Role Mining in the Presence of Separation of Duty Constraints

Prasuna Sarana[1], Arindam Roy[2], Shamik Sural[1(✉)], Jaideep Vaidya[3], and Vijayalakshmi Atluri[3]

[1] School of Information Technology, IIT Kharagpur, Kharagpur, India
saranaprasuna257@gmail.com, shamik@sit.iitkgp.ernet.in
[2] Advanced Technology Development Centre, IIT Kharagpur, Kharagpur, India
arindam.roy@iitkgp.ac.in
[3] MSIS Department, Rutgers University, Newark, USA
{atluri,jsvaidya}@rutgers.edu

Abstract. In recent years, Role Based Access Control (RBAC) has emerged as the most popular access control mechanism, especially for commercial applications. In RBAC, permissions are assigned to roles, which are then assigned to users. The key to the effectiveness of RBAC is the underlying role set that is used. The process of identifying an appropriate set of roles that optimally meets the organizational requirements is called role mining. One of the most useful constraints that can be expressed in RBAC is Separation of Duty (SoD). SoD constraints allow organizations to put a restriction on the minimum number of users required to complete a critical task. However, existing role mining algorithms do not handle SoD constraints and cannot be easily extended to incorporate SoD constraints. In this paper, we consider the problem of role mining when SoD constraints are present. We develop three alternative approaches that can be applied either during or after role mining. We evaluate the performance of all three approaches on several real world data sets and demonstrate their effectiveness.

Keywords: RBAC · Role mining · Separation of duty · SMER constraints

1 Introduction

Resources are protected in organizations by providing appropriate and selective permissions to users. Traditionally, access control policies were directly specified in terms of users and their permissions. However, such an access control method increases the burden on system administrators when the number of users or permissions increases. RBAC (Role Based Access Control) [1,2] reduces this administrative overhead by assigning permissions to users through roles. Roles in RBAC make task re-assignment easier and reduce the complexity as well as chances of error compared to direct assignment of permissions to users.

S. Jajodia and C. Mazumdar (Eds.): ICISS 2015, LNCS 9478, pp. 98–117, 2015.
DOI: 10.1007/978-3-319-26961-0_7

However, RBAC is effective only when the set of roles matches the organization's functional requirements. Therefore, an important step in RBAC deployment is to define the requisite roles. This process, known as role engineering [3], can be carried out using top down, bottom up or hybrid approaches [12]. In the top down approach, roles are formed by identifying independent business processes that are associated with permissions. This approach is often difficult to handle as organizations consist of hundreds of business processes and is also known to be cost intensive. On the other hand, the bottom up approach uses the existing user permission assignments for identifying roles – a procedure referred to as *role mining* [4]. A hybrid approach [3] combines elements of both top-down and bottom-up approaches to include business process knowledge as well as existing user-permission assignment information.

Note that, the user permission assignment (*UPA*) information can be represented as a matrix, in which rows represent users and columns represent permissions. Thus, a value of *1* in the (u_i, p_j) entry of a *UPA* matrix denotes the fact that the permission p_j is assigned to the user u_i. Role mining decomposes the given *UPA* matrix into two boolean matrices: User Assignment matrix (*UA*) and Permission Assignment matrix (*PA*), of which *UA* depicts the assignment of roles to users and *PA* depicts the assignment of permissions to roles. While many different UA and PA combinations exist that can correctly specify the UPA, the main challenge for role mining algorithms is to find the UA and PA that can do so optimally. Here, optimality is in terms of some metric such as the number of roles. Several metrics have been identified in the literature [5,13].

Besides ease of administration, a key benefit of RBAC is that it also allows specification and enforcement of policies with various constraints such as cardinality, prerequisite, and Separation of Duty (SoD), which match real life situations [1]. Cardinality constraints limit the maximum number of roles a user or a permission can belong to, the maximum number of permissions a role can have or the maximum number of users a role can be assigned to. SoD is considered to be an important constraint in computer security for the prevention of fraud. Typically, an SoD constraint (also called an SoD policy) states that at least k users are required to complete a task that requires n number of permissions, for given values of k and n.

While many algorithms have been developed for role mining [4,8], none of these handle the different constraints that can be expressed in RBAC. Harika et al. [14] were the first to comprehensively address cardinality constraints in the process of role mining. However, they also do not address SoD constraints, which are actually the most important constraints that need to be enabled to eliminate fraud. In this paper, we address precisely this problem – how to identify an appropriate set of roles while also taking into consideration the existing SoD constraints.

Note that, while SoD restricts the set of users in terms of permissions, in order to implement it, RBAC uses Statically Mutually Exclusive Roles (SMER) constraints [7]. A t-m SMER constraint ensures that no user is allowed to be a member of t or more roles out of a given set of m roles. It is actually quite challenging

to convert a set of SoD constraints into a corresponding set of SMER constraints that can enforce the given SoD constraints. Therefore, it is not straightforward to just use an existing role mining algorithm to identify the UA and PA, and then to generate the SMER constraints over this to precisely enforce the given SoD policies.

Thus, our objective in this paper is to take a *UPA* matrix and a set of SoD constraints as input, and find a *UA* and a *PA* matrix consistent with the *UPA* along with a set of SMER constraints that correctly enforce the given SoD constraints while minimizing the number of roles. We develop three alternative strategies to solve this problem. The developed solutions fall into two broad categories, namely *SoD-aware* and *post-processing*, based on whether constraints are considered during or after the process of role mining.

The rest of the paper is organized as follows: In Sect. 2, we present the preliminaries necessary to understand the rest of the paper. In Sect. 3, we introduce the problem of generating SMER constraints in role mining and describe the proposed algorithms. We present the results of experimental evaluation of our work in Sect. 4. We discuss prior work related to this paper in Sect. 5. Finally, we conclude the paper in Sect. 6 and discuss directions for future research.

2 Preliminaries

We now present some of the basics of the RBAC model, SoD and SMER constraints, and Role Mining.

