Dynamic Enforcement of Separation-of-Duty policies

Jianfeng Lu
College of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan 430074, P. R. China
e-mail: lujianfeng@smail.hust.edu.cn

Ruixuan Li
College of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan 430074, P. R. China
e-mail: rxli@hust.edu.cn

Zhengding Lu
College of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan 430074, P. R. China
e-mail: zdlu@hust.edu.cn

Yanan Jin
College of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan 430074, P. R. China
e-mail: jyamail@163.com

***Abstract*—Separation-of-duty (SoD) policy is widely considered to be a fundamental security principle for prevention of fraud and errors in computer security. A static SoD (SSoD) policy states that in order to have all permissions necessary to complete a sensitive task, the cooperation of at least a certain number of users is required. In this paper, we study the problem of dynamic enforcement of SSoD policies in access control systems. We formally define the notion of an SSoD policy, and introduce the problem of dynamic safety checking problem (DSCP) which asks whether an access control state satisfies a given SSoD policy, and show that it is intractable (NP-complete) for directly enforcing SSoD policies in access control. Furthermore, we design and evaluate an improvement algorithm for solving DSCP.**

Keywords-Separation-of-Duty; dynamic enforcement; access control; computational complexity

I. INTRODUCTION

Separation-of-duty (SoD) policy is widely considered to be a fundamental security principle for prevention of fraud and errors in computer security, and widely applied in business, industry, and government [1, 2]. A SoD policy requires that there should not exist a set of users of size less than some threshold is able to perform a task that requires all the permissions needed to complete the task. We call such a requirement a static SoD (SSoD) policy, following the terminology in [3]. The SoD policy can be enforced either statically or dynamically. In static enforcement, one ensures that each access control state that can be reached is safe with respect to the SoD policy for the task. An access control state is safe if each user-set such that users in the set together have all the permissions satisfies the security requirement that each set of size less than the threshold. However, static enforcement is usually in an intractable computational complexity class or even impossible to generate efficiently-verifiable constraints to precisely capture a high-level policy. For example, Li et al. [3] showed that there exist SSoD policies such that no set of static mutually exclusive role (SMER) constraints can precisely capture them. In dynamic enforcement, one identifies all permissions in performing the task, and maintains, for each instance of the task, the history of which user has performed which steps. When a user requests to perform the next step, the request is authorized only when the overall security requirement can be met by allowing this user to perform the next step. As in situations where dynamic and real time enforcement is desired, it is interesting and necessary to study its performance for instances that are likely to occur practice, since

In this paper we study dynamic enforcement SSoD policies. We formally define the notion of SSoD policies, and introduce the problem of dynamic safety checking problem (DSCP), which asks whether an access control state satisfies a given SSoD policy, and show that it is intractable (NP-complete). We present an improvement algorithm for DSCP, the algorithm uses preprocessing and static pruning techniques that reduce the number of users and usersets need to be considered. The study of the instance shows how the improvement algorithm works, and the experimental results shows that our algorithm can efficiently solve the DSCP.

The rest of this paper is organized as follows. Section 2 describes related works. Section 3 gives the definition of SSoD, DSCP, and studies the computational complexities of DSCP. Section 4 presents an improvement algorithm for DSCP and an implementation of the algorithm. Finally, Section 5 concludes this paper.

978-0-7695-3843-3/09 $26.00 © 2009 IEEE
DOI 10.1109/MINES.2009.102

II. RELATED WORK

The concept of SoD can be traced back to 1975 when by Saltzer and Schroeder [4] took it as one of the design principles for protecting information, under the name "separation-of-privilege". Since Clark and Wilson [1] applied SoD principle to the data objects to ensure integrity and to control frauds along with well-formed transactions as two major mechanisms for controlling fraud and error. SoD has been studied by various researchers as a principle to avoid frauds. For example, SoD policies have been recognized that "one of RBAC's great advantages is that SoD rules can be implemented in a natural and efficient way" [5].

The specification of SSoD in this paper is directly motivated by the word by Li et al. [3], it can be regarded as a generalization of the one in Li et al.: letting the constraint set of user set is all possible users. We also adsorb the work by Jason [6], and make no attempt to define the conditions that must be met for the constraint to be satisfied. There are many approaches to enforce a SoD policy. One approach is called as dynamic SoD (DSoD) in [7]. DSoD constraint prevents a user from simultaneously activating mutually exclusive roles in a session. It can be enforced by maintaining a history which included information on who performed each step. Sandhu adopted this idea and presented a history based mechanism for dynamically enforcing SoD policies [8]. Another approach to enforce SoD policies is to use SSoD policies. Ferraiolo and Kuhn [9] used the terms static and dynamic SoD to refer to static and dynamic enforcement of SoD, to our knowledge. Sandhu et al. [5] stated, in their widely cited paper that introduced the highly influential RBAC96 family of RBAC models, "The most common RBAC constraint is mutually exclusive roles. The same user can be assigned to at most one rule in a mutually exclusive set. This supports separation of duties, which is further ensured by a mutual exclusion constraint on permission assignment." Li et al. [3] clearly illustrated the difference between SSoD policies and SMER constraints. And formulated and studied fundamental computational problems related to the use of SMER constraints to enforce SSoD policies. However, as we show that the work by Li et al. is computationally expensive, and it is even impossible to generate efficiently-verifiable constraints to precisely capture an SSoD policy.

