Role Mining Based on Weights

Xiaopu Ma
xpma@smail.hust.edu.cn

Ruixuan Li*
rxli@hust.edu.cn

Zhengding Lu
zdlu@hust.edu.cn

Intelligent and Distributed Computing Laboratory
School of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan 430074, Hubei, China

ABSTRACT

Role mining from the existing permissions has been widely applied to aid the process of migrating to an RBAC system. While all permissions are treated evenly in previous approaches, none of the work has employed the weights of permissions in role mining to our knowledge, thus providing the motivation for this work. In this paper, we generalize this to the case where permissions are given weights to reflect their importance to the system. The weights can correspond to the property of operations, the sensitive degree of objects, and the attribute of users associated with permissions. To calculate the weight of permissions, we introduce the concept of similarity between both users and permissions, and use a similarity matrix to reinforce the similarity between permissions. Then we create a link between the reinforced similarity and the weight of permissions. We further propose a weighted role mining algorithm to generate roles based on weights. Experiments on performance study prove the superiority of the new algorithm.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Access Controls*; H.2.8 [**Database Management**]: Database Application—*Data Mining*

General Terms

Security, Management

Keywords

RBAC, role engineering, role mining, weight, similarity

1. INTRODUCTION

Role-based access control (RBAC) [5, 14] 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 [25]. Hence, how to generate such candidate roles has become the major challenge in implementing RBAC system [18]. As a solution to create a comprehensive framework for defining the architectural structure of RBAC, *role engineering* is introduced [2].

Essentially, there have 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 [20]. Hence, this approach can well reflect the functional requirements of the organizations. However, this approach is time consuming and costly because there may have dozens of business processes and tens of thousands of users in an organization. As a result, researchers have proposed to use data mining techniques to discover roles from the existing user-permission assignments. Such a bottom-up approach is called *role mining*.

The role mining approach starts from the existing user-permission assignments before RBAC is implemented and aggregates permissions into roles. Hence, this approach is likely to ignore the business functions of the organizations [9] but can generate the architectural structure of RBAC automatically. Role mining has raised significant interests in the research community because much of its process can be automated or semi-automated that can be used as a tool, in conjunction with a top-down approach, to identify potential or candidate roles [11, 20]. Fundamentally, all these role mining approaches use an agglomerative technique to find inherent roles given assigned permissions. The intuition behind the proposed approaches is permissions that are often assigned together should be assigned together. This collection of permissions that are usually assigned implies a role.

In the role mining approach, however, a key challenge that has not been adequately addressed so far is how to identify the weights of permissions and how to discover roles based on weights. In other words, most of the existing role mining approaches didn't consider the different nature and importance of each permission, or treated the permissions evenly. But this is not always the case. For example, the permission "read" of the patient's personal information may be more important than the permission "write" to the patient's personal information. This is because the "read" permission usually leads to more information leakage, but the standard

*Corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SACMAT'10, June 9–11, 2010, Pittsburgh, Pennsylvania, USA.
Copyright 2010 ACM 978-1-4503-0049-0/10/06 ...$10.00.

role mining simply ignores this difference. In another case, the permission "write" to the students' achievement may be more significant than the permission "read" of the students' achievement. Such important permissions may be given to a small number of users. However, these permissions may represent an important role that can only be given to a very few members of staff. As we know, most of the administrative operations are more sensitive and important than the ordinary ones of general users. The permissions of the administrative operations may be assigned to only a small number of administrators, while the permissions of ordinary operations are usually associated with huge amounts of assignments. The permission sets with small number of users may not be identified by the traditional role mining techniques since they didn't take into account the different importance of these operations.

The weight of permissions can be viewed as a function of selected attributes attached to permissions, such as different properties of operations, different sensitive degrees of objects, or different attributes of users associated with permissions. In order to depict the effect of user-permission assignments on the weight of permissions, we introduce the concept of similarity between both users and permissions, and represent it as a similarity matrix *NSM*. The similarity between users can identify whether these users are in one group, which means the more similarity the two users have, the more likely they are in the same department or in the same position. Correspondingly, the similarity between permissions is examined to see whether clusters of permissions can be formed. We use the proportion of the number of common permissions to all permissions the two users have to represent the original similarity of users. Analogously, the original similarity of permissions is defined as the proportion of the number of common users to all users associated with these two permissions. Intuitively, users with similar permissions could have similar roles, while permissions with common users could usually be assigned together. Certainly, this is not enough. The matrix of original similarity may be sparse. It can be reinforced by some other types of similarities through certain complementary matrices, which makes the contained information more dense and meaningful. We contend that matrix representation and processing are effective approaches for combing similarities from different sources. Then we use the techniques that have been used in many real-world domains [8] to compute repetitiously over the *NSM* to improve the quality and utility of the similarity. We introduce the notion about the weight of permissions and compute the weights based on the reinforced similarity. Furthermore, we give an algorithm of role mining based on weights that can produce relevant roles more efficiently. The experiments are carried out to evaluate the performance of the weighted approach and the results prove its superiority.

The remainder of the paper is organized as follows. We discuss the related work in Section 2. The limitations in existing application for role engineering drive our motivation and Section 3 proposes our notion about the reinforced similarity and how to define the weight of permissions. Furthermore, we design a new role mining algorithm based on weights to find roles which scans the user-permission assignments only once. A summary of our experimental results on simulated data 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 [2]. It leads to the initial top-down process oriented approaches for role definition. With the top-down approach, people starts from requirements and successively refines the definitions to result in permissions and roles respectively based on the business functions [4, 13]. 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 [25]. In [15], 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 each permission can only be found along one role path of the hierarchy. Hence, Vaidya et al. [20] propose a subset enumeration approach *RoleMiner* to overcome the above limitation.

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. [24] 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. [3] 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. [6] 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. [16] define a measure for goodness of an RBAC state and use similarity-based approach for optimal mining of role hierarchy. Vaidya et al. [18, 19] and Lu et al. [10] theoretically analyse the role mining problem (*RMP*) and also give two different variations of the *RMP*, call the δ-approx *RMP* and the minimal noise *RMP* 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.