Definition 1 *RBAC. The Role Based Access Control (RBAC) model comprises the following components [1]:*

- *U, P, R are respectively the sets of users, permissions and roles*
- *$UA \subseteq U \times R$, a many-to-many mapping of users to roles*
- *$PA \subseteq R \times P$, a many-to-many mapping of roles to permissions*
- *Cardinality, Separation of Duty and Prerequisite constraints*

We leave out other components like sessions and role hierarchy as they are not directly related to the work reported in this paper. Also, cardinality constraints have been considered in the context of role mining in recent literature [6,9,14]. We, instead, focus on the Separation of Duty constraints. In this paper, we use the term *UA* (respectively, *PA*) to denote the user-role assignment (respectively, role-permission assignment) relation as well as its representation in the form of a boolean matrix.

Definition 2 *k-n SoD Constraint. A k-n SoD constraint states that at least k users are required together to have a given set of n permissions. It can be expressed as sod<$\{p_1,p_2,...,p_n\}$,k> where each p_i is a permission for $1 \leq i \leq n$, and n, k are integers such that $2 \leq k \leq n$.*

Typically, these n permissions are required to carry out a sensitive task and the constraint specifies that no set of k-1 users should be able to complete it. While this constraint restricts a set of users in terms of their permissions, in RBAC, users get permissions through roles. In order to implement SoD, RBAC uses Statically Mutually Exclusive Roles (SMER) constraints as defined below [7].

Definition 3 *t-m SMER Constraint. A t-m SMER constraint specifies a set of m roles and no user is allowed to be a member of t or more of these m roles. A t-m SMER constraint can be expressed as* $smer\langle\{r_1,r_2,...,r_m\},t\rangle$*, where each* r_i *is a role for* $1 \leq i \leq m$*, and t, m are integers such that* $2 \leq t \leq m$*.*

It has earlier been shown that any *t-m* SMER constraint can be represented using a set of *t-t* SMER constraints, which is defined below [7].

Definition 4 *t-t SMER Constraint. A t-t SMER constraint specifies a set of t roles and no user is allowed to be a member of all the t roles. It is expressed as* $smer\langle\{r_1,r_2,...,r_t\},t\rangle$*, where each* r_i *is a role for* $1 \leq i \leq t$*, and t is an integer such that* $t \geq 2$*.*

The problem of determining whether the *UA* and *PA* of an RBAC system together satisfy an SoD constraint has been shown to be coNP-complete [7]. Unlike SoD constraints, which restrict permissions of a set of users, SMER constraints restrict role membership for a single user, and hence, whether an SMER constraint holds in the *UA* of an RBAC system can be checked in polynomial time (*PA* is not required for checking any violation of SMER constraints). However, if SMER constraints are to be used to enforce SoD constraints, one needs to first *generate* a set of SMER constraints (using the PA) that are adequate to enforce a given set of SoD constraints.

We consider the problem of generating SMER constraints from SoD constraints as an added requirement in role mining. In this paper, we present algorithms for generating SMER constraints from SoD constraints concurrently with the process of role mining and also alternatively as a post-processing step after an initial stage of unconstrained role mining. The basic unconstrained Role Mining Problem (RMP) (i.e., role mining without any constraints) is defined as follows [4]:

Definition 5 *Basic Role Mining Problem (RMP). Given a set of users U, a set of permissions P and a user-permission assignment matrix UPA, find a set of roles R, a user-to-role assignment matrix UA and a role-to-permission assignment matrix PA such that the UA and PA are consistent with the UPA and* $|R|$ *is minimized.*

Basic RMP has been shown to be NP-Complete [4]. Significant work [3–5,15–17] has already been done to find efficient algorithms for obtaining approximate solutions. However, as mentioned before, none of these handles SoD constraints.

3 Role Mining and SMER Constraint Generation

In this section, we formally introduce the problem of role mining in the presence of SoD constraints (*RMP_SoD*) and present our solution approaches.

3.1 Problem Definition

The *RMP_SoD* problem aims to find an appropriate set of roles that satisfy a given set of SoD constraints. Thus, it can be defined as follows:

Definition 6 *RMP_SoD. Given a set of users U, a set of permissions P, a user-permission assignment matrix UPA and a set E of SoD constraints, find a set of roles R, a user-to-role assignment matrix UA, a role-to-permission assignment matrix PA and a set C of SMER constraints such that the UA and PA are consistent with the UPA, C enforces E, and |R| is minimized.*

The following example illustrates this. Consider a set of users $U = \{u_1,u_2,u_3, u_4,u_5\}$, a set of permissions $P = \{p_1,p_2,p_3,p_4,p_5,p_6\}$, a *UPA* matrix as shown in Table 1 and a set $E = \{e_1, e_2\}$ of SoD constraints as shown below:

$$e_1 = \langle\{p_3, p_5\}, 2\rangle \tag{1}$$

$$e_2 = \langle\{p_1, p_5, p_6\}, 2\rangle \tag{2}$$

Table 1. Example UPA matrix

	p_1	p_2	p_3	p_4	p_5	p_6
u_1	0	1	0	0	1	0
u_2	0	1	0	0	1	0
u_3	1	1	0	1	1	0
u_4	1	1	1	0	0	0
u_5	0	0	0	0	0	1

After role mining, a *UA* (depicted in Table 2) and a *PA* (depicted in Table 3) would be generated along with a set $C = \{C_1 \cup C_2\}$ of SMER constraints as shown below:

$$C_1 = \{\langle\{r_1, r_2\}, 2\rangle\} \tag{3}$$

$$C_2 = \{\langle\{r_1, r_3, r_4\}, 3\rangle, \langle\{r_2, r_3, r_4\}, 3\rangle\} \tag{4}$$

A comparison of the *UA* and *PA* matrices as well as the generated set of SMER constraints (Expressions 3 and 4) with the given *UPA* matrix and the set of SoD constraints (Expressions 1 and 2) shows that the decomposition is

Table 2. Generated UA matrix

	p_1	p_2	p_3	p_4	p_5	p_6
r_1	1	1	1	0	0	0
r_2	1	1	0	1	0	0
r_3	0	1	0	0	1	0
r_4	0	0	0	0	0	1

Table 3. Generated PA matrix

	p_1	p_2	p_3	p_4	p_5	p_6
r_1	1	1	1	0	0	0
r_2	1	1	0	1	0	0
r_3	0	1	0	0	1	0
r_4	0	0	0	0	0	1

correct and the SoD policies can be enforced using the SMER constraints. It can be verified that this decomposition is the smallest correct decomposition – i.e., it is not possible to obtain a correct decomposition of the *UPA* into *UA* and *PA* using fewer roles.

