Mathematical and Computer Modelling 55 (2012) 87–96

Contents lists available at SciVerse ScienceDirect

Mathematical and Computer Modelling

journal homepage: www.elsevier.com/locate/mcm

Mining constraints in role-based access control

Xiaopu Ma [a,b], Ruixuan Li [a,*], Zhengding Lu [a], Wei Wang [a]

[a] School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, PR China

[b] School of Computer and Information Technology, Nanyang Normal University, Nanyang 473061, PR China

ARTICLE INFO

Article history:
Received 24 September 2010
Received in revised form 14 January 2011
Accepted 29 January 2011

Keywords:
Role-based access control (RBAC)
Role engineering
Constraints
Constraint mining

ABSTRACT

Constraints are an important aspect of role-based access control (RBAC) and sometimes argued to be the principal motivation of RBAC. While role engineering is proposed to define an architectural structure of the organization's security policies, none of the work has employed constraint mining in migrating a non-RBAC system to an RBAC system to our knowledge, thus providing the motivation for this work. In this paper, we first define a wide variety of constraints, which are the best-known ones to date, and then create a relationship between the conventional data mining technology and the constraints. We further propose an anti-association rule mining algorithm to generate the constraints. Experiments on performance study prove the superiority of the new algorithm.

1. Introduction

Role-based access control (RBAC) [1] is the most popular access control model at present that has been widely deployed in enterprise security management products. In this security model, a set of permissions are assigned to users through roles. This change on how to assign the permissions often reduces the complexity of access control because the number of users is generally much larger than that of roles in an organization [2]. Furthermore, it can support three well-known security principles: least privilege, separation of duties and data abstraction. As a result, RBAC has been implemented successfully in a variety of commercial systems, such as insurance company [3] and bank [4], and has become the norm in many applications. Hence how to create a comprehensive framework for defining the architectural structure of RBAC has become a challenging task before all the benefits of RBAC can be realized [5]. As a solution to facilitate the process to migrate a non-RBAC system to an RBAC system, *role engineering* is introduced [6].

Essentially, there are two basic approaches towards role engineering: the *top-down* and the *bottom-up*. Under the top-down approach, roles are derived by carefully analyzing particular business functions and then assigning the needed permissions to create roles for these business functions [7]. However, this approach is time consuming and costly while this approach can well reflect the functional requirements of the organization. Under the bottom-up approach, roles can be aggregated through permissions automatically from the existing user-permission assignments before RBAC is implemented. Hence, this approach is likely to ignore the business functions of the organization but can generate the architectural structure of RBAC automatically [8].

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 generate constraints. In other words, most of the existing role engineering approaches just captured the organization's business rules, as these relate to access control, and reflect these rules in defining, naming, structuring, and constructing a valid set of roles. There are no approaches to mine constraints from the existing

* Corresponding author.
E-mail addresses: xpma@smail.hust.edu.cn (X. Ma), rxli@hust.edu.cn (R. Li), zdlu@hust.edu.cn (Z. Lu), ybwei.wang@gmail.com (W. Wang).

0895-7177/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mcm.2011.01.053

user-permission assignments while constraints are an important aspect of RBAC. A common example is that of *mutually exclusive roles*, such as purchasing manager and account payable manager. In most organizations, the same individual will not be permitted to be a member of both roles, because this creates a possibility for committing fraud. In another case, there is only one person in the role of chairman in a department. Similarly, the number of roles to which an individual user can belong could also be limited. We call this *cardinality constraint* [1].

In this paper, we first define a set of constraints, which include the best-known ones, and then show the relationship of these constraints with the problem already identified in the data analysis literature. By showing that these constraints are in essence reducible to the known problem, we can directly borrow the existing implementation solutions and guide further research in this direction. Finally, we design an anti-association rule mining algorithm to mine these constraints. Experiments on performance study are also given.

The remainder of this paper is organized as follows. We discuss related work in Section 2. The limitations in existing application of constructing RBAC systems drive our motivation and Section 3 defines a variety of constraints. Furthermore, we design a new algorithm based on the traditional algorithm *Apriori* to find constraints which scan the user-permission assignments only once. A summary of our experimental results is discussed in Section 4. Finally, Section 5 provides some insight into our ongoing and future work.

2. Related work

Role engineering is introduced in 1995, which aims to define an architectural structure that is complete, correct and efficient to specify the organization's security policies and business functions [6]. It leads to the initial top-down process oriented approaches for role definition. With the top-down approach, people start from requirements and successively refine the definitions to result in permissions and roles respectively based on the business functions that decompose from the business processes. Since there are often dozens of business processes and ten thousands of users, it makes top-down approach impracticable in medium to large size enterprises. As a result, the researchers have changed the focus to the bottom-up approach that utilizes the existing user-permission assignments to formulate roles. In the bottom-up, especially the application of data mining techniques for role extraction and definition has raised interests in the research community [8]. In [9], Schlegelmilch et al. propose an algorithm *ORCA* to build a hierarchy of permission clusters using role mining. However, the algorithm randomly selects a pair when more than one pair has the maximum overlap and permission can only be found along one role path of the hierarchy. Hence, Vaidya et al. propose a subset enumeration approach *RoleMiner* to overcome the above limitation [10].

