Semantic Analysis of Role Mining Results and Shadowed Roles Detection

Safaà Hachana[a,c], Frédéric Cuppens[b], Nora Cuppens-Boulahia[a,b], Joaquin Garcia-Alfaro[b]

[a] *Swid Web Performance Service, Rennes, France, Email: safa@swid.fr*
[b] *Institut Telecom-Mines/Telecom Bretagne, Dépt. LUSSI, Rennes, France*
Emails: {frederic.cuppens; nora.cuppens; joaquin.garcia}@telecom-bretagne.eu
[c] *École Nationale Supérieure de Mécanique et d'Aérotechnique, LISI, Poitiers, France*

Abstract

The use of role engineering has grown in importance with the expansion of highly abstracted access control frameworks in organizations. In particular, the use of role mining techniques for the discovery of roles from previously deployed authorizations has facilitated the configuration of such frameworks. However, the literature lacks from a clear basis for appraising and leveraging the learning outcomes of the role mining process. In this paper, we provide such a formal basis. We compare sets of roles by projecting roles from one set into the other set. This approach is useful to measure how comparable the two configurations of roles are, and to interpret each role. We formally define the problem of comparing sets of roles, and prove that the problem is NP-complete. Then, we propose an algorithm to map the inherent relationship between the sets based on boolean expressions. We demonstrate the correctness and completeness of our solution, and investigate some further issues that may benefit from our approach, such as detection of unhandled perturbations or source misconfiguration. In particular, we emphasize that the presence of shadowed roles in the role configuration increases the time complexity of sets of roles comparison. We provide a definition of the shadowed roles problem and propose a solution that detects different cases of role shadowing.

Keywords: Access Control; Role Mining; IT Security; Boolean Logic;

1. Introduction

Access control is a security fundamental concern which covers a wide area of applications including operating systems, database systems, enterprise resource planning systems, and workflow systems. Role Based Access Control (RBAC) [1] [2] is the dominant model for access control in both commercial and research fields. It structures into roles the concrete agents who have access requirements to secured objects, such as employees, programs, or processors.

A role can be viewed from two perspectives: a set of authorizations that are recurrently assigned together, or a set of agents that are granted the same set of authorizations. As RBAC platforms have become essential to large enterprises, the major difficulty is still the configuration and establishment of the RBAC model in the organization.

Role engineering is the practical discipline of implementing the RBAC model into organizations. Essentially, role engineering is the process of structuring the different agents of an organization into roles, and the association of the corresponding set of authorizations to every role. Top-down and bottom-up are the two common strategies in order to apply role engineering. Under the top-down approach, the roles are defined by carefully analyzing the business process associated with the organization, and decomposing it into smaller units in a functionally independent manner. Under the bottom-up approach, roles emerge from existing configurations of authorizations already deployed over the organization. The role engineering task is extremely time consuming and costly [3]. For this reason, the use of 'automatic' bottom-up role engineering techniques is growing in importance. These techniques get inspiration from data mining techniques, hence, they are known as *r*ole mining (RM). Several efficient role mining algorithms have been proposed in the literature [4]. However, they still require intensive human interpretation before leading to appropriate results. To the best of our knowledge, no efficient solutions have been proposed to assist a security administrator in the task of leveraging the outcomes of a role mining process. Moreover, as the years elapse since the wide adoption of RBAC in organizations, a new requirement is strongly emerging. In addition to the traditional application of role engineering upon organizations where there is only a direct user-permission assignment framework, we may also need to apply role engineering in organizations where a pre-existing RBAC configuration exists but is no longer optimized. This may happen because the RBAC configuration is getting old and recurrently updated, or does not fit the dynamic nature of the organization anymore. In such cases, keeping as close as possible to the original RBAC configuration is the proper optimization criteria of existing role mining strategies [5]. In this regard, the ability of mapping the outcomes of a role mining process to the set of original authorizations may be highly beneficial in order to understand and validate the discovery process, as well as to detect unhandled perturbations over the deployed configurations, or misconfiguration within the original ones. Similarly, we also rise the necessity of comparing the outcomes of two different role mining processes. Indeed, several role mining tools have been proposed in the literature. Each new role mining technique needs to be tested and evaluated before being released and used in real configurations. The test stage usually consists of running the role mining algorithm on real and synthetic concrete data, generated from already known consistent RBAC state. The RBAC state obtained from the role mining is then compared to the original RBAC state, in a reverse engineering manner. However, we do not find any analytic comparison tool between two RBAC configurations of roles in the literature. Thus, an automatic matching tool between roles from two sets is obviously needed, either for the assessment of the role mining methods, or for

the interpretation of the mined roles.

In this paper, we formally define the problem of comparing two sets of roles according to the underlined requirements extracted from the literature. We demonstrate that the problem is NP-complete. We present a greedy solution that handles the motivation problem. Then, we prove the correctness and completeness of the solution. We define a sufficient condition that guarantees preciseness of the comparison between the original set of roles and mined set of roles. The experimental results confirm the validity of our approach, as a beneficial solution to analyze and better understand the set of mined roles when an original set of roles exists.

Paper organization. Section 2 reviews some technical definitions about the RBAC model and role mining problem and its algebraic representation. Section 3 presents the use cases considered in this paper, introduces an example of motivation, and provides a comparison with related work. Section 4 formally defines the problem to solve. Section 5 presents our solution to the problem, underlines some of the properties of our solution, and elaborates on some heuristics to enhance our approach. Section 6 presents the shadowed roles problem, links it with the role set comparison problem, and proposes an algorithm for shadowing detection. Section 7 presents experimental results. Section 8 concludes the paper.

This paper is an extended version of the previous work presented at ARES 2012 [6]. In particular, the motivation use cases has been emphasized and illustrated by an example in Section 3. The algorithmic complexity evaluation in Section 5 has been revised and enhanced by the proposition of a Heuristic. Section 6 has been added to handle the shadowed roles problem. Section 7 has been upgraded with experimental results of tests with the shadowing detection algorithm introduced in section 6.

2. Preliminaries

In this section, we present the background on the Role Based Access Control (RBAC) model. In addition, we state a formulation of the role mining problem, and introduce the algebraic presentation of the involved data in role mining.

2.1. Role Based Access Control Model

The RBAC model is a NIST standard [2]. It introduces the notion of "role" in order to make the access control system more compact and comprehensive, compared to the direct user-permission assignments. Definition 1 presents the basic model RBAC_0 [2] without considering sessions.

Definition 1. *RBAC*
An RBAC configuration, denoted as $RC = (ROLES, UR, RP)$*, is characterized by:*

- *U, ROLES, OPS, and OBJ, which are the sets of users, roles, operations, and objects*

- $UR \subseteq U \times ROLES$, *a many-to-many mapping user-to-role assignment relation*
- $PRMS \subseteq \{(op, obj) | op \in OPS \wedge obj \in OBJ\}$, *a set of permissions, where a permission is an operation over an object*
- $RP \subseteq ROLES \times PRMS$, *a many-to-many mapping of user-to-permission assignment relation*[1]
- $assigned\_users(R) = \{u \in U | (u, R) \in UR\}$, *the mapping of role R onto a set of users*
- $assigned\_permissions(R) = \{p \in PRMS | (R, p) \in RP\}$, *the mapping of role R onto a set of permissions*

2.2. The Role Mining Problem

Intuitively, role mining is the process of extracting a configuration of roles from a discretionary user permission assignment relation. The users granted with similar permissions are structured into similar roles. Symmetrically, permissions with common users are assigned together, so that they belong to the same roles. Optionally, some information about the business process or the user attributes may be provided, and the is called a hybrid role mining. Nevertheless, there is no consensus about the formal role mining problem definition. Several role mining techniques are proposed in the literature, with different assumptions and optimization criteria leading to different solutions. For instance, the first explicit formal definition for the *role mining problem* is given in [7]. Vaidya et al. suggest the minimization of the number of generated roles as an optimization criteria, and show that this problem is NP-complete. The *inference role mining problem* presented by Frank et al. [8] focuses on the problem from a general perspective. In all cases, role mining assumes implicitly that an underlying RBAC configuration exists in the deployed authorizations, and aims to reveal this underlying configuration.

Definition 2. *Inference Role Mining Problem*
Let U be a set of users, PRMS a set of permissions, UPA a user-permission assignment relation, and, optionally, part of the top-down information TDI be given. Infer the unknown RBAC configuration $RC^* = (ROLES^*, UR^*, RP^*)$, *under the following assumptions:*

1. *An underlying RBAC configuration exists*
2. *Exceptions (may) exist*
3. *TDI (if given) influences* RC^*.

[1] In the original NIST standard RP is defined as $RP \subseteq PRMS \times ROLES$ rather than $ROLES \times PRMS$.

Definition 2 provides a unified view of both bottom-up and hybrid role mining. Unlike other definitions, it does not give the function to optimize, i.e. the precise way to solve the problem. Therefore, to validate that the problem is solved, we must know the underlying RBAC configuration RC^*. We can compare the set of mined roles with the set of original roles in experimental scenarios, as well as in application scenarios where an old RBAC configuration exists.

Algebraic Representation of the Problem. To unify the representation of the inputs and outputs of role mining, the involved entities are represented with boolean matrices as follows: given m users, n permissions and k roles (i.e., $|U| = m$, $|PRMS| = n$, $|ROLES| = k$), the user-to-role mapping UR is represented as an $m \times k$ matrix where 1 in cell $\{ij\}$ indicates the assignment of role j to user i. Similarly, the role-to-permission mapping RP is represented as a $k \times n$ matrix where 1 in cell $\{ij\}$ indicates the assignment of permission j to role i. Finally, the user-to-permission mapping UPA is represented as an $m \times n$ matrix where 1 in cell $\{ij\}$ indicates the assignment of permission j to user i. The relationship between UPA, UR and RP can be expressed with the boolean matrix multiplication.

