International Journal of Computer Information Systems and Industrial Management Applications (IJCISIM)
ISSN: 2150-7988 Vol.1 (2009), pp.258-265
http://www.mirlabs.org/ijcisim

Role Mining in Access History Logs

Mohammad Jafari[*], Amir H. Chinaei[*], Ken Barker[*], Mohammad Fathian[#]
[*]*Department of Computer Science, University of Calgary, Alberta, CANADA*
[#]*Department of Electronic Commerce, Iran University of Science and Technology, Tehran, IRAN*
{jafarm, achinaei, kbarker}@ucalgary.ca, fathian@iust.ac.ir

Abstract

*A novel approach for role mining in the context of role engineering for role-based access control is developed in this paper. We propose a simple algorithm, based on the assumption that permissions from the same role appear near each other in the access history log. Closely co-occurring groups of permissions are selected as candidate roles and are ranked based on a novel heuristic, called role cohesion, that quantizes the permission proximity of a candidate role in the access log. High-rank roles are identified using the algorithm, which is tested with a simulation scenario.***

1. Introduction

Role-based access control (RBAC) has been an interesting research topic for at least the past two decades. It has been clearly defined by introducing reference models [8], and it is finally codified in a standard form [2]. However, before a system can benefit from RBAC's advantages, there must be a process of role engineering, in which the enterprise roles and their corresponding permissions are identified [3]. Role mining is a well-known concept in role engineering, since it is understood that successful implementations, particularly in legacy systems, depend on the development of auto matic or semi-automatic mechanisms. The core idea is to utilize data mining techniques to infer access roles from the information implicit in the system definition. This has been referred to as a *bottom-up* approach, which contrasts with the *top-down* one wherein high-level system artifacts—such as organizational structure, business processes, and job descriptions—are studied to determine access roles [4, 7, 10]. The idea of role mining is based on the presumption that access roles are already implicit in user-permission assignments [8] and a reverse engineering approach is very likely to successfully discover them. This assumption is reasonable for two reasons: first, according to the principle of least-privilege, permissions are granted to users based on their job needs; and, security officers often have the organizational roles in mind when assigning permissions to a user.

Kuhlmann and Schimpf [6] developed the first role mining framework by formulating role mining as an inter-disciplinary topic that links data-mining technologies to RBAC. They introduce a seven-step process for role mining, starting from choosing the information source, and leading to role inference and role creation. Based on the well-known *k*-means algorithm, their proposed algorithm clusters permissions into a predetermined number of permission sets. The authors affirm, however, that the resulting roles should pass plausibility and correctness checks by information technology experts before implementation.

Schlegelmilch *et al.* define a concrete role-mining scheme [9] by exploiting a hierarchical clustering algorithm. They propose a role mining tool that uses current user-permission assignments to discover role patterns. They use the membership cardinality of each candidate role as a heuristic for ranking the resulting role set. The assumption is that candidate roles with a significant number of members, have a higher chance of being real roles.

Furthermore, Vaidya *et al.* propose a role mining mechanism that utilizes a subset enumeration algorithm to infer access roles from user-permission assignments [11]. The algorithm examines all possible subsets of existing permissions and assigns a score to each. Subsets are then sorted based on their scores, so that better candidates sit on top of the list and thereby expert review becomes easier. Similar to Schlegelmilch *et al.* [9], the ranking is based on the number of members in each candidate role. Subset enumeration has exponential complexity and the algorithm is not feasible unless for very few number of permissions. A faster yet less accurate version of the authors' other algorithm, named FastMiner, is proposed that finds candidate roles based on intersecting user permissions and has $O(n^2)$ complexity.

Our work significantly differs from other role mining approaches in the choice of source data. Our approach utilizes permissions that are actually practiced by the system users as opposed to making use of user-permission assignments that may not necessarily reflect reality. Further arguments in favour of this choice are given in Section 2.1. Before proceeding to that section, we argue for role mining over a top-down role engineering in Section 1.1.

