Mining Roles from Web Application Usage Patterns

Nurit Gal-Oz[1], Yaron Gonen[1], Ran Yahalom[1], Ehud Gudes[1], Boris Rozenberg[1], and Erez Shmueli[2]

[1] Deutsche Telekom Laboratories and Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel
{galoz,yarongon,yahalomr,ehud}@cs.bgu.ac.il, rozenbu@bgu.ac.il

[2] Deutsche Telekom Laboratories and Department of Information Systems Engineering, Ben-Gurion University of the Negev, Beer Sheva, Israel
erezshmu@bgu.ac.il

Abstract. Role mining refers to the problem of discovering an optimal set of roles from existing user permissions. In most role mining algorithms, the full set of user-permission assignments (UPA) is given as input. The challenge we are facing in the current paper is mining roles from actual web-application usage information. This information is collected by monitoring the access of users to application during a period of time. We analyze the actual permissions required to access the application in each user's session, and construct a set of user-permission assignments, which result in an incomplete UPA. We propose an algorithm that uses the session permission information to overcome the deficient data. We show by example how each step of the algorithm overcomes by heuristic instances of higher uncertainty. We demonstrate by simulation the efficiency of our algorithm in handling different levels of deficient data.

1 Introduction

Role-based access control is a common approach for authorization in web applications. The problem is that the roles originally defined are often not the ones actually used. Assume we have a web application accessing some database via a web server. The users of the application perform various actions which are translated to database operations and carried out by the web-server. In this situation the users of the application have access rights to certain modules of the application, however, their rights at the database objects level are not explicitly defined. Tracking the application users' activities allows us to learn the permissions they have in terms of access rights to database objects and create permission-profiles of users or in other words, roles. Such roles can be used to derive the actual access rights of a user or a group of users. A deviation from such a grouping may indicate an attempt for intrusion, thus it can be used for possible intrusion detection.

The challenge we are facing in the current paper is mining roles from actual web-application usage information. We monitor the permission usage habits of

S. Furnell, C. Lambrinoudakis, and G. Pernul (Eds.): TrustBus 2011, LNCS 6863, pp. 125–137, 2011.

users during a period of time. Our data is a list of monitored tuples of the type (user, set of permissions) reflecting a user's access to some database objects, possibly within a single application operation or a set of related operations. We refer to such an operation as a logical session. Each set of permissions used together within a logical session is a hint that these permissions should be granted as a group. The data we collect holds a lot of valuable information such as the set of permissions a user has, the set of users that access the same sets of permissions, the frequency of using sets of permissions together, etc.

To align with existing research on the subject of role mining we could simply create a user-permission assignment (UPA) table from all user/permissions tuples in the collected data. Following [5] we may say that this is a UPA with the presence of noise where each permission that was not demonstrated is considered to be noise. From this point on, one could apply an existing role mining algorithm such as Role Miner [6]. However, these algorithms do not take advantage of the knowledge hidden in the monitored data (e.g., session associations). Thus, we propose an algorithm which utilizes this knowledge in order to get a more accurate mapping of users-permissions needs in the application.

Since we are dealing with limited finite samples, it is very likely that rarely used permissions will not be demonstrated by all users who actually have them. This may be seen as a sort of Subtractive noise as defined by [7]. A subtractive noise refers to the case in which a permission could be incorrectly revoked i.e. when a user is only given a subset of the overall permissions he may ultimately need. However it differs from our sampling noise since we may miss permissions that were actually given due to the quality of the sampling.

To the best of our knowledge, our approach differs from the state of the art work on role mining (e.g., [6,1,2,3]) in two aspects: first, the input we use is not a complete UPA but rather a sampled set of user-permissions usage collected from logs of users' actions during system operation. Second, we use the concept of logical sessions to add semantics to the role mining process. Our goal is to reconstruct the roles which best match the monitored data taking into account the above special characteristics.

