How to Find a Rigorous Set of Roles for Application of RBAC

Lijun Dong, Xiaojun Kang, Maocai Wang
School of Computer, China University of Geosciences, Wuhan, P.R. China
Email: donglijun.cug@gmail.com

***Abstract*—Role-based access control (RBAC) has been adopted successfully by a variety of security system by reducing the complexity of the management of access control. The least privilege principle is a very important constraint policy of RBAC. Devising a complete and correct set of roles for supporting the least privilege principle has been recognized as one of the most important tasks in implementing RBAC. A key problem is how to find such sets of roles which have the least permissions. In fact, when the number of role-permission assignments is large, it is almost impossible to find a rigorous set of roles which has the completely same set of permissions required by a user. To address this problem, we research the problem how to find such the rigorous combinations obeying the principle of least permissions. By bringing forward the concept of the l*east privilege mining problem,* we describe the methods to resolve the problem and some instances of its applications, too. Moreover, the corresponding algorithms are displayed. Specially, by analyzing the complexity of least privilege mining problem, the method based on evolutionary algorithm is shown appreciate. Correspondingly, the experiments are accomplished to prove our opinions. Finally, the paper is concluded and some future work is posed.**

***Index Terms*—information security, role-based access control, least privilege, evolutionary algorithm**

I. Introduction

It is well known that the principle of least privilege is a design principle to which access control models and systems should adhere. In role-based access control (RBAC), roles represent organizational agents that perform certain job functions within the organization. Users, in turn, are assigned appropriate roles based on their qualifications [1-3]. One of the major tasks in implementing RBAC is to enforce the principle of least privilege by managing role assignments.

Principle of least privilege means that in a computing environment, every module (such as a process, a user or a program on the basis of the layer we are considering) must be able to access only such information and resources that are necessary for its legitimate purpose [4,5]. However, when there is great number of permissions and roles, it is very troublesome to find an appropriate set of roles that owns a minimal set of permissions. In fact, when the number of both roles and permissions are large, it is almost impossible to find a completely accurate set of roles which has the completely same set of permissions required by a user. Therefore, an important problem is how to find such combinations and which combination has the least permissions? In other words, which role set is the most rigorous for application of RBAC? We call it LPMP (Least Privilege Mining Problem).

To explore this problem, we consider several LPMPs including the basic LPMP ant the approximate LPMP which can formally define the least privilege mining problem in RBAC. We also introduce some methods to resolve these problems.

This paper is organized as follows. In section 2 we review the RBAC model and some preliminary definitions employed in the paper. In section 3, some related researches are introduced. In section 4, we define our basic least privilege mining problem as well as its variations, followed by section 5 in which we explore the algorithms to address these problems. The experiments based on the algorithms are displayed in section 6. Finally, section 7 concludes our work and provides some insight into our ongoing and future research.

Manuscript received March 25, 2011; accepted September 7, 2011.

II. Related Work

A number of approaches have been proposed in the literature to accomplish the task of ensuring the least privilege principle in access control system.

Timothy E. Levin etc. extends the separation kernel abstraction to represent the enforcement of the principle of least privilege [6]. Levin introduces an approach that supports an orthogonal, finer-grained flow control policy by extending the granularity of protected elements to subjects and resources. In fact, this is just an access control solution.

For RBAC, the least privilege principle can be enforced by multiple ways. An important strategy is *constraints* [1,2] in RBAC, which is implemented by *separation of duty* (SoD). SoD can be static or dynamic, that has been described in [7,8].The next generation of RBAC will be dynamic activation and revocation of roles [9]. The detailed mechanism of dynamic activation and revocation of sessions is given in [10]. The separation of duty is implemented to avoid one man control. By dynamic SoD, the least privilege principle is enforced indirectly [11].

Besides, Muhammad Asif Habib has described the complexities and complications which can be faced after implementing separation of duty in terms of mutually exclusive roles (MER) [12]. According to the point of Habib, both mutual exclusion and role inheritance will affect least privilege. Liang Chen etc. defines a family of a simple roles-based models that provide support multiple hierarchies and temporal constraints [13]. By this means, the inter-domain role mapping (IDRM) problem is brought forward. Then, it is implemented to investigate least privilege and the IDRM problem in the presence of multiple role hierarchies and temporal constraints.

All these researches are helpful for our work. They provide some interesting ideas for our research on the least privilege problem.

Last but not least, it must be mentioned that our work benefits from the research of Jaideep Vaidya in [14] greatly. The content of Vaidya's paper is to define the role mining problem (RMP) for discovering an optimal set of roles from existing user permissions in RBAC. In fact, its main field is role engineering [15] but not least privilege. But the methods of defining problems of this paper provide an important reference for our work. We introduce the basic LPMP, δ-Approx LPMP and MinNoise LPMP that do formally root in the definitions of the basic RMP, δ-approx RMP and the Minimal Noise RMP. The main relation between our work and Vaidya's is that the same formalized methods are applied into the different sides of RBAC.

III. Preliminaries

We adopt the NIST standard of the Role Based Access Control (RBAC) model. For the sake of simplicity, we restrict ourselves to RBAC0 without sessions, role hierarchies or separation of duties constraints in this paper.