However, the traditional role mining approach assumes that permissions have the same importance without taking account of their weight within user-permission assignments. Hence, if a permission set that represent important roles is only given together to a small number of users, it may not be identified by the traditional role mining techniques. In data mining scope, nevertheless, there already have many approaches to mining association rules based on weights. In [22], Wang et al. mine weighted association rule by first ignoring the weight and finding the frequent itemsets, followed by introducing the weight during the rule generation process. Cai et al. [1] use the concept of k-support bound to find the candidate frequent weighted itemsets and then generate the whole frequent weighted itemsets. Subsequently, some weighted association rules can be deduced. Tao et al.

[17] and Yun et al. [23] also focus on mining those significant relationships involving items with significant weights. Most of these methods show good performance on mining frequent itemsets when there exist different weighted attributes in the items. In role mining area, we can borrow this idea and discover roles based on weights in terms of user-permission assignments.

To this aim, this research tries to assign weight to each permission in a feasible way. We introduce the concepts of similarity between both users and permissions, and then propose a similarity matrix to represent the similarity between users and permissions. Our focus is on how to calculate the similarity and how to define the weights of permissions based on the similarity. We also present a new weighted role mining algorithm *WRM* to address the above problem. The experimental results are tested to show the effectiveness of our findings.

3. ROLE MINING BASED ON WEIGHTS

3.1 Preliminaries

We develop the material in this paper in the context of the NIST standard, the most widely known RBAC model [5]. It consists of RBAC0, RBAC1, RBAC2 and RBAC3. The last two incorporate separation of duty constraints. For the sake of simplicity, we do not consider sessions or separation of duties constraints in this paper.

Definition 1. 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;
- $RolePerms(R) = \{p \in P | (p, R) \in PA\}$, the mapping of role R onto a set of permissions.

Here we can use an $m \times n$ binary matrix M to describe the relationships between users and permissions 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 $u_i (i = 1, ..., m)$ to indicate the ith user and $UserPerms(u_i)(i = 1, ..., m)$ to indicate the set of permissions assigned to the ith user, $p_j (j = 1, ..., n)$ to indicate the jth permission and $PermUsers(p_j)(j = 1, ..., n)$ to indicate the set of users that possess permission p_j. Since users are represented by permission sets and permissions can be described using user sets, we can give a measure of original similarity between users or permissions based on this.

Definition 2. The original similarity between the ith user and the jth user is defined as

$$\mathrm{sim}(u_i, u_j)^{origsim} = \frac{|UserPerms(u_i) \cap UserPerms(u_j)|}{|UserPerms(u_i) \cup UserPerms(u_j)|}$$

This measure is based on a statistical similarity measure called Jacard co-efficient, where 0 implies no similarity between the ith user and the jth user, and 1 represents an exact match between these two users. Analogously, we give the definition of original similarity between the ith permission and the jth permission as follows.

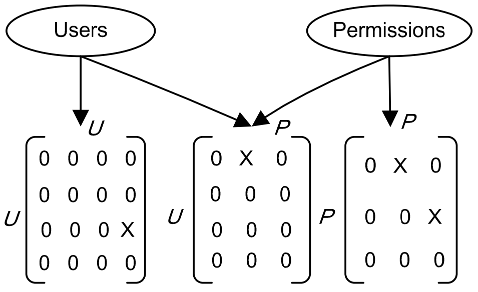

Figure 1: Matrix representations of relationships between users and permissions

Definition 3. The original similarity between the ith permission and the jth permission is defined as

$$\mathrm{sim}(p_i, p_j)^{origsim} = \frac{|PermUsers(p_i) \cap PermUsers(p_j)|}{|PermUsers(p_i) \cup PermUsers(p_j)|}$$

According to the above definition, it just focuses on using a single type of relationship to calculate the similarity between two users or between two permissions. For example, it just uses the set of users that possess certain permissions to calculate the similarity between permissions or the set of permissions assigned to certain users to compute the similarity between users. Although this method is useful to compute the similarity between users or permissions, it is not good at using the external information to make the similarity more accurate. For example, as shown in Figure 1, the relationships between permissions that we call intra-relationships can affect the similarity between permissions on one hand. On the other hand, the relationships between users and permissions that we call inter-relationships can also influence the similarity between permissions. Hence, we can use the inter-relationships and the intra-relationships to reinforce the similarity between permissions. The formal definition of the new similarity matrix that represents both the inter-relationships and intra-relationships is given below. We use intra-type relationships $R_{users} \subseteq U \times U$ to represent the relationships between users, $R_{perms} \subseteq P \times P$ to describe the intra-relationships between permissions . We also use inter-type relationships $R_{users,perms} \subseteq U \times P$ to represent the relationships between users and permissions, $R_{perms,users} \subseteq P \times U$ to describe the inter-relationships between permissions and users. The intra-type relationships R_{users} can be represented as an $m \times m$ matrix AUR (where m is the total number of users). Inside matrix AUR cell $AUR\{i, j\}$ represents the original similarity between the ith user and the jth user. The intra-type relationships R_{perms} can be represented as an $n \times n$ matrix APR (where n is the total number of permissions). Inside matrix APR cell $APR\{i, j\}$ represents the original similarity between the ith permission and the jth permission. The inter-type relationships $R_{users,perms}$ can be represented as an $m \times n$ matrix $AUPR$ (where m is the total number of users, and n is the total number of permissions), where the value of cell $AUPR\{i, j\}$ represents the original similarity between the ith user and the jth permission. Here, we have no prior knowledge of original similarity between permissions and users, then each element in the matrix is set to $1/n$, where n is the total number of permissions. This is equivalent to using a random similarity to represent no-similarity. The inter-type relationship $R_{perms,users}$ can be represented as an $n \times m$ matrix $APUR$ (where n is the total number of permissions, and m is the total number of users), where the value of cell $APUR\{i,j\}$ represents the original similarity between the ith permission and the jth user. Here, we also use the same method as $AUPR$ to set each element in the matrix $APUR$. Hence we get a new similarity matrix NSM that represents both the inter-relationships and the intra-relationships between users and permissions as follows.