III. DYNAMIC SAFETY CHECKING PROBLEM FOR STATIC SEPARATION OF DUTY POLICIES

We now give a formal basis for representing SSoD policies in the context of access control. The concrete formulation of SSoD policies adopts the notion of such policies from Li et al. [3], but ours captures restrictions on user set involved in the sensitive task which related to the SSoD policy. In practice, the number of users in any organization is bounded. Thus it needs to consider the SSoD policies with an upper bound on the number of users in an access control state.

Definition 1. An SSoD policy is expressed as

$$ssod\langle P, U, k\rangle$$

where P=$\{p_1, \ldots, p_m\}$, U= $\{u_1, \cdots, u_n\}$, each p_i in P is a permission, u_j in U is a user, m, n, and k are integers, such that $1 \le k \le min(m, n)$, min returns the smaller value of the two. The policy $ssod\langle P, U, k\rangle$ means that at least k users from U are required to perform a task that requires all these permissions in P.

We now introduce the notion of an access control state, and the dynamic safety checking problem.

Definition 2. An access control state ε is given by a binary relation $UP \subseteq U \times P$, where U denotes the set of users, P denotes the set of all permissions.

We are not assuming permissions are directly assigned to users; rather, we assume only that one can calculate the relation UP from the access control state.

Definition 3. Given an initial access control state $\varepsilon = UP$, when a new access request (u, p) is generated, determining whether $\varepsilon' = UP \cup \{(u, p)\}$ is *safe* with respect to an SSoD policy $e = ssod\langle P', U', k\rangle$, if in state ε' no k-1 users from U' together have all the permissions in P'. We call it the dynamic safety checking problem (DSCP). Formally:

$$\forall\{u_1 \cdots u_{k-1}\} \subseteq U'\left(\bigcup_{i=1}^{k-1} auth_{P_{\varepsilon'}}(u_i) \not\supseteq P'\right)$$

Observe that if no k-1 users together have all the permissions in a policy, then no set of fewer than k users together have all the permissions.

Theorem 1. DSCP is coNP-complete.

Proof. The proof is similar to the one in [3] for the theorem that checking whether an RBAC state is safe or not with respect to a set of SSoD policies is coNP-complete. Although we study the SSoD policies in access control system is not restricted to RBAC systems, we also use a role to denote a set of users that have some common permissions, and let the constraint set of user set is all possible users. We reduce the set-covering problem [10] to the complement of DSCP. In the set covering problem, the inputs are a finite set S, a family $F = \{S_1, \cdots, S_l\}$ of subsets of S, and a budget B. The goal is to determine whether there exists B sets in F whose union is S. This problem is NP-complete. The reduction is as follows. Given S, F, and B, construct an SSoD policy e 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 have constructed an SSoD policy $ssod\langle S, U, B + 1\rangle$, construct an access control state as follows. For each different subset $S_i (1 \le i \le l)$ in F, create a user $u_i \in U$ to which all permissions in S_i are assigned. The resulting SSoD configuration is not enforceable if and only if B sets in F cover S. □

IV. AN ALGORITHM FOR DSCP

Even though DSCP is computationally intractable (coNP-complete) in theory, it is interesting and necessary to study its performance for instances that are likely to occur practice. In this section, we present an algorithm for the DSCP.

A. Description of the algorithm

In the dynamic enforcement of SSoD policies, a straightforward algorithm is as follows:

Straightforward Algorithm Given the initial state *UP*, when a new access request (u, p) is generated, the system checks whether $UP \cup \{(u, p)\}$ satisfies the $e = ssod\langle P', U', k\rangle$ policy. If the answer is "yes" for each of that sets, we know $UP \cup \{(u, p)\}$ satisfies *e*, and the access request (u, p) will be permitted. Otherwise, the access request (u, p) will be denied.

Our algorithm is based on this idea but we add the following improvements that greatly reduce the running time.

Preprocessing Given a state *UP* which includes the new access request (u, p), and a policy $e= ssod\langle P', U', k\rangle$. We only consider those users from U' who are authorized for at least one permission in P'. In fact that many information in *UP* is irrelevant to the result of DSCP with respect to *e*. Therefore, we remove all pairs (u, p) from UP if $p \notin P'$ or $u \notin U'$, we also remove all users u from U if u does not have any permission in *P*. In other words, There exists a special case where at least one permission in *P* doesn't appear in *UP*, then we know the state is safe with respect to *e*.