The rest of this paper is organized as follows. In section 2 we discuss related work that serves as the background for our work. In section 3 we describe the Smart Database Audit (SDA) role mining problem and in section 4 we present our approach for solving it. In section 5 we present experimental results and in section 6 we conclude and discuss further research directions.

2 Related Work

The role mining problem (RMP) was defined by Vaidya et al. [4] as the problem of discovering an optimal set of roles from existing user permissions. Their definition of RMP bound the approximation produced by an inaccurate number of roles found: Given a set of users U, a set of permissions P, and a user-permission assignment UPA, find a set of roles, R, a user-to-role assignment UA, and a role-to-permission assignment PA $\delta - consistent$ with UPA and minimizing the number of roles, k.

In most role mining researches the user-permission assignments (UPA), are the basic information given as input to any RMP algorithm. This information is derived from actual permissions given at the database level and the task of RMP is to find the optimized set of roles that can cover the UPA. Vaidya et al. [5] have defined a noise model for the role mining problem which refers to errors in the UPA table as noise. Additive noise which refers to permissions incorrectly given (e.g., a permission given to a user to accomplish some task, but was not revoked after the task/duration is complete). The second is Subtractive noise which refers to permissions incorrectly revoked e.g., a user is only given a subset of the overall permissions he may ultimately need. To minimize the effect of noise, instead of bounding the approximation, and minimizing the number of roles in RMP, they suggest doing the reverse - bound the number of roles, and minimize the approximation- Minimal Noise Role Mining Problem (MinNoise RMP). CompleteMiner algorithm [6] starts by creating an initial set of roles from a known set of User-Permissions. Then it computes all possible intersection sets which results exponential running time algorithm. A more practical algorithm presented in is FastMiner algorithm which identifies only a subset of the potential roles, very fast (complexity is only n^2). The authors argued that the roles discovered by this algorithm are sufficient for practical purposes.

A key challenge that has been raised by Molloy et al. [2] is how to discover roles with semantic meanings. They argue that roles that are discovered by existing role mining approaches are no more than a set of permissions and it is unclear whether such roles correspond to any real-world concepts, such as a job position or a work location. Without semantical meanings, such roles may be hard to use and maintain in practice. Molloy et al. [2] study the problem in two settings with different information availability. When the only information is user-permission relation, they propose to discover roles whose semantic meaning is based on formal concept lattices. When user-attribute information is also available, they propose to create roles that can be explained by expressions of user-attributes. Since an expression of attributes describes a real-world concept, the corresponding role represents a real-world concept as well. Our approach is somewhat similar to [2] in that we use the concept of a session as representing the semantics of a role.

3 The SDA Role Mining Problem

The information we have for constructing a UPA is derived from monitoring user activities for a period of time. In contrast with [4] there is no guarantee that the data we have actually demonstrates each and every permission that should be granted to a user in the application. It is possible for example, that users will have the permission to view information related to their employment but only some of them actually access this information regularly. A few others prefer to call the HR department and ask for the information by phone. In this case we will not be able to capture the fact that the latter have the required permission since they don't use it. The subtractive noise in our problem is much more difficult to realize as it is highly dependent on the quality of the sampled data.

Given a deficient dataset our task is to come up with assignments that were not explicitly observed, but are implied from the data. However an assignment of a role to a user can be a mistake if this role grants the user a permission that is not truly hers. Thus, inferring the user roles is prone to error due the subtractive noise in the input data and we define its effect on the resulting roles in terms of possible mistakes. We aim at minimizing the set of resulting roles while also minimizing the effect of noise, i.e., the amount of mistakes.

3.1 Semantic Information and Logical sessions

A logical session is part of a user login session in which the user performs a single logical operation. For example a student may ask to register to a course. The registration is a single logical operation although it may consist of several database actions such as: check if the student is allowed to register this course academically, check if the student has payed the registration fees in the current semester, etc. These actions may access several database objects and require various permissions related to these objects. One may think of a logical session as an operation a user may carry out using her permissions in a specific role. In this sense the semantics are hidden behind the gathering of actions within a single unit.

