International Journal of Computational Intelligence Systems, Vol. 5, No. 5 (September, 2012), 860-867

Safety and Availability Checking for User Authorization Queries in RBAC

Jian-feng Lu*, Jian-min Han, Wei Chen
School of Mathematics-Physical & Information Engineering, Zhejiang Normal University
Jinhua, 321004, China

Jin-Wei Hu
Department of Computer Science, College of Engineering, QatarUniversity
Doha, 2713, Qatar

Received 27 November 2011

Accepted 15 June 2012

Abstract

This paper introduces the notion of safety and availability checking for user authorization query processing, and develop a recursive algorithm use the ideas from backtracking-based search techniques to search for the optimal solution. For the availability checking, we introduce the notion of max activatable set (MAS), and show formally how MAS can be determined in a hybrid role hierarchy. For the safety checking, we give a formal definition of dynamic separation-of-duty (DSoD) policies, and show how to reduce the safety checking for DSoD to a SAT instance.

Keywords: safety, availability, authorization, separation-of-duty.

1. Introduction

Handling the user authorization query[1] processing in an efficient way is a key issue related to the enforcement of access control policies in RBAC systems[2-3]. In RBAC, permissions are associated with roles, and users are granted membership in appropriate roles, thereby acquiring the roles' permissions. This simple user authorization model in RBAC is sufficient in well organized systems when a user is typically assigned to a small number of roles. However, in a hybrid hierarchy, maintaining permission acquisition and role activation semantics can become quite challenging, especially in complex and collaborative systems. When a user requests a particular set of permissions, the authorization system should be able to determine whether to grant that request or not. When determining which set of roles should be activated in a single session for a particular set of permissions requested by a user, *safety* and *availability* checking should be concerned.

- *Safety checking* determines whether the set of roles satisfy all constraints governing role activation, such as dynamic separation-of-duty (DSoD) policies. DSoD can be implemented in a natural and efficient way has been recognized as one of RBAC's great advantages[4]. Furthermore, the safety checking should follow the least privilege principle, which makes it more complex.
- *Availability checking* determines whether the requested permissions are covered by the set of roles and hence are available to the session. Furthermore, the presence of hybrid hierarchies makes it more complex.

Previous work for this problem can be found in reference 5, Zhang and Joshi introduced the User Authorization Query (UAQ) Problem. They proposed a two-step algorithm for the UAQ problem. In the first

*Corresponding author: lujianfeng@zjnu.edu.cn

step, the algorithm uses a greedy search to find a role set that cover the desired permissions while following the least privilege principle. They referred to this as the *role mapping* problem. The safety and availability checking are considered in the second step. The algorithm checks whether the set of roles selected in the first step is available and satisfies all the DSoD and cardinality constraints. They referred to this as the *activation checking* problem. Obviously, this algorithm has some false negatives, such as falsely rejecting some legal success. For example, the algorithm finds a role set in the step 1, but it may violate the safety and availability checking in the step 2. In fact, there exists a set of roles that satisfies the safety and availability checking, but cannot be found in the step 1 by the two-step algorithm.

In order to address the above false negative, Wickramaarachchi et al.[6] introduced a recursive algorithm that uses the constraints to guide the search. The algorithm's ideas come from DPLL algorithm for SAT[7], it considers the effect of DSoD constraints and can choose a set of roles satisfying the safety checking. They assumed that issues of hybrid hierarchies are handled in the generation of the UAQ instances, and there is no availability checking. Moreover, the DSoD constraints were considered in reference 6 is actually dynamic mutually exclusive role (DMER) constraints, the distinction between DSoD policies as objectives and DMER constraints as a mechanism is not clearly [8].

Based on the above discussion, in this paper, we present an efficient approach for solving the problem of safety and availability checking for user authorization query processing in RBAC systems. Firstly, we compute the set of roles that can be activated by a user. This work can prune the search space and hence enhance the search efficiency, and make sure the availability be satisfied. Secondly, we develop a recursive algorithm use the ideas from backtracking-based search techniques that use the DSoD policies to search for the optimal solution, and the safety checking during the search process. We recognize that make sure the safety checking during the search process is very important to address the false negatives that falsely rejecting some legal success. The contributions of this paper include the following:

- We introduce the notion of safety and availability checking for user authorization query (SAC-UAQ) processing in RBAC systems extended with hybrid role hierarchy.
- We introduce the notion of max activatable set (MAS) and show formally how MAS can be determined in a hybrid role hierarchy.
- We give a set-based specification of DSoD policies, the problem of safety checking for DSoD policies (SC-DSoD) in the context of RBAC systems, and show that it is intractable (coNP-complete) for directly enforcing DSoD policies in RBAC. We then show how to reduce the SC-DSoD to a SAT instance, thus to use SAT solvers to produce accurate results efficiently.
- We present a recursive algorithm use the ideas from backtracking-based search techniques that use the DSoD policies to find a role set that cover the desired permissions while following the least privilege principle, and satisfying the DSoD policies.

