RoleMiner: Mining Roles using Subset Enumeration*

Jaideep Vaidya
MSIS Department and CIMIC
Rutgers University
Newark, NJ USA
jsvaidya@cimic.rutgers.edu

Vijayalakshmi Atluri
MSIS Department and CIMIC
Rutgers University
Newark, NJ USA
atluri@rutgers.edu

Janice Warner
MSIS Department and CIMIC
Rutgers University
Newark, NJ USA
janice@cimic.rutgers.edu

ABSTRACT

Role engineering, the task of defining roles and associating permissions to them, is essential to realize the full benefits of the role-based access control paradigm. Essentially, there are two basic approaches to accomplish this: the *top-down* and the *bottom-up*. The top-down approach relies on a careful analysis of the business processes to define job functions and then specify appropriate roles from them. While this approach can aid in defining roles more accurately, it is tedious and time consuming since it requires that the semantics of the business processes be well understood. Moreover, it ignores existing permissions within an organization and does not utilize them. On the other hand, the bottom-up approach starts with existing permissions and attempts to derive roles from them, thus helping to automate role definition. In this paper, we present an unsupervised approach called *RoleMiner* that mines roles from existing user-permission assignments. Since a role is nothing but a set of permissions, when no semantics are available, the task of role mining is essentially that of clustering users that have same (or similar) permissions. However, unlike the traditional applications of data mining that ideally require identification of non-overlapping clusters, roles will have overlapping permission needs and thus permission sets that define roles should be allowed to overlap. It is this distinction from traditional clustering that makes the problem of role mining non-trivial. Our experiments with real and simulated data sets indicate that our role mining process is quite accurate and efficient.

Categories and Subject Descriptors

K.6 [**Management of Computing and Information Systems**]: Security and Protection

*This work is supported in part by the National Science Foundation under grant IIS-0306838.

CCS'06, October 30–November 3, 2006, Alexandria, Virginia, USA.

Keywords

RBAC, role engineering, role mining

1. INTRODUCTION

Role-based access control (RBAC) has now become a well-accepted model in many organizations for enforcing security. Roles represent organizational agents that perform certain job functions within the organization. Users in turn are assigned appropriate roles based on their qualifications. According to the RBAC reference model [4], roles describe the relationship between users and permissions. Users are human beings and permissions are a set of many-to-many relations between objects and operations. Roles can be hierarchically structured, where senior roles generally inherit the permissions assigned to junior roles. Additionally, constraints such as separation of duties may be associated with the roles.

Employing RBAC is not only convenient but reduces the complexity of access control because the number of roles in an organization is significantly smaller than that of users. Moreover, the use of roles as authorization subjects, instead of users, avoids having to revoke and re-grant authorizations whenever users change their positions and/or duties within the organization. RBAC essentially decouples users and permissions, thereby reducing the burden of security administration. As a result, RBAC has been implemented successfully in a variety of commercial systems, and has become the norm in many applications.

It has been identified by Edward Coyne [2] that the goal of *role engineering* is to define a set of roles that is complete, correct and efficient. He has pointed out that this task is essential before all the benefits of RBAC can be realized. In particular, role engineering requires defining roles and assigning permissions to them. Often, this is a cooperative process where various authorities from different disciplines should understand the semantics of operations of one another. According to a study by NIST [5], role engineering is the costliest component of RBAC implementation. Since role engineering is a time-consuming and a costly process, organizations are often reluctant to move to RBAC. Therefore, a systematic and efficient means of accomplishing definition of roles is essential to minimizing the expenses of RBAC implementation and maintenance.

There are essentially two ways of accomplishing the task of role engineering: *top-down* and *bottom-up*. In the top-down approach, roles are defined by carefully analyzing and decomposing business processes into smaller units in a functionally independent manner. These functional units are

then associated with permissions on information systems. In other words, this approach begins with defining a particular job function and then creating a role for this job function by associating needed permissions. With dozens of business processes, tens of thousands of users and millions of authorizations, this is seemingly a difficult task. Therefore, relying on a top-down approach in most cases is not viable, although, as shown in case studies such as that presented at SACMAT 2001 by Schaad et al. [13], is being done successfully by some organizations. In contrast, the bottom-up approach starts from the existing permissions before RBAC is implemented and aggregates them into roles. One may also use a mixture of these two approaches to conduct role engineering. While the top-down model is likely to ignore the existing permissions, a bottom-up model may not consider business functions of an organization [8]. However, the bottom-up approach is advantageous in that, much of the role engineering process can be automated. Role mining can be used as a tool, in conjunction with a top-down approach, to identify potential or candidate roles which can then be examined to determine if they are appropriate given existing functions and business models.

In this paper, we propose a bottom-up approach to *mine* roles from the existing permissions to aid in migrating to role based access control. Our approach not only eases the task of role engineering, it also helps in providing the security administrators an insight into the user-role assignment. It serves as a tool to assist the security administrator in discovering potential roles. It does, however, require an expert review of the results to choose which discovered roles are most advantageous to implement.

In order to mine roles, one must first start with a definition of a role. We begin with the fact that a role is simply a set of permissions. With respect to mining roles from permissions, we make the following observations.

- Roles can be assigned overlapping permissions.
- The above also implies that a particular permission might be held by members of different roles. That is, permissions are not exclusive to roles nor are they exclusive to a hierarchy.
- A user may play several different roles, and the user may have a certain permission due to more than one of those roles (since multiple roles may include the same permission).

In our approach, we take into account all the above observations. However, it is important to note that ignoring certain observations would make the job of role mining easier, but may result in more inaccurate set of roles. Unlike the traditional applications of data mining that ideally require identification of clusters that do not overlap, role mining indeed requires identification of overlapping clusters. This is because it is often the case that a user may belong to more than one role. It is this distinction that makes the problem of role mining non-trivial. Therefore, aggregation of permissions to create roles by employing a simple clustering algorithm will likely not result in the discovery of actual roles. This is where we differ from earlier research. In particular, our proposed technique of mining roles is accomplished via subset enumeration. Our algorithm computes all possible interesting roles. The roles are then prioritized according to certain metrics, currently based on counts of users who have the permission set of the discovered roles, and presented as candidate roles.