Following [2], we adopt the idea of a lattice however we do not use the concept lattice since concepts relates to maximal itemsets and we are looking for the information hidden in the usage of partial sets of items. In the next section we describe the initial lattice and our heuristic approach to mine roles from it.

4 The SDA Role Mining Solution

One of the major goals of the SDA project is to identify logical sessions by monitoring web applications (the session identification problem is out of the scope of this paper). The input we get from the logical session identified, consists of a set of tuples (one for each session), containing a session number, a user id and a set of permissions. We represent this input in a table in which each row represents a session and there is a column for each permission identified in the system. The cell representing session i and permission j contains the value 1 if the permission was used within this logical session and 0 otherwise. Figure 1 describes an example input table. Due to space limitations we grouped together sessions in which the same user used the same permissions. Table (a) is sorted by users and the support column states the number of times a session of this type occurred. Table (b) is sorted by the size of the permissions set used within the sessions. Our solution is composed of several stages. In each stage we attempt to reduce the number of roles while trying to minimize the amount of potential mistakes. In the first stage we construct the *Permission Usage Lattice (PUL)*. A node in a PUL represents a set of permissions that appeared within a session. Each node in the lattice is attached with a usage-table that details the users that had such logical sessions and the number of times they had it, i.e. support. A node SN in the lattice is a sub-node of another node N if

User	P1	P2	P3	P4	P5	P6	Support
1	x						1
1				x			2
1	x	x					6
1	x	x	x	x			3
2	x						2
2	x	x					2
2	x					x	2
2	x	x				x	3
2	x	x	x	x			2
3	x						1
3				x			1
3	x					x	1
3				x	x		2
3	x	x	x	x			2
4	x						1
4				x	x		2
4	x	x				x	5
5					x		1
6					x		2
7					x		5
7	x	x					3
7				x	x		3
7	x	x				x	6

(a)

User	P1	P2	P3	P4	P5	P6	Cardinality
1	x						1
3	x						1
4	x						1
2	x						1
3				x			1
1				x			1
5					x		1
6					x		1
7					x		1
2	x	x					2
7	x	x					2
1	x	x					2
3	x					x	2
2	x					x	2
3				x	x		2
4				x	x		2
7				x	x		2
2	x	x				x	3
4	x	x				x	3
7	x	x				x	3
2	x	x	x	x			4
3	x	x	x	x			4
1	x	x	x	x			4

(b)

Fig. 1. Permission usage: (a) sorted by users (b) sorted by the cardinality of permissions in a session

$setofPermissions(SN) \subset setofPermissions(N)$. A node N of cardinality i is connected to its sub-node SN of cardinality $j < i$ if there is no other sub-node SN' of N with cardinality k, $j < k < i$ s.t, N is connected to SN' and SN' is connected to SN. In this case SN is a *DirectSon* of N.

Figure 2 presents the PUL created from the input in Figure 1 (b). The usage table in each node lists the users that had sessions represented by this node and for each user it stores two values: original support and dynamic support. Original support counts the number of times (sessions) the user used the set of permissions represented by this node. Dynamic support represents the updated support in later stages of our role mining process. There are two measures we define in this respect. The first is *Total Role Support*, which sums up the total actual support of the role. The second is *Users per Role* which counts the number of active users (users with support > 0) that appear in the node's usage table. For each of the two measures we set a threshold: $MinSupport$ which defines the minimum required value of $TotalRoleSupport$, and $MinUsers$ which defines the minimum required value of $UsersperRole$ for any given node. These thresholds will serve us in the role mining process to determine the importance of keeping a node as a role on its own. Algorithm 1 describes the lattice construction phase.

A node in the PUL represents an *Initial Role Candidate*. In rare cases the PUL can represent the optimal set of Roles. In most cases an operation requires part of the set of permission in a role. Therefore, our goal is to use some statistical measures to determine which nodes in the lattice should become permanent roles and which nodes should be eliminated or incorporated in other nodes.

