proceedings of the 15th ACM Symposium on Access Control Models and Technologies,Pittsburgh*, PA, USA, 2010;65-74.

6. Ian Molloy, Hong Chen, Tiancheng Li, Qihua Wang, Ninghui Li, Elisa Bertino, Seraphin Calo and Jorge Lobo. Mining roles with semantic meanings. *In Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, Colorado, USA, 2008; 21-30.
7. Haibing Lu, Jaideep Vaidya and Vijayalakshmi Atluri. Optimal boolean matrix decomposition: application to role engineering. *In Proceedings of the 24th IEEE International Conference on Data Engineering (ICDE)*, Washington USA: IEEE Computer Society, 2008; 297-306.
8. Jaideep Vaidya, Vijayalakshmi Atluri and Janice Warner. Roleminer:mining roles using subset enumeration. *In Proceedings of the 13th ACM Conference on Computer and Communications Security*, Alexandria, USA, 2006; 144-153.
9. Dana Zhang, Kotagiri Ramamohanarao, Steven Versteeg and Rui Zhang. Graph Based Strategies to Role Engineering. *In Proceedings of the 6th Annual Workshop on Cyber Se-curity and Information Intelligence Research*, New York USA:ACM, 2010; 1-4.
10. Dana Zhang, Kotagiri Ramamohanarao, Tim Ebringer and Trevor Yann. Permission set mining: discovering practical and useful roles. *In Proceedings of the 2008 Annual Computer Security Applications Conference (ACSAC)*, Washington USA: IEEE Computer Society, 2008; 247-256.
11. Xiaopu Ma, Ruixuan Li, Zhengding Lu and Wei Wang. Mining Constraints in role-based access control. *Mathematical and Computer Modelling* 2012; 55(1-2): 87-96.
12. Ravi S.Sandhu, Edward J.Coyne, Hal L.Feinstein and Charles E.Youman. Role-based Access Control Models. *IEEE Computer* 1996; 29(2): 38-47.
13. Ravi Kumar, Shamik Sural and Arobinada Gupta. Mining RBAC Roles under Cardinality Constraint. *In Proceedings of the 6th International Conference on Information Systems Security*, Berlin, Heidelberg, 2010; 171-185.
14. Carlo Blundo, Stelvio Cimato. Constrained Role Mining, *Security and Trust Management* 2013;289-304.
15. Manisha Hingankar, Shamik Sural. Towards Role Mining with Restricted User-Role Assignment. *In Proceedings of the 2th Wireless VITAE Conference*, 2011; 1-5.
16. John C. John, Shamik Sural, Vijayalakshmi Atluri and Jaideep S.Vaidya. Role Mining under Role-Usage Cardinality Constraint. *In Proceedings of the 27th Conference on Information Security and Privacy*, 2012; 150-161.
17. Pullamsetty Harika, Marreddy Negajyothi, John C. John, Shamik Sural, Jaideep Vaidya and Vijayalakshmi Atluri. Meeting Cardinality Constraints in Role Mining. *IEEE Transaction on Dependable and Secure Computing*, DOI 10.1109/TDSC. 2014. 2309117.
18. Haibing Lu, Jaideep Vaidya, Vijayalakshmi Atluri and Yuan Hong. Constraint-Aware Role Mining Via Extended Boolean Matrix Decomposition. *IEEE TDSC*, 2012; 655-669.
19. Dana Zhang, Kotagiri Ramamohanarao and Tim Ebringer. Role engineering using graph optimisation. *In Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, Antipolis, France, 2007; 139-144.
20. Mario Frank, Joachim M. Buhmann and David Basin. On the Definition of Role Mining. *In Proceedings of the 15th ACM Symposium on Access Control Models and Technologies,Pittsburgh*, PA, USA, 2010; 35-44.