When compared to the earlier role mining proposals, our approach enjoys several advantages. First, as mentioned earlier, it is capable of detecting roles with overlapping permission sets. Second, given the set of user-permission assignments, the subset enumeration employed in our RoleMiner allows us to identify the set of *all possible roles* within an organization, ordered such that the most useful roles are found at the top of the set. Third, our approach does not force every permission to be part of a role, but instead identifies the permissions that distinguish the role. Other permissions held by users could be identified as anomalous. Fourth, unlike some of the previously proposed approaches that are sensitive to the order in which initial role sets are considered, our approach is order-independent. Finally, it is incremental in that when new users and permissions are added, our RoleMiner updates the roles in an incremental fashion, rather than starting the discovery process from scratch.

This paper is organized as follows. Section 2 discusses some preliminaries to our work, including RBAC and Data Mining. Section 3 presents our algorithm for mining roles. Section 4 presents our experimental evaluation and summarizes the results found. Section 5 discusses some of the consequences of the algorithm. Related work is presented in Section 6. Finally, Section 7 concludes the paper and presents avenues for future research.

2. PRELIMINARIES

In this section, we briefly review the RBAC and clustering concepts.

2.1 Role-based Access Control

We adopt the NIST standard of the Role Based Access Control (RBAC) model [4]. For the sake of simplicity, we do not consider sessions or separation of duties constraints.

Definition 1. [RBAC]

- $U, ROLES, OPS$, and RES are the set of users, roles, operations, and resources.
- $UA \subseteq U \times ROLES$, a many-to-many mapping user-to-role assignment relation.
- PRMS (the set of permissions) $\subseteq \{(op, res) | op \in OPS \bigwedge res \in RES\}$
- $PA \subseteq PRMS \times ROLES$, a many-to-many mapping of permission-to-role assignments.
- $assigned\_users(R) = \{u \in U | (u, R) \in UA\}$, the mapping of role R onto a set of users.
- $assigned\_permissions(R) = \{p \in PRMS | (p, R) \in PA\}$, the mapping of role R onto a set of permissions.

2.2 Clustering

Clustering divides a data set into different groups. The goal of clustering is to find groups that are very different from one another, and whose members are very similar to one another. In clustering, at start, one does not know what the clusters will be, or even which attributes will be used to cluster the data. Consequently, domain experts are needed to interpret the results. Once reasonable clusters

have been found, these can then be used to classify new data. Some of the common algorithms used to perform clustering include Kohonen feature maps [9] and K-means. Berkhin [1] provides an excellent survey of clustering.

Clustering seems to be a ready solution to our problem, since our end goal is to cluster the permissions into roles. One problem is that clustering typically applies to numeric data, while user/permissions data is categorical (in fact, boolean). There has been recent work in the data mining field on clustering categorical data [6, 7]. However, even this suffers from the problem that typically there is single assignment of points to clusters. Thus, in our case, a permission would be assigned to a single role – which is clearly an artificial constraint. This is precisely the drawback of Schlegelmilch and Steffens's role mining approach [14] which does in fact use an agglomerative clustering algorithm to find the roles. Some sort of alternative approach is required to handle the drawbacks.

3. ROLE MINING THROUGH SUBSET ENUMERATION

Our approach to role mining is to discover roles from the data set of permissions possessed by all users. We begin with the assumption that roles are nothing but groups of permissions. In this sense, if the number of total different permissions is k, the number of all possible roles is 2^k. However, the meaningful and existing roles are a subset of these. Clearly, it is infeasible to enumerate all roles. A data driven technique is necessary to efficiently discover roles. First, one must consider the interesting properties of the data that are relevant to role discovery.

We now look at some guiding principles that are relevant to automated mining of roles.

Principle 1: Defacto role definitions are embedded in existing permissions. This basic assumption related to the presence of a role is as follows: If none of the users of a corporation have a particular permission set such as $\{p_a, p_b\}$, then this permission set probably does not form a role. The reason for using the word "probably" is that it is possible that such a role exists but no user has been assigned to that role. However, in such a case, role mining is not going to provide a meaningful answer without supervision. It is impossible for an automated program to distinguish between whether no role exists or no user belonging to such a role is present. Thus, we limit our attention to those subset of permissions that are owned by at least one user.

A lazy technique might work where we enumerate a role only if it is present. Thus, for a user, one could enumerate all subsets of the permissions the user possesses – however, this is also flawed, since very often we have users having hundreds of permissions or even more. It is impossible to enumerate all possible roles even in this limited case.

Principle 2: Identify roles with as many permissions as possible. This principle states that if two roles *always* co-occur in the dataset (i.e., any user will have either both of the roles or neither of them), then these two roles will be together detected as one role consisting of all the permissions. For example, if all users either have both the permissions p_a and p_b or neither of the permissions p_a and p_b, then the permission set $\{p_a, p_b\}$ will be detected as one role. This happens, even if in reality, there are two roles, one consisting of the permission $\{p_a\}$ while the other consists of the permission $\{p_b\}$. However, this is unavoidable in a bottom-up algorithm that utilizes no semantics. Without semantics, it is impossible to differentiate between a single role with all the permissions and two roles each having a subset of the permissions.

This also implies that only those subsets of permissions that are maximally common between at least two users should be enumerated as a role. For example, consider a dataset of only 2 users. If the first user has the permissions $\{p_a, p_b, p_c, p_d\}$, while the other has the permissions $\{p_a, p_b, p_e\}$, $\{p_a, p_b\}$ should be enumerated as a role. Only $\{p_a\}$ or only $\{p_b\}$ should not be considered as roles, since there is no way without the use of semantics to justify the decision. However, if there is a third user with perhaps the permissions $\{p_b, p_f\}$, then $\{p_b\}$ should also be identified as a potential role.

