Supporting User Authorization Queries in RBAC Systems by Role-Permission Reassignment

Jianfeng Lu(✉), Yun Xin, Hao Peng, Jianmin Han, and Feilong Lin

Department of Computer Science and Engineering,
School of Mathematics-Physical and Information Engineering,
Zhejiang Normal University, Jinhua, Zhejiang, China
lujianfeng@zjnu.cn

Abstract. The User Authorization Query (UAQ) Problem is a key issue related to efficiently handling users' access requests in RBAC systems. In practice, there may not exist any solution for the UAQ problem, as missing any requested permissions may make the failure of this task, while any extra permissions may bring the intolerable risk to the system. Hence, making a desirable update of the RBAC system state to support the UAQ problem is desirable. However, this task is generally complex and challenging as usually the resulting state is expected to meet various necessary objectives and constraints. In this paper, we study a fundamental problem of how generate a valid role-permission assignment to satisfy all objectives and constraints, such as reassignment objectives, prerequisite constraints and permission-capacity constraints. The computational complexity result shows that it is intractable (NP-complete) in general. We also propose an approach to reduce it to SAT that benefit from SAT solvers to reduce the running time. Experiment results show that the proposed approach scales well in large RBAC systems.

Keywords: RBAC · User authorization query · Computational complexity · Constraint · SAT solver

1 Introduction

Role based access control (RBAC) has received considerable attention over the past two decades, and established itself as the predominant model for advanced access control in many organizations and enterprises [1]. Several beneficial features, such as policy neutrality, support for least privilege and efficient self-management are associated with RBAC models. Such features make RBAC better suited for handling access control requirements of diverse organizations [2]. A fundamental problem in RBAC is to determine whether there exists an optimum set of roles to be activated to provide a particular set of permissions requested by a user, which is introduced as the user authorization query (UAQ) problem by Zhang et al. [3]. UAQ has been the subject of considerable research in recent years, and is widely accepted as a key issue related to efficiently handing users' access requests in RBAC [4–8]. Ideally, the chosen set of roles to be activated to exactly satisfy a user's permissions request. However, this is not always possible since we cannot find any combination of roles that can activate only requested permissions in many situations. Hence we have to find a set of roles to activate a set of permissions that is as close as possible to those requested permissions. Wickramaarachchi et al. [5] specified the UAQ problem by considering a lower bound P_{LB} and an upper bound P_{UB} for the set of requested permissions. There are two possible optimization objectives that should be included in the UAQ problem. One is prefer to minimize the number of extra permissions beyond the requested permissions, which is motivated by the principle of least privilege, as too many extra permissions may bring the intolerable risk to the system. The other is prefer to minimize the number of missing permissions, as the unavailability of too many of the requested permissions may make it difficult for a user to carry out the required task. Existing approaches to the UAQ problem primarily focus on how to design approximate or exhaustive solutions [5–7]. However, there may not exists any solution for UAQ as we cannot find any combination of roles that have permissions between P_{LB} and P_{UB}. Hence, a novel approach for supporting the UAQ problem by making a desirable update of the RBAC system state is desirable.

An RBAC system state is determined by three types of assignments: *user-role* assignment (UA), *role-role* assignment (RH), and *permission-role* assignment (PA). To make the most RBAC benefits, proper considerations should be taken in the entire life cycle of roles, which includes four stages: role analysis, role design, role management, and role maintenance. Particularly, the role maintenance stage concerns the changes related roles in access control system. Hu et al. [9] refer to the updating of UA, RH and PA in the role maintenance stage as *role updating*. In our observation, UA is business-driven, since user's role memberships are determined by their attributes, such as jobs, titles, and etc. Hence, in this paper, we focus on the updating of RH and PA, which we refer to as *role-permission reassignment* (RPR). It should be noted that role hierarchy play crucial roles in policy specification and security management in an organization, by allowing permission-inheritance, role hierarchies reduce overhead associated with the permission administration. When the role-permission assignments are renewed, administrators can accomplish the role-role assignments and permission-role assignments straightforwardly, so do the user-role assignments.