Definition 3. *Boolean Matrix Multiplication*
A Boolean matrix multiplication between Boolean matrices $A \in \{0,1\}^{n \times k}$ *and* $B \in \{0,1\}^{k \times m}$ *is* $A \otimes B = C$ *where* $C \in \{0,1\}^{n \times m}$ *and* $c_{ij} = \bigvee_{l=1}^{k} \wedge a_{il} b_{lj}$.

In an RBAC configuration, we have $UPA = UR \otimes RP$. A user i is granted permission j if, at least, one of its roles is assigned to this permission. The role mining process approximates the above decomposition by extracting from a given UPA the relations UR^* and RP^* such that: $UPA \approx UR^* \otimes RP^*$.

3. Motivation

3.1. Use Cases for Comparing Two Sets of Roles

The requirement for comparing two sets of roles occurs in several use cases related to role mining. The first use case is the assessment of role mining algorithms. An algorithm designer usually evaluates the performance of its RM algorithm with the reverse engineering technique: starting from an RBAC configuration, apply role mining over the resulting UPA, and compare the obtained roles with the original roles. Likewise, we can compare the outputs of several role mining algorithms to test their performance under different constraints. The second use case is the enforcement of role mining results. A security administrator can require assistance toward migrating to a new RBAC configuration from an old RBAC configuration. A similar application is migrating from discretionary UPA to an RBAC configuration. Considering the set of permissions of each user as a pseudo-role, the administrator may need assistance to assign each user to the appropriate roles which guarantee the required permissions. A

mapping of the pseudo-roles with the obtained new RBAC roles may solve the problem.

In almost all the above mentioned cases, we have typically a set of roles of reference or original roles OR and a set of mined roles MR. The first set is a set of mastered roles since they are well known by the security administrator, in opposition to the set of new roles MR. The two configurations of roles are defined for the same set of discretionary security rules *UPA*. However, they may have been calculated with respect to different constraints. For instance, one set can correspond to a hierarchical structure of roles whereas the other is flat. Similarly, the roles of one set may be partially overlapping whereas the roles of the other set are orthogonal.

Role mining algorithms usually output a list of roles, possibly overlapping or even redundant which means that a role can be fully covered by a union of a subset of other roles from the same list. When migrating to a new RBAC configuration, the security administrator is confronted to such a list of new roles, and he has to assign them to the users and manage them to suit the organization's evolutive requirements. The administrator has to respect three access control rules: provisioning, security and maintainability. For provisioning, each user should have access in the new configuration of roles to all its privileges in the old one. For that, the administrator needs to know how to optimally cover the permissions provided by each role from OR using roles from MR. Security consists of not allowing any user to access extra-privileges. For this purpose, the administrator needs to ensure that the new assigned roles to the users do not exceed the privileges provided by their old roles. Finally, the maintainability aims to make simple and safe the evolution of the structure of roles by adding and retrieving permissions and users to the roles, according to the evolving requirements of the organization. One key characteristic to ensure the maintainability is mastering the configuration of roles. Thus, the security administrator, who masters the old roles or the old *UPA*, would be very interested in leveraging his experience with the old roles in order to master the new configuration of roles more quickly. Unfortunately, mapping the new roles with the old ones manually is not a viable task. It is essential then to assist the security administrator with automated tools to analytically understand the new roles. In particular, assistance is needed to find where the permissions of an old role has been distributed in the new configuration of roles, especially when moving between a flat role structure and a hierarchical role structure. In addition, such tool may help to find how to optimally assign the permissions to the users without adding extra-permissions using the new roles.

3.2. Motivating Example

We provide the following motivation example to illustrate the problems handled in the paper. Table 1 shows two equivalent RBAC configurations of a hypothetical financial department of an organization. They could be an old and a new representations of a subset of the Access Control policy of the organization. We imagine the following scenario. At some point of time, security experts have defined the set of original roles in OR (see Table 1(a)). In this repartition of

Table 1: Motivating example: original and mined equivalent RBAC configurations

(a) Original Roles Matrix OR

Original Roles	cTrans(p1)	rP-order(p2)	cP-order(p3)	vTrans(p4)
Handle Order(r1)	1	1	0	0
Create Order(r2)	0	0	1	0
Supervise Transfer(r3)	0	1	0	1

(b) Mined Roles Matrix MR

Mined Roles	cTrans(p1)	rP-order(p2)	cP-order(p3)	vTrans(p4)
Manage Order(R1)	1	1	1	0
Validate Transfer(R2)	0	0	0	1

(c) Original User to Role Assignment Matrix UR1

User-ORole	r1	r2	r3
U1	1	1	0
U2	1	1	1
U3	0	0	0
U4	1	1	0
U5	1	1	0

(d) Mined User to Role Assignment Matrix UR2

User-MRole	R1	R2
U1	1	0
U2	1	1
U3	0	0
U4	1	0
U5	1	0

(e) Current User to Permission Assignment Matrix UPA

User-Perm	p1	p2	p3	p4
U1	1	1	1	0
U2	1	1	1	1
U3	0	0	0	0
U4	1	1	1	0
U5	1	1	1	0

roles, they have planned that any employee of the organization who needs to make a purchase has to contact an employee from the *financial department*, who will *create a purchase-order*. Some employees in the department are in charge of *reading all the existing purchase-orders* and deciding to *create cash transfers* for the relevant orders according to their priority. An employee with higher responsibility has a *reading access to all the purchase-orders* and must *validate the cash transfers* before they become effective. We suppose that after some time, the progress in the staffing of the organization has leaded to the user to role assignments in UR1(see Table 1(c)). In the terminology of the RBAC Definition 2.1, the operations are $OPS = \{$*create (c), read (r), validate (v)*$\}$ and the objects are $OBJ = \{$*purchase-order (p-Order), cash transfer (Trans)*$\}$. The set of permissions is then: $PRMS = \{$*cTrans, rP-order, cP-order, vTrans*$\}$. The first set of roles is $Original\_ROLES = \{$ *Handle Order(r1), Create Order(r2), Supervise Transfer(r3)*$\}$. The combination of the role-to-permission and the user-to-roles relations OR and UR1 gives the user-to-permission access control relation UPA (see Table 1(e)): UR1$\otimes$OR = UPA.

If we focus on the user-to-role assignment in the original RBAC configuration UR1(in Table 1(c)), we notice that $r1$ and $r2$ are always assigned to the users together, and that $r3$ is always assigned to the users with $r1$ and $r2$ which means that the permission $p3$ in $r3$ does not have any impact on the UPA since it is also provided by $r2$ to the same user. These remarks may warn that the old configuration of roles is no more suitable to the new policy of access control and to the current distribution of responsibilities over the employees. In our scenario, we suppose that security administrator has decided to apply some role mining technique on the current UPA matrix (see Table 1(e)), and has obtained a new RBAC configuration with the new set of roles MR (see Tab 1(b)). The set of mined roles is $Mined\_ROLES$ ={ *Manage Order(R1), Validate Transfer(R2)*}. It contains only two non overlapping roles $R1$ and $R2$. The new user to role assignment relation is UR2 (in Table 1(d)). Actually, most of role mining techniques calculate only the Role to permission matrix RP2 and the security administrator has to assign the users to the new roles manually to get UR2.

The two RBAC configurations in Table 1 are such that: UR2⊗OR = UR1⊗OR = UPA, which means that the two sets of roles are *equivalent* according to the following definition.

Definition 4. *Equivalent Sets Of Roles*
Two sets of roles RP1 and RP2 are equivalent *if they may lead to a same User-to-Permission-Assignment relation UPA i.e. there exists two respective User-to-Role matrices UR1 and UR2 for RP1 and RP2 such that* $UR1 \otimes RP1 = UR2 \otimes RP2 = UPA$;

In order to understand and analyze the new set of roles, and to assign the users to the new roles, the security administrator needs to compare the mined roles with the original roles that he used to control. Comparing the mined roles to the original roles should reveal that $R1$ is the combination of $r1$ and $r2$, i.e. $R1 = r1 \cup r2$ and that $R2$ is $r3$ minus the permission $p2$ that was shadowed in RP1, since it is also assigned through the role $r1$, so $R2 = r3 \cap \neg r1$. It is easy to establish manually such a relationship between the old roles and the new roles in a simple example like in Table 1, but in real life applications, it is very hard to handle a large number of roles manually, and automatic solutions are required to perform this task. However, the existing solutions in literature are unable to find a satisfactory comparison between two sets of roles.

3.3. Related Work

Research on role mining has revealed different requirements for the comparison of roles. Several propositions in this field have been presented.

In [9], Kuhlmann et al. address the problem by counting how many of the roles lying in the original reference set are exactly the same as those discovered by their proposed role mining method. Similarly, Vaidya et al. in [10] consider the average, instead of the total number, of original roles that are exactly discovered by their role mining technique, as an evaluation metric for role mining.

Thus, their metric provides means to compare different role mining approaches, being not influenced by the number of roles, and giving a similarity of one for total coincidence, and zero for total difference. The main drawback of these two first metrics is that they discard roles that are very similar but not exactly the same as roles in the original role set. For instance, if we compare the two sets of roles OR and MR presented in the previous example (Table 1) using either the method in [9] or [10], and since none of the original roles has been discovered by the role mining algorithm, we will obtain Similarity(OR,MR) = 0, which suggests that the two sets of roles are not comparable, whereas they are equivalent actually.