3.2 Generating SMER Constraints After Unconstrained Role Mining

We now discuss several alternative *post-processing* approaches that generate SMER constraints after unconstrained role mining. The initial stage of unconstrained role mining can employ any of the existing role mining algorithms that minimize the number of roles [4,8]. In the second stage, SMER constraints are generated, methods for which are discussed in the following sub-sections. Note that for this stage, only the *PA* obtained after unconstrained role mining is used along with the given set of SoD constraints. The output of the second stage is a set C of SMER constraints.

Naïve Approach. One possibility is to simply use an existing SMER generation algorithm in conjunction with an existing role mining algorithm. We term this as the naïve approach. As discussed earlier, many role mining algorithms exist. For SMER constraint generation, Li et al. [7]. propose a method that works in two phases. The first phase translates the given SoD constraints into Role-level Static Separation of Duty (RSSoD) requirements, i.e., restrictions on permissions in SoD constraints are mapped to restrictions on role memberships, and the second phase is the generation of SMER constraints from RSSoD requirements using an *SMER-Gen* procedure [7].

Definition 7 *k-n RSSoD Requirement. A k-n RSSoD requirement states that at least k users are required together to have n roles. A k-n RSSoD requirement can be expressed as* $rssod\langle\{r_1,r_2,\ldots,r_n\},k\rangle$*, where each* r_i *is a role for* $1 \leq i \leq n$*, and* n*,* k *are integers such that* $2 \leq k \leq n$*.*

After getting the RSSoD requirements, all singleton sets of t-m SMER constraints that are minimal for enforcing the SoD policies are generated. The main drawback of this approach is that, generation of RSSoD requirements involves

finding a minimal set of roles having all the permissions in the SoD policy, which is computationally very expensive. The methods proposed in the next two subsections avoid this shortcoming.

Generation of 2-2 SMER Constraints. In the first of our proposed post-processing approaches, we generate a set of 2-2 SMER constraints required to enforce the given set of SoD constraints. The input is a *PA* matrix and a set $E = \{e_1,e_2,...,e_m\}$ of *k-n* SoD constraints. The output is a set $C = \{c_1,c_2,...,c_q\}$ of 2-2 SMER constraints such that C enforces E. Each 2-2 SMER constraint is expressed as: $c = \langle\{r_i,r_j\},2\rangle$, which means that, r_i and r_j are two mutually exclusive roles, i.e., no user is allowed to be a member of both the roles. Although 2-2 SMER constraints are expected to be quite restrictive in nature, we still consider generation of 2-2 SMER constraints as they are sufficient to enforce any enforceable SoD constraint. An SoD constraint e_i is not enforceable, if one of the following conditions hold: (i) all the permissions in e_i are assigned to a single role in the *PA* matrix and (ii) the given *UA* matrix already has user assignments violating the generated 2-2 SMER constraints.

To generate 2-2 SMER constraints for an SoD constraint e_i, we first find a set of roles S such that each role in S has at least one permission in e_i. Next, we check whether any role in S has all the permissions in e_i. If so, we declare e_i as not enforceable; otherwise, all valid pairs of mutually exclusive roles in S are generated. If two roles are mutually exclusive, then each role contains at least one mutually exclusive permission which is not present in the other role and permission set of one role is not a subset of the other.

As an example, consider an SoD constraint $\langle\{p_a, p_b, p_c\},2\rangle$ in which p_a, p_b and p_c are three permissions and at least two users are required together to have all these three permissions. If there are two roles of which one role has p_a and p_b, and the other role has p_b and p_c, then we declare the two roles as mutually exclusive as these two roles are not subsets of each other and therefore each role has at least one permission which is not in the other. This can be determined by checking two conditions, namely, (i) $assign\_perms[r_i] \nsubseteq assign\_perms[r_j]$ and (ii) $assign\_perms[r_j] \nsubseteq assign\_perms[r_i]$. $assign\_perms[r]$ contains the permissions of SoD assigned to the role r, which can be found by using the *PA* matrix.

The algorithm *2-2_SMER_Post_Processing* for generating 2-2 SMER constraints from a *PA* matrix and a set E of *k-n* SoD constraints is shown in Algorithm 1. Here S denotes the set of roles that are affected by an SoD constraint $e_i \in E$. Lines 5 to 8 of the algorithm determine whether a constraint is enforceable or not. If it is enforceable, then for every pair of roles in S, Line 10 finds whether they should be declared mutually exclusive. Line 11 verifies whether the given UA matrix satisfies the mutual exclusivity constraint. If Line 11 returns false, the SoD constraint is declared as not enforceable. The above steps are repeated for every SoD in E (Lines 3 to 18).

Generating 2-2 SMER constraints for an SoD can be done in polynomial time. If the number of roles in the set S for the SoD is m, then the time complexity of finding whether the SoD constraint is enforceable or not is $O(m)$ and the time

Algorithm 1. 2-2_SMER_Post_Processing

```
1: Required: UA, PA, a set E of k − n SoD constraints
2: C = φ
3: for each SoD e_i in E do
4:    Using PA, find a set S of roles having at least one permission in e_i
5:    if any role in S has all the permissions in e_i then
6:       Declare e_i as not enforceable
7:       continue
8:    end if
9:    for each pair of roles (r_i,r_j) in S do
10:      if assign_perms[r_i] ⊈ assign_perms[r_j] ∧ assign_perms[r_j] ⊈
         assign_perms[r_i] then
11:         if UA matrix satisfies ⟨{r_i,r_j},2⟩ then
12:            C = C ∪ {⟨{r_i,r_j},2⟩}
13:         else
14:            Declare e_i as not enforceable
15:         end if
16:      end if
17:   end for
18: end for
```

complexity for generating mutually exclusive roles is $O(m^2)$. So the total time complexity is $O(m+m^2)$, which is $O(m^2)$.