An inherent problem with all of the above approaches is that there is no formal notion for *goodness* of a role. Molloy et al. [11] develop a hierarchical miner to discover good roles based on formal semantic analysis. Zhang et al. [12] present an algorithm to reduce the administration required for role related assignments through reducing the number of role-to-user and permission-to-role assignments. Ene et al. [13] use heuristics and graph theory to reduce the graph representations to find the smallest collection of roles that can be used to implement a pre-existing user-permission assignments. Frank et al. [14] provide a probabilistic model to analyze the relevance of different kinds of business information for defining roles that can both explain the given user-permission assignments and describe the meaning from the business perspective. Takabi et al. [15] define a measure for goodness of an RBAC state and use similarity-based approach for optimal mining of role hierarchy. Vaidya et al. [16] and Lu et al. [17] theoretically analyze the role mining problem and also give two different variations of the role mining problem, call the δ-approx role mining problem and the minimal noise role mining problem that have pragmatic implications. They also show the problem of finding the minimal set of descriptive roles and relationships that equal to the user-permission assignments is NP-complete problem.

However, the traditional role engineering approaches only identify comprehensive roles and place them into a hierarchy while constraints in RBAC are very important. It may result in some problems if we are no idea about constraints in RBAC system.

1. Constraints are a set of imposed rules on RBAC, and they are an important aspect of RBAC. In other words, constraints are one of the most distinctive and important feature of the RBAC approach. We can get an incomplete architectural structure of RBAC if there are no constraints;
2. In RBAC system, its emphasis on controlling who has access to operations on what objects is fundamentally different from information flow security in multi-level secure systems, and therefore can support three well-known security principles: least privilege, separation of duties and data abstraction. Let us assume that there is a requested permission set $RQ = \{p_1, p_2, p_3, p_4\}$ (the casual user requires permission p_1, p_2, p_3, p_4 to perform the task). Although role set $\{r_1, r_2\}$ can enforce the principle of least privilege for the requested permission set RQ (which means that r_1 and r_2 can exactly cover all of the permissions in the requested permission set RQ). However, this approach about enforcing the principle of least privilege for the requested permission set may be wrong if there is no idea about constraints in RBAC system. For example, we do not know whether r_1 and r_2 are mutually exclusive roles or not;
3. Constraints are a powerful mechanism for laying out higher-level organizational policy. They enable the chief security officer to lay out the broad scope of what is acceptable and impose this as a mandatory requirement on other security officers and users who participate in RBAC management. If there is no idea of the constraint in RBAC system, the administrator cannot do this.

To this aim, this research tries to mine constraints in a feasible way. We first propose a similarity way to represent these constraints like the traditional association rule mining algorithm *Apriori*. However, the key problem with the traditional algorithm *Apriori* is its computational complexity. It needs to scan the database many times. This is quite infeasible, except for very small database. Hence we present a new constraint mining algorithm *Anti-apriori* to address the above problem which just scans the database only once that makes it feasible for even large database. The experimental results are tested to show the effectiveness of our findings.

3. Constraint mining

3.1. Preliminaries

We develop the material in this paper in the context of the NIST standard, the most widely known RBAC model. It consists of RBAC0, RBAC1, RBAC2 and RBAC3. The last two incorporate separation of duty constraint. For the sake of simplicity, we do not consider sessions in this paper.

Definition 1 (*RBAC Model*)**.** The RBAC model has the following components:

- U, R, P, users, roles and permissions respectively;
- $PA \subseteq P \times R$, a many-to-many mapping of permission-to-role assignments;
- $UA \subseteq U \times R$, a many-to-many user to role assignment relationships;
- $auth\_perms(R) = \{p \in P | (p, R) \in PA\}$, the mapping of role R onto a set of permissions.

We can use an $m \times n$ binary matrix M to describe the relationships between users and permissions before RBAC is implemented. Where m is the number of users and n is the number of permissions. The element $M\{i, j\} = 1$ denotes that the ith user has the jth permission or the jth permission belongs to the ith user. We use u_i $(i = 1, \ldots, m)$ to indicate the ith user and $user\_perms(u_i)$ $(i = 1, \ldots, m)$ to indicate the set of permissions assigned to the ith user, p_j $(j = 1, \ldots, n)$ to indicate the jth permission and $perm\_users(p_j)$ $(j = 1, \ldots, n)$ to indicate the set of users that possess permission p_j.

When RBAC system has been constructed through role engineering, we can use an $m \times k$ binary matrix N to describe the relationships between users and roles. Where m is the number of users and k $(k \leq n)$ is the number of roles. The element $N\{i, j\} = 1$ denotes that the ith user has the jth role or the jth role belongs to the ith user. We use u_i $(i = 1, \ldots, m)$ to indicate the ith user and $user\_roles(u_i)$ $(i = 1, \ldots, m)$ to indicate the set of roles assigned to the ith user, r_j $(j = 1, \ldots, k)$ to indicate the jth role and $role\_users(r_j)$ $(j = 1, \ldots, k)$ to indicate the set of users that possess role r_j. Hence, we can define a variety of constraints as follows.

Definition 2 (*Mutually Exclusive Roles*)**.** Given a set of roles $rs \subseteq R$, we say $r_i \in rs$ and $r_j \in rs$ $(i \neq j)$ are mutually exclusive roles if $role\_users(r_i) \cap role\_users(r_j) = \emptyset$. We will use the notation $\overline{r_i \wedge r_j}$ to specify that role r_i and r_j are mutually exclusive roles.

This constraint in terms of *UA* specifies that one individual cannot be a member of both mutually exclusion roles. In other words, the same user can be assigned to at most one role in a mutually exclusive role set. For example, the user cannot be a member of both accounts-manager role and purchasing-manager role. More general, we allow have more than one role in the notation. For example, $\overline{r_1 r_2 \wedge r_3 r_4}$ describes that the user cannot have both the role set $\{r_1, r_2\}$ and $\{r_3, r_4\}$. Analogously, we give the definition of mutually exclusive permissions.