RBAC Model [1,2]

- *U, ROLES, OPS, and OBJ are the set of users, roles, operations and objects.*
- *$UA \subseteq U \times ROLES$, a n-to-n mapping user-to-role assignment relation.*
- *PRMS (the set of permissions) $\subseteq \{(op, obj) | op \in OPS \wedge obj \in OBJ\}$.*
- *$PA \subseteq PRMS \times ROLES$, a n-to-n mapping of permission-to-role assignments.*
- *$UPA \subseteq U \times PRMS$, a n-to-n mapping of user-to-permission assignments.*
- *$assigned\_users(r) = \{u \in U | (u,r) \in UA\}$, the mapping of tole r onto a set of users.*
- *$assigned\_permissions(r) = \{p \in PRMS | (p,r) \in PA\}$, the mapping of role r onto a set of permissions.*
- *$required\_permissions(u) = \{p \in PRMS | (u,p) \in UPA\}$, the mapping of user u onto a set of permissions.*

From definition 1, we can conclude that a role is really a set of permissions, so there is a relation: $ROLES \subseteq 2^{PRMS}$.

In this paper, any subset of relation UA is represented as a boolean matrix, called UR matrix, denoted as *M(UA),* where a *true* in cell $\{ur\}$ indicates the assignment of role *r* to user *u*. Similarly, any subset of relation PA can be represented as a boolean matrix, called RP matrix, denoted as *M(PA),* where a *true* in cell $\{rp\}$ indicates the assignment of permission *p* to role *r*. Finally, any subset of relation UPA can be represented as a boolean matrix, called UP matrix, denoted as *M(UPA),* where a *true* in cell $\{up\}$ indicates the assignment of permission *p* to user *u*. Here, $u \in U$, $r \in ROLES$, and $p \in PRMS$. Any one-dimensional set can be regarded as a relation with its owner, and its relation matrix is denoted as M(D). For simplicity we denote *true* as *1* and *false* as *0*.

For UP matrix, RP matrix and UP matrix, operations are necessary. They are all Boolean matrix, so we introduce a operation named boolean matrix congregation (BMC). BMC between boolean matrices $A \in \{true, false\}^{m \times k}$ and $B \in \{true, false\}^{k \times n}$ is $C = A © B$ where each cell of C should be *true* if and only if the corresponding cells of both A and B are *true*.

For two relation sets X, Y, and their relation matrixes *M*(*X*), *M*(*Y*), it is claimed that *M*(*X*) is δ-Consistency by *M*(*Y*) or *M*(*Y*) is within δ of M(X) if and only if $Y \subseteq X \;\wedge\; |X - Y| \leq \delta$, denoted as $M(X) \succ_\delta M(Y)$.

For UR matrix *URM*, *RP* matrix RPM, and UP matrix *UPM*, $URM © RPM \succ_\delta UPM$ means that *UPM* should be within δ of the user-permission matrix generated from *URM* and *RPM.* When δ=0, a user will obtain not only all permissions which he requires, but also none of permissions which he does not require. δ-Consistency binds the degree of difference between UA, PA, and UPA.

Now, based on these preliminaries, we will formalize the least privilege mining problem: how to find a minimum set of roles which contains all essential permissions with minimum redundancy.

IV. Least Privilege Mining Problem

A. Basic LPMP

Giver a user *u*, there may be a corresponding set of permissions expressed as *required_permissions(u)*. In order to implement this relation, *u* must be mapped onto a corresponding set of roles expressed as *assigned_user⁻(u)* where *assigned_user⁻* is the inverse function of *assigned_user*. Therefore, the basic LPMP for *u* asks us to find a user-to-role assignment UA such that UA and the role-to-permission assignment PA can exactly describe the user-to-permission assignment UPA while excluding any redundant permission. Put another way, LPMP require us to find an optimum combination of roles for user to follow the least privilege principle.

LPMP can be described formally as shown in Figure 1.

Figure 1 shows a process: Given a set of permissions PRMS, a set of roles ROLES, a role-to-permissions assignment PA, a user-to-permissions assignment UPA, and a user u, find a user-to-role assignment UA for user u, with M(UA)©M(PA) being 0-Consistency by M(UPA).

LPMP:
Conditions:

$$\begin{cases} u \in U \\ S = required\_permissions(u) \\ Qu = \{(u,p) \mid p \in S\} \end{cases}$$

Aim:

$$\begin{cases} Pu = \{(u,p) \mid p \in T\} \\ M(Pu) \succ_0 M(Qu) \end{cases},$$

Subject to:

$$\begin{cases} T = \bigcup_{r \in A} assigment\_permissions(r) \\ A \ = assigned\_user^{-}(u) \subseteq ROLES \end{cases}$$

Figure 1. The basic least mining problem (LPMP)

For an application system equipped with RBAC, roles are steadier than users. When a user needs to access the system, he will always hope for all permissions what he wants. However, there may be any combination of roles including the same permissions exactly. For example, supposed that there are 10 roles for 1000 permissions, a user requests 120 permissions. Which roles should be assigned to this user? There are 2^{10}-1 possible combinations of roles, that should contain more than one combination which covers all of the 120 permissions only if its any role is not be eliminated.

Therefore it could hardly be accomplished to find an exact match with 0-consistency at most time.