While 2-2 SMER constraints can be generated quite efficiently as compared to the naïve approach of Sub-sect. 3.2, in some cases too many SoD constraints might become non-enforceable if the UA matrix already has user-role assignments that violate the generated 2-2 SMER constraints. We next present a method which makes an estimate of the highest possible value of t for which a given SoD constraint can be enforced using t-t SMER constraints. This is expected to be less restrictive as compared to using only 2-2 SMER constraints. At the same time, it ought to be more efficient than the naïve approach.

Generation of *t-t* SMER Constraints. Our second post-processing approach also considers a PA matrix and a set $E = \{e_1,e_2,...,e_m\}$ of k-n SoD constraints as input and generates a set $C = \{c_1,c_2,...,c_q\}$ of $t-t$ SMER constraints such that C enforces E. To generate $t-t$ SMER constraints for an SoD e_i, the first step, like the previous algorithm, is to find the set of roles S so that each role in S has at least one permission in e_i. The next step is to determine whether e_i is enforceable or not. Two cases for which an SoD e_i cannot be enforced are: (i) The number of roles in the set S is less than k in e_i and (ii) At least one role in the set S has all the permissions in e_i. If e_i is enforceable, it is checked whether the permission set of any role is a subset of another role (here, by permission set we mean only those permissions of a role that are included in e_i). If so, then only 2-2 SMER constraints can be generated; else, we determine the largest value of t for which the following condition is satisfied.

$$|S| > (k-1)(t-1) \tag{5}$$

Algorithm 2. t-t_SMER_Post_Processing

```
1: Required: UA, PA, a set E of k − n SoD constraints
2: C = φ
3: for each SoD e_i in E do
4:    Using PA, find a set S of roles having at least one permission in e_i
5:    if number of roles in S is less than k of e_i then
6:       Declare e_i as not enforceable
7:       continue
8:    end if
9:    if any role in S has all the permissions in e_i then
10:      Declare e_i as not enforceable
11:      continue
12:   end if
13:   if permission set of any role is a subset of another (permission set considers only
      the permissions of e_i that are in the role) then
14:      Generate 2-2 SMER constraints similar to Algorithm 1
15:   else
16:      Find the largest value of t such that |S|>(t-1)(k-1)
17:      for each subset of roles R of size t from S do
18:         if UA satisfies ⟨R, t⟩ then
19:            C = C ∪ {⟨R, t⟩}
20:         else
21:            Declare e_i as not enforceable
22:         end if
23:      end for
24:   end if
25: end for
```

Theorem 1 given below establishes the reason for using this condition. Finally, we include every subset of t roles in S as a t-t SMER constraint in the set C. The above steps are repeated for every $e_i \in E$.

The algorithm for generating t-t SMER constraints (*t-t_SMER_Post_Processing*) is shown in Algorithm 2. Lines 5 to 8 and 9 to 12 determine whether the constraint is enforceable or not. Lines 13–14 generate 2-2 SMER constraints if it is determined that t-t SMER constraints cannot be generated. Line 16 finds the desired value of t. Lines 17–18 verify whether the UA matrix satisfies the t-t SMER constraints. If Line 18 returns false for any of the t-t SMER constraints, the SoD constraint is declared as not enforceable, else, it is included in the set of SMERs.

The time complexity for generating $t-t$ SMER constraints can be computed as follows. If the number of roles in the set S is m, then the time complexity for determining whether a constraint is enforceable or not is $O(m)$. Time complexity to determine whether to generate 2-2 SMER constraints or t-t SMER constraints is $O(m^2)$ and the time complexity for generating all combinations of t out of m roles is $O(m\ {}^nC_k)$. So the overall time complexity is $O(m^2+m\ {}^nC_k)$ which is $O(m\ {}^nC_k)$.

Theorem 1. *Given a k-n SoD constraint and a set S of roles, the largest value of t for which t-t SMER constraints can be generated to enforce the SoD constraint is given by* $|S| > (t\text{-}1)(k\text{-}1)$.

Proof. Roles in set S are the roles containing permissions required to complete a task needing separation of duty. To enforce a k-n SoD constraint, the task needs at least k users to get all the n permissions through the roles in S. So, we need to find a value of t such that even if we assign $(t\text{-}1)$ distinct roles to $(k\text{-}1)$ distinct users, these $(t\text{-}1)(k\text{-}1)$ roles should not be equal to the number of roles in S, i.e., $|S| \neq (t\text{-}1)(k\text{-}1)$.

So, after assigning $(t\text{-}1)$ roles to $(k\text{-}1)$ users, at least one role should be left in S which is assigned to the k^{th} user. Hence, $|S| > (t\text{-}1)(k\text{-}1)$.

It may be noted that, the value of t obtained as above is a conservative estimate. However, finding an exact bound for t would require a computationally expensive step of examining all possible subsets of roles actually assigned to users.

3.3 SoD-Aware Role Mining

From the discussions so far, it may be observed that, while the naïve approach of Sub-sect. 3.2 can precisely enforce the given set of SoD constraints, the 2-2 SMERs and the t-t SMERs generated in Sub-sects. 3.2 and 3.2, being more restrictive, might not be enforceable in the UA matrix (which was obtained by an unconstrained role mining algorithm from the given UPA). On the other hand, the naïve approach is computationally expensive and might not be feasible to implement in real-life applications. It may be recalled that the first step in the naïve approach is to find a minimal set of roles in the *PA* that together have n permissions corresponding to a k-n SoD constraint for generating RSSoD requirements (refer to Definition 7). For this step, initially a set of roles is determined such that, each role in the set has at least one permission in the SoD. However, all the roles are not allowed to be included in the minimal set. The roles that are not included in the minimal set have the same permissions that are covered by the roles in the minimal set. Finding that set takes a substantial amount of time.

As an example, consider that role r_i is included in the minimal set and role r_j is not included in the minimal set. One of the following three cases may arise:

Case 1: r_i and r_j have the same set of permissions. In this case, two RSSoD requirements need to be considered, one including r_i and the other including r_j.

Case 2: Permission set of r_j is a subset of the permission set of r_i.

Case 3: Permission set of r_j is already covered by the remaining roles in the minimal set.

So, if we ensure that the above three cases do not occur while forming the roles, and thus make the role mining step *SoD-aware*, we can avoid the exponential time required for finding the minimal set as mentioned above.