1.1 Role mining vs. top-down role engineering

Importance of role mining has been extensively addressed in the literature. First, role engineering is a time-consuming and a very costly phase, possibly the most expensive phase, of adopting the RBAC paradigm [5]. This partially arises because of the large variety of expertise required. Thus, much information sharing is necessary and many different authorities must be involved, so the scope of the problem rapidly exceeds what can be accomplished manually [10]. It may also entail detailed examination of organizational units, business processes, job positions, and job functions [7]. Thus, role engineering is the most problematic impediment in adopting RBAC [5]. Therefore, developing any automatic or semi-automatic tool to accelerate or circumvent this phase is very valuable. Organizational obstacles to adopting RBAC are another important motivation for using role mining. Since role engineering is a prelude to transparent permissions and job definitions, which often implies tighter compliance with the principle of least privilege, an organization's staff tends to be reluctant to cooperate with the role engineering team. This is particularly true if they feel they may lose their current access privileges that have been gained and accumulated in the course of time by negotiation and persuasion [9]. Therefore, the ability to perform such tasks with less human intervention and in a less interactive manner is very important.

Finally, role-mining is the ability to discover the system's real behavior. Role engineering, as a top-down approach, is normally based on studying documented system artifacts and its operational units. Yet, the system may not necessarily behave as documented. For example, there may be some obscure documented privileges that are not being used by the system users any more, if they ever were. Role mining, as a bottom-up approach, provides the opportunity to base role engineering on the system's real behavior and to carry out more realistic role engineering —particularly when there is a gap between documentation and reality.

1.2. Organization of this work

The rest of this work is organized as follows. Section 2 describes our approach, the design rationale, and the underlying algorithms. Section 3 illustrates a validation methodology followed by our simulation setup and discusses the experimental results of the proposed role mining approach. Finally, Section 4 summarizes our contributions and briefly introduces next steps to pursue this approach further.

2. Our approach

In this section, we introduce a role mining scheme, including the choice of source data, underlying algorithms, and design rationale.

2.1. The choice of source data

The choice of source data for role mining has not been considered much in the literature. Although it has been treated as one of the role mining steps [6], almost all proposed approaches use current user-permission assignments as the input data for role mining with hardly any discussion or rationalization. Vaidya *et al.* formulate the source data in the form of a binary matrix [12], in which rows correspond to users, columns correspond to permissions, and the intersections determine whether a user is granted the corresponding permission.

In our approach, we exploit the *access history log* (AHL), which contains permissions that users have actually practiced during the system operations, as the source data. We assume that the access history log is typically a sequence of triplets in the form of <*user, permission, timestamp*>, each of which is called *log entry* and shows an access event in the system. An AHL shows the history of how users have practiced their permissions over time. We also define AHL_U as a projection of AHL containing only those entries that pertain to user *U*. In our approach, we concentrate on the AHL_U of each user, independent from those of the others, and therefore, we ignore the relations between AHL_U's of different users as well as all information related to the concurrent nature of an AHL.

It is important that the given AHL reflect a comprehensive sequence of log entries in terms of all users for all their effective permissions. Otherwise, the result may not reflect a correct set of roles. We assume that the given AHL reflects the actual behavior of the system. Questions of how to gather such a comprehensive sequence or how to identify whether or not an AHL is good enough are beyond our context, as this paper's contribution is the development of the required algorithms. Intuitively, larger amount of log will produce better results. In general, lengthening the sampling period and adding information from samplings that are more diverse have proven to improve the results in our experiments.

We emphasize that this choice of source data has advantages over the other options, as follows:

Eliminating unused permissions. By choosing the access history log as the source data, the effect of legacy permissions that are obtained and retained unnecessarily—called *accumulated permissions* by Schlegelmilch and Steffens [9]—are mitigated. If a user no longer needs some permission in their daily job, it is less likely that they activate that permission frequently or systematically. Hence, it may appear very rarely, if at all, in the access history log and has no significant effect in the output of our algorithm. Similarly, exceptional permissions granted in special cases cannot affect the result of role mining significantly, since users do not systematically activate them. This *purification effect* is due to the statistical nature of actual behaviour of the system. Note that we assume that the system is working as normal: no malicious user is *poisoning* the access history log with unusual behavior.