B. A Typical Eaxmples

The basic LPMP provides us an important reference for researching the least privilege mining problem. But it is not appreciate for most real instances. For illuminating this point, in this section, we introduced several typical examples to discuss

In Figure 2, we have 30 permissions and 6 roles. Now a user needs 10 permissions. Obviously, the circumstance of Fig.2 is very troublesome. Its relations can be displayed:

Roles' assignment:

– *assigned_permissions(r1)=* {*p2,p3,…,p9,p15*};
– *assigned_permissions(r2)=* {*p2,p3,…,p6,p12,p30*};
– *assigned_permissions(r3)=* {*p20,p21,…,p28* };
– *assigned_permissions(r4)*
 = {*p1,p9,p10,…p17,p19*};
– *assigned_permissions(r5)*
 = {p2,p3,p16,p17,…,p21};
– *assigned_permissions(r6)=* {*p1,p2,p29*};
– *assigned_permissions(r7)=* { *p2,p3…,p16,p29*};

User' assignments:

– *case 1: required_permissions*(*u*)
 = {*p1,p2,…,p9,p15,p29*};

In this case, user will need role r1 and r6 which will exactly match the permissions required by user. In other word, there exists a 0-consistency assignment for case 1.

– *case 2: required_permissions*(*u*)
 = {p2,p3,…,p6,p9,p15};

In this case, both {*r1*} and {*r1,r2*} will contain the permissions, but none is 0-consistency. {*r1*} is 3-consistency; {*r1,r2*} is 5-consistency.

– *case 3: required_permissions*(*u*)
 = {p1,p2,p3,p10,…p21};

In this case, {*r3,r4,r5*} is a 7-consistency assignment.

– *case 4: required_permissions*(*u*)
 = {p1,p2,…,p9, p12,p29};

This case is similar to case 3.

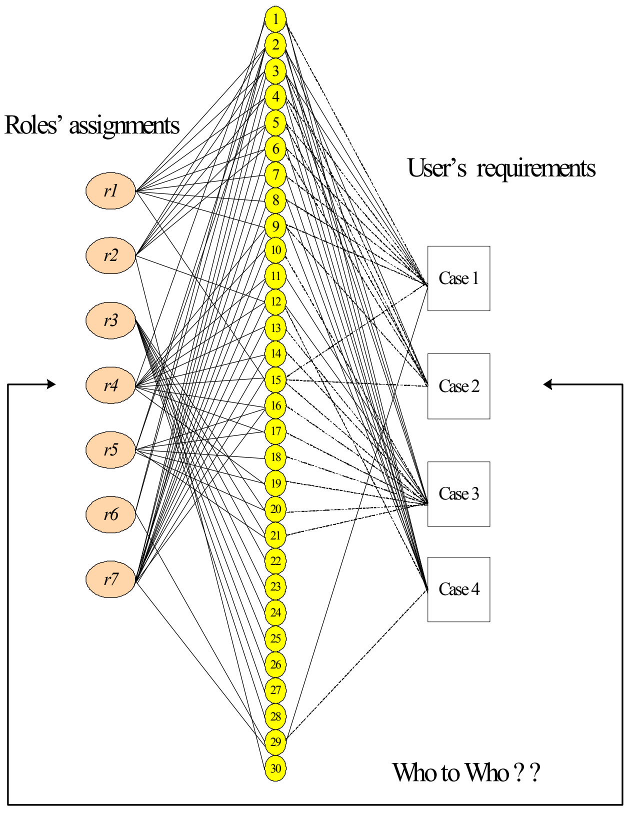

Figure 2. A typical example for LPMP

As a matter of fact, sometime we will find there is not any 0-consistency assignment thoroughly because of the original architecture of an RBAC system. For a simple example:

– *PRMS* = { *p1, p2, p3, p4, p5, p6*};
– *assigned_permissions(r1)* = {*p1, p2, p4, p6*};
– *assigned_permissions(r2)* = {*p1, p2, p5, p6*};
– *assigned_permissions(r3)* = {*p2, p3, p5*};
– *required_permissions(u)* = {*p1, p3, p5* }

It can be seen that there does not exist any set of roles matching user's request exactly but 3 approximate sets {*r1,r3*}, {*r2,r3*} and {*r1,r2,r3*}. Further, we can find that {*r2,r3*} is the best one with 2-consistency redundancy only. {*r2,r3*} will be the most appropriate redistribution for security administrator.

So, we can see that it is very hard to find an exact match for LPMP. Furthermore, if permissions are assignment to roles at random, it is almost impossible to find an exact 0-consistency match for user's permissions [14]. If we allow approximate matching – i.e. if it is good enough to match all permissions required by user besides some redundant permissions. As long as we make the redundant permissions few enough, we might consider that we have found the least permissions combination. This significantly reduces the burden of maintenance on the security administrator while generating only a few redundant permissions which may be shielded specially. As a matter of fact, sometime we will find there is not any 0-consistency assignment thoroughly because of the original architecture of an RBAC system.

So we should consider the approximate LPMP. Approximate match might be a prudent choice for an RBAC system with dynamic user-permission assignments.

C. Approximate LPMPs

In this section, we introduce two approximate LPMPs: δ-Approx LPMP and MinNoise LPMP

Definition 1 (*δ*-Approx LPMP).
Given a set of permissions PRMS, a set of roles ROLES, a role-to-permissions assignment PA, a user-to-permissions assignment UPA, and a user u, find a user-to-role assignment UA for user u, with M(UA)©M(PA) being δ-Consistency by M(UPA) and minimizing the number of roles.