Ideally, for role engineering, we must identify potential roles as well as express some confidence in them. These are two different steps – identifying all possible roles, and then prioritizing / selecting the final set of roles from among them. Our proposed approach to discovering roles, called *RoleMiner* takes into consideration the above guiding principles. The RoleMiner algorithm consists of two parts: role identification and role prioritization. We present the details below.

3.1 Role Identification

We give two algorithms to perform role identification. The first algorithm, called CompleteMiner, is complete in the respect that all potential roles are identified. However, the runtime complexity is exponential – thus it is infeasible, except for fairly small data sets. This is presented here for clarity. The second algorithm, called FastMiner, is fast (the complexity is only n^2). The drawback is that it identifies only a subset of the potential roles. However, as will be argued later, the roles discovered by this algorithm are sufficient for practical purposes.

3.1.1 CompleteMiner

The CompleteMiner consists of three phases, described below:

1. **Identification of Initial Set of Roles:** In this phase, we group all users who have the exact same set of permissions. This can be done in a single pass over the data by maintaining a hash table of the sets of permissions seen. This significantly reduces the size of the data set since all users who have the same roles have the exact same permissions. These form the initial set of roles, say *InitRoles*.

2. **Subset Enumeration**: In this phase, we determine all of the potentially interesting roles by computing all possible intersection sets of all the roles created in the initial phase. Let this set be *GenRoles*. Each unique intersection set is added to the set of generated roles (thus only the unique set of intersections is maintained). Though the number of intersections is equal to the size of the power set of the initial roles ($2^{InitRoles}$), the actual roles enumerated are much smaller (since many intersections result in the empty set, or in the same intersection set). Note that the generated roles are the maximal set of interesting roles. Basically any set of permissions unique to a group of users is detected as a role. However, there is no need-

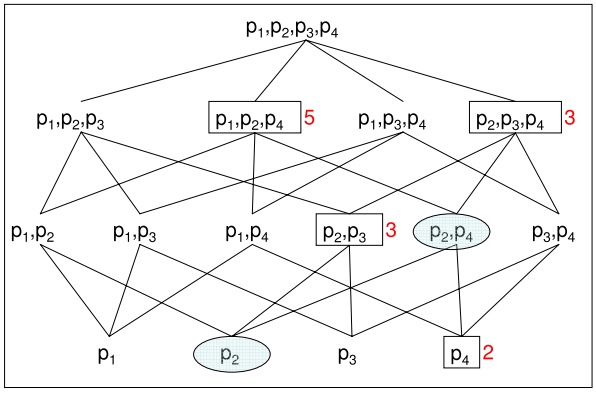


Figure 1: Example

User	p_1	p_2	p_3	p_4
u_1	0	0	0	0
u_2	1	1	0	1
u_3	0	1	1	0
u_4	1	1	0	1
u_5	1	1	0	1
u_6	0	1	1	1
u_7	0	1	1	1
u_8	0	1	1	0
u_9	0	1	1	0
u_{10}	0	0	0	1
u_{11}	0	0	0	1
u_{12}	0	0	0	0
u_{13}	1	1	0	1
u_{14}	1	1	0	1
u_{15}	0	1	1	1

(a) Sample Dataset for an example organization

	Original Count	Generated Count
p_1, p_2, p_4	5	0
p_2, p_3, p_4	3	0
p_2, p_3	3	3
p_2, p_4	0	8
p_2	0	11
p_4	2	8

(b) Results for the sample dataset

Table 1: Example data and results

less bifurcation of permissions. Thus if $\{p_1, p_2, p_3\}$ are owned by several users, but no users have any subset of those permissions, then only $\{p_1, p_2, p_3\}$ is reported as a role.

3. **User Count Computation:** In this phase, for each generated role in *GenRoles*, we count the number of users who have the permissions associated with that role. We actually maintain two sets of counts: (i) the *orig_count(i)*, the original number of users who have exactly the set of permissions corresponding to role i and nothing else, and (ii) *count(i)*, an updated count of users whose permissions are a superset of the permissions associated with this role i. It should be obvious that each generated role will have the original count set to 0, whereas each initial role will have an updated count greater than its original count if and only if it is a subset of one of the other initial roles.

Algorithm 1 gives the detailed steps. The first phase consists of lines 4-11. The for loop iterates over all users while the if statement at line 5 either increments the count of the role (if present) or adds it to the initial set. Phase 2 consists of lines 12-21. The for loop in line 13 iterates over all of the roles initially created. The for loop in lines 15-17 intersect this set with all of the remaining roles initially created and adds all the unique intersections formed to GenRoles. The for loop in lines 18-20 ensures the intersection with all of the roles formed in GenRoles as well. This is necessary to ensure that all possible intersections take place. Phase 3 consists of lines 22-29. The for loop in line 23 iterates over all the generated roles. The for loop in line 24 iterates over all the initial roles. Line 25 checks for existence of the subset relationship. If a generated role is a subset of an initial role, the count of users of the initial role gets added to its count (in line 26).