The rest of this paper is organized as follows. In Section 2, we formally define the SAC-UAQ problem. In Section 3 we introduce our approach to SAC-UAQ problem. The evaluation and illustration of our approaches are given in Section 4. Section 5 discusses related work, and Section 6 concludes and discusses the future work.

2. The Main Text Problem Definition

When a user requests a particular set of permissions P_{req} to carry out a particular task, the authorization system takes P_{req} as input, and tries to find an optimum set of roles R_{sat} to be activated in a single session, R_{sat} should satisfies the user's permission request and the safety and availability checking. This means that R_{sat} contains the minimal number of permission such that at least the permissions in P_{req} are activated, the extra permissions than P_{req} should be minimized. Inspired by the definition of the UAQ problem by references 5 and 6, we define the notion of safety and availability checking for user authorization query (SAC-UAQ) process in RBAC systems extended with hybrid role hierarchy. SAC-UAQ takes three groups of inputs: one is the permission request information P_{req}, another is the RBAC state information (*UA*, *PA*, *RH*), and the other is DSoD policies.

Definition 1. *SAC-UAQ (safety and availability checking for user authorization queries). Given the request permission P_{req}, an RBAC state ε, and the DSoD policy set D. The SAC-UAQ problem output a role set $R_{sat} \subseteq R$ such that the following conditions hold:*

- $P_{req} \subseteq perm(R_{sat})$.
- *Activation of R_{sat} satisfies all the DSoD policies in D.*
- *for any $R' \subseteq R$ that also satisfies the above two conditions, we have $perm(R_{sat}) \subseteq perm(R')$.*

where R denotes the set of all roles, perm(R) denotes the set of all permissions for which R is a member.

An RBAC state determines the set of roles of which a user is a member and the set of permissions for which a user is authorized.

Definition 2. *An RBAC state is a 3-tuple (UA, PA, RH)*

- *U, R, P denote the set of all users, the set of all roles, the set of all permissions respectively.*
- *UA* $\subseteq U \times R$*, a user-role assignment relation.*
- *PA* $\subseteq P \times R$*, a permission-role assignment relation.*
- *RH* $\subseteq R \times R$ *is a partial order on R called the inheritance relation, written as* $\geq_i$, $\geq_a$ *and* $\geq$.

As pointed out by Li et al. [8], it is very danger with equating DMER constraints with DSoD policies. A DSoD policy states that in order to have all permissions necessary to complete a sensitive task in a single session, the cooperation of at least a certain number of users is required. Inspired by reference 8, we give a formally definition of DSoD policies that consider the total number of available users as a limitation factor through referring to the Jason's work[9]. The definition of DSoD policy is based on following three requirements: (1) a DSoD policy must be a high-level requirement; (2) a DSoD policy must be expressed in terms of restrictions on permissions; (3) a DSoD policy must capture restrictions on user set involved in the task.

Definition 3. *A DSoD policy ensures that at least k users from a user set* $\{u_1,...,u_n\}$ *are required to perform a task that requires all these permissions in* $\{p_1,...,p_m\}$ *in a single session. Formally,*

$$dsod\left\langle \{p_1,\cdots,p_m\},\{u_1,\cdots,u_n\},k\right\rangle$$

where each p_i *is a permission that needed to complete a sensitive task, each* u_j *is a user that authorized to complete the task, m, n and k are integers, such that* $2 \leq k \leq min(m,n)$*, min returns the smaller value of the two.*

3. Approach for SA-UAQ

In this section we propose an approach for solving the SAC-UAQ problem.

3.1. *Max Activatable Set*

We introduce the notion of MAS associated with a hybrid role hierarchy that indicates the access capabilities of a user resulting from his membership to the roles in the hierarchy. MAS is similar to the notion of uniquely activatable set (UAS) in reference 1, which requires that each role set in UAS can be activated by a user in a single session, the permission set in each element of the UAS is uniquely, and the role set has the smallest cardinality for the same permission set. Thus, UAS is mainly relevant from the perspective of the principle of least privilege. MAS is less restrictive than UAS, the uniquely and least privilege semantics will be ensured in Section 3.3.