δ-Approx LPMP:
Conditions:

$$\begin{cases} u \in U \\ S = required\_permissions(u) \\ Qu = \{(u,p) \mid p \in S\} \\ \delta = C \ (C \text{ is a constant}) \end{cases}$$

Aim:

$$\begin{cases} B \subseteq ROLES \\ T = \bigcup_{r \in B} assigment\_permissions(r) \\ Pu = \{(u,p) \mid p \in T\} \\ M(Pu) \succ_x M(Qu) \end{cases}$$

Subject to:

$$\begin{cases} AS = \{A \mid A = assigned\_user^{-}(u)\} \\ B \in AS \\ \forall A \in AS, \ |B| \le |A| \\ x \le \delta \end{cases}$$

Figure 3. *δ*-Approx LPMP

Obviously, the basic LPMP is simply a special case of the *δ*-Approx LPMP with *δ*=0 and no minimizing the number of roles. The reason of restricting the number of roles is that there may be more than one combination of roles with δ-Consistency but we prefer to the one with least roles. The merit of minimizing the number of roles is that less roles would be convenient for the pre-distribution for security administrator.

δ-Approx LPMP brings a more flexible method for security administrator if only restrict the value of *δ*. However, sometimes user mainly cares the least permissions but not the least roles for stricter security principle. Consequently, instead of bounding the approximation, and minimizing the number of roles, it might be interesting to do the reverse: bound the number of roles, and minimize the approximation. We call this the Minimal Noise Least Privilege Mining Problem (MinNoise LPMP). The security administrator might want to do this when he is looking for the top-k roles that can contain the problem space well enough, and are still robust to noise. Actually, MinNoise LPMP and *δ*-Approx LPMP are important complementarities of two sides of LPMP.

Definition 2 (MinNoise LPMP). *Given a set of permissions PRMS, a set of roles ROLES, a role-to-permissions assignment PA, a user-to-permissions assignment UPA, and a user u, find a user-to-role assignment UA for user u, where* ***not more than k*** *roles will assure that M(UA)©M(PA) is δ-Consistency by M(UPA) with minimizing the δ.*

MinNoise LPMP:
Conditions:

$$\begin{cases} u \in U \\ S = required\_permissions(u) \\ Qu = \{(u,p) \mid p \in S\} \\ k = C \ (C \text{ is a constant}) \end{cases}$$

Aim:

$$\begin{cases} B \subseteq ROLES \\ T = \bigcup_{r \in B} assigment\_permissions(r) \\ Pu = \{(u,p) \mid p \in T\} \\ M(Pu) \succ_{\varphi} M(Qu) \\ \varphi = \min\{\delta \mid M(Pu) \succ_{\delta} M(Qu)\} \end{cases}$$

Subject to:

$$\begin{cases} AS = \{A \mid A = assigned\_user^{-}(u) \ \wedge \ |A| \le k\} \\ B \in AS \end{cases}$$

Figure 4. MinNoise LPMP

D. Application of LPMPs

For an RBAC application, actually, different initializations will correspond to different solutions for LPMP. If you are fortunate enough, maybe a basic LPMP (0-approx LPMP) would take effect, otherwise you must select δ-Approx LPMP or MinNoise LPMP. In order to clarify these problems further by means of an example, we will discuss several cases by Table III that shows a sample user-permission assignment (UP matrix: $M(UPA)$) with 3 users and 5 permissions.

Firstly, Table I (b) shows an ideal user-role assignment (UR matrix: $M(UA)$) by which a role-permission assignment (RP matrix: $M(PA)$) in Table I (a) can completely describe the given user-permission assignment ($M(UA)©M(PA)=M(UPA)$) for all of three users.

TABLE I. USER-PERMISSION ASSIGNMENT (UP MATRIX)

	p1	*p2*	*p3*	*p4*	*p5*
u1	1	0	0	1	0
u2	1	0	1	1	1
u3	0	0	1	1	1

TABLE II. BASIC LPMP

(a) RP assignment relation matrix

	p1	*p2*	*p3*	*p4*	*p5*
r1	0	0	1	1	1
r2	0	1	0	1	1
r3	1	0	0	1	0

(b) UR assignment relation matrix

	r1	*r2*	*r3*
u1	0	0	1
u2	1	0	1
u3	1	0	0

TABLE III. δ-APPROX LPMP

(a) RP assignment relation matrix

	r1	*r2*
u1	0	1
u2	1	1
u3	1	0

(b) UR assignment relation matrix

	p1	*p2*	*p3*	*p4*	*p5*
r1	0	0	1	1	1
r2	1	0	0	1	0

TABLE IV. MINNOISE LPMP

(a) RP assignment relation matrix

	r1	*r2*
u1	0	1
u2	1	1
u3	1	0

(b) UR assignment relation matrix

	p1	*p2*	*p3*	*p4*	*p5*
r1	0	0	1	1	1
r2	1	0	0	1	1

Secondly, Table II(a) depicts the optimal user-role assignment ($M(UA)$) under the conditions of a role-permission assignment ($M(PA)$) in Table II(b). In this case, both *u2* and *u3* have still user-role assignments that can completely describe the given user-permission assignment, but *u1* could only get a 1-consistent assignment. It must be mentioned that {*r1,r3*} is also a 1-consistent assignment for u1, but it is eliminated because of redundant roles.

Lastly, Tables III(a) and 6(b) show the optimal user-role assignment and role-permission assignment for MinNoise LPMP with k=2.

V. EVOLUTIONARY ALGORITHM FOR LPMPS

We have had the formal definitions of the least privilege mining problem. But we have not the corresponding formal algorithms still. In fact, the algorithm to address LPMP is very important.