Static pruning The number of size-(*k*-1) user sets among *n* users is $C_n^{k-1} = \frac{n!}{(k-1)!(n-(k-1))!}$, it is close to n^{k-1} when k-1 is small compared with *n*. But we observe that not all these sets need to be considered. In particular, we are only interested in those user sets that have the closure property. In the following, we describe a static pruning technique that aims at reducing the number of users that need to be taken into account.

Definition 4. Among all users in *UP*, we say a user u_1 dominates another user u_2 if u_1's set of permissions is a superset of u_2's. We say a set of users U_1 dominates another set U_2 if there is a bijection between users in U_2 and U_1 such that for every user u in U_2, the corresponding user in U_1 dominates the user *u*.

Theorem 2. Let U_1 and U_2 are two sets of users, assuming that U_1 dominates U_2, $UP_1 \subseteq U_1 \times P$, $UP_2 \subseteq U_2 \times P$ represent the relation user-permission of U_1 and U_2 respectively, if UP_1 is safe with respect to an SSoD policy $e= ssod\langle P', U', k\rangle$, then UP_2 is also *safe* with respect to *e*.

Proof. Observe that if no *k*−1 users together have all the permissions in *P*, then no set fewer than *k* users together have all the permissions. Assuming that UP_2 is not safe with respect to $e=ssod\langle P', U', k\rangle$, that means that in state UP_2, there exist *k*−1 users $\{u_1,\cdots,u_{k-1}\}$ together have all the permissions in P', obviously, $\{u_1,\cdots,u_{k-1}\} \subseteq U_2$.

By definition, if U_1 dominates U_2, then there exists a bijection *f* between U_1 and U_2, such that $f(u) = v$ implies user $v \in U_1$ dominates users $u \in U_2$. Without loss of generality, for every user $u' \in \{u_1,\cdots,u_{k-1}\}$, if $u' \in U_1$, then $f(u')= u'$. Observe that if *f* does not satisfy this property for some $u' \in \{u_1,\cdots,u_{k-1}\} \wedge u' \in U_1$, then there exist $u_1 \in U_1$ and $u_2 \in U_2$ such that $f(u')= u_1$, and $f(u_2)= u'$. As the domination relation is transitive, we can then assign $f(u')= u'$ and $f(u_2)=u_1$. By repeating this process, we can arrive at a bijection *f* as above.

Therefore, for every users in $\{u_1,\cdots,u_{k-1}\}$, we can find a user set $\{u'_1,\cdots,u'_{k-1}\} \subseteq U_1$, for each user *u* in $\{u_1,\cdots,u_{k-1}\}$, the corresponding user u' in $\{u'_1,\cdots,u'_{k-1}\}$ dominates the user *u*. As the *k*-1users in $\{u_1,\cdots,u_{k-1}\}$ together have all the permissions in *P*, the *k*-1users in $\{u'_1,\cdots,u'_{k-1}\}$ can also together have all the permissions in *P*, thus UP_1 is not *safe* with respect to an SSoD policy *e*. Obviously, it violates the definition, that means the assumption is wrong. Therefore, if UP_1 is *safe* with respect to an SSoD policy $e = ssod\langle P', U', k\rangle$, then UP_2 is also *safe* with respect to *e*. □

In other words, Theorem 2 means that if a size-(*k*-1) userset *U* does not covers all the permissions in an SSoD policy, then other usersets which are dominated by *U* also does not cover all the permissions, and if *U* covers all the permissions, then other usersets which dominated *U* also cover all the permissions. In this way, we only need to systematically generate only size-(*k*-1) user sets which are not dominated by any other user sets, which we call the user set have the closure property.

Definition 5. Given a set of users $U = \{u_1,\cdots,u_n\}$, we say a set $U^* \subseteq U$ has the closure property if an only if for any $u_i \in U^*$, and any $u_j \in U$ such that u_j dominates u_i, we have $u_j \in U^*$.

The algorithm that directly generates only the user sets need to be considered which have the closure property as follows. First of all, we sort all users based on the number of permissions they have, in decreasing order, and assign each user an index, that is, users are listed as $u_1,\cdots,u_n$. If $1 \leq i < j \leq n$, then u_i has at least as many permissions as u_j. Secondly, we use an index *t* that initially has value *k*-1. We generate the first size-(*k*-1) set $\{u_1,\cdots,u_t\}$, and then increase the index *t* by one each time and generate all user sets that include u_t and are not dominated by any other set generated before. Theorem 3 shows that we need to generate only

the user sets that satisfy the closure property.

Theorem 3. Given an SSoD policy e= $ssod\langle P', U', k\rangle$, and a system state ε=UP, let $A \subseteq U'$ be a size-(k-1) user set that satisfies the closure property. We say ε is safe with respect to e if and only if there does not exist any A covers all the permissions in P.