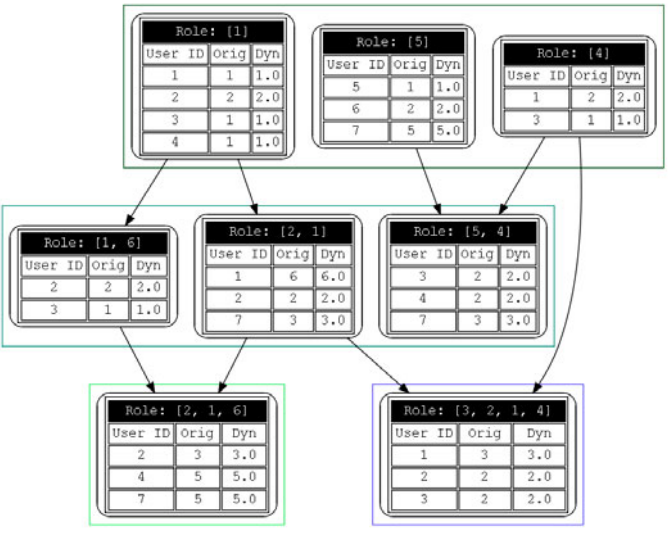

Fig. 2. Permission Usage Lattice (PUL)

4.1 Generalizing Initial Roles

Preserving all sets of permissions that appeared in the application log data may result with a large number of initial role candidates. In the worst case it may lead to a PUL with 2^n nodes, where n is the total number of permissions. However, aiming at a significantly smaller number of roles, our first steps is towards reducing the number of partial roles which are roles whose set of permissions is contained in the set of permissions of some other, more general role. For example, a teaching assistant would require a subset of the permissions that a professor requires. To do this we eliminate nodes that represent partial roles, i.e. nodes that capture a subset of permissions of other nodes. The roles represented by these nodes are *generalized* into wider roles. Intuitively it means that a user may not always use the complete set of permissions enabled by a role she own but only a subset of these permissions. This will result in two initial roles where one of them is a subset of permissions of the other one, while in fact the latter is sufficient. We say that eliminating the former is a legal operation since it does not grant the user any additional permission.

The generalization stage is a BFS scan of the lattice. The *Level* of a node is defined by the cardinality of its permission set, e.g., a node that consists of two permissions $\{P1, P2\}$ will be of level 2. We start with nodes of the lowest level and attempt to move the support of each user in this node to nodes of higher level which are ancestors of this node. Once we find the set of ancestor nodes which contain the user in their usage table, we divide the support of the user in the original node among them according to the relative support that user has in each ancestor node (the support of each ancestor node is proportional to the

Algorithm 1. Constructing the Permission Usage Lattice
Input
Partial User Permission Assignment Table pUPA
Output Permission Usage Latice PUL

```
1: sort pUPA by the cardinality of the permissions set in the sessions
2: for i=1 to MaxCardinality do
3:    level(i)= createLatticeLevel(i)
4:    for each session s of cardinality i, associated with user u and permission set p
      do
5:       if there is no node N s.t. N.setofPermissions = p then
6:          Node = level(i).AddNewNode
7:          Node.setofPermissions = p
8:          Node.usageTable.insertUser(u)
9:          Node.usageTable.users(u).support = 1
10:         Connect Node to all of its direct sons
11:      else
12:         {A node for the set of permissions p already exists}
13:         if u ∉ Node.usageTable then
14:            Node.usageTable .insertUser(u)
15:            Node.usageTable.users(u).support+=1
```

probability that the user used this role). Nodes left with a usage table having a total support of 0 may be ignored. The generalization process is described in Algorithm 2.

The example in figure 3 demonstrates the result of the generalization stage of the PUL described in figure 3. The usage table has two columns, the left column keeps the original support values and the right column keeps the current dynamic support for each user, that is, it takes into account the support added upon user node generalization.