Algorithm 3. SoD_aware_Role_Mining

1: Required: UPA, a set E of $k - n$ SoD constraints
2: Determine $UserUnc[u]$ for all users u and $PermUnc[p]$ for all permissions p from the UPA
3: $\mathcal{U}$ represents the set of selected users and $\mathcal{P}$ represents the set of selected permissions to form a role
4: **while** there exists at least one user u or permission p with uncovered edges **do**
5: Set $\mathcal{U}=\phi$, $\mathcal{P}=\phi$
6: Select vertex v with minimum number of uncovered edges
7: **if** v is a user **then**
8: Call *UserSelected_FormRole* procedure
9: **else**
10: Call *PermissionSelected_FormRole* procedure
11: **end if**
12: **end while**

It may be noted that, these conditions could be embedded in any unconstrained role mining algorithm. In this paper, we use the Minimum Biclique Cover (MBC) based approach proposed in [8] as the unconstrained role mining algorithm and show how it can be made SoD-aware. In this approach, the *UPA* matrix is mapped to an undirected bipartite graph $G = (\{V_1, V_2\}, E)$. The two disjoint sets of vertices V_1 and V_2 in the *UPA* are U and P (the sets of users and permissions), respectively. The edge set E consists of tuples (u, p) where $u \in U$, $p \in P$ and permission p is assigned to user u in the *UPA*. The basic Role Mining Problem is mapped to the Minimum Biclique Cover finding problem for this bipartite graph. Each biclique in the minimum biclique cover represents a role. Since MBC is known to be NP-Complete, a number of different heuristics were tried in [8]. It was reported that selecting a vertex with minimum number of uncovered incident edges as the greedy choice in each iteration gives better result. We use the same heuristic in the approach presented below.

To make the MBC approach for solving RMP SoD-aware, whenever a new role *newR* is formed, for every SoD e_i, we determine which permissions of *newR* are in e_i. If the common permissions of *newR* and e_i form a subset of a previously created role r, then we modify *newR* and r as it leads to one of the three cases described above. The common permissions in *newR* and r that belong to e_i are removed and a new role is created with these permissions.

After the biclique cover is obtained and the UA and PA get created, we derive RSSoD requirements in linear time by finding the roles having at least one permission in an SoD. Finally, we generate a set of SMER constraints in which every SMER constraint is minimal for enforcing an SoD constraint. It is to be noted that both *t-m* as well as *t-t* SMER constraints might get generated using this method.

The overall procedure for SoD-aware role mining is shown in Algorithm 3 (*SoD_aware_Role_Mining*). Initially, determine $UserUnc[]$ and $PermUnc[]$ for all users and permissions. $UserUnc[u]$ contains uncovered permissions for a user

Algorithm 4. UserSelected_FormRole

```
1: for each p ∈ UserUnc[v] do
2:    Add p to 𝒫
3: end for
4: for each SoD e_i in E do
5:    Determine perms_SoD[e_i]
6:    for each role r do
7:       if perms_SoD[e_i] ⊆ assign_perms[r] then
8:          Call Modify_UA_PA procedure
9:       end if
10:   end for
11: end for
12: if 𝒫 ≠ φ then
13:   Add v to 𝒰
14:   for each user u ≠ v do
15:      if 𝒫 ⊆ assign_perms[u] and at least one element of UserUnc[u] is an element
         of 𝒫 then
16:         Add u to 𝒰
17:      end if
18:   end for
19: end if
20: Form a role with 𝒰 and 𝒫
```

u and $PermUnc[p]$ contains uncovered users for a permission p. $\mathcal{U}$ and $\mathcal{P}$ respectively represent sets of selected users and permissions for the newly formed role. Repeat the process given below until there is no vertex with uncovered edges.

Select a vertex v, which can be either user or permission, with minimum number of uncovered edges. If v is a user, then call $UserSelected\_FormRole$ procedure (Algorithm 4); else, call $PermissionSelected\_FormRole$ procedure (dual of Algorithm 4 - not shown separately). In Algorithm 4, Lines 1 to 3 find uncovered permissions of user v and store them in $\mathcal{P}$. For each SoD e_i in E, determine $perms\_SoD[e_i]$ (Line 5). It contains the permissions that are

Algorithm 5. Modify_UA_PA

```
1: Required: Role r and constraint e_i
2: for each permission p common to perms_SoD[e_i] and assign_perms[r] do
3:    Remove p from assign_perms[r] and P
4:    Add p to tempP
5: end for
6: for each user u ≠ v do
7:    if tempP ⊆ assign_perms[u] then
8:       Add u to tempU
9:    end if
10: end for
11: Form a role with tempU and tempP
```

common to $\mathcal{P}$ and SoD e_i. If there is any role r having the same permissions as in $perms\_SoD[e_i]$ (Lines 6 to 10), then call *Modify_UA_PA* procedure (refer to Algorithm 5). In Algorithm 5, Lines 2 to 5 remove common permissions in $\mathcal{P}$ and role r and store them in *tempP*. Lines 6 to 10 find the users whose permission sets are subsets of *tempP* and store them in *tempU*. Line 11 forms a role with *tempU* and *tempP*. It then returns to Algorithm 4. If $\mathcal{P}$ is not null, then find the users whose permission set is a subset of $\mathcal{P}$ and at least one permission must be uncovered (Lines 14 to 18) and store them in $\mathcal{U}$. $assign\_perms[u]$ contains the permissions assigned to the user u. Finally, $\mathcal{U}$ and $\mathcal{P}$ form a role.

4 Experimental Evaluation

All the algorithms presented in Sect. 3 have been implemented in C on a 3.1 GHz Intel i5-2400 CPU having 4 GB RAM. Nine real world data sets [8,14] shown in Table 4 were initially considered for the experiments. However, after studying the data sets, it was found that the Domino, FW1, FW2 and HC data sets cannot be meaningfully used to study the performance of role mining algorithms under SoD constraints (although they can be used for testing unconstrained role mining algorithms) since some sets of users in these data sets have all the permissions assigned to them (thus they will violate any SoD). Hence, valid SoD constraints cannot be generated from them.