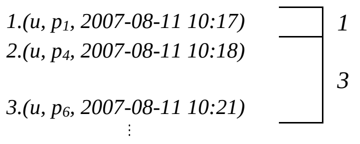
Better Choice of Heuristic. The fact that the user-permission assignments do not represent the actual information active during system run time has led other role mining approaches to make poor heuristic decisions, i.e. the number of assigned members, for ranking the candidate roles. This unfair choice may cause the roles that have fewer members, such as management roles, to have less chance to survive. By using the access history log as the source data, this is less likely to happen if such roles are activated frequently enough to be visible in the log. In our approach, we consider the difference between permission activation timestamps as a heuristic for our role conjecture. Permissions belonging to a single role are likely to be activated closely together; i.e. the temporal distance between their activation is significantly less than the temporal distance of permissions belonging to different roles. This reflects how the system is being used by its users; and from this point of view, an access history log is the trace of running business processes and job functions. Therefore, we observe that mining roles from such a log is a bottom-up approach that also reflects the higher-level system entities and takes them into account indirectly. The *bundle of rights*, which are sets of permissions needed to carry out a job function (as defined by Roeckle *et al.* [7]), are exactly what our algorithm finds in the stream of access log entries. This improves our role mining approach from a pure bottom-up to a semi-hybrid approach, in which some higher-level information is also considered to a limited extent.

2.2. A mining algorithm

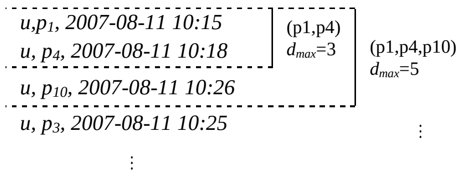
The rationale behind our algorithm is that co-occurring permissions are likely to belong to the same role. In other words, we assume that roles are sets of permissions which appear together frequently in the access history log. This is generally true, with some exceptions:

- In roles with very large number of permissions—so called *long roles*—users may activate different groups of permissions of the role at different times. This implies that the role actually comprises smaller sub-roles that are activated by the user separately.
- Roles that co-occur frequently and systematically are also an exception. Our algorithm may be misled to amalgamate such roles since their permission always appear together.

From a data-mining viewpoint, our algorithm can be thought of as a special case of *frequent itemset mining* (FIM) which is a well-studied problem in the data mining literature [1]. However, there are important differences between our version of FIM and the classic one. First, in our version, there is no known value for *minimum support*, because we do not have *a priori* knowledge of how frequent a pattern should be to be considered *frequent*. To handle this, we have implemented a score-based mechanism in which we always keep a sorted list of all top-scored itemsets, which enables us to eliminate the lowest scored itemsets if memory concerns arise. Second, in our version of FIM, there is no notion of *transaction* or *item basket*. An access history log is a stream of activated permissions; hence, there are no separate groups of permissions visible. In fact, finding groups of frequent permission sets is the final goal of our algorithm. We can assume that the border between transactions is fuzzy and they cannot be identified deterministically. One example of such an interpretation is shown in Figure 1(a). Based on this notion of fuzzy permission sets, we define *degree of cohesion* for a permission set. Intuitively, degree of cohesion denotes how strongly the members of the set are coupled. In particular, in Figure 1(a), the second entry can form a transaction with either the first or the third entry, each with a different degree of confidence, based on the temporal distance between them.


1.(u, p_1, 2007-08-11 10:17)
2.(u, p_4, 2007-08-11 10:18)
3.(u, p_6, 2007-08-11 10:21)
1
3

(a) The fuzzy nature of *transaction*


u,p_1, 2007-08-11 10:15
u, p_4, 2007-08-11 10:18
u, p_{10}, 2007-08-11 10:26
u, p_3, 2007-08-11 10:25
(p1,p4) d_{max}=3
(p1,p4,p10) d_{max}=5

(b) Maximum distance as a heuristic against cohesion

Figure 1. Exploiting AHL for role mining

Following our assumption of co-occurring permissions, we are ready to present the first version of our algorithm (Algorithm 1), inspired by the subset enumeration approach introduced by Vaidya *et al.* [11]. The complexity of this algorithm is exponential and therefore it is not a feasible choice in practice, but it can illustrate the main idea of our role mining approach. (We illustrate a faster version of this algorithm in next section.)