Definition 3 (*Mutually Exclusive Permissions*)**.** Given a set of permissions $ps \subseteq P$, we say $p_i \in ps$ and $p_j \in ps$ are mutually exclusive permissions if $p_i \in auth\_perms(r_i)$, and $p_j \notin auth\_perms(r_i)$ for all $r_i \in R$. We will use the notation $\overline{p_i \wedge p_j}$ to specify that permission p_i and p_j are mutually exclusive permissions.

This constraint in terms of *PA* specifies that the mutually exclusive permissions cannot be assigned to the same role. For example, the permission to issue checks and the permission to audit the operation should not be assigned to the same role. More general, we allow have more than one permission in the notation. For example, $\overline{p_1 p_2 \wedge p_3 p_4}$ describes that the role cannot have both the permission set $\{p_1, p_2\}$ and $\{p_3, p_4\}$ or $\{p_1, p_2\}$ and $\{p_3, p_4\}$ cannot be given to the same role.

Definition 4 (*Cardinality Constraint of Role*)**.** The cardinality constraint of role is defined as the maximum number of roles to which an individual user can belong. We will use the notation $\max(r)$ to specify this constraint.

In RBAC, the principle of least privilege is one of the most important principles in the design of protection mechanisms for secure computer systems. Whenever possible, a user should be given no more access to resources than is required to complete the task at hand. In other words, the computer system should be able to determine the minimum set of privileges required for the user to perform the task and guarantee that the user is only granted those privileges and no more. Let us assume that the requested permission set $RQ = \{p_1, p_2, p_3, p_4, p_5, p_6\}$, $auth\_perms(r_1) = \{p_1, p_2, p_3\}$, $auth\_perms(r_2) = \{p_3, p_4\}$ and $auth\_perms(r_3) = \{p_5, p_6\}$. In this situation, role set $\{r_1, r_2, r_3\}$ can enforce the principle of least privilege for the requested permission set RQ if there is no constraint on the maximum number of roles which an individual user can belong.

If each user has not more than two roles, in this situation there is no role set can satisfy the principle of least privilege for the requested permission set. Hence, this constraint is very important in RBAC systems, especially for enforcing the principle of least privilege for the requested permission set. Similarly, we can define the cardinality constraint of user and permission as follow.

Definition 5 (*Cardinality Constraint of User*)**.** The cardinality constraint of user is defined as the maximum number of users to which a role can have. For example, there is only one person in the role of chairman of a department. We will use the notation $\max(u)$ to specify this constraint.

Definition 6 (*Cardinality Constraint of Permission*)**.** The cardinality constraint of permission is defined as the maximum number of roles to which a permission can belong. We will use the notation $\max(p)$ to specify this constraint.

In data mining field, mining of association rules has become a fundamental problem and it has been studied extensively. Many efficient algorithms such as *FP*-growth, *Apriori* and *Eclat* have been developed to solve this problem on databases containing transactions [18]. The following is a formal statement of the problem.

Let D be a set of transactions, where each transaction T is an itemset that $T \subseteq I$ (A set of items $X \subseteq I$ is called an itemset, where $I = \{i_1, i_2, \ldots, i_n\}$ is a set of attributes over the binary domain $\{0, 1\}$). A tuple T of the database D is represented by identifying the attributes with value 1 where associated with each transaction is a unique identifier. We say that a transaction T contains an itemset X, if $X \subset T$. An association rule is an implication of the form $X \Rightarrow Y$, where $X \subset T$, $Y \subset T$, and $X \cap Y = \emptyset$. The rule $X \Rightarrow Y$ holds in the transaction set D with *confidence* c if $c\%$ of transactions in D that contain X also contain Y. The rule $X \Rightarrow Y$ has *support* s in the transaction set D if $s\%$ of transactions in D contain $X \cup Y$. The problem of mining association rules is to generate all association rules that have certain user-specified minimum support (called *minsup*) and minimum confidence (called *minconf*).

Before RBAC model is implemented, there are only user-permission assignments in the system, in this situation we regard permission as attribute, the collection of all user-permission assignments as a set of transactions and each user-permission assignment as a transaction. Then we can define an anti-association rule between permissions as follows.

Definition 7 (*Anti-Association Rule between Permissions*)**.** An anti-association rule between permissions is an implication of the form $\overline{X \Rightarrow Y}$, where $X \subset P, Y \subset P, X \cap Y = \emptyset$ and $user\_perms(u_i) \cap X = \emptyset \vee user\_perms(u_i) \cap Y = \emptyset$ for each $u_i \in U$ $(i = 1, \ldots, m)$.

In reality, access control configurations in any large organization are noisy [19]. Hence, we relax the restriction that the anti-association rule $\overline{X \Rightarrow Y}$ holds in the user-permission assignments if the rule has certain user-specified minimum support and confidence. There are three cases to compute the support and confidence of an anti-association rule between permissions as follows.