Proof. For the "only if" direction, we show that if ε is safe with respect to e, then there does not exist any A covers all the permissions in P. It is obvious that if ε is safe with respect to e, it means there does not exist a set of fewer than k users that together have all the permissions in P, A cannot cover all the permissions in P as A is a size-(k-1) user set of U.

For the "if" direction, assuming that there exists a size-(k-1) user set B does not have the closure property, and B covers all the permissions in P. Since B does not have the closure property, there is a user $u_i \in B$ such that there exist $u_j \notin B$ such that u_j dominates u_i. We change B to A by substituting u_j with u_i, that is, A=$B\backslash\{u_i\}\cup\{u_j\}$. Clearly, A dominates B. If A still does not satisfy the closure property, we can repeat the substitution process until the resulting set has closure property. In this way, A satisfies the closure property, and covers all the permissions in P, that violates the condition in the theorem. Therefore, ε is safe with respect to e. □

B. Implementation and evaluation

In order to show the effect of the improvement algorithm, we prototyped the algorithm described in Section 4.1 and have performed some experiments. Our prototype is written in Java, the experiments were carried out on a notebook with an Intel(R) Core(TM)2 Duo CPU T5750 running at 2.0GHz, and with 2GB of RAM running Microsoft Windows XP Professional.

Some of our experimental results are presented in Table 1. As we can see in Table 1, our improvement algorithm (IA) solves DSCP efficiently than the straightforward algorithm (SA). It is capable to solve DSCP instances with nontrivial size in a relatively short time.

TABLE I. A TABLE THAT SHOWS THE SIZE OF RUNTIME TESTING OF WHETHER A STATE IS SAFE WITH RESPECT TO AN SSOD POLICY.

SSoD policy	P	U	Users	UP	Safe?	Size-(k-1) usersets		Runtime	
						SA	*IA*	*SA*	*IA*
ssod⟨P, U, 3⟩	5	5	10	15	Yes	32	3	0.042 s	0.004 s
ssod⟨P, U, 4⟩	10	10	20	30	Yes	963	23	2.0 s	0.049 s
ssod⟨P, U, 5⟩	10	15	20	40	No	3215	221	9.3 s	0.71 s

V. CONCLUSION

In this paper, we have formally defined SSoD policies and dynamic safety checking problem (DSCP) which asks whether an access control state satisfies a given SSoD policy, we show that it is intractable (NP-complete) for directly enforcing SSoD policies in access control. Computationally expensive notwithstanding, we design and evaluate an improvement algorithm for solving DSCP. Our algorithm uses preprocessing and static pruning techniques that reduce the number of users and usersets needed to be considered. The experimental results show that the improvement algorithm can efficiently solve the DSCP.

ACKNOWLEDGMENT

This work is supported by National Natural Science Foundation of China under Grant 60873225, 60773191 and 60403027, National High Technology Research and Development Program of China under Grant 2007AA01Z403. The authors would like to thank the anonymous reviewers for their valuable comments.

REFERENCES

[1] Clark, D.D., Wilson, D.R.: A comparison of commercial and military computer security policies. In: IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Los Alamitos, 1987, pp. 184-195.

[2] Clark, D.D., Wilson, D.R.: Evolution of a Model for Computer Integrity. In Report of the Invitational Workshop on Data Integrity, Z.G. Ruthberg and W.T.Polk(eds.), NIST Special Publication 500-168, Appendix A , 1989, pp. 1-3.

[3] Ninghui Li, Maheshv. Tripunitara, and Ziad Bizri. On Mutually Exclusive Roles and Separation-of-Duty. ACM Transactions on Information and System Security, Vol. 10, No. 2, May 2007, pp. 1-36.

[4] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. Proceedings of the IEEE, 63(9), September 1975, pp. 1278-1308.

[5] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. IEEE Computer, 29(2), February 1996, pp. 38-47.

[6] Jason Crampton. Specifying and enforcing constraints in role-based access control. In Proceedings of 8th ACM Symposium on Access Control Models and Technologies, 2003, pp. 43-50.

[7] [7]Foley, S. N. The specification and implementation of 'commercial' security requirements including dynamic segregation of duties. In Proceedings of the 4th ACM Conference on Computer and Communications Security, 1997, pp.125–134

[8] Sandhu, R. S. Transaction control expressions for separation of duties. In Proceedings of the Fourth Annual Computer Security Applications Conference, 1988, pp. 282-286.

[9] Ferraiolo, D. F. and Kuhn, D. R. Role-based access control. In Proceedings of the 15th National Information Systems Security Conference, 1992, pp. 554-563.

[10] C. H. Papadimitriou. Computational Complexity. Addison Wesley Longman, 1994.