Table 4. Data set details

Data sets	# Users	# Permissions	# Roles	Time (s)
Americas-large (AL)	3485	10127	423	78.78
Americas-small (AS)	3477	1587	213	6.31
APJ	2044	1164	456	5.60
Customer (Cus)	10961	284	276	4.66
Domino (Dom)	79	231	20	<0.01
EMEA (EM)	35	3046	34	0.02
Firewall1 (FW1)	365	709	69	0.11
Firewall2 (FW2)	325	590	10	0.15
Healthcare (HC)	46	46	15	<0.01

The rest of the data sets (in the form of *UPA* matrices) are given as input to the SoD-aware role mining approach. For post-processing approaches, the *UA* and *PA* obtained after applying the unconstrained MBC algorithm is given as input. Details of the number of roles generated and execution time needed using the unconstrained MBC algorithm are shown in Table 4. Since the data sets do not inherently contain any SoD constraints, we introduce different k-n SoD constraints using a simulator. In the simulator, for given values of k and n,

Table 5. Average number of SMER constraints generated by SoD-aware role mining, post-processing 2-2 SMER generation and post-processing t-t SMER generation for different number SoD constraints with different values of k and n. (a) 2-2 SoD (b) 2-3 SoD (c) 3-6 SoD (d) 5-10 SoD. Numbers in square brackets represent the percentage of SoD constraints that could be enforced.

(a)

Data set	$2-2$ *SoD Constraints*					
	20 SoDs			50 SoDs		
	SoD_aware	2-2_post	t-t_post	SoD_aware	2-2_post	t-t_post
AL	20 [100]	984 [95]	984 [95]	50 [100]	2461 [92]	2568 [92]
AS	20 [100]	441 [80]	439 [80]	49 [100]	1279 [84]	1272 [84]
APJ	20 [100]	40 [80]	39 [80]	50 [100]	124 [84]	124 [84]
Cus	19 [100]	19 [95]	15 [80]	48 [100]	48 [96]	36 [78]
EM	19 [100]	57 [80]	54 [80]	48 [100]	142 [84]	138 [84]

(b)

Data set	$2-3$ *SoD Constraints*					
	20 SoDs			50 SoDs		
	SoD_aware	2-2_post	t-t_post	SoD_aware	2-2_post	t-t_post
AL	20 [100]	3307 [100]	3306 [100]	50 [100]	7697 [98]	7694 [98]
AS	20 [100]	995 [80]	993 [80]	50 [100]	3365 [84]	3360 [84]
APJ	20 [100]	146 [100]	128 [100]	50 [100]	345 [100]	297 [100]
Cus	20 [100]	32 [60]	19 [100]	50 [100]	81 [58]	49 [98]
EM	20 [100]	172 [95]	163 [95]	50 [100]	425 [98]	404 [98]

(c)

Data set	$3-6$ *SoD Constraints*					
	20 SoDs			50 SoDs		
	SoD_aware	2-2_post	t-t_post	SoD_aware	2-2_post	t-t_post
AL	520 [100]	11334 [90]	11334 [90]	1301 [100]	31155 [88]	31155 [88]
AS	488 [100]	2120 [50]	2120 [50]	1309 [100]	5081 [48]	5081 [48]
APJ	506 [100]	553 [95]	574 [95]	1262 [100]	1498 [94]	1556 [94]
Cus	479 [100]	46 [15]	372 [100]	1196 [100]	97 [14]	929 [100]
EM	430 [100]	696 [100]	697 [100]	1103 [100]	1819 [100]	1831 [100]

(d)

Data set	$5-10$ *SoD Constraints*					
	20 SoDs			50 SoDs		
	SoD_aware	2-2_post	t-t_post	SoD_aware	2-2_post	t-t_post
AL	5281 [100]	24908 [70]	24908 [70]	13297 [100]	62058 [74]	62058 [74]
AS	4746 [100]	1307 [15]	1307 [15]	11889 [100]	3831 [16]	3831 [16]
APJ	4843 [100]	1459 [85]	1614 [85]	12299 [100]	3924 [90]	4258 [90]
Cus	4531 [100]	7 [5]	2180 [100]	11392 [100]	25 [2]	5461 [98]
EM	3022 [100]	1591 [100]	1594 [100]	7851 [100]	4168 [100]	4178 [100]

Table 6. Average time (in seconds) for the generation of SMER constraints by SoD-aware role mining, post-processing 2-2 SMER generation and post-processing t-t SMER generation for different number SoD constraints with different values of k and n. (a) 2-2 SoD (b) 2–3 SoD (c) 3–6 SoD (d) 5–10 SoD

(a)

Data set	$2-2$ *SoD constraints*					
	20 SoDs			50 SoDs		
	SoD_aware	2-2_post	t-t_post	SoD_aware	2-2_post	t-t_post
AL	236.43	139.71	144.70	484.39	366.24	379.54
AS	13.68	10.12	10.17	26.04	29.27	30.21
APJ	481.18	0.30	0.44	518.63	0.97	1.28
Cus	279.30	< 0.01	0.24	281.49	< 0.01	0.57
EM	48.32	0.02	0.02	69.77	0.05	0.06

(b)

Data set	$2-3$ *SoD constraints*					
	20 SoDs			50 SoDs		
	SoD_aware	2-2_post	t-t_post	SoD_aware	2-2_post	t-t_post
AL	246.48	505.56	509.40	531.47	1203.94	1212.81
AS	14.44	30.45	31.84	28.97	89.90	96.28
APJ	486.62	1.58	1.39	527.31	3.61	3.14
Cus	279.27	0.49	0.26	281.48	1.27	0.66
EM	56.16	0.07	0.07	85.03	0.02	0.17

(c)

Data set	$3-6$ *SoD constraints*					
	20 SoDs			50 SoDs		
	SoD_aware	2-2_post	t-t_post	SoD_aware	2-2_post	t-t_post
AL	274.23	2022.69	2054.26	663.16	5762.90	5861.89
AS	17.34	117.65	143.63	39.96	280.90	337.96
APJ	496.46	6.70	7.10	552.33	17.98	19.65
Cus	279.26	1.18	5.01	281.45	2.69	12.12
EM	72.57	0.29	0.29	131.09	0.76	0.77

(d)

Data set	$5-10$ *SoD constraints*					
	20 SoDs			50 SoDs		
	SoD_aware	2-2_post	t-t_post	SoD_aware	2-2_post	t-t_post
AL	314.58	6164.60	6267.25	844.92	14667.29	15008.82
AS	22.01	260.65	371.36	51.37	5.89	900.52
APJ	509.36	19.72	23.35	580.08	47.04	59.35
Cus	279.24	0.95	28.19	281.40	2.34	70.54
EM	90.60	0.66	0.67	176.17	1.74	0.67

Table 7. Average number of roles generated for SoD-aware role mining for different number of SoD constraints and different values of k and n (The number of roles for the other approaches is the same as that shown in Table 4. These values are repeated in the first column below for ease of referencing).