EXAMPLE 1. The following toy example demonstrates the working of the algorithm. Assume a hypothetical organization with 15 users and 4 permissions. Table 1(a) shows one sample database with the assignment of permissions to users. Since there are 4 permissions, and a role is defined as a collection of permissions, the number of possible different roles is $2^4 = 16$ (i.e., the size of the powerset). Figure 1

Algorithm 1 CompleteMiner

Require: Dataset $D \equiv (U, P)$
Require: $P(u)$ gives the set of permissions assigned to user u
Require: $R(x)$ represents a role consisting of the set of permissions x
Require: $Count(i)$ gives the count of users associated with role i

1: {Cluster users into initial roles based on exact match of the set of permissions}
2: $InitRoles \leftarrow \{\}$
3: $GenRoles \leftarrow \{\}$
4: **for** each user $u \in U$ **do**
5: **if** $R(P(u)) \notin InitRoles$ **then**
6: Set orig_count of $R(P(u))$ to 1
7: $InitRoles \leftarrow InitRoles \cup R(P(u))$
8: **else**
9: Increment *orig_count* of $R(P(u))$
10: **end if**
11: **end for**
12: {Enumerate all possible interesting roles}
13: **for** each Role $i \in InitRoles$ **do**
14: $InitRoles \leftarrow InitRoles - i$
15: **for** each Role $j \in InitRoles$ **do**
16: $GenRoles \leftarrow GenRoles \cup (i \cap j)$
17: **end for**
18: **for** each Role $j \in GenRoles$ **do**
19: $GenRoles \leftarrow GenRoles \cup (i \cap j)$
20: **end for**
21: **end for**
22: {Count number of users belonging to each candidate role}
23: **for** each Role $i \in GenRoles$ **do**
24: **for** each Role $j \in InitRoles$ **do**
25: **if** $i \subset j$ **then**
26: $Count(i) \leftarrow Count(i) + orig\_count(j)$
27: **end if**
28: **end for**
29: **end for**

depicts 15 of those roles (all excluding the empty set - i.e., the role with no permissions).

Now, in the first phase of our algorithm, the set *InitRoles* gets initialized to $\{\{p_1, p_2, p_4\}, \{p_2, p_3, p_4\}, \{p_2, p_3\}, \{p_4\}, \{\}\}$. These roles along with their corresponding counts are rectangled in Figure 1. The empty set, along with its count of 2 is not shown in the figure since it does not add to the computation in phase 2 or 3. In phase 2, we enumerate all possible unique intersection sets of the initial roles found in phase 1. These result in two additional roles, $\{\{p_2, p_4\}, \{p_2\}\}$, which are ovaled in the figure. Since $\{p_2, p_3\}, \{p_4\}$ and $\{\}$ are also the result of some intersections, at the end of phase 2, *GenRoles* gets set to the roles $\{\{p_2, p_3\}, \{p_2, p_4\}, \{p_2\}, \{p_4\}, \{\}\}$. In phase 3, the generated roles are matched to the corresponding counts which are 6,8,11,10, and 2. These are shown in table 1(b).

3.1.2 FastMiner

The key problem with the CompleteMiner algorithm is its computational complexity. Since we compute *all* possible intersections, the running time is exponential. This is quite infeasible, except for very small data sets. Also, phase 3 of Algorithm 1 is unnecessary. An efficient implementation should be able to count number of users in phase 2 itself. This brings us to the efficient algorithm for role mining: The FastMiner algorithm has two main improvements. First, the only intersections performed are between pairs of initial roles. Also, the user counts are computed while intersecting the initial roles. Thus, the computational complexity is $O(n^2)$, instead of exponential. The first phase (lines 4-11) is identical to the CompleteMiner. In phase 2 (lines 13-33), all intersecting roles between pairs of users are found. Lines 17-31 update the counts for the generated role. The key is to maintain a list of contributing initial roles for each generated role (and only add counts if an initial role has not already contributed). This ensures that there is no double counting of users. Algorithm 2 provides the details.

Algorithm 2 FastMiner

Require: Dataset $D \equiv (U, P)$
Require: $P(u)$ gives the set of permissions assigned to user u
Require: $R(x)$ represents a role consisting of the set of permissions x
Require: $Count(i)$ gives the count of users associated with role i

1: {Cluster users into initial roles based on exact match of the set of permissions}
2: $InitRoles \leftarrow \{\}$
3: $GenRoles \leftarrow \{\}$
4: **for** each user $u \in U$ **do**
5: **if** $R(P(u)) \notin InitRoles$ **then**
6: Set orig_count of $R(P(u))$ to 1
7: $InitRoles \leftarrow InitRoles \cup R(P(u))$
8: **else**
9: Increment *orig_count* of $R(P(u))$
10: **end if**
11: **end for**
12: {Enumerate all intersecting roles between pairs of users}
13: **for** each Role $i \in InitRoles$ **do**
14: $InitRoles \leftarrow InitRoles - i$
15: **for** each Role $j \in InitRoles$ **do**
16: $NewRole \leftarrow i \cap j$
17: **if** $NewRole \notin GenRoles$ **then**
18: $Count(NewRole) \leftarrow Count(NewRole) + orig\_count(i)$
19: $Count(NewRole) \leftarrow Count(NewRole) + orig\_count(j)$
20: Add i, j to the list of contributors for NewRole
21: $GenRoles \leftarrow GenRoles \cup NewRole$
22: **else**
23: **if** i has not contributed before to NewRole **then**
24: $Count(NewRole) \leftarrow Count(NewRole) + orig\_count(i)$
25: Add i to the list of contributors for NewRole
26: **end if**
27: **if** i has not contributed before to NewRole **then**
28: $Count(NewRole) \leftarrow Count(NewRole) + orig\_count(j)$
29: Add j to the list of contributors for NewRole
30: **end if**
31: **end if**
32: **end for**
33: **end for**