Definition 4. The new similarity matrix NSM is defined as

$$NSM = \begin{vmatrix} AUR_{m\times m} & AUPR_{m\times n} \\ APUR_{n\times m} & APR_{n\times n} \end{vmatrix}$$

Then we use the same techniques in [8] to present the reinforced similarity between permissions as:

$$\mathrm{sim}(p_i,p_j)^{resim} = \alpha \times \mathrm{sim}(p_i,p_j)^{origsim} +$$

$$\beta \times \frac{\sum\limits_{\forall x,y\in(P\cup U)} \mathrm{sim}(p_i,x)^{origsim}\mathrm{sim}(p_j,y)^{origsim}\mathrm{sim}(x,y)^{origsim}}{(|P|+|U|)^2}$$

Where $|P|$ is the number of permissions, $|U|$ is the number of users, α and β are positive parameters used to adjust the relative importance of the similarity reinforced by inter-type and intra-type relationships. Hence, we can get a reinforced similarity matrix between permissions as follows. In the matrix, each value can reflect the influence of users and permissions to the similarity between permissions.

$$\begin{array}{c|ccc} & p_1 & \cdots & p_n \\ \hline p_1 & \mathrm{sim}(p_1,p_1)^{resim} & \ldots & \mathrm{sim}(p_1,p_n)^{resim} \\ p_2 & \mathrm{sim}(p_2,p_1)^{resim} & \ldots & \mathrm{sim}(p_2,p_n)^{resim} \\ \vdots & \vdots & \ddots & \vdots \\ p_n & \mathrm{sim}(p_n,p_1)^{resim} & \ldots & \mathrm{sim}(p_n,p_n)^{resim} \end{array}$$

In practice, the weight of permission is a value attached to a permission representing its importance. We denote it as w_p. Depending on the domain, there could be any variable ranging from the types of operations to the types of objects. In other words, the weight of permission is a function of selected weighting attributes therefore denoted as $w(p) = f(a)$. Here, we assume that the permission with the same reinforced similarity has the same effect to the weight of permission. For the sake of simplicity, we can define the weight of permission as follows.

Definition 5. The weight of permission p_i is defined as

$$w_{p_i} = \gamma \times \frac{\sum\limits_{j=1,j\neq i}^{n} \mathrm{sim}(p_i,p_j)^{resim}}{(n-1)} + \delta \times w_o$$

where w_o is the initial weight of permission p_i preset by the system based on the knowledge of comprehensive effect of all factors on permission p_i; γ and δ are parameters used to adjust the relative importance about the weight of permission p_i corresponding to the reinforced similarity and initial weight. If we have no prior knowledge of the initial weight, we can set γ to 1 and δ to 0 respectively.

3.2 Algorithm

In data mining field, association rule mining 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 [7]. Most role mining methods use this idea for extracting frequent itemsets and then define roles [25]. We can analogously use this idea for extracting frequent permission sets and then define roles based on weights. The process of association rules mining algorithm can be decomposed into two sub-processes:

- Scan the database to find all combinations of items whose *support* is greater than *minimum support*. Call those combinations frequent itemsets. The support of an itemset PS in a database D is defined as the ratio of the number of transactions in D containing itemset PS to the number of all transactions in D. An itemset is said to be frequent if its *support* is larger than a user-specified value which called *minimum support*. If the itemset PS has k items, we call it k-frequent itemsets.
- Use the frequent itemsets to generate the desired rules.

In RBAC system, we just use the first step, we regard permissions as items, collection of all user-permission relationships as database and each user-permission assignment as a transaction. Then we can use the same idea to find all combinations of permission sets whose support is greater than minimum support as role sets. In the scope of data mining, there have algorithms such as $MINWAL(O)$ and $MINWAL(W)$ in [1] can mine rules based on weights. However, these algorithms are time costly. Then we design a new algorithm WRM to find the frequent permission sets as roles that costs less time.

We assign a real number $w_{p_i} \in [0,1]$ to p_i for each permission $p_i \in P(i = 1,...,n)$, which we call the weight of permission that can be calculated by the Definition 5. Since the mining of frequent permission sets is to find implications between the elements in 2^P, not P, we must define weights for all elements in 2^P. This can be done as follows.

For any permission set $PS \in 2^P$, suppose that there have $PS = \{p_1,...,p_k\}$, where $p_i \in P(i = 1,...,k)$. We define the weight of PS as follows.

Definition 6. The weight of permission set PS is defined as

$$w_{PS} = \sum_{i=1}^{k} w_{p_i}$$

According to the traditional support function used in *Apriori* algorithm, we define the support based on weights for any $PS \in 2^P$ as follows.

Definition 7. The weighted support of permission set PS is defined as

$$wsf(PS) = w_{PS} \times \frac{numUsers(PS)}{numUsers(All)}$$

where $numUsers(PS)$ is the number of users which possess permission set PS that presented in the user-permission assignments, $numUsers(All)$ is the total number of users in the user-permission assignments. In order to find the important permission set that may not be owned by many users, we choose the product of the weight and the support of the permission set as the weighted support for this permission set. For those permission sets that may not be very important, the weighted support can also be effective if they have been assigned to many users.

The traditional algorithm *Apriori* depends on the downward closure property which governs that subsets of a large itemset are also large. However, it is not always true for the weighted case in our definition. Hence, we use the candidate permission set to describe the candidate permissions that may be the large. If the candidate permission set has k permissions, we call it k-frequent candidate permission set. Algorithm 1 gives the detailed steps about how to find frequent permission sets based on weights.

The algorithm consists of three phases. The first phase consists of lines 3-6. We first compute the original similarity between users and permissions respectively, and then compute the reinforced similarity between permissions. Finally, we generate the weight of each permission and denote the obtained sequence as S.