Definition 4. *MAS* (*max activatable set*)*. Let H=*(*R,* [*f*])*, where* [*f*] $\subseteq \{\geq_i, \geq_a, \geq\}$*, be a hybrid role hierarchy. Then MAS*(*H, u*) *is the max activatable set with largest cardinality, such that*

$$\forall r_i \in R,\ active(u,r_i,s_j) \rightarrow r_i \in MAS(H,u)$$

Where $active(u,r_i,s_j)$ *denotes that* r_i *can be activated in the single session* s_j *by user u .*

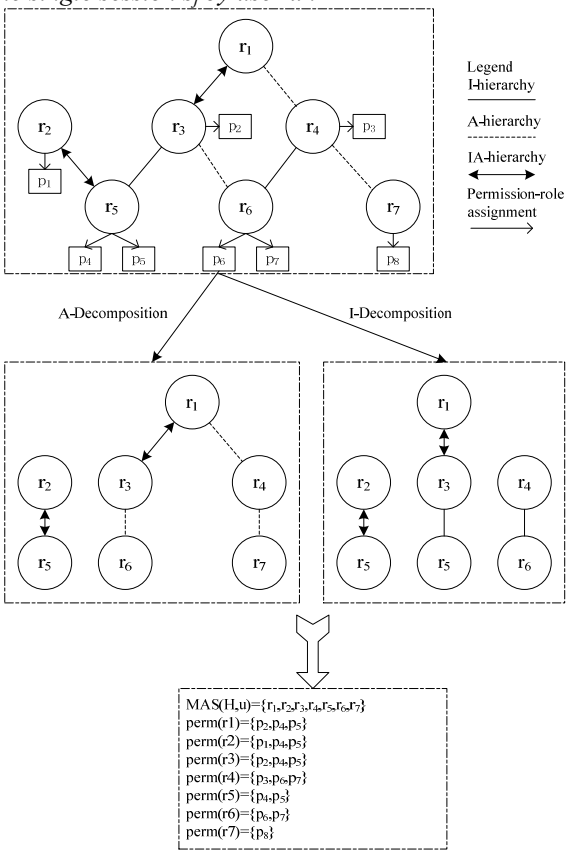

Fig.1. An example of monotype decomposition of a hybrid hierarchy.

Identifying MAS is essential, while making an authorization decision about whether or not a user should be allowed to activate a particular combination of roles in a single session. Since in a hierarchy that allows coexistence of the multiple hierarchy types, the permission-inheritance and role-activation semantics can be complex, thus making administration and management of large hierarchies difficult. In order to computing the MAS, one should consider the union of the A-hierarchy and the IA-hierarchy. When computing the permissions available to each role in MAS, one should consider the union of the I-hierarchy and IA-hierarchy. Therefore, when computing the MAS, one should decompose of a hybrid role hierarchy into its monotype components. We now reproduce the definitions of monotype decomposition of a hybrid hierarchy from reference 1.

Definition 5. (*Monotype decomposition of H*)*: Let H be a hybrid hierarchy, then we define I and A-decomposition of H as follows*

- *The monotype hierarchy HI is said to be the I-*

Decomposition of H if the following holds: $\forall r_1, r_2 (r_1 \geq_i r_2) \in H \vee (r_1 \geq r_2) \in H \rightarrow (r_1 \geq_i r_2) \in HI$.

- *The monotype hierarchy HA is said to be the A-Decomposition of H if the following holds:* $\forall r_1, r_2, (r_1 \geq_a r_2) \in H \vee (r_1 \geq r_2) \in H \rightarrow (r_1 \geq_a r_2) \in HA$.

Fig. 1 shows an example of monotype decomposition of a hybrid hierarchy. We assume that the user u is assigned to the roles r_1 and r_2, it can be straightforwardly to calculate the MAS(H,u) and the permissions available to each role in MAS(H,u) by the monotype decomposition.

3.2. *Safety Checking for DSoD policies*

Not all the roles in MAS(H,u) can be activated by the user u, since the DSoD policies that may prevent the user from activating some roles in MAS(H,u). E.g., in Fig.1, MAS(H,u)=$\{r_1,r_2,r_3,r_4,r_5,r_6,r_7\}$, assume that there is a DSoD policy d= $dsod<\{p_7, p_8\},\{u, u'\},k>$, where u' is another user. Then u is prevented to active $\{r_4, r_7\}$ or $\{r_6,r_7\}$ in a single session. We now introduce the problem of safety checking for DSoD policies.

Definition 6. *We say that an RBAC state ε is safe with respect to a DSoD policy* $d= dsod<\{p_1,\ldots,p_m\},\{u_1,\ldots,u_n\}, k>$, *which we denote by* $safe_d(\varepsilon)$, *if and only if in state ε no k-1 users from* $\{u_1,\ldots,u_n\}$ *together active all the permissions in* $\{p_1,\ldots,p_m\}$ *in a single session. Formally,*

$$\forall\{u_1',\cdots,u_{k-1}'\}\subseteq\{u_1,\cdots,u_n\}\left(\bigcup_{i=1}^{k-1} active(u_i')\not\supseteq\{p_1,\cdots,p_m\}\right)$$

Observe that if no k-1 users together active all the permissions in a single session, then no set of fewer than k users together active all the permissions. An RBAC state ε is safe with respect to a set D of DSoD policies, which we denote by $safe_D(\varepsilon)$, if and only if ε is safe with respect to every DSoD policies in D. If the security administrator wants to specify a DSoD policy, he should first identify a sensitive task, and then identify the permissions in $\{p_1,\ldots,p_m\}$ are needed to carry out a sensitive task, the constraint set of user set $\{p_1,\ldots,p_m\}$, and determine the minimum number k of collaborating users should be actived to complete it.