Key features

While the FastMiner algorithm is quite practical, the trade off is that only a subset of the roles are identified. Specifically, only those roles that are maximally common to any 2 users are found. Thus, any role that is common to at least 3 users, but not maximally common to any 2 of them, will not be identified by the algorithm. Figure 2 highlights this problem. Figure 2(a), 2(b) and 2(c) show collections of permissions that will be detected as roles by FastMiner. However, the collection of permissions shown in Figure 2(d) will not be detected by FastMiner. These will be detected as a role only if we consider intersections between triples of initial roles (rather than pairs). This would result in an $O(n^3)$ algorithm. However, even in this case intersections between quadruples would not be detected. In the most general case, this degenerates back to the CompleteMiner algorithm. One might think that as we increase the number of roles considered in the intersections, we would also increase the accuracy of the results. While increasing complexity would improve accuracy in most data mining problems, it is not necessarily the case in role mining. Essentially, performing more intersections would result in a large number of candidate roles, making identification of useful roles more difficult. Indeed, it is not clear how one could formally quantify the correctness of the algorithm. This would require that we *know* the real set of roles. However, different system administrators might choose to have different roles from the same set of candidates due to differing situations and personalities. For example, one system administrator might choose to merge two roles and create a third super-role for the sake of convenience even though having the original two roles is sufficient to describe the organization. There are no right or wrong ways of specifying roles – the final set of roles is unique to each situation. A formal metric is needed to help understand what set of discovered candidate roles are "good" or useful roles. By quantifying this, one could determine if increasing the number of intersections (and thus the complexity) is worth the additional results. We plan on doing this as part of future work. For now, we argue that intersections between pairs are typically sufficient for detecting most roles, as long as most users have few roles. The experimental results further elucidate this point. Interestingly, the FastMiner algorithm gives the exact same results as the CompleteMiner algorithm for the toy example given earlier.

3.2 Role Prioritization

Since the FastMiner algorithm typically identifies lots of potential roles, these need to be prioritized/ordered in some way. Without the use of semantics, there is very little that we can do. One possibility is to simply order the roles according to the number of users having that role. However, instead of simply using the final Count, we use the following predicate to sort: $(orig\_count * priority + Count)$. Here, priority is simply the multiplication factor used to bias the results towards the roles found in the initial phase (i.e., do not report a generated set as a role unless it is really interesting). Thus, this ensures that a well supported original role does not get lost in the chaff of generated roles (i.e., roles that are not part of *InitRoles*). A priority factor of 0 simply sorts the roles according to the user count. We experimented with several different values for priority.

One problem with this procedure is that rare roles will not be found easily (since they will be significantly lower in the sorted list). As can be seen, the postprocessing in this step is currently quite simple. There are many other possibilities, and this is a significant task for further research. The use of semantics will also make this step significantly better. In the ideal, we wish to develop a confidence metric for a candidate role that would enable us to determine how certain we are about the suitability of the role.

Algorithm 3 CreateTestData(NRoles, NUsers, NPermissions, MRolesUsr, MPermissionsRole)

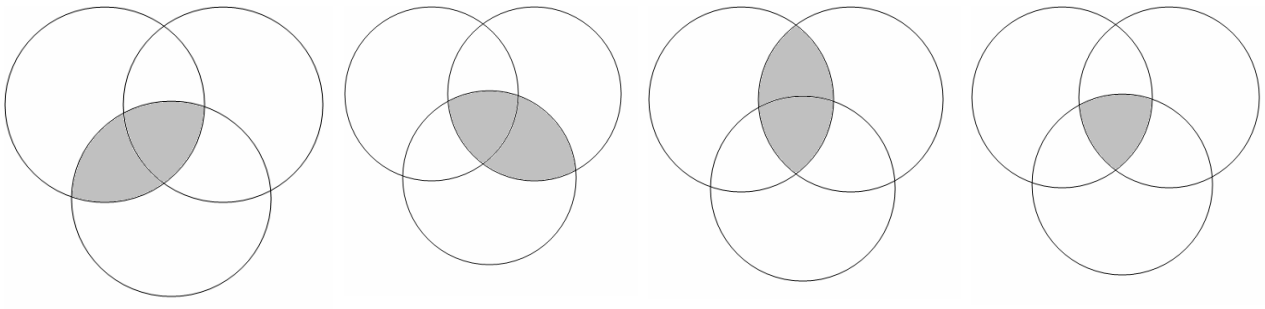
Require: NRoles, the number of roles to be generated
Require: NUsers, the number of users to be generated
Require: NPermissions, the total number of permissions
Require: MRolesUsr, the maximum number of roles a user can have
Require: MPermissionsRole, the maximum number of permissions a role can have
1: {Create the Roles}
2: **for** $i = 1 \ldots NRoles$ **do**
3: Set nrt to a random number between $1, \ldots, MPermissionsRole$
4: Set Roles[i] to nrt randomly chosen permissions
5: **end for**
6: {Create the Users}
7: **for** $i = 1 \ldots NUsers$ **do**
8: Set nrl to a random number between $0, \ldots, MRolesUsr$
9: Randomly select nrl roles from Roles
10: Set Users[i] to the permissions in the union of the selected roles
11: **end for**
12: Return Users

4. RESULTS SUMMARY

In order to validate our algorithm, we created a test data generator. The test data generator performs as follows: First a set of roles are created. For each role, a random number of permissions up to a certain maximum are chosen to form the role. The maximum number of permissions to be associated with a role is set as a parameter to the algorithm. Next, the users are created. For each user, a random number of roles are chosen. Again, the maximum number of concurrent roles a user can have is set as a parameter to the algorithm. Finally, the user permissions are set according to the roles the user has. In some cases, the number of roles randomly chosen is 0 – indicating that the user has no roles – and therefore no permissions. Algorithm 3 gives the details.

Since the test data creator algorithm is randomized, we actually ran it 5 times on each particular set of parameters to generate the datasets. Our role mining algorithm was run on each of the created data sets. All results reported for a specific parameter set are averaged over the 5 runs.

For each set of experiments, we report the speed as well as the accuracy of the fast miner algorithm. To measure the accuracy, we find the percentage of true roles found in 1x the number of real roles, as well as in 2x the number of roles in the results. Thus, if the real number of roles is 200, we count the percentage of real roles found in the first 200, as well as the first 400 roles reported in the results of the algorithm.