Hassan Takabi et al. in [11], consider the case in which an initial RBAC configuration is upgraded during the role mining process. The similarity between roles is one of the two optimization criteria used by their role mining approach, the minimality of the resulting number of roles being the second criteria. In order to address similar issues, Vaidya et al. propose in [5] a series of similarity and dissimilarity metrics based on the Jaccard Coefficient and Jaccard Distance[2]. Their metrics have been recurrently used in the literature to compare roles and sets of roles; and to evaluate role mining algorithms [4]. Applied to our motivation example, Similarity(MR,OR)= (Jacc(R1, r1) + Jacc(R2, r3))/2 = (0.66 + 0.5)/2 = 0.58. However, these Jaccard-based measurements still suffer from several important limitations. First, it computes role similarity for one role from one set with respect to only one role in the compared set. In the motivation example, we get Jacc(R1,r1) = 0.66 whereas it would be more accurate to calculate Jacc(R1, r1 $\cup$ r2) = 1. Second, the same role may be used to be compared with several roles from the other set. Thus, as a quality performance criteria, it may artificially be affected by reference sets with a high number of closely similar roles. For instance, if we suppose that the role mining algorithm has produced another role $R3$ = {p1, p2, p4}, the similarity becomes Similarity(MR,OR) = (Jacc(R1,r1) + Jacc(R2,r3) + Jacc(R3,r1))/3 = 0.61>0.58. Thus, the role $r2$ is used twice, and the similarity metric is enhanced by the role $R3$ that is not useful at all.

Alternatively, P. Streich et al. [12] propose a one-to-one matching for the comparison of two sets, based on the average Hamming distance[3]. Their method outputs a single permutation of all roles for the second set, that gives a one-to-one mapping between the same number of roles kept in the first set. This avoids the drawback of the previous proposition of mapping independently many discovered roles to the same original role. However, the method is not specifically designed for role mining applications, but for clustering in general. We think

[2]The Jaccard Coefficient and the Jaccard Distance are well-known measurements on the asymmetric information field. For non-binary data, the Jaccard Coefficient corresponds to $JC_{AB} = \frac{|A \cap B|}{|A \cup B|}$, and the Jaccard Distance to $JD_{AB} = 1 - JC_{AB}$.

[3]The Hamming distance between two vectors of n elements from an alphabet A: $a = (a_i)_{i \in [1,n]}$ and $b = (b_i)_{i \in [1,n]}$ is the number of elements in a that are different from the elements in b at the same positions: $Hamming\_Distance(a, b) = |\{i, a_i \neq b_i\}|$.

that the Hamming distance is not appropriate for role-to-role distance measure. In fact, it does not take into account the similarity of the roles, but only the difference (e.g., Hamming(r2,R1) = Hamming(r2,R2) = 2; but in reality r2 shares a permission with R1 and has nothing to do with R2).

More recently, Molloy et al. [13] has proposed a one-to-one mapping between the roles from two sets, also based on the Jaccard Coefficient. In order to evaluate the role stability in the presence of noise as defined in [14], they compare the two sets of roles generated by role mining over noisy data, and clean data. They assimilate the role mapping problem to a bipartite matching optimization problem known as *The Minimum Weighted Bipartite Matching Problem* (MWBM). Given two sets of roles SR1 and SR2, they create one vertex in V1 for each role in SR1 and one vertex in V2 for each role in SR2. For each pair of roles (R1,R2), such that R1 is included in V1, and R2 is included in V2, they add an edge (R1,R2) with weight Jaccard(R1,R2). Then, they run an algorithm solving the maximum weighted bipartite matching problem. The solution contains the closest pairwise matching by maximizing the global similarity metric. Each role in SR1 is matched with only one role in SR2, such that the sum of the distances is minimized. Finally, the metric is the sum of the distances between the matched roles. Applied to the motivation example proposed in section 3.2, this method will match the role R1 with r1 and R2 with r3, with a similarity metric value of 0.58. This method has addressed the problem of matching the same role in one set several times while other interesting roles are discarded. But the similarity measure is still straightforward. In general, we may wish to define similarity between sets of roles in a much more sophisticated fashion. For example, note that all the above measures still assume that a role can only be mapped to another role. Usually, this is not true. For instance, in the motivation example, it would be much more interesting for a security administrator to know that R1 is the union of r1 and r2, and that R2 is equal to r3 minus the permission shared with r1.

The previous work only focuses on the issue of comparing two sets of roles from the perspective of finding a similarity metric for the assessment of a set of roles with reference to another set of roles, and do not care about providing assistance to the administrator to analyze and leverage role mining results. In this paper, we consider the problem of comparing two sets of roles from a new perspective. Our approach allows a semantic analysis of the resulting set of mined roles in comparison with the preexisting set of roles, and the detection of misconfiguration of roles. We provide a tool that can assist a security administrator in the comprehension of the mined roles and how to enforce them.

4. Role Set Comparison Problem

We propose a new approach to the problem of comparing two sets of roles. Our approach targets to leverage all the relationships between roles from the two sets: merged roles, partitioned roles, hierarchical relationships, exceptions, etc. These relationships allow to analytically understand the new set of roles while leveraging the experience with the set of roles of reference. Our method to

reach this target is to express each role from one set as an algebraic formula of the roles from the other set. We find that the *Disjunctive Normal Form (DNF)* is a natural choice to structure this formula.

4.1. Problem Statement

The problem addressed in this paper is formally stated as follows:

Definition 5. *Role Set Comparison Problem*
Given two sets of roles OR and MR, both defined onto the same set of permissions PRMS, find for each role R in OR a minimal Disjunctive Normal Form DNF of roles in MR, such that DNF is included in R, and DNF maximally covers the permissions of R. By minimal DNF we mean that the size then the number of conjunctive clauses in the DNF are minimized.

Definition 5 proposes to express each role from OR with an algebraic formula of mined roles. The formula is presented as a disjunction of conjunctions of mined roles, i.e. a *Disjunctive Normal Form (DNF)* expression. The structure of a DNF fits naturally the requirement of projecting a role R from OR in MR. The permissions of the role R are divided into several roles in MR or gathered with other roles in a larger role, or both. Thus R may be expressed by a combination of unions of roles and/or intersections of roles. The conjunctions express hierarchical relationships: when the role R becomes an intersection of roles in the new configuration MR, this means that the projection in MR of R or of a subset of it is a super role. Moreover, the use of negation of roles in combination with conjunction aims to retrieve a role from a role expressing exceptions. The disjunctions accumulate and cover the permissions of R when they are spread in MR. The problem also states that the DNF should still be included in R. This condition guarantees the security rule, and avoid assigning extra-privileges to the users. Symmetrically, the problem states that the coverage of the permissions should be maximized since the purpose is to understand the projection of the role in the new configuration. This is an approximation problem since the exact DNF can not always be found because of role mining errors, exceptions and shadowed roles. This issue is handled in the next section more in depth. The existence of the exact DNF matching, and the complexity of the DNF, are good indications about whether the two sets of roles are comparable or not. Besides, there may be different possibilities of DNF from MR to maximally cover a given role R. In such case, we find the DNF which involves less conjunctions of roles in order to reduce the size of the obtained hierarchical relationships. In a second stage, we also minimize the number of clauses. Finally, we note that the comparison is intended to be from original roles to mined roles, but it obviously can be done in the opposite way.

4.2. Complexity

We demonstrate that the problem stated in Definition 5 is NP-complete. We first decompose the problem considering the comparison of one role to a set of roles, and then we reformulate it as a decision problem, in order to apply the NP Completeness theory.

Definition 6. *Role-to-Role Set Comparison Decision Problem*
Given a role R and a set of roles MR, defined onto the same set of permissions PRMS, and given two integers k and l, is there a Disjunctive Normal Form DNF of roles from MR, with less than l literals per clause and less than k clauses, such that DNF equals R?

To prove that the problem is NP-complete, we show that:

1. The problem is NP.
2. There exists another known NP-complete problem, any instance of which can be reduced in a polynomial time to an instance of our problem, such that resolving our problem infers resolving the other problem.

Definition 7. *Set Covering Problem*
Given a universe U, a family of subsets S of U and an integer k. A cover is a subfamily $C \subseteq S$ of sets whose union is U. Is there a cover of size at most k?

Theorem 1. The *Role-to-Role Set Comparison Decision Problem* is NP-complete.

Proof. First, the problem is NP since checking the validity of a solution, i.e. calculating the set of permissions of a DNF and comparing it with R and counting the number of clauses and the number of literals in each clause, can be done in polynomial time. Second, the set covering problem can be reduced to our problem. The universe U is assimilated to the role R. The family of subsets S is projected to the set of roles MR, which are sets of permissions. The size of the cover k is the same as the size of the DNF k, and we set l to 1. We get a special case of the *Role-to-Role Set Comparison Problem*, where all the roles of MR are included in R. Resolving this instance of the *Role-to-Role Set Comparison Problem* implies finding a DNF in MR, that maximally covers R with the minimal number of clauses and no conjunction at all. It infers finding a minimal cover for the universe U in S if such a cover exists. The transformation is obviously polynomial since it is a one to one mapping. □

5. Role Set Comparison Solution

In this section we provide an algorithm to solve the *Role Set Comparison Problem*. Afterwards, we analyze some important properties of our algorithm, and show its relevance in the context of role mining.