RPR is demanded not only for the UAQ problem when there doesn't exist any solution for it, but also needed in many access control scenarios, such as misconfiguration repair, proper satisfaction and role hierarchy transformation. However, RPR is generally complex and challenging, especially for large-scale RBAC systems. This is because the resulting state usually is expected to meet a variety of constraints, which makes it impossible to assign permissions to roles. For example, an objective of RPR may require that the chosen set $\mathcal{R}$ of roles can activate permissions between a lower bound P_{LB} and an upper bound P_{UB}. Obviously, such an objective states an overall request that must be satisfied, the set $\mathcal{R}$ of selected roles together activate the requested permissions, rather than restrict which roles are allowed to activate the individual permissions. Moreover, a prerequisite constraints require that the permissions can be activated by a role set $\mathbb{R}$ must also be activated by some other roles, can be used in cases, where a number of responsibilities are prerequisites for a certain task [10]. In addition, a permission-capacity constraint is satisfied in an RBAC system if and only if all the permission in $\mathbb{Z}$ can be activated by the set R_{LB} of roles, and any role not included in R_{UB} cannot activate any permission in $\mathbb{Z}$. Here, R_{LB} and R_{UB} are the lower bound and the upper bound role sets of $\mathbb{Z}$, such that $R_{LB} \subseteq R_{UB}$ [11].

To help system managers understand and manage RBAC policies, various RBAC policy analysis tools have been developed [10, 11], which focus on whether the given state-change rules can be satisfied, and do not care what the resulting states look like. In addition, they focus on user-role assignments rather than role-permission assignments. Meanwhile, there also exists a wealth of literature on role engineering [12]. However, RAR has two main differences from role engineering. First, role engineering focus on how to generate an appropriate set of roles, whereas RAR aims to determine whether an update can achieve with the request without violating any security constraints. Second, RAR works when RBAC states have been defined and possibly deployed, whereas role engineering usually define roles from scratch. The most similar work with RAR is RBAC updating. Ni et al. [13] studied the role adjustment problem (RAP) in the context of role-based provisioning based on machine learning algorithms. The main difference between our work to Ni's is that, RAR is request-driven, whereas RAP is a learning process. Specially, the administrator submits a specific reassignment objective, which tries to find the expected update. On the contrary, RAP is supplied by administrators with provisioning data and output a set of mappings from roles to entitlements. Hu et al. [9] proposed an approach for assistanting administrators with the updating of user-role, role-role, and permission-role relations. They also presented a tool, named RoleUpdater, which answers administrator's high-level update request for role-based access control systems. However, it is worth observing that there appears to be no good reason to assume that user's privilege escalation is forbidden, that is, the update will make the user's permission sets remain the same or deplete. In addition, they made sure that users' permission set varies from a lower bound to former, did not consider any security constraint.

In the above work, the system manager may change the system configuration in a trial-and-error way, which is effort-consuming, inefficient, and most importantly counter productive to security. Therefore, in this paper, we advocate for an automatic approach to RPR. The system manager needs only to specify the objectives and constraints: an RPR solution, if any, is automatically generated so that the system managers can follow to accomplish reassignment. In order to achieve this aim, we reduce the RPR to SAT that benefit from several decades of research in designing SAT solvers to reduce the running time. In general, if a truth assignment is found for the SAT instance, we can construct a valid *role-permission* (RP) assignment for the system configuration. We propose an approach for RGP by reducing RGP to the Boolean satisfiability (SAT) that to resolve it, which enables us benefit from several decades of research in designing SAT solvers to reduce the running time. Experiment results show the effective of our proposed approach.

2 Definition of the Reassignment Generation Problem

An RBAC state determines the set of permissions for which a role is assigned and the user can acquire the associate permissions via roles to take the associated tasks [1]. We assume that an RBAC state builds upon three countable infinite sets: U (the set of all possible users), R (the set of all possible roles) and P (the set of all possible permissions). The formal definition of an RBAC state is defined as follows.