(a) $A \cap B$ (b) $A \cap C$ (c) $B \cap C$ (d) $A \cap B \cap C$

Figure 2: Limitations of FastMiner

Dataset	Parameters				
	NRoles	NUsers	NPermissions	MRolesUsr	MPermissionsRole
data1	10	2000	100	3	10
data2	100	2000	500	3	50
data3	100	2000	1000	3	100
data4	100	2000	2000	3	200

Table 2: constant number of users/roles, varying permissions

In the first set of experiments, we kept the number of users and roles constant, while changing the number of permissions (and correspondingly, the number of permissions per role). Table 2 describes the test parameters. Figure 3(b) shows the time taken by the algorithm, while Figure 3(a) shows the accuracy of the algorithm. It can be clearly seen that the accuracy is close to 100% when we look at twice the number of real roles, and close to 85% when we look at exactly the number of real roles in the results. This is quite good. The algorithm also runs quite fast, taking around 5 minutes on the largest datasets.

In the second set of experiments, we kept the number of roles and permissions constant while varying the number of users. Table 3 describes the test parameters. Figure 4(a) shows the time taken by the algorithm, while Figure 4(b) shows the accuracy of the algorithm. Here, we can see that accuracy actually increases with the number of users. The algorithm performs quite well as long as the number of roles is not more than 1/10th the number of users. This is quite realistic, and therefore the algorithm should perform well in real situations. Again, the speed of the algorithm is quite good, with the largest datasets (of 5000 users) taking approximately 23 minutes.

In the third set of experiments we keep the permissions constant, and vary the number of users as well as the number of roles. Table 4 describes the test parameters. Figure 5(a) shows the time taken by the algorithm, while Figure 5(b) shows the accuracy of the algorithm. This actually gives a surprising result – the accuracy of the algorithm significantly drops off at the largest data point (with 5000 users/500 roles). Similarly, the running time drastically increases as well. However, this is not due to the number of users (as the earlier experiment showed), but rather due to the number of roles vis-a-vis the number of permissions. When the ratio of permissions to roles is small (in this case, $1500/500 = 3$), the algorithm performs quite poorly. This is because many spurious intersections are formed which do not contribute to real roles. As long as the number of permissions is at least 5 times the number of roles, the algorithm performs quite well.

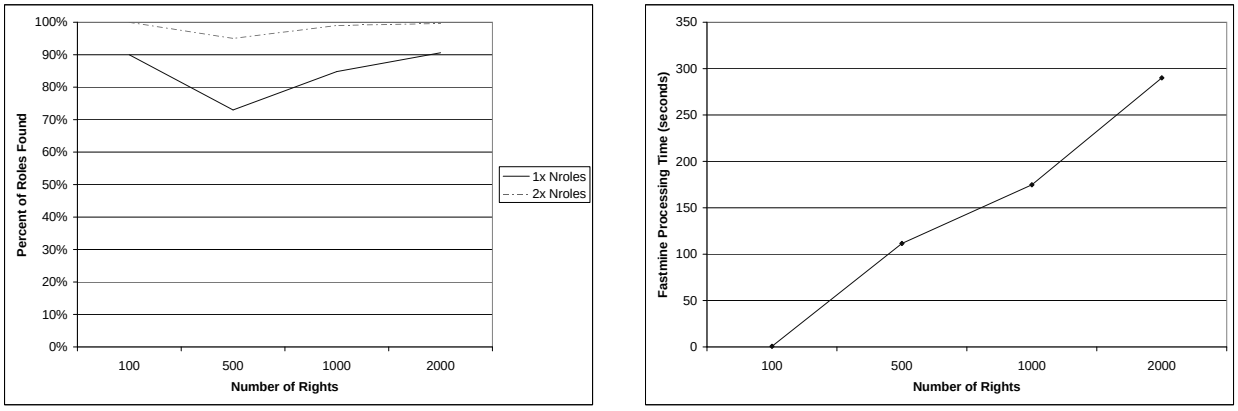
We also ran experiments on the data set provided by Ulrike Steffens [14], consisting of over 6000 users and 1671 permissions. This dataset was processed by the FastMiner algorithm in 17 minutes, which is quite fast. Interestingly, around 1400 users had no permissions at all. Out of the 679 roles discovered by OR-CA, our algorithm discovered 118 matching roles. The rest of the roles discovered by our algorithm were different from the roles in OR-CA. This is at least partially due to the fact that many of the roles we discovered have overlapping rights while this only occurs in a single tree path along a role hierarchy in ORCA. These results have been communicated to Steffens for verification (since the data was anonymized, we had no way of checking the results). We are now awaiting further communication with them.

5. DISCUSSION OF OUR APPROACH

In this section, we discuss some of the key properties of our approach. As part of identifying roles, the algorithm can also make suggestions on candidate roles that should be more heavily considered. For example, the top 100 roles reported by the algorithm are good candidates to be selected as roles based on the data set. This can some times show fallacies in pre-existing role sets.

Another useful property of the algorithm is that it is order independent. Irrespective of the order of users or permissions, the same subsets are generated. Therefore the same set of roles is generated. This is not true of many clustering algorithms and therefore is a very valuable property to have.

The role prioritization phase should actually be much more complex. For example, one could find the minimal set of roles such that all users are covered by those roles.



(a) Accuracy

(b) Speed

Figure 3: Constant number of users

Dataset	Parameters				
	NRoles	NUsers	NPermissions	MRolesUsr	MPermissionsRole
data1	200	500	1500	3	150
data2	200	1000	1500	3	150
data3	200	3000	1500	3	150
data4	200	5000	1500	3	150

Table 3: Constant number of permissions/roles, varying users

Organization structure could be used to arrange roles in a hierarchy. Other analysis is also possible. We are experimenting with several metrics currently.

Furthermore, the algorithm is incremental. Only the final list of roles along with their associated information needs to be maintained. When a new user is added to the system, the users set of permissions can be intersected with the existing roles to determine if any additional roles are created. Updating the existing roles can also be done at the same time. This especially makes a lot of sense when a new batch of users is to be added. All of the prior work on intersection no longer needs to be repeated.