5.1. The Algorithm

Algorithm 1 is a greedy algorithm which solves the *Role Set Comparison Problem*. The algorithm takes as input two sets of roles, and gives as output, for each role R in the first set setR1, an algebraic expression formulated as a generalized union of intersections of roles taken in the second set of roles setR2 and their negations. The obtained expression, denoted DNF, is equal to R if any possibility to construct such an expression exists among setR2. Otherwise, it is the best approximation of R included in it

Algorithm 1 Compare(setR1,setR2)

Input: nRoles1×nPerms reference role-permission relation setR1
Input: nRoles2×nPerms compared role-permission relation setR2
Output: a DNF of roles from setR2 for each role in setR1. The DNF is equal to the role, or the closest included possible one if equality is not possible.

```
1: CandidateRoles ← setR2 ∪¬setR2
2: for each role R in setR1 do
3:    UncoveredPerms ← R
4:    DNF ← {}
5:    CandidateClauses ← CandidateRoles
6:    DiscardedClauses ← {}
7:    RIsCovered ← False
8:    ConjunctiveClauseLevel ← 1
9:    while ¬RIsCovered and CandidateClauses ≠ ∅ do
10:      for each clause C in CandidateClauses do
11:         if ¬RIsCovered then
12:            if C ⊆ R then
13:               if C ∩ UncoveredPerms ≠ ∅ then
14:                  DNF ← DNF ∪ C
15:                  UncoveredPerms←UncoveredPerms∩¬C
16:                  if UncoveredPerms == ∅ then
17:                     RIsCovered ← True
18:                  end if
19:                  for each clause C' in DNF do
20:                     if C' ⊂ (DNF - C') then
21:                        remove C' from DNF
22:                     end if
23:                  end for
24:               end if
25:               DiscardedClauses←DiscardedClauses∪C
26:            end if
27:         end if
28:      end for
29:      if ¬RIsCovered then
30:         ConjunctiveClauseLevel ← +1
31:         CandidateClauses←GenerateConjunctiveClauses(CandidateRoles,
            DiscardedClauses, ConjunctiveClauseLevel)
32:      end if
33:   end while
34: end for
35: return DNF for role R
```

The universe of literals which can be used to build the DNF is CandidateRoles, containing all the roles in setR2 in addition to the negation of each

Algorithm 2 GenerateConjunctiveClauses(CandidateRoles, DiscardedClauses,k)

Input: CandidateRoles: a role-permission relation representing the literals from which we build conjunctive clauses
Input: DiscardedClauses: a role-permission relation representing the set of clauses of less than k literals to discard from the resulting clauses
Input: k: the number of literals in each resulting clause
Output: a set of conjunctive clauses of k literals each

```
1: GeneratedClauses ← all the combinations of k roles from the CandidateRoles
2: for each clause C in DiscardedClauses do
3:   for each clause C' in GeneratedClauses do
4:     if C ⊂ C' then
5:       remove C' from GeneratedClauses
6:     end if
7:   end for
8: end for
9: return GeneratedClauses
```

of them (line1). The algorithm is structured in two main nested loops. The first one is a *for* loop which handles the roles of setR1 sequentially. For each role, we create a variable DNF which will contain the solution and a vector called UncoveredPerms initialized to the whole set of permissions of R (line 3–4). This vector indicates the current coverage of the permissions of the role R gradually with the temporary values of DNF during the progress of Algorithm 1. The second main loop is a *while* loop (line 9–33). It checks all the possibilities of disjunctive clauses from CandidateRoles until the role R is totally covered where the obtained DNF exactly matches R, or until all the combination of clauses are tested, where the obtained DNF is the best approximation of R. Since a DNF is a disjunction of conjunctions, the DNF is built by adding each time a new conjunctive clause to the main disjunction. A clause is a conjunction of 1 to nRoles2 literals from CandidateRoles. For this purpose, the clauses are tested one by one. The number of literals in the disjunctive clauses called ConjunctiveClauseLevel is increased gradually. If the role is not covered by the current ConjunctiveClauseLevel, Algorithm 2 is called to calculate all the combinations of conjunctive clauses of a higher ConjunctiveClauseLevel. DiscardedClauses are the clauses which are included in R and do not contribute in covering the UncoveredPerms more than the current DNF, or are already included in the DNF. The following running example illustrates the algorithm.

5.2. Running Example

We consider the configurations of roles in Table 2. We run the Algorithm 1 with the inputs (MR,OR).

Algorithm 1 projects the roles R1 and R2 in the set of roles OR sequentially. We handle only on the projection of role R1in this demonstration, thus, we

Table 2: Running Example

(a) Original Roles Matrix OR.

Original Roles	p1	p2	p3	p4	p5	p6	p7
r1	1	1	0	0	0	0	0
r2	1	0	1	0	0	0	0
r3	0	0	1	0	1	1	1

(b) Mined Roles Matrix MR.

Mined Roles	p1	p2	p3	p4	p5	p6	p7
R1	1	1	0	0	1	1	1
R2	0	0	1	0	0	0	0

will run the main *for* loop (line 2–35) only one time. So we focus on the rest of Algorithm 1. First of all, the candidate literals are CandidateRoles = {$r1$, $r2$, $r3$, $\neg r1$, $\neg r2$, $\neg r3$} (line 1). After the initialization step (line 3–8), we get: UncoveredPerms = R; DNF = {}; CandidateClauses = CandidateRoles; DiscardedClauses = {}; RIsCovered = False; and ConjunctiveClauseLevel = 1, which means that we will first try to cover R using disjunctions of roles from setR2. In the first iteration of the *while* loop (line 9–33) of Algorithm 1: We enter the *for* loop (line 10–28) which tests the clauses in CandidateClauses one by one. Only the clause r1 is included in R, and satisfies the conditions of line 12 and line 13. Thus, {r1} is added to DNF, and UncoveredPerms is updated to {p5, p6, p7} (line 14–15). At the end of the *for* loop (line 10–28), since the value of RIsCovered is still False, the number of literals per clause ConjunctiveClauseLevel is increased to 2, meaning that we will test the clauses of disjunctions of two literals next. Algorithm 2 is called to generate a new set of clauses (line 30–31) . The result of this call is CandidateClauses={$r2 \cap r3$, $r2 \cap \neg r1$, $r2 \cap \neg r3$, $r3 \cap \neg r1$, $r3 \cap \neg r2$, $\neg r1 \cap \neg r2$, $\neg r1 \cap \neg r3$, $\neg r2 \cap \neg r3$}. It is all the combinations of conjunction of two roles from CandidateRoles, minus the clauses involving DiscardedClauses. We notice that r1 does not take part in any of the candidate clauses, since it is a discarded clause now. We also exclude the clauses involving a role and its negation because it is the empty set. In the second iteration of the *while* loop (line 9–33) of Algorithm 1: the *for* loop (line 10–28) tests the new set of CandidateClauses one by one again. The first clause to pass the test in line 12 (inclusion in R) is $r3 \cap \neg r2$. This clause covers all the remaining UncoveredPerms. So DNF is updated to {{$r1$}, {$r3$, $\neg r2$}} which is interpreted: DNF = $r1 \cup (r3 \cap \neg r2)$. UncoveredPerms is updated to $\emptyset$ (line 15), and so RIsCovered is set to true (line 17). The *for* loop (line 19–23) checks that the new added clause does not cover the previously added clauses to DNF, in order to remove the obsolete clauses. There will be no further loops and no further generation of clauses of conjunction of higher number of literals because RIsCovered is set to true.

5.3. Properties of Algorithm 1

Theorem 2. Correctness. For each role R in OR, the returned DNF is included or equal to R.

Proof. Lines 12 and 13 of Algorithm 1 guarantee that the DNF is always a subset of or equal to R. □

Theorem 3. Completeness. For each role R in OR, if a disjunctive normal form of roles from MR equal to R exists, Algorithm 1 returns an exact matching DNF for R. If no exact matching DNF exists for the role R, Algorithm 1 returns a maximal DNF of roles from MR included in R (with respect to set inclusion).

Proof. Algorithm 1 tests exhaustively all the possible configurations in the worst case, enhancing the covering of R gradually. Thus the returned DNF maximally covers R. □

Theorem 4. Compactness. For each role R in OR, if several different disjunctive normal forms of roles from MR with the maximum coverage of R exist, Algorithm 1 returns a DNF with a minimal level of conjunctions and a minimal number of clauses with respect to that level.

Proof. The proof derives from the structure of the algorithm in two nested loops. The $ConjunctiveClauseLevel$ is increased gradually, after all the clauses in the lower conjunctive levels are tested. If no enhancement of the coverage can be achieved in a conjunctive level, no clauses are picked in that level. Besides, the number of clauses is minimal because a clause is added only if it enhances the coverage. Each time a new clause is added to the DNF, the for loop (line 19–23) in Algorithm 1 checks if any other clause previously added has became redundant and retrieves it from the DNF to keep the number of clauses minimal. A clause is redundant if all the permissions it covers are already covered by one or multiple other clauses. □

A Similarity Metric. The result of Algorithm 1 can be leveraged to obtain a similarity metric between two sets of roles. Indeed, we can sum the distances between each role from setR1 and its DNF in setR2, and then divide the sum by the number of roles in setR1 to get the distance between the two sets of roles:

$$similarity(R, DNF) = \frac{coveredPerms(R, DNF)}{Assigned\_Perms(R)}$$

and

$$similarity(setR1, setR2) = \frac{1}{|setR1|} \times \sum_{R_i \in setR1} similarity(R_i, DNF(R_i))$$

It should be noted that this similarity metric is not a 'metric' in the mathematical sense because it does not fill the symmetry property. Indeed, we can have $similarity(setR1, setR2) \neq similarity(setR2, setR1)$. Still, the obtained distance is a good indicator of how well the setR1 is represented in the setR2, and may be considered as more accurate than the metrics presented in the related work.