Definition 7. *SC-DSoD* (*safety checking for a DSoD policy*). *Given an RBAC state ε and a DSoD policy d, determine whether* $safe_d(\varepsilon)$ *is true.*

Theorem 1. *SC-DSoD is coNP-complete.*

Proof. Consider the complement of SC-DSoD, i.e., given a state ε and a DSoD policy d, determine if $safe_d(\varepsilon)$ is false, which is denoted by $\overline{\text{SC-DSoD}}$.

We first show that $\overline{\text{SC-DSoD}}$ is in NP. If an access control state ε is not *safe* with respect to a DSoD policy $d=dsod<\{p_1,\ldots,p_m\},\{u_1,\ldots,u_n\},k>$, there must exist k-1 users in $\{u_1,\ldots,u_n\}$ that together active all the m permissions in $\{p_1,\ldots,p_m\}$. If one correctly guesses the k-1 users that together have all the m permissions in the policy, verifying that the guess is correct can be done in polynomial time: compute the union of the k-1 users' permissions and check whether it is a superset of the set of permissions in the DSoD policy. But when verifying $safe_d(\varepsilon)$ is true, one must compute the set of permissions of every size-(t-1) user sets in $\{u_1,\ldots,u_n\}$, and check whether it is a superset of $\{p_1,\ldots,p_m\}$. The running time for this straightforward algorithm grow polynomially in the number of users and permissions and exponentially only in k. Therefore, $\overline{\text{SC-DSoD}}$ is in NP.

We now show that $\overline{\text{SC-DSoD}}$ is NP-hard by reducing the NP-complete set covering problem[10] to it. In the set covering problem, the inputs are a finite set S, a family $F=\{S_1,\ldots,S_l\}$ of subsets of S, and a budget B. The goal is to determine whether there exist B sets in F whose union is S. This problem is NP-complete. The reduction is as follows. Given S, F and B, construct a DSoD policy d as follows: for each element in S, we create a permission for it, let k be B+1 and let m be the size of S. We construct a DSoD policy $dsod<\text{S},\{u_1,\ldots,u_n\},B+1>$, and construct an access control state as follows. For each different subset $S_i(1\leq i\leq \text{l})$ in F, create a user $u_i\in\{u_1,\ldots,u_n\}$, to which all permissions in S_i are assigned. The resulting $safe_d(\varepsilon)$ is false if and only if B sets in F cover S. Therefore, $\overline{\text{SC-DSoD}}$ is NP-hard. □

The facts that SC-DSoD is intractable and SAT solvers can produce accurate results efficiently appear to provide a justification for using SAT solvers to enforce the SC-DSOD problem. We now show how the SC-DSoD problem can be reduced to a SAT instance. The SAT solver we use is SAT4J[11]. The translation works as follows.

Reduction to SAT: Given a DSoD policy $d=dsod<P,U,k>$ and an RBAC state ε=(UA, PA, RH), for each $u_i\in U$, we have a propositional variable v_i. This variable is true if u_i is a member of size-(k-1) user set $U'\subseteq U$ to active all the permissions in P. Then we have the following two kinds of constraints. For each $p_j\in P$, let $u_{i1},u_{i2},\ldots,u_{ix}$ be the users who are authorized for the permission p. We add the first constraint $v_{i1},v_{i2},\ldots,v_{ix}\geq 1$, which ensures that all the permissions in P are covered by U'. There are $|P|$ such constraints. Then we add the second constraint $v_1+v_2+\ldots+v_n\leq 1 (n=|U|)$, which ensures that $|U'|\leq$k-1. There is only one such constraint. If the return for the algorithm is "true", then we know that $safe_d(\varepsilon)$ is false, otherwise, $safe_d(\varepsilon)$ is true.

3.3. *A Recursive Algorithm for SAC-UAQ*

We present a recursive algorithm for SAC-UAQ as shown in Algorithm 1, the ideas of this algorithm comes from backtracking-based search techniques that use the DSoD policies to find a role set that cover the desired permissions while following the least privilege principle. In Algorithm 1, the request permission by the user is denoted as P_{req}, the extra permissions than P_{req} is denoted as P_{extra}, the permissions that have not been covered by the selected roles is denoted as P_{rem}, the set of roles which can satisfy the three conditions in Definition 1 is denoted as R_{sat}, the set of roles which can be active by the user is denoted as R_{active}, and denote the set of solutions and the current solution as R_{sel} and R_{sol} respectively, the set of all DsoD policies as D.