In the former examples, we can straightly calculate the appreciate assignments for a user without any algorithm, since there are a few of roles and permissions. However, when there are a great number of roles and permissions, it is almost impossible to calculate straightly. Given a case with an accurate number of roles and permissions, there will be a great deal of probability about how to assign roles to user. Only those combinations that can cover the request of user are appropriate.

A. Traditional Algorithm

When both m and n are finite number, 2^n-1 and k must be finite, too. So we can design two simple algorithms by searching all of combinations.

As long as computation environment is good enough, the final correct combination will must be found. We call such algorithms *ordinary LPMP algorithms* (OLAs).

We have defined three formal least privilege mining problems: the basic LPMP, δ-Approx LPMP and MinNoise LPMP. Since the basic LPMP is simply a special case of the δ-Approx LPMP with δ=0 and no minimizing the number of roles, we need only two algorithms for δ-Approx LPMP and MinNoise LPMP.

Figure 5 shows an algorithm for δ-Approx LPMP, called ALPMP.

Figure 6 shows an algorithm for MinNoise LPMP, called MLPMP.

The key steps of ALPMP are:

1. Line 1-4: Some conditions.
2. Line 5-6. Some initialization operations are performed. *S* is initialized as *ROLES* for maximizing |*S*|, because we need pick out a minimal set.
3. Line 8-16. Find all appreciate sets of roles for *u*. All subsets of *ROLES* are traversed.
4. Line 18-24. Pick out the minimal set of roles from all appreciate sets found in step 2.

5. Line 25: Get the final results.

Algorithm 1 (OLA 1): ALPMP

```
1   input: $u \in U$, $\delta$
2         $Q$=$required\_permissions(u) \subseteq PRMS$, $Q \neq \varnothing$
3   output: $S \subseteq ROLES$
4
5   {Initialization operations}
6   $AS=\varnothing$, $V=\varnothing$, $P=\varnothing$, $S=ROLES$
7
8   {Find all appreciate sets of roles for $u$}
9   for each $R \in 2^{ROLES}$
10    for each $r \in R$
11      $P=P \cup assignment\_permission(r)$
12    end for
13    if $Q \subseteq P \wedge |P-Q| \leq \delta$
14      $V=V \cup \{R\}$
15    end if
16  end for
17
18  {Find the minimal set of roles}
19  for each $v \in V$
20    if $|v|<|S|$
21      $S=v$
22    end if
23  end for
24
25  $assigned\_user(u)^{-}=S$
```

Figure 5. The ordinary algorithm for δ-Approx LPMP

Algorithm 2 (OLA 2): MLPMP

```
1   input: $u \in U$, $k$
2         $Q$=$required\_permissions(u) \subseteq PRMS$, $Q \neq \varnothing$
3   output: $S \subseteq ROLES$
4
5   {Initialization operations}
6   $AS=\varnothing$, $W=\varnothing$, $P=\varnothing$, $\Gamma=|PRMS|$, $S=\varnothing$
7
8   {Find all appreciate sets of roles for $u$}
9   for each $R \in 2^{ROLES}$
10    for each $r \in R$
11      $P=P \cup assignment\_permission(r)$
12    end for
13   if $Q \subseteq P \wedge |R| \leq k$
14      $W=W \cup \{(R, P)\}$
15   end if
16  end for
17
18  {Find the most appreciate set of roles}
19  for each $(x,y) \in W$
20    if $|y|<\Gamma$
21      $\Gamma=|y|$
22      $S=x$
23    end if
24  end for
25
26  $assigned\_user(u)^{-}=S$
```

Figure 6. The ordinary algorithm for MinNoise LPMP

The key steps of MLPMP are:

1. Line 1-4: Some conditions.
2. Line 5-6. Some initialization operations are performed.
3. Line 8-16. All subsets of *ROLES* are traversed. All appreciate sets of roles for *u* and corresponding sets of permissions are cached by relation *PA*.
4. Line 18-25. Pick out the set of roles that contains the least permissions from all appreciate sets found in step 2.
5. Line 26: Get the final results.

B. Complexity of LPMP

The mechanisms of OLA are very simple. But its complexity needs to be considered much. OLS need search the whole space of solutions of LPMP, so its efficiency is not so satisfying.

Given a case with an accurate number of roles and permissions, there will be a great deal of probability about how to assign roles to user. When n roles hold m permissions averagely, there are 2^n-1 combinations of roles. Clearly, when n is very large, such approaches are computationally heavy, since searching and computing 2^n combinations will take an exponential time in the number of roles.

In fact, all LPMPs are optimization problems. It can be proved that all LPMPs are NP- complete.

First, we select a basic NP- complete problem:

Definition 3 (SBP: Set Basis Problem) [14]. Given a collection C of subsets of a finite set S, and a positive integer K $\leqslant$|C|, is there a collection B of subsets of S with |B| = K such that, for each c $\in$ C, there is a sub-collection of B whose union is exactly c?

This problem has been proved NP- complete [14].

For the basic LPMP, S demotes the assignments of role-permission. C denotes user-permission assignments. Every set c $\in$ C stands for one user u. Therefore, LPMP can be mapped to the SBP directly by polynomial transformation. By this way, LPMP can be proved NP-completes [16], too.