Phase 2 consists of lines 8-25. The *for* loop in line 8 iterates over all the permissions in the user-permission assignments and generates the 1-frequent permission sets and 1-candidate permission sets. Line 9 calculates the weighted support of p_i. Line 11 inserts the 1-frequent permission p_i into F_1 and C_1. Line 14 to line 22 find the candidate permission set which has not 1-frequent permission. Line 14 removes the weight w_{p_i} from S and line 15 sorts the weight sequence in a decreasing order. Line 16 to line 23 find the 1-candidate permission set. The idea behind line 16 to 23 is to generate the candidate permission sets based on the theory described in [1].

Phase 3 consists of lines 27-29. The *for* loop in line 27 iterates over the set of all $(k-1)$-candidate permission sets. The frequent permission set generation procedure returns the set of all k-frequent permission sets and k-candidate permission sets. The procedure of *FrequentPermissionGen* consists of lines 32-41. The *for* loop in line 32 iterates over all the (k-1)-candidate permission sets to generate the k-frequent permission sets and k-candidate permission sets. We assume that the permissions in a permission set are ordered by the subscript. Line 34 finds the users containing $X \cup Y$. Line 35 computes the number of users containing $X \cup Y$ and then gets the support *wsf*. Line 36 to line 40 finds the k-frequent permission sets that satisfy the minimum support and also generates the k-candidate permission sets. The idea behind lines 34 and 35 is based on the following theorem.

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

$$PermUsers(PS) = PermUsers(p_1) \cap ... \cap PermUsers(p_k);$$
$$numUsers(PS) = countNum(PermUsers(PS)).$$

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

$$PermUsers(\{p_1, p_2\}) = PermUsers(p_1) \cap PermUsers(p_2)$$
$$= \{u_1, u_2, u_3\} \cap \{u_1, u_2, u_4\} = \{u_1, u_2\};$$
$$numUsers(\{p_1, p_2\}) = countNum(PermUsers\{p_1, p_2\})$$
$$= countNum(\{u_1, u_2\}) = 2.$$

The algorithm has two main improvements. First, the key problem with the algorithm $MINWAL(O)$ used in [1] 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. Second, the algorithm considers the different nature and importance of each permission which can be taken as the cluster tendency in pattern extraction [21] with respect to the *Apriori* algorithm.

Algorithm 1 *WRM*

```
Require: $D \equiv (M, N, UP_{M\times N}, wminsup, F, C)$
Require: $M$, the number of users
Require: $N$, the number of permissions
Require: $UP_{M\times N}$ represents user-permission assignments
Require: $wminsup$,the weighted support threshold
Require: $F$ represents all the frequent permission sets
Require: $C$ represents all the candidate permission sets
1: {The Main Procedure}
2: {Initialization}
3: Compute origsim between users
4: Compute origsim between permissions
5: Compute resim between permissions
6: Compute the weight of permissions
7: {Generate the frequent 1-permission sets}
8: for $(i = 1; i \leq N; i++)$ do
9:     $wsf(p_i) = w_{p_i} \times numUsers(p_i)/M$
10:    if $wsf(p_i) \geq wminsup$ then
11:        insert $p_i, wsf(p_i), PermUsers(p_i)$ into $F_1, C_1$
12:    else
13:{Generate the candidate permission sets }
14:        $Remove(S, w_{p_i})$
15:        $Sort(S)$
16:        for $(j = 2; j \leq maxSize; j++)$ do
17:            $maxWeight = w_{p_i} + Sum(S, j-1)$
18:            $minC = \lceil wminsup \times N/maxWeight \rceil$
19:            if $numUsers(p_i) \geq minC$ then
20:                insert $p_i, wsf(p_i), PermUsers(p_i)$ into $C_1$
21:                break
22:            end if
23:        end for
24:    end if
25:end for
26:{Generate the frequent $k$-permission sets}
27:for $(k = 2; C_k \neq \emptyset, k{+}{+})$ do
28:    $F_k = FrequentPermissionGen(C_{k-1}, wminsup)$
29:end for
30: $F = \cup_k F_k$
31:{$FrequentPermissionGen(C_{k-1}, wminsup)$}
32:for $X$ and $Y$ are in $C_{k-1}$ do
33:    if first $k$-2 permissions of $X$ and $Y$ are
                                              the same then
34:        $PermUsers(X \cup Y) = PermUsers(X)$
                                 $\cap PermUsers(Y)$
35:        $wsf(X \cup Y) = (w_X + w_Y)\times$
              $numUsers(PermUsers(X \cup Y))/M$
36:        if $wsf(X \cup Y) \geq wminsup$ then
37:            insert $X \cup Y, wsf(X \cup Y)$,
              $PermUsers(X \cup Y)$ into $F_k, C_k$
38:        else
39:            Compute $minC$ and insert the right candidate
               permission sets into $C_k$
40:        end if
41:    end if
42:end for
43:Return $F$
```

Table 1: Sample matrix for an example organization

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

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 as follows.