Algorithm 1. SAC-UAQ (P_{req}, R_{active}, D)

Input: P_{req}, R_{active}, D
Output: R_{sat}
1: R_{sat},R_{sol},R_{sel},$P_{extra} \leftarrow \varnothing$
2: $P_{rem} \leftarrow P_{req}$
3: **if** $P_{req} \not\subseteq perm(R_{active})$ **then**
4: **return** $\varnothing$
5: **end if**
6: **for each** $r \in R_{active}$ **do**
7: **if** ($perm(r) \cap \mathrm{P}_{req} = \varnothing$) **then**
8: $R_{active} = R_{active} \setminus r$
9: **end if**
10: **end for**
11: **selectRoles** ($P_{rem}, R_{sel}, P_{extra}, R_{active}, \mathrm{P}_{req}$);

12: **selectRoles** ($P_{rem}, R_{sel}, P_{extra}, R_{active}, \mathrm{P}_{req}$){
13: **if** $P_{rem} = \varnothing$ **then**
14: **if** $R_{sat} = R_{sel}$ **then**
15: $R_{sat} = R_{sel}$
16: **else if** $| perm(R_{sat}) | > | perm(R_{sel}) |$ **and** $| R_{sat} | > | R_{sel} |$ **then**
17: $R_{sat} = R_{sel}$
18: $R_{sol} = R_{sol} \cup R_{sel}$
19: **end if**
20: **return**
21: **end if**
22: **for each** $d \in D$ **do**
23: **if**(*SC-DSoD*(d, ε)) **then**
24: $R_{sel} = R_{sol} \setminus r$
25: **end if**
26: **end for**
27: **if** $| P_{extral} | > | perm(R_{sat}) |$ **and** $| perm(R_{sat}) | > 0 |$ **then**
28: **return**
29: **end if**
30: **if** $R_{active} = \varnothing$ **then**
31: **return**
32: **end if**
33: select next $r \in R_{active}$ that maximize $\dfrac{| perm(r) \cap \mathrm{P}_{rem} |}{| perm(r) \setminus \mathrm{P}_{req} \setminus P_{extral} |}$
34: **selectRoles (** $\mathrm{P}_{rem} \setminus perm(r)$, $R_{sel} \cup r$, $\mathrm{P}_{extral} \setminus (perm(r) \setminus \mathrm{P}_{req})$, $R_{active} \setminus r$, P_{req})
35: **selectRoles** ($P_{rem}, R_{sel}, P_{extra}, R_{active} \setminus r, \mathrm{P}_{req}$)}

Algorithm 1 performs the following steps:

(1) Make a simple decision that if the request permission P_{req} beyond the availability checking, that means such a matching role set doesn't exist.
(2) Prune the search space that remove from the candidate set R_{active} any role that does not include any permission in P_{req}.
(3) Do a backtracking based search where the search tree is traversed recursively, and performs the following four steps:
   a) Check whether the current set of selected roles R_{sol} is a solution, such as covers all the requested permissions P_{req}. If so, check whether it is a better solution than the stored current best solution R_{sat} . If so, record R_{sol} as the new best solution R_{sat}.
   b) Safety checking for each DSoD policies. The input includes a DSoD policy and the new RBAC state, we reduce it to a SAT problem, when the SAT solver returns false, then remove the new role from the current solution R_{sol}.
   c) Return if there is no role in R_{active}.
   d) Heuristically select the next role from the candidate roles. We prefer to select the role whose permission sets overlap the most with P_{rem}, and the least with extra permissions than P_{extra}.
   e) Recursively call itself with the next role selected.
   f) Recursively call itself with the next role not selected and removed from the candidate set R_{active}.

We now show that the output R_{sat} for Algorithm 1 is the best solution for the problem of SAC-UAQ.

Theorem 2. *Let R_{sat} denotes the output of the algorithm SAC-UAQ (R_{sat}, R_{active}, D), then the following conditions hold:*

(1) *$R_{sat} = \varnothing$ if and only if SAC-UAQ(P_{req},R_{active},D) is a non-solution problem.*
(2) *There does not exist another solution for SAC-UAQ (P_{req},R_{active},D), and R' is better than R_{sat} .*

Proof. (1)For the "only if part", suppose, for the sake of contradiction that there exists a solution

$R_{sol}=\{r_1,\ldots,r_s\}$ for SAC-UAQ(P_{req},R_{active},D), and $R_{sol}\neq\varnothing$. Then $\{r_1,\ldots,r_s\}$ cannot be removed from the candidate set R_{active} in step 1 and 2. Since all the roles in $\{r_1,\ldots,r_s\}$ be active by the user will not violate any DSoD policies, and $perm(R_{sol})=P_{req}$, then all the roles will be selected by recursively in step 3. If there does not exist another better resolution, $R_{sat}=\{r_1,\ldots,r_s\}$. This would contradict the assertion that $R_{sat}=\varnothing$. Therefore, if $R_{sat}=\varnothing$, then SAC-UAQ(P_{req},R_{active},D) is a non-solution problem. For the "if part", if SAC-UAQ(P_{req},R_{active},D) is a non-solution problem, there are two cases. The first case is that availability checking is false, such as $P_{\text{req}} \not\subseteq perm(R_{active})$, which will return a empty set by step i in Algorithm 1. The second case is that safety checking is false, that means any set that cover the request permissions be active by the user will violate the DSoD policy set. Ofcource return the empty set. Therefore, if SAC-UAQ(P_{req},R_{active},D) is a non-solution problem, $R_{sat}=\varnothing$.