Algorithm 1. Basic version of the role mining algorithm

```
Inputs: Access history log (AHL)
Outputs: Inferred RBAC schema
1.  allPermissions = all permissions in AHL
2.  allUsers = all users in AHL
3.  P = Power set of allPermissions
4.  roles = new List <Role>()
5.  for each p in P do
6.      role = new Role(p)
7.      role.permissions = p
8.      role.users = users having all permissions in p
9.      role.score = degree of cohesion of p
10.     roles.add(role)
11. sort (roles) // based on the scores
12. finalRoles = new List<Role>()
13. coveredPermissions = new List<Permission>()
14. coveredUsers = new List<User>()
15. for each role in roles do
16.     finalRoles.add(role)
17.     coveredPermissions.add(role.permissions)
18.     coveredUsers.add(role.users)
19.     if coveredPermission contains allPermissions
          and coveredUsers contains allUsers break
20. return finalRoles
```

The algorithm begins by extracting the sets of users and permissions. This can be carried out trivially by scanning the access history log (AHL) and collecting all permissions and user names appeared in the log. It then continues to generate all candidate roles and assigns each role a degree of cohesion (Line 9). Candidate roles are generated by enumerating every possible set of permissions, which are all members of the power set of permissions. Degree of cohesion is used to determine how good a role is, and which roles are better. It is calculated based on the probability—or simply frequency—of coincidence of the permissions of a candidate role. By scanning AHL_U (for each user *U)*, the score of a potential role is incremented each time all permissions of that role are seen together. We also consider the maximum pair wise temporal distance of permissions as a heuristic against the cohesion of the role. Therefore, in each occurrence of permissions the score is incremented by $1/d_{max}$, where d_{max} denotes the maximum distance between two consecutive entries. This means, permission sets that are temporally closer together, receive higher scores. We do not calculate this maximum among all pairs since this would lead to an unfair decline of the score for long roles when their activation spans a longer time slice. Figure 1(b) illustrates an example of how maximum distance is calculated. If the temporal distance between two entries is very large at some points in the access history log, it can be assumed to be composed of two different logs. This can be done in a straightforward manner: e.g. by employing a clustering algorithm to dissect the sequence from its largest temporal gaps. In our implementation, we have simply treated the access history log of each day separately. Finally, members of each role are found and then a sufficient number of top-scored roles are selected to cover all permissions and users. In particular, the top-score roles are selected until the set of all permissions appear in all roles contain all permissions existing in the log, and until the set of corresponding users of all selected roles contain all users existing in the log (Lines 15-20).

2.3 Boosting the algorithm

Since the subset enumeration is of exponential order, the basic role mining algorithm presented in previous section is impractical—except for small number of permissions. However, by considering an alternative method of candidate generation, we can circumvent the subset enumeration and develop a polynomial-complexity algorithm. Accordingly, we concentrate on the access history log itself as the source of candidate roles, and instead of examining all potential roles, which requires checking a large amount of irrelevant groups of permissions, we generate candidate roles directly from what appear in the log. On this basis, we present Algorithm 2 with complexity of O(*l.n.m*), in which *l* is the length of the sequence, *n* is the number of permissions, and *m* is the number of users.

Algorithm 2 begins with separating the entries of each user from the others, i.e. extracting AHL_U for each user. It then goes through each AHL_U and gathers neighboring permissions in form of candidate roles. As a practical point, and for economy of memory usage, we maintain a large but fixed-size list of candidate roles and remove low-scored roles when the list grows beyond this fixed size. This is implemented at the end of the loop initiated in Line 7. Moreover, in Line 16, we avoid adding sub-roles of currently gathered roles, which implies preference of maximal roles. This is a practical decision that has been introduced by the well-known *a priori effect* [1], which states subsets of a frequent itemset are also frequent and hence may cause noise roles to appear in the output. This is confirmed by our experiments too (described in Section 3).

3. Simulation and Results

To validate our proposed approach, we have developed a simulation experiment. In this section, this experiment is presented and its structure is described. This simulation experiment is general and can be used for comparing other role-mining algorithms too. Figure 2 illustrates its general structure.