4.2 Eliminating Redundant Roles

While generalization stage may eliminate a lot of roles, some roles may be left with a few users or a very low node support. To decide whether a node should be kept to represent a role we have to define some reasonable thresholds. Assume we have a node N representing permissions P1 and P2 and its usage table containing user u with support 2. All other users has support 0 (they were assigned a wider role in the generalization stage). User u does not appear in any other node's usage table. Is it reasonable to have node N representing a role for just one person? Should we assign user u to a wider role even if we have no explicit evidence that the user has the extra permissions in the wider role? Although there is no one clear answer to these questions we know that keeping roles that are too specific for one or even a few persons can cause over fitting and we may end up with a lot of roles($2^{|permissions|}$). In addition we know that it is very likely that some users will not use all of their permissions within the monitored period of time.

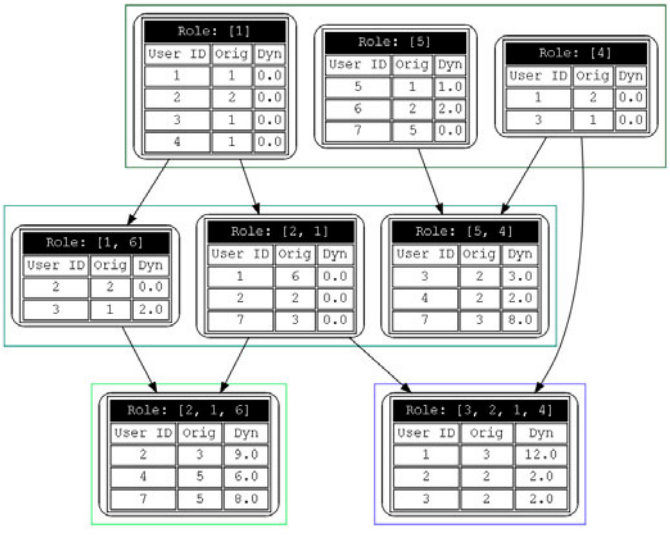

Fig. 3. Generalized Lattice

In this stage we attempt to eliminate redundant roles. A *Redundant Role* is a node in the PUL for which $UsersPerRole < MinUsersPerRole$ or $RoleSupport < totalRoleSupport$. Once we have identified a node N to be a redundant role we search for the best wider roles to assign the users left in the redundant role.

The best wider role for a user u must be an immediate ancestor of N (a node in the closest level that exists for N) to ensure minimum extra permissions assigned to u. In addition it should be a role that is most likely to be assigned to the user. The *Common Node Users (CNU)* of u is the set of users that appeared together with u in the same nodes denoted
$CNU(u) = \{u_i | \exists\ node\ N\ s.t.\ u \in N\ and\ u_i \in N\}$

A *MatchingLevel* of N_{super}, an ancestor of node N with respect to user u, is the number of users of N_{super} that appear in $CNU(u)$. Intuitively it defines to what extent we are implied that node N_{super} represents a role that should be granted to u. We take this strategy one step further and search for the MatchingLevel of the the sub-lattice whose root is N_{super} with respect to u. This extension allows us to learn the matching level of a node from a more global view of the lattice. A formal definition follows. A *MatchingLevel* of two sets of users A,B is $MatchingLevel(A, B) = |A \cap B|$. A *MatchingLevel* of a node N to its ancestor node N_{super} with respect to user u that appears in N's usage table is expressed by:
$MatchingLevel(N, N_{super}, u) = |users(Lattice(N_{super})) \cap CNU(u)|$
where $Lattice(N_{super})$ is the sub-lattice whose root is N_{super} and $users(Lattice)$ is the union of all the users appearing in nodes of Lattice. Let $SN(N) = \{SN_1, SN_2, ..., SN_n\}$ be a set of ancestors of node N, the $BestWiderRole(u, N)$ for user u in node N is a node $SN_i \in SN$ with the minimal node level that