(2)For the sake of contradiction, we assume that there exists another solution R' for SAC-UAQ(P_{req}, R_{active},D), and R' is better than R_{sat}. By the Definition 1, we have (i) $P_{req}\subseteq perm(R_{sat})$, $P_{req}\subseteq perm(R')$; (ii) Activation of R_{sat} or R' satisfies all the DSoD policies in D. (iii) $perm(R')\subseteq perm(R_{sat})$. Algorithm 1 do a backtracking based search for better solution. Assume that the stored current best solution is R_{sat}, and the current selected solution is R'. In step 3.1, checking whether R' is a better solution than the stored current best solution R_{sat}. If so, record R_{sol} as the new best solution R_{sat}. where the case (iii) $perm(R')\subseteq perm(R_{sat})$ is false. Therefore, There does not exist another solution R' for SAC-UAQ(P_{req}, R_{active},D), and R' is better than R_{sat}. □

4. Evaluation and Illustration

In reference 5, Zhang and Joshi introduced two approaches for the UAQ problem. The first approach is a two-step algorithm for the UAQ problem based on greedy approach. As discussed in Section 1, this approach has some false negatives, such as falsely rejecting some legal success. The second approach is a naive brute-force algorithm that goes through ever subset of the set of all roles available to the user. In this section, we implement our proposed approach for SAC-UAQ problem. We first compare our results with those from the greedy approach introduce in reference 5. We second compare our results with those from the brute-force approach introduce in reference 5. The implementation of the three algorithms was written in Java. Experiments were carried out on a notebook with an Intel Core i7 2630QM running at 2.0GHz, and with DDR3 2GB 1066MHz, running Microsoft Windows 7 Home Basic. The methodology that we use in generating test instances is as follows.

- The ratio of roles to users is 5:1
- The ratio of roles to DSoD policies is5:1
- The ratio of roles to permissions is1:5
- The ratio of roles to permission request is 2:1

4.1. Comparison with Greedy Approach

In order to compare our results with the greedy approach proposed in reference 5, we first give a running example to show the greedy approach may produce a solution satisfies the "role mapping" module but fails the "activation checking module", and ultimately the user request get rejected.

Example 1. Consider the role set R in an RBAC system, the role-permission relationships, the role set available for a user u, the role activation of another user u', and DSoD policies considered for this example.

- $R=\{r_1,r_2,r_3,r_4,r_5,r_6,r_7,r_8,r_9,r_{10}\}$
  - $perm(r_1)=\{p_1,p_3,p_6\}$
  - $perm(r_2)=\{p_1,p_5,p_9,p_{12},p_{14}\}$
  - $perm(r_3)=\{p_2,p_3,p_4,p_8,p_{11}\}$
  - $perm(r_4)=\{p_1,p_6,p_{13},p_{14},p_{16},p_{19},p_{20}\}$
  - $perm(r_5)=\{p_3,p_6,p_7,p_9,p_{10}\}$
  - $perm(r_6)=\{p_5,p_7,p_{10},p_{15},p_{17},p_{18},p_{20}\}$
  - $perm(r_7)=\{p_1,p_4,p_{15}\}$
  - $perm(r_8)=\{p_3,p_7,p_{16},p_{18},p_{19}\}$
  - $perm(r_9)=\{p_2,p_5\}$
  - $perm(r_{10})=\{p_7,p_9,p_{11},p_{20}\}$
- $MAS(H,u)=\{r_1,r_3,r_7,r_9,r_{10}\}$
- $Active\_Role(u')=\{r_1,r_4,r_5,r_8\}$
- DSoD policy: $d=ssod<\{p_8,p_{11}\},\{u,u'\},2>$

Table 1. Comparison greedy approach with ours

Permission Request	Greedy Approach	Our Approach
$\{p_1,p_3,p_5,p_7,p_9\}$	$\{r_2,r_5\}$ Reject:un-available	$\{r_1,r_9,r_{10}\}$ Grant
$\{p_1,p_3,p_4,p_5,p_9,p_{11}\}$	$\{r_1,r_3,r_{10}\}$ Reject: un-safe	$\{r_1,r_7,r_9,r_{10}\}$ Grant

The results are summarized in Table 1. When the user request to active the permissions in $\{p_1,p_3,p_5,p_7,p_9\}$, the greedy approach produces a solution $\{r_2,r_5\}$, while both r_2 and r_5 are unavailable for u, thus the permission request will be rejected. But our approach can produce the best solution $\{r_1,r_9,r_{10}\}$. When the permission request is $\{p_1,p_3,p_4,p_5,p_9,p_{11}\}$, the greedy approach produces a solution $\{r_1,r_3,r_{10}\}$, although all of the roles in $\{r_1,r_3,r_{10}\}$ are available for u, let u active r_3 violates the DSoD policy d, thus the permission request will also be rejected. In fact, $\{r_1,r_7,r_9,r_{10}\}$ is the best resolution for this problem.