Definition 1 (RBAC State). *An RBAC state γ is a tuple $\langle U, R, P, UR, RP \rangle$, where U, R, P denote the set of all users, the set of all roles, the set of all permissions, respectively. $UR \subseteq U \times R$ associates users with roles, $RP \subseteq R \times P$ associates roles with permissions.*

Given a UAQ problem, ideally, the chosen set of roles should activate the permissions not beyond the scope of $[P_{LB}, P_{UB}]$. However, this is not always possible when any combinations of roles fail to active any permission set P' such that $P_{LB} \subseteq P' \subseteq P_{UB}$. In this case, it is necessary to change the RP assignments that make sure there exists at least a solution for the UAQ problem. Given a requested permission region $[P_{LB}, P_{UB}]$, and a set $\mathcal{R} \subseteq R$ of roles, we write $RO\langle \mathcal{R}, P_{LB}, P_{UB} \rangle$ to express the RPR objective that we can find at least a combination of roles that have permissions between P_{LB} and P_{UB}. In the following, we give the definition of the reassignment objective, which determines whether the reassignment achieves the request.

Definition 2 (Reassignment Objective). *A reassignment objective is represented as $RO\langle \mathcal{R}, P_{LB}, P_{UB} \rangle$, where $\mathcal{R} \subseteq R$ is a role set, and P_{LB}, P_{UB} ($P_{LB} \subseteq P_{UB} \subseteq P$) are called the lower bound and upper bound for the set of requested permissions.*

A reassignment objective $RO\langle \mathcal{R}, P_{LB}, P_{UB} \rangle$ is satisfied if and only if $P_{LB} \subseteq Perm(\mathcal{R}) \subseteq P_{UB}$. In other words, every permission in P_{LB} must be assigned to at least one role in $\mathcal{R}$, and any permission in $P \backslash P_{UB}$ can not be assigned to any role in $\mathcal{R}$. In the remainder of this section, we introduce two types of security constraints, such as prerequisite and permission-capacity. These two constraints (or their special forms) have been considered in existing literature [9–11].

Definition 3 (Prerequisite constraint). A prerequisite constraint is represented as $PRE\langle cond, \mathbb{R} \rangle$, where $\mathbb{R} \subseteq$R is a role set, *cond* is called prerequisite condition on $\mathbb{R}$ that it is an expression consisting of roles, conjunctive operator $\wedge$, disjunctive operator $\vee$, and negation operator $\neg$.

A prerequisite constraint $PRE\langle cond, \mathbb{R} \rangle$ is satisfied if and only if for any member p of roles in $\mathbb{R}$, the role membership of p satisfies *cond*. Prerequisite constraints state that if a role takes a certain responsibility, she is also required to take some other responsibilities. In particular, role hierarchy can be represented and enforced using prerequisite constraints.

Definition 4 (Permission-Capacity Constraint). *A permission-capacity constraint is represented as $PC\langle \mathbb{Z}, R_{LB}, R_{UB} \rangle$, where $\mathbb{Z}$ is a permission set, and $R_{LB} \subseteq R_{UB} \subseteq R$ are called the lower bound and the upper bound role sets of all the permissions in $\mathbb{Z}$ is a member of, respectively.*

$PC\langle \mathbb{Z}, R_{LB}, R_{UB} \rangle$ is satisfied if and only if $\mathbb{Z} \subseteq Perm(R_{LB})$ and $\forall r \notin R_{UB}$ such that $Perm(r)$ $\cap \mathbb{Z} = \emptyset$. In practice, many permissions are related to security or privacy focus, these permissions should be assigned to a few key roles. In contrast, we require a set of permissions be assigned to at least a certain number of roles so as to meet workload or resiliency requirement. When $R_{LB} = \emptyset$, there is no limitation on the minimum roles that the permissions in $\mathbb{Z}$ must be a member of, and $R_{UB} = R$ means there is no limitation on the maximum roles that should be assigned to.

Definition 5 (Reassignment Configuration). *Given an RBAC state γ, an RPR configuration is denoted as a 3-tuple $\langle \gamma, C, O \rangle$, where γ is an RBAC state, C is a set of constraints where each constraint takes one of the form of prerequisite and permission-capacity constraint, and O is a set of reassignment objectives.*

When the reassignment configuration $\langle \gamma, C, O \rangle$ is given, a interesting problems arise, such as "How to generate a valid RP assignment under $\langle \gamma, C, O \rangle$?". We define it as and Reassignment Generation Problem (RGP) as follows.