The basic LPMP is NP- complete, and the approximate LPMPs are the variations of the basic LPMP. We have seen that not only δ-Approx LPMP but also MinNoise LPMP does not change the real matter of the basic LPMP. In fact, he approximate LPMPs are more common than the basic LPMP. So the approximate LPMPs should be NP- complete [14,16].

About how to detail the complexity of LPMP, this would go beyond the scope of this paper. We will further research this problem in our future work.

We have known LPMP is NP-complete. OLA algorithm is an immediate way to address LPMP, so OLA is NP-hard, too. Now our task is to construct a simple substitute algorithm to solve this NP-hard problem.

C. Evolutionary LPMP Algorithm (ELA)

In order to constructing a simple substitute algorithm to solve the NP- complete LPMP, we bring forward an method based on evolutionary algorithm [17,18].

Evolutionary algorithm can perform well approximating solutions to all types of problems because they ideally do not make any assumption about the underlying fitness landscape; this generality is shown by successes in fields as diverse as engineering, art, biology, economics, marketing, genetics, operations research, robotics, social sciences, physics, politics and chemistry.

An EA uses some mechanisms inspired by biological evolution: reproduction, mutation, recombination, and selection. Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines the environment within which the solutions "live". Evolution of the population then takes place after the repeated application of the above operators.

Algorithm 3: MELA

```
1   input: $u \in U$, $k$
2           $Q$=$required\_permissions(u) \subseteq PRMS$, $Q \neq \varnothing$
3   output: $S \subseteq ROLES$
4
5   {Initialization operations}
6   $CHS$=$\varnothing$, $OES$=$\varnothing$
7   $gen$=0, $MaxGen \leftarrow$ a constant, $mxm$=0
8   Generate $n$ $k$-length chromosomes whose each
    locus denotes a role
9   $CHS \leftarrow n$ chromosomes
10  $objc$ is a chromosomes variable
11
12  {Search the excellent chromosomes}
13  while $gen < MaxGen$
14     $OES = \varnothing$
15     $TmpS$=$CHS$
16     crossover: recombine the $n$ chromosomes in
         $CHS$ by crossover operations
17     mutation: mutate each chromosomes of $CHS$
18      $TmpS$=$CHS \cup TmpS$
19     selection: select $n$ chromosomes from $TmpS$
         by the degressive order of $oe$
20     $CHS \leftarrow n$ chromosomes
21     for each $ch \in CHS$
22        $oe$ = Objective($ch$, $Q$)
23        $OES = OES \cup \{(ch, oe)\}$
24     end for
25  end while
26
27  {Search the most excellent chromosome}
28  for each $(x, y) \in OES$
29     if $y >= mxm$
30        $y$=$mxm$
31        $objc$=$x$
32     end if
33  end for
34
35  Compute the role set $S$ by chromosomes $objc$
36  $assigned\_user(u)^{-}$ =$S$
```

Figure 7. The evolutionary algorithm for MinNoise LPMP

We introduce an evolutionary LPMP algorithm (ELA) to resolve our problems. In ELA, each combination of roles is look upon as a chromosome. By through a series of operations such as selection, crossover and mutation, some more excellent chromosomes are generated recursively, until the termination conditions are triggered.

Evolutionary algorithm can be used to resolve many actual problems. Due to limited space, we only introduce a simple algorithm, called MELA that adapts to MinNoise LPMP. In fact, minimal noise problem is very general in the actual applications [6], not only the basic LPMP but also the δ-Approx LPMP can be resolved with ELA on basis of MELA.

Algorithm MELA is displayed in Figure 7. Its key steps are:

1. Line 5-10. Some initialization operations are performed. Each possible combination of roles is mapped onto a chromosome.
2. Line 12-25. Through the recursive operations, some more excellent chromosomes are held. In this process, EvauFun, an evaluation function of chromosome, is used. The algorithm EvauFun is shown in Figure 8.
3. Line 27-33. Pick out the most excellent chromosome form the result in step 2.
4. Line 35-36. Reconstruct the object set of roles by the chromosome in step 3.

Algorithm 4: Objective

```
1   input: $ch$ , where $ch$ is a chromosome
2           $Q \subseteq PRMS$, $Q \neq \varnothing$
3   output: $e$ , where $e$ is a real number
4
5   $P$=$\varnothing$
6   for each locus of $ch$
7      Generate the role $r$ corresponding to this locus
8      $P$=$P \cup assignment\_permission(r)$
9   end for
10  if $Q \subseteq P$
11        $e = 1 - \frac{|P| - |Q|}{|Q|}$
12  else
13     $e$=-1
14  end if
15  return $e$
```

Figure 8. The evaluation function of chromosome

In most of real applications of EAs, computational complexity is a prohibiting factor. In fact, this computational complexity is due to fitness function evaluation. Fitness approximation is one of the solutions to overcome this difficulty. However, seemingly simple EA can solve often complex problems; therefore, there may be no direct link between algorithm complexity and problem complexity.

In our MELA, there is a fitness function named **Objective** at line 22 in Figure 7. Algorithm Objective is displayed in Figure 8. Its key steps are:

1. Line 6-9. For each locus of a chromosome, compute the role set corresponding to this locus.
2. Line 10-14. Compute the relative error.
3. Line 15. Return the relative error which is the fitness of a chromosome.

VI. EXPERIMENTS

We have the ordinary LPMP algorithm (OLA) and the evolutionary LPMP algorithm (ELA). In this section, we will compare the two kinds of algorithms by some experiments.