Table 2. The results out of the greedy approach

Out of solution	Percentage
unavailable	43%
unsafe	27%
non-optimal	9%
optimal	21%

We second implement 100 test cases with varying test instances for greedy approach. As shown in Table 2, 79% of the test cases produced an incorrect solution, where 43% violated the availability checking, and 27% violated the safety checking. The remaining 30% produced a correct solution. However, 9% produced sub-optimal solutions which did not follow the least privilege principle. On the other hand, as proved in Theorem 2, our approach always produces the best solution for SAC-UAQ.

4.2. *Comparison with Brute-Force Approach*

In order to understand the effectiveness of our approach, we have implemented two algorithms: one is our approach, the other the brute-force approach. Fig. 2 shows the result of running the experiments for the two approaches. When the number of roles is small, the two approaches perform produce comparable results. As the number of roles increase, the overall trend in time taken increases exponentially making the brute-force approach impractical for implementation in dynamic systems. On the other hand, our approach takes a few seconds, even for a larger number of roles.

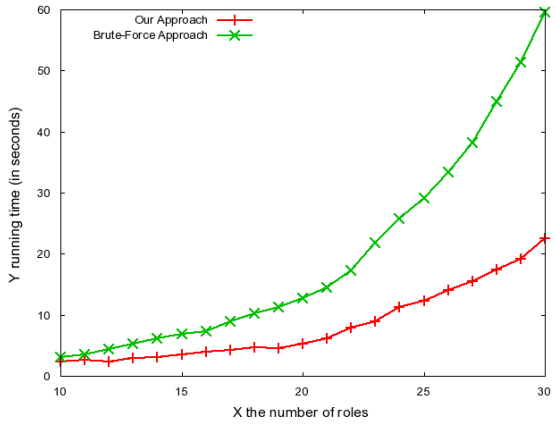

Fig. 2. CPU time for our approach and brute-force approach

5. Related Work

There are a number of user authorization frameworks in the literature which follow different approaches for user authorization queries processing in RBAC systems.

To our knowledge, the notion of Uniquely Activatable Set (UAS) first appeared in Joshi et al.[1], which is similar to our notion of MAS. UAS is essentially a set of role sets and each element in it is unique in the term of permissions and can be activated by the user in one session. They introduce UAS to simplify the user authorization query processing. They also propose a set of theorems to compute the UAS from the hybrid hierarchy. And proposed two algorithms to compute UAS for a user, one is called the decomposition based algorithm, and the other is called the derived relations based algorithm. However, the complexities of both algorithms are not polynomial[12]. While computing the complete set of UAS is complex, checking whether a set of roles is within the UAS can be done within polynomial time[13]. Zhang and Joshi[5] combined the UAS checking and role mapping together with the user authorization processing. They introduced the generalized notion of UAS where users can be assigned to any role or multiple roles that appear in the hierarch. The paper proposed a greedy approach and a naive brute force approach for the UAQ problem. As shown in Section 4, the greedy search algorithm does not consider the effect of any constraint and may choose a set of roles violating some constraint. And the brute-force approach is impractical for implementation in a large dynamic system. A more general definition of UAQ problem where the permission grant includes both a lower bound and an upper bound is introduced by Wickramaarachchi et al.[6], They introduced two approaches for the UAQ problem. The first approach extends the Davis-Putnam-Logemann-Loveland (DPLL) algorithm, the second approach reduce the UAQ problem to the MAXSAT problem.

However, the above work considered DMER constraints rather than DSoD policies that affect the solution of UAQ. Li et al.[8] considered that the distinction between DSoD policies as objectives and DMER constraints as a mechanism is not clearly will rise the security risks. SoD policy is a fundamental principle of information security, the concept of SoD can be traced back to 1975 when Saltzer and Schroeder[14] took it as one of the design principles for protecting information, under the name “separation-of-privilege”. Later on, SoD has been vastly studied by various researchers as a principle to avoid frauds. There exists a wealth of literature on SoD policies in the context of RBAC[5]. It has been recognized that “one of RBAC’s great advantages is that SoD rules can be implemented in a natural and efficient way”.