Algorithm 2. Generalizing Initial Roles
Input
Initial PUL
Output
Generalized PUL

```
for i=1 to MaxLatticeLevel-1 do
  for each node N of level i do
    for each user u of N.usageTable do
      for j=i+1 to MaxLatticeLevel do
        setOfAncestros = {N_s | N_S ∈ N.ancestors.level(i) and u ∈ N_s.users}
        if setOfAncestros ≠ empty then
          assignDynamicSupportByProbability(N,u,setOfAncestros)
          N.usageTable.users(u).dynamicSupport= 0
          break
```

has the maximal MatchingLevel with N with respect to u. A formal definition follows:
$BestWiderRole(N,u) = \{SN_i | MatchingLevel(N,SN_i,u) > MatchingLevel(N,SN_k,u)$
$\forall SN_i, SN_k \in MinAncestorLevel(N)\}$
where $MinAncestorLevel(N) = \{SN_i | level(SN_i) \leq level(SN_j)\, \forall SN_i, SN_j \in SN(N)\}$
If we have more than one best wider role candidate we randomly select one of them according to the distribution of their matching levels. The stage of eliminating redundant roles is also a BFS scan of the PUL but it only scans active nodes that is nodes that have dynamic support greater than zero 0. Algorithm 2 describes this stage in details. Figure 4 describes the result PUL of our example after eliminating the redundant roles. Note that now the left most role in the middle row can be eliminated, however user 3 now was assigned to his ancestor

Algorithm 3. Eliminating Redundant Roles
Input
Generalized PUL
Output
Reduced PUL

```
BFS scan of Generalized PUL
for i=1 to MaxLatticeLevel-1 do
  for each node N of level i do
    if redundantRole(N) then
      for each user u of N.usageTable do
        widerNode=findBestWiderRole(N,u)
        widerNode.usageTable.insertUser(u)
        widerNode.usageTable.users(u).originalSupport= 0
        widerNode.usageTable.users(u).dynamicSupport=
        N.usageTable.users(u).dynamicSupport
        N.usageTable.users(u).dynamicSupport= 0
```

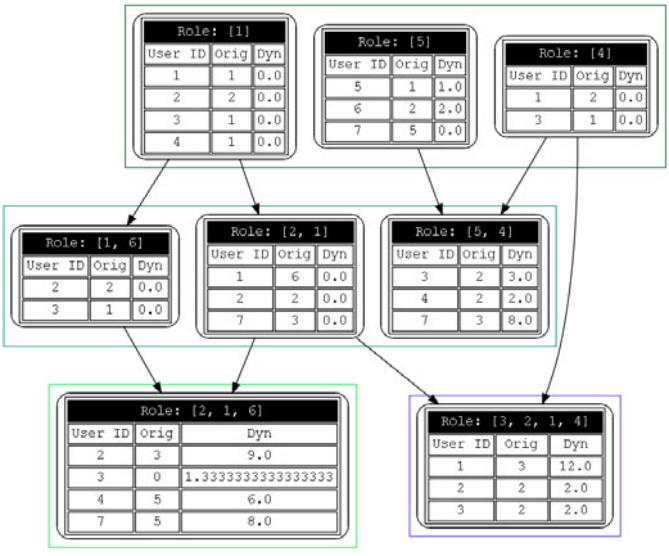

Fig. 4. Reduced Lattice

node resulting with extra permission 2. This may or may not be a potential mistake.

Within the redundant roles that we could not eliminate at the previous stages we distinguish roles that had high support in the original PUL from the others. The former roles became redundant due to our attempt to generalize them. However, the fact that we could not completely generalize them or eliminate them, and the high support they had in the initial PUL, increase the confidence that they can become permanent. In our example, if $MinSupp$ is set to 4, the role containing permission 5 meets this criteria. In the final stage we add these roles to the set of permanent roles. The set of redundant roles left consists of roles that have no justification in terms of support to become permanent.