For simply displaying our experiments, we introduced several abbreviations about experimental parameters:

– NoR: the number of roles, corresponding to the constrained number of roles—k in MinNoise LPMP.
– NoT: the times of an experiment with a fixed value of NoR.
– TC: the average time consumed when finds out a fitness solution for MinNoise LPMP.
– RE: the relative error which is materially the error of every final fitness solution relative to the ideal solution.

Now, we will present some experimental results obtained by applying the OLA and ELA to several cases about MinNoise LPMP and compare their performance. We have implemented OLA and ELA with Matlab and C++. So we will display two groups of experiments which are performed with Matlab and C++ respectively.

Our experiments are performed on a DELL PC (2GHz Intel Core Duo, 2GB 667MHz DDR2 SDRAM).

Besides, in our experiments, the number of permissions is 3000 and the set of permissions required by user is generated at random each time. But both OLA and ELA use just the same condition parameters each time.

A. Experiment by Matlab

TABLE V. EXPERIMENT OF OLA (MATLAB)

NoR	NoT	TC(seconds)	RE(acute value)
5	10	0.029	0.241
6	10	0.057	0.026
7	10	0.116	0.117
8	10	0.257	0.173
9	10	0.581	0.079
10	10	1.306	0.111
11	10	2.883	0.144
12	10	6.999	0.089
13	10	16.407	0.097
14	10	42.779	0.181
15	10	121.628	0.061
16	6	391.546	0.064
20	3	90045.208	0.047
300	—	—	—

In this experiment, both OLA and ELA are implemented by using MATLAB 7.0. The results of OLA and ELA are displayed in Table V and Table VI respectively.

TABLE VI. EXPERIMENT OF ELA (MATLAB)

NoR	NoT	TC(seconds)	RE
5	10	2.320	0.241
6	10	3.288	0.026
7	10	3.252	0.117
8	10	3.318	0.173
9	10	3.928	0.079
10	10	4.218	0.111
11	10	5.052	0.144
12	10	5.32	0.089
13	10	6.01	0.122 ↑
14	10	5.965	0.181
15	10	5.464	0.061
16	6	6.387	0.074 ↑
20	3	5.355	0.057 ↑
300	1	320.186	0.086

There are 10 groups of tests with the number of roles being not more than 15; there are 6 groups with 16 roles, 3 groups with 20 roles. There is 1 group test with 300 roles for ELA only, since it could hardly be accomplished by using OLA for too many roles.

Experiment by C++

TABLE VII. EXPERIMENT OF OLA (C++)

NoR	NoT	TC(seconds)	RE(acute value)
5	20	0.023	0.248
6	20	0.04	0.109
7	20	0.072	0.112
8	20	0.201	0.076
9	20	0.488	0.102
10	20	0.73	0.15
11	20	1.219	0.142
12	20	3.46	0.07
13	20	9.303	0.095
14	20	14.338	0.173
15	20	45.06	0.055
16	15	127.5	0.028
20	10	16000.2	0.06
300	—	—	—

TABLE VIII. EXPERIMENT OF ELA (C++)

NoR	NoT	TC(seconds)	RE
5	20	1.12	0.248
6	20	1.154	0.109
7	20	2.501	0.112
8	20	2.8	0.086 ↑
9	20	3.02	0.102
10	20	2.88	0.152 ↑
11	20	3.336	0.142
12	20	3.093	0.092 ↑
13	20	3.6	0.095
14	20	4.206	0.173
15	20	3.885	0.055
16	15	4.38	0.029 ↑
20	10	4.46	0.06
300	10	288.46	0.075

In this experiment, both OLA and ELA are implemented by using C++. The results of OLA and ELA are displayed in Table VII and Table VIII respectively.

Because C++ is more efficient than Matlab, we can perform more tests in this experiment. But for NP-hard OLA, when there are too many roles, it is still too hard to accomplish. We do 20 groups of tests with no more than 15 roles; 15 groups with 16 roles, 6 groups with 20 roles. We perform 10 group tests with 300 roles for ELA only but not for OLA.

C. Analysis of Experimrnts

From the experimental results we can conclude:

1. With not only Matlab but also C++, when the number of roles is small, OLA is more efficient than ELA. But when the number of roles exceeds 11, ELA is clearly more efficient than OLA. Furthermore, along with the increase of roles, ELA's superiority is more and more obvious.
2. ELA owns better robustness than OLA. When there 20 roles, ELA still needs only 4~6 seconds, but OLA needs more than 90000 seconds for Matlab or 16000 seconds for C++. Even in the face of 300 roles, ELA can still finish the test in limited time. For OLA, however, we can not accomplish the test on our PC at all, with using not only easy Matlab but also efficient C++.
3. The relative error of OLA is exact, which can reach to the theoretic minimum. Correspondingly, the results of ELA are rough a bit, which have been marked with "↑" in Table VI and VIII. But this is tolerable relative to its efficiency. From Fig.9, we can see this point, too.