Algorithm 2. Improved version of our role miner algorithm

```
Inputs: Access history log  (AHL)
Outputs: Inferred RBAC schema
1.  allPermissions = all permissions in AHL
2.  allUsers = all users in AHL
3.  allRoles = new List<Role>()
4.  for each user in users
5.    s = AHL_user
6.    for baseIndex=0 to s.size
7.      for length=2 to allPermission.size
8.        ss =
              subsequence from baseIndex to baseIndex+ length
9.          role = new Role()
10.         role.permissions = permissions in ss
11.         role.users= users having all permissions in ss
12.       if allRoles.contains(role)
13.         allRoles.role.score +=
                  1/maximum distance in role.permissions
14.       else
15.         role.score =
                  1/maximum distance in  role.permissions
16.       allRoles.add(role)
17. sort (roles)  // based on the scores
18. finalRoles = new List<Role>()
19. coveredPermissions = new List<Permission>
20. coveredUsers= new List<User>
21. for each role in allRoles do
22.   if role is a sub-role of current roles in allRoles  continue
23.    finalRoles.add(role)
24.   coveredPermissions.add(role.permissions)
25.   coveredUsers.add(role.users)
26.   if (coveredPermission=allPermissions
        and coveredUsers =allUsers)  break
27. return finalRoles
```

At the first step, a random RBAC schema is generated. This step is inspired by the work accomplished by Vaidya *et al.* [11], with some modifications. The algorithm receives the number of roles, permissions and users as input, and generates a random RBAC schema in which each role is assigned a random number of random permissions. Moreover, a random number of random roles are selected for each user. To avoid extreme cases, the algorithm guarantees not to leave any role, permission, or user unassigned. This means each of the permissions is assigned to at least one role and each role consists of at least one permission, so each role is assigned to at least one user and each user has at least one role. The algorithm to generate such a random schema is straightforward. However, to make the simulation more realistic, we have defined the notion of *user profiles*. A user's profile reflects their distinguished habits and work patterns. We have used a very simple implementation of user profiles, which contains only two pieces of information: working days and permission activation pattern. Working days indicates the days of the week in which the user works (assuming the existence of part-time employees). Permission activation pattern is a factor to denote how many permission in a time unit a user activates. Fast working users, or users who have simpler jobs, activate more permission in a same unit of time. This implies that different users with different work habits and different kinds of jobs may generate different volumes of access logs for a single role. In our simulation, we generate a random profile for each user. This factor brings more randomness to the experiment and makes it more similar to reality, we assume. More complicated user profiles are also possible in which more aspects of the real system are grasped.

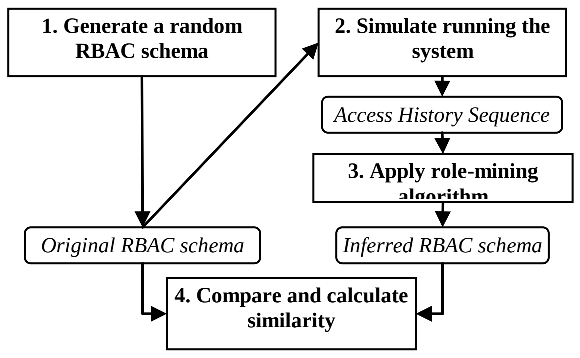

1. Generate a random RBAC schema
2. Simulate running the system
Access History Sequence
3. Apply role-mining algorithm
Original RBAC schema
Inferred RBAC schema
4. Compare and calculate similarity

Figure 2. The structure of our simulation scenario

The aim of simulation (Step 2) is to emulate user's behaviour at work. Our implementation uses a number of assumptions for the sake of simplicity. These assumptions are summarized here.

- Each working day begins from 8 A.M. to 4 P.M.
- Each user works all day long on their working days.
- Each user selects one of his/her roles at the beginning of a work day, randomly.
- At the end of each working hour, users may change their role or continue working in their current role with equal probabilities.

It is obvious that implementing these assumptions is trivial.

3.1. Evaluation scheme

To evaluate the output of our role mining algorithm, we compare the original schema with the output of the algorithm (Step 4 in Figure 2): the greater the similarity, the more accurate the algorithm.