5 Evaluation

We have developed the Session Generator, a simulation tool that generates user sessions. The session generator gets as an input a file containing roles (sets of permissions) and $maxNumOfUsers$ - the number of users in the experiment; $maxRolesPerUser$ - the highest number of roles assigned to each user; $maxSessionsPerUser$ - the highest number of sessions produced per user and $BernoulliProb$ - the probability for a permission that belongs to a role to appear in a session of this role (conforms to the Bernoulli sampling).

Using the input file, the session generator produces sessions as follows: For each user it selects a random number of roles and a random number of sessions.

It then creates a user session by randomly selecting the active role for this session from that user's roles. For each permission of the active role it uses the Bernoulli Prob parameter to decide if the permission is used in the session or not. The algorithm produces an output file containing the permissions that a user used in each session. A complete UPA (user permission assignment) matrix can be obtained by listing the permissions of each user as derived from the roles assigned to the user. Assigning a value of p to the BernoulliProb parameter in the experiment, we get an output file that covers approximately p of the permissions in the complete UPA matrix. Thus the deficiency measure (noise) is approximately $1 - p$. The session generator output file along with the threshold parameters $MinSupport$ and $MinUsers$ are the input file for the role mining algorithm. The roles dataset we use is a synthetic dataset based on a template used in [2]. We generated a session dataset based on this template by creating users and assigning roles to them using the session generator described above. The dataset contains 7 roles and 12 permissions. We generated datasets of sessions for 100 users with different values of the Bernoulli parameter ranging from 1-0.6 (constructing datasets of approximately 0 - 40% deficiency level respectively). We also used different values of the generalization threshold minNumOfUsers, minSupp ranging from 2 -20.

5.1 Evaluation Results

The results are summarized in figure 5. In this chart the horizontal axis represents the level of deficiency of the dataset. The vertical axis represents the average error measured by the ratio of the number of discovered roles with respect to the number of original roles. Each column in the chart represents an error with respect to a threshold levels.

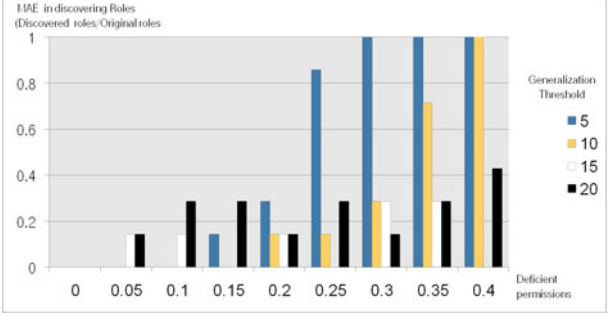

Fig. 5. MAE in discovering roles from noisy data

The majority of the tests that discovered more than 7 roles, the 7 actual original roles were discovered. Only in 4% of these tests - one of the original roles was absent. In all tests that discovered less than 7 roles, the set of discovered roles was a subset of the original roles. The 7 roles were identified from large amounts of initial role candidates for example: For 10% deficiency: 92 initial roles (1020 sessions) For 20% deficiency: 131 initial roles (989 sessions) A

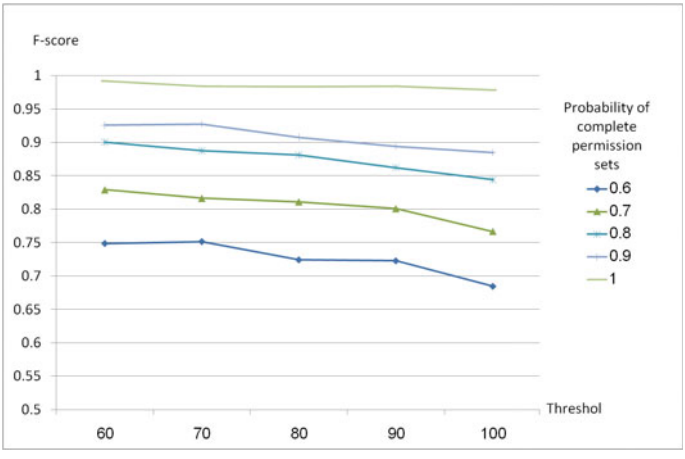

Fig. 6. F-score with respect to threshold and probability of complete permission sets