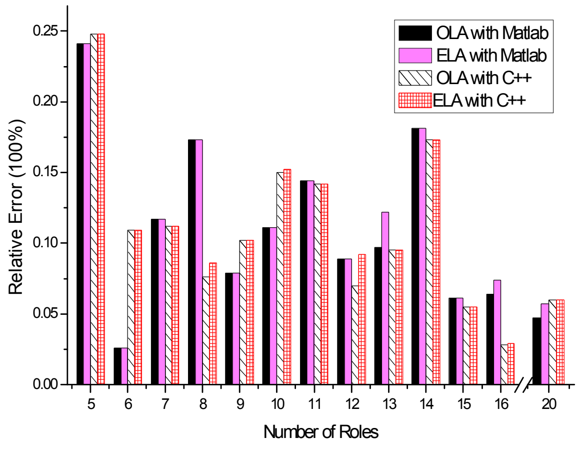

Figure 9. Comparison of OLA & ELA

It can be concluded that when need to assign a few roles, OLA is more efficient, and otherwise, ELA is a more appreciate substitute.

VII. Conclusion

In this paper, we have formally defined *the least privilege mining problem* (LPMP) for RBAC. Besides the basic LPMP, we define the approximate LPMPs including not only δ-Approx LPMP but also MinNoise LPMP. It has been mentioned that the basic LPMP is a special case of δ-Approx LPMP. Both δ-Approx LPMP and MinNoise LPMP are useful in the real RBAC applications. The former emphasizes the scale of roles but the latter emphasizes the minimal redundancy of permissions, while they all provide the least privilege principle for applications. Some material cases are discussed to clarify these problems further. The algorithms are displayed simply in order to clarify how to resolve LPMP.

The main contributions of this paper is to provide a material aim for research the least privilege principle of RBAC by mapping this problem to some formal definitions in mathematics. But there is some insufficiency that is our future work.

1. For simplify the work, we restrict ourselves to RBAC0. However the standard RBAC is more complicated. We need to consider the standardization and application of LPMP in the complete RBAC.
2. We introduce two kinds of simple algorithms: OLA and ELA. Despite ELA is more efficient in general, its LPD is not so accurate as OLA's. We need research more appreciate evolutionary algorithms to improve our work.
3. We have provide the definition of LPMP, but there is not a complete model including some necessary formalized definitions and predicates. This will be our future work.

Acknowledgment

This research is supported by by the Special Fund for Basic Scientific Research of Central Colleges, China University of Geosciences (Wuhan) No. CUGL100228.

References

[1] American National Standards Institute, Inc.: American National Stnadard for Inormation Technology-Role Based Access Control (ANSI INCITS359-2004), 2004.
[2] R.S. Sandhu et al. Roles-based Access Control Models. IEEE Computer, p38-47, Febuary 1996
[3] D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli. Proposed NIST Standard for Role-based Access Control. TISSEC, 2001
[4] F.B. Schneider. Least privilege and more [computer security]. IEEE Security & Privacy, 2003, 1(5):55-59.
[5] Maxwell Krohn, Petros Efstathopoulos, Cliff Frey et al. Make least privilege a right (not a privilege). Proceedings of the 10th conference on Hot Topics in Operating Systems, 2005, 10:21-29.
[6] Timothy E. Levin, Cynthia E. Irvine and Thuy D. Nguyen. Least Privilege in Separation Kernels. Communications in Computer and Information Science, 2008, 9:146-157

[7] V. Gligor, S. Gavrila, and D. Ferraiolo. On the formal definition of separation-of-duty policies and their composition. In Proceedings of 1998 IEEE Symposium on Research in Security and Privacy, 1998, p172-183
[8] R. Simon, and M. Zurko. Separation of duty in role-based environments. In Proceedings of 10th IEEE Computer Security Foundations Workshop, 1997, p183-194
[9] Ravi Sandhu and Venkata Bhamidipati, The ASCAA Principles for Next-Generation Role-Based Access Control. Proc. 3rd International Conference on Availability, Reliability and Security, 2008, p532-537
[10] Jorg R. Muhlbacher, Christian Praher. DS RBAC-Dynamic Sessions in Role Based Access Control. Journal of Universal Computer Science, 2009, 15(3):538~554
[11] M.A. Habib, Christian Praher. Object based dynamic separation of duty in RBAC. Internet Technology and Secured Transactions (ICITST), 2009
[12] M.A. Habib. Mutual exclusion and role inheritance affecting least privilege in RBAC. 2010 International Conference for Internet Technology and Secured Transactions (ICITST), 2010, p1-6
[13] Liang Chen and Jason Crampton. Inter-domain role mapping and least privilege. Proceedings of the 12th ACM symposium on Access control models and technologies, 2007, p157-162
[14] Jaideep Vaidya, Vijayalakshmi Atlur, Qi Guo. The role mining problem: finding a minimal descriptive set of roles. Proceedings of the 12th ACM symposium on Access control models and technologies, 2007, p175-184.
[15] Charles E. Youman, Ravi S. Sandhu, Edward J. Coyne. Role engineering. Proceedings of the first ACM Workshop on Role-based access control, 1995
[16] M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness, chapter 3. W. H. Freeman, 1979.
[17] Daniel Ashlock. Evolutionary Computation for Modeling and Optimization. Springer Verlag (2006). ISBN 978-0-387-22196-4
[18] Thomos Blickle. Theory of Evolutionary Algorithms and Application to System Synthesis. Ph. D Thesis, Swiss Federal Institute of Technology, Zurich,1996

Lijun Dong, born in 1978. Dong received his Ph.D. degree in Computer Architecture from Huazhong University of Science & Technology in Wuhan, P.R. China in 2008. Dr. Dong is currently a lecturer for School of Computer, China University of Geosciences, Wuhan, P.R. China. His main research interests include network security, operating system security, access control and evolutionary algorithms. Email: donglijun.cug@gmail.com.