The first paper on SoD policies in RBAC is proposed by Ferraiolo and Kuhn[16], who used the terms static and dynamic SoD to refer to static and dynamic enforcement of SoD. Li et al.[8] used static mutually exclusive role (SMER) constraints to enforce SSoD policies in RBAC. They defined an RSSoD constraint (essentially as described in Problem 19 above), but provide no analysis of the complexity of computing the

set of all such constraints. Note that an RSSoD constraint is a set of roles that cover Q and contains no redundancy. In other words, the RSSoD generation problem is identical to the irreducible cover enumeration problem and is, therefore, NP-hard (Theorem 2). The above results are summarized in the following theorem[17]. Inspired by the work by Li et al.[18], they tackled the reduction of the verification of resiliency checking problem to a SAT instance. Their approach considers a set of static, mutually exclusive role constraints. We reduce the SC-DSoD problem to a SAT instance builds on their findings, which enables us to use existing SAT solvers in our implementation and benefit from several decades of research in designing SAT solvers.

6. Conclusions

In this paper, we defined the notion of safety and availability checking for user authorization query (SAC-UAQ) processing in RBAC systems extended with hybrid role hierarchy, and presented a recursive algorithm use the ideas from backtracking-based search techniques that use the DSoD policies to find a role set that cover the desired permissions while following the least privilege principle, and satisfying the DSoD policies. We compared our approach with the greedy approach and brute-force approach in reference 5, and found that our approach always provide an optimal solution in an efficient way.

Acknowledgements

This work is supported by National Natural Science Foundation of China under Grant 61170108, Zhejiang Provincial Natural Science Foundation of China under Grant LQ12F02005, MOE (Ministry of Education in China) Project of Humanity and Social Science under Grant 12YJCZH142, Opening Fund of Key Discipline of Computer Software and Theory of Zhejiang Province at ZJNU under Grant ZSDZZZZXK23.

References

1. J. B. D. Joshi, E. Bertino, A. Ghafoor and Y. Zhang, Formal Foundations for hybrid hierarchies in GTRBAC, in *ACM Trans. Inform. Syst.* 10(4) (2008), pp. 1-39.
2. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C.E. Youman, Role-based access control models, in *IEEE Computer*, 29(2)(1996), pp. 38-47.
3. ANSI. 2004. *American national standard for information technology-role based access control* (ANSI INCITS 359-2004, 2004).
4. D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli, *Role-Based Access Control* (Artech House, 2003).
5. Y. Zhang and J. B. D. Joshi. Uaq: a framework for user authorization query processing in rbac extended with hybrid hierarchy and constraints, in *Proc. 13th ACM symposium on Access control models and technologies,* (New York, NY, USA, 2008), pp. 83-92.
6. T. Guneshi, H. Q. Wahbeh, and Ninghui Li, An Efficient Framework for User Authorization Queries in RBAC Systems, in *Proc.* 14th ACM symposium on Access control models and technologies, (Stresa, Italy, 2009), pp. 23-32.
7. C. Sinz, Visualizing sat instances and runs of the dpll algorithm, in *J. Autom. Reason*, 39(2)(2007), pp. 219-243.
8. N. Li, M. Tripunitara, and Z. Bizri, On Mutually Exclusive Roles and Separation-of-Duty, in *ACM Trans. Inform. Syst.* 10(2) (2007), pp. 1-35.
9. J. Crampton, Specifying and enforcing constraints in role-based access control, in *Proc. 8th ACM Symposium on Access Control Models and Technologies*, (Villa Gallia, Como, Italy, June 2003), pp.43-50.
10. C. H. Papadimitriou, *Computational Complexity*, (Addison Wesley Longman, 1994).
11. D. L. Berre (project leader), *SAT4J: A satisfiability library for Java*. URL http://www.sat4j.org/, January 2006.
12. S. M. Chandran, and J. B. D. Joshi, Towards Administration of a Hybrid Role Hierarchy, in *Proc. 6th IEEE International Conference on Information Reuse and Integration*, (Las Vegas Hilton, Las Vegas, NV, USA, 2005), pp. 849-866.
13. S. Du, and J. B. D. Joshi, Supporting Authorization Query and Inter-domain Role Mapping in Presence of Hybrid Role Hierarchy, in *Proc. 11th ACM Symposium on Access Control Models and Technologies*, (Lake Tahoe, California, USA, June 2006), pp. 228-236.
14. J. H. Saltzer and M. D. Schroeder, The protection of information in computer systems, *Proceedings of the IEEE*, 63(9)(1975), pp. 1278-1308.
15. R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, Role-Based Access Control Models, in *Computer*, 29(2) (1996), pp. 38-47.
16. D. F. Ferraiolo, and D. R. Kuhn. *Role-based access control*, in *Proc. 15th National Information Systems Security Conference*, (Baltimore, MD, October 1992), pp. 554-563.
17. C. Liang and J. Crampton. Set Covering Problems in Role-Based Access Control, In *Proc. 14th European Symposium on Research in Computer Security*, (Saint-Malo, France, 2009), pp. 689-704.
18. N. Li, M. V. Tripunitara, and Q. Wang, Resiliency Policies in Access Control, in *ACM Trans. Inform. Syst.* 12(4)(2009), pp. 113-137.