6. RELATED WORK

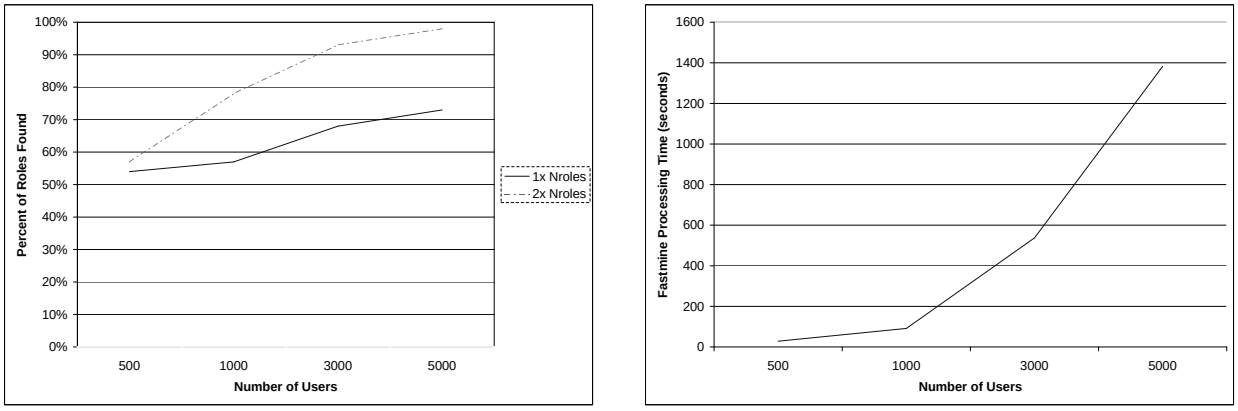
A number of approaches have been proposed in the literature to accomplish the task of role engineering, which can be categorized into three approaches: top-down, bottom-up, and hybrid. While the top-down approach defines roles by examining the business processes, the bottom-up approach typically aggregates existing permissions to come up with roles.

Coyne [2] is the first to describe the role engineering problem, and to present the concepts of the top-down approach. Later, Roeckle et al. [12] present a process-oriented approach, which first analyzes business processes to deduce roles and access permissions on systems are assigned to the roles. Shin et al. [15] present a system-centric approach that examines backward and forward information flows and employs UML to conduct a top-down engineering of roles. Thomsen et al. [16] propose a bottom-up approach, which derives permissions from objects and their methods and then derive roles derived from these permissions. Neumann and Strembeck [11] consider usage scenarios as a semantic unit for deriving permissions, which are then aggregated into roles. Epstein and Sandhu [3] propose to use UML for facilitating role engineering, where roles can be defined in either a top-down or a bottom-up manner. Kern et al. [8], propose a life-cycle approach, an iterative-incremental process, that considers different stages of the role life-cycle including role analysis, role design, role management, and role maintenance.

Kuhlmann, Shohat, and Schmipf [10] present another bottom-up approach, which employs a clustering technique similar to the k-means clustering. As such, it is required to first pre-define the number of clusters.

Comparison with ORCA: More recently, Schlegelmilch and Steffens [14] have proposed a role mining approach (known as ORCA). Since our proposed approach is close to this work, we provide more details on ORCA.

The fundamental difference between our RoleMiner and ORCA lies in the way clusters are formed. ORCA clusters access permissions (which they call rights) based on the users having those permissions. These initial clusters are then merged on the basis of maximal overlap between users assigned to the permissions to form a role hierarchy. Specifically, the ORCA algorithm begins with one cluster for every permission (c_r) which initially make up the set Clusters. ORCA defines permissions(c_r) = the set of permissions associated with the cluster and members(c_r) = the set of people that have permission r. It starts with a partially ordered set of clusters ($\prec$), the cluster hierarchy. Initially ($\prec$) is an empty set. Note that, just because many people have that set of permissions, that cluster is not necessarily a role.



(a) Accuracy

(b) Speed

Figure 4: Constant number of permissions/roles, varying users

Dataset	Parameters				
	NRoles	NUsers	NPermissions	MRolesUsr	MPermissionsRole
data1	10	100	1500	3	150
data2	50	500	1500	3	150
data3	100	1000	1500	3	150
data4	500	5000	1500	3	150

Table 4: Constant number of permissions, varying users/roles

Therefore, ORCA may not result in the actual set of roles.

Pairs of clusters with maximal overlap among their members are first merged and placed in the $\prec$. That is, for every pair of clusters $\langle c_i, c_j \rangle$, members($\langle c_i, c_j \rangle$) = members(c_i) $\cap$ members(c_i) and permissions($\langle c_i, c_j \rangle$) = permissions(c_i) $\cup$ permissions(c_j). The pairs of clusters having the most number of members in common are considered, and the pair with the largest set of permissions selected. If there is more than one pair with the largest set of permissions, one cluster is picked randomly. Let this selected cluster be c_s merged from c_a and c_b. c_s is added as a super-cluster of c_a and c_b, and c_a and c_b are no longer considered. The steps of finding a pair of clusters with maximal overlap and maximum permissions is then repeated until no more clusters can be found. It could be the case that the overlapping users may be members of two roles, however, ORCA assumes that the merged one is always a super-role of the two.