5.4. Time Complexity of Algorithm 1

The time complexity of Algorithm 1 depends on both the size and the nature of the input data. If the inputs are two sets of roles of size n and m respectively, then the worst case complexity is $O(n \times 4^m)$. The worst case is when the two sets of roles are not comparable, meaning that for each role in setR1 there exist no DNF of roles from setR2 that exactly matches with it. The algorithm will test, for each of the n roles of the first set, all the possible configurations of disjunction of conjunction of the m roles in setR2, increasing gradually the number of roles in the conjunction from one to 2m, and the outputted DNFs will be only approximations of the roles. Thus, we calculate the worst case complexity as $O(n \times \sum_{k=1}^{2m} C_{2m}^k) = O(n \times 4^m)$, with $C_n^k = \frac{n!}{k! \times (n-k)!}$.

The best case complexity is $O(n \times 2m)$, when the two sets of roles are identical. In such case, for each of the n roles from the first set, Algorithm 1 glances through the set of m roles and their negations only once and finds the exact match. We note that the complexity is independent from the number of permissions and users.

5.5. Heuristic to Reduce the Complexity

Algorithm 1 solves an NP-complete problem. The worst case time complexity is high. However, more than the size of the input data, the time complexity depends on the nature of this data. Indeed, the complexity is exponential on the number of roles from the second set when the compared sets of roles are incomparable, whereas the complexity is much lower when the sets of roles are comparable. Since we intend to use the algorithm in a context of role mining where the roles are comparable, we expect that the exact matching for each role exists and will be found. In addition, it will be generally found at a low level of ConjunctiveClauseLevel. Otherwise, we can guess that no matching DNF can be found for the handled role, and stop exploring further combinations of roles prematurely. An intuitive heuristic to reduce the time complexity of the algorithm is to set a limit for the ConjunctiveClauseLevel. We provide the possibility for the user of the algorithm to set a threshold, in order to avoid that the algorithm explores all the combinations involving clauses of cardinality beyond this threshold. Besides, such option may be required by the user. Indeed, a security administrator may prefer that the obtained DNFs keep simple and short for easier management than more complex and accurate. The threshold can be given as input to the algorithm and taken into account with a minor modification of Algorithm 1 by adding the condition (ConjunctiveLevel $\leq$ threshold) to the line 9. Algorithm 3 is a new version of Algorithm 1 where we show only the modified lines. To enhance the accuracy of matching between each role and its approximated DNF, the user can increase the threshold gradually.

For n mined roles, m original roles, and a threshold t of the maximal number of roles in a conjunction, the worst case complexity becomes $O(n \times \sum_{k=1}^{2t} C_{2m}^k) = O(n \times 4^m \times \frac{t}{m})$. The heuristic slightly decreases the execution time since t will

Algorithm 3 Compare(setR1,setR2,threshold)

Input: nRoles1×nPerms reference role-permission relation setR1
Input: nRoles2×nPerms compared role-permission relation setR2
Input: threshold is an integer that represents the maximum supported number of roles in a conjunctive clause in the returned DNF
Output: a DNF of roles from setR2 for each role in setR1. The DNF is equal to the role, or the closest included possible, with no more than *threshold* roles in each conjunctive clause of the DNF.

...
9: **while** ¬RIsCovered and CandidateClauses $\neq \emptyset$ and ConjunctiveClauseLevel $\leq$ threshold
...
35: **return** DNF for role R

be generally set much lower than m, and the complexity is still high. But this heuristic is applicable in the general case, to any sets of roles provided as input. To enhance the time complexity more significantly, we have to focus on the input data characteristics. We have to formally study the notion of *comparable* sets of roles, and its impact on the time complexity of the algorithm. In particular, since our intended context of application is role mining, we need to characterize more specifically the use cases related with role mining, where the exact match DNFs could or could not be found. Then we will be able to preprocess the input data accordingly, with the objective to avoid providing roles with no exact match DNFs to Algorithm 1. This approach is detailed in the following section.

6. Shadowed Roles Detection

In this section, we first investigate the characteristics of the input data and their impact on Algorithm 1 of role set comparison. We characterize, under realistic assumptions related to role mining context, some properties of roles that guarantee the existence of the exact matching DNFs. We demonstrate that there is a correlation between the misconfiguration problem of shadowed roles and the complexity of execution of Algorithm 1. Second, we focus on shadowed roles detection as a standalone problem. We provide a definition of the problem of shadowing. Finally we propose an algorithm that detects and reports shadowed roles in a given configuration of roles.

6.1. Detecting Shadowed Roles with the Role Set Comparison Algorithm

Based on our analysis of the the time complexity of the Algorithm 1, the execution time is expected to be low when the exact matching DNFs exists for all the roles. We are interested in characterizing the input data properties that guarantee the existence of the exact matching DNFs. Besides, though Algorithm 1 can compare two arbitrary sets of roles defined over the same set of permissions, it is intended to be used in the context of role mining. Hence, we

focus on the cases related to role mining, where the two sets of roles provided as input to Algorithm 1 are *equivalent*.

Theorem 5. Constrained Completeness Guarantees. Given two *equivalent* sets of roles RP1 and RP2; For any role R from RP1, if there exists a subset of k users such that $\bigcap_{i=1}^{k} assigned\_perms(U_i) = R$, then there exists at least a Disjunctive Normal Form DNF of roles from RP2 that exactly matches with R.

Proof. Since the two configurations of roles are *equivalent* according to Definition 4 , the k users which have exactly the permissions of the role R in common, must have exactly the same permissions in the two equivalent RBAC configurations. Let the roles assigned to user U_i in RP2 be referred as r_{ij}, j from 1 to l_i, where l_i is the number of roles assigned to the user U_i in RP2. Then we have: $assigned\_perms(U_i) = \bigcup_{j=1}^{l_i} r_{ij}$. R is the intersection of the permissions of the users U_i, i from 1 to k, thus it can be denoted $R = \bigcap_{i=1}^{k} \bigcup_{j=1}^{l_i} r_{ij}$, which is a Conjunctive Normal Form (CNF) of roles from RP2. Using the generalization of the Morgan's law [15], we can transform the obtained CNF to a DNF. □

Theorem 5, in combination with the Completeness Theorem 3, states a sufficient condition that guarantees the existence of exact matching DNFs for the roles. In the case of role mining without errors, the original and the mined sets of roles are *equivalent*. Algorithm 1 finds for each role from one set, that satisfies the condition mentioned in Theorem 5 (there exists a subset of k users such that $\bigcap_{i=1}^{k} assigned\_perms(U_i) = R$) a DNF of roles from the other set which exactly matches with it. We note that this condition is independent of the role mining technique used to calculate the sets of roles. Moreover, this condition is closely related with the concept of *role*. Indeed, a role can be viewed as a set of permissions assigned together to a set of users.

The following corollary states that the aforementioned condition is satisfied by the roles belonging to a non overlapping configuration of roles:

Corollary 1. Completeness Guarantees for Non Overlapping Roles. Given two "equivalent" sets of roles RP1 and RP2; If the roles of the set RP1 are not overlapping, then, for any role R in RP1, if R is assigned at least to a user, and there is no other role $R' \in$ RP1 such that assigned_users(R) $\subseteq$ assigned_users(R'), there exists at least a Disjunctive Normal Form DNF of roles from RP2 that exactly matches with R.

Proof. reductio ad absurdum:
Assume that $\bigcap_{U_i \in assigned\_users(R)} assigned\_perms(U_i) = R \cup P$ with P a set of permissions such that $P \cap R = \emptyset$ and $P \neq \emptyset$. Then P is the intersection of the permissions assigned to all the roles assigned to the users assigned to R, minus the permissions assigned to R.
Formally: $P = \bigcap_{U_i \in assigned\_users(R)} \left(\bigcup_{R_j \in assigned\_roles(U_i) \setminus R} assigned\_perms(R_j) \right)$.

However, there is no role shared by all the users assigned to R other than R. Since the roles are non overlapping. Then, $P = \emptyset$.
Consequently, $\bigcap_{U_i \in assigned\_users(R)} assigned\_perms(U_i) = R$, and R satisfies the condition of Theorem 5. □

Thus, if Algorithm 1 does not find an exact matching DNF for a given role in a context of role mining, this means that the conditions of Theorem 5 or Corollary 1 are not filled, and can be interpreted as a possible misconfiguration of that role in its original set of roles. The first possibility is that the User-to-Role assignment is incomplete, and that this role has not been assigned to any user yet. For instance, a job position has been created but the assigned employees are not hired yet. The second possibility is that the role is always assigned to the users simultaneously with another role, so the two roles can be merged in a single role. Another possibility is that the role is overlapping with other roles in the set, and some permissions of the role are shadowed by other roles. This can happen if the role is always assigned to users who receive the shadowed permissions from their other assigned roles simultaneously. The shared permissions can thus underline a misconfiguration of the role or a non optimal hierarchy in the structure of roles. To summarize, when used to compare two sets of *equivalent* roles, Algorithm 1 can warn about shadowed roles in the configuration of roles when an exact matching DNF for a certain role is not found.