1. Compute the original similarity between users based on Definition 2.
   - The set of permissions assigned to each user can be got as follows:
     $$UserPerms(u_1) = \{p_2, p_5\}$$
     $$UserPerms(u_2) = \{p_1, p_2, p_3, p_5\}$$
     $$UserPerms(u_3) = \{p_1, p_2, p_4, p_5\}$$
     $$UserPerms(u_4) = \{p_1, p_2, p_3\}$$
   - Then we can get the original similarity matrix between users as follows:
     $$\begin{array}{c c} & \begin{array}{cccc} u_1 & u_2 & u_3 & u_4 \end{array} \\ \begin{array}{c} u_1 \\ u_2 \\ u_3 \\ u_4 \end{array} & \begin{pmatrix} 1 & 0.5 & 0.5 & 0.25 \\ 0.5 & 1 & 0.6 & 0.75 \\ 0.5 & 0.6 & 1 & 0.4 \\ 0.25 & 0.75 & 0.4 & 1 \end{pmatrix} \end{array}$$
2. Compute the original similarity between permissions based on Definition 3.
   - The set of users that have possessed each permission can be got as follows:
     $$PermUsers(p_1) = \{u_2, u_3, u_4\}$$
     $$PermUsers(p_2) = \{u_1, u_2, u_3, u_4\}$$
     $$PermUsers(p_3) = \{u_2, u_4\}$$
     $$PermUsers(p_4) = \{u_3\}$$
     $$PermUsers(p_5) = \{u_1, u_2, u_3\}$$
   - Then we can get the original similarity matrix between permissions as follows:
     $$\begin{array}{c c} & \begin{array}{ccccc} p_1 & p_2 & p_3 & p_4 & p_5 \end{array} \\ \begin{array}{c} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{array} & \begin{pmatrix} 1 & 0.75 & 0.67 & 0.33 & 0.5 \\ 0.75 & 1 & 0.5 & 0.25 & 0.75 \\ 0.67 & 0.5 & 1 & 0 & 0.25 \\ 0.33 & 0.25 & 0 & 1 & 0.33 \\ 0.5 & 0.75 & 0.25 & 0.33 & 1 \end{pmatrix} \end{array}$$
3. We can get the new similarity matrix *NSM* as follows:
   $$\begin{array}{c c} & \begin{array}{cccccc} u_1 & \cdots & u_4 & p_1 & \cdots & p_5 \end{array} \\ \begin{array}{c} u_1 \\ \vdots \\ u_4 \\ p_1 \\ \vdots \\ p_5 \end{array} & \begin{pmatrix} 1 & \cdots & 0.25 & 0.2 & \cdots & 0.2 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0.25 & \cdots & 1 & 0.2 & \cdots & 0.2 \\ 0.25 & \cdots & 0.25 & 1 & \cdots & 0.5 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0.25 & \cdots & 0.25 & 0.5 & \cdots & 1 \end{pmatrix} \end{array}$$
4. We can get the reinforced similarity between permissions as follows (where $\alpha = 0.9, \beta = 0.1$):
   $$\begin{array}{c c} & \begin{array}{ccccc} p_1 & p_2 & p_3 & p_4 & p_5 \end{array} \\ \begin{array}{c} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{array} & \begin{pmatrix} 0.91 & 0.68 & 0.61 & 0.30 & 0.46 \\ 0.68 & 0.91 & 0.45 & 0.23 & 0.68 \\ 0.61 & 0.45 & 0.90 & 0.00 & 0.23 \\ 0.30 & 0.23 & 0.00 & 0.90 & 0.30 \\ 0.46 & 0.68 & 0.23 & 0.30 & 0.91 \end{pmatrix} \end{array}$$
5. The weight of each permission is:
   $$w_{p_1} = 0.51, \quad w_{p_2} = 0.51, \quad w_{p_3} = 0.32$$
   $$w_{p_4} = 0.21, \quad w_{p_5} = 0.42$$
6. Now, Phase 3 finds the frequent permission sets and candidate permission sets. The first iteration finds the 1-frequent permission set F_1 and 1-candidate permission set C_1 (where $wminsup = 0.4$):
   $$F_1 = \{(p_2, u_1u_2u_3u_4, 0.51)\}$$
   According to Definition 7, p_2 is the only one permission that satisfies the weighted support threshold. However, we can get $minC$ of the permission p_1 is 2 ($minC = \lceil 0.4 \times 4/(0.51+0.51) \rceil = 2$, $numUsers(p_1) = 3$), which implies that it may be the subset of the large 2-frequent permission sets. Similarly, we get the 1-candidate permission set as follows:
   $$C_1 = \{(p_1, u_2u_3u_4, 0.38)\,(p_2, u_1u_2u_3u_4, 0.51)$$
   $$(p_3, u_2u_4, 0.16)\,(p_5, u_1u_2u_3, 0.31)\}$$
7. The next iteration of Phase 3 finds the 2-frequent permission set F_2 and 2-candidate permission set C_2:
   $$F_2 = C_2 = \{(p_1p_2, u_2u_3u_4, 0.77)\,(p_1p_3, u_2u_3, 0.42)$$
   $$(p_1p_5, u_2u_3, 0.465)\,(p_2p_3, u_2u_4, 0.42)$$
   $$(p_2p_5, u_2u_3, 0.465)\}$$
8. In the next iteration of Phase 3 we get
   $$F_3 = \{(p_1p_2p_3, u_2u_4, 0.67)\,(p_1p_2p_5, u_2u_3, 0.72)\}$$
   $$C_3 = F_3 \cup \{(p_1p_3p_5, u_2, 0.31)\,(p_2p_3p_5, u_2, 0.31)\}$$
9. In the final iteration of Phase 3 we get
   $$F_4 = C_4 = \{(p_1p_2p_3p_5, u_2, 0.44)\}$$

Hence the resulting role is r_1 (where $RolePerms(r_1) = \{p_1, p_2, p_3, p_5\}$) .

4. EXPERIMENTAL RESULTS

To show the advantage of our algorithm, we compare our algorithm with *Apriori* algorithm (*Apriori* is a classic algorithm for learning association rules that scans the user-permission assignments many times without using weight) and *ORCA* algorithm (*ORCA* is a role mining algorithm for analyzing on permission assignments to build a hierarchy of permission clusters without using weight). We implement these algorithms on a Pentium (R)D 2.80G PC with 1GB memory to evaluate how well our algorithm performs using different metrics and present the most relevant results. The running platform is Windows XP. We choose the parameters $\alpha = 0.9, \beta = 0.1, \gamma = 1, \delta = 0$.