Data set	$2-2$ *SoD*		$2-3$ *SoD*		$3-6$ *SoD*		$5-10$ *SoD*	
	20 SoDs	50 SoDs	20 SoDs	50 SoDs	20 SoDs	50 SoDs	20 SoDs	50 SoDs
AL [423]	443	484	458	519	496	616	550	745
AS [213]	218	254	238	285	257	344	291	411
APJ [456]	461	473	467	481	476	505	489	531
Cus [276]	276	276	276	276	276	276	276	276
EM [34]	48	68	56	83	72	128	90	173

n permissions are chosen randomly from the set of all permissions. We study the approaches for 4 different types of SoD constraints: 2-2 SoD, 2–3 SoD, 3-6 SoD and 5-10 SoD. The number of SoD constraints considered for each of the four types are 20 and 50. Although we use real-world UPA matrices, since the SoDs are synthetically generated, the experiments were repeated 30 times for each combination of parameters. The (rounded off) mean of the results over 30 repetitions are reported in the tables included in this section.

It may be noted that, for comparative study, we had also implemented the naïve approach described in Sub-sect. 3.2 that uses Li et al.'s [7] algorithm. However, generation of RSSoD requirements was found to take an inordinate amount of time. Although, the algorithm would produce an output for small test data sets, for the data sets listed in Table 4, even after running for more than 24 h, the program did not reach completion. Hence, the results could not be meaningfully reported in this paper. On the other hand, all of the proposed approaches worked for large data sets as well.

Table 5 shows the number of SMER constraints generated by the three proposed approaches and Table 6 shows their execution time. The number of SoD constraints that could not be enforced are also reported in Table 5. From the table it is observed that, for both the post-processing approaches, i.e., 2-2 post-processing and t-t post-processing, there are a certain number of SoD constraints that could not be enforced. The SoD-aware approach, however, could enforce all the constraints. The number of SMER constraints generated by the SoD-aware approach is also, in general, less compared to the other two methods. For the cases where some of the SoDs were not enforceable in the post-processing approaches, the number of corresponding SMERs is less. The number of non-enforceable SoDs is usually more for the 2-2 post-processing approach compared to the t-t post-processing approach. A second point to note is that, some of the entries in the two columns *2-2_post* and *t-t_post* are the same, which implies that the value of t was obtained as 2 even in the t-t SMER constraint generation algorithm (Lines 13-14 of Algorithm 2).

A further observation, which is intuitively obvious, is that, the number of SMER constraints (Table 5) and time taken to generate them (Table 6) tend

to increase as the number of SoD constraints increases and the length of the SoD increases. The time taken for 2-2 post-processing and t-t post-processing approaches grow rapidly as the length of the SoDs increase. For example, while for 20 2-2 SoDs, it takes 139 s for the AL data set, the time required for 20 5–10 SoDs for the same data set is 6,164 s. The corresponding values for 50 SoDs are 366 and 14,667 s, respectively. On the other hand, the time required for the SoD-aware role mining does not vary so much with the length of the SoD. The time needed for the t-t post-processing approach is comparable with that for 2-2 post-processing. Although it might be felt that the post-processing approaches (2-2 post and t-t post) should take less time than the SoD-aware approach, the main component of execution time in the post-processing approaches goes into checking whether any of the existing users in the *UA* matrix violates the generated SMER constraint (Line 11 of Algorithm 1 and Line 18 of Algorithm 2).

Table 7 shows the number of roles generated for different lengths of SoD constraints and different number of SoD constraints. We only show the results for the SoD-aware role mining algorithm since for the post-processing approaches, the roles are already generated by the chosen unconstrained role mining algorithm and the number of roles is not changed by either 2-2 post-processing or t-t post-processing algorithms. From the table, it is also observed that, SoD-aware role mining generates more number of roles compared to unconstrained role mining using the MBC approach. This is because, extra roles are created by Algorithm 5 called from Step 8 of Algorithm 4, which in turn, is called from Step 8 of Algorithm 3, to avoid the three cases (Cases 1–3) as explained in Sub-sect. 3.3. Further, there is a significant dependency of the number of roles in the SoD-aware role mining algorithm on the length of the SoD.

By comparing results across the three tables (Tables 5, 6 and 7), one can conclude that the three approaches have their own strengths and shortcomings. While SoD-aware role mining generates more number of roles, it significantly outperforms t-t post-processing in terms of the number of generated SMER constraints, especially when the lengths of the given SoD constraints are short. Execution time for the SoD-aware role mining algorithm is also comparable with t-t post-processing when lengths of SoD constraints are less, and is significantly less when the SoD constraints are longer (except for the *APJ* and *Customer* data sets). The number of SMER constraints for 2-2 post-processing approach is much higher compared to SoD-aware role mining. Further, the post-processing approaches often end up in situations where certain constraints are not enforceable, which is their main drawback.

5 Related Work

Role mining is the problem of decomposing a given *UPA* into *UA* and *PA* matrices while optimizing a metric like the number of roles (often called the Basic Role Mining Problem - RMP). Vaidya et al. [4] formally define basic RMP and introduce two different variations of basic RMP, namely $\delta - approx$ RMP and *MinNoise* RMP. They proved all the problems to be NP-Complete. An approach

called Largest Uncovered Tile Mining (LUTM) was proposed in [4] to find roles by mapping RMP to the database tiling problem. Two algorithms named as CompleteMiner and FastMiner were proposed in [10] of which CompleteMiner uses subset enumeration to find interesting roles. As subset enumeration takes exponential time, FastMiner is used to reduce the time complexity by finding the intersection for every pair of users. Lu et al. [5] present a unified framework for modeling the optimal binary matrix decomposition problem using binary integer programming. Zhang et al. [3] use the permission assignment relation *PA* to obtain an optimal role hierarchy graph. It uses the initial *PA* and reduces the number of roles by identifying pairs of roles such that merging and splitting this pair results in a least cost graph. The work in [8] maps the basic role mining problem to the minimum biclique cover finding problem for bipartite graphs.