Definition 6 (RGP). *Given an reassignment configuration $\langle \gamma, C, O \rangle$, the Reassignment Generation Problem (RGP) returns a valid role-permission assignment relation RP under $\langle \gamma, C, O \rangle$.*

3 The Complexity of the Reassignment Generation Problem

Theorem 1. *RGP is NP-complete*

Proof. On one hand, we show that it is efficient to check whether the returned RP relation is valid under the reassignment configuration $\langle \gamma, C, O \rangle$. We only need to check two things: (1) each reassignment objective in O is satisfied; (2) no constraint in C is violated. It is obvious that there exist many efficient algorithms to check them, and hence RGP is in NP.

On the other hand, we show that RGP is NP-hard by reducing the NP-complete monotone SAT problem to its subcase that determines whether such a valid role-permission assignment exists. In monotone SAT, given an expression ϕ in conjunctive normal form (CNF) and ask whether there exists a truth assignment for variables appeared in ϕ such that ϕ is evaluated to true. Let $\phi = \phi_1 \wedge \cdots \wedge \phi_m$, where $\phi_i = l_{i_1} \vee \cdots \vee l_{i_l}$ is a clause and l_{i_j} is a literal, each clause contains either only positive literal or only negative literal. Let $\{v_1, \cdots, v_n\}$ be the set of variables appeared in ϕ. Without loss of generality, assume that no clause contains both v and $\neg v$. Given a monotone SAT instance, we call a clause with only positive literals a positive clause, denoted as ϕ^+, and otherwise a negative clause, denoted as ϕ^-, and construct an reassignment configuration $\langle R, P, C, O \rangle$ as follows: For each clause $\phi^+ \in \phi$, create a permission p_{ϕ^+}; for each clause $\phi^- \in \phi$, create a permission p_{ϕ^-}. Denote $P^+ = \bigcup_{\phi^+ \in \phi} p_{\phi^+}$, $P^- = \bigcup_{\phi^- \in \phi} p_{\phi^-}$, and $P = P^+ \bigcup P^-$. For each variable $v \in V$, create a corresponding role r_v, let $R = \bigcup_{v \in V} r_v$. Let $(r_v, p_{\phi^+}) \in RP$ if and only if ϕ^+ contains the variable v, and $(r_v, p_{\phi^-}) \in RP$ if and only if ϕ^- contains the variable $\neg v$. For each ϕ^+, let $R_{\phi^+} = \{r_v | \phi^+$ contains the variable $v\}$, and construct an RO objectives $RO\left\langle R_{\phi^+}, P^+_{LB}, P^+_{UB} \right\rangle$, where $P^+_{LB} = P^+_{UB} = \bigcup_{r_v \in R_{\phi^+}} \text{Perm}(r_v) \cap P^+$. For each ϕ^-, let $R_{\phi^-} = \{r_v | \phi^-$ contains the variable $\neg v\}$, and construct an RC constraint $RO\langle R_{\phi^-}, P^-_{LB}, P^-_{UB} \rangle$, where $P^-_{LB} = P^-_{UB} = \bigcup_{r_v \in R_{\phi^-}} \text{Perm}(r_v) \backslash P^+$. We construct a PC constraint $PC\langle \mathbb{Z}, R_{LB}, R_{UB} \rangle$ as follows: let $R_{LB} = R_{\phi^+}$, $R_{UB} = R$, and $\mathbb{Z} = P^+$. Now, we prove that ϕ is satisfiable if and only if there exists a valid role-permission assignment RP under $\langle \gamma, C, O \rangle$.