(1) $X \Rightarrow \overline{Y}$: In this situation, we denote the *confidence* as the ratio of the $numUsers(X\overline{Y})$ to the $numUsers(X)$ (where $numUsers(X\overline{Y})$ is the number of users which possess permission set X and do not contain permission set Y at the same time that presented in the user-permission assignments, $numUsers(X)$ is the number of users which possess the permission set X in the user-permission assignments), the *support* as the ratio of the $numUsers(X\overline{Y})$ to the $numUsers(All)$ (where $numUsers(All)$ is the total number of users in the user-permission assignments);
(2) $\overline{X} \Rightarrow Y$: In this situation, we denote the *confidence* as the ratio of the $numUsers(\overline{X}Y)$ to the $numUsers(\overline{X})$ (where $numUsers(\overline{X}Y)$ is the number of users which possess permission set Y and do not contain permission set X at the same time that presented in the user-permission assignments, $numUsers(\overline{X})$ is the number of users which do not possess the permission set X in the user-permission assignments), the *support* as the ratio of the $numUsers(\overline{X}Y)$ to the $numUsers(All)$;
(3) $\overline{X} \Rightarrow \overline{Y}$: In this situation, we denote the *confidence* as the ratio of the $numUsers(\overline{X}\,\overline{Y})$ to the $numUsers(\overline{X})$ (where $numUsers(\overline{X}\,\overline{Y})$ is the number of users which do not possess permission set X and Y that presented in the user-permission assignments), the *support* as the ratio of the $numUsers(\overline{X}\,\overline{Y})$ to the $numUsers(All)$.

Similarity, when RBAC system has been constructed through role engineering, we regard each role as attribute, collection of all user-role relationships as transactions and each user-role assignment as a transaction. Then we can define an anti-association rule between roles as follows.

Definition 8 (*Anti-Association Rule between Roles*)**.** An anti-association rule between roles is an implication of the form $\overline{X \Rightarrow Y}$, where $X \subset R$, $Y \subset R, X \cap Y = \emptyset$ and $user\_roles(u_i) \cap X = \emptyset \vee user\_roles(u_i) \cap Y = \emptyset$ for each $u_i \in U$ $(i = 1, \ldots, m)$. The anti-association rule $\overline{X \Rightarrow Y}$ holds in the user-role assignments if the rule has certain user-specified minimum support and confidence.

3.2. Algorithm

In RBAC system, the constraints should be enforced. In other words, we need generate a set of constraints when we migrate a no-RBAC system to an RBAC system. In this subsection, we will use the algorithm for mining association rules to solve this problem. We now briefly present the *Apriori* algorithm as follows.

The process of *Apriori* algorithm can be decomposed into two sub-processes:

(1) Scan the database to find all combinations of items whose *support* is greater than *minsup*. Call those combinations frequent itemsets. If the itemsets has k items, we call it k-frequent itemsets. In order to count the support of each itemsets, the algorithm needs to scan the database once;
(2) Generate the desired rules. The general idea is that if, say, *ABCD* and *AB* are frequent itemsets, then we can determine if the rule $AB \Rightarrow CD$ holds by computing the ratio $r = support(ABCD)/support(AB)$. Only if $r \geq minconf$, then the rule holds.

Now we design an anti-association rule mining algorithm to generate the constraints as follows (Here we just consider how to generate mutually exclusive permissions from user-permission assignments because the same algorithm also can generate mutually exclusive roles from user-role assignments. We also do not take into account other cardinality constraints for the sake of simplicity). We split the anti-association rule mining algorithm between permissions into three phases:

Step1. Scan the user-permission assignments matrix M to find all combinations of permissions whose *support* is greater than *minsup*. We denote the set of all frequent permission set as F;
Step2. Set each element in $M\{i, j\}$ $(i = 1, \ldots, m, j = 1, \ldots, n)$ as $\overline{M\{i, j\}}$ in the user-permission assignments, we denote the binary matrix as $\overline{M}$. Then scan it to find all combinations of permissions whose *support* is greater than *minsup*. We denote the set of all frequent permission set as $\overline{F}$;
Step3. Generate rules from the frequent permissions found in Steps 1 and 2 using the method like the *Apriori*.

Algorithm 1 gives the detailed steps about how to generate mutually exclusive permissions. The algorithm consists of three phases. The first phase consists of lines 2–6. The *for* loop in line 2 iterates over all the permissions in the user-permission assignments and generates the 1-frequent permission set F_1 and $\overline{F}_1$. Line 3 calculates the support of p_i and $\overline{p}_i$. Line 4 inserts the 1-frequent permission p_i into F_1. Line 5 inserts the 1-frequent permission $\overline{p}_i$ into $\overline{F}_1$.

Phase 2 consists of lines 8–14. The *for* loop in line 8 iterates over the set of all $(k-1)$-frequent permission set F_{k-1} to generate the k-frequent permission set F_k. The *for* loop in line 11 iterates over the set of all $(k-1)$-frequent permission set $\overline{F}_{k-1}$ to generate the k-frequent permission set $\overline{F}_k$. The frequent permission set generation procedure returns the set of all k-frequent permission set. The procedure of *FrequentPermissionGen* consists of lines 16–22. The *for* loop in line 16 iterates over all the $(k-1)$-frequent permission set to generate the k-frequent permission set. We assume that the permissions in a permission set are ordered by the subscript. Line 18 finds the users containing $X \cup Y$. Line 19 computes the number of users containing $X \cup Y$ and then gets the support *sf*. Line 20 finds the k-frequent permission set that satisfy the minimum support. The idea behind lines 18 and 19 is based on the following theorem.