We have developed a scheme for measuring the similarity between two RBAC schemas, by which we can calculate the degree of similarity as a real number between 0 and 1. Comparing two RBAC schemas involves comparing two sets of roles. Naively, we can consider the number of common members (the intersection of two sets) as a preliminary criterion to measure their similarity. For example, if two sets of roles have three roles in common, they are more similar than two sets with only one common role. However, a more sophisticated criterion should consider the internal structure of roles as well. Consider the following two role sets:

$r_1=\{p_1,p_2,p_3,p_4\}$, $r_2=\{p_5\}$, $r_3=\{p_1,p_2,p_3$, $r_4=\{p_6\}$
$R_1=\{r_1, r_2\}$, $R_2=\{r_3, r_2\}$, $R_3=\{r_4, r_2\}$

It is obvious that $|R_1 \cap R_2| = |R_1 \cap R_3|$. Yet, it is also obvious that the two pairs do not have identical similarities. Although r_2 is a common member in all the three sets, r_1 differs from r_3 only in a single permission, while r_4 is utterly a different role. Therefore, the simple intersection cannot accurately show the similarity of two role sets.

Since all roles are subsets of the set of all permissions, by assuming each permission a dimension, each role can be thought of as a vector in an *n*-dimensional space. Simple Euclidean distance can then be used as the criterion for comparing two roles. To compare two sets of roles, namely R_1 and R_2, the nearest members of R_1 to each role in R_2, and then the nearest members of R_2 to each role in R_1 are found. We calculate the distance in both ways, once from R_1 to R_2 and once from R_2 to R_1, in order to make the relation symmetric in case the sets do not have equal number of members. In addition, distances are normalized by dividing them to the maximum possible Euclidean distance in the *n*-dimensional space, which is the square root of *n*. Finally, the average of all distances is taken as the distance between the two sets. This scheme is depicted in Formula 1. $r_i^{p_j}$ is 1 if p_j is a member of r_i, and 0 otherwise . For instance, in the above example, distances may be calculated as $d(R_1,R_2)$=0.223 and $d(R_2,R_3)$=0.447, which clearly reflects our intuition about their similarity.

$$d(R_1,R_2)=\frac{\sum_{r_1\in R1}\frac{\min_{r_2\in R_2}\left(E(r_1,r_2)\right)}{E_{\max}}+\sum_{r_2\in R2}\frac{\min_{r_1\in R_1}\left(E(r_1,r_2)\right)}{E_{\max}}}{|R_1|+|R_2|}$$

$$E(r_1,r_2)=\sqrt{\left(r_1^{p_1}-r_2^{p_1}\right)^2+\ldots+\left(r_1^{p_n}-r_2^{p_n}\right)^2}$$

3.2 Results

In this section, the results of our simulation experiments are presented. We have done three different sets of experiments to test how accuracy is changed with number of permissions, number of roles, and variance in role size.

Accuracy / Number of Permissions: In the first set, we have raised the number of permissions from 5 to 100 with steps of 5. For each value for number of permissions, we made 30 different experiments each of which with a random number of roles. This is a realistic assumption as long as we can assume a normal distribution for the number of roles. The result of this set of experiments is depicted in Figure 3(a). As the results show, the accuracy of algorithm declines as the number of permissions grows although the steep is decreasing. Falling below 50%, the accuracy is almost intolerably low when number of permissions is beyond 40-50, which implies that the algorithm is not usable in systems with large number of permissions. However, the real number of permissions in a single subsystem is normally below this limit. For instance, although the overall number of permissions in a portal is large, there are far less permissions in each site or sub-site. Therefore, the algorithm can still be used in many real cases with a divide-and-conquer approach of mining roles in independent system units, separately.

Accuracy / Number of Roles: In the second set of experiments, we have conducted a full-round experiment for four different numbers of permissions. By full-round testing, we mean testing all different possible role sizes for a fixed number of permissions (except the extreme cases of 1 and $|P|$) and averaging the results. Yet, for each role and permission pair, we have repeated the experiment 30 times and calculated the average to make the results as smooth as possible. The trend of accuracy with growth in the number of roles is depicted in Figure 3(b). As it is observable in the diagram, at first, the accuracy declines sharply as number of roles grows. However, after it exceeds 7-10, the accuracy remains almost the same, or in some cases increases. The interesting fact is that the trend is similar with different number of permissions chosen. This implies that the accuracy of the algorithm depends mainly on the number of roles and not on the number of permissions. Therefore, even in large systems in which there are a lot of permissions, the algorithm performs well if the number of roles is not large.