For the "only if" part, suppose that τ is a truth assignment that makes ϕ true. Then RP consists of: removing all (r_v, p) from RP where $\tau(v) = 1$ and $p \in P^-$, or $\tau(v) = 0$ and $p \in P^+$. Since ϕ is true, all ϕ^+ is true. For each ϕ^+, there must exists a variable v such that $\tau(v) = 1$. Then $\{r_v | \tau(v) = 1\}$ is a role in R_{ϕ^+} whose permission set is exactly P^+. Thus $PC\langle \mathbb{Z}, R_{LB}, R_{UB} \rangle$ is fulfilled. For each $RO\langle R_{\phi^-}, P^-_{LB}, P^-_{UB} \rangle$, and for each $r \in R_{\phi^+}$ and any permission $p \in P^-$ that (r, p) has been removed from RP, and any permission $p \in P^+$, (r, p) is still unchanged. Hence, $RO\left\langle R_{\phi^+}, P^+_{LB}, P^+_{UB} \right\rangle$ is satisfied. Similarly, for each $RO\langle R_{\phi^-}, P^-_{LB}, P^-_{UB} \rangle$, for each $r \in R_{\phi^-}$ and any permission $p \in P^+$ that (r, p) has been removed from RP, that means $P^+ \cap Perm(R_{\phi^-}) = \emptyset$, hence, $RO\langle R_{\phi^-}, P^-_{LB}, P^-_{UB} \rangle$ is also satisfied. For the "if" part, suppose RP is valid under $\langle \gamma, C, O \rangle$. We construct a truth assignment τ over $\{v_1, \cdots, v_n\}$ that makes ϕ be evaluated to true as follows: $\tau(v) = 1$ if and only if $Perm(r_v) \subseteq \mathbb{Z}$, otherwise, $\tau(v) = 0$. We now show that τ is a truth assignment that makes ϕ true as follows. Suppose, for the sake of contradiction, that there exists $\phi^+ = v_1 \vee \cdots \vee v_k$ is false under τ, it means that $\tau(v_i) = 0 \quad (1 \leq i \leq k)$, by the above constructions, $Perm(r_{v_i}) \subsetneq \mathbb{Z} (1 \leq i \leq k)$. $RO\left\langle R_{\phi^+}, P^+_{LB}, P^+_{UB} \right\rangle$ requires that $Perm(R_{\phi^+}) = \bigcup_{1 \leq i \leq k} Perm(r_{v_i}) \subseteq P^+_{UB} \subseteq \mathbb{Z}$, Thus, we reach a contradiction. On the other hand, suppose there exists $\phi^- = \neg v_1 \vee \cdots \vee \neg v_l$ is false under τ, that means $\tau(v_i) = 1 \ (1 \leq i \leq l)$, hence $Perm(r_{v_i}) \subseteq \mathbb{Z} (1 \leq i \leq l)$ by the above constructions. However, $RO\langle R_{\phi^-}, P^-_{LB}, P^-_{UB} \rangle$ requires that $Perm(R_{\phi^-}) = \bigcup_{1 \leq i \leq l} Perm(r_{v_i}) \subseteq P^-_{UB} \backslash \mathbb{Z}$, that means $Perm(r_{v_i}) \subsetneq \mathbb{Z} (1 \leq i \leq l)$, which reachs a contradiction, and hence RGP is NP-hard. □

4 An Approach for the Reassignment Generation Problem

In this section, we describe an approach for RGP by reducing the RGP to SAT, that is, we can benefit from several decades of research on the design of SAT solvers to reduce the running time. For each $(r_i, p_j) \in RP$, we specify a variable v_{ij} in our SAT instance. Variable v_{ij} being set to true indicates that $(r_i, p_j) \in RP$. If no satisfying truth assignment is found, then ϕ is unsatisfiable. Otherwise, we get a truth assignment A, we can construct a valid RP for $\langle \gamma, C, O \rangle$ in the following way: $RP = \{(r_i, p_j) | (v_{ij} = true) \in A\}$. Given a fixed RP $(r_i, p_j) \in RP$, we just need to specify a clause v_{ij}, which forces any truth assignment that satisfies $(r_i, p_j) \in RP$.

Reassignment objectivests: given a reassignment objective $RO\langle \mathcal{R}, P_{LB}, P_{UB} \rangle$, we can reduce it to Conjunctive Normal Form (CNF) in efficiently. (1) For every permission p_j in P_{LB}, and every role r_i in $\mathcal{R}$, we specify a clause $\phi = \bigvee_{r_j \in \mathcal{R}} v_{ij}$; (2) For every permission p_j in $P \backslash P_{UB}$, and every role r_i in $\mathcal{R}$, we specify a clause $\phi = \bigwedge_{r_j \in P \backslash P_{UB}} \neg v_{ij}$.