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

	No. of users	*No. of permissions*
data1	2000	500
data2	2000	1000
data3	2000	1500
data4	2000	2000

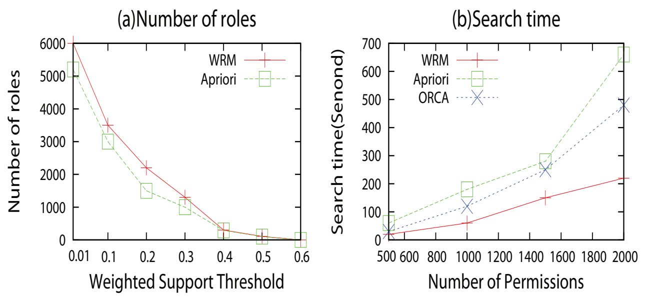

Figure 2: Performance comparison under the fixed number of users

4.1 The Performance of the Algorithm

To study the performance of our algorithm, we generate the synthetic test data as follows. First, we use *for* loop to create the relationships between users and permissions. For each user, a random number of permissions are chosen. We use the variable *zeroflag* to eliminate the users with non-permissions. 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 compute each permission's weight. Algorithm 2 gives the details.

We present the evaluation of our algorithm with *Apriori* and *ORCA* 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 the algorithms. We are interested in two things: the number of roles that it finds in different *weighted support threshold* (Here we use the weighted support threshold as the support threshold for *Apriori* algorithm. We also just compare the different number of roles in different weighted support threshold between *WRM* and *Apriori* because there has no support threshold concept in *ORCA*.) and how quickly to find them. The average number of roles found by the different algorithms for various weighted support thresholds is shown in Figure 2(a). From the figure we can see our algorithm generates more roles than *Apriori* algorithm. It means that we consider the support (the ratio of the number of users that contains the permissions or permission sets to the number of users) and weight (the importance of the permissions or permission sets) at the same time. For example, if we separate the support and weight, we can only find permission sets having both sufficient support and weight. However, if a permission is very important, then even if not so many users contain it or if a permission is not very important while many users own it, the traditional *Apriori* algorithm may ignore some roles. Figure 2(b) shows the average search time for role mining under different number of permissions. Here, the

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

	No. of users	*No. of permissions*
data1	500	1500
data2	1000	1500
data3	1500	1500
data4	2000	1500

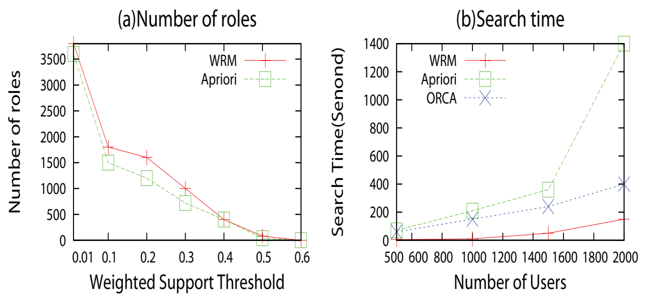

Figure 3: Performance comparison under the fixed number of permissions

```
Algorithm 2 CreateTestData1
Require: $D = (M, N, UP_{M\times N})$
Require: $M$, the number of users
Require: $N$, the number of permissions
Require: $UP_{M\times N}$, represents the relationships between
         users and permissions
1: for $i = 1...M$ do
2:   $zeroflag$=true
3:   while $zeroflag$ do
4:     for $j = 1...N$ do
5:       set $UP\{i, j\}$ to randomly binary number
6:       if $UP\{i, j\} = 1$ then
7:         $zeroflag$=flase
8:       end if
9:     end for
10:  end while
11: end for
12:Compute the weight of each permission
```

search time shows the feasibility of our approach, taking into 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 weighted support of frequent permission sets.

In the second experiment, we fix the number of permissions, while changing the number of users. Table 3 shows the test parameters. Figure 3(a) shows the average number of roles found under different *weighted support threshold*. Figure 3(b) shows the average search time under different number of users. From the figures, it can be seen that when the *weighted support threshold* is low, there is a large number of identified frequent permission sets. As the *weighted support threshold* increases, the number of permission sets that occur above the chosen threshold will decrease. We can also see that our algorithm costs less time and generates more roles than others. If the number of permissions and users in the user-permission assignments is larger, the advantage of our algorithm will be more obvious.

Table 4: Parameter settings for testing accuracy and coverage(Fixed users and permissions, varying roles)

	No. of users	*No. of permissions*	*No. of roles*
data1	1000	100	50
data2	1000	100	100
data3	1000	100	150
data4	1000	100	200

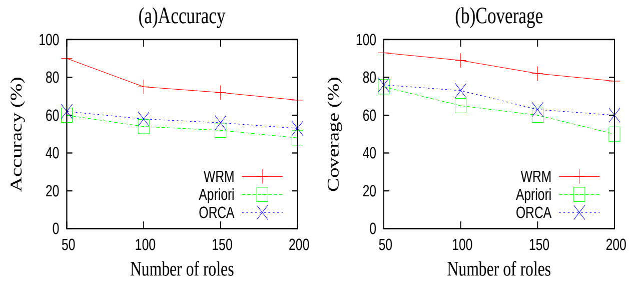

Figure 4: Accuracy and coverage comparison under the parameter settings in Table 4

Table 5: Parameter settings for testing accuracy and coverage(Fixed users and roles, varying permissions)

	No. of users	*No. of permissions*	*No. of roles*
data1	2000	100	100
data2	2000	500	100
data3	2000	1000	100
data4	2000	1500	100

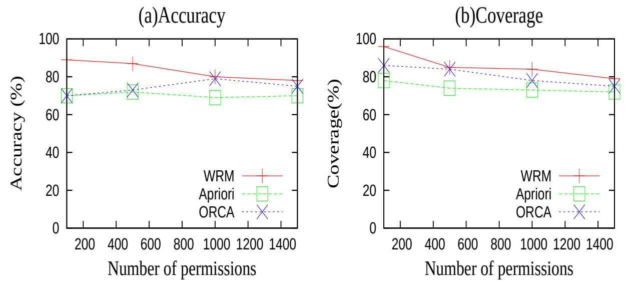

Figure 5: Accuracy and coverage comparison under the parameter settings in Table 5

4.2 The Accuracy and Coverage of the Algorithm

In order to compare the accuracy and coverage of our algorithm with *Apriori* and *ORCA*, we create another synthetic dataset. It performs as follows. First, the maximum number of users, permissions and roles are created respectively. Then we use *for* loop to create the relationships between roles and permissions. For each role, a random number of permissions are chosen. We will delete the roles with all 0 and eliminate the roles with the same permission sets. Then we use another *for* loop to create the relationships between users and roles. Finally, we get the relationships between users and permissions and then compute each permission's weight. Algorithm 3 gives the details.