Some of the cardinality constraints considered during role mining include restricting the number of permissions to a user and the number of permissions to a role. Kumar et al. [9] propose the Constrained Role Miner Algorithm (CRM), which limits the number of permissions that can belong to a role. John et al. [6], propose two alternative approaches for restricting the number of roles for a user. One is the RPA (Role Priority based Approach), which prioritizes roles based on number of permissions and then limits the number of roles assigned to a user, and the other is the CPA (Coverage of Permissions based Approach), which chooses roles by iteratively picking the role having the largest number of permissions that are yet uncovered for that user. Harika et al. [14] impose role-usage cardinality and permission-distribution cardinality constraints in both concurrent and post-processing frameworks. Role-usage cardinality constraint limits the maximum number of roles a user can have and permission-distribution cardinality constraint limits the maximum number of roles a permission can belong to.

Another important constraint is Separation of Duty (SoD), which is used in computer security to prevent fraud. Li et al. [7] introduce how SoD constraints can be implemented in RBAC. It is equivalent to a post-processing approach to role mining under SoD constraints, which generates a set of SMER constraints from a set of SoDs and a given *PA* matrix such that the SMER constraints enforce the SoD constraints. Lu et al. [11] propose Extended Boolean Matrix Decomposition (EBMD), which extends BMD [5] by allowing negative authorizations for implementing SoD constraints to solve the constraint-aware role mining problem. The work presented in the current paper is the first ever attempt to do role mining in the presence of SoD constraints that generates SMER constraints.

6 Conclusion and Future Directions

We have proposed a number of alternative approaches for role mining in the presence of separation of duty constraints. Besides generating the UA and PA matrices from a given UPA matrix as done in any unconstrained role mining algorithm, we also derive a set of SMER constraints that enforce the given set of SoD constraints. After suggesting a naïve way of handling the problem using

an existing algorithm, we have introduced three new methods. Two of these generate the SMER constraints from an initial unconstrained decomposition of the UA matrix using any existing role mining algorithm. The third approach considers the SoDs at the time of role mining and the roles are formed in such a way that enforceable SMERs can be easily generated from the set of mined roles. We have evaluated the proposed algorithms on several real world data sets and compared their performance in terms of the number of SMERs, number of roles, as well as their execution time. The experiments show that the naïve approach is not at all scalable and does not work for many data sets. On the other hand, the proposed approaches were able to handle all of the standard real datasets used to evaluate role mining and are quite scalable.

In the future, we plan to consider generation of t-t SMER constraints while doing role mining. Additionally, in this paper we restricted our attention to minimizing the number of roles. In the future, we plan to consider different metrics such as the number of SMER constraints, sum of roles and SMER constraints, Weighted Structural Complexity (WSC) [13], etc.

References

1. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role based access control models. IEEE Comput. **29**, 38–47 (1996)
2. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Richard Kuhn, D., Chandramouli, R.: Proposed NIST standard for role-based access control. In: ACM TISSEC, pp. 224–274 (2001)
3. Zhang, D., Kotagiri, R., Tim, E.: Role engineering using graph optimization. In: ACM SACMAT, pp. 139–144 (2007)
4. Vaidya, J., Atluri, V., Guo, Q.: The role mining problem: finding a minimal descriptive set of roles. In: ACM SACMAT, pp. 175–184 (2007)
5. Lu, H., Vaidya, J., Atluri, V.: Optimal boolean matrix decomposition: application to role engineering. In: IEEE ICDE, pp. 297–306 (2008)
6. John, J.C., Sural, S., Atluri, V., Vaidya, J.S.: Role mining under role-usage cardinality constraint. In: Gritzalis, D., Furnell, S., Theoharidou, M. (eds.) SEC 2012. IFIP AICT, vol. 376, pp. 150–161. Springer, Heidelberg (2012)
7. Li, N., Tripunitara, M.V., Bizri, Z.: On mutually exclusive roles and separation of duty. In: ACM TISSEC, pp. 5–39 (2007)
8. Alina, E., William, H., Nikola, M., Prasad, R., Robert, S., Robert, T.E.: Fast exact and heuristic methods for role minimization problems. In: ACM SACMAT, pp. 1–10 (2008)
9. Kumar, R., Sural, S., Gupta, A.: Mining RBAC roles under cardinality constraint. In: Jha, S., Mathuria, A. (eds.) ICISS 2010. LNCS, vol. 6503, pp. 171–185. Springer, Heidelberg (2010)
10. Vaidya, J., Atluri, V., Warner, J.: Role miner: mining roles using subset enumeration. In: ACM CCS, pp. 144–153 (2006)
11. Lu, H., Vaidya, J., Atluri, V., Hong, Y.: Constraint-aware role mining via extended boolean matrix decomposition. In: IEEE TDSC, pp. 655–669 (2012)
12. Coyne, E.J.: Role engineering. In: ACM Workshop on RBAC, pp. 15–16 (1996)
13. Molloy, I., Chen, H., Li, T., Wang, Q., Li, N., Bertino, E., Calo, S.B., Lobo, J.: Mining roles with multiple objectives. In: ACM TISSEC, pp. 1–35 (2010)

14. Harika, P., Nagajyothi, M., John, J.C., Sural, S., Vaidya, J., Atluri, V.: Meeting cardinality constraints in role mining. IEEE TDSC **12**(1), 71–84 (2015)
15. Ye, W., Li, R., Gu, X., Li, Y., Wen, K.: Role mining using answer set programming. In: FGCS (2014)
16. Li, R., Li, H., Gu, X., Li, Y., Ye, W., Ma, X.: Role mining based on cardinality constraints. In: Concurrency and Computation Practice and Experience (2015). doi:10.1002/cpe.3456
17. Ma, X., Li, R., Wang, H., Li, H.: Role mining based on permission cardinality constraint and user cardinality constraint. In: Security and Communication Networks (2014). doi:10.1002/sec.1177