PRE constraints: given a prerequisite constraint $PRE\langle cond, \mathbb{R}\rangle$, such a constraint essentially states that $\mathbb{R} \rightarrow cond$, which can be equivalently written as $\neg\mathbb{R} \vee cond$. For every permission p_j in P, we construct a clause $\phi = f(\neg\mathbb{R} \vee cond)$, where the function f constructs a clause from $\neg\mathbb{R} \vee cond$ by replacing every role with a variable: any role r_i in $\neg\mathbb{R} \vee cond$ is replaced with v_{ij}.

PC constraints: given a permission-capacity constraint $PC\langle \mathbb{Z}, R_{LB}, R_{UB}\rangle$, we can reduce it to CNF in efficiently: (1) For every role r_i in R_{LB}, and every permission p_j in $\mathcal{R}$, we specify a clause $\phi = \bigvee_{r_j \in R_{LB}} v_{ij}$; (2) For every role r_i in $R \backslash R_{UB}$, and every permission p_j in $\mathcal{R}$, we specify a clause $\phi = \bigwedge_{p_j \in P_{PC}} \neg v_{ij}$.

We have prototyped the proposed approach and performed some experiments using randomly generated instances. Our prototype is written in Java and use sat4j [14]. To generate the reassignment configuration $\langle \gamma, C, O\rangle$, we adapt data generation algorithm motivated by [7]. In order to compare our experimental results in different cases in convenient, we set the ratio of $|R| : |P|$ is 1:2 and 1:10 in each test case. All the experiments are carried out on a standard desktop PC with an Intel Core i7-4790 running at 3.6 GHz, and with DDR3 8 GB 1600 MHz, running the 64-bit Windows 7 operating system.

The following experimental results show the CPU time taken by each test case. Figure 1(a) shows our performance for different number of roles (i.e., no.R). We observe that our approach is highly resilient to an increase in no.R. On the one hand, the running time will increase as the number of roles increases. The main reason behind this phenomenon is that the computational complexity increase with the number of roles, and the larger range of optional collections, such as $\mathcal{R}$, R_{LB}, R_{UB}. On the other hand, the running time increases with the ratio of permissions and roles. The reason is that the larger of the ratio, the larger number of permissions, that the optional scope of the sets such as P_{LB}, P_{UB}, $\mathbb{Z}$ will get larger to increase the computational complexity. Figure 1(b) shows the performance for different number of RO constraints (*i.e.*, no. RO). The time taken is almost polynomial to no.RO. We tried for up to no.RO = 20 and our approach goes only up to fewer than 0.22 s when $|R| : |P|$ equals 1:10 and 0.13 s when $|R| : |P|$ equals 1:2. Figure 1(c) shows the performance of different number of PRE constraints (*i.e.*, no.PRE). The CPU time taken is almost polynomial to no.PRE. We tried for up to no.PRE = 20 and our approach goes only up to fewer than 0.25 s when $|R| : |P| = 1{:}10$, and 0.14 s when $|R| : |P|$ is 1:2. The higher ratio of permissions to roles, the more running time will be spent. This is due to fact that the more no. RO or no.PRE, the larger size and the higher complexity of the problem is, thereby, increasing the CPU time. Figure 1(d) shows the performance of different number of PC constraints (*i.e.*, no.PC). The CPU time increases as no.PC increases in both of the two cases. Our approach is also resilient with the increasing of no.PC. The major difference is that the increasing range is smaller than the former three cases obviously. Our experimental results show that our approach scales reasonably well with the larger RBAC system when no.R, no.P, no.RC,no.PC and no.PRE is large. In particular, our approach turns more effective when the ratio of permissions to roles is small.