But, the conditions of Theorem 5 or Corollary 1 are still restrictive, and they do not exhaustively specify when exact matching DNFS exists. For instance, they can not tell about overlapping roles. Besides Algorithm 1 can not exhaustively detect all the shadowed roles. Indeed, if the conditions are satisfied, then the algorithm will find exact matching DNFs for the roles, but if the Algorithm finds an exact matching DNFs this does not mean that the conditions are satisfied by the role. There may be still some undetected shadowed roles. Moreover, the detection of shadowed roles with Algorithm 1 is very costly in time. And the algorithm cannot tell if the role is entirely or partially shadowed. It would be more interesting to detect shadowed roles separately from the comparison algorithm, and then, handle the misconfiguration cause or discard the shadowed roles from their configuration, before comparing it with another configuration of roles. This would enhance the efficiency of the comparison algorithm. Besides, independently from the comparison of roles, exploring the shadowed roles in a given configuration of roles could be a useful application for a security administrator. For all these reasons, we now show how to handle the shadowing problem separately from the role set comparison problem.

6.2. Definition of Shadowed Roles

Towards handling the problem of shadowed roles detection, we have first to provide our formal definition of shadowed roles in a given role based access control configuration.

Definition 8. *Shadowed Role*
A role R is shadowed *in a given RBAC configuration RC = (ROLES, UR, RP) if it matches one or several of the following cases:*

1. *R is not assigned to any user in UR: i.e. assigned_users(R) = ∅.*
   or
2. *There exists (at least) another role R′ in ROLES that has always similar user assignment as R: i.e. assigned_users(R) = assigned_users(R').*
   or
3. *R is overlapping with one or several roles in ROLES, and there exists (at least) a permission p ∈ assigned_perms(R) such that the users assigned to R have always p in their other assigned roles.*

The idea behind Definition 8 is that a role is considered as shadowed in a given access control configuration if we cannot find it in the user-to-permission assignment relation *UPA*. Typically, a role mining algorithm is not likely to calculate such a role since it is based on the *UPA* relation to extract the roles. The first case of shadowing is when the role has not been assigned to any user. The second case of shadowing is when the role is always assigned together with another role, which may indicate that it may be merged with another role. Finally, the third case of shadowing may occur only in RBAC configuration with overlapping roles. Different roles may share a subset of permissions. Shadowing happens when one or several permissions of a given role are never assigned to a user who is not already assigned these permission through his other roles. In this case, removing the permission from the role will not affect the *UPA* relation. An example of this situation is provided in the Table1(a) of the afore presented motivation example. The permission $p2$ is shared by the roles $r1$ and $r3$. However, the role $r3$ is never assigned to a user who has not also the role $r1$. So the permission $p2$ is actually shadowed in role $r3$.

6.3. Shadowed Roles Detection Algorithm

We define a new algorithm that examines an RBAC configuration and detects the shadowed roles.

Algorithm 4 takes as input the matrices of user-to-role assignment UserRole and role-to-permission assignment RolePermission of an RBAC configuration. We Consider that a role is characterized by the set of permissions assigned to it, since the user-to-roles assignment is more dynamic and changes more often. The set of roles is then the raws of the matrix RolePermission. The algorithm checks the roles in the RolePermission matrix for the three cases of shadowing presented in Definition 8 and reports for each role if it is shadowed or not. For partially shadowed roles, the algorithm also reports which are the affected permissions.

First of all, Algorithm 4 calls the function *Check For Partitions Of Roles* (line 1). This function checks all the roles of the configuration for the second case of shadowing stated in Definition 8. It detects the roles with similar sets of assigned_users. These roles are partitions of a larger role. Each column in

Algorithm 4 Detect Shadowed Roles(UserRole, RolePermission)

Input: UserRole: a user to role assignment matrix
Input: RolePermission: a role to permission assignment matrix
Output: A specification for each role in UserRole matrix: whether it is "not shadowed", "not assigned", or "shadowed" with the specification of the "shadowed permissions".

```
1: CandidateRoles ← Check For Partitions Of Roles(UserRole)
2: NBUPA ← UserRole × RolePermission
3: RecurrentPerms ← permissions assigned to a user more than once in
   NBUPA
4: if RecurrentPerms ≠ ∅ then
5:   for each role R in CandidateRoles do
6:     RIsShadowed ← False
7:     ShadowedPerms ← {}
8:     if assigned_users( R ) ≠ ∅ then
9:       Report: "The role <R> is not assigned to any user."
10:    else
11:      ProblematicPerms ← assigned_perms(R) ∩ RecurrentPerms
12:      for each permission p in ProblematicPerms do
13:        OnceUsers ← assigned_users( R ) ∩ assigned_users_only_once( p)
14:        if OnceUsers == ∅ then
15:          RIsShadowed ← True
16:          ShadowedPerms ← ShadowedPerms ∪ {p}
17:        end if
18:      end for
19:      if RIsShadowed then
20:        Report: "The role <R> is shadowed. The shadowed permissions
           are <ShadowedPerms>."
21:      else
22:        Report: "The role <R> is not shadowed."
23:      end if
24:    end if
25:  end for
26: else
27:  for each role R in CandidateRoles do
28:    if assigned_users( R ) ≠ ∅ then
29:      Report: "The role <R> is not assigned to any user."
30:    else
31:      Report: "The role <R> is not shadowed."
32:    end if
33:  end for
34: end if
```

the matrix UserRole represents the set of assigned_users of a role. The function

Algorithm 5 Check For Partitions Of Roles (UserRole)

Input: RolePermission: a role to permission assignment matrix
Output: NonPartitionsRoles
Output: This algorithm should check for Roles that are always assigned together to the same users and report them as shadowed roles.

1: Apply a Hash function on each column of RolePermission
2: Sort the columns of RolePermission according to the Hash values
3: NonPartitionsRoles ← Roles with a unique instance of Hash Value
4: Report: "The roles in <RolePermission∪NonPartitionsRoles> are shadowed. They are partitions of roles."
5: **return** NonPartitionsRoles

Check For Partitions Of Roles, which pseudocode is presented in Algorithm 5 applies a hash function to each column in the matrix, then it sorts the columns based on the hash values. Thus roles that share exactly the same set of assigned_users are gathered since they have the same hash value. Each subset of roles that share the same users are reported as partitions of a single role. They are reported as *Shadowed roles*. The function returns the remaining roles, that present unique hash value, because they do not fit the second case of shadowing considered in Definition 8. They become *CandidateRoles* to be checked for the two other cases (1 and 3) of shadowing in Definition 8.

The *CandidateRoles* may be totally shadowed if they are not assigned to any user, and this is easy to check. By contrast, the third case of shadowing, is more complex to be verified because a role may be partially shadowed by one or several roles. Only overlapping roles are potentially affected by this case. The partially shadowed roles involve necessarily overlapping roles that have been assigned simultaneously to a set of users. Consequently, a subset of permissions are assigned multiple times to the same users. This is why we use the algebraic multiplication instead of the boolean multiplication traditionally used in role mining issues: the algebraic multiplication gives how many times the same permission has been assigned to a user. We calculate the non boolean user to permission assignment matrix *NBUPA* (line 2). In this matrix, each column represents a permission, each row represents a user, and the cell c_{ij} represents how many times the permissions p_j has been assigned to the user u_i. Then, from this new matrix *NBUPA*, we deduce the set *RecurrentPerms* (line 3) that contains the permissions that have been assigned to a user, through more than one role. This set represents the only potentially shadowed permissions.

If the set *RecurrentPerms* is empty, then there exist no overlapping roles assigned simultaneously to the users, and consequently, there are no roles affected by the third case of shadowing in the Definition 8. We move to lines 26 to 34 where we check if each role has been assigned at least to a user. If *RecurrentPerms* is not empty (line 8), then there are overlapping roles which are assigned together to users in the input RBAC configuration. So there is a risk of partially shadowed roles. Algorithm 4 handles the *CandidateRoles* one

by one in the *for* loop in line 5. First, it checks if the role has been assigned at least to a user. The algorithm simply looks at the column correspondent to the role in the UserRole matrix to get the assigned_users(R). If the column is null, then the role is not assigned to, and is considered shadowed and we move to the next role. If there is a non null value in this column the role is assigned to one or several users, and we need to check if some of the assigned_permissions(R) are shadowed. We have to handle the permissions assigned to the current role one by one to ensure that each permission has been assigned at least to one user only from the role R. This proves that the permission is not shadowed in the role. The role is not shadowed only if all its permissions assigned to R are not shadowed in it. Otherwise, the role is shadowed, and the current permission is one of the shadowed permissions. But hopefully, we do not need to check all the permissions assigned to a role since only the set of *RecurrentPerms* are potentially shadowed permissions. So, for each role R, we calculate the intersection between its assigned_roles(R) and the *RecurrentPerms*, and we call it the *ProblematicPerms* (line 11). Finally, for each problematic permission, we calculate the intersection between the set of users assigned to the role, and the set of the users to whom the permission has been assigned from only one role, which we can find directly in the matrix *NBUPA* (see line 13). If the set is empty then the role is shadowed, and we add this permission to the set of *ShadowedPerms* in the current role, before we move to exploring the remaining permissions in *ProblematicPerms*, to get the exhaustive set of shadowed permissions in this role. Otherwise, the current permission is not shadowed in this role, and the algorithm continues testing the other roles in *ProblematicPerms*. The role is reported to be unshadowed only if all the *ProblematicPerms* are not shadowed in it.