Accuracy / Variance of Role Size: In the third set of experiments, we have focused on the size of roles which we had guessed to be an important factor in the accuracy of the algorithm. However, after testing different role sizes, we found out that accuracy does not exhibit any visible correlation with role size. Instead, it is the role size variance that seems to have a meaningful relation with accuracy. In other words, more variance in role size results in less accurate results. In our experiment, we simulated different RBAC schemas with variable number of permissions from 5 to 70 but with only two roles. Regarding the difference between the size of the two roles and by performing a simple normalization by dividing it to the number of permissions, (which is the maximum difference possible), we came to the results depicted in Figure 3(c), which shows an apparently descending trend line. The reason behind this decline is that the access log of shorter roles is buried among the access log of longer roles. This effect is especially intensified when shorter roles are subsets of longer roles, or have significant number of permissions in common. This latter case happened frequently about long roles in our simulation experiment since permissions of each role are chosen at random and a long role have more chance to share one of its permissions with other roles.

4. Conclusion and future work

We proposed the simplest approach to role mining based on system usage information. We argued why better performance should be expected by this approach as compared to other role mining approaches appearing to date. Furthermore, we tested our algorithm using a number of experiments and discussed advantages and disadvantages.

As immediate directions to pursue this work further, more complex usage information such as different access logs, concurrent logs of different users, etc. can also be used to develop more complex role mining schemes based on running information of the system. Since role mining algorithms (including the algorithm proposed in this paper) are normally inefficient as number of permissions grow, another topic of interest for future work is to study how to apply role mining algorithm independently in different subsystems and then merge the results.

5. References

[1] Agrawal, R., Srikant, R., “Fast Algorithms for Mining Association Rules”, *Proceedings of the 20th International Conference on Very Large Databases (VLDB)*, 1994, 487-499.

[2] American National Standards Institute (ANSI), American National Standard for Information Technology, Role Based Access Control, ANSI/INCITS 359, 2004, 2004.

[3] Coyne, E.J. “Role-Engineering”, *Proceedings of the 1st ACM Workshop on Role-Based Access Control*, 1995, Article No. 4.

[4] Epstein, P., Sandhu, R.S., “Engineering of Role/Permission Assignments”, *Proceedings of the 17th Annual Computer Security Applications Conference*, 2001, 127-136.

[5] Gallagher, M.P, O'Connor, A.C., Kropp, B., “The Economic Impact of Role-Based Access Control”, *Planning Report 02-1, National Institute of Standards and Technology*, 2002.

[6] Kuhlmann, M., Schimpf, G., “Role Mining- Revealing Business Roles for Security Administration using Data Mining Technology”, *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies*, 2003, 179-186.

[7] Roeckle H., Schimpf G., Weidinger R., “Process-Oriented Approach for Role-Finding to Implement Role-Based Security Administration in a Large Industrial Organization”, *Proceedings of the 5th ACM Workshop on Role-Based Access Control*, 2000, 103-110.

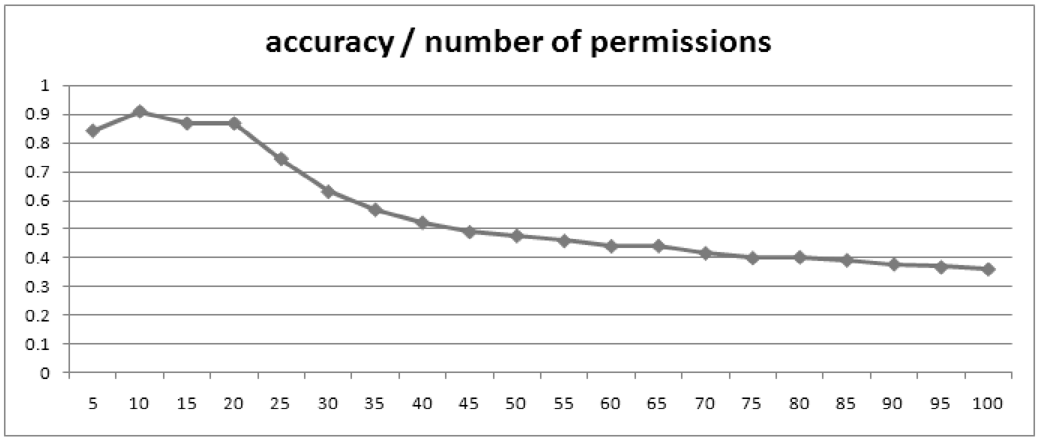
[8] Sandhu, R., Coyne, E.J., Feinstein, H.L., Youman, C. E., “Role-Based Access Control Models”, *IEEE Computer*, 29(2), 1996, 38-47.