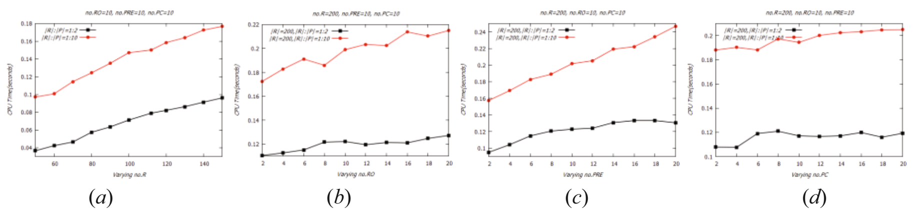

Fig. 1. Performance for different number of (a) roles (i.e., no.R); (b) RO constraints (i.e., no. RO); (c) PRE constraints (i.e., no.PRE); (d) PC objectives (i.e., no.PC).

5 Conclusion

We have introduced reassignment objective, prerequisite constraint, and permission-capacity constraint to role-permission reassignment, formally defined, studied the computational complexity, and proposed an approach for RGP. Our work will assistant administrators to answer whether the rule updates can achieve the request.

Acknowledgment. This work is supported by National Natural Science Foundation of China under Grant 61402418, 61503342, 61672468, 61602418, Social development project of Zhejiang provincial public technology research under Grant 2017C33054, 2016C3316.

References

1. ANSI.: American national standard for information technology-role based access control, ANSI INCITS 359-2004 (2004)
2. Xu, D., Kent, M., Thomas, L., et al.: Automated model-based testing of role-based access control using predicate/transition nets. IEEE Trans. Comput. **64**(9), 2490–2505 (2015)
3. Zhang, Y., Joshi, J.B.D.: UAQ: a framework for user authorization query processing in RBAC extended with hybrid hierarchy and constraints. In: 13th ACM Symposium on Access Control Models and Technologies, New York, USA, pp. 83–92 (2008)
4. Lu, J., Joshi, J.B.D., Jin, L., Liu, Y.: Towards complexity analysis of user authorization query problem in RBAC. Comput. Secur. **48C**, 116–130 (2015)
5. Wickramaarachchi, G.T., Wahbeh, H.Q., Li, N.: An efficient framework for user authorization queries in RBAC systems. In: 14th ACM Symposium on Access Control Models and Technologies, Stresa, Italy, pp. 23–32 (2009)
6. Armando, A., Ranise, S., Turkmen, F., Crispo, B.: Efficient run-time solving of RBAC user authorization queries: pushing the envelope. In: 17th ACM Conference on Data and Application Security and Privacy, San Antonio, Texas, USA, pp. 241–248 (2012)
7. Mousavi, N., Tripunitara, Mahesh V.: Mitigating the intractability of the user authorization query problem in role-based access control (RBAC). In: Xu, L., Bertino, E., Mu, Y. (eds.) NSS 2012. LNCS, vol. 7645, pp. 516–529. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34601-9_39
8. Chen, L., Crampton, J.: Set covering problems in role-based access control. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 689–704. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04444-1_42

9. Hu, J., Khan, K. M., Zhang, Y., Bai, Y., Li, R.: Role updating in information systems using model checking. Knowl. Inf. Syst. (2016). doi:10.1007/s10115-016-0974-4
10. Sun, Y., Wang, Q., Li, N., et al.: On the complexity of authorization in RBAC under qualification and security constraints. IEEE Trans. Dependable Secure Comput. **8**(6), 883–897 (2011)
11. Lu, J., Xu, D., Jin, L., Han, J., Peng, H.: On the complexity of role updating feasibility problem in RBAC. Inf. Process. Lett. **114**(11), 597–602 (2014)
12. Verde, N.V., Vaidya, J., Atluri, V., Colantonio, A.: Role engineering: from theory to practice. In: 2nd ACM Conference on Data and Application Security and Privacy, San Antonio, Texas, USA, pp. 181–192 (2012)
13. Ni, Q., Lobo, J., Calo, S.B., Rohatgi, P., Bertino, E.: Automating role-based provisioning by learning from examples. In: 14th ACM Symposium on Access Control Models and Technologies, Stresa, Italy, pp. 75–84 (2009)
14. SAT4 J: A satisfiability library for Java, January 2006, http://www.sat4j.org/