Theorem 6. Completeness. Given two matrices UR and RP; For any role R from RP, if R fills one or several of the shadowing cases specified in Definition 8 then Algorithm 4 will report it as a shadowed role.

Proof. The function *Check For Partitions Of Roles* (Algorithm5) checks all the roles for the case 2 of shadowing. All the roles that are not in the case 2 of shadowing are set to *CandidateRoles*. In Algorithm 4, all the *CandidateRoles* are checked for the case of shadowing, either in the for loop (lines 5 to 9) if the roles are overlapping or in the for loop (lines 27 to 30) if the roles are not overlapping. The case 3 of shadowing is possible only in overlapping configurations of roles, and only overlapping roles are assigned together to the same users. Then, the *If* condition (line 8) ensures that all the *CandidateRoles* are checked for this case of shadowing only if there exist permissions assigned more than once to a user. These roles that are not concerned with the case 1 of shadowing are tested for the case 3 of shadowing. For each of these roles, the algorithm verifies that each permission of the role has been assigned at least once to a user whose other roles does not grant this permission. Otherwise the role is reported to be shadowed. □

Time Complexity. Algorithm 4 is a light application that permits to detect and report the shadowed roles in a given RBAC configuration. Its time complexity is polynomial since it is O(n×m×k) with n the number of users, m the number of permissions and k the number of roles in the input RBAC configuration. We are much far from the exponential complexity of Algorithm 1. Moreover, we can avoid the worst case complexity of Algorithm 1 if we preprocess its input with Algorithm 4.

7. Experimental Results

7.1. RBAC State Generator

We have implemented a random RBAC configuration generator to obtain the synthetic data input for our experiments. This tool takes as input the characteristics of an RBAC configuration, notably the number of users, the number of permissions and the number of roles. In addition, it uses two parameters: the density of the user-role relation *UR*, and the density of the role-permission relation *RP*. The outputs are the three boolean matrices *UR*, *RP*, and *UPA* where $UPA = UR \otimes RP$. The matrices UR and RP are randomly generated based on the provided densities which determine the probability to get 1 in a particular cell. The obtained configuration may contain partially overlapping roles, and may assign multiples roles to a user. We have implemented the plateform of test in MATLAB_R2011a. All the experiments have been made with a 2.26 GHz Intel Core 2 Duo processor on Mac OS X V 10.7.2.

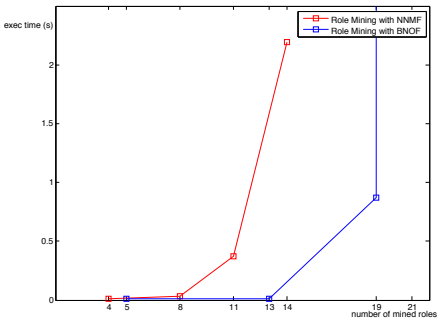
7.2. Experiment with the Role Set Comparison Algorithm

For this section, we generate eight data sets with different sizes described in Table 3 with the density parameter set to 0.1. The results of the experiments are summarized in Figure 1.

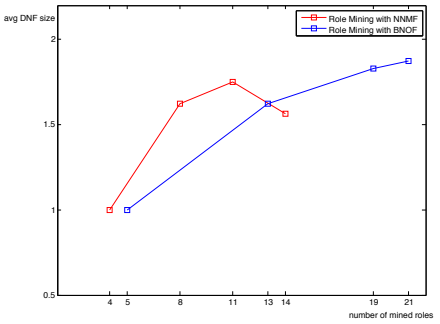
Table 3: Size of the Original RBAC Data Sets Used for the Role Set Comparison Algorithm Experiemnts

n	nUsers	nPerms	nOR
1	10	15	4
2	18	25	8
3	25	30	12
4	35	40	16
5	200	350	20
6	400	500	30
7	500	800	40
8	600	1000	50

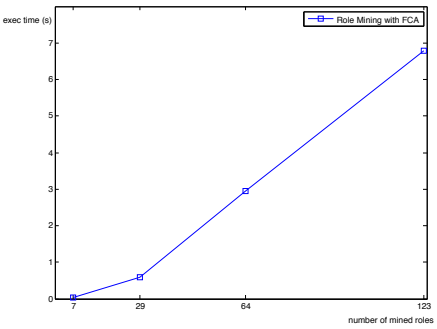
First, we use the data sets 1, 2, 3 and 4 in Table 3 for role mining tests. We provide the *UPA* matrix of each data set as input to a given role mining algorithm, and we compare the original roles in *UR* with the mined roles in UR^* using the Algorithm 1. We test three different role mining algorithms.



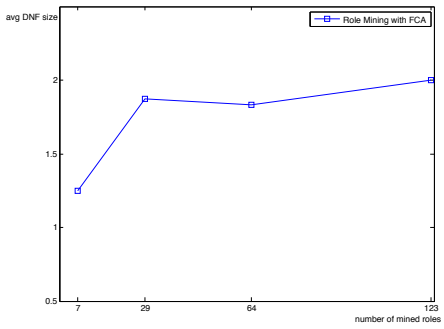
(a) Execution time when comparing roles of the RBAC data sets 1, 2, 3 and 4 with the mined roles by NNMF and BNOF



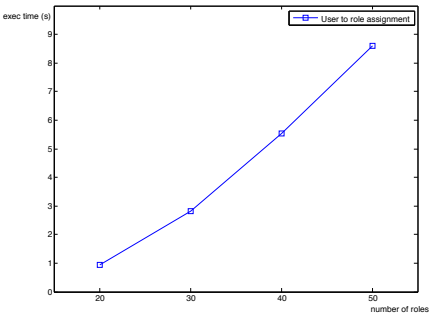
(b) Average DNF size with NNMF and BNOF



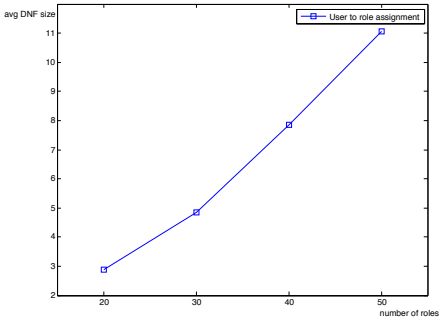
(c) Execution time when comparing roles of the RBAC data sets 1, 2, 3 and 4 with the mined roles by FCA



(d) Average DNF size with FCA



(e) Execution time when comparing *UPA* with *RP*



(f) Average number of assigned roles to each user in RBAC data sets 5, 6, 7 and 8.

Figure 1: Experimental Results with the Role Set Comparison Algorithm

We adapt two techniques of matrix compression usually used in data mining, namely the *Non Negative Matrix Factorization (NNMF)* [16] and the *Binary Non-Orthogonal Decomposition (BNOF)* [17]. Both techniques take an *UPA* matrix from a data set and give an approximate decomposition of the matrix into the product of two matrices of lower rank, namely *UR* and *RP*. This is done by emphasizing the frequent patterns in the original matrix, which can be interpreted as roles in the context of role mining. However, the two techniques differ in their constraints and optimization objective. NNMF gives a factorization into two matrices of non negative integers, while minimizing the root-mean-squared. We transform the obtained positive matrices into boolean matrices. BNOF uses another decomposition technique which gives a boolean decomposition of overlapping roles, but a user can be assigned to only one role. The results of this first series of tests with NNMF and BNOF are given in Figure 1 (a) and (b). The purpose is to observe the behavior of Algorithm 1 when comparing two sets of roles which have been configured with different objectives. All role mining processes presented in this paper have finished without approximation errors. The number of original roles rise from 4 to 16, but the number of mined roles with NNMF and BNOF is slightly different from the number of original roles. In Figure 1 (a), we plot the execution time of Algorithm 1 in function of the number of mined roles because, as stated in the section 5.4, the complexity of the algorithm mainly depends on this latter parameter. The execution time increases with the number of mined roles. Besides, the curve of the role mined with BNOF shows an exponential increase in the fourth data set (19 mined roles, 253 s). This is because there is an original role for which no exact DNF can be calculated. This role is shadowed in the original configuration of roles. Thus, Algorithm 1 reaches the worst complexity case for this data set. The execution time does not only depend on the number of mined roles, but it is mostly dependent of the nature of the compared roles and how similar they are. In Figure 1 (b), we plot the average size of the obtained DNFs for each data set. By size of DNF we mean here the number of involved roles, no matter if in conjunctive or disjunctive clauses. The figure shows that the average size of the DNFs also increases with the number of mined roles. We notice a correlation between the size of the DNFs and the execution time. In particular, the execution time grows with the maximum number of conjunctions in the DNFs, which increases from one to three in this series of tests.

The third role mining algorithm we use is a *Formal Concept Analysis (FCA)* algorithm [18]. Considering the users as objects, and the permissions as their attributes, FCA calculates a lattice of all the possible formal concepts. We consider each Formal concept as a mined role. Role Mining algorithms in literature usually process to a pruning of the lattice to discard a subset of the concepts with regard to some optimization criteria, but in this paper, we keep all the concepts. This explains the high number of mined roles increasing from 7 to 123 in Figure 1 (c) and (d), corresponding to sets of original roles of only 4 to 16 roles. Our purpose is to test the ability of Algorithm 1 to compare a hierarchical configuration of roles with a flat configuration of roles. We aim also at testing the scalability of Algorithm 1 in such cases. The results in Figure 1 (c) and (d)

show that the algorithm execution time scales well with the number of mined roles, where all the roles can match their exact DNFs. This is also explained by the fact that the average size of the obtained DNFs is still between 1 and 2, meaning that Algorithm 1 succeeds in matching the original roles with the appropriate roles in the hierarchical configuration of roles. Actually, the size of the DNFs varies between 1 to 6, and the number of conjunctions does not exceed three roles, which is an intelligible amount of data that can be managed by a security administrator.