In order to verify the accuracy and coverage of the solutions obtained, we implement the same metrics as in [25] and [12]. We regard the roles from Algorithm 3 as the original roles and the roles from *WRM* as the generated roles. The accuracy and coverage of the algorithm are defined as follows.

```
Algorithm 3 CreateTestData2
Require: $D = (O, M, N, RP_{O\times N}, UP_{M\times N})$
Require: $O$, the number of roles
Require: $M$, the number of users
Require: $N$, the number of permissions
Require: $RP_{O\times N}$ represents the relationships between
   roles and permissions
Require: $UP_{M\times N}$ represents the relationships between
   users and permissions
1:{Create relationships between roles and permissions}
2: for $i = 1...O$ do
3:   for $j = 1...N$ do
4:     set $RP\{i,j\}$ to randomly binary number
5:   end for
6: end for
7: Delete the roles with all 0
8: Eliminate roles having the same permission sets
9:{Initialize relationships between users and permissions}
10: for $i$=1...$M$ do
11:   for $j$=1...$N$ do
12:     $UP\{i,j\} = 0$
13:   end for
14: end for
15:{Create relationships between users and permissions}
16: for $i = 1...M$ do
17:   for $j = 1...O$ do
18:     Randomly generate binary number $temp$
19:     if $temp$=1 then
20:       for $k$=1...$N$ do
21:         $UP\{i,k\} = UP\{i,k\}|RP\{j,k\}$
22:       end for
23:     end if
24:   end for
25: end for
26: Delete the rows with all 0
27: Compute the weight of each permission
```

Definition 8. Given an original role sr, a generated role dr is said to exactly match the original role sr if and only if $RolePerms(dr) \subseteq RolePerms(sr)$ and $RolePerms(sr) \subseteq RolePerms(dr)$.

If we denote the number of all the original roles as *numRoles(SAll)* and the number of the exactly match between the original role sets and the generated role sets as *numRoles(MAll)*, we can give the definition of the accuracy of the algorithm as follows.

Definition 9. The accuracy of the algorithm is defined as the ratio of the number of generated roles exactly matching the original role sets to the number of original role sets.

$$acc = \frac{numRoles(MAll)}{numRoles(SAll)}$$

Definition 10. Given the generated role sets drs and the original role sets srs, the total number of original permis-

Table 6: Parameter settings for testing accuracy and coverage(Fixed permissions and roles, varying users)

	No. of users	*No. of permissions*	*No. of roles*
data1	1000	500	100
data2	2000	500	100
data3	2500	500	100
data4	3000	500	100

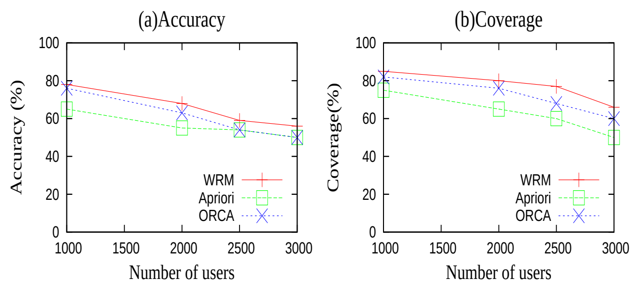

Figure 6: Accuracy and coverage comparison under the parameter settings in Table 6

sions is defined as

$$orn = \mid srs_1 \cup srs_2 ... \cup srs_n \mid (srs_j \subseteq srs, j = 1...n)$$

the number of permissions covered by the generated roles is defined as

$$orc = \mid (drs_1 \cup drs_2 ... \cup drs_n) \cap (srs_1 \cup srs_2 \cup ... \cup srs_m) \mid$$

where $drs_j \subseteq drs(j = 1...n)$ and $srs_k \subseteq srs(k = 1...m)$

Hence the coverage of the algorithm can be defined as follows.

Definition 11. The coverage of the algorithm is defined as the ratio of the number of permissions covered by the generated roles to the number of total original permissions.

$$cov = \frac{orc}{orn}$$

We compare the accuracy and coverage of our algorithm with the other two algorithms on four different synthetic user-permission assignments. In the first set of assignment matrix, we fix the number of users and permissions, while changing the number of roles. Table 4 shows the test parameters. In the second set of assignment matrix, we fix the number of users and roles, while changing the number of permissions. Table 5 shows the test parameters. In the third set of assignment matrix, we fix the number of permissions and roles, while changing the number of users. Table 6 shows the test parameters. In the final set of assignment matrix, all the parameters are variable. Table 7 shows the test parameters. Figure 4 shows the accuracy and coverage of the different algorithms under the parameter settings of Table 4. Figure 5 shows the accuracy and coverage of the different algorithms under the parameter settings of Table 5. Figure 6 shows the accuracy and coverage of the different algorithms under the parameter settings of Table 6. Figure 7 shows the accuracy and coverage of the different algorithms under the parameter settings of Table 7. From these figures, we can see that, for all cases, our algorithm generates more roles exactly covered by the original role set than others. It means that our algorithm can find more roles that reflects the real organization's security requirements. Hence, it can decrease the administrator's workload. On the other hand, our algorithm can offer the largest amount of coverage. If implemented, these are the permission sets that can be identified as roles, which offers large amounts of administration benefit.