lower Bernoulli parameter = a lower coverage of the UPA, required higher levels of minNumOfUsers minSupp to discover the exact roles. Next, we used the F-score, which is defined as the the harmonic mean of precision and recall measure, to evaluate the ability of our algorithm to discover roles with respect to the original roles. Figure 6 demonstrates the correlation between the deficiency of the dataset and the Generalization threshold required to discover the original roles. This is explained as follows. The role miner algorithm generalizes redundant roles by unifying them with wider roles that contain them. A Redundant Role is a node in the Lattice for which $UsersPerRole < minNumOfUsers$ or $RoleSupport < minSupp$. If a role was not generalized although there is a wider role that contains it, it implies that $UsersPerRole >= minNumOfUsers$ and $RoleSupport >= minSupp$. By increasing the generalization thresholds, minNumOfUsers and minSupp, we allow more nodes to be generalized. The permission sets deficiency increases as the bernoulliProb parameter decreases and as a result more combinations of permission subsets are created for each role. Thus, the role mining algorithm starts with a large number of initial roles (nodes). According to the algorithm, nodes of high cardinality (relatively high number of permissions), generalize their sub-nodes of lower cardinality and get their relative support. However a higher level of generalization threshold is required to guarantee generalization. The threshold parameter in this experiment was more significant when we examined deficient permission data.

6 Conclusions

We introduced an algorithm for mining roles from data gathered by monitoring users access to a web application. The algorithm is carried out in three stages:

construction of user permission lattice, generalizing initial roles and eliminating redundant roles. We demonstrated the ability of the algorithm to overcome the major problem we have with such data, a deficient user permission assignment data. While most of the roles are discovered after the third stage applying an additional stage of accepting redundant roles with initial high support improves the results. Examining the additional roles generated beyond the original ones, we found out that these are of high cardinality. They were not generalized since they had relatively high values of support due to the many nodes generalized to them in the earlier steps of the algorithm. This implies that the support should not be unified for the nodes in the lattice and should be adjusted to the state of the lattice in each step of the algorithm. In future work we intend to examine the use of dynamic threshold values that are determined according to the state of each node traversed in the lattice.

References

1. Steffens, U., Schlegelmich, J.: Role mining with orca. In: SACMAT 2005: Proceedings of the Tenth ACM Symposium on Access Control Models and Technologies. ACM Press, Stockholm (2005)
2. Molloy, I., Chen, H., Li, T., Wang, Q., Li, N., Bertino, E., Calo, S.B., Lobo, J.: Mining roles with semantic meanings. In: SACMAT, pp. 21–30 (2008)
3. Molloy, I., Li, N., Qi, Y. (A.), Lobo, J., Dickens, L.: Mining roles with noisy data. In: SACMAT, pp. 45–54 (2010)
4. Vaidya, J., Atluri, V., Guo, Q.: The role mining problem: finding a minimal descriptive set of roles. In: Proceedings of the 12th ACM Symposium on Access Control Models and Technologies, pp. 175–184. ACM, New York (2007)
5. Vaidya, J., Atluri, V., Guo, Q., Lu, H.: Role mining in the presence of noise. In: Foresti, S., Jajodia, S. (eds.) Data and Applications Security and Privacy XXIV. LNCS, vol. 6166, pp. 97–112. Springer, Heidelberg (2010)
6. Vaidya, J., Atluri, V., Warner, J.: RoleMiner: mining roles using subset enumeration. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, pp. 144–153. ACM, New York (2006)
7. Vaidya, J., Atluri, V., Guo, Q.: The role mining problem: A formal perspective. ACM Trans. Inf. Syst. (2010)