Finally, we use the remaining last four sets of data in Table 3, to test the ability of Algorithm 1 to perform a user to role assignment. We provide as input both the *UPA* (which may be considered as pseudo-roles as explained previously), and the set of roles generated by the random generator. The results are given in Figure 1 (e) and (f). Figure 1 (e) shows that the algorithm scales well with large data sets, since the execution time does not exceed 9 seconds. All the users are assigned to their appropriate roles, and the average size of the DNFs presented in Figure 1 (f) going up to 11 roles is simply the average number of roles assigned to each user, since no conjunctions has been involved in the generated DNFs in all the four tests.

7.3. Experiments with the Shadowed Roles Detection Algorithm

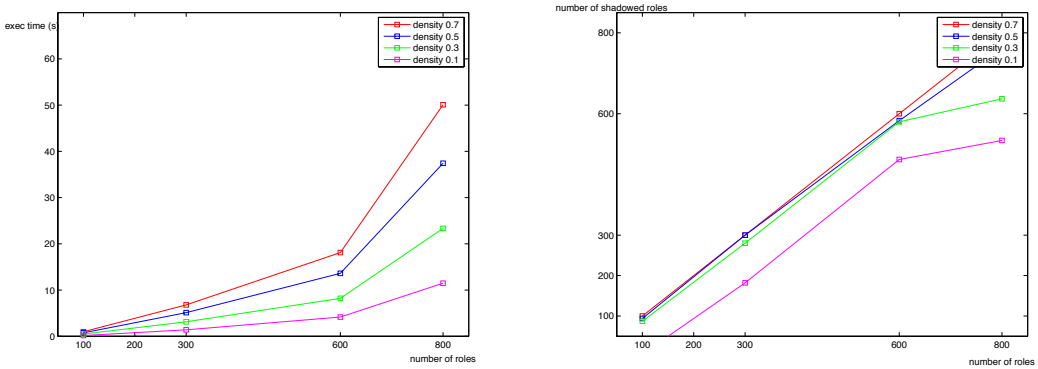
To test the Shadowed Role detection tool described in Algorithm 4, we have generated a set of RBAC configurations. The sizes of these configurations are presented in Table 4. For each size, we have generated a series of RBAC configuration with different values of the density parameter, namely: 0.1, 0.3, 0.5 and 0.7. Figure 2 summarizes the results from these experiments. In Figure 2(a), we show the execution time of the algorithm with each of the data size in function of the number of roles in the RBAC configuration. We see that the execution time increases with the number of roles, but it is still low (few seconds) even in the higher case of 800 Roles. In real application, the number of roles is not executed to exceed one thousand. Moreover, since we have generated sets of roles with similar sizes but different densities, we can notice in Figure 2(a) that the execution time is higher for sets of roles with higher density. By increasing the density, the number of shadowed roles increases in the configuration, especially the number of partially shadowed roles. The random decision of the generator in the assignment of the permissions to the roles is likely to result in more misconfiguration of roles than in real configurations. Algorithm 4 is sensitive to the existence of overlapping roles because it performs an additional processing in this case (see Algorithm 4 - line 8). Figure 2(b) presents the number of shadowed roles detected by the Algorithm 4, and confirms that the number of shadowed roles increases when the density increases.

8. Conclusion

Despite the importance of RBAC systems in today organizations, security administrators still lack for automated tools to assist them in the tasks related

Table 4: Size of RBAC Data Sets Used for the Shadowing Detection Algorithm

n	nUsers	nPerms	nRoles
9	800	1000	300
10	1000	1200	600
11	1500	2000	800



(a) Execution time as function of the number of roles

(b) Number of shadowed roles in the data sets

Figure 2: Experimental Results with the Shadowed Roles Detection Algorithm

to the evaluation and management of roles. Such tasks are very sensitive for the security of the organization system, and it is hard and risky to handle them manually. In this paper, we have addressed two important issues related to role management.

First, we have focused on the comparison of two configurations of roles, which is an important issue in the research and the application of Role Based Access Control, particularly with the accession of the role mining techniques. In this paper, we have provided a formal approach to handle the problem of comparing two sets of roles. We have stated the Role Set Comparison Problem (RSCP) as the problem of finding for each role in one set an expression of a disjunctive normal form of roles from the other set. We have demonstrated that RSCP is NP-complete. Afterwards, we have presented a greedy algorithm which solves it. We have proved the correctness and the completeness of the comparison algorithm. We have evaluated the time complexity of our algorithm. Finally, we have presented some experimental results which validate the efficiency of our algorithm in comparing configurations of roles defined with different criteria, such as structured and non structured hierarchies of roles. We have also tested the algorithm skills in deploying an RBAC policy by assigning the mined roles to the users.

Second, we are interested in the problem of detecting shadowed roles in an

RBAC configuration. Shadowing is usually due to a misconfiguration of roles. We have formally defined the shadowing cases. We have provided a solution to detect shadowed roles. And, we have formally linked the problem of shadowed roles to the problem of comparing sets of roles. Indeed, in most cases related to role mining, preprocessing the compared sets of roles by eliminating shadowing from the set of roles significantly lowers the complexity of the RSCP solution. We have finally presented the experimental results that show the efficiency and the scalability of the shadowing detection solution. As perspective of this work, we plan to integrate these algorithms in RBAC administration tools.

Acknowledgment

For this research, Safaa Hachana was partially supported by the ITEA2 Role-ID (Grant agreement no.08007) and Nora Cuppens-Boulahia, Frédéric Cuppens and Joaquin Garcia-Alfaro are partially supported by the ITEA2 Predykot project (Grant agreement no.10035).

References

[1] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman, Role-based access control model, IEEE Computer 29 (1996) 38–47.

[2] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, R. Chandramouli, Proposed *NIST* Standard for Role-Based Access Control, standard, NIST, 2001.

[3] M. P. Gallaher, A. C. O'Connor, B. Kropp, The Economic Impact of Role-Based Access Control, Technical Report, RIT for NIST (National Institute of Standards and Technology), 2002.

[4] I. Molloy, N. Li, T. Li, Z. Mao, Q. Wang, J. Lobo, Evaluating role mining algorithms, in: Proceedings of the 14th ACM symposium on Access control models and technologies SACMAT '09, ACM, 2009, pp. 95–104.

[5] J. Vaidya, V. Atluri, Q. Guo, N. R. Adam, Migrating to optimal *RBAC* with minimal perturbation, in: Proceedings of the 13th ACM Symposium on Access Control Models and Technologies Proceedings SACMAT '08, ACM, 2008, pp. 11–20.

[6] S. Hachana, F. Cuppens, N. Cuppens-Boulahia, J. Garcia-Alfaro, in: In Proceedings of the 7th International Conference on Availability, Reliability and Security ARES'12.

[7] J. Vaidya, V. Atluri, Q. Guo, The role mining problem: finding a minimal descriptive set of roles, in: Proceedings of the 12th ACM Symposium on Access Control Models and Technologies SACMAT'07, ACM, 2007, pp. 175–184.

[8] M. Frank, J. M. Buhmann, D. Basin, On the definition of role mining, in: Proceeding of the 15th ACM symposium on Access control models and technologies SACMAT '10, ACM, 2010, pp. 35–44.

[9] M. Kuhlmann, D. Shohat, G. Schimpf, Role mining - revealing business roles for security administration using data mining technology, in: Proceedings of the 8th ACM symposium on Access control models and technologies SACMAT '03, ACM, 2003, pp. 179–186.

[10] J. Vaidya, V. Atluri, J. Warner, Roleminer: mining roles using subset enumeration, in: Proceedings of the 13th ACM Conference on Computer and Communications Security CCS'06, ACM, 2006, pp. 144–153.

[11] H. Takabi, J. B. Joshi, Stateminer: An efficient similarity-based approach for optimal mining of role hierarchy, in: Proceedings of the 15th ACM symposium on Access control models and technologies SACMAT '10, ACM, 2010, pp. 55–64.

[12] A. P. Streich, M. Frank, J. M. Buhmann, Multi-assignment clustering for boolean data, in: Proceedings of the 26th Annual International Conference on Machine Learning, ACM, 2009, pp. 969–976.

[13] I. Molloy, N. Li, Y. A. Qi, J. Lobo, L. Dickens, Mining roles with noisy data, in: Proceedings of the 15th ACM symposium on Access control models and technologies SACMAT '10, ACM, 2010, pp. 45–54.

[14] A. Colantonio, R. D. Pietro, A. Ocello, N. V. Verde, Mining stable roles in *RBAC*, in: Proceedings of the 24th International Information Security Conference IFIP SEC'09, Springer, 2009.

[15] R. L. Goodstein, "Boolean Algebra", Dover Publications, 2007.

[16] D. D. Lee, H. S. Seung, Algorithms for non-negative matrix factorization, in: Neural Information Processing Systems (NIPS), pp. 556–562.

[17] M. Koyuturk, A. Grama, N. Ramakrishnan, Nonorthogonal decomposition of binary matrices for bounded-error data compression and analysis, ACM Transactions on Mathematical Software 32 (2006) 33–69.

[18] A. Colantonio, R. D. Pietro, A. Ocello, Leveraging lattices to improve role mining, in: Proceedings of the 23rd International Information Security Conference IFIP SEC'08, Springer, 2008, pp. 333–347.