Theorem 1. *For any permission set* $PS = \{p_1, p_2, \ldots, p_k\} \subseteq 2^P$, *where* $p_i \in P$ $(i = 1, \ldots, k)$, *we have*

$$perm\_users(PS) = perm\_users(p_1) \cap \cdots \cap perm\_users(p_k)$$
$$numUsers(PS) = countNum(perm\_users(PS)).$$

The proof is trivial. We use an example to describe this theorem. For example, suppose $perm\_users(p_1) = \{u_1, u_2, u_3\}$, $perm\_users(p_2) = \{u_1, u_2, u_4\}$, which means p_1 belongs to u_1, u_2, u_3, and u_1, u_2, u_4 all have the permission p_2, we have

$$\begin{aligned} perm\_users(\{p_1, p_2\}) &= perm\_users(p_1) \cap prem\_users(p_2) \\ &= \{u_1, u_2, u_3\} \cap \{u_1, u_2, u_4\} = \{u_1, u_2\} \\ numUsers(\{p_1, p_2\}) &= countNum(perm\_users(\{p_1, p_2\})) \\ &= countNum(\{u_1, u_2\}) = 2. \end{aligned}$$

Phase 3 consists of lines 25–30. The *for* loop in line 25 iterates over the set of all frequent permission set $\overline{F}$ to generate the anti-association rules like $\overline{X} \Rightarrow \overline{Y}$. The *for* loop in line 28 iterates over the set of all frequent permission set F and $\overline{F}$ to generate the anti-association rules like $X \Rightarrow \overline{Y}$ or $\overline{X} \Rightarrow Y$. The general idea behind line 29 is that if, say, p_1p_2 is 2-frequent permission set in F_2, p_3p_4 is 2-frequent permission set in $\overline{F}_2$, then we can determine if the anti-association rule $p_1p_2 \Rightarrow \overline{p_3}\,\overline{p_4}$ holds by computing the ratio

$$r = \frac{support(p_1p_2\overline{p}_3\overline{p}_4)}{support(p_1p_2)} = \frac{countNum(perm\_users(\{p_1, p_2, \overline{p}_3, \overline{p}_4\}))}{countNum(perm\_users(\{p_1, p_2\}))},$$

only if $r \geqslant mincof$, then the rule holds.

The algorithm has two improvements. The key problem with the algorithm *Apriori* is its computational complexity. It needs to scan the user-permission assignments many times. This is quite infeasible, except for very small user-permission assignments. However, our algorithm scans the user-permission assignments only once that makes it feasible for even large data sets. Furthermore, the algorithm can generate the anti-association rules as the constraints at the same time.

Algorithm 1 *Anti-apriori* algorithm

Require: $D \equiv (M, N, UP_{M\times N}, minsup, minconf, F, \overline{F})$
Require: M, N, the number of users and permissions
Require: $UP_{M\times N}$ represents the user-permission assignments
Require: $minsup, minconf$, the minimum support and confidence
Require: $F, \overline{F}$ represents all the frequent permission set in the matrix M, $\overline{M}$
1:{Generate the frequent 1-permission set}
2: **for** $(i = 1; i \leq N; i++)$ **do**
3: $sf(p_i) = numUsers(p_i)/M, \quad sf(\overline{p}_i) = numUsers(\overline{p}_i)/M$
4: insert $p_i, sf(p_i), perm\_users(p_i)$ into F_1 if $sf(p_i) \geq minsup$
5: insert $\overline{p}_i, sf(\overline{p}_i), perm\_users(\overline{p}_i)$ into $\overline{F}_1$ if $sf(\overline{p}_i) \geq minsup$
6: **end for**
7:{Generate the frequent k-permission set and $\overline{k}$-permission set}
8: **for** $(k = 2; F_k \neq \emptyset; k{+}{+})$ **do**
9: $F_k = FrequentPermissionGen(F_{k-1}, minsup)$
10:**end for**
11:**for** $(k = 2; \overline{F}_k \neq \emptyset, k{+}{+})$ **do**
12: $\overline{F}_k = FrequentPermissionGen(\overline{F}_{k-1}, minsup)$
13:**end for**
14: $F = \cup_k F_k$, $\overline{F} = \cup_k \overline{F}_k$
15:{$FrequentPermissionGen(T_{k-1}, minsup)$}
16:**for** X and Y are in T_{k-1} **do**
17: **if** first k-2 permissions of X and Y are the same **then**
18: $perm\_user(X \cup Y) = perm\_users(X) \cap perm\_user(Y)$
19: $sf(X \cup Y) = numUsers(perm\_users(X \cup Y))/M$
20: insert $X \cup Y, sf(X \cup Y), perm\_users(X \cup Y)$ into T_k if $sf(X \cup Y) \geq minsup$
21: **end if**
22:**end for**
23:Return T_k
24:{Generate the anti-association rules}
25:**for** each Y in $\overline{F}$ **do**
26: Generate rules using the same method as *Apriori* algorithm
27:**end for**
28:**for** each X in F and Y in $\overline{F}$ **do**
29: Generate rules using the same method as *Apriori* algorithm
30:**end for**

3.3. Running example

The following example demonstrates the effectiveness of the proposed algorithm. Assume a hypothetical organization has 4 users and 5 permissions. Table 1 shows the relationship matrix with the assignment of permissions to users. Now we apply the algorithm to find the frequent permission sets and then generate the constraints as follows.

(1) Phase 1 generates the 1-frequent permission set $F_1, \overline{F}_1$
  (a) The set of users that have possessed each permission can be got as follows:
$$perm\_users(p_1) = \{u_1, u_2, u_3\}, \quad perm\_users(p_2) = \{u_1, u_2, u_3, u_4\}$$
$$perm\_users(p_3) = \{u_1, u_2, u_4\}, \quad perm\_users(p_4) = \{u_3\}$$
$$perm\_users(p_5) = \{u_2\}.$$
  (b) Then we can get the support of each permission as follows:
$$sf(p_1) = numUsers(p_1)/4 = 0.75, \quad sf(p_2) = numUsers(p_2)/4 = 1$$
$$sf(p_3) = numUsers(p_3)/4 = 0.75, \quad sf(p_4) = numUsers(p_4)/4 = 0.25$$
$$sf(p_5) = numUsers(p_5)/4 = 0.25.$$
  (c) Then we can get the 1-frequent permission set F_1 as follows (where $minsup = 0.5$):
$$F_1 = \{(p_1, u_1u_2u_3, 0.75)(p_2, u_1u_2u_3u_4, 1)(p_3, u_1u_2u_4, 0.75)\}.$$
  (d) The set of users that have not possessed each permission can be got as follows:
$$perm\_users(\overline{p}_1) = \{u_4\}, \quad perm\_users(\overline{p}_3) = \{u_3\}$$
$$perm\_users(\overline{p}_4) = \{u_1, u_2, u_4\}, \quad perm\_users(\overline{p}_5) = \{u_1, u_3, u_4\}.$$