Table 7: Parameter settings for testing accuracy and coverage(varying users, permissions and roles)

	No. of users	*No. of permissions*	*No. of roles*
data1	500	100	100
data2	1000	500	150
data3	2000	1000	200
data4	2500	1500	300

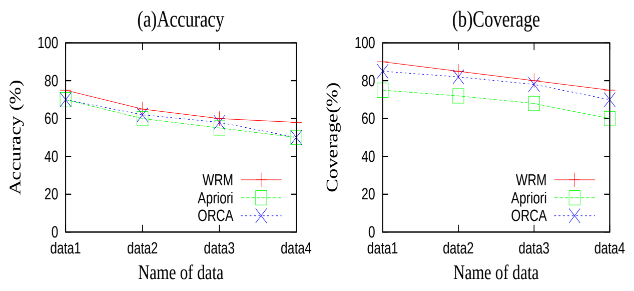

Figure 7: Accuracy and coverage comparison under the parameter settings in Table 7

5. CONCLUSIONS AND FUTURE WORK

While there are many role mining approaches have been proposed recently, none of them considered the nature and importance of permissions. It may fail to find the important roles whose permissions are just given to a very few members of staff. In this paper, we present a role mining approach based on weights of permissions. We use the similarity between both users and permissions to compute the weight of permissions. We also propose a weighted role mining algorithm to generate roles based on weights. It can find frequent permission sets based on weights scanning the database only once, while the traditional role mining methods 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 algorithms in both speed and generating relevant roles. For the future work, we will try to find more significant attributes besides the similarity effect to make the weight of permissions more accurate and meaningful, such as the different types of operations and the sensitive degree of objects. They could be used to further refine the potential roles.

6. ACKNOWLEDGMENTS

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, Open Foundation of State Key Laboratory of Software Engineering under Grant SKLSE20080718, Innovation Fund of Huazhong University of Science and Technology under Grant Q2009021. We thank the anonymous reviewers for their helpful comments.

7. REFERENCES

[1] C. H. Cai, W. C. Fu, C. H. Cheng, and W. W. Kwong. Mining association rules with weighted items. In *Proceedings of the 1998 International Symposium on Database Engineering and Applications*, pages 68-77, 1998.

[2] E. J. Coyne. Role engineering. In *Proceedings of the 1th ACM Workshop on Role-Based Access Control*, 1995.

[3] A. Ene, W. Horne, N. Milosavljevic, P. Rao, R. Schreiber, and R. E. Tarjan. Fast exact and heuristic methods for role minimization problems. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, pages 1-10, June 2008.

[4] E. B. Fernandez and J. C. Hawkins. Determining role rights from use cases. In *Proceedings of the 2th ACM Workshop on Role-Based Access Control*, pages 121-125, 1997.

[5] D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli. Proposed nist standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):224-274, 2001.

[6] M. Frank, A. P. Streich, D. Basin, and J. M. Buhmann. A probabilistic approach to hybrid role mining. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 101-111, November 2009.

[7] F. Geerts, B. Goethals, and T. Mielikainen. Tiling databases. In *Proceedings of the 7th International Conference Discovery Science*, pages 278-289, 2004.

[8] G. Jeh and J. Widom. SimRank: A measure of structural-context similarity. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 538-543, 2002.

[9] A. Kern, M. Kuhlmann, A. Schaad, and J. Moffett. Observations on the role life-cycle in the context of enterprise security management. In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies*, pages 43-51, June 2002.

[10] H. Lu, J. Vaidya, and V. Atluri. Optimal boolean matrix decomposition: application to role engineering. In *Proceedings of the IEEE 24th International Conference on Data Engineering*, pages 297-306, 2008.

[11] I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. Calo, and J. Lobo. Mining roles with semantic meanings. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, pages 21-30, June 2008.

[12] I. Molloy, N. Li, T. Li, Z. Mao, Q. Wang, and J. Lobo. Evaluating role mining algorithms. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies*, pages 95-104, June 2009.

[13] G. Neumann and M. Strembeck. A scenario-driven role engineering process for functional RBAC roles. In *Proceedings of the 7th ACM Symposium on Access Control Models and Technologies*, pages 33-42, June 2002.

[14] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38-47, February 1996.

[15] J. Schlegelmilch and U. Steffens. Role mining with ORCA. In *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies*, pages 168-176, June 2005.

[16] H. Takabi and J. B. D. Joshi. An efficient similarity-based approach for optimal mining of role hierarchy. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, poster session, November 2009.

[17] F. Tao, F. Murtagh, and M. Farid. Weighted association rule mining using weighted support and significance framework. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 661-666, August 2003.

[18] J. Vaidya, V. Atluri, and Q. Guo. The role mining problem: finding a minimal descriptive set of roles. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, pages 175-184, June 2007.

[19] J. Vaidya, V. Atluri, Q. Guo, and N. Adam. Migrating to optimal RBAC with minimal perturbation. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, pages 11-20, June 2008.

[20] J. Vaidya, V. Atluri, and J. Warner. Roleminer: mining roles using subset enumeration. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 144-153, October 2006.

[21] L. Wang, X. Geng, J. C. Bezdek, C. Leckie, and K. Ramamohanarao. Specvat: enhanced visual cluster analysis. In *Proceedings of the 8th IEEE International Conference on Data Mining*, pages 638-647, December 2008.

[22] W. Wang, J. Yang, and P. S. Yu. Efficient mining of weighted association rules (WAR). In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 270-274, 2000.

[23] U. Yun, and J. J. Leggett. WFIM: weighted frequent itemset mining with a weight range and a minimum weight. In *Proceedings of the 5th SIAM International Conference on Data Mining*, pages 636-640, August 2005.

[24] D. Zhang, K. Ramamohanarao, and T. Ebringer. Role engineering using graph optimisation. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, pages 139-144, June 2007.

[25] D. Zhang, K. Ramamohanarao, T. Ebringer, and T. Yann. Permission set mining: discovering practical and useful roles. In *Proceedings of the 2008 Annual Computer Security Applications Conference*, pages 247-256, December 2008.