[9] Schlegelmilch, J., Steffens, U., “Role Mining with ORCA”, *Proceedings of the 10th ACM Symposium on Access Control Models and Technologies*, 2005, 168-176.

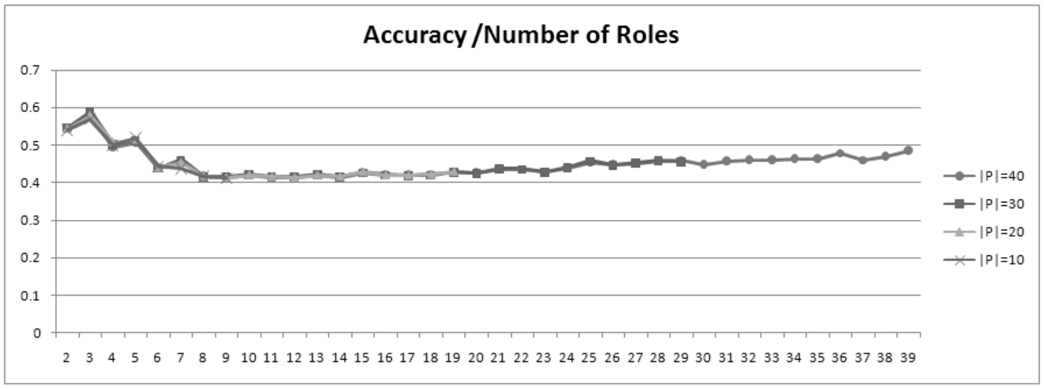
[10] Shin, D., Ahn, G.J., Cho, S., Jin, S., “On modeling System-Centric Information for Role Engineering”, *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies*, 2003, 169-178.

[11] Vaidya, J., Atluri, V., Warner, J., “RoleMiner: Mining Roles Using Subset Enumeration”, *Proceedings of the 13th ACM Conference on Computer and Communications Security*, 2006, 144-153.

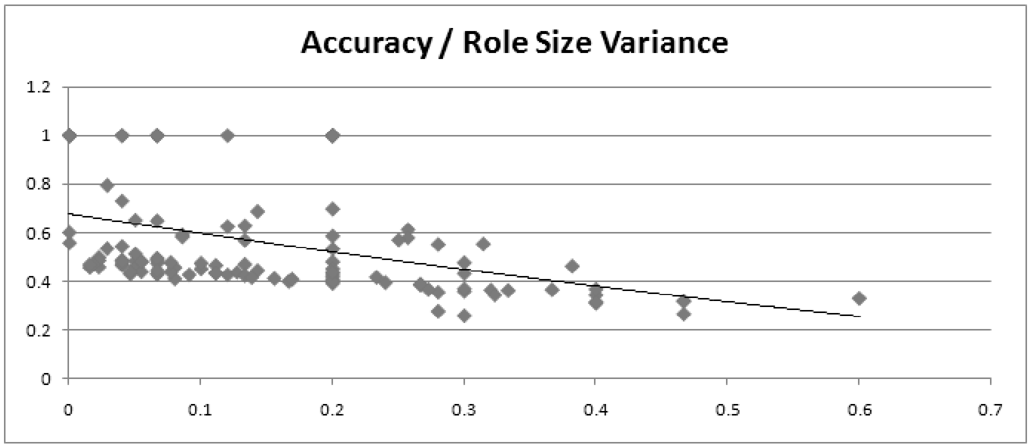
[12] Vaidya, J., Atluri, V., Guo, Q., “The Role Mining Problem: Finding a Minimal Descriptive Set of Roles”, *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies*, 2007, 175-184.



(a) Change in accuracy as the number of permissions grows



(b) Change in accuracy as the number of roles grow while the number of permissions is fixed



(c) Scatter diagram of the effect of role size variation on accuracy of the algorithm

Figure 3. Experimental Results