Table 1
Sample matrix for an example organization.

	p_1	p_2	p_3	p_4	p_5
u_1	1	1	1	0	0
u_2	1	1	1	0	1
u_3	1	1	0	1	0
u_4	0	1	1	0	0

(e) Then we can get the support of each permission as follows:

$$sf(\overline{p}_1) = numUsers(\overline{p}_1)/4 = 0.25, \qquad sf(\overline{p}_3) = numUsers(\overline{p}_3)/4 = 0.25$$
$$sf(\overline{p}_4) = numUsers(\overline{p}_4)/4 = 0.75, \qquad sf(\overline{p}_5) = numUsers(\overline{p}_5)/4 = 0.75.$$

(f) Then we can get the 1-frequent permission set $\overline{F}_1$ as follows (where $minsup = 0.5$):

$$\overline{F}_1 = \{(\overline{p}_4, u_1u_2u_4, 0.75)(\overline{p}_5, u_1u_3u_4, 0.75)\}.$$

(2) Now, Phase 2 finds the frequent permission sets. The first iteration finds the 2-frequent permission set $F_2, \overline{F}_2$:

$$sf(p_1, p_2) = 0.75, \qquad sf(p_1, p_3) = 0.5, \qquad sf(p_2, p_3) = 0.75, \qquad sf(\overline{p}_4, \overline{p}_5) = 0.5.$$

Then we get the 2-frequent permission set $F_2, \overline{F}_2$ as follows:

$$F_2 = \{(p_1p_2, u_1u_2u_3, 0.75)\ (p_1p_3, u_1u_2, 0.5)(p_2p_3, u_1u_2u_4, 0.75)\}$$
$$\overline{F}_2 = \{(\overline{p}_4\overline{p}_5, u_1u_4, 0.5)\}.$$

(3) The next iteration of Phase 3 finds the 3-frequent permission set F_3:

$$F_3 = \{(p_1p_2p_3, u_1u_2, 0.5)\}.$$

(4) Phase 3 generates all the candidate anti-association rules based on Definition 7

$$p_3 \Rightarrow \overline{p}_4(confidence = 1), \qquad \overline{p}_4 \Rightarrow \overline{p}_5(confidence = 0.67).$$

If we set $minconf = 0.6$, we can get the resulting anti-association rule is $\overline{p_3 \Rightarrow p_4}$ and $\overline{p_4 \Rightarrow p_5}$ (Hence when a casual user wants to get the requested permissions p_3, p_4 to complete the task at hand, the administrator cannot allocate the permissions to the user).

4. Experimental results

In this section, we will implement the proposed algorithm of mining constraints on an Intel (R) Core (TM)D 2.2G machine with 2 GB memory to evaluate how well our algorithm performs using different metrics like in [20]. The running platform is Windows XP.

4.1. The performance of the algorithm

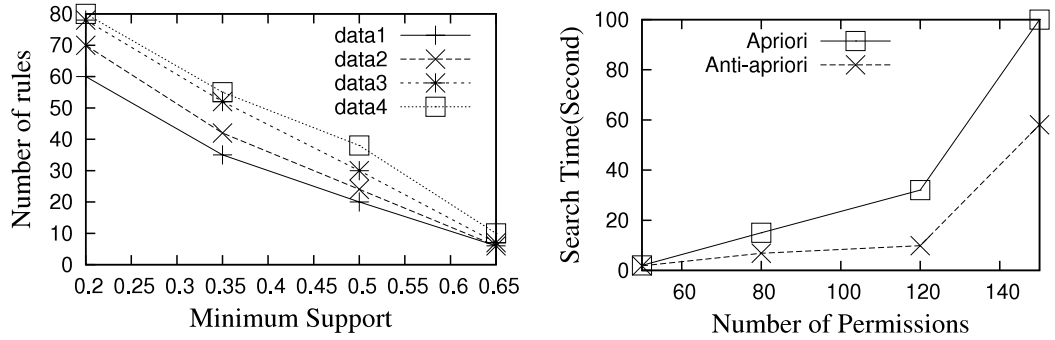
To study the performance of our algorithm, we generate the synthetic test data as follows. First, the maximum number of users and permissions are created respectively. Then we use *for* loop to create the relationships between users and permissions. For each user, a random number of permissions are chosen. The value of each element in the matrix is randomly chosen as 0, indicating that the user has no such permission, or 1, indicating that the user has such permission. Finally, we get the relationships between users and permissions.

We present the evaluation of our algorithm with *Apriori* on two different user-permission assignments. For the first set of assignments, we fix the number of users, while changing the number of permissions. Table 2 shows the test parameters. Each test is repeated ten times to evaluate the performance of our algorithm. We are interested in two things: the number of rules generated by the *Anti-apriori* algorithm, and how quickly it finds it. Fig. 1(a) shows the average number of rules found by the *Anti-Apriori* algorithm in different values of minimum support. From the figure we can see that our algorithm can generate a certain amount of rules. It means that our algorithm can reflect more the real organization's security requirements. Hence, it can decrease the risk for the security system. Fig. 1(b) shows the average search time under the different number of permissions. Here, the search time shows the feasibility of our approach, taking account much more permissions. Indeed, on the largest user-permission assignments, our algorithm also runs quite fast. This is reasonable because we scan the user-permission assignments only once to compute the support of frequent permission set.

In the second experiments, we fix the number of permissions, while changing the number of users. Table 3 shows the test parameters. Fig. 2(a) shows the number of rules under the second set of assignments for various minimum supports, while Fig. 2(b) shows the search time under the different number of users. We also can see that our algorithm costs less time and generates a certain amount of rules than *Apriori*. If the number of permissions and users in the user-permission assignments is larger, the advantage of our algorithm will be more obvious.

Table 2
Parameter settings for testing performance (Fixed users, varying permissions).

	$\|User\|$	$\|Permission\|$
data1	500	50
data2	500	80
data3	500	120
data4	500	150

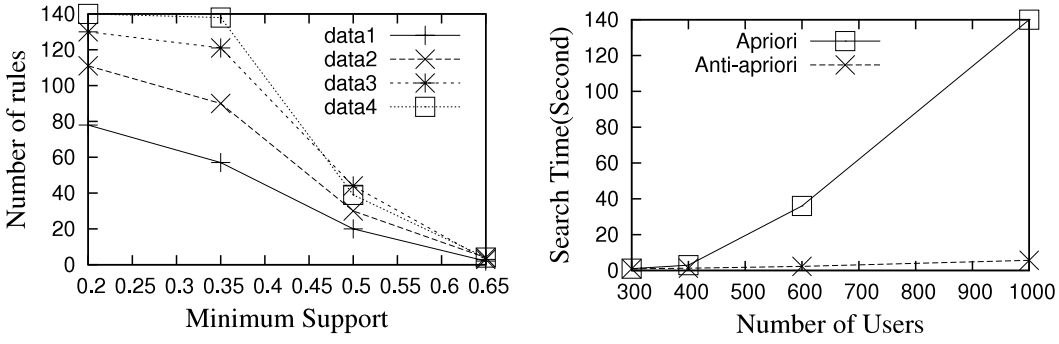


(a) Number of rules. (b) Search time ($minconf = 0.35$).

Fig. 1. Performance comparison under the fixed number of users.

Table 3
Parameter settings for testing performance (Fixed permissions, varying users).

	$\|User\|$	$\|Permission\|$
data1	300	100
data2	400	100
data3	600	100
data4	1000	100



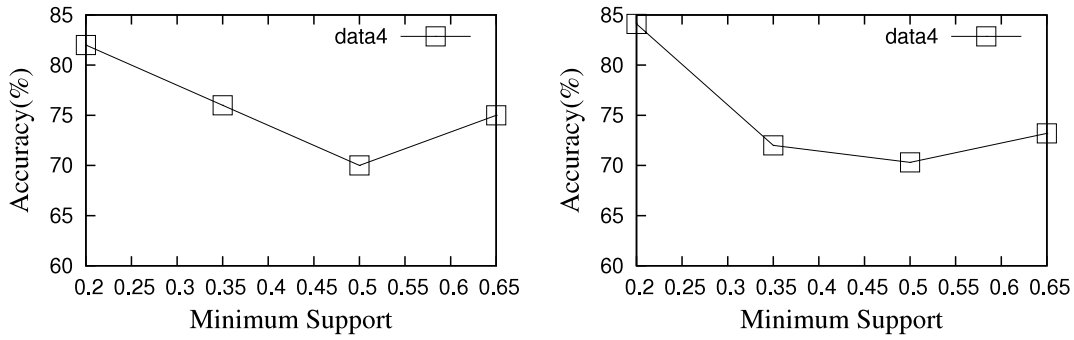
(a) Number of rules. (b) Search time ($minconf = 0.35$).

Fig. 2. Performance comparison under the fixed number of permissions.

4.2. The accuracy of the algorithm

In order to study the accuracy of our algorithm, we create another synthetic data set. It performs as follows. First, we set a set of mutually exclusive permissions, the maximum number of users and permissions respectively. Then we use *for* loop to create the relationships between users and permissions. For each user, a random number of permissions are chosen. The value of each element in the matrix is randomly chosen as 0, indicating that the user has no such permission, or 1, indicating that the user has such permission. If the user has two mutually exclusive permission sets, we will set one of them to 0.

Fig. 3(a) shows the average accuracy of the results that it finds in different minimum supports under $|User| = 500$, $|Permission| = 150$. From the figure, we can see that, when the value of the minimum support is low, there is a high accuracy. As the value of the minimum support increases, the accuracy of the results will decrease. This is because the algorithm is always able to generate more frequent permission set under the low minimum support. However, all of the accuracy is more than 70%, which means that our algorithm can generate quality results. Fig. 3(b) shows the accuracy of



(a) $|User| = 500$, $|Permission| = 150$. (b) $|User| = 1000$, $|Permission| = 100$.

Fig. 3. Accuracy comparison under the different parameters.

the results that it finds in different minimum supports under $|User| = 1000$, $|Permission| = 100$. Again, the accuracy of the algorithm is quite good, with the largest data and largest minimum support getting approximately to 75%.

5. Conclusions and future work

While there are many role mining approaches have been proposed recently, none of them considered how to mine constraints. It may fail to reflect the constraints in the security systems. In this paper, we present a constraint mining approach based on the association rule mining algorithm. We first define a variety of constraints and also propose an anti-association rule mining algorithm to generate the mutually exclusive permissions. It can find mutually exclusive permissions that just scans the database only once, while the traditional association rule mining method need to scan database many times. We carry out the experiments to illustrate the effectiveness of the proposed techniques. As a result, the proposed approach has superior performance to traditional algorithm in both speed and generating mutually exclusive permissions.

The current implementation for anti-association rule mining generates the mutually exclusive permissions that are weighted purely on frequency. Hence, if a permission set is only given together to a small number of users, it may not be identified by the algorithm. We intend to investigate this to create a set of tools for mining the mutually exclusive permissions based on weights. Moreover, we will try to introduce the semantic information to further refine the potential constraints.

Acknowledgements

This work is supported by National Natural Science Foundation of China under Grant 60873225, 60773191, 70771043, National High Technology Research and Development Program of China under Grant 2007AA01Z403, Natural Science Foundation of Hubei Province under Grant 2009CDB298, Wuhan Youth Science and Technology Chenguang Program under Grant 200950431171, Open Foundation of State Key Laboratory of Software Engineering under Grant SKLSE20080718, and Innovation Fund of Huazhong University of Science and Technology under Grant 2010MS068 and Q2009021.

References

[1] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, C.E. Youman, Role-based access control models, IEEE Computer 29 (2) (1996) 38–47.
[2] D. Zhang, K. Ramamohanarao, T. Ebringer, T. Yann, Permission set mining: discovering practical and useful roles, in: Proc. 24th International Conference on Annual Computer Security Applications, Califonia, USA, 2008, pp. 247–256.
[3] M.P. Gallagher, A. O'Connor, B. Kropp, The economic impact of role-based access control, Planning report 02–1, National Institute of Standards and Technology, March 2002.
[4] A. Schaad, J. Moffett, J. Jacob, The role-based access control system of a European bank: a case study and discussion, in: Proc. 6th ACM Symposium on Access Control Models and Technologies, Chantilly, Virginia, USA, 2001, pp. 3–9.
[5] X. Ma, R. Li, Z. Lu, Role mining based on weights, in: Proc. 15th ACM Symposium on Access Control Models and Technologies, Pittsburgh, PA, USA, 2010, pp. 65–74.
[6] E.J. Coyne, Role engineering, in: Proc. 1th ACM Workshop on Role-Based Access Control, 1995.
[7] J. Vaidya, V. Atluri, J. Warner, Roleminer: mining roles using subset enumeration, in: Proc. 13th ACM Conference on Computer and Communications Security, Alexandria, USA, 2006, pp. 144–153.
[8] M. Frank, J.M. Buhmann, D. Basin, On the definition of role mining, in: Proc. 15th ACM Symposium on Access Control Models and Technologies, Pittsburgh, PA, USA, 2010, pp. 35–44.
[9] J. Schlegelmilch, U. Steffens, Role mining with ORCA, in: Proc. 10th ACM Symposium on Access Control Models and Technologies, Stockholm, Sweden, 2005, pp. 168–176.
[10] J. Vaidya, V. Atluri, J. Warner, Roleminer: mining roles using subset enumeration, in: Proc. 13th ACM Conference on Computer and Communications Security, Alexandria, USA, 2006, pp. 144–153.
[11] I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. Calo, J. Lobo, Mining roles with semantic meanings, in: Proc. 13th ACM Symposium on Access Control Models and Technologies, Colorado, USA, 2008, pp. 21–30.
[12] D. Zhang, K. Ramamohanarao, T. Ebringer, Role engineering using graph optimisation, in: Proc. 12th ACM Symposium on Access Control Models and Technologies, Antipolis, France, 2007, pp. 139–144.

[13] A. Ene, W. Horne, N. Milosavljevic, P. Rao, R. Schreiber, R.E. Tarjan, Fast exact and heuristic methods for role minimization problems, in: Proc. 13th ACM Symposium on Access Control Models and Technologies, Colorado, USA, 2008, pp. 1–10.
[14] M. Frank, A.P. Streich, D. Basin, J.M. Buhmann, A probabilistic approach to hybrid role mining, in: Proc. 16th ACM Conference on Computer and Communications Security, IL, USA, 2009, pp. 101–111.
[15] H. Takabi, J.B.D. Joshi, StateMiner: an efficient similarity-based approach for optimal mining of role hierarchy, in: Proc. 15th ACM Symposium on Access Control Models and Technologies, Pittsburgh, PA, USA, 2010, pp. 55–64.
[16] J. Vaidya, V. Atluri, Q. Guo, N. Adam, Migrating to optimal RBAC with minimal perturbation, in: Proc. 13th ACM Symposium on Access Control Models and Technologies, Colorado, USA, 2008, pp. 11–20.
[17] H. Lu, J. Vaidya, V. Atluri, Optimal boolean matrix decomposition: application to role engineering, in: Proc. 24th International Conference on Data Engineering, Cancun, Mexico, 2008, pp. 297–306.
[18] F. Geerts, B. Goethals, T. Mielikainen, Tiling databases, in: Proc. 7th International Conference Discovery Science, Padova, Italy, 2004, pp. 278–289.
[19] I. Molloy, N. Li, J. Lobo, L. Dickens, Mining roles with noisy data, in: Proc. 15th ACM Symposium on Access Control Models and Technologies, Pittsburgh, PA, USA, 2010, pp. 45–54.
[20] I. Molloy, N. Li, T. Li, Z. Mao, Q. Wang, J. Lobo, Evaluating role mining algorithm, in: Proc. 14th ACM Symposium on Access Control Models and Technologies, Stresa, Italy, 2009, pp. 95–104.