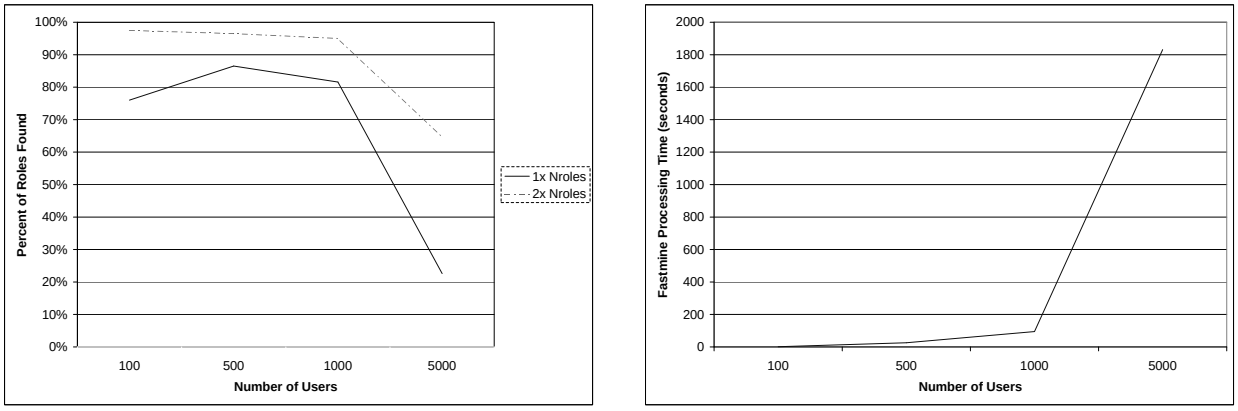
One of the main limitations of ORCA is that permission cannot be associated with more than one role unless the roles are along the same path of the role hierarchy tree. The authors indeed point out that this is a major issue stating that "permissions are seldom used by one role only and may be necessary for incomparable roles". Another issue is that their algorithm assumes a cluster is a good candidate for a role if it has the maximal number of people in common. In reality, this is not necessarily true, and a good role might as well have very few members. A third issue is that when clusters are merged, it is assumed that the merged cluster is a super-role of the two from which it is formed. While this may sometimes be the case, it may also be the case that the merged roles are unrelated except that several people are members of both roles. In addition, the algorithm proposed in ORCA is more computationally intensive because it begins by forming a cluster for every permission.

Since roles are nothing but groups of permissions, all users playing a role will have similar (or the same set of) permissions. Therefore, we cluster users having similar access permissions and assume this as our initial set of roles. We begin by forming a cluster for every group of users having the exact same permissions. Unless every user has a completely different set of permissions, the number of initial clusters must be smaller than the initial set obtained by ORCA.

7. CONCLUSIONS AND FUTURE RESEARCH

Role mining is a critical process for migrating existing systems with many access permissions and users to RBAC. It takes an agglomerative approach of finding inherent roles given assigned permissions. In this paper, we presented an unsupervised role mining process, *RoleMiner* which has three major advantages as compared to earlier role mining proposals. First, it is capable of detecting roles with overlapping permission sets. Second, it employs subset enumeration on permissions to allow us to identify the set of *all possible roles* in the organization. Third, we do not assume that every permission must be part of a role. Permissions that distinguish a role are found. Other permissions held by users could be identified as anomalous and be added upon further analysis.

While our process was shown to be very accurate in finding candidate roles, there is much additional work that we plan to accomplish. We have only a rudimentary approach to deciding which candidate roles are most useful based on



(a) Accuracy

(b) Speed

Figure 5: Constant number of permissions, varying users/roles

user counts. We are investigating other metrics that could additionally or alternatively be used to identify the best roles to consider.

Other work includes using the semantics associated with permissions. The only data set available to us had no semantics and so our process found roles purely on the basis of whether a user had a permission or not. If information on the type of permissions or resources were available, it could be used to further refine the roles. Permissions that are semantically related to different functions could be separated when post-processing the results, thus uncovering whether a potential role (cluster of permissions) was actually a role or a blend of two or more roles. Semantics would also be helpful in pre-processing data. By knowing the semantics of the resources, permissions associated with both organizational as well as functional roles might be distinguishable, making further refinement of roles found more accurate.

8. REFERENCES

[1] Pavel Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002.

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

[3] P. Epstein and R. Sandhu. Engineering of role/permission assignment. In *17th Annual Computer Security Application Conference*, December 2001.

[4] D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *TISSEC*, 2001.

[5] M. P. Gallagher, A.C. O'Connor, and B. Kropp. The economic impact of role-based access control. *Planning report 02-1, National Institute of Standards and Technology*, March 2002.

[6] Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. Cactus: Clustering categorical data using summaries. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 73–83, New York, NY, USA, 1999. ACM Press.

[7] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. ROCK: A robust clustering algorithm for categorical attributes. *Information Systems*, 25(5):345–366, 2000.

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

[9] Teuvo Kohonen. The self organizing map. *IEEE Transactions on Computers*, 78(9):1464–1480, 1990.

[10] Martin Kuhlmann, Dalia Shohat, and Gerhard Schimpf. Role mining - revealing business roles for security administration using data mining technology. In *Symposium on Access Control Models and Technologies (SACMAT)*. ACM, June 2003.

[11] G. Neumann and M. Strembeck. A scenario-driven role engineering process for functional rbac roles. In *7th ACM Symposium on Access Control Models and Technologies*, June 2002.

[12] Haio Roeckle, Gerhard Schimpf, and Rupert Weidinger. Process-oriented approach for role-finding to implement role-based security administration in a large industrial organization. In ACM *RBAC*, 2000.

[13] A. Schaad, J. Moffett, and J. Jacob. The role-based access control system of a European bank: A case study and discussion. *In Proceedings of ACM Symposium on Access Control Models and Technologies*, pages 3–9, May 2001.

[14] Jürgen Schlegelmilch and Ulrike Steffens. Role mining with ORCA. In *Symposium on Access Control Models and Technologies (SACMAT)*. ACM, June 2005.

[15] Dongwan Shin, Gail-Joon Ahn, Sangrae Cho, and Seunghun Jin. On modeling system-centric information for role engineering. In *8th ACM Symposium on Access Control Models and Technologies*, June 2003.

[16] D. Thomsen, D. O'Brien, and J. Bogle. Role based access control framework for network enterprises. In *14th Annual Computer Security Application Conference*, pages 50–58, December 1998.