International Journal of Innovative Computing, Information and Control
Volume **17**, Number **4**, August **2021**

ICIC International ©2021 ISSN 1349-4198
pp. 1373–1390

ROLE-ENGINEERING OPTIMIZATION WITH MUTUALLY EXCLUSIVE PERMISSIONS CONSTRAINTS AND PERMISSION-TO-ROLE CARDINALITY CONSTRAINTS

Wei Sun and Hui Su

Center of Network Information and Computing
Xinyang Normal University
No. 237, Nanhu Road, Xinyang 464000, P. R. China
{ sunny810715; suhuixy }@xynu.edu.cn

Received January 2021; revised May 2021

Abstract. *Role-based access control (RBAC) has been regarded as one of the most popular access-control mechanisms for meeting organizational requirements, and it provides various security policies such as mutually exclusive constraints and cardinality constraints. Role-engineering technology is an effective method to discover optimal roles and construct RBAC systems. However, the role-mining scales are large, and the mining results lack interpretability. Conventional role-engineering methods do not consider the mutually exclusive constraints and cardinality constraints simultaneously. To address these issues, this paper proposes a novel role-engineering method. First, to reduce the mining scale, we convert the role mining problem into a clustering problem, adopt four tuples of clusters to generate user clusters, and reconstruct the original access control matrix. Second, to discover optimal roles, we implement role mining with the mutually exclusive constraints, and utilize the separation-of-duty constraint and the method of permissions covering in order to ensure the soundness and completeness of the mining process. Third, to further verify whether the permission-to-role cardinality constraints can be satisfied, we propose an optimization algorithm to construct an optimized RBAC system, and evaluate its performance using synthetic and real-world datasets. The experimental results demonstrate that the proposed method not only alleviates the management burdens, but also ensures the security and integrity of the system.*

Keywords: Role engineering, Role mining, Role optimization, Mutually exclusive permissions constraints, Permission-to-role cardinality constraints

1. Introduction.

With the rapid development and comprehensive application of network information technology, a large amount of information storages and exchanges are required in large-scale and complex information-management systems [1]. An increasing number of enterprises and organizations have adopted role-based access control (RBAC) as their main access-control mechanism over the last three decades, as it makes security administration more flexible and manageable [2-7]. With the successful implementation of RBAC systems, devising an accurate and effective set of roles and constructing a good RBAC system, which can satisfy actual application requirements, have become critical tasks. The bottom-up role-engineering technology [8-12] aims to migrate from non-RBAC systems to RBAC systems. It starts from the original user-permission assignments and aggregates them into roles by applying data mining techniques, which is also known as role mining and has gained considerable attention in recent years.

DOI: 10.24507/ijicic.17.04.1373

Actually, role mining is the task of clustering users with identical or similar permissions and constructing different roles with these permissions [13]. Roles containing several identical permissions are frequently assigned to users. Frequently usable roles can facilitate the management and maintenance of the system and decompose the set of users into clusters of users with different attribute properties [14]. To reduce the mining scales, it is indeed necessary to cluster users with the same attribute properties. However, due to the diversity of the attribute properties of entities and the variability of accesses, the mining scales are large and the management burdens of systems are very heavy using conventional role-mining methods.

A key characteristic of RBAC is that it allows the specification and enforcement of various types of security policies [15,16], such as mutually exclusive constraints, cardinality constraints and separation-of-duty constraints, which can reflect the security requirements of organizations and ensure the security of RBAC systems. Specifically, there are four different types of cardinality constraints: user-to-role, permission-to-role, role-to-permission, and role-to-user cardinality constraints, which limit the maximum number of roles related to users or to permissions, the maximum number of permissions a role can have, or the maximum number of users to which a role can be assigned [17]. For example, the general-manager role in a company must be assigned to only one person; ordinary users should not have too many roles, and otherwise there is the possibility for users to abuse their privileges. Another important constraint that is usually used in actual application environments is the mutually exclusive constraint. There are two kinds of such constraints [18]: mutually exclusive roles constraints, and mutually exclusive permissions constraints, which restrict the role memberships assigned to a single user, and the permission memberships assigned to a single role, respectively. For example, the account-manager and financial-auditor roles cannot be assigned to the same person; the deposit and withdrawal operations in a bank cannot be assigned to the same role. In the approaches for construction of role engineering, however, most existing methods do not consider the mutually exclusive constraints and cardinality constraints simultaneously, and the mining results lack interpretability.

To address the above issues, this paper proposes a novel method called role-engineering optimization with mutually exclusive permissions constraints and permission-to-role cardinality constraints (REO_MEPC&PRCC). In summary, the main contributions of this work are as follows.

1) To reduce the mining scales and satisfy the given mutually exclusive permissions constraints, we adopt the clustering technique with the four tuples of clusters to generate user clusters and reconstruct the original access matrix, and then implement the constrained role mining to derive optimal roles.
2) To further verify whether the permission-to-role cardinality constraints can be satisfied, we present the definition of the role-engineering optimization problem, propose an optimization algorithm, and construct an optimized RBAC system.
3) The separation-of-duty constraint is a critical security policy. We utilize the permission-oriented separation-of-duty constraint in order to ensure the soundness of the deployment of the constraints. Further, to make sure the mining results cover all the permissions while ensuring the integrity of the mining process, we propose an algorithm to dynamically adjust the unvisited permissions.

The remainder of the paper is organized as follows. In Section 2, we discuss the related work and present the necessary preliminaries. Section 3 proposes a novel research method that includes three aspects: preprocessing, constrained role mining, and role-engineering optimization. We present the theoretical analyses and running examples in Section 4, and show the experimental evaluations in Section 5. Section 6 concludes the paper and discusses future work.

2. Related Work and Preliminaries.

2.1. Methods of role engineering.

To discover interesting roles from existing permission assignments, two algorithms called the Complete Miner and Fast Miner were proposed [13]. Both the two algorithms use subset enumeration and allow overlapping roles. While the first algorithm enumerates all potential roles, its computational complexity is exponential. The second algorithm improves the mining process, and its computational complexity is remarkably reduced. Vaidya et al. [19] converted the role mining into a matrix decomposition problem and presented a definition for the basic role mining problem (basic RMP). The basic RMP has been proven to be *NP*-complete, for which several existing studies have already been done to find efficient solutions. In order to avoid an abuse of privileges, Blundo and Cimato [20] proposed a heuristic capable of returning a complete set of roles, thereby satisfying the same cardinality constraint as above. John et al. proposed two alternative approaches for restricting the number of roles assigned to a user: role priority-based approach (RPA) and the coverage of permissions-based approach (CPA). The RPA prioritizes roles based on the number of permissions and assigns optimal roles to users, according to the priority order. The CPA chooses roles by iteratively picking the role with the largest number of permissions that are yet uncovered and then ensures that no user is assigned more than a given number of roles [21]. In order to limit the maximum number of users or permissions related to a role, Ma et al. [22] proposed a role mining algorithm to generate roles based on permission cardinality constraints and user cardinality constraints. In order to simultaneously limit the maximum number of roles assigned to a user and a related permission, Harika et al. proposed two role-optimization methods: post processing and concurrent processing. In the first method, roles are initially mined without taking the constraints into account. The user-role and role-permission assignments are then checked for constraint violation in the optimization process and appropriately re-assigned, if necessary [23]. The concurrent processing method implements the optimization with double constraints during the process of role mining. In addition to these methods for satisfying cardinality constraints, Sarana et al. [24] proposed three role-optimization methods, including separation-of-duty constraints either during, or after the mining process. In order to satisfy separation-of-duty constraints and ensure authorization security, we proposed a method called role-mining optimization with separation-of-duty constraints and security detection for authorizations [25].

Two main limitations are apparent in the existing studies. The first limitation is that the role-mining scales are very large, and the management burdens of systems are very heavy. The second limitation is that, most conventional role-engineering methods do not consider the mutually exclusive constraints and cardinality constraints simultaneously. If the number of permissions assigned to some role exceeds particular value, or the number of roles related to some permission exceeds another particular value, then there is the possibility of abuse of privileges, and the system is not secure. Hence, we propose a novel role-engineering method, in order to alleviate the management burdens while satisfying the mutually exclusive constraints and cardinality constraints. We also demonstrate the effectiveness of the proposed method using synthetic and real-world datasets.

2.2. Preliminaries.

Before actually proposing our novel method, we present some preliminaries that are discussed in this work, which include the basic components of role engineering, basic RMP problem and its Fast Miner method, mutually exclusive constraints, separation-of-duties constraints, and cardinality constraints.

2.2.1. *Basic components of role engineering.* According to the NIST standard of RBAC, conventional role engineering consists of the following basic components:

1) U, P and R are the basic elements of RBAC, which represent the sets of users, permissions and roles, respectively;
2) $UPA \subseteq U \times P$ represents a many-to-many mapping relationship of user-permission assignments;
3) $URA \subseteq U \times R$ represents a many-to-many mapping relationship of user-role assignments;
4) $RPA \subseteq R \times P$ represents a many-to-many mapping relationship of role-permission assignments;
5) $user\_perms(u) = \{p | \exists p \in P, \exists r \in R : ((u, r) \in URA) \wedge ((r, p) \in RPA)\}$, which represents the set of permissions assigned to user u;
6) $role\_perms(r) = \{p | \exists p \in P : (r, p) \in RPA\}$, which represents the set of permissions assigned to role r;
7) $perm\_roles(p) = \{r | \exists r \in R : (r, p) \in RPA\}$, which represents the roles associated with permission p.

2.2.2. *The basic RMP problem and Fast Miner method.* The basic RMP [19] can be formally represented below:

$$\begin{cases} \min |R| \\ URA \otimes RPA = UPA \end{cases} \tag{1}$$

For convenience, the UPA, URA, and RPA are used to represent their respective assignment relationships, as well as the corresponding matrices. The Fast Miner method [13] mainly consists of the following steps:

1) According to the hash mapping rule, a given access control matrix is converted into the user-permission assignments relationship;
2) To reduce the size of the original data set, different users who have the same permissions in the permission assignments are grouped together, and an initial set of roles is generated;
3) All the potentially interesting roles are identified by implementing intersections between any pair of the initial roles, and an initial RBAC system is constructed.

2.2.3. *Mutually exclusive permissions constraints (MEPC).* The t-m MEPC [18] states that, given m permissions $p_1, p_2, \ldots, p_m$, no role is allowed to have t or more of these m permissions. It is expressed as $mepc < \{p_1, p_2, \ldots, p_m\}, t >$, where m and t are integers such that $2 \leq t \leq m$, which can be formalized as follows:

$$\forall r \in R : |\{p_1, p_2, \ldots, p_m\} \cap role\_perms(r)| < t \tag{2}$$

2.2.4. *Permission-oriented separation-of-duty constraints (PSOD).* The k-n PSOD [18] states that, given a task consisting of n different permissions $p_1, p_2, \ldots, p_n$, at least k roles are required together to have all these n permissions, in order to perform this task. Any $(k-1)$ roles cannot have all these permissions. It is expressed as $psod < \{p_1, p_2, \ldots, p_n\}, k >$, where each p_i is a permission, and n and k are integers such that $2 \leq k \leq n$. It can be formalized as follows:

$$\forall (r_1, r_2, \ldots, r_{k-1}) \in R : \{p_1, p_2, \ldots, p_n\} \not\subset \bigcup_{i=1}^{k-1} role\_perms(r_i) \tag{3}$$

2.2.5. *Permission-to-role cardinality constraints (PRCC).* The PRCC [17] conveys that, for the given set P of permissions, set R of roles, and threshold $MRC_{permission}$, the number of roles to which any permission can be assigned should not exceed $MRC_{permission}$. This can be formalized as follows:

$$\forall p \in P : |perms\_roles(p) \cap R| \leq MRC_{permission} \tag{4}$$

In addition, there are another three cardinality constraints in RBAC, which are not discussed in this work.

3. **Proposed Method.**

In this section, we propose a novel research method named as REO_MEPC&PRCC, which includes three aspects: 1) preprocessing for generation of user clusters, 2) constrained role mining with the MEPC, and 3) role-engineering optimization with the PRCC. Specifically, to reduce the mining scale, we define a four tuple using the clustering technique to represent and store access control information, and generate user clusters. Subsequently, we implement role mining using the mutually exclusive permissions constraints, and construct optimal roles. Last, to further satisfy the constraint requirements and enhance the interpretation of the mining results, we implement role-engineering optimization using the permission-to-role cardinality constraints. An overall view of the proposed framework is presented as shown in Figure 1.

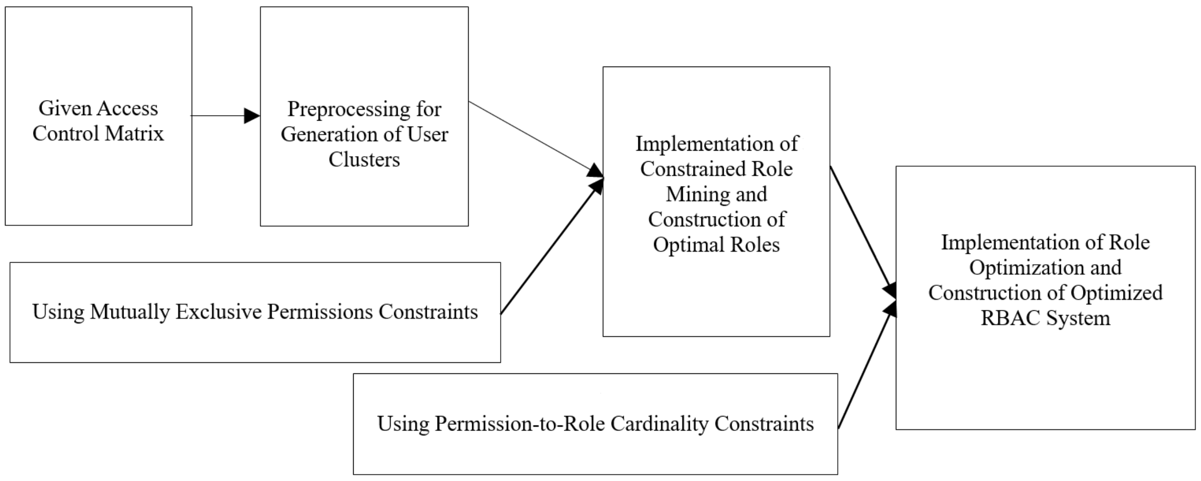
Given Access Control Matrix
Preprocessing for Generation of User Clusters
Implementation of Constrained Role Mining and Construction of Optimal Roles
Implementation of Role Optimization and Construction of Optimized RBAC System
Using Mutually Exclusive Permissions Constraints
Using Permission-to-Role Cardinality Constraints

FIGURE 1. Overview of the proposed role-engineering optimization framework

3.1. **Preprocessing for generation of user clusters.**

According to the Fast Miner method, different users with the same permissions are grouped together, which can be regarded as a user group. To reduce the mining scale, we represent the groups as different user clusters, and adopt four tuples to store them as well as other properties. This is easy to implement, and we present its definition as follows.

Definition 3.1. *(Four tuples of user clusters) Represent the users having the same permissions, as well as their properties, which is denoted as a four-tuple form* $< c, user\_perms(c), count\_users(c), count\_unvistedperms(c) >$*, where c is a user cluster,* $user\_perms(c)$ *is the permission set associated with c,* $count\_users(c)$ *is the number of users included in c, and* $count\_unvistedperms(c)$ *is the number of permissions unvisted recently in c. Obviously,* $count\_unvistedperms(c)$ *is equal to the value of* $|user\_perms(c)|$ *before role mining.*

```
Algorithm 1. Generation of user clusters.
Input: original data set $D = <U, P>$
Output: four tuples of clusters
 1. Create and initialize $Init\_Roles = \Phi$, $C = \Phi$;
 2. for each $u$ in $U$ do
 3.    if $perm\_roles(user\_perms(u))$ is not included in $Init\_Roles$ then
 4.       $Init\_Roles = Init\_Roles \cup \{perm\_roles(user\_perms(u))\}$;
 5.       $c = \{u\}$, $count\_users(c) = 1$;
 6.       $count\_unvistedperms(c) = |user\_perms(c)|$;
 7.       $C = C \cup \{c\}$;
 8.    else
 9.       $c = c \cup \{u\}$, $count\_users(c){+}{+}$;
10.    end if
11.    Sort the clusters in $C$ according to $count\_unvistedperms(c)$ in descending order;
12. end for
```

Then, we present the generation process in Algorithm 1.

In Algorithm 1, we first create and initialize two sets in line 1: *Init_Roles* and C, which are used to store initial roles and cluster users, respectively. Next, different users with the same permission assignments are grouped into cluster c, and new roles are generated and then inserted into *Init_Roles* in lines 4-9. Also, the number of users in the cluster and that of the uncovered permissions are identified in the iteration process.

Example 3.1. *Consider a matrix $UPA_{original}$ of original user-permission assignments, which comprises 15 users and 4 permissions as shown in Table 1.*

Table 1. The $UPA_{original}$

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

According to the descending order of the permissions that are associated with users, we can obtain the sorted matrix UPA_{sorted} with respect to matrix $UPA_{original}$, as shown in Table 2. As a result, it is easy to derive the corresponding matrix $CPA_{cluster}$ of cluster-permission assignments, as well as tuples of the clusters according to Algorithm 1 as shown in Tables 3 and 4.

Table 2. The UPA_{sorted}

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

Table 3. The $CPA_{cluster}$

	p_1	p_2	p_4	p_3
c_1	1	1	1	0
c_2	0	1	1	1
c_3	0	1	0	1
c_4	0	0	1	0

Table 4. Four tuples of the clusters

User cluster (c)	Initial roles ($Initial\_Roles(c)$)	Original user number ($count\_users(c)$)	Unvisited permission number ($count\_unperms(c)$)
$c_1 = \{u_2, u_4, u_5, u_{13}, u_{14}\}$	$\{p_1, p_2, p_4\}$	5	3
$c_2 = \{u_6, u_7, u_{15}\}$	$\{p_2, p_3, p_4\}$	3	3
$c_3 = \{u_3, u_8, u_9\}$	$\{p_2, p_3\}$	3	2
$c_4 = \{u_{10}, u_{11}\}$	$\{p_4\}$	2	1

3.2. **Constrained role mining with MEPC.** To discover optimal roles, according to the cluster tuples generated by Algorithm 1, we present the process of the role mining for a given MEPC $mepc < \{p_1, p_2, \ldots, p_m\}, t >$ in Algorithm 2.

In Algorithm 2, we first create and initialize several variables in lines 1-4, including *Optim_Roles*, *maxcount_users*, *cand_cluster*, and C'. For each cluster in C', we calculate the number of derived users from the cluster hierarchies, and then identify the candidate cluster and its maximal number of users in lines 5-15. Lines 16 and 17 update the *Optim_Roles*, and remove the candidate clusters. Next, for each cluster, we remove the permissions assigned to *cand_cluster*, which are covered by other clusters, and remove the clusters of which all the permissions have been covered from C' (lines 18-27).

Example 3.2. *Based on the results of Example 3.1, consider the constraint and set threshold* $t = 3$.

Algorithm 2. Role mining with MEPC.

```
Input: set $C$ of user clusters, constraint threshold $t$
Output: optimized role set $Optim\_Roles$
 1. Create and initialize $Optim\_Roles = \Phi$;
 2. Create and initialize $maxcount\_users = 0$, which represents the maximal number of
    users included in a cluster;
 3. Create and initialize $cand\_cluster = \Phi$, which represents the cluster involving the
    maximal number of users;
 4. Create and initialize a temporary cluster set $C' = C$;
 5. while $C' != \Phi$ do
 6.   for (each $c_i$ in $C'$) $\wedge(count\_unvistedperms(c_i) \leq t)$ do
 7.     for each $c_j$ in $C' \backslash \{c_i\}$ do
 8.       if $user\_perms(c_i) \subseteq user\_perms(c_j)$ then
 9.         $count\_users(c_i) += count\_users(c_j)$;
10.       end if
11.       if $maxcount\_users < count\_users(c_i)$ then
12.         $maxcount\_users = count\_users(c_i)$;
13.         $cand\_cluster = c_i$;
14.       end if
15.     end for
16.     $Optim\_Roles = Optim\_Roles \cup perm\_roles(user\_perms(cand\_cluster))$;
17.     $C' = C \backslash \{cand\_cluster\}$;
18.     for (each $c_k$ in $C'$) $\wedge$ ($user\_perms(cand\_cluster) \subseteq user\_perms(c_k)$) do
19.       for each $p$ in $user\_perms(cand\_cluster)$ do
20.         $user\_perms(c_k) = user\_perms(c_k) \backslash \{p\}$;
21.         $count\_unvistedperms(c_k) - -$;
22.       end for
23.       if $count\_unvistedperms(c_k) == 0$ then
24.         $C' = C \backslash \{c_k\}$;
25.       end if
26.     end for
27.   end for
28.   Sort the clusters in $C'$ according to $count\_unvistedperms(c)$ in descending order;
29. end while
```

TABLE 5. The mining process

Step	Maximal number of users ($maxcount\_users$)	Candidate cluster ($cand\_cluster$)	Mining roles ($Optim\_Roles$)	Updated C'
1	$2+3+5=10$	c_4	$\{p_4\}$	$\{c_2, c_3, c_1\}$
2	$3+3=6$	c_3 (or c_2)	$\{p_2, p_3\}$	$\{c_1\}$
3 (stop)	5	c_1	$\{p_1, p_2\}$	Φ

We repeatedly call Algorithm 2, and present the mining process in Table 5. As a result, the optimal role set $Optim\_Roles = \{\{p_4\}, \{p_2, p_3\}, \{p_1, p_2\}\}$, and the role-permission assignments are shown in Table 6, which can satisfy the constraints MEPC.

3.3. **Role-engineering optimization with PRCC.** To further satisfy the constraint requirements for user clusters in RBAC systems while enhancing the interpretation of

TABLE 6. The role-permission assignments

	p_1	p_2	p_3	p_4
r_1	0	0	0	1
r_2	0	1	1	0
r_3	1	1	0	0

the mining results, the PRCC should be taken into consideration in the role engineering. Specifically, the unconstrained mining results are checked to identify whether they violate the given permission-to-role cardinality constraints. If there are no constraint violations, they are regarded as efficient solutions. First, we present the definition of the role-engineering optimization problem as follows.

Definition 3.2. *(Role-engineering optimization problem) Given a matrix $CPA_{n\times m}$ of cluster-permission assignments, the preprocessed results CRA and RPA matrices, and a particular threshold $MRC_{permission}$, find an optimal set Optim_Roles of roles, such that the CRA and RPA are consistent with the CPA, the number of roles to which any permission can be assigned is less than or equal to $MRC_{permission}$, and the number of the optimal roles is minimum. This process can be formalized as follows:*

$$\begin{cases} \min |Optim\_Roles| \\ CRA \otimes RPA = CPA \\ \sum\limits_{j} RPA[j][t] \le MRC_{permission} \le |Optim\_Roles|, \quad \forall t \in [1, m] \end{cases} \tag{5}$$

According to Definition 3.2, in order to satisfy the role-engineering optimization in the presence of PRCC, we present the optimization process in Algorithm 3.

In Algorithm 3, we first define two functions in line 1: *count_perm_roles*(p) and *count_role_perms*(r). Line 2 aims to construct the initial matrices *CRA* and *RPA* from the given *CPA* and the *Optim_Roles* from Algorithm 2. Line 3 determines whether the number of roles to which any permission is assigned exceeds $MRC_{permission}$. For each violating permission p, k number of roles is represented by calculating $(count\_perm\_roles(p) - (MRC_{permission} - 1))$ in line 4, and the top k roles that possess the maximum number of permissions are chosen to constitute set S in line 5. We intersect the permissions of all the k roles into set P_S, and assign P_S to the newly created role while retaining the other $(MRC_{permission} - 1)$ roles related to permission p (lines 6 and 7). Then, the new role is inserted into the *CRA* and *RPA*. We update matrices *CRA* and *RPA* based on the new role in lines 8-22. As a result, the PRCC is satisfied with a possible reduction in the number of roles related to the violating permission, because the newly created role is used instead of the k roles that need to be intersected.

4. **Analysis of Soundness and Completeness.**

4.1. **Soundness of the constraint deployments.**

The processes of the role mining and optimization are implemented by utilizing permission-oriented mutually exclusive constraints and cardinality constraints, and it is convenient and feasible to derive optimal roles satisfying the constraint requirements from the bottom permission assignments. With the increasing development and comprehensive application for information technology, the security of RBAC systems has gained more and more attention in recent years, while it is influenced by the enforcement of the role-mining methods. The role result depends on the setting of mutually exclusive constraints, and different mining results

Algorithm 3. Role-engineering optimization with PRCC.

```
Input: matrix $CPA$, optimized role set $Optim\_Roles$, and constraint threshold $MRC_{permission}$
Output: optimized matrices $CRA$ and $RPA$
 1. Create and identify $count\_perm\_roles(p)$ and $count\_role\_perms(r)$, which represent the
    number of roles related to permission $p$, and that of permissions assigned to role $r$,
    respectively;
 2. Utilize the methods of Fast Miner and Boolean matrix decomposition to construct the
    initial matrices $CRA$ and $RPA$ from the given $CPA$ and $Optim\_Roles$, such that $CRA \otimes$
    $RPA = CPA$;
 3. while ($\exists p \in P$: $count\_perm\_roles(p) > MRC_{permission}$) do
 4.   $k = count\_perm\_roles(p) - (MRC_{permission} - 1)$;
 5.   Choose the top $k$ roles from $p$ with the highest $count\_role\_perms(r)$ values to constitute
      a temporary set $S$;
 6.   Intersect the permissions of all the roles in $S$, and denote the intersection as set $P_S$;
 7.   Create a new role $r_{nr}$ such that, $role\_perms(r_{nr}) = P_S$;
 8.   for each $u_i$ in $C$ do
 9.     if $user\_roles(c_i) \supseteq r_{nr}$ then
10.       $CRA[i][nr] = 1$;
11.     else
12.       $CRA[i][nr] = 0$;
13.     end if
14.   end for
15.   for each $r_j$ in $S$ do
16.     if $\forall p_t \in P_S$: $RPA[j][t] == 1$ then
17.       $\forall p_t \in P_S$: $RPA[j][t] = 0$;
18.       $RPA[nr][t] = 1$;
19.     else
20.       $RPA[nr][t] = 0$;
21.     end if
22.   end for
23.   Update $count\_perm\_roles(p)$ and $count\_role\_perms(r)$;
24. end while
```

perhaps breach the system security. The PSOD is widely regarded as a fundamental principle in computer security, which can detect the roles that breach the security. Thus, it is necessary to study how to deploy the MEPCs based on the PSOD in order to ensure the system security.

First, we present the definition of the safety of the system status in the following.

Definition 4.1. *(Safety of the system status, $safe_{psod}(\gamma)$) Given an RBAC system status γ and a k-n PSOD constraint $psod = <\{p_1, p_2, \ldots, p_n\}, k>$, if any $(k-1)$ roles cannot have all the n permissions under γ, then γ is safe with respect to the psod, which is denoted as $safe_{psod}(\gamma) = 1$; otherwise, γ is unsafe with respect to the psod, which is denoted as $safe_{psod}(\gamma) = 0$. Let the set of different k-n PSOD constraints be $\xi = \{psod_1, psod_2, \ldots\}$; if γ is safe with respect to each $psod_i$, then γ is safe with respect to ξ, which is denoted as $safe_\xi(\gamma) = 1$. Otherwise, γ is unsafe with respect to ξ, which is denoted as $safe_\xi(\gamma) = 0$.*

The safety of the system status can be formally expressed as:

$$\forall (r_1, r_2, \ldots, r_{k-1}) \in R : \{p_1, p_2, \ldots, p_n\} \not\subset \bigcup_{i=1}^{k-1} role\_perms(r_i) \Rightarrow safe_{psod}(\gamma) = 1 \tag{6}$$

Then,

$$\forall psod_i \in \xi : safe_{psod_i}(\gamma) = 1 \Rightarrow safe_\xi(\gamma) = 1 \tag{7}$$

Similarly, the unsafety of the system status can be formally expressed as:

$$\exists\{r_1, r_2, \dots, r_{k-1}\} \in R : \bigcup_{i=1}^{k-1} \mathit{role\_perms}(r_i) \supseteq \{p_1, p_2, \dots, p_n\} \Rightarrow \mathit{safe}_{\mathit{psod}}(\gamma) = 0 \tag{8}$$

Then,

$$\exists \mathit{psod}_i \in \xi : \mathit{safe}_{\mathit{psod}_i}(\gamma) = 0 \Rightarrow \mathit{safe}_{\xi}(\gamma) = 0 \tag{9}$$

Statement 4.1. *Given a system status γ and a PSOD constraint set $\xi = \{psod_1, psod_2, \dots\}$, the process for verifying whether γ is safe with respect to ξ is in P.*

Proof: We prove it through the following three steps: 1) identify *role_perms*(r) for role r under γ, 2) identify the number of the permissions of *role_perms*(r) present in any *psod*, which is denoted as s, and 3) compare s with k included in the *psod*. These steps can be readily computed. If $|R|$, $|P|$, and $|\xi|$ are used to represent the total number of roles, permissions, and PSOD constraints, respectively, then the computational complexity for verifying whether or not $\mathit{safe}_{\xi}(\gamma)$ is true is $O(|R| \times |P| \times |\xi|)$, which is polynomial time. Thus, the verification process for enforcing the PSOD constraints is in P. □

Statement 4.2. *Given a system status γ and a set of MEPC constraints, the process of verifying whether γ is satisfied is also in P.*

Proof: This verification is similar to that of Statement 4.1, and the detailed process is omitted due to limited space. Here, the enforcement of the MEPC constraints under γ is also available in polynomial time. □

Then, we present an approach for constructing *t-m* MEPC constraints as shown in Algorithm 4, which takes a *k-n* PSOD constraint as input and outputs a set of MEPC constraints that can implicitly enforce PSOD.

Algorithm 4. Construction of *t-m* MEPC constraints.

```
Input: k-n PSOD constraint $psod = <\{p_1, p_2, \dots, p_n\}, k>$, where $2 \le k \le n$
Output: set $\psi$ of t-m MEPC constraints
1.  Initialize $\psi = \varnothing$;
2.  if $k == 2$ then
3.     $\psi = \{<\{p_1, p_2, \dots, p_n\}, n>\}$;
4.  else if $k == n$ then
5.     $\psi = \{<\{p_1, p_2, \dots, p_n\}, 2>\}$;
6.  else
7.     for $t = 2$ to $\lfloor \frac{n-1}{k-1} \rfloor + 1$ do
8.        $m = (k-1) \times (t-1) + 1$;
9.        for any subset $\{p_1, p_2, \dots, p'_m\}$ in $\{p_1, p_2, \dots, p_n\}$ do
10.          $\psi = \psi \cup \{<\{p_1, p_2, \dots, p'_m\}, t>\}$;
11.       end for
12.    end for
13. end if
```

As observed from Algorithm 4, the third statement is presented as follows.

Statement 4.3. *Given a permission set P, a t-m MEPC requirement and the optimized role set Optim_Roles in Algorithm 2, the system is safe if* $t \le \left(\left\lfloor \frac{|P|-1}{|\mathit{Optim\_Roles}|-1} \right\rfloor + 1\right)$.

Proof: According to the specifications of the MEPC and PSOD, any role is allowed to possess $(t-1)$ permissions at most, and then $(|Optim\_Roles|-1)$ roles are allowed to possess $(|Optim\_Roles|-1) \times (t-1)$ permissions at most. We use the method of contradiction to prove it. Without a loss of generality, assume that $t = \left(\left\lfloor \frac{|P|-1}{|Optim\_Roles|-1} \right\rfloor + 1\right) + 1$, the system is still safe, i.e., the number of permissions related to $(|Optim\_Roles|-1)$ roles is $(|Optim\_Roles|-1) \times \left(\left\lfloor \frac{|P|-1}{|Optim\_Roles|-1} \right\rfloor + 2 - 1\right) \approx |P| - 1 + |Optim\_Roles| - 1 > |P|$, which breaches the PSOD constraint. Thus, the assumption is false. □

4.2. **Completeness of the mining results.** According to the optimal results of the role mining and their associated user clusters, in order to avoid the repeated mining, we need to remove the roles that have been derived in other clusters and update the unvisited permissions in the clusters until all the permissions are visited. Owing to the diversity of system resources and the variability of resource access, the conventional counting methods that directly remove the permissions of clusters as shown in Algorithm 2, seem to be complicated and cannot ensure the completeness of the mining results. The two-dimensional Boolean matrix in computer science is an effective method to store and represent a large-scale permissions assignment, which dynamically adjusts the unit values of different matrices according to the mining roles until the values of all matrix units are 0. Therefore, it is necessary to study how to calculate the number of the remaining units with values of 1 in the matrix, in order to assist in identification of the number of unvisited permissions.

To make sure the mining results cover all the permissions while ensuring the integrity of the mining process, according to Algorithms 1 and 2, the method of determining whether or not the permissions are visited, is presented in Algorithm 5.

Algorithm 5. Identification of unvisited permissions.

```
Input: permission set $P$, cluster set $C$, optimized role $or \in Optim\_Roles$
Output: $count\_unvistedperms(c)$ of Algorithm 2, where $c \in C$
 1. According to the hash-mapping reverse rule, convert the cluster-permission assign-
    ments into matrix $CPA_{m\times n}$, where $m = |C|$, and $n = |P|$;
 2. for each row $CPA[i]$ in $CPA_{m\times n}$ do
 3.   Initialize $count\_unvistedperms(c) = 0$;
 4.   for each column $CPA[j]$ in $CPA_{m\times n}$ do
 5.     if $(or \in perm\_roles(user\_perms(c))) \wedge (p_j \in user\_perms(c))$ then
 6.       $CPA[i][j] = 0$;
 7.     else
 8.       $count\_unvistedperms(c) += CPA[i][j]$;
 9.     end if
10.   end for
11. end for
```

5. **Experiments and Analyses.** In this section, we perform two groups of experiments for the REO_MEPC&PRCC. The first group of experiments is used to evaluate its performance with respect to different constraint values. The second group is to compare its performance with the existing methods. We consider four real-world datasets from the work in [17], and adopt the mining tool RMiner [26] to evaluate the performance of the unconstrained role mining. The original datasets also include the density of each dataset and the number of the candidate role sets *Cand_Roles* as shown in Table 7. All experiments are implemented on a standard desktop PC with an Intel i5-7400 CPU, 4

Table 7. The original datasets

Dataset	$\lvert U\rvert$	$\lvert P\rvert$	$\lvert UPA\rvert$	Density	$\lvert Cand\_Roles\rvert$
Healthcare 1	46	315	1,486	10.3%	15
Domino	79	231	730	4%	20
Firewall 1	365	709	31,951	12.3%	69
Firewall 2	325	590	36,428	19%	10

GB RAM, and 160 GB hard disks, running a 128-bit Windows 10 operating system. All simulations are compiled and executed under the Java environment.

5.1. **Performance evaluations of the REO_MEPC&PRCC.** To evaluate the performance of our method under the MEPC in the role mining process, we consider the number of the mining roles *Optim_Roles* as the evaluation criterion and the value of the constraint t varies from 1 to $|P|$. We repeatedly implement the experiments 5 times, and take their average values. The results are shown in Figures 2-5, where the lateral axis represents the constraint value, and the vertical axis represents the number of roles.

Taking the dataset Healthcare 1 as an example for implementing the experiments, Figure 2 shows that the number of roles tends to decrease as the value of t increases. Specifically, when $t = 2$ (that is, the MEPC can be denoted as $mepc < P, 1 >$), 315 roles need to be developed, and each role includes only one permission, which is not practical; when $2 < t < 6$, the number of roles decreases remarkably from 48 to 18 with the increasing value of t; when $6 \leq t < 21$, the number of roles decreases slightly from 18 to 14 with the increasing value of t; when $t \geq 21$, it grows to be linearly as the value of t increases. That is because, the less the value of t is, the stronger the constraints and the more roles that satisfy the constraint requirements will be. The greater the value of $|Optim\_Roles|$ before the value of t reaches 6, the higher the redundancy of the mining results. On the contrary, the greater the value of t, the weaker the constraints and the fewer roles that satisfy the constraint requirements, especially the value of $|Optim\_Roles|$ decreases when the value of t exceeds 6, and then the redundancy of the mining results becomes lower and lower. According to the analysis in Section 4.1, the system is unsecure once t is too large, though the value of $|Optim\_Roles|$ tends to be stable when t exceeds 21. Thus, it is concluded that, the system is secure when t varies in $[6, 21)$ and $|Optim\_Roles|$ varies in $[14, 20)$ for the dataset Healthcare 1.

The performances of our method on the other three datasets are presented as shown in Figures 3-5. Figure 3 shows that, when $t \geq 52$, the number of roles first decreases slightly and then grows to be linearly as the value of t increases. Figure 4 shows that, when $t \geq 5$, the number of roles first decreases slightly and then grows to be linearly as the value of

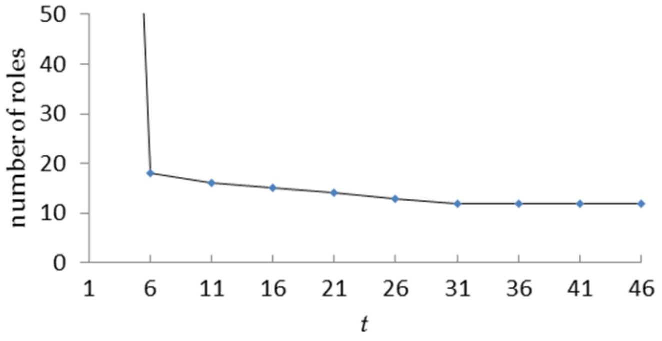

Figure 2. The mining results on Healthcare 1

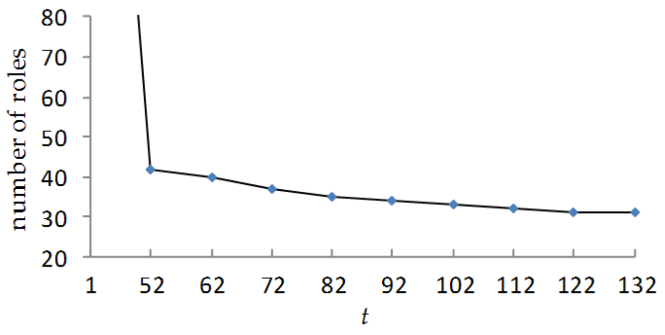

FIGURE 3. The mining results on Domino

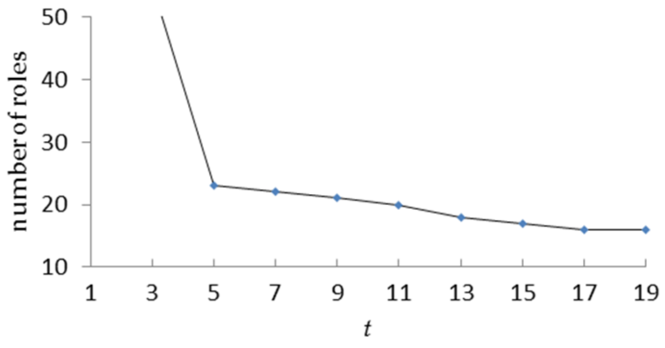

FIGURE 4. The mining results on Firewall 1

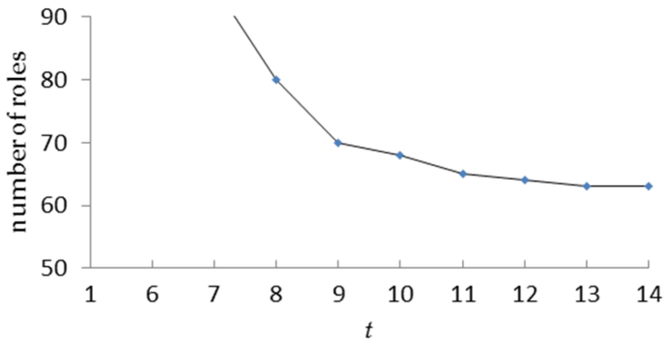

FIGURE 5. The mining results on Firewall 2

TABLE 8. Comparison of the mining results

Dataset	Enumeration method [13]	REO_MEPC&PRCC
Domino	674	$[34, 40)$
Healthcare 1	31	$[14, 20)$
Firewall 1	62	$[20, 25)$
Firewall 2	278	$[66, 71)$

t increases. Figure 5 shows that, when $t \geq 9$, the number of roles first decreases slightly and then grows to be linearly as the value of t increases. Similar to the analysis of Figure 2, it is concluded that, the system is secure when $|Optim\_Roles|$ varies in $[34, 40)$, $[20, 25)$, and $[66, 71)$, respectively. Furthermore, it is observed from Table 8 that, the number of the mining roles using the enumeration method is much greater than that of our method.

5.2. **Performance comparisons with the existing method.** To further evaluate the performance of our method under the PRCC in the role-engineering optimization process, we implement the experiments on the datasets Domino and Healthcare 1 as shown in Table 7. The representative post-processing method [23] is used for the performance comparison and the results with error bars are presented as shown in Figures 6 and 7, where the lateral axis represents the $MRC_{permission}$, and the vertical axis represents the number of roles.

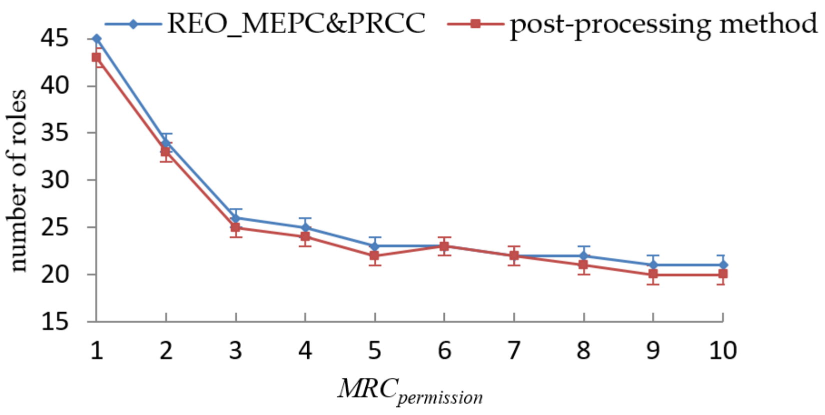

FIGURE 6. Performance comparison on Domino

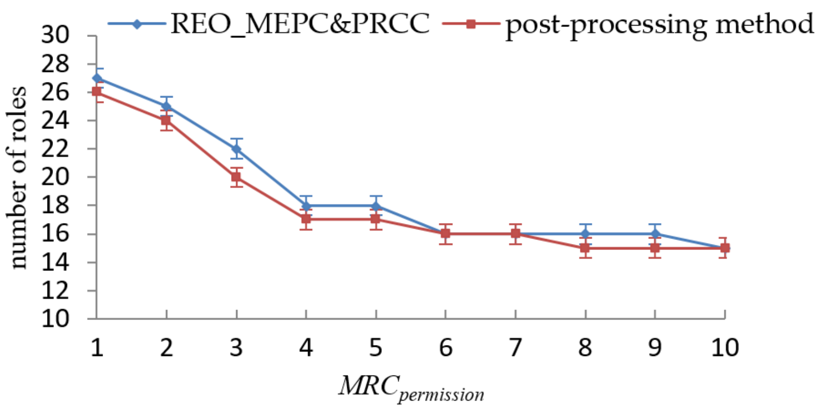

FIGURE 7. Performance comparison on Healthcare 1

It is observed from Figure 6 that, the number of roles decreases as $MRC_{permission}$ increases for the REO_MEPC&PRCC, which tends to be stable as $MRC_{permission}$ increases to a certain value. Specifically, the number of roles does not obviously vary and remains close to 20 when the value of $MRC_{permission}$ exceeds 8. A further observation is that the number of roles first varies slightly and then increases significantly as $MRC_{permission}$ decreases. That is because, the greater the value of $MRC_{permission}$, the weaker the constraint and the more roles related to any permission. In other words, with a greater value of $MRC_{permission}$, regular roles are more applicable and can be utilized more frequently. Thus, fewer irregular roles need to be created, and the number of roles does not vary considerably. Similarly, it is observed from Figure 7 that, the number of roles also decreases as MRC_{user} increases, which also tends to be stable and remains close to 15 when $MRC_{permission}$ increases to a certain value.

Furthermore, it is observed from the figures that for the two methods, the number of the optimized roles using our method is close to that of the post-processing method. Detailed analyses are not discussed in this paper as similar discussions have been presented in our research work [27]. Therefore, our method is as effective as the post-processing method for the Domino and Healthcare 1 datasets.

5.3. **Advantages and shortcomings of the REO_MEPC&PRCC.** From the above analyses, we find the REO_MEPC&PRCC has the following main advantages.

1) In the preprocessing stage, it can reduce the mining scales and alleviate the management burdens by adopting the four tuples of clusters to generate user clusters and reconstruct the original access matrix.
2) In the constrained role-mining stage, it can implement role mining with the mutually exclusive permissions constraints, while ensuring system security and integrity by utilizing the separation-of-duty constraint and the method of permissions covering.
3) In the role-engineering optimization stage, it can implement the optimization process with the permission-to-role cardinality constraints by constructing an optimized RBAC system.

Compared with the existing studies, the security features of the proposed method are shown in Table 9, where a tick √ indicates that the feature is available. It can be seen from table that, however, our proposed method does not consider whether the other three cardinality constraints can be satisfied.

Table 9. Comparison of security features

Feature	Blundo and Cimato [20]	John et al. [21]	Ma et al. [22]	Harika et al. [23]	Sarana et al. [24]	Sun et al. [25]	Proposed method
PRCC		√		√			√
Other cardinality constraints	√		√	√			
Mutually exclusive constraints					√	√	√
Separation of duty constraint					√	√	√
Reduction of mining scales						√	√

6. **Conclusions.** A novel role-engineering method, called REO_MEPC&PRCC, was proposed in this paper. We first converted the role mining problem into a clustering problem by generating the four tuples of user clusters, and then reconstructed the original access control matrix. Next, we implemented the role-engineering optimization using the permission-oriented mutually exclusive constraints and cardinality constraints simultaneously, derived optimal roles, and constructed an optimized RBAC system. We also utilized the separation-of-duty constraint and the method of permissions covering to ensure the soundness and completeness of the optimization process. The experiments demonstrated the effectiveness of the proposed method. However, a few interesting issues remain to be solved. To further enhance the interpretability of mining results, one issue is to consider

how to implement the other cardinality constraints or mutually exclusive constraints for the role-engineering optimization in future work.

Acknowledgment. This work is partially supported by the Natural Science Foundation of China (61501393), the Natural Science Foundation of Henan Province of China (182300410145, 182102210132), and the Foundation of Henan Educational Committee under Contract No. 20B520031.

REFERENCES

[1] W. Sun, H. Su and H. Xie, Policy-engineering optimization with visual representation and separation-of-duty constraints in attribute-based access control, *Future Internet*, vol.12, no.10, p.164, 2020.

[2] G. Batra, V. Atluri, J. Vaidya and S. Sural, Deploying ABAC policies using RBAC systems, *Journal of Computer Security*, vol.27, no.4, pp.483-506, 2019.

[3] M. Ghafoorian, D. Abbasinezhad-Mood and H. Shakeri, A thorough trust and reputation based RBAC model for secure data storage in the cloud, *IEEE Trans. Parallel and Distributed Systems*, vol.30, no.4, pp.778-788, 2019.

[4] J. P. Cruz, Y. Kaji and N. Yanai, RBAC-SC: Role-based access control using smart contract, *IEEE Access*, no.6, pp.12240-12251, 2018.

[5] R. I. Putri, I. N. Syamsiana and M. Rifa'i, Power extraction optimization of PMSG wind turbine system based on simple modified perturb & observe, *ICIC Express Letters, Part B: Applications*, vol.11, no.1, pp.17-24, 2020.

[6] N. Pan, Z. Zhu, L. He and L. Sun, An efficiency approach for RBAC reconfiguration with minimal roles and perturbation, *Concurrency and Computation: Practice and Experience*, vol.30, no.11, DOI: 10.1002/cpe.4399, 2018.

[7] B. Mitra, S. Sural, J. Vaidya and V. Atluri, Migrating from RBAC to temporal RBAC, *IET Information Security*, vol.11, no.5, pp.294-300, 2017.

[8] A. Baumgrass and M. Strembeck, Bridging the gap between role mining and role engineering via migration guides, *Information Security Technical Report*, vol.17, no.4, pp.148-172, 2013.

[9] X. Ma, J. Wang, L. Zhao and R. Li, Mutual exclusion role constraint mining based on weight in role-based access control system, *International Journal of Innovative Computing, Information and Control*, vol.12, no.1, pp.91-101, 2016.

[10] N. Gal-Oz, Y. Gonen and E. Gudes, Mining meaningful and rare roles from web application usage patterns, *Computers & Security*, vol.82, pp.296-313, 2019.

[11] W. Bai, Z. Pan, S. Guo and Z. Chen, RMMDI: A novel framework for role mining based on the multi-domain information, *Security and Communication Networks*, 2019.

[12] S. D. Stoller and T. Bui, Mining hierarchical temporal roles with multiple metrics, *Journal of Computer Security*, vol.26, no.1, pp.121-142, 2018.

[13] J. Vaidya, V. Atluri and Q. Guo, The role mining problem: A formal perspective, *ACM Trans. Information and System Security*, vol.13, no.3, 2010.

[14] A. Colantonio, R. D. Pietro, A Ocello and N. V. Verde, Mining business-relevant RBAC states through decomposition, *Proc. of the 25th IFIP TC-11 International Information Security Conference*, Brisbane, Australia, pp.19-30, 2010.

[15] J. D. Ultra and S. Pancho-Festin, A simple model of separation of duty for access control models, *Computers & Security*, vol.68, pp.69-80, 2017.

[16] F. Nazerian, H. Motameni and H. Nematzadeh, Emergency role-based access control (E-RBAC) and analysis of model specifications with alloy, *Journal of Information Security and Applications*, vol.45, pp.131-142, 2019.

[17] B. Mitra, S. Sural, J. Vaidya and V. Atluri, A survey of role mining, *ACM Computing Surveys*, vol.48, no.4, 2016.

[18] N. Li, M. V. Tripunitara and Z. Bizri, On mutually exclusive roles and separation-of-duty, *ACM Trans. Information and System Security*, vol.10, no.2, 2007.

[19] J. Vaidya, V. Atluri and Q. Guo, The role mining problem: Finding a minimal descriptive set of roles, *Proc. of the 12th ACM Symposium on Access Control Models and Technologies*, Sophia Antipolis, France, pp.175-184, 2007.

[20] C. Blundo and S. Cimato, Constrained role mining, in *Security and Trust Management. STM 2012. Lecture Notes in Computer Science*, A. Jøsang, P. Samarati and M. Petrocchi (eds.), Berlin, Heidelberg, Springer, 2012.

[21] J. C. John, S. Sural, V. Atluri and J. Vaidya, Role mining under role-usage cardinality constraint, in *Information Security and Privacy Research. SEC 2012. IFIP Advances in Information and Communication Technology*, D. Gritzalis, S. Furnell and M. Theoharidou (eds.), Berlin, Heidelberg, Springer, 2012.

[22] X. Ma, R. Li, H. Wang and H. Li, Role mining based on permission cardinality constraint and user cardinality constraint, *Security and Communication Networks*, vol.8, no.13, pp.2317-2328, 2015.

[23] P. Harika, M. Nagajyothi, J. C. John, S. Sural, J. Vaidya and V. Atluri, Meeting cardinality constraints in role mining, *IEEE Trans. Dependable and Secure Computing*, vol.12, no.1, pp.71-84, 2015.

[24] P. Sarana, A. Roy, S. Sural, J. Vaidya and V. Atluri, Role mining in the presence of separation of duty constraints, *Information Systems Security. ICISS 2015. Lecture Notes in Computer Science*, S. Jajoda and C. Mazumdar (eds.), Cham, Springer, 2015.

[25] W. Sun, S. Wei, H. Guo and H. Liu, Role-mining optimization with separation-of-duty constraints and security detections for authorizations, *Future Internet*, vol.11, no.9, p.201, 2019.

[26] R. Li, H. Li, W. Wei, X. Ma and X. Gu, RMiner: A tool set for role mining, *Proc. of the 18th ACM Symposium on Access Control Models and Technologies*, Amsterdam, the Netherlands, pp.193-196, 2013.

[27] W. Sun, H. Su and H. Liu, Role-engineering optimization with cardinality constraints and user-oriented mutually exclusive constraints, *Information*, vol.10, no